AppDynamics APM Platform Documentation 21.6

AppDynamics Essentials

Welcome to the AppDynamics Application Performance Management (APM) Platform. Our documentation set introduces you to the AppDynamics APM Platform. AppDynamics helps you understand and optimize your business performance, ranging from software to infrastructure to business journeys.

We detail common concepts and procedures across all product modules, including APM, Application Analytics, End User Monitoring, and more. We also cover the Controller UI, which is the browser-based console that you use to understand, configure, and troubleshoot your application environment.

To report issues with the documentation, email the AppDynamics Docs Team at docs@appdynamics.com.

New to AppDynamics?

- Deployment Planning Guide
- Getting Started
- AppDynamics Concepts
- Deployment Options

Collaborating

- Tenant User Management
- Custom Dashboards
- Virtual War Rooms
- Health Rules

Product Homes

- Application Monitoring
- End User Monitoring
- Database Visibility
- Server Visibility
- Analytics

Other Resources

- Community Forums and Knowledge Base
- AppDynamics Help Center
- AppDynamics Exchange
- Support (login account required)

AppDynamics has implemented restrictions and will not enter into any negotiation with, or seek to solicit any interest from, any third party in relation to the sale or use of the AppDynamics products or software or any part thereof, whether directly or indirectly with any country or region subject to an embargo including but not limited to Cuba, Iran, North Korea, Sudan, Syria, and Crimea.

AppDynamics Concepts

Related pages:

- Flow Maps
- SaaS Domains and IP Ranges
- Tiers and Nodes

This page describes the AppDynamics Application Performance Management (APM) Platform.

AppDynamics APM Platform Overview

The AppDynamics APM Platform enables management and monitoring of your application delivery ecosystem, ranging from mobile/browser client network requests to backend databases/servers and more.

AppDynamics APM offers a global view across your application landscape and allows you to quickly navigate through the distributed application into the call graphs/exception reports generated on individual hosts.

Application Performance Monitoring

At the tier level, AppDynamics provides a view of the runtime operation of your code via an AppDynamics App Server Agent. The agent detects calls to a service entry point at the tier and follows the execution path for the call through the call stack. It sends data about usage metrics, code exceptions, error conditions, and exit calls to backend systems to the Controller, either a SaaS or on-premises:

To get started with application monitoring, see Install App Server Agents.

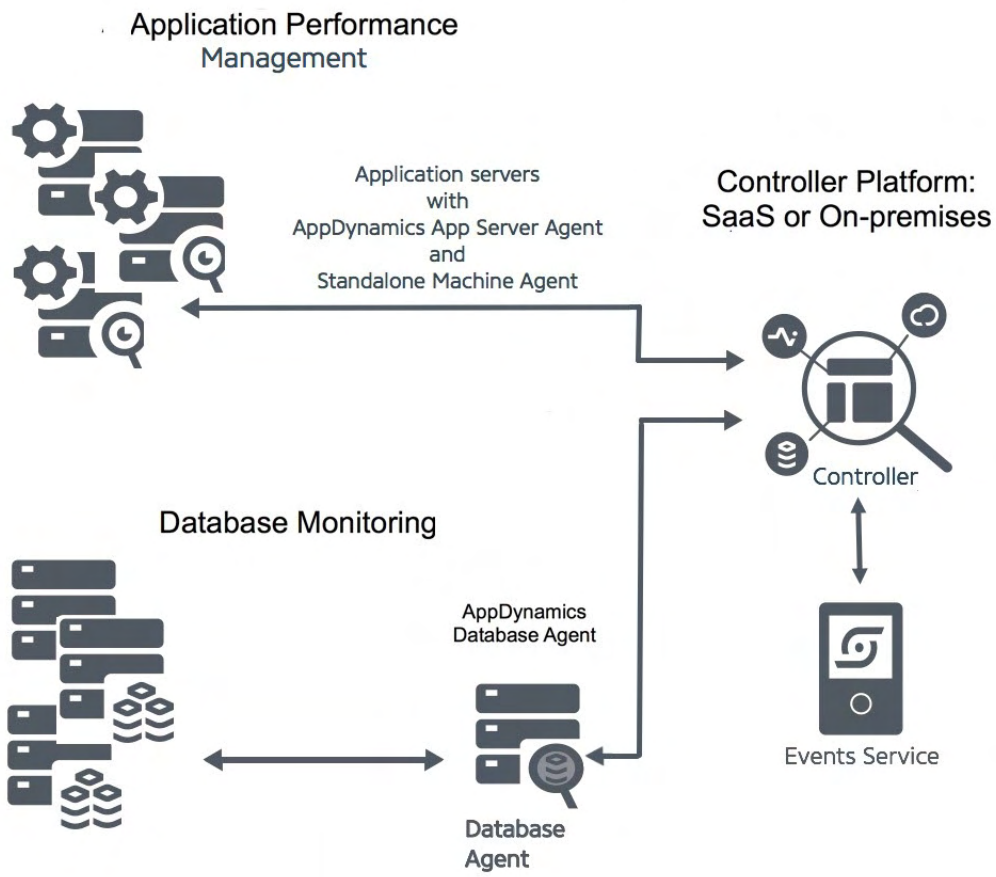
Most application environments contain more than one application server. They may contain distributed, interconnected servers and processes that participate in fulfilling a given user request. In this context, AppDynamics tracks transactions across distributed, heterogeneous services.

Infrastructure Visibility with Database Visibility

For greater visibility into your application delivery environment, you can add AppDynamics Database Visibility to the deployment.

App Agents provide information about calls to backend databases, including errors and call counts. The Database Visibility module extends your visibility into the workings of the database server itself by providing you with information about query execution and performance with an agent-less profile.

AppDynamics Server Visibility contributes to your view of the data center by adding valuable information on the performance of the machines and networks in your environment.



In this deployment, the Database Agent collects information from the database servers and sends it to the Controller, which persists some of that information in the Events Service. Database Analytics features may use the Events Service, the document storage component of the platform that AppDynamics has optimized for searching and storing high volumes of information.

End User Monitoring for Client Experience

While server-side monitoring provides insight into the end user's experience with application performance and suggests performance improvements to the server, end-user monitoring extends those insights from the initial client request to the client device response. AppDynamics End User Monitoring allows you to collect the information about where your requests are coming from, what devices/channels your users are using, and your code performance once deployed on your users' devices. Additionally, AppDynamics provides you with the visibility you need to investigate mobile crashes by displaying stack traces and other contextual data at the time of the crash and tying that to the Business Transaction data from the server.

Business iQ and Analytics for Business Impact

How does the overall performance of your application environment affect your business? Business iQ, powered by AppDynamics Analytics, helps you understand how the performance of your application environment and end-user applications ties to the business data of the transactions. It lets you sort, order, and understand the data that composes the Business Transactions. It also enables you to drill into the varieties of log data that your environment generates. See Using Analytics Data for information about how to install and use Analytics.

Using Metrics

A metric is a particular class of measurement, state, or event in the monitored environment. Many defaults relate to the overall performance of the application or business transaction, such as request load, average response time, or error rate. Others describe the state of the server infrastructure, such as percentage CPU busy or percentage of memory used.

Agents register the metrics they detect with the Controller. They then report measurements or occurrences of the metrics (depending on the nature of the metric) to the Controller at regular intervals. You can view metrics using the Metric Browser in the Controller UI.

Information points are a particular type of metric that enables you to report on how your business (as opposed to your application) is performing. For example, you could set up an information point to total the revenue from the purchase on your website of a specific product or set of products. You can also use information points to report on how your code is performing, for example, how many times a specific method is called and how long it is taking to execute.

You can create extensions that use the machine agent to report custom metrics that you define. These metrics are base-lined and reported in the Controller, just like the built-in AppDynamics metrics.

As an alternative to using the Controller UI, you can access metrics programmatically with the AppDynamics APIs.

Baselines and Thresholds

The AppDynamics Platform uses both self-learned baselines and configurable thresholds to help identify application issues. A complex distributed application has a large number of performance metrics, and each metric is important in one or more contexts. In such environments, it is difficult to:

- Determine the values or ranges that are normal for a particular metric
- Set meaningful thresholds on which to base and receive relevant alerts
- Determine what is a 'normal' metric when the application or infrastructure undergoes change

For these reasons, anomaly detection based on dynamic baselines or thresholds is one of the essential features of the AppDynamics platform.

The AppDynamics platform automatically calculates dynamic baselines for your metrics, defining what is 'normal' for each metric based on actual usage. Then the platform uses these baselines to identify subsequent metrics whose values fall out of this normal range. Static thresholds that are tedious to set up and, in rapidly changing application environments, error-prone, are no longer needed.

You can create health rules with conditions that use baselines, allowing you to trigger alerts or kick off other types of remedial actions when performance problems are occurring or may be about to happen. See Alert and Respond and Health Rules and Dynamic Baselines for more detail.

AppDynamics thresholds help you to maintain service level agreements (SLAs) and ensure optimum performance levels for your system by detecting slow, very slow and stalled transactions. Thresholds provide a flexible way to associate the right business context with a slow request to isolate the root cause. See Transaction Thresholds.

Health Rules, Policies, and Actions

AppDynamics uses dynamic baselining to establish what is considered normal behavior for your application automatically. Then you can set up health rules against those standard baselines (or use other health indicators) to track non-optimal conditions. A health rule might be, for example, to create a critical event when the average response time is four times slower than the baseline.

Policies that allow you to connect such problematic events (such as the health rule critical event) with actions that can trigger alerts/remedial behavior addresses the system's issues long before your users will be affected.

AppDynamics supplies default health rules. You can customize the default health rules and create new rules specific to your environment.

The out-of-the-box health rules test business transaction performance as follows:

- Business Transaction response time is much higher than normal
  Defines a critical condition as the combination of an average response time higher than the default baseline by three standard deviations and a load greater than 50 calls per minute. This rule defines a warning condition as the combination of an average response time higher than the default baseline by two standard deviations and a load greater than 100 calls per minute.
- Business Transaction error rate is much higher than normal
  Defines a critical condition as the combination of an error rate greater than the default baseline by three standard deviations and an error rate higher than ten errors per minute and a load greater than 50 calls per minute. This rule defines a warning condition as the combination of an error rate greater than the default baseline by two standard deviations and an error rate greater than five errors per minute and a load greater than 50 calls per minute.

For more information, see Alert and Respond.

Infrastructure Monitoring

While Business Transaction performance is typically the focus of a performance monitoring strategy, monitoring infrastructure performance can add insight into underlying factors about performance. AppDynamics can alert you of the problem at the Business Transaction and infrastructure level.

AppDynamics provides preconfigured application infrastructure metrics and default health rules to enable you to discover and correct infrastructure problems. You can also configure additional persistent metrics to implement a monitoring strategy specific to your business needs and application architecture.

In addition to health rules, you can view infrastructure metrics in the Metric Browser. In this context, the Correlation Analysis and Scalability Analysis graphs are useful to understand how infrastructure metrics can correlate or relate to Business Transaction performance.

Integrating and Extending AppDynamics

AppDynamics provides many ways for you to extend AppDynamics Pro and integrate metrics with other systems. The AppDynamics Exchange contains many extensions that you can download and if you cannot find what you need, you can develop your own.

AppDynamics Extensions are available in the following categories:

- Monitoring Extensions add metrics to the existing set of metrics that AppDynamics agents collect and report to the Controller. These can include metrics that you obtain from other monitoring systems. They can also include metrics that your system extracts from services that are not instrumented by AppDynamics, such as databases, LDAP servers, web servers, or C programs. To write specific monitoring extensions, see Extensions and Custom Metrics.
- Alerting Extensions let you integrate AppDynamics with external alerting or ticketing system and create custom notification actions. To learn how to write specialized custom notification see Build a Custom Action. Also, see Email Templates and HTTP Request Actions and Templates.
- Performance testing extensions consist of performance-testing extensions.
- Built-in integration extensions are bundled into the AppDynamics platform and only need to be enabled or configured. These include:
    - Integrate AppDynamics with Splunk
    - Integrate AppDynamics with DB CAM

To create custom extensions and integration components for AppDynamics, see AppDynamics APIs.

Deployment Planning Guide

This page describes best-practice deployment planning guidelines for the AppDynamics Application Performance Management (APM) Platform.

It is important to plan your AppDynamics deployment according to your environment. Taking time to outline your system strategy provides a smooth process flow. Use these guidelines to start working with AppDynamics.

Once you install the necessary agents, AppDynamics automatically builds an environment of your applications.

The Controller:

- Monitors your application workload.
- Uses machine learning to determine what is normal for your environment.
- Applies sensible defaults for detecting abnormal activity and application errors.

You can start using AppDynamics dashboards, flow maps, and monitoring tools in the Controller UI immediately without instrumentation and configuration. Later, you can customize the configuration for your specific environment and requirements.

> AppDynamics does not support user or agent requests that originate from any URL other than the Controller URL provided at registration or otherwise edited by AppDynamics. Due to system security improvements, requests from custom URLs will be rejected and may cause a service disruption.
>
> If a custom URL is required, we recommend that you create a forward proxy in your environment.

Network Port Requirements

To deploy AppDynamics, you may need to modify network components configuration to permit access to ports used in the AppDynamics deployment. These pages describe the specific ports used in the system:

- For SaaS: SaaS Domains and IP Ranges.
- For the on-premises platform: Port Settings.

Getting Started

Related pages:

- Release Upgrade Checklist
- Ask the AppDynamics Community
- Help and Support
- AppDynamics University

This page provides an overview of the options for creating an account and how to get started using your AppDynamics online Account Management Portal and Application Performance Monitoring (APM) Platform.

As part of your deployment, the AppDynamics APM provides:

- A Controller for data collection.
- A Controller UI for data analysis.
- An online Account Management Portal to manage your account, users, subscriptions, license usage, and role functionality.

As a best practice, we recommend that you follow the Deployment Planning Guide and create a deployment plan for your environment and needs.

Create an Account

Your first step to working with AppDynamics is to create an account. AppDynamics provides both SaaS and on-premises deployment options to fit your data security needs, application workflow, and business environment. Reviewing the Deployment Planning Guide will help you determine if the type of data you will be monitoring requires a SaaS, on-premises, or hybrid solution.

AppDynamics has implemented restrictions and will not enter into any negotiation with, or seek to solicit any interest from, any third party in relation to the sale or use of the AppDynamics products or software or any part thereof, whether directly or indirectly with any country or region subject to an embargo including but not limited to Cuba, Iran, North Korea, Sudan, Syria, and Crimea.

Choose an account type and deployment option.

Gain System Access

After you complete registration and create an account, you will have access to your Account Management Portal for online account administration and, if you opted for a SaaS deployment, access to your hosted Controller platform.

Online Account Administration

The Account Management Portal provides a secure online area where administrators can:

- Manage the company account and personal profile separately.
- See a company overview of account details, subscription information, open support tickets, and active resource projects.
- Launch available AppDynamics Controllers.
- Add, edit, and delete users, as well as administer user groups, subscriptions, and roles.
- Download agents, platforms, and product updates.

Your account type will determine the services available in the Account Management Portal. See Account Overview.

1. Access the Welcome email sent to the address you used to create your account.
2. Click **Get Started**.
3. Follow the instructions.

Controller UI

An AppDynamics Controller is fundamental to every deployment. Integrated agents collect data from a monitored environment and send it to the Controller. The Controller UI is where you can view, understand, and analyze that data.

How you access your controller depends on your deployment. Your account will include a subscription license for each product you purchased during registration. A subscription can have one or more modules and you can have multiple subscriptions, each one having a distinct lifecycle.

Choose your deployment configuration.

Install Application Agents

An AppDynamics deployment consists of multiple application agents to monitor different components in your environment. For installing most types of agents, you can use the **Getting Started Wizard** in the Controller UI.

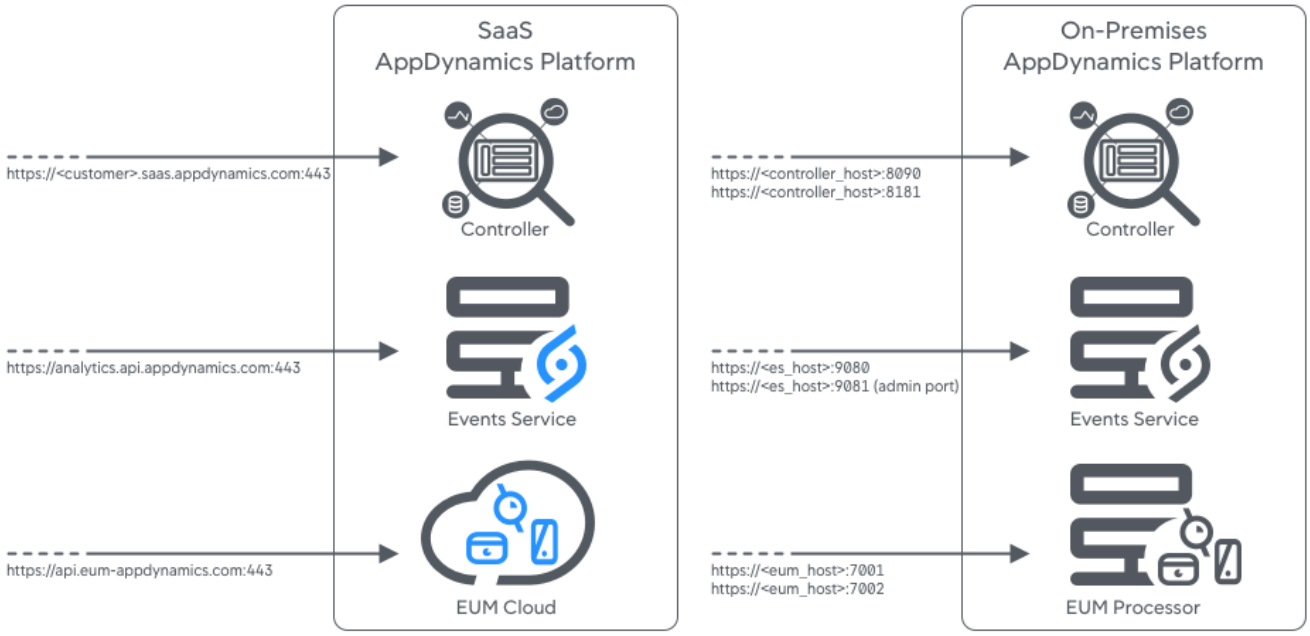
Log in to the Controller UI and navigate to **Applications** > **Create an Application** > **Create an Application using the Getting Started Wizard**. Follow the instructions in the Wizard to install agents in your application environment.

Review the Install App Server Agents documentation for instructions and guidance. AppDynamics provides an Agent Installer that simplifies the agent installation process and streamlines the deployment of the Java and Machine Agents.

Connect Agents to the Controller

The **Getting Started Wizard** automatically configures settings for connecting agents to the Controller. If you manually download agents, you must configure the Controller connections. You may need to adjust the configuration of network components, such as firewalls or proxies, to permit traffic from the agent to the Controller. The connection is unidirectional, meaning that agents always initiate the connections to the Controller.

The following graphic shows the connections agents use to reach the platform. The actual connections vary between on-premises and SaaS platforms.



Continue with Your AppDynamics Deployment

Best Practice

We recommend that you follow the Deployment Planning Guide and create a deployment plan for your environment and needs.

You can start using AppDynamics immediately without instrumentation and configuration. Then, at a later date, you can optimize the configuration to make the best use of AppDynamics for your environment and requirements. After you install agents and restart your applications, AppDynamics automatically builds an environment of your application. You can see the model in the dashboards and flow maps view of the Controller UI. The Controller monitors your application workload, uses machine learning to determine what is normal for your environment, and applies sensible defaults for detecting abnormal activity and application errors.

The timeline in the Deployment Planning Guide outlines a typical start to your AppDynamics journey.

Tutorials

Learn more about specific areas of the AppDynamics APM Platform by following one of these tutorials:

- Platform Installation Quick Start (on-premises installations only)
- Install the Java Agent
- Install the .NET Agent for Windows
- Manual Injection of the JavaScript Agent
- Instrument iOS Applications
- Instrument Android Applications
- Instrument Xamarin Applications
- Instrument Applications with the IoT C/C++ SDK

- Instrument Applications with the IoT Java SDK

Download AppDynamics Software

This page describes AppDynamics Download options and provides links to instructions for installing the appropriate AppDynamics software.

> ⚠ AppDynamics has implemented restrictions and will not enter into any negotiation with, or seek to solicit any interest from, any third party in relation to the sale or use of the AppDynamics products or software or any part thereof, whether directly or indirectly with any country or region subject to an embargo including but not limited to Cuba, Iran, North Korea, Sudan, Syria, and Crimea.

AppDynamics Download Options

You can download the AppDynamics software packages available for your account based on your subscription type from AppDynamics Downloads.

- If you purchased a SaaS subscription exclusively, you have access to Agents downloads.
- If you purchased an on-premises or dual subscription, you also have access to Platforms downloads.

Once you have an AppDynamics account:

1. Log in to accounts.appdynamics.com.
2. Select **Downloads.**
3. Follow the instructions on the Account Management Portal Downloads page to:
    - Download agents
    - Download the on-premises platform
    - Automate downloads with cURL
    - Validate package downloads

Deployment Options

This page describes the available AppDynamics deployment options.

> Before you select your AppDynamics deployment option, review the Deployment Planning Guide.

AppDynamics provides two deployment options:

- A cloud-based Software as a Service (SaaS) solution hosted and maintained by AppDynamics.
- An on-premises deployment where you install and manage all the platform components internally.

Every deployment includes:

- A Controller for data collection.
- A Controller UI to view, understand, and analyze the data.
- An online Account Management Portal to manage your account, users, subscriptions, license usage, and role functionality.

Every deployment requires:

- Installed agents to collect data from a monitored environment.
- Network port requirements.
- Health rules to monitor system parameters.
- Validated users.

> AppDynamics does not support user or agent requests that originate from any URL other than the Controller URL provided at registration or otherwise edited by AppDynamics. Due to system security improvements, requests from custom URLs will be rejected and may cause a service disruption.
>
> If a custom URL is required, we recommend that you create a forward proxy in your environment.

Controller UI Overview

This page describes the features of the AppDynamics Controller UI, which is the primary interface you use to monitor, troubleshoot, and analyze your entire application landscape from backend infrastructure/servers to the end user's application.

Supported Web Browsers

The AppDynamics Controller UI is an HTML5-based browser application that works best with the latest version of any modern browser.

The Controller UI has been tested with and supports the latest versions of these browsers:

- Firefox
- Google Chrome
- Internet Explorer
- Microsoft Edge
- Safari

Certain types of ad blockers can interfere with features in the Controller UI. We recommend disabling AdBlock when using the Controller UI to avoid issues loading particular screens.

General Navigation

As you start using the Controller UI to configure AppDynamics and monitor your environment, it is helpful to familiarize yourself with the most commonly-used elements of the Controller UI that are frequently referenced in this documentation.

The precise controls and features that appear in the UI depend on your user account permissions and product modules licenses. For example, you would have additional controls in application monitoring screens with an Analytics license. See Create and Manage Custom Roles.

The UI elements include:

- Top Navigation Bar: Use to navigate the major product areas and cross-product feature areas, such as **Alert & Respond** and **Dashboard & Reports**.
- Settings Menu: Access pages where you can customize the UI environment, view your account information, or, if permitted by your user role, configure administration settings such as adding user accounts, configuring authentication, and more.
- Time Range Menu: Controls what metric data AppDynamics represents in the UI in the selected time range. See the following section for more information on the time range menu.
- Left Navigation Menu: Use to access sections specific to the product area.

Administration Settings Overview

Create and administer users for a specific tenant in the Tenant UI. Use the Account Management Portal for global user account management and tenant license subscription assignments.

You must be logged in as a user with the Account Owner role to see Administration options.

The Administration UI allows you to set up several user options specific to the Tenant. If you have multiple SaaS accounts or a multi-tenant on-premises Controller, each account must be configured individually for each Tenant.

- **Users**—Create and administer users.
- **API Clients**—Create and use an API Client to provide secure access to the Tenant through REST API calls. See API Clients.
- **Groups**—Create and use groups to assign and manage roles for users collectively.
- **Roles**—Create and use roles to define user privileges in the Tenant UI.
- **Authentication Provider**—Configure external authentication settings for the tenant user. See LDAP Authentication or SAML Authentication.
- **Integrations**—Set up extension integrations to supplement AppDynamics capabilities. See Integration Modules.

Time Ranges and Settings

The Time Range menu is available throughout the UI. Your selection determines the range of data displayed in the UI, except in the case of pages such as the Scalability Analysis, where other time contexts apply. When you choose a new time range, it remains the active range as you navigate to other pages.

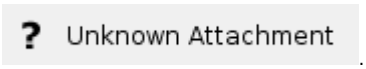
You can select a preconfigured **Standard** time range or enter a **Custom** one. Custom time ranges can be set to terminate in the past, making them useful for analyzing trends over time.

To create a custom time range:

1. Click **Custom** and select  .
2. Enter your start and end date and time selections in the dialogues.
3. Click  .
4. Enter a **Name** and optional **Description**.

5. Optionally, select **Share with Everyone** to share the range with all the other users of the Controller.

To manage a custom time range:

1. Click 

.
2. Select a time range.
3. Select to **Add**, **Edit**, **Delete**, or **Copy**.
4. Click **Apply** to save the changes and update the session.

To enable 5 and 15-minute auto-refresh for a flow map:

1. Select **Standard**.
2. Select 5 or 15 minutes to display the **Auto-refresh** checkbox at the bottom of the list.
3. Select the **Auto-refresh** checkbox to automatically refresh the data.

Share Views With Other Users

Troubleshooting an application issue is often a collaborative effort that may involve multiple users. You can share a particular view of the UI with a shareable URL that retains the time range currently active in the UI. Time ranges that are relative to the current time, such as "last 15 minutes," are represented as fixed time ranges in the URL.

To share a view:

1. Select **Settings** > **Copy a link to this screen to the clipboard**.
2. Paste and share the URL where desired.

AppDynamics Mobile App

The AppDynamics mobile application lets you monitor the health and performance of your applications on iOS and Android devices. Alerts on a device can notify you of application issues within minutes of their occurrence.

Supported Environments and Versions

Using the AppDynamics Mobile App requires:

- Controller >= 4.0
- For the device, these operating systems:
  - iPhone, iPad, or other iOS-based devices: iOS >= 8.0
  - Android: Android >= 4.0 (Ice Cream Sandwich)

You can view custom dashboards on the mobile app, but node- or tier-level custom dashboards are not available for viewing in the Mobile App.

Get the Mobile App

To get the AppDynamics Mobile App:

- For an iPhone, search for AppDynamics in the Apple App Store or directly from the AppDynamics Mobile App page.
- For an Android device, search for AppDynamics in Google Play.

After installation, set up your account from within the application using one of two methods:

- By scanning a QR code in the Controller UI with your mobile device.
- By configuring the Controller account settings manually.

Configure your Account by Scanning a QR Code

To set up your account by QR code:

1. In the Controller UI, click Settings and select **My Preferences**.
2. Click the **View Configuration Code** link.
3. Follow the instructions to set up your device with the QR code.

Configure your Account Manually

1. Launch the mobile app.
2. Click **Add New Account** if you have already set up an account and want to set up another.
3. Enter your Controller URL and click **Next**.
   If you are using SSL, make sure to check the **Use HTTPS** box.
4. Enter your account name based on your Controller type:
   - For a SaaS or multi-tenant Controller, enter the account name provided when you signed up for a SaaS Controller or the account name added by your administrator for a multi-tenant Controller.
   - For a single-tenant on-premises Controller, use the built-in default account, `Customer1`.
5. Click **Next**.
6. Enter your username and password and click **Next**.

Enable Notifications on the Device

You can set Mobile App to notify you when selected applications have an event or anomaly, which can help you to significantly reduce downtime. When notifications are enabled, you will receive them even when Mobile App is closed.

> An on-premises Controller is not configured to generate push notifications by default. To enable push notifications, follow the instructions to Enable Notifications from an On-Premises Controller.

If notifications fail, AppDynamics captures the error in the Controller log.

Enable Notifications from an On-Premises Controller

To generate notifications, the on-premises Controller must be able to reach this domain address: `https://mobile-push.api.appdynamics.com/`

You may need to configure your network or intervening firewalls to enable the Controller to access the domain address. Once you have ensured that the Controller can access the AppDynamics site:

1. Open the Administration Console.
2. From the **Controller Settings** page, locate and set the `push.notification.service.enabled` flag to `'true'.` The change takes effect immediately.

AppDynamics Support

Have a question about using AppDynamics or running into a problem? This page describes the necessary steps to help resolve your questions.

Search the Documentation

You can search for information from:

- The **Search** field on the right side of the top menu bar to initiate a search of the entire documentation set.
- The **Search** field in the left navigation pane to search within the current version only.

You can search across AppDynamics information resources, including the Community, Knowledge Base, Support, and more in the AppDynamics Support Center.

Ask the Community

If you have questions about using AppDynamics, ask the AppDynamics Community.

Contact Support

If you need further assistance, contact your Account Representative or Technical Support.

For Technical Support, click the Help tab while logged in with your AppDynamics account in the AppDynamics Support Center.

When requesting support, attach relevant logs for your issue(s):

- For log files from the Controller, see Platform Log Files.
- For the heap, histogram, and thread dumps, see Controller Dump Files.

AppDynamics has implemented restrictions and will not enter into any negotiation with, or seek to solicit any interest from, any third party in relation to the sale or use of the AppDynamics products or software or any part thereof, whether directly or indirectly with any country or region subject to an embargo including but not limited to Cuba, Iran, North Korea, Sudan, Syria, and Crimea.

Account Management

AppDynamics provides account administration through an Account Management Portal as well as the Tenant UI.

Use the Account Management Portal to:

- Manage user accounts globally.
- Assign Tenant license subscriptions to users.
- Inactivate or edit accounts for all AppDynamics components simultaneously.

Use the Tenant Administration UI to:

- Create and add users to the Tenant.
- Set role and group security specific to the Tenant.

User Account Management

- Account Management Portal
- User Management
- Tenant User Management
- Create and Manage Custom Roles

Permissions Management

- Application Permissions
- Transaction Analytics Permissions
- Custom Dashboard Permissions
- Database Permissions
- End-User Monitoring Permissions
- Server Visibility Permissions

Licensing and External Authentication

- License Management
- License Usage
- LDAP Authentication
- SAML Authentication

Additional Resources

- Download AppDynamics Software
- University
- Resources
- Community

Account Management Portal

This page describes the secure online Account Management Portal for managing AppDynamics accounts.

The Account Management Portal is a secure online means for account administrators to manage users and account resources and for non-administrators to edit their profile, review account details, and access AppDynamics Downloads.

> ⚠ Use the Accounts Management Portal to fully inactivate or edit an account for all AppDynamics components and Tenants simultaneously. User account management through a Tenant UI affects permissions for that Tenant only.

Account Management Portal Overview

These secure administrative functions are available:

- Overview—This page displays selective account summary information and tools based on your user type and role permissions.
- License Usage—Illustrates product usage by subscription.
- User Management—Allows user account administrative functions.
- Resources—Displays a list of the services, support, and education resources available for your account.
- Downloads—Allows downloading AppDynamics software packages that are available for your account.

These non-administrative functions are available to all users from any active tab:

- Profile—Provides user profile options.
- University—Provides training information and learning tools.
- Community—Provides a discussion forum, knowledge base, and user community.

Upon login, the Account Management Portal opens to the Overview tab. Review the User Permissions Matrix for your user role options.

Edit Profile Settings

To edit different aspects of your profile, click your name and select **Profile**. Once finished with your edits, click your name and select **Account** to return to the Overview page.

To edit your user details:

1. Select the Personal Summary tab and click **Edit**.
2. Update your **User Details** accordingly.
3. Click **Submit**.

To update your password:

1. Select the Password tab and click **Edit**.
2. Enter your current password.
3. Create a new password according to the listed rules and click **Save**.

To customize your notifications:

1. Select the Notifications tab.
2. Select the **Notify me when a new version of AppDynamics is released** checkbox to receive emails of updates, hotfixes, and new releases.
3. Select the checkbox in the **License Notifications** column heading to receive emails whenever a license detail changes for all associated licenses.
4. Alternatively, you may select individual license names to reduce notifications.

   > ⓘ The primary contact must receive license notifications. Click **Email Us** in Account Details to contact your AppDynamics support representative to change the primary contact.

5. Select the checkbox in the **Maintenance Notifications** column to receive emails when planned maintenance or outages occur.
6. Alternatively, you may select individual license names to reduce notifications.
7. Click **Save**.

Access AppDynamics Downloads

You can download AppDynamics software packages available for your account based on your subscription type from AppDynamics Downloads. See Download AppDynamics Software.

Select **Downloads** and use the links at the top of the page for quick access to:

- Our **Getting Started** page for instructions to begin working with your subscription.
- License **Entitlements**.
- Our **Updates** page for product announcements.

- Instructions for the **Automation** of new version downloads.

Access AppDynamics University

AppDynamics University provides subscription-based educational tools to help you configure, manage, and monitor applications on your platform. Several subscription options and private training options are available.

The University offers self-paced and instructor-led courses, private training, and certification programs based on your subscription. Administrators have the option to delegate subscriptions to other users within their organization to bolster team competency.

Access AppDynamics Community

AppDynamics Community is an online customer platform where you can join discussion forums, access our knowledge base, and post questions to others in the user community.

1. Click your name and select **Community**.
2. Sign in using your AppDynamics credentials.

> Your profile menu on this site mirrors your Account Management Portal profile menu.

3. Review the **Latest Solutions** to current issues.
4. Review the **Latest Discussions** among users concerning current technologies.
5. Review the **Latest Posts from the Knowledge Base**.
6. Click **Start a Discussion** to ask questions of community users and experts.
7. Click **Access Knowledge Base** to search for answers to known issues.
8. Click a contributor's name to send them a message and view their profile.

Review Account Details

The Account Details section displays:

- The information provided when your company registered with AppDynamics.
- AppDynamics customer support contacts assigned to your account.
- An online tool for scheduling a 1:1 meeting with a technical consultant.
- An automated email link directed to your assigned customer success and/or sales representative.

Access a Controller

AppDynamics refers to a Controller license as a subscription. You can view your subscriptions and launch an available Controller in several ways.

Users must be assigned a license to a Controller for access. When a non-administrative user is assigned a Controller license, they become a Licens

To find a specific subscription, you can:

- Click a table heading to sort and include other Controllers in the summary view.
- Begin typing in the **Search subscription** field. The list will automatically display only those Controllers with the characters you enter.
- Click **View all subscriptions** to display the full list.

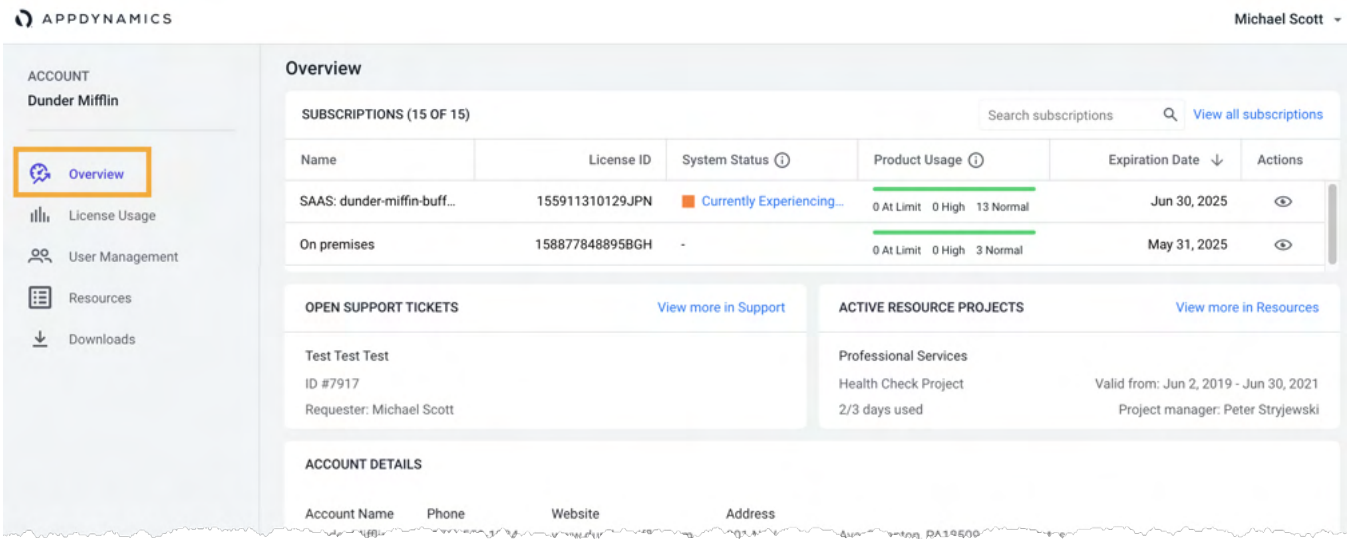
To launch a Controller, you can:

- Click a subscription name.
- Click your name and select a Controller from the list, or select **View All**.
- Click your name, select **Profile > Personal Summary**, and select a Controller URL from the **Assigned Subscriptions** list.

Overview

This page describes how to navigate the **Overview** page in the Account Management Portal.

The **Overview** page is role-based and presents a summary of account components according to the user's permissions. Components are presented in grouped panes, each with a link to access the full detail page.



Select a user type to see the options that display for that role.

License Usage

This page describes how to navigate the License Usage page in the Accounts Management Portal.

You must be a Company Administrator or License Administrator to access the License Usage page.

Navigation Overview

The License Usage page provides summary usage data for Tenant license subscriptions associated with your License Administrator account. The usage shown is based on the current or most recent usage period, including the data captured and aggregated from the previous day.

You can filter or arrange your subscription list in several ways:

- Click **View options** to select which Tenant type and subscription status to display.
- Begin typing in the autocomplete **Search** field to filter the list as you type.
- Click a column heading to sort the subscription list according to that column.
- Click (where available) to display information about the corresponding data.

Access Product Usage Detail

AppDynamics captures usage data in five-minute intervals and aggregates it into larger intervals.

There are several product families and each family has several products associated with it. Usage is measured differently for each product. See License Entitlements and Restrictions.

1. Select a **Subscription** to access usage detail for each product associated with the license, including:
   - Each **Product** associated with the selected license.
   - The **Product Family** category.
   - The current **Usage** of the product for a specified usage period.
   - The **Usage Period** in which the product can consume licenses before the usage limit is reset.
     The usage periods that display for each product in the detailed usage graphs are set as default time ranges for efficient page loading.
2. Select a **Product** to view:
   - License information
   - Maximum and average usage metrics
   - Total (cumulative) license usage for that product and product family.
     For the APM Any Language product, select or deselect the language to view or remove usage data in the graph.
3. Click one of the in the page navigation bread crumbs to select a different license or different product within that license.
   Only products associated with the chosen license display in the dropdown menu.
4. Click to adjust the time range, date, and time zone of data you want to view in the graph.
   You can select daily, hourly, weekly, or monthly data granularity.
   If you select a timeframe that is too long for the aggregation, certain granularity options are not visible.
   For example, if you choose a timeframe longer than one month, hourly granularity is disabled.
5. Click to toggle the data display as either a line graph or bar graph.
   For certain products under the Application Performance Management (APM) and End User Monitoring product families, there are two graph types. You can specify the data granularity and timeframe you want to view in both types.
   - A cumulative plot line graph for a given usage cycle.
   - A non-cumulative plot line graph for the same usage cycle.
6. Review the Max and Average usage metrics.
7. Hover over a data point on the graph to view granular metrics.
   If you set a timeframe to less than one month, hourly metrics are available. If you select a timeframe greater than or equal to one month, daily data is available.
8. When available, select **View usage as *<metered units>*** or **View usage as license units**.
   - Viewing usage by metered units displays consumption according to how many events you consumed due to license unit consumption. Each product has distinct meters on which licenses are consumed.
   - Viewing usage by license units displays how many licenses have been consumed.
     The usage data is the same; the x-axis displays the same usage on both graphs, while the y-axis reflects the view you select.

License units are always rounded up to the next highest integer because that is the minimum amount you would need for your monitoring.

Max and Average Usage Metrics

You can view these metrics reporting for the timeframe you chose in the date range selector UI:

- **Max**—The maximum usage of all agents aggregated over the chosen timeframe. It is the maximum of all the data points on the graph.
- **Monthly Average**—The average of all agents reporting usage during the chosen timeframe. It is the average of all the average values on the graph.

For certain products under the Application Performance Management (APM) and Infrastructure Visibility product families, the x-axis is the time frame and the y-axis is the number of license units used.

The APM Any Language product includes multiple agents. Click a corresponding checkbox in the filter bar to drill down into usage for a particular agent (such as Java). Agents that do not report data during the selected timeframe are grayed out in the filter.

You can interpret metrics according to the view:

- When you select all agent types, the maximums and the averages of all agents display reporting usage over the selected timeframe.
- When you select one agent type, the results are restricted to that agent's data.
- When you select a subset of agent types, the maximums and averages are calculated based only on the agents chosen.

Cumulative Plot Graph

This graph displays the total number of units used during either a usage cycle or a timeframe you choose.

- The y-axis represents the units used, either as the metered unit or as license units.
- The x-axis represents the time period.

For a given usage period, the **Usage Overview** will show usage as a percentage to display the quantity consumption against your total allocation.

Real User Monitoring, Synthetics Monitoring, and Transaction Analytics are a few examples of products that are metered on a cumulative basis.

Non-Cumulative Plot Graph

This graph plots the usage for a product's given usage cycle or timeframe and the granularity you select. You can use the non-cumulative graph to help you determine how much of the product is being used at a regular cadence and whether there are spikes in usage.

Hover over a data point on the graph to view granular metrics. You can also view an average data point for the set granularity.

APM Any Language, Server Visibility, and Databases Monitoring are a few examples of products that are metered on a non-cumulative basis.

User Management

This page provides details for managing users in the Account Management Portal.

Company Administrators and License Administrators can manage user access for AppDynamics accounts through the Account Management Portal. User accounts on a Tenant are managed based on the authentication options selected in the Tenant UI. See Tenant User Management.

> ⚠ Editing an account through the Accounts Management Portal affects the user for all AppDynamics components and some Tenant rights simultaneously. User account management through a Tenant UI affects permissions for that Tenant only.

Every AppDynamics user that has been registered in the User Management UI has subscription-based access to AppDynamics University and unlimited access to AppDynamics Community for FAQs and user forums.

User Management Overview

The Accounts Management Portal allows administrators to create and globally manage user accounts through the **User Management** UI. SaaS customers use the same login for both the Tenant and the Account Management Portal. However, on-premises users may have a different login credential for their Controller and the Account Management Portal.

User Status Options

A user account can have one of three possible statuses:

- Active—User has a valid email account that authenticates with the AppDynamics Identity Provider (IDP).
- Pending—User has been created in the system, but the email is not yet validated by the user.
- Inactive—User is restricted from accessing AppDynamics components.

When a user is created through the Account Management Portal, they receive a time-sensitive email prompting them to activate their account by creating a username and password. A user is in Pending status until they have completed the activation.

Once complete, the user can:

- Access to protected pages on appdynamics.com and accounts.appdynamics.com
- Submit Support tickets.
- Download AppDynamics Software with cURL
- Manage their own profile.

However, they do not have access to a Tenant until or unless an administrator adds the user account to a Tenant.

UI Options

The User Management UI displays users in a grid with sortable columns and an autocomplete search feature.

Editing functions become visible based on the selected user's status.

Option User Status	Add	View	Edit	Deactivate	Activate	Delete
Active	✓	✓	✓	✓	✗	✗
Pending	✓	✓	✓	✗	✗	✓
Inactive	✓	✓	✓	✗	✓	✓

User Security Role Options

There are four security roles available through **User Management** > **Edit**. This table explains the functionality and the User Permissions Matrix shows the specific permissions per role.

Role	Functionality
Company Admin	• Has the highest level of access. • Can view all licenses or Tenants the company owns.

License Admin	• Requires one user per Tenant to be designated as a Primary Contact with AppDynamics and is mandated to receive maintenance and licens • Can only view and administer licenses for a license to which they have been assigned. • Cannot administer a Tenant. • Can add or remove license admins on the same license (unless that admin is the Primary Contact).  To remove the Primary Contact from the role, you must open a Support request with AppDynamics and provide a new Primary Conta
Support	Provide managers with the ability to open and manage Support requests with AppDynamics.
Tenant User	• Is assigned to every user created on a Tenant. • Is the most basic user role. Specific permissions for this user are set at the Tenant level. See Tenant User Management and Create and Ma • Shows only once per user regardless of how many tenants they are associated with.

User Permissions Matrix

Roles / Permissions	Company Admin	License Admin	Support User	Tenant User (SaaS)	Company User
Add User	✓	✓	✗	✗	✗
Edit User Name	✓	✓	✗	✗	✗
Edit User Status	✓	✗	✗	✗	✗
Delete User	✓	✗	✗	✗	✗
Assign Company Admin Role	✓	✗	✗	✗	✗
Assign License Admin Role	✓	✓	✗	✗	✗
Assign Support Role	✓	✗	✗	✗	✗
View Professional Services Projects	✓	✗	✗	✗	✗
Assign Education Subscriptions	✓	✗	✗	✗	✗
Edit License MAC Address	✓	✓	✗	✗	✗
View License Usage	✓	✓	✗	✗	✗
Assign Licenses	✓	✓	✗	✗	✗
Launch a Tenant	✓	✓	✗	✗	✗
Log in to an assigned Tenant	✓	✓	✗	✓	✗
View Open Tickets	✓	✗	✓	✗	✗
Create a New Ticket	✓	✗	✓	✗	✗
View Active Resources	✓	✗	✗	✗	✗
Access Downloads	✓	✓	✓	✓	✓
View Account Details	✓	✓	✓	✓	✓
Access AppDynamics University	✓	✓	✓	✓	✓
Access AppDynamics Community	✓	✓	✓	✓	✓

Manage Profile	✓	✓	✓	✓	✓
Customize Notifications	✓	✓	✓	✓	✓

Create and Manage a New User

Resources

This page describes how to navigate the Resources page in the Account Management Portal.

You must be a Company Administrator or License Administrator to access the License Usage page.

The Resources page provides a complete list of services and other components associated with your AppDynamics account. Resources can include:

- Professional Services
- Support
- Education
- Community

Click  (where available) for information about the corresponding resource.

Professional Services

You can use Professional Services to plan deployments, configurations, workstreams, and other technical projects.

The table automatically populates with these details when your AppDynamics account representative registers your account with the requested service:

- Resource name
- Number of days used/total number of days
- Entitlement effective dates
- Project manager

If you need additional services, click **Schedule a call with a consultant** or reach out to your AppDynamics account representatives provided on the Overview page in Account Details.

Support

You can use Support services to troubleshoot your account and technical issues. This section displays your current AppDynamics support agreements and their effective dates.

For additional support:

1. Click **Actions** .
2. Click **Visit Help and Support** for support options.
3. Click **View Support Requests** to review active and past support tickets.

Education

You can view your current Education benefits, including AppDynamics University subscription, Instructor Day(s), and exam vouchers. You can also assign users to Premium University subscription(s) if available. See AppDynamics University Registration and Subscriptions.

The Company Administrator or License Administrator must have access to the account that holds University subscriptions in order to view the vouchers associated with that account.

You may have to provide subscription details when you register for a course or class. These subscription details are available for your Education benefits:

- Standard, premium, enterprise, 12-user education, and exam voucher code(s)
- Number of seats purchased and consumed
- Number of exam attempts purchased and consumed
- Number of training units purchased and consumed
- Number of Instructor Days purchased and consumed
- Validity dates
- Expired vouchers and expired training units

To assign users to a University subscription:

1. Click **user seats activated**.
2. Click **Assign Users**.
3. Enter a valid email address.
   If the user email is not recognized, click **Go to User Management** to add the email address.
4. Click **Assign additional users** as necessary (up to the number of user seats available).
5. Click **Save**.

You can create an alias (friendly name) for your voucher code that relates to the course to make it easier to remember. To add or edit a voucher code or alias:

1. Click **Edit** .
2. Add the voucher code provided by AppDynamics or enter an alias.
3. Click **Submit**.

For additional support:

1. Click **Actions** .
2. Click **Visit University** for learning options.
3. Click **Contact University Team** to enter a message.
   An AppDynamics University Team member will contact you.
4. Click **Learn about subscriptions** for subscription options.

Community

You can use the AppDynamics Community to engage in user discussions and access Knowledge Base articles.

1. Click **Actions** .
2. Click **Visit Community** to view or browse the latest AppDynamics solutions and user discussions.
3. Click **Access Knowledge Base** to browse the latest How-To articles.

Downloads

This page describes AppDynamics Downloads.

AppDynamics Downloads is a central location to download Agents and the AppDynamics On-premises platform. The site provides download availability acc ording to your subscription type and permissions.

> ⚠ AppDynamics has implemented restrictions and will not enter into any negotiation with, or seek to solicit any interest from, any third party in relation to the sale or use of the AppDynamics products or software or any part thereof, whether directly or indirectly with any country or region subject to an embargo including but not limited to Cuba, Iran, North Korea, Sudan, Syria, and Crimea.

Navigation Overview

There are quick links provided to assist with management functions:

- **Getting Started**—provides instructions for creating an account, gaining system access, and so on.
- **Entitlements**—
- Our **Updates** page for product announcements
- Instructions for the **Automation** of new version downloads

Download Agents

AppDynamics agents monitor every line of code in your instrumented environment by using unique tags that are assigned to method calls and request headers. The agents capture performance activity and report to the Controller in real-time. Use the **Agents** download tab to access every available agent in each applicable version for three operating systems to match your environment.

Under Available Downloads, use the filter options or the **Search** field to find an agent.

Filters:

Each filter dropdown menu provides options that, when selected, automatically updates the list of available agents. If no agents display, edit your filter selections.

- **Type**—select one or more agents for download. The four most popular agents display at the top of the list with all additional agents listed alphabetically.
- **Version**—choose the latest agent version or one or more previous versions.
- **Operating System**—select one or more operating systems.
- **Compatible With Controller**—choose the latest Controller version or one or more previous versions.

Search Field:

1. Type the name and/or version of the desired agent in the **Search** field.
2. Press Enter on your keyboard to update the list of available agents for download.
3. If no agents display, there are none that match your search criteria. You may want to try the filter options.

Once you have a list of available agents:

1. Hover over **Checksums** in the **Description** column to view the MD5 and SHA256 hash functions for the component.
2. Click **View** for the agent to review the latest release notes.
3. Click **Download** to obtain the Agent.zip file.
4. Click **cURL** to generate code for cURL that includes a time-sensitive OAUTH access token.
5. Select **Copy to Clipboard** and paste the code in your server, VM, or docker container within an hour.

Download the On-Premises Platform

An on-premises AppDynamics Platform installation consists of several, separately installed, and configured components. These include the Enterprise Console, AppDynamics Cluster Manager, Synthetic Server, Events Service, and the optional EUM Server. Use the **Platforms** download tab to access each platform application in all available versions for three operating systems to build your environment.

Navigate to AppDynamics Downloads and select the **Platforms** tab. Use the filter options or the **Search** field to filter the list of available platform applications.

Filters:

Each filter dropdown menu provides options that, when selected, automatically updates the list of available agents. If no agents display, edit your filter selections.

- **Type**—select one or more agents for download. The four most popular agents display at the top of the list with all additional agents listed alphabetically.
- **Version**—choose the latest agent version or one or more previous versions.

- **Operating System**—select one or more operating systems.

Search Field:

1. Type the name and/or version of the desired component in the **Search** field.
2. Press Enter on your keyboard to update the list of available components for download.
3. If none display, there are none that match your search criteria. You may want to try the filter options.

Once you have a list of available agents:

1. Hover over **Checksums** in the **Description** column to view the MD5 and SHA256 hash functions for the agent.
2. Click **View** for the agent to review the latest release notes.
3. Click **Download** to obtain the software package file.
4. Click **cURL** to generate code for cURL that includes a time-sensitive OAUTH access token.
5. Select **Copy to Clipboard** and paste the code in your server, VM, or docker container within an hour.

Automate AppDynamics Software Downloads with cURL

When you download files, always copy or transfer the files in binary mode. If you want to transfer a file that you have downloaded on Windows to a Linux machine, use binary mode in your transfer program when you move the file to the destination Linux environment.

Where applicable, AppDynamics also publishes software for distribution on package manager repositories such as RPM, npm, pip, NuGet, and more. For products that are available via a package manager, you can find relevant instructions to retrieve and install the software in Install the Machine Agent.

The following steps demonstrate how to use cURL to get the latest version of an AppDynamics software download.

1. Retrieve an OAUTH token with the download scope:

   ```
   curl -X POST -d '{"username": "<username>","password": "<password>","scopes": ["download"]}' https://identity.msrv.saas.appdynamics.com/v2.0/oauth/token
   ```
2. View the latest version of each available product from https://download.appdynamics.com/download/downloadfilelatest. For example:

   ```
   curl https://download.appdynamics.com/download/downloadfilelatest/
   ```
3. Inspect the response (or parse via script) to find the `download_path` of the latest product version.
4. Download the binary by running the following command:

   Replace `access_token` with your OAUTH token from Step 1, and replace `url_to_file` with the `download_path` value from the response in Step 3.

   ```
   curl -L -O -H "Authorization: Bearer <access_token>" <url_to_file>
   ```

Validate Software Package Downloads

You can use checksum validation and a digital signature (for specific packages) to validate software downloaded from AppDynamics Downloads. This validation requires the machine on which you validate a package to include a TLS 1.2 implementation.

Checksum Validation

1. Navigate to AppDynamics Downloads.
2. Click **Checksums** beneath the software package description to display the MD5 and SHA256 checksums.
3. After your download completes, run a checksum tool and compare the results against the checksum information on the downloads page.

Digital Signatures

AppDynamics digitally signs the following packages using a certificate signed by a publicly known certificate authority:

- .NET Agent
- AppDynamics Controller for Windows MSI installer

AppDynamics digitally signs the following packages using a PGP key:

- Java Agent
- Standalone Machine Agent
- Standalone Machine Agent RPM package
- Python Agent pip package

The AppDynamics PGP public key is hosted on https://pgp.mit.edu under the User ID "help@appdynamics.com". For information about using a PGP signature to validate a software package, see Verify AppDynamics Software Downloads with PGP.

Tenant User Management

This page describes managing a user on a Tenant.

What is a Tenant?

An AppDynamics Controller can host one or more accounts, where each account represents one tenant on that Controller.

AppDynamics cloud-based Software as a Service (SaaS) deployment is a multi-tenant environment that allows you to access multiple tenants independently. On-premises deployments are single-tenancy by default, however, you can enable multi-tenancy if necessary. See Multi-Tenant On-Premises Accounts.

Whether you have a SaaS deployment or on-premises deployment, you can manage users through the Tenant UI. While both deployments provide an administrative UI, user functionality differs slightly. Once set up, you can add user accounts in the Tenant UI, allowing other users to access the UI and configure AppDynamics.

Role-Based Access Control (RBAC) Overview

AppDynamics uses Role-Based Access Control (RBAC) to set user permissions and privileges for only those functions necessary in defined job responsibilities. Each user account can have varying levels of access based on their role(s). See Create and Manage Custom Roles.

The tenant can authenticate users against local user accounts or external LDAP or SAML-based authentication providers. The user account for a SaaS Tenant user authenticates through the AppDynamics Identity Provider (IDP) in the Cloud rather than by an external authentication provider. See External Authentication Providers.

A group is a collection of users with a given set of assigned permissions that apply to the users in the group. Groups are used to manage roles for users collectively.

A role is a collection of permissions that define what actions a user can perform; RBAC. When a user is assigned a role, they inherit the specified permissions. A user's group membership and defined role remain constant for the duration of their login.

Permissions grant users the ability to perform an action on the platform. You can set permissions at a granular level to determine:

- Which business applications the user can monitor.
- What parts of the UI are visible.
- Types of configuration changes a user can make.

> AppDynamics University offers courses in Administrator functions.

Tenant User Management Overview

AppDynamics user credentials for both SaaS and on-premises deployments are managed according to the authentication options selected in **Settings** > **Administration** > **Authentication Provider**. There are three user authentication options:

Authentication Provider	User Type	Description
AppDynamics	Local User	⚠ Managing a Local User through the Tenant UI affects permissions for that tenant only. The user account retains Active status with existing permissions on other associated Tenants. Use the Account Management Portal to fully deactivate or edit an account for all AppDynamics components and Tenants simultaneously. • Users authenticate through AppDynamics IDP for SaaS deployment. • AppDynamics manages user account credentials. • User type can exist in parallel and access the system even when using SAML and LDAP authentication.
LDAP	LDAP User	• Users authenticate through your IDP. • You manage user account credentials through LDAP integration. • Non-LDAP users cannot access the system unless they have also been set up as a Local User.

SAML	SAML User	• Users authenticate through your IDP using the SAML 2.0 protocol. • You manage user account credentials through SAML integration. • Non SAML users cannot access the system unless they have also been set up as a Local User.

With a SaaS deployment, when you add a new local user through the Tenant UI, an email is sent to that user's valid address prompting them to create their own profile name and password. The user's email serves as their username. Only the account user can create their own password. Once completed, an account with the proper credentials is added to the tenant and authenticated through the AppDynamics IDP providing the user unified access to the Account Management Portal, University, Community, and role-specific functions on the Tenant UI.

With an on-premises deployment, when you add a new local user through the Tenant UI, an account with the proper credentials is created and stored on that tenant.

You can create and manage users, groups, roles, and permissions on the corresponding page through **Settings** > **Administration**.

- Create and Manage Tenant Users
- Create and Manage Groups
- Create and Manage Custom Roles

Create and Manage Tenant Users

This page explains how to create and administer users for a specific Tenant in the Tenant Administration UI.

Use the Account Management Portal for global user account management, Tenant license subscription assignments, and to fully inactivate or edit an account for all AppDynamics components simultaneously.

Use the Tenant Administration UI to create and manage user options specific to the Tenant. If you have multiple SaaS accounts or a multi-tenant on-premises Controller, you must configure each account individually for each Tenant. See Multi-Tenant On-Premises Accounts.

When creating a new user, AppDynamics recommends:

- Assigning at least one role for the new user. Although it is possible to assign a role after creation, the user has no functionality in the UI until you assign a role.
- For on-premises deployments, use only ASCII characters for usernames and passwords because of browser incompatibilities.

Create a New User

You must be logged in as a user with the Account Owner role to access Administration options and manage users. However, only the user has perm their password. Account owner roles and administrator roles cannot create or change user passwords for SaaS user accounts.

1. Navigate to **Settings** > **Administration** > **Users**.
2. Click **Create** and enter a unique **Name** and unique, valid **Email Address**.
3. **Add Roles** and **Groups** respectively and click **Save**.
   You can add or remove Roles and Groups any time after you save the new user account.

With a SaaS deployment, the new user instantly receives an email with a link to create their profile name and password. When finished, they are automatically redirected to their Tenant Home page with the permissions assigned at account creation. See Tenant User Management.

With an on-premises deployment, the new user is created and receives an email informing them that they have been added to the Tenant.

After creating a user, you can modify, delete, or duplicate the user account, or assign the user to a group or role. Use **View user details in Accounts** to view the new user in the Account Management Portal.

If the deleted user owns a custom dashboard, the dashboard and its associated shares and reports cease to function properly. See Dashboard Recovery.

Enforce Strong User Passwords (On-Premises Only)

Beginning with 21.2, AppDynamics requires new SaaS Local Users to comply with strong user password guidelines. Progressive use of this version will cause existing SaaS Local User accounts to require mandatory strong passwords. See Tenant User Management.

As an account administrator, you can require Local Users (those authenticated by AppDynamics) to use strong passwords.

Strong password requirements are not enforced by default, which allows a user to configure passwords of any length or complexity.

To require strong passwords:

1. Navigate to **Administration** > **Authentication Provider**.
2. Select **Password Requirements**.
3. Select **Require Strong Passwords**.

Enabling the requirement does not affect existing passwords. Existing weak passwords will continue to function after you enable strong passwords.

When enabled, passwords must:

1. Contain at least eight characters.
2. Contain both upper and lower case letters.
3. Contain at least one number.
4. Not be the username or user email.

Multi-Tenant On-Premises Accounts

This page describes how to create and manage accounts in a multi-tenant Controller. The tenant mode determines whether the Controller UI offers single or multiple environments. See Controller Deployment.

Switch from Single-Tenant to Multi-Tenant Mode

> ⚠ Switching from single-tenancy to multi-tenancy mode is supported. However, switching from multi-tenancy to single-tenancy is not. Take precautions to ensure multi-tenancy is the correct mode for your environment.

If multi-tenancy is enabled for an on-premises Controller, users must enter the account name in the **Account** field when logging in to the Controller UI.

1. Navigate to the Administration Console.
2. Locate the `multitenant.controller` setting.
3. Set the value to `true`.

Create Accounts in Multi-Tenant Mode

In multi-tenant mode, you can add accounts as follows:

1. Log in to the AppDynamics Administration Console as the AppDynamics root user.
2. Click **Account Settings** and then **Add**.
3. Define the licensing entitlements that apply to the account.
   Account-level license unit limits let you prevent a particular account from using more licensing units than it should. You can view the total license units available through **Settings** > **Admin** > **License**. See License Management.

   > ⓘ The overall license limits applicable at the Controller level are independent of any specific limits you apply at the account level.

   Agent-based Licensing: For example, if an account is set up with a Java Agent limit of 100, you can ensure that the new account never interferes with the license availability of another account by setting the **Java Units Provisioned** value for the account to a much smaller limit. However, if you set it to 100 and other accounts are also set to that amount, the first 100 agents that connect to the Controller would occupy those units, regardless of the accounts they report in to. Similarly, you can limit the lifespan of the account by setting an expiration date for the license.

   Infrastructure-based Licensing: For example, if an account is set up with an Infrastructure Monitoring limit of 100, you can ensure that the new account never interferes with the license availability of another account by setting the **Infrastructure Monitoring** value for the account to a much smaller limit. However, if you set it to 100 and other accounts are also set to that amount, the first servers with CPU cores totalling up to 100 would occupy those units, regardless of the accounts they report in to. Similarly, you can limit the lifespan of the account by setting an expiration date for the license.
4. When finished defining entitlements, click **Save**.

After enabling multi-tenant mode, users must specify the account they want to log into in the **Account** field in the Controller UI login screen. See:

- Java Agent Configuration Properties
- .NET Agent Configuration Properties
- Database Agent Configuration Properties
- Machine Agent Configuration Properties

Create and Manage Groups

This page explains how to create and manage user groups in AppDynamics.

A user can belong to one or more groups. Groups let you assign and manage roles for users collectively.

> If you use LDAP to authenticate all AppDynamics Tenant users, you do not need to create AppDynamics groups.

To create a new group:

1. Navigate to **Settings** > **Administration** > **Groups**.
2. Click **Create** and enter a unique **Name** and optional **Description**.
3. Add **Users** and **Roles** respectively and click **Save.**

After creating the group, you can assign users and roles by selecting the group and selecting the **Member** checkboxes for each user or role you want to associate.

Alternatively, navigate to **Roles** > (Select a Role) > **Users and Groups with this Role**.

Create and Manage Custom Roles

This page provides information and instructions for managing roles in AppDynamics.

A role is a collection of permissions that define what actions a user can perform within the AppDynamics-managed environment. You can use groups to manage roles collectively. A user or group can have more than one role but must have at least one defined role.

Administrators can grant user permissions at a granular level, including which business applications they can monitor, what parts of the UI are visible, and the types of configuration changes they can make.

Predefined Roles

The Tenant UI includes predefined roles for administrators and read-only users that you cannot change, however, you can create new custom roles.

Roles	Permissions
Account Owner	• Access to Administration, Agent, and Getting Started Wizard UIs. • Add or edit users, groups, roles, and the authentication provider. • Possesses most of the account-level permissions and is sometimes known as the account administrator. See Account Permissions. • Create War Rooms. • Configure alert, email, and HTTP request templates, emails, reports, Universal Agents, and licenses. • View licenses, business flows, reports, and Universal Agent.
Administrator	• View and modify components that change state, such as applications, business transactions, dashboards, and so on. • Create War Rooms, view business flows, and view and configure scheduled reports. • Cannot add or edit users, groups, or roles.
Analytics Administrator	• View and grant access to Analytics features, such as creating API keys, creating metrics, creating extracted fields, and granting access for viewing analytics data. • Controls which roles have access to specific applications or log source types. • The only role in charge of saved searches. Saved searches provide different data access levels to analytics users. See Analytics and Data Security and Transaction Analytics Permissions.
Applications and Dashboards Viewer	• View all applications and their dashboards. • Cannot edit dashboards (formerly known as the Read-Only User).
Dashboards Viewer	• View application dashboards. • Create custom dashboards.
DB Monitoring User	• Manage Events. • View the Database Monitoring UI. • Cannot add, edit, or delete database collectors.
DB Monitoring Administrator	• View the Database Monitoring UI. • Add, edit or delete database collectors.
Server Monitoring Administrator	• View the Server Monitoring UI. • Configure Service Monitoring features including Service Availability Monitoring.
Server Monitoring User	• View Server Monitoring UI. • Cannot configure Server Monitoring features.

Universal Agent Administrator	• Configure and view the Universal Agent. • Manage Events. • View the Universal Agent.
Universal Agent User	View the Universal Agent.
Workflow Executor	Execute Workflows.

Create Custom Roles

You must have the Account Owner role or the Administration, Agents, Getting Started Wizard permission to create new custom roles.

Roles can be managed collectively by using groups. A user or group must have at least one role but can have multiple. You have the option to add users and groups to predefined roles.

As a starting point for creating your own customized roles, you can copy predefined roles. If you are working under a custom role and copy a predefined role, some permissions may not be visible to you. You must adjust permissions for your customized role to ensure that you get the correct RBAC result.

To create a custom role:

1. Navigate to **Settings** > **Administration** > **Roles**.
2. Click **Create** and enter a unique **Name** and optional **Description**.
3. Configure permissions through the UI.
    a. **Account**—set role permissions
    b. **Applications**—set access and function permissions for applications and tiers
        i. **End User Monitoring**—enable end-user monitoring
        ii. **Server Visibility**—enable users to view and/or configure server components
    c. **Databases**—set access and function permissions for databases
    d. **Analytics**—set general, log, search, and event data view permissions
    e. **Dashboards**—set default and custom dashboard management permissions
    f. **User and Groups with this Role**—view existing or add new users and groups to the role
4. Click **Save**.

Account Permissions

Related pages:

- Create and Manage Custom Roles
- Create and Manage Groups
- Tenant User Management

This page provides the account-level permissions available in AppDynamics. You can set account permissions for custom roles from the Account tab in the Controller Administration UI. Most installations have one account per Controller. Usually, only very large installations or installations that have very distinct sets of users may require multiple accounts.

You can have multiple accounts when you select multi-tenant mode on your Controller. See Controller Deployment.

- In multi-tenant mode, the Controller UI context is divided into separate accounts. Each account has its own set of users, agents reporting to it, and application monitoring configuration.
- The account predefined role is the Account Owner. This role is also known as the account administrator and has all account-level permissions except Execute Workflows. This role can also create applications and dashboards and has all database visibility-related permissions.
- Most Account-level permissions can be considered administrative permissions and apply account-wide across business applications, products, and multiple application instances for the same account.

The following table lists the permissions that can be set at the account level.

Permission	Activities enabled	More Information
Administration, Agents, Getting Started Wizard	• Administer users, groups, roles, authentication, and so on. <table><tr><th>Users</th><th>Groups</th><th>Roles (non-default)</th><th>Authentication Provider</th></tr><tr><td>Add, edit, disable</td><td>Add, edit, delete</td><td>Add, edit, delete</td><td>Add, edit</td></tr></table>• View AppDynamics Agents. • Download agents using the Agent Download Wizard.	Create and Manage Custom Roles
Configure Alerting Templates	Create, edit, and delete alerting templates.	Configure and Manage Alerting Templates
Configure Cisco Secure Application	Configure and monitor the security of the APM-managed application using Cisco Secure Application. This grants permission to configure security settings, create or modify policies, add or edit notes, set status, and set severity.	Monitor Application Security Using Cisco Secure Application
Configure Email / SMS	Edit email and SMS settings used by AppDynamics to send alerts.	Enable an Email Server
Configure Email Templates	Configure email templates for use in notification alerts.	Email Templates
Configure HTTP Request Templates	Configure the HTTP request templates used by HTTP request actions, which are triggered by AppDynamics policies.	HTTP Request Actions and Templates
Configure Scheduled Reports	Create, delete, send, or update scheduled reports.	Reports
Configure Universal Agent	Use Universal Agent REST APIs that add or change Universal Agent configuration.	Universal Agent REST APIs
Create War Rooms	Create (start) a war room. War rooms are collaborative custom dashboards created in real-time.	Virtual War Rooms
View and Configure Licenses	View and create rules to specify the number of Application Performance Monitoring licenses to allocate to specified applications and machines.	License Management
View Business Flow	View all applications in a multi-business-application flow map, including those for which they are not granted explicit application permissions. Does not grant permission to drill down to applications that the user does not have permission to view. To see the downstream metrics and snapshots for the correlated application, the user must be a member of a role with view permissions to that business application.	Cross Application Flow
View Cisco Secure Application	View and monitor the security of the APM-managed application using the Cisco Secure Application dashboard.	Monitor Application Security Using Cisco Secure Application
View Scheduled Reports	View scheduled reports from the Dashboards and Reports section of the UI.	Reports
View Synthetic Credential Vault	Manage user permissions for the Synthetic Credential Vault.	Synthetic Credential Vault
Install Agent	Use the Agent Installer to deploy the Agent Installer Platform, Java Agent, and Machine Agent.	Agent Installer

Application Permissions

This page provides an overview of application permissions in AppDynamics. Application permissions follow an inheritance model with three levels listed in order from highest (default) to lowest (tier-specific):

- Default permissions
- Application-wide permissions
- Tier-specific permissions

By default, each level inherits from the one above it, unless you customize permissions at a lower level. This mechanism enables you to grant access to groups or users for specific business applications in the Tenant UI.

Customized permissions at a specific level override more general permissions at another level. That is, tier-specific permissions take precedence over application-specific permissions, and application-specific permissions override default permissions. Not all permissions can be customized at the tier-level.

You can set application permissions for custom roles from the Applications tab in the Tenant Administration UI. You can also assign the **Can Create Applications** permission to a custom role. See Create and Manage Custom Roles.

Create Default Permissions

All new applications inherit default permissions.

Configure Default Application Permissions

1. Log in to the Tenant Administration UI and select **Roles**.
2. Add a new role or select a custom role.
3. Click **Can Create Applications** to grant the role permission.
4. Under **Default Permissions**, select the default permissions for this role: **View**, **Edit**, or **Delete**.
    a. To give all permissions to all applications, click **Edit**.
    b. To specify permissions for specific application configurations for all applications, deselect **Edit**, and then click **Edit (None)**.
    c. In the **Edit Permissions** panel, select specific permissions.
    d. Click **Delete** to grant permissions to delete any application. To grant permission to delete a specific application, customize the permission at the application level. See Application and Tier Level Permissions.
5. Click **OK** then click **Save.**

Customize Application Permissions

To customize business application-level permissions:

1. Set the **Permissions** dropdown to **Custom**.
2. Select **View** and then **Edit (None)**.
   You can also grant permission to delete a specific application.

To customize permissions at the tier level,

1. Click **Add** to add tiers or select an existing tier.
2. Select **Edit**.
3. Select the individual permissions for the specific tier.
4. Click **OK** then click **Save**.

Overlapping Role Permissions Examples

Within specific and default permissions, granting a specific permission takes precedence over denying the same permission elsewhere. For example, if a user is assigned two roles and one grants a permission and the second role denies it, the user will have permissions for the activity.

These examples show how overlapping permissions of different roles interact. The examples enable view, edit, and delete permissions to applications for two Groups. The last column shows the resulting permissions for a specific user with roles that are assigned to each group.

Group 1			Group 2	
	Default Permissions (view, edit delete all applications)	**Explicit permissions (view, edit delete application-1)**	**Default Permissions (view, edit delete all applications)**	**Explicit permissions (view, edit delete application-1)**
A	None	Yes	Yes	None
B	Yes	None	Yes	Yes
C	Yes	None	None	Non

- Result for example A: User has view, edit, and delete permissions to all applications, including application-1.
- Result for example B: User has view, edit, and delete permissions to all applications, including application-1.
- Result for example C: User has view, edit, and delete permissions to all applications, excluding application-1.

General Permissions

Permission	Activities Enabled	More Information
Can Create Applications	Create business, browser, and mobile applications. Also controls the Archive Snapshot action.	Business Applications
View, Edit, and Delete permissions for new applications can be set as part of the default permissions for a custom role	View, edit, or delete business applications (and the tiers and nodes), browser, and mobile applications. Setting default delete permissions allows the user to delete all three artifacts from the application model.	Business Applications Tiers and Nodes

Application and Tier Permissions

You can grant the following permissions as specified. Permissions that you can customize at the tier level are indicated in the Description of Activities Enabled column. Asterisks (*) in the permissions table indicate permissions that are considered sensitive for security and data privacy purposes. Carefully consider the security and data privacy policies of your organization before granting these permissions.

Permission	Description of Activities Enabled	More Information
Configure Transaction Detection*	Create, edit, or delete transaction detection - can be at the tier level.	Transaction Detection Rules
Configure Backend Detection	Create, edit, or delete backends - can be at the tier level.	Backend Detection Rules
Configure Error Detection	Create, edit, or delete error detection.	Error Detection
Configure Diagnostic Data Collectors*	Create, edit, or delete diagnostic data collectors.	Data Collectors
Configure Call Graph Settings	• Edit call graph settings (no SQL) • Turn on or off capture raw SQL (call graph and SQL bind must both be on)	Call Graph Settings
Configure JMX	Create, edit, or delete JMX metrics.	Configure JMX Metrics from MBeans
Configure Memory Monitoring	Configure which custom classes are tracked by Object Instance Tracking. To enable or disable Object Instance Tracking, you need the Configure Agent Properties permission.	Object Instance Tracking for Java
Configure EUM (for Browser RUM)	See End User Monitoring Permissions.	Configure the Controller UI for Browser RUM
Configure EUM (for Mobile RUM)	See End User Monitoring Permissions.	Configure the Controller UI for Mobile RUM
Configure Information Points*	Create, edit, or delete information points.	Information Points
Configure Health Rules	Create, edit, or delete health rules.	Configure Health Rules
Configure Actions	Create, edit, or delete actions on agent properties UI. Create, edit, or delete email digests.	Alert and Respond Actions Email Digests
Configure Policies	Create, edit, or delete policies.	Configure Policies

Configure Business Transactions	Organize Business Transactions including: ▪ Group Business Transactions ▪ Exclude/un-exclude Business Transactions ▪ Delete Business Transactions ▪ Enable Business Transaction lockdown ▪ Rename Business Transactions Configure Business Transaction thresholds. Configure snapshot settings. Set as a background task. Configure data collectors. Enable End User Monitoring. Enable analytics for business transactions. Enable or disable GUID injection.	Organize Business Transactions Transaction Thresholds Transaction Snapshots Monitor Background Tasks Data Collectors Set Up and Access Browser RUM Collect Transaction Analytics Data Business Transaction and Log Correlation
Configure Baselines	Create, edit, or delete baselines.	Dynamic Baselines
Configure SQL Bind Variables*	Turn on or off capture raw SQL (also requires Configure Call Graph Settings).	Call Graph Settings
Configure Agent Properties	Create, edit, or delete agent configuration (can be at the tier level). Enable or disable automatic leak detection (can be at the tier level). Enable or disable object instance tracking (can be at the tier level). Enable or disable custom memory structure (can be at the tier level).	App Agent Node Properties Object Instance Tracking for Java Custom Memory Structures for Java
Agent Advanced Operation	Reset agent from the node dashboard. Request agent thread dumps. Request agent debug logs.	Manage App Agents Diagnostic Actions Request Agent Log Files
Set JMX MBean Attributes and Invoke Operations	Edit MBean attributes or invoke actions on operations.	Monitor JMX
Configure Service Endpoints	Create, edit, or delete service endpoints.	Service Endpoint Detection
Configure Monitoring Level (Production /Deployment)	Switch between production and development mode.	Development Level Monitoring
Configure 'My Dashboards' for Tiers and Nodes	Create, edit or delete custom dashboards (can be at the tier level).	Create and Manage Custom Dashboards and Templates Custom Dashboards
Create Events	Create, edit, or delete events.	Alert and Respond API
Start Diagnostic Sessions	Start a diagnostic session.	Diagnostic Sessions
View Sensitive Data*	In combination with the Configure Transaction Detection permission, enables the use of Live Preview and Business Transaction Discovery features to stream live data from your application.	Custom Match Rule Live Preview

Transaction Analytics Permissions

Related Pages:

- Create and Manage Custom Roles
- Analytics and Data Security

This page provides an overview of permissions in Analytics.

The predefined role for Analytics is the Analytics Administrator. You can set analytics permissions for custom roles from the Analytics tab in the Tenant UI. When creating a new role, select the permissions that you want to assign to the new role.

There are several analytics permissions categories:

- Transactions
- Logs
- Browser Requests
- Mobile Requests and Sessions
- Synthetic Sessions
- Connected Devices
- Custom Analytics Events

For a user to create metrics from Analytics searches, you must assign that user to a role with View access to the Analytics Application in the Applications tab.

Asterisks (*) in the permissions tables on this page indicate permissions that should be considered sensitive for security and data privacy purposes. Consider the security and data privacy policies of your organization before granting these permissions. For details on permissions related to Analytics data and functionality, see the More Information column.

Permission Name	Activities Enabled	More Information
General > Manage Centralized Log Config	Configure source rules for Centralized Log Management.	Configure Log Analytics Using Source Rules
General > Manage Fields*	Show and hide fields in analytics data so that you can restrict sensitive data to proper roles and users.	Manage Field Visibility
General > Manage APIs*	View the analytics API tab to create and manage API authentication keys. Users with this API permission have access to all analytics data since they can use cURL with the access key and access all data.	Manage API Keys
General > Manage Metrics	Create metrics from analytics searches. This permission controls which roles and users can create metrics from analytics searches. Once the metric exists, you can set up alerts in the usual way.	Create Analytics Metrics From Scheduled Queries
Search Permissions > Can Create a Search*	Create and save an analytics search. Configure view, edit, and delete permissions for each saved search.	Analytics and Data Security
Search Permissions > Saved Searches*	View, edit, or delete a specific saved analytics search.	Analytics and Data Security
Transaction Permissions*	View analytics transaction data from all applications or specific applications.	Analytics and Data Security
Log Permissions*	View analytics data for all Source Types or specific log sources.	Analytics and Data Security
Browser Requests and Sessions Permissions*	View analytics data from Browser requests.	Analytics and Data Security
Mobile Requests and Sessions Permissions*	View analytics data from mobile requests and crash reports.	Analytics and Data Security
Synthetic Sessions Permissions	View synthetic data from all or specific applications.	Analytics and Data Security
Connected Devices Permissions	View analytics data for connected devices.	IoT Monitoring
Custom Analytics Events Permissions*	Query custom analytics events data. Permissions can be granted to view data for all custom analytics events or on an event-by-event basis.	Analytics and Data Security

Custom Dashboard Permissions

Related pages:

- Create and Manage Custom Roles
- Custom Dashboards

This page provides an overview of permissions for custom dashboards in AppDynamics.

The predefined role is the Dashboards Viewer. The permissions of this role are limited to viewing custom dashboards in the Tenant UI. The permissions of this role apply if you do not set more specific permissions for an individual custom dashboard.

Each custom dashboard inherits the default custom dashboard permissions unless you override the defaults by configuring separate explicit permissions for individual dashboards.

For example, you could have a custom dashboard called SalesDashboard and a custom role SalesRole, and a second custom dashboard called FinanceDashboard, and a second custom role FinanceRole. The SalesRole could be configured to have permissions in the SalesDashboard but not in the FinanceDashboard and so on.

> Creating custom dashboard templates requires the **Configure 'My Dashboards' for Tiers and Nodes** permission, which you can set at the application level. See Application Permissions.

Default Permissions

Changes made to the default or custom permissions are automatically applied to existing and future dashboards. These permissions can be used to create special permissions for specific dashboards and override default permissions.

You can configure default permissions on the `Admin.jsp` page as follows:

1. Click **Settings** and select **Administration.**
2. Click **Roles**.
    a. Select **General** and click **Create** Unknown Attachment.
    b. Provide details (name and description) about the role.
    c. Click **Dashboards** and select the permissions that you want this role to acquire.
    d. Select **Can Create Dashboards** to grant role permission to create new dashboards.
    e. Select **Default Permissions** for this role: **View**, **Edit**, **Delete**, or **Share**.
    f. Click **Save**.

    > You cannot edit the permissions of the listed dashboard roles because they are set by default.
    >
    > If any of the dashboard permissions in the custom permissions list matches with the default permissions, that entry will be removed from the custom list.

3. Click **Users**.
    a. Create a new user or select an existing user.
    b. Select one or more options to assign the required role to the user.
    c. You can assign all the available roles to the user at once or disable them all at once.
4. Click **Save**.

Dashboard Permissions

Permission	Activities Enabled	More Information
View	View specific custom dashboards.	Custom Dashboards
Edit	Edit specific custom dashboards.	
Delete	Delete specific custom dashboards.	
Share	Share specific custom dashboards.	
Can Create Custom Dashboards	Create new custom dashboards.	

Share Permission

You can allow the users to share their dashboards publicly (without requiring a login) with the share permission. This permission allows you to control the permissions that a user or a role can acquire.

Enable the Share Permission

You can enable the **Share** icon on the dashboard after configuring the default share permissions:

1. Navigate to **Dashboards & Reports** and click **Dashboards**.
2. Right-click a dashboard.
3. Select **Share** > **Share Dashboard**.
   This table shows you how the **Share** icon appears on the dashboard menu based on the values assigned to share permission and the shared status:

Share Permission	Is shared	Share Icon
True	Yes	Active
True	No	Active
False	Yes	Inactive
False	No	Inactive

When the share permission is activated yet no role with share access exists, dashboard users cannot share the dashboard. You must update all default roles with `share=true` when appropriate.

Database Permissions

This page provides an overview of permissions for databases in AppDynamics.

You can set Database permissions for custom roles from the **Databases** page in the Controller Administration UI. See Create and Manage Custom Roles.

Navigate to **Administration** > **Roles** > **Applications** > **Databases** and ensure that the **Custom view** permission is set to active to view metrics associated with database collectors. You can also configure database collector permissions for custom roles.

For each custom role, you can select which databases the user is allowed to view, edit, or delete, and you can enable global permissions applying to database collectors.

Permission	Activities Enabled	More Information
Can Configure Collectors	Configure baselines, wait state filtering, and custom metrics, as well as enable or disable the masking of query literals.	Discover Normal Database and Server Activity Wait State Filtering Configure Custom Metrics Configuring Query Literals Security
Can Create Collectors	Create database collectors for any database.	Add Database Collectors
View	View all existing and new database collectors, and therefore the metrics for the databases associated with those collectors.	Database Dashboard
Edit	Edit database collector fields for any database.	Add Database Collectors
Delete	Delete database collectors for any database.	Add Database Collectors

End User Monitoring Permissions

Related pages:

- Configure the Controller UI for Browser RUM
- Configure the Controller UI for Mobile RUM

This page provides an overview of permissions for End User Monitoring.

You can set End User Monitoring permissions for custom roles from the Applications tab in the Controller Administration UI. You need the Configure EUM permission to configure Browser and Mobile RUM. You can add this permission as the default permission for all applications, or you can specify the permission at the application level.

Browser RUM

The Configure EUM permission permits the following actions for Browser RUM:

- Configure and download the JavaScript Agent.
- Add include/exclude rules for base pages, Ajax requests, and virtual pages.
- Enable JavaScript error capture and add rules to ignore specific JavaScript errors.
- Set thresholds for slow end-user experience.
- Enable the collection of slow, periodic, and error snapshots.

See Configure the Controller UI for Browser RUM for instructions and a list of the supported Browser RUM configurations.

Browser Synthetic Monitoring

The Configure EUM permission permits these actions for Browser Synthetic Monitoring: view, schedule, edit, and delete synthetic jobs.

Mobile RUM

The Configure EUM permission permits these actions for Mobile RUM:

- Upload dSYM files for iOS or ProGuard files for Android.
- Add include/exclude rules for network requests and the Events Service.
- Set thresholds for slow end-user experience.
- Configure mobile crash alerts.
- Configure mobile screenshots.

See Configure the Controller UI for Mobile RUM for instructions and a list of the supported Mobile RUM configurations.

Server Visibility Permissions

Related pages:

- Create and Manage Custom Roles
- Server Visibility

This page provides an overview of permissions for server visibility in AppDynamics.

When you create custom roles, you can assign the server visibility permissions to users as part of the default permissions for all applications or create customized applications permissions for new roles.

1. From the Controller Administration UI, navigate to **Roles > Applications**.
2. Select an existing role or create a new role.
3. From the Server Monitoring dropdown, select **Customized**.
4. Click **View** and **Edit** to see the available permissions.

Predefined Roles

- Server Visibility Administrator
- Server Visibility User

Permissions

Permission Name	Activities Enabled	More Information
Configure Server Visibility	View all Servers tabs and windows, and configure Service Availability Monitoring.	Service Availability Monitoring
View Server Visibility	View all Servers tabs and windows.	Server Visibility

External Authentication Providers

This page provides an overview of external authentication providers in AppDynamics.

User Authentication Types

AppDynamics manages user credentials according to your type of deployment. See Deployment Options.

- Local User—AppDynamics manages user account credentials (SaaS)
- SAML User—You manage user account credentials via SAML integration (SaaS and on-premises)
- LDAP User—You manage user account credentials via LDAP integration (on-premises)

You create and manage Local User accounts in the Tenant Administration UI. However, you can authenticate and authorize Tenant users with external LDAP or SAML-based authentication providers.

Authentication Options

Log in to the Tenant UI as an administrator and navigate to **Settings** > **Administration** > **Authentication Provider**.

By default, Tenant authentication is configured to the local authentication option, **AppDynamics**. With this configuration, AppDynamics manages the users and they authenticate through the AppDynamics Identity Provider (IDP)

Alternatively, you can have an external LDAP or SAML system perform user authentication and authorization.

- LDAP Authentication
- SAML Authentication

LDAP Authentication

This page provides instructions on how to integrate an AppDynamics Tenant with LDAP directory servers.

This page assumes a Controller and a Tenant are one and the same.

LDAP Support

You can delegate Tenant UI authentication and authorization to external directory servers that comply with LDAP (Lightweight Directory Access Protocol) version 3.

While a Tenant should be able to work with any LDAPv3-compliant server, these LDAP products have been verified:

- Microsoft Active Directory for Windows Server 2008 SP2 or later
- OpenLDAP 2.4 or later

To configure LDAP authentication on a Tenant, you must configure connection settings to the LDAP server and the queries that return user or group data. By mapping LDAP groups to roles, you can provision permissions in the AppDynamics Tenant based on LDAP groups.

- What happens if the LDAP server becomes unavailable?
  If the LDAP server configured for Tenant authentication becomes unavailable for any reason, the Tenant falls back to local user authentication. Given this possibility, you should provision local user accounts in AppDynamics for the administrative users who will need access if the LDAP server becomes unavailable.
- What happens if a user cannot be found in the LDAP directory?
  If a user cannot be found in the LDAP directory, the authentication failure event is logged as a warning. The user, whether a regular Tenant user or a REST client user, may still be authenticated through local authentication.

Prepare the LDAP Directory for AppDynamics Integration

To use an LDAP authentication provider, your AppDynamics Tenant must be able to connect to the external LDAP server. We recommend creating a user account in LDAP specifically for the Tenant to use to authenticate itself to the server and run the queries. The Tenant user only needs to have search privileges in LDAP.

You can map existing LDAP group definitions to roles in AppDynamics, however, your existing groups may not correspond directly to those roles. You can map LDAP groups to Tenant roles by creating a group in LDAP for each role you want to map in AppDynamics. LDAP groups for each role provide you with a manageable, one-to-one correspondence between your LDAP groups and AppDynamics roles.

This is a possible LDAP group scheme for mapping in AppDynamics:

- AppDynamics-AppA-ReadOnly
- AppDynamics-AppA-Admins
- AppDynamics-AppA-DashboardViewers
- AppDynamics-AppB-ReadOnly
- AppDynamics-AppB-Admins
- AppDynamics-AppB-DashboardViewers

The sample group names imply having custom roles in AppDynamics targeted to specific applications, AppA and AppB.

Naming the groups with a common prefix, as the `AppDynamics-` prefix in our sample, allows you to use an LDAP group filter. A group filter for the sample groups could be:

```
(&(objectClass=group)(cn=AppDynamics-*))
```

Use Paged Results for Large Result Sets

LDAP servers are sometimes configured to limit the number of entries they can return in a query response. If the results of your user or group query exceed that limit, AppDynamics reports a `max_results_exceeded` error.

To avoid this error, refine your query filter to produce a smaller result set. The results must include the users who will need to access the AppDynamics UI.

If your LDAP server supports it, you can enable paged results in the Tenant LDAP configuration. With paged results, the LDAP server divides the result set into separately transmitted blocks.

The paged results feature applies to the behind-the-scenes interaction between the AppDynamics Tenant and the backend LDAP server. It does not affect the UI view of the data.

LDAP Authentication with a SaaS AppDynamics Controller

Depending on your organization's security policies, it may not be possible to use LDAP authentication with the SaaS AppDynamics Tenant. Doing so requires opening your firewall to permit Tenant access to your corporate LDAP server.

To enable LDAP authentication with an AppDynamics SaaS Tenant, however, you must permit access through the firewall for the AppDynamics IP ranges listed in SaaS Domains and IP Ranges. The firewall rule should allow incoming LDAP requests from the Tenant at the LDAP port you configure.

Before You Start

To perform LDAP configuration, you must have:

- An LDAP server. There is a one-to-one correspondence between an AppDynamics account and an LDAP server.
- An account on an AppDynamics SaaS or on-premises Tenant
- Account administrator privileges on the AppDynamics Tenant, as described in Update the Root User and Glassfish Admin Passwords.
- Network connectivity between your LDAP server and the Controller. If using a SaaS Tenant, the LDAP server may not be accessible to the Tenant without enabling access through your network firewall. See LDAP Authentication with a SaaS AppDynamics Controller.

Configure LDAP Authentication

The high-level procedure to set up LDAP authentication includes:

- Configure the connection to the LDAP server.
- Configure and test the LDAP query that returns users to be provisioned in the AppDynamics Tenant.
- Configure the LDAP query that returns the LDAP groups to be mapped to AppDynamics roles.
- Map the users or groups to roles in AppDynamics.

Configure the Connection to the LDAP Server

As an Administrator or Account Owner in the Tenant UI, you can configure LDAP authentication by navigating to **Settings > Administration** and selecting **LDAP** in **Authentication Provider**.

Use the following paged results configuration if the user or group query you need to use returns more entries than the LDAP server permits:

- **Enable Paging**: Check this option to have the Controller request paged results from the server when submitting user or group queries.
- **Page Size**: Enter the number of entries per round-trip from the AppDynamics Controller to the LDAP server. The default is 500.

The page size should equal the total number of entries to be returned divided by the tolerable number of round trips between the LDAP server and the Tenant. For example, if you expect to receive 1200 results in a query and you can tolerate a maximum of two round trips, set the page size to 600 (1200 /2). See Using Paged Results for Large Result Sets.

Configure the LDAP connection settings:

- **Host:** Required address of the LDAP server.
- **Port:** Port on which the LDAP server listens. The default is 636 for an SSL connection and 389 if not using SSL. Required.
- **Use SSL:** Enabled by default to use a secure connection to the LDAP server. Clear if not using SSL.
- **Enable Referrals:** Enabled by default to support LDAP referrals. A referral is when an LDAP server forwards an LDAP client request to another LDAP server. Each referral event is referred to as a hop.
- **Maximum Referral Hops:** The maximum number of referrals that AppDynamics follows in a sequence of referrals. The default is five.
- **Bind DN:** Distinguished Name of the user on the LDAP Server on whose behalf the AppDynamics application searches. Required.
- **Password:** Password of the user on the LDAP server. Required.

Configure Users

In the **LDAP Configuration** page, configure information to find LDAP users:

- **Base DN:** Location in the LDAP tree to begin recursively searching for users. Required.
- **Filter:** Optional LDAP search string that filters the items matched from the base DN. See RFC2254 for information about LDAP search filters.
- **Login Attribute:** The LDAP field that corresponds to the username users will enter when logging in to the AppDynamics UI. The default is `uid`. For Active Directory, this would typically be `sAMAccountName`.
- **Display Name Attribute:** The LDAP field to use as the user's display name.
- **Group Membership Attribute:** Optional user group membership field. Recommended for faster retrieval.

- **Email Attribute:** Optional user email address.

Select **Test Query** to check the connection. If successful, a screen displays the first few users returned by the query. The test does not return the entire result set if the result set is large.

Configure Groups

Optionally, you can map LDAP groups to user roles in the AppDynamics Tenant. To do this, you must set up the LDAP query that returns the LDAP groups to map:

- **Base DN**: Location in the LDAP tree to begin recursively searching for groups. Required.
- **Enable Nested Groups:** Option to include nested LDAP groups to a depth of 10.
- **Filter:** Optional LDAP search string that filters the items matched from the base DN. See RFC2254 for information about LDAP search filters.
- **Name Attribute:** The LDAP field that contains the name of the group. Default is `cn`. Required.
- **Description Attribute:** The LDAP field that contains a description of the group. Optional.
- **User Membership Attribute:** Identifies members of the groups. Optional.
- **Referenced User Attribute:** Optional child attribute of the User Membership Attribute. Disabled if the parent is empty. Identifies the property of the user that the user membership attribute contains.

Select **Test Query** to check the connection. If successful, the first few groups returned by the query are shown.

You can now assign permissions in the AppDynamics Tenant to users or groups.

Assign AppDynamics Permissions to an LDAP User

1. In **Settings > Administration**, click **Users**. If LDAP is enabled and correctly configured, the AppDynamics Tenant fetches the user names from the LDAP server.
2. Select the name of the user to whom you want to assign permissions.
3. Add or delete the **Roles** that you want to assign to this user. You can assign multiple roles to a user.
4. Click **Save**.

Assign AppDynamics Permissions to an LDAP Group

LDAP Group configuration is optional.

1. In **Settings > Administration**, click **Groups**. If LDAP is enabled and correctly configured, AppDynamics fetches the group names in LDAP.
2. Select the name of the group to which you want to assign permissions.
3. Add or delete the **Roles** that you want to assign to this group. You can assign multiple roles to a group.
4. Click **Save**.

Configure the LDAP Cache Synchronization Frequency

The Tenant keeps information about LDAP users and groups in a local cache. It regularly connects to the LDAP server to synchronize its cache with the LDAP server.

The Tenant caches information about users and group membership. It does not cache user passwords. Accordingly, the Tenant authenticates the user credentials against the LDAP server at the start of every user session.

If you remove a user account from LDAP, the change reflects immediately and the user will not be able to log in to the Tenant UI. However, if the user has an existing session in the Tenant UI, that session continues until the user logs out or the session expires.

If the user's access to the Tenant is based on group membership and you remove the user from the group but maintain an account in the LDAP server, the user is able to log in to the Tenant until the next LDAP server synchronization. The default synchronization frequency setting would allow the ability to access the Tenant UI for up to an hour.

Configure the LDAP Synchronization Frequency

To modify the default synchronization frequency of one hour:

1. Stop the Tenant application server.

- From Linux, run:

```
platform-admin.sh stop-controller-appserver
```

- From Windows, run this command from an elevated command prompt (which you can open by right-clicking the **Command Prompt** in the Windows **Start** menu, and selecting **Run as administrator**):

```
platform-admin.exe cli stop-controller-appserver
```

2. Open the `<Controller-Installation-Directory>/appserver/glassfish/domains/domain1/config/domain.xml` file for editing.
3. In the `<jvm-options>` element, add a system property named `appdynamics.ldap.sync.frequency` with the desired synchronization frequency in milliseconds.
   For example, to have the Tenant synchronize to the LDAP server every 15 minutes (900000 milliseconds), add:

```
<jvm-options>-Dappdynamics.ldap.sync.frequency=900000</jvm-options>
```

   The default is 3600000 milliseconds (1 hour).
4. Save the file.
5. Restart the Tenant app server:
   - From Linux, run:

```
platform-admin.sh start-controller-appserver
```

   - From Windows, run this command from an elevated command prompt:

```
platform-admin.exe cli start-controller-appserver
```

SAML Authentication

This page provides an overview of Security Assertion Markup Language (SAML) authentication in AppDynamics.

This page assumes a Controller and a Tenant are one and the same.

The AppDynamics Tenant can use an external SAML identity provider (IDP) to authenticate and authorize users. See Configure Basic SAML Authentication Configuration.

Supported Identity Providers

AppDynamics certifies support for the following identity providers (IDPs):

- Okta
- Onelogin
- Ping Identity
- Azure AD
- IBM Cloud Identity
- Active Directory Federation Service (AD FS)

Other IDPs supporting HTTP POST binding should also be compatible with AppDynamics SAML authentication. If you are having issues setting up SAML with your IDP, contact your AppDynamics account representative for help.

Binding Support

AppDynamics supports identity federation with SAML 2.0, an open standard used by many IDPs. This identity federation enables single sign-on (SSO) with HTTP POST binding for the SAML request and HTTP POST binding for the IDP response.

The bindings have the following requirements:

- HTTP is the required transport. Optionally, you can also configure HTTPS transport.
- The AppDynamics Tenant uses HTTP GET or POST for the authentication request to the identity provider for the sign-out message to the identity provider. The IDP also uses HTTP GET and POST to return the response.

How SAML Authentication Works with AppDynamics

With SAML authentication enabled:

1. Navigate to your Tenant login page and enter your account name.
   The Tenant redirects you to the external SAML IDP.
2. From the IDP, enter your credentials.
3. The IDP redirects and logs you into the Tenant.

To log in to the Tenant UI, users require access to both the Tenant and the identity provider service through the network from their computer. You can configure the Tenant to assign roles to authenticated users based on group attributes in their SAML responses. See Map SAML-Authenticated Users to AppDynamics Roles.

Who Can Configure SAML

Only users assigned to the AppDynamics role Account Owner can configure SAML authentication in the Tenant and assign other users to the Account Owner role. Roles govern user privileges in the Tenant UI. See Create and Manage Custom Roles.

Configure Basic SAML Authentication

Related pages:

- Map SAML Group to AppDynamics Roles

This page provides guidelines for configuring basic SAML authentication.

Configure SAML Authentication for the Identity Provider

You can configure an identity provider to enable single sign-on access to the AppDynamics Controller using the SAML 2.0 protocol. Refer to the documentation of your identity provider for detailed configuration instructions.

SAML Settings for the Identity Provider

Your identity provider requires the following information about your AppDynamics Controller for the SAML settings. The `<controller_domain>` can be the domain of one of the AppDynamics SaaS Controllers or your on-premises Controller.

Setting	Description
Audience URI (Service Provider Entity ID)	The unique identifier intended for the SAML assertion. In most cases, it is the Service Provider Entity ID, unless the Service Provider decides to use a different identifier. • Syntax: `http://<controller_domain>/controller` • Example: `http://yourcompany.saas.appdynamics.com/controller`
Single Sign-On URL (Assertion Consumer URL)	The AppDynamics endpoint to service SAML Authentication. You need to specify your AppDynamics account name with the query string parameter `accountName` as shown below. • Syntax: `http://<controller_domain>/controller/saml-auth?accountName=<account_name>` • Example: `http://yourcompany.saas.appdynamics.com/controller/saml-auth?accountName=myaccount`

SAML Attributes for the Identity Provider (Recommended)

You set attributes with your identity provider that map to SAML users in your AppDynamics account. When the attributes are set, the Controller displays the user's information, such as the username and email. Changes to these attributes on the IDP will update the mapped SAML attributes on AppDynamics Controller when the user successfully logs in.

The table shows how IDP example attributes map to the **Username Attribute**, **Display Name Attribute**, and the **Email Attribute** settings of the Controller.

Example Attribute Name	Example Attribute Values	Description
Username Attribute	`User.loginName`	Unique identifier for the user in the SAML response. This value corresponds to the AppDynamics `username` field, so the value must be unique among all SAML users in the AppDynamics account. If no username is mapped, AppDynamics obtains the `username` from the `NameId` containing the `emailaddress` field.
Display Name Attribute	`User.fullName`	Informal name for the user corresponding to the AppDynamics `Name` field.
Email Attribute	`User.email`	User's email address, corresponding to AppDynamics `email` field.

Configure SAML Authentication from the Controller

To configure SAML authentication from the Controller:

Configure SAML Authentication

1. Navigate to your Controller.
2. Log in as the Account Owner. See Who Can Configure SAML.
3. As a user with AppDynamics account administrator privileges in the Controller UI, click **Settings**  **> Administration**.
4. Click the Authentication Provider tab and select **SAML**.
5. From **Authentication Provider** > **SAML**, enter the following SAML configuration settings:

- **Login URL:** The SAML Login URL where the Controller routes Service Provider (SP)-initiated login requests. This login URL is required.
- **Logout URL:** The URL where the Controller redirects users after they log out. If you do not specify a logout URL, users will get the AppDynamics login screen when they log out.
- **Certificate:** The X.509 certificate from your identity provider configuration. Paste the certificate between the `BEGIN CERTIFICATE` and `END CERTIFICATE` delimiters. Avoid duplicating `BEGIN CERTIFICATE` and `END CERTIFICATE` delimiters from the source certificate itself.

Configure SAML Attribute Mapping (Optional)

From **SAML Attribute Mappings**, you can specify how SAML-authenticated users are identified in the AppDynamics Controller with the following:

- **Username Attribute:** Unique identifier for the user in the SAML response. This value corresponds to the AppDynamics `username` field, so the value must be unique among all SAML users in the Controller account. Given the sample response below, the value for this setting would be `User.OpenIDName`.
- **Display Name Attribute:** The informal name for the user corresponding to the AppDynamics Name field. Given the sample response, this value would be `User.fullName`.
- **Email Attribute:** The user's email address, corresponding to AppDynamics email field. Given the sample response, this value would be `User.email`.

Map SAML-Authenticated Users to AppDynamics Roles

From **SAML Group Mappings,** you can map SAML-authenticated users to one of the Controller roles:

- **Default Role**: If a user's identity assertion has no SAML group attribute, the authenticated user is assigned the SAML default role upon the first login. The default role cannot be removed, and you are recommended to provide minimum permissions. An AppDynamics administrator can verify and adjust the roles for users manually in AppDynamics once those users have accounts.
- **SAML Group**: You can map SAML group membership attributes to roles in AppDynamics. Using this method, each time the user authenticates, the Controller checks the SAML assertion and updates the role assignment if needed.
- **Internal Group**: If a SAML-authenticated user has the same username as an AppDynamics internal user account and the SAML assertion does not contain mapped SAML group attributes, the Controller gives the user the roles for the internal AppDynamics account.

Configure Default Permissions

Instead of mapping SAML attributes to roles, you can also assign users to a default role with the permissions you specify:

1. To use default permissions, edit the **Default Permissions** settings in the **SAML Group Mappings** list.
2. In the **Default Group Mapping** dialog, choose the AppDynamics roles that all authenticated users get.

Verify the SAML Authentication Configuration

The best way to verify that you have configured SAML authentication correctly is to log in to your AppDynamics Controller.

This procedure shows the SAML flow from the service provider (your Controller) and describes the SAML requests and responses. You can also start the SAML flow from the IDP.

1. Navigate to your AppDynamics Controller.
2. You will see the **Login** dialog for the 3rd-party service, which is your IDP.
3. Click **Login**.
4. After you are redirected to your IDP, enter and submit your credentials.
5. The IDP redirects you to your AppDynamics Controller.

> ⚠ (On-premises only) If you are redirected, but the Controller fails to load, this could be because the request URL from the IDP is different from your internal Controller URL. The most common causes are your Controller is configured to use HTTPS but the redirect used HTTP, or you changed the default port of the Controller. See the suggested resolution in Support Advisory: SAML Authentication Fails after 4.3 Upgrade.

6. From the Controller, if you set SAML attributes to map to the user account, you can view the user info by clicking **Settings** **> My Preferences**.
7. If you set default permissions, the user is assigned to the default role, which can be viewed by clicking **Settings** **> Administration**.

Map SAML Group to AppDynamics Roles

This page describes how to configure SAML attributes to role mapping and the SAML group attribute value mapping options.

If the identity assertion from the SAML provider includes group names that correspond to AppDynamics roles, you can configure mappings between those group names and the roles. The **SAML Group Mappings** settings in **SAML Configuration > Authentication Provider** control the mappings.

Configure SAML Attribute to Role Mapping

To configure the SAML attribute to role mapping:

1. In the **SAML Group Attribute Name** field, enter the `Name` attribute value that identifies the SAML Attribute element with group affiliations for the user. For example, given the following response snippet, use `SAML groups-Membership` in the **SAML Group Attribute Name** field.

   ```
   <saml:Attribute NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic" Name="Groups-Membership">
      <saml:AttributeValue xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">
         {group1};{group2}
      </saml:AttributeValue>
   </saml:Attribute>
   ```

2. Use the **Group Attribute Value** and **Mapping of Group to Roles** settings to describe the structure of the SAML group attribute from which AppDynamics needs to extract the group value and the roles associated with those values. The Controller can extract Group Attribute values based on the following options:
   - Singular Group Values: The response contains an `AttributeValue` element with a single group-mapping value.
   - Multiple Nested Group Values: The response contains more than one `AttributeValue` element, each with a single group-mapping value.
   - Singular Delimited Group Value: The response contains one `AttributeValue` element with multiple, delimiter-separated group-mapping values.
   - Regex on Singular Group Value: The response contains a single `AttributeValue` element from which you want to extract the group-mapping value with a regular expression.

   The next sections provide examples for each option.
3. With any option selected, select the **Value is in LDAP Format** checkbox if the value or values returned by the group attribute value is in LDAP format. For example: `OU=AppDynamics-Users`. With this option enabled, only `AppDynamics-Users` is used to map to the SAML Group name.

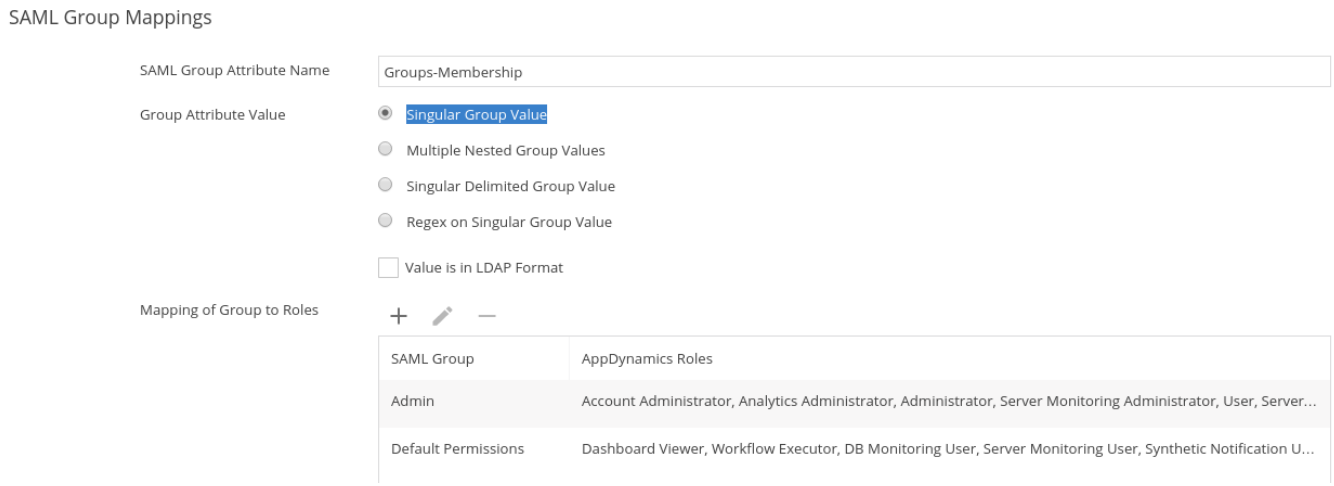
SAML Group Mappings

Singular Group Values

Select **Singular Group Value** if the SAML group attribute contains a single group, as in the following example.

```
<saml:AttributeStatement>
   <saml:Attribute Name="Groups-Membership" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
      <saml:AttributeValue xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">Admin<
/saml:AttributeValue>
   </saml:Attribute>
</saml:AttributeStatement>
```

For this example, AppDynamics would extract the value `Admin` and associate the user with a SAML Group with the same name. In this sample configuration, the user would get the configured roles assigned to the Admin SAML group, such as Account Administrator and Analytics Administrator.

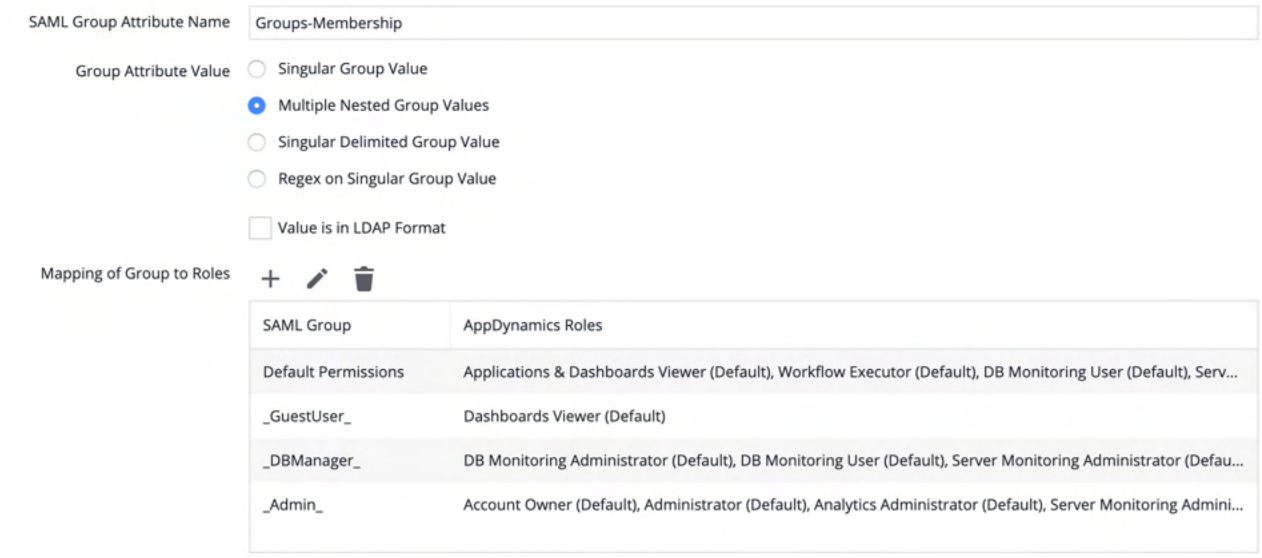


Multiple Nested Group Values

With **Multiple Nested Group Values** selected, AppDynamics expects multiple `AttributeValue` child elements under the SAML Attribute with the group information, as in the following example:

```
<saml:Attribute Name="Groups-Membership" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
   <saml:AttributeValue xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">_Admin_<
/saml:AttributeValue>
   <saml:AttributeValue xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">_DBManager_<
/saml:AttributeValue>
</saml:Attribute>
```

AppDynamics would extract `_Admin_` and `_DBManager_` from the example. In this sample configuration, the user with the previous response would receive the roles from the `_Admin_` and `_DBManager_` groups.



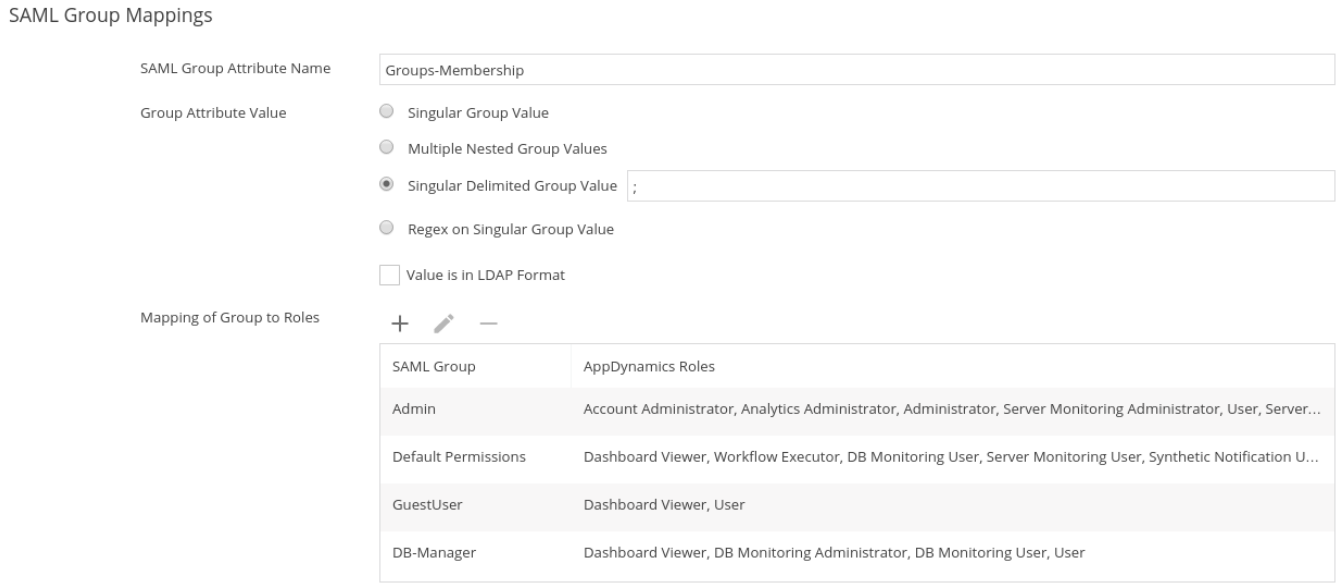
Singular Delimited Group Value

With this option selected, AppDynamics expects a single `AttributeValue` element with multiple, delimiter-separated values, as in the following example:

```
<saml:Attribute Name="Groups-Membership" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
   <saml:AttributeValue xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">Admin;DB-
Manager</saml:AttributeValue>
</saml:Attribute>
```

Specify the delimiter that separates the values to extract, such as a semi-colon in the example.

In this sample configuration, the user would get the AppDynamics roles associated with both the Admin and DB-Manager groups, such as the Dashboard Viewer, User, and DB Monitoring Administrator.



Regex on Singular Group Value

Choose this option to have AppDynamics extract group mapping values using a regular expression. Regular expressions enable you to pull group values from unstructured contexts, such as from within a larger string, as in the following response example:

```
<saml:AttributeStatement>
   <saml:Attribute Name="Groups-Membership" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
   <saml:AttributeValue xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">User
memberships in _Admin_ and _DBManager_ groups.</saml:AttributeValue>
   </saml:Attribute>
</saml:AttributeStatement>
```

In this example, the group names `_Admin_` and `_DBManager_` are embedded in the `AttributeValue` string. To extract those names, you can use a regular expression such as `_[a-zA-Z]_`. Like other types of group attribute sources, AppDynamics assigns all roles associated with both the `_Admin_` an d `_DBManager_` SAML Groups, as in this sample configuration:

SAML Group Mappings

SAML Group Attribute Name: Groups-Membership

Group Attribute Value:

- ○ Singular Group Value
- ○ Multiple Nested Group Values
- ○ Singular Delimited Group Value
- ◉ Regex on Singular Group Value: _[a-zA-Z]+_

☐ Value is in LDAP Format

Mapping of Group to Roles

SAML Group	AppDynamics Roles
GuestUser	User, Dashboard Viewer
DBManager	DB Monitoring Administrator, DB Monitoring User, User, Dashboard Viewer
Admin	Account Administrator, Analytics Administrator, Administrator, Server Monitoring Administrator, User, Server...
Default Permissions	Dashboard Viewer, Workflow Executor, DB Monitoring User, Server Monitoring User, Synthetic Notification U...

Encrypt SAML Responses

This page provides instructions on configuring encryption for SAML responses.

You can improve the security of the SAML authentication by encrypting the SAML response from the IDP to the service provider (your Controller). Your Controller shares a public key with the IDP and stores a private key to decrypt the public key.

You must be an Account Owner to configure the Controller to accept encrypted SAML responses.

Install OpenSSL

You will need to install the cryptography and SSL/TLS toolkit OpenSSL to generate the private key and certificate required for encrypting SAML responses.

Generate an x509 Certificate

1. Generate a private key:

```
openssl genrsa -out privatekey.pem 2048
```

2. Convert the private key into `pkcs8` format:

```
openssl pkcs8 -in privatekey.pem -topk8 -nocrypt -out privatekey.p8
```

3. Generate a certificate from the private key:

```
openssl req -new -x509 -key privatekey.pem -out ssocert.pem
```

Enable SAML Encryption

After you generate the private key in `pkcs8` format and the x509 certificate:

1. From your Controller, navigate to **AppDynamics > Administrator > Authentication Provider > SAML**.
2. Check **Enable** for **SAML Encryption**.

Add the Encrypted Certificate and the Private Key to the Controller

You must add the generated x509 certificate so that it can be shared with the IDP. You must also provide the private key so that the Controller can decrypt the SAML response from the IDP.

1. In the **SAML Encryption Certificate** text area, paste the content of your x509 certificate between the sections `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE----:`

```
-----BEGIN CERTIFICATE-----
// Insert x509 certificate content here
-----END CERTIFICATE----
```

2. In the **SAML Encryption Key** text area, paste the content of your p8 key file between the sections `-----BEGIN PRIVATE KEY-----` and `-----END PRIVATE KEY-----`:

```
-----BEGIN PRIVATE KEY-----
// Insert p8 key content here
-----END PRIVATE KEY-----
```

3. Click **Save**.

Configure IDP to Encrypt Response

Each IDP requires a different configuration for encrypting responses. Follow this high-level procedure:

1. From your IDP, enable SAML encryption. In some IDPs, this is known as *assertion encryption*.
2. Select an encryption algorithm. You can select any encryption method, and some IDPs will choose one for you.
3. Upload the x509 certificate that you generated in Generate an x509 Certificate.

Verify SAML Responses Are Encrypted

If you have enabled encryption for SAML in your Controller and the SAML responses from your IDP are not encrypted, your Controller will reject the SAML authentication when using SAML authentication to log in to your Controller.

To verify that the SAML response is encrypted:

1. Sign in to your Controller using the SAML flow from the service provider (your Controller) described in Verify the SAML Authentication Configuration.
2. From the Networks tab of the developer console of your browser, find the network request similar to the following, where `<controller_domain_name>` is the domain name of the machine hosting your on-premises Controller, and `<account_name>` is your AppDynamics account name.

   ```
   http://<controller_domain_name>:8090/controller/saml-auth?accountName=<account_name>
   ```

3. Select this network request and locate the parameter `SAMLResponse` in the response. You should see a long hash representing the encrypted SAML response:

   ```
   SAMLResponse=PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz48c2FtbDJwOlJlc3BvbnNlIERlc3RpbmF0aW9uPSJodHRwOi8vZWMyLTU0LTIxMi0wLTIxNi51cy13ZXN0LTIuY29tcHV0ZS5hbWF6b25hd3MuY29tOjgwODA]gRm9yb...
   ```

4. You can decode and inflate the encrypted SAML response with the `base64` command-line utility or an online tool:

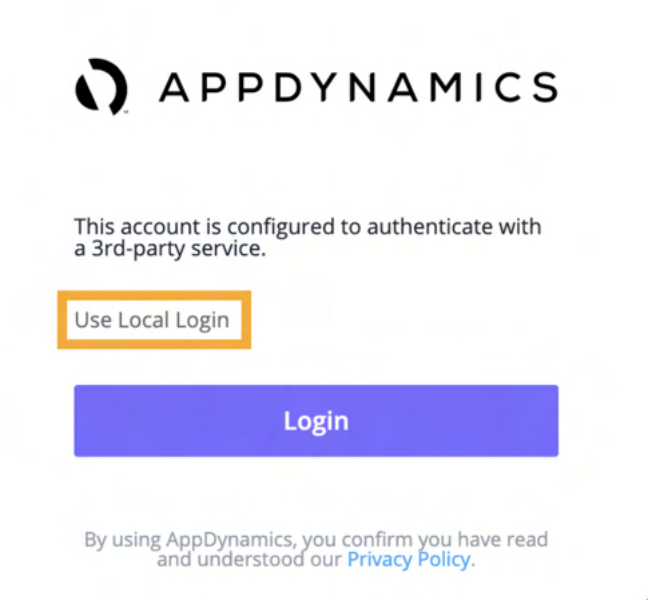
   ```
   base64 --decode
   SAMLResponse=PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz48c2FtbDJwOlJlc3BvbnNlIERlc3RpbmF0aW9uPSJodHRwOi8vZWMyLTU0LTIxMi0wLTIxNi51cy13ZXN0LTIuY29tcHV0ZS5hbWF6b25hd3MuY29tOjgwODA]gRm9yb...
   ```

   If your SAML Response was encrypted, the decoded and inflated string should contain the SAML XML response.

Disable SAML Authentication

This page provides instructions on disabling SAML authentication.

1. Navigate to your Controller and click **Use Local Login**.



2. Enter your local credentials and click **Login**.
3. From the Controller UI, navigate to **Administration > Authentication Provider**.
4. Select **AppDynamics** as the Authentication Provider
5. Click **Save**.

User Preferences

This page provides an overview of user preferences in AppDynamics. Controller UI users can change their passwords, account settings, date and time format, and other preferences in the **My Preferences** page.

User Preferences

You can access the controls for changing your display name, contact email, or password from the **My Preferences** page by clicking the **Edit** link under your email. The controls require you to enter your password to complete the change.

The display name is the name that the Controller uses to identify you in specific screen text and messages. For example, it appears in notifications to other Controller users when you share a dashboard with them. You cannot change your username. To change your username, you need to have an administrator delete your account and create another one with the new name.

You can display non-English languages in the Controller UI. To do so, enable your font preference to use system fonts. Using system fonts directs the UI to display text in the font configured for the system running the browser.

Some user account settings are attributes of local users (that is, users validated against credentials stored in AppDynamics). If your Controller is configured to use an external authentication mechanism instead, such as SAML or LDAP, you need to change the equivalent settings in the external system instead.

Strong Passwords

Your Controller administrator may require strong passwords for local accounts. A secure password must meet these requirements:

- At least eight characters in length
- Contain both uppercase and lowercase letters
- Include at least one number
- Not the same as your username or email address

View Preferences

The Controller UI provides the following user preferences:

- **Date Format**: By default, the format is MM/DD/YY (for example, 09/25/14). Choose an alternate format from the dropdown.
- **Use 24-Hour Time Format**: Enable this option if you want the UI to represent time in 24-hour time format instead of 12-hour clock format.
- **Enable Help Pop-ups**: Help popups provide contextual help text in the Controller UI. By default, they are enabled. To prevent help popups from appearing in the UI, clear this checkbox. Alternatively, you can prevent individual popups by selecting the **Don't Show Again** checkbox when the popup appears. To clear the list of popups marked as *Don't Show Again*, click **Reset All**.

You may need to log out of the UI and log back in to see the effects of your changes.

Advanced User Preferences

The user preferences page presents advanced options that include debug mode.

Debug Mode

The debug mode in the Controller UI is primarily intended for internal use by the AppDynamics development team. In some cases, you may be asked to enable debug mode when consulting with AppDynamics Support, for example, when you are troubleshooting an issue. However, it is important to note that specific debug mode options can negatively impact Controller performance. For this reason, you should only enable debug mode when directly advised to do so by AppDynamics Support.

AppDynamics Licensing

On this page:

Related pages:

- License Entitlements and Restrictions
- License Management in the Controller

There are two licensing models for AppDynamics products:

- Infrastructure-based Licensing - new model launched February 23, 2021
- Agent-based Licensing - original model

Infrastructure-based Licensing Model

The Infrastructure-based Licensing model offers license packages for Application Performance Monitoring (APM), End User Monitoring (EUM), Application Analytics, and security. Each package includes a combination of AppDynamics Agents and is based on a single metering unit. Because each agent can pull from a single metering unit, Infrastructure-based Licensing can support diverse applications and provide better license usage estimates.

This table shows how licenses in the Infrastructure-based Licensing model bundle multiple AppDynamics products.

Monitoring Types	AppDynamics Product	Infrastructure-based Licensing Packages						
		Enterprise	Premium	Infrastructure Monitoring	Real User Monitoring Peak	Real User Monitoring Pro	Log Analytics Pro	Secure Application
Application Monitoring	Application Performance Monitoring	✓	✓					
	Server Visibility	✓	✓	✓				
	Network Visibility	✓	✓					
	Database Monitoring	✓	✓					
	Transaction Analytics	✓						
End User Monitoring	Real User Monitoring (Browser and Mobile RUM)*				✓	✓		
	Real User Monitoring Analytics*				✓			
Log Analytics	Log Analytics*						✓	
Security	Secure Application*							✓

Additional information:

- *Real User Monitoring (and RUM Analytics), Log Analytics, and Secure Application licenses are available outside the Infrastructure-based Licensing Model.
- For Infrastructure-based Licensing package entitlements, restrictions, and definitions, see License Entitlements and Restrictions.
- If you have an Infrastructure-based Licensing license and Controller >= 21.2.0, you have a new UI to manage licenses in the Controller. See the Infrastructure-based Licensing tab on License Management in the Controller.

Agent-based Licensing Model

If you purchased AppDynamics licenses before February 23, 2021, you most likely have licenses in the Agent-based Licensing model. Agent-based Licensing is the original model in which each AppDynamics agent is licensed and metered individually.

Additional information:

- For Agent-based Licensing license entitlements, restrictions, and definitions, see License Entitlements and Restrictions.
- If you have an Agent-based Licensing license, you have the original UI to manage licenses in the Controller. See the Agent-based Licensing tab on License Management in the Controller.

License Entitlements and Restrictions

Related pages:

- Licensing
- License Management in the Controller

This page describes AppDynamics license entitlements, restrictions, and definitions.

License Entitlements in the Infrastructure-based Licensing Model

The table below describes license packages and package editions within the Infrastructure-based Licensing model. For information on the Infrastructure-based Licensing model, see Licensing.

License	Entitlements
Application Performance Monitoring	
Enterprise	• Monitoring for servers, supported application languages, networks, and databases instrumented with AppDynamics for up to 1 CPU Core • Access to the AppDynamics Events Service • Event Service data retention is limited to 8 days • Transaction Analytics for Enterprise: Instrument up to 400,000 business transaction events per 24-hour period and access to the AppDynamics-hosted Events Service, with a data maximum of 50 GB per account per day for the following product types: AppDynamics for Java, .NET, and Node.js. Data retention limited to 8 days. Additional retention of 30, 60, or 90 days is available as an add-on. • See entitlements for Premium package

Premium	• Monitoring for servers, supported application languages, networks, and databases instrumented with AppDynamics for up to 1 CPU Core
	• See entitlements for:
	  • APM Any Language
	  • Network Visibility
	  • Database Visibility
	  • Infrastructure Monitoring package

Infrastructure Monitoring	• Monitoring for servers instrumented with AppDynamics for up to 1 CPU Core
	• See entitlements for:
	  • Server Visibility (SIM mode)
	  • Machine Agent (MA, MA+ mode)
	  • Cluster Agent
	  • .NET Machine Agent

Security	
AppDynamics with Cisco Secure Application	• Monitoring for supported applications languages instrumented with AppDynamics for up to 1 CPU Core • See entitlement for: • Java
End User Monitoring	
Real User Monitoring (SaaS) - Peak Edition	• One unit of Real User Monitoring (SaaS) • Access to Analytics functionality for Browser Real User Monitoring (SaaS) • Access to Analytics functionality for Mobile Real User Monitoring (SaaS)
Real User Monitoring (on-prem) - Peak Edition	• One unit of Real User Monitoring (on-prem) • Access to Analytics functionality for Browser Real User Monitoring (on-prem) • Access to Analytics functionality for Mobile Real User Monitoring (on-prem)

Real User Monitoring (SaaS) - Pro Edition	• 10,000,000 RUM Tokens • Access the AppDynamics-hosted EUM Cloud and AppDynamics-hosted Events Service. • See entitlement for Browser Real User Monitoring (SaaS) • See entitlement for Mobile Real User Monitoring (SaaS)
Real User Monitoring (on-prem) - Pro Edition	• 10,000,000 RUM Tokens • See entitlement for Browser Real User Monitoring (on-prem) • See entitlement for Mobile Real User Monitoring (on-prem)

License Entitlements in the Agent-based Licensing Model

The table below describes licenses and license editions within the Agent-based Licensing model. For information on the Agent-based Licensing model, see Licensing.

License	Entitlement
Application Performance Monitoring	
APM Any Language / AppDynamics for Java, .NET or Node.js (SaaS) - Peak Edition	• One unit of APM Any Language / AppDynamics for Java, .NET or Node.js (SaaS); and • One unit of Transaction Analytics (SaaS); and • One unit of Network Visibility; and • One unit of Server Visibility
APM Any Language / AppDynamics for Java, .NET or Node.js (on-prem) - Peak Edition	• One unit of APM Any Language / AppDynamics for Java, .NET or Node.js (on-prem); and • One unit of Transaction Analytics (on-prem); and • One unit of Network Visibility; and • One unit of Server Visibility
APM Any Language Microservices (SaaS) - Peak Edition	• The same entitlement as 5 APM Any Language (SaaS) units, restricted to use with the following: Docker Containers, all CloudFoundry-based providers, Redhat OpenShift, Heroku Dyno, Microsoft Azure App Services (including Azure WebApps, Azure WebJobs and Azure API Apps), Microsoft Azure Functions hosted on a Microsoft App Service plan, Microsoft Azure Service Fabric, Microsoft Azure Containers, Amazon Elastic Beanstalk, Oracle PaaS (Java and Node.js only) and Bluemix Containers; and • One unit of Transaction Analytics (SaaS); and • One unit of Network Visibility; and • One unit of Server Visibility
APM Any Language Microservices (on-prem) - Peak Edition	• The same entitlement as 5 APM Any Language (on-prem) units, restricted to use with the following: Docker Containers, all CloudFoundry-based providers, Redhat OpenShift, Heroku Dyno, Microsoft Azure App Services (including Azure WebApps, Azure WebJobs and Azure API Apps), Microsoft Azure Functions hosted on a Microsoft App Service plan, Microsoft Azure Service Fabric, Microsoft Azure Containers, Amazon Elastic Beanstalk, Oracle PaaS (Java and Node.js only) and Bluemix Containers; and • One unit of Transaction Analytics (on-prem); and • One unit of Network Visibility; and • One unit of Server Visibility
SAP Bundle (SaaS) - Peak Edition	• One unit of SAP - ABAP Agent; and • One unit of Datavard Insights; and • One unit of Transaction Analytics (SaaS); and • One unit of Network Visibility; and • One unit of Server Visibility

SAP Bundle (on-prem) - Peak Edition	• One unit of SAP - ABAP Agent; and • One unit of Datavard Insights; and • One unit of Transaction Analytics (on-prem); and • One unit of Network Visibility; and • One unit of Server Visibility
APM Any Language - Advanced Edition	• One unit of APM Any Language (on-prem or SaaS as applicable); and • One unit of Network Visibility; and • One unit of Server Visibility
APM Any Language Microservices - Advanced Edition	• The same entitlement as 5 APM Any Language (on-prem or SaaS as applicable) units, restricted to use with the following: Docker Containers, all CloudFoundry-based providers, Redhat OpenShift, Heroku Dyno, Microsoft Azure App Services (including Azure WebApps, Azure WebJobs and Azure API Apps), Microsoft Azure Functions hosted on a Microsoft App Service plan, Microsoft Azure Service Fabric, Microsoft Azure Containers, Amazon Elastic Beanstalk, Oracle PaaS (Java and Node.js only) and Bluemix Containers; and • One unit of Network Visibility; and • One unit of Server Visibility
APM Any Language - Pro Edition	One license unit from the following product types: Java, .NET, Node.js, PHP, Python, Web Server, C++, Go, IBM Integration Bus, or the rights to monitor 16 Value Units of IBM Mainframe capacity.
APM Any Language Microservices (5pack) - Pro Edition	The same entitlement as 5 APM Any Language units, restricted to use with the following: Docker Containers, all CloudFoundry-based providers, Redhat OpenShift, Heroku Dyno, Microsoft Azure App Services (including Azure WebApps, Azure WebJobs and Azure API Apps), Microsoft Azure Functions hosted on a Microsoft App Service plan, Microsoft Azure Service Fabric, Microsoft Azure Containers, Amazon Elastic Beanstalk, Oracle PaaS (Java and Node.js only) and Bluemix Containers.
Java Agent - Pro Edition	Instrument a single JVM instance; snapshots are retained for 14 days unless archived
Java Microservices - Pro Edition	Instrument a single JVM instance with a heap allocation less than or equal to 1 GB Note: This entitlement shall only apply to existing customers of AppDynamics for Java Microservices that have licensed a SKU for AppDynamics for Java Microservices. This entitlement is mutually exclusive with the entitlement to use AppDynamics for Any Language licenses.
.NET Agent - Pro Edition	• Instrument an unlimited number of CLRs, including .NET web applications, .NET Windows services, and .NET standalone apps, on a single Windows OS instance; snapshots are retained for 14 days unless archived • For APM Any Language Microservices, instrument an unlimited number of CLRs on a single instance of an application • For .NET on Microsoft Azure for Cloud Services, instrument an unlimited number of CLRs running in a single instance of a Microsoft Azure Web role or Worker role
PHP Agent - Pro Edition	Instrument a single PHP runtime instance; snapshots are retained for 14 days unless archived
Node.js Agent - Pro Edition	Instrument up to 10 Node.js processes per OS instance; snapshots are retained for 14 days unless archived
Python Agent - Pro Edition	Instrument an unlimited number of Python processes running on a single OS instance; snapshots are retained for 14 days unless archived
Web Server Agent - Pro Edition	Instrument an unlimited number of Apache Web Server instances on a single OS instance; snapshots are retained for 14 days unless archived
C++ Agent - Pro Edition	Instrument an unlimited number of C++ runtime applications on up to 3 OS instances using the software development kit; snapshots are retained for 14 days unless archived Instrument an unlimited number of Ruby runtime applications on up to 3 OS instances using the Ruby agent; snapshots are retained for 14 days unless archived
Go Agent - Pro Edition	Instrument up to 3 Go processes; snapshots are retained for 14 days unless archived. This is available as part of the APM Any Language license only.

SAP - ABAP Agent - Pro Edition	Instrument a single application server
IBM Integration Bus - Pro Edition	Instrument an unlimited number of IBM Integration Bus processes on 1 OS instance. This is available as part of the APM Any Language license only.
Universal Agent - Pro Edition	Deploy on a single instance of an OS
Datavard Insights - Pro Edition	Instrument a single application server
Debugging	
AppDynamics Deep Code Insights, powered by Rookout	Application debugging support for one license unit of AppDynamics APM Any Language, configured for the following product types only: AppDynamics for Java, .NET, Node.js, PHP, Python.
Infrastructure Visibility	
Machine Agent - Pro Edition	Instrument a single instance of an OS
Server Visibility - Pro Edition	Instrument a single instance of an OS Monitor a single Kubernetes cluster
Network Visibility - Pro Edition	Instrument an unlimited number of Java nodes on a single instance of an OS
Service Availability Monitoring - Pro Edition	Instrument 60 HTTP-based services
Database Visibility - Pro Edition	Monitor one Database Instance of the following*: Couchbase Server, IBM DB2, Microsoft SQL Server, MySQL, PostgreSQL, Sybase ASE, Sybase IQ, Oracle. Monitor one Microsoft Azure SQL database or one Microsoft Azure SQL Managed Instance. Monitor one MongoDB replica set (or, if no replica sets exist, one standalone MongoDB instance). *For clustered database products (e.g. Couchbase or Oracle RAC), monitor one Database Instance per node of the cluster. When multiple Database Instances are grouped together under a single Collector using the Collector/Sub-Collector configuration, licensing is per Database Instance as above. Event Data is retained for 10 days for each Database Instance monitored by AppDynamics for Database Visibility.
AppDynamics for NetApp (on-prem) - Pro Edition	Monitor one NetApp controller for the following NetApp controller types: FAS2nnn and FAS3nnn. Monitor ½ NetApp controller for the following NetApp controller types: FAS6nnn and FAS8nnn series; two license units are required to monitor a full NetApp controller of series FAS6nnn or FAS6nnn. Event Data is retained for 14 days by default for each NetApp controller monitored by a unit of AppDynamics for NetApp (on-prem).
Application Analytics	
Browser Analytics (SaaS)	Access to Analytics functionality for Browser Real User Monitoring (SaaS). Access to AppDynamics-hosted EUM Cloud and AppDynamics-hosted Events Service. For Browser Real User Monitoring data storage and retention entitlements, see details under “Browser Real User Monitoring (SaaS)”

Browser Analytics (on-prem)	Access to Analytics functionality for Browser Real User Monitoring (on-prem). Customer is not entitled to access the AppDynamics-hosted EUM Cloud or AppDynamics-hosted Events Service. For Browser Real User Monitoring data storage and retention entitlements, see details under “Browser Real User Monitoring (on-prem)”
Mobile Analytics (SaaS)	Access to Analytics functionality for Mobile Real User Monitoring (SaaS). Access to AppDynamics-hosted EUM Cloud and AppDynamics-hosted Events Service. For Mobile Real User Monitoring data storage and retention entitlements, see details under “Mobile Real User Monitoring (SaaS)”
Mobile Analytics (on-prem)	Access to Analytics functionality for Real User Monitoring (on-prem). Customer is not entitled to access the AppDynamics-hosted EUM Cloud or AppDynamics-hosted Events Service. For Mobile Real User Monitoring data storage and retention entitlements, see details under “Mobile Real User Monitoring (on-prem)”
Log Analytics (SaaS)	Publish/index 5 GB of log data per 24-hour period (limited to 8 days of data storage) and access the AppDynamics-hosted Events Service. Additional retention beyond 8 days can be purchased as an add-on.
Log Analytics (on-prem)	Publish/index 5 GB of log data per 24-hour period (default data storage is 8 days and available up to 90 days). Customer is not entitled to access the AppDynamics-hosted Events Service.
Transaction Analytics (SaaS)	Instrument up to 1,000,000 business transaction events per 24-hour period and access to the AppDynamics-hosted Events Service, with a data maximum of 50 GB per account per day for the following product types: AppDynamics for Java, .NET, and Node.js. Data retention limited to 8 days. Additional retention of 30, 60, or 90 days available as an add-on.
Transaction Analytics (on-prem)	Instrument up to 1,000,000 business transaction events per 24-hour period (limited to 90 days of data storage) per license unit. Customer is not entitled to access the AppDynamics-hosted Events Service.
End User Monitoring	
Browser Real User Monitoring (SaaS) - Peak Edition	One unit of Browser Real User Monitoring (SaaS) and one unit of Browser Analytics (SaaS)
Browser Real User Monitoring (on-prem) - Peak Edition	One unit of Browser Real User Monitoring (on-prem) and one unit of Browser Analytics (on-prem)
Mobile Real User Monitoring (SaaS) - Peak Edition	One unit of Mobile Real User Monitoring (SaaS) and one unit of Mobile Analytics (SaaS)
Mobile Real User Monitoring (on-prem) - Peak Edition	One unit of Mobile Real User Monitoring (on-prem) and one unit of Mobile Analytics (on-prem)
Browser RUM (SaaS) - Pro Edition	• Instrument 10 Million Pageviews per 12-month period following the license provision date • Access to the AppDynamics-hosted EUM Cloud and AppDynamics-hosted Events Service • Raw page requests, sessions, and resources are retained as Event Data with a default limit of 100 GB per account per day for each Event Data type. • AppDynamics reserves the right to drop customer traffic if the foregoing limitation is exceeded by the end user. • Customers may also choose to store raw Ajax or other events requests as Event Data if they also have access to Analytics, and every 5 requests stored consumes one Pageview. Event Data is retained for 8 days. • Additional retention of Event Data for 30, 60 or 90 days is available as an add-on at additional cost, and requires the purchase of AppDynamics Browser Analytics (SaaS). Resource Performance data is retained for 6 days.

Browser RUM (on-prem) - Pro Edition	Instrument 10 Million Pageviews per 12-month period; raw page requests and sessions are retained as Event Data. Customers may also choose to store raw Ajax or other events requests as Event Data, and every 5 requests stored consumes one Pageview. Event Data is retained for 8 days by default. Customer is not entitled to access the AppDynamics-hosted EUM Cloud or AppDynamics-hosted Events Service.
Mobile RUM (SaaS) - Pro Edition	Instrument 5,000 Active Agents per calendar month and access the AppDynamics-hosted EUM Cloud and AppDynamics-hosted Events Service. In addition, for each Active Agent unit, 500 events per day of the following types are stored on the event service as Event Data: network requests, sessions, breadcrumbs and custom data. Crash reports are retained for 365 days and all other Event Data is retained for 8 days. Additional retention of Event Data for 30, 60 or 90 days is available as an add-on at additional cost, and requires the purchase of AppDynamics Mobile Analytics (SaaS).
Mobile RUM (on-prem) - Pro Edition	Instrument 5,000 Active Agents per calendar month. In addition, for each Active Agent unit, 500 events per day of the following types are stored on the event service as Event Data: network requests, sessions, breadcrumbs and custom data. Crash reports are retained for 365 days and all other Event Data is retained for 8 days by default. Customer is not entitled to access the AppDynamics-hosted EUM Cloud or AppDynamics-hosted Events Service.
Browser Synthetic Monitoring - Hosted Agent (SaaS) - Pro Edition	40,000 units of 5-second Synthetic Time Blocks on AppDynamics' hosted synthetic network per calendar month, and use Business Performance Monitoring features on the synthetic Events Data, if applicable, which is retained for 13 months. Each run of a synthetic job is rounded up to the next 5 seconds. Unused time is not rolled over to the following month. Customer is entitled to access the AppDynamics-hosted EUM Cloud or AppDynamics-hosted Events Service, both of which are required to use this product.
Browser Synthetic Monitoring - Private Agent - Per Location (SaaS) - Pro Edition	Run a synthetic job that hits 1 page from 1 location (multiple units add together to run jobs that hit multiple pages from multiple locations), and use Business Performance Monitoring features on the synthetic Events Data, if applicable, which is retained for 13 months. A job that fails to hit any pages is counted as 1 page. Customer is entitled to access the AppDynamics-hosted EUM Cloud or AppDynamics-hosted Events Service, both of which are required to use this product.
Browser Synthetic Monitoring - Private Agent - Unlimited Locations (SaaS) - Pro Edition	Run a synthetic job that hits 1 page from any number of locations, and use Business Performance Monitoring features on the synthetic Events Data, if applicable, which are retained for 13 months. Customer is limited to 500,000 Pageviews per calendar month period. Unused Pageviews are not rolled over to the following month. A job that fails to hit any pages is counted as 1 page. Customer is entitled to access the AppDynamics-hosted EUM Cloud or AppDynamics-hosted Events Service, both of which are required to use this product.
Browser Synthetic Monitoring - Private Agent - Per Location (on-prem) - Pro Edition	Run a synthetic job that hits 1 page from 1 location (multiple units add together to run jobs that hit multiple pages from multiple locations), and use Business Performance Monitoring features on the synthetic Events Data, if applicable, which is retained for 13 months. A job that fails to hit any pages is counted as 1 page. Customer is not entitled to access the AppDynamics-hosted EUM Cloud or AppDynamics-hosted Events Service.
Browser Synthetic Monitoring - Private Agent - Unlimited Locations (on-prem) - Pro Edition	Run a synthetic job that hits 1 page from any number of locations, and use Business Performance Monitoring features on the synthetic Events Data, if applicable, which are retained for 13 months. Customer is limited to 500,000 Pageviews per calendar month period. Unused Pageviews are not rolled over to the following month. A job that fails to hit any pages is counted as 1 page. Customer is not entitled to access the AppDynamics-hosted EUM Cloud or AppDynamics-hosted Events Service.
IoT for Connected Devices (SaaS) - Pro Edition	Instrument 1,000 application instances that are embedded on a connected device (or multiple devices) per calendar month and access to the AppDynamics-hosted Events Service, with a data maximum of 50 GB per account per day. Only customers with at least 100,000 application instances that are (i) embedded on a connected device (or multiple devices), (ii) eligible to be instrumented AppDynamics for IoT and (iii) are actively in use by customer's end users shall be entitled to use AppDynamics for IoT. AppDynamics for IoT is only available to AppDynamics SaaS customers. An "eligible" application instance is an application that can be instrumented using AppDynamics' C/C++ SDK, Java SDK or AppDynamics Rest APIs (each for IoT). Event Data is retained for 8 days. Additional retention of Event Data for 30, 60 or 90 days is available as an add-on at additional cost.

License Restrictions

The table below describes license restrictions that are applicable to all products.

Category	Restriction

Third-party components	License restrictions with respect to any third party components included in the AppDynamics software, with which customers are required to comply, are located on the Legal Notices page.
Compatibility	Compatibility limitations with respect to any third party components are located on the Supported Environments page.
Use of MySQL database(s)	Customers are prohibited from using the MySQL database(s) included with the AppDynamics software with any other product or for any other purpose other than for the AppDynamics software as provided.
Machine and Universal Agents	Each unit of the products in the Application Performance Management includes one unit of Machine Agent and Universal Agent at no cost, but such Machine Agent and Universal Agent shall only be used on the instance of the OS being monitored by the purchased unit.
APM Peak Edition	AppDynamics for Java, .NET or Node.js : During the License Term of the Subscription Licenses (or, if applicable, during the Maintenance and Support Term of perpetual licenses) on the applicable Order Form, End User may, once per calendar quarter and upon 24 business hours' written notice, exchange licenses among the following types of Software units: AppDynamics Pro Edition for Java, .NET and Node. js on a one-for-one basis, and separately AppDynamics Test & Dev Edition (each an “Exchangeable Group”) for the same products on a one-for-one basis. For clarity, the list prices of all products within each Exchangeable Group (Pro Edition and Test & Dev Edition, respectively) are the same and through such exchanges, End User may not at any time use products with cumulative value exceeding the total list price of products on the applicable Order Form.
APM Any Language	APM Any Language customers are prohibited from instrumenting APM Any licenses on SAP application servers unless such customer has purchased the requisite AppDynamics for SAP - ABAP Agent(s).
EUM Peak Edition	Browser or Mobile RUM: During the License Term of the Subscription Licenses (or, if applicable, during the Maintenance and Support Term of perpetual licenses) on the applicable Order Form, End User may, once per calendar quarter and upon 24 business hours' written notice, exchange licenses among the following types of Software units: AppDynamics Pro Edition for Browser Real User Monitoring and Browser Analytics for units of Mobile Real User Monitoring and Mobile Analytics on a one-for-one basis (and vice versa), and separately AppDynamics Test & Dev Edition for the same products (and vice versa) on a one-for-one basis. End User may not exchange any AppDynamics for Browser Real User Monitoring licenses during any year in which End User has used any pageviews of such licenses. For clarity, the list prices of all products within each Exchangeable Group (Pro Edition and Test & Dev Edition, respectively) are the same and through such exchanges, End User may not at any time use products with cumulative value exceeding the total list price of products on the applicable Order Form.
Upgrades	Upgrade entitles the customer to the entitlement of the product to which the customer is upgrading.
Standard University Subscription	Customers that have purchased any of the software listed below under the heading "Application Performance Management" shall be entitled to access the AppDynamics University Self-Paced Library, subject to the conditions set forth in this paragraph. No greater than 20 unique users per month per AppDynamics account are entitled to this access. Access shall be limited to customers that have purchased this software either (i) on a subscription basis but remain current on payment of all subscription license fees, if any or (ii) on a perpetual basis but remain current on payment of all maintenance and support fees, if any. Customers may log in using AppDynamics account credentials.

Data Storage and Retention Entitlements

Deployment	Entitlement
SaaS	• Metric Data in one-minute increments are retained for 8 days • Metric Data in one-hour increments are retained for 365 days • Event Data is retained for 8 days ⓘ For the Enterprise package, additional retention of 30, 60, or 90 days is available as an add-on.
On-Premises	• Metric Data in one-minute increments are retained for 4 hours • Metric Data in ten-minute increments are retained for 48 hours • Metric Data in one-hour increments are retained for 365 days • Event Data is retained for 8 days, except that retention for on-premises deployments is configurable

License Entitlement Definitions

- **Active Agent**: A unique installation of an application instrumented with an AppDynamics Mobile Real User Monitoring Agent that was launched on a device in a given calendar month
- **Database Instance**: A complete running database environment, including the software, table structure and other functionality. Typically this corresponds to a host/port combination where the port is the listener port for the database software. Database administrators might create multiple database environments on the same physical hardware, with each such environment having its own unique listener port.
- **Dedicated Controller**: A version of the central repository and analytics engine component of the Software where End User’s data is logically separated from that of other customers
- **EUM Cloud**: The end user monitoring component that receives and processes data received from end user devices

- **Event Data**: The raw data generated by the Software and stored in the Events Service
- **Events Service**: The component that stores the Event Data
- **GB**: gigabyte
- **Instrument**: To collect data, monitor, troubleshoot and manage using the AppDynamics software
- **Metric Data**: The aggregate value of the data collected by the Software over a given period of time
- **Pageview**: A Pageview is an instance of a base page, virtual page, or iFrame loaded by a web browser. Each base page view, iframe view, and virtual page view is counted as a Pageview. Repeated views of one page are counted as separate Pageviews.
- **RUM Token**: The common unit of measure for consumption of Real User Monitoring products during each 12 calendar month period in a License Term, with the first such period beginning on the start date of the relevant License Term; one Pageview consumes one RUM Token, and deployment of one Active Agent consumes 160 RUM Tokens.
- **CPU Core**: Each logical thread core or processor reported by the operating system on a unique instance of a physical or virtual host machine and detected by AppDynamics. Where AppDynamics cannot detect CPU Core counts on a host, AppDynamics will calculate the licensed CPU Cores on a per-Agent basis as follows:

    - The quantity of APM Agents deployed multiplied by 4; and
    - The quantity of Database Agents deployed multiplied by 12 per unique database host

License Management in the Controller

Related pages:

- License Entitlements and Restrictions
- Licensing

This page describes how to manage licenses in the Controller.

In the Controller, you can view and manage license usage and create rules for APM licenses. The Controller UI differs based on what license you have. The instructions below are separated between two licensing models: Infrastructure-based Licensing and Agent-based Licensing. To find out what license you have, see License Entitlements.

License Best Practices and Troubleshooting

This page describes best practices and troubleshooting for AppDynamics licenses. The information is organized by licenses model: Infrastructure-based Licensing and Agent-based Licensing. To learn about the licensing models, see Licensing.

Apply or Update a License File

For AppDynamics Pro trial installations and SaaS Controllers, the license is applied and installed automatically.

If you are using AppDynamics on-premises, you must manually apply the license file after installing the Controller. Whenever you purchase additional units, you must update the license file. If you update Application Analytics licenses in an on-premises environment, you may have to do some manual updates when you update your Controller license. See License Entitlements and Restrictions.

Apply a License

1. Copy the license file to the Controller installation directory (typically found within {controller_home}/platform/product/controller/). After copying the file, allow up to five minutes for the license change to take effect.

   If the license has not been detected after five minutes, try the following:

   - Linux: Execute the following command:

     ```
     touch license.lic
     ```

   - Windows: Replace the license file again.

2. To verify that the license file is applied, check that the license you purchased appears in the **Peak Usage** tab of the **License** page.

Apply a License to a Multi-Tenant Controller

If you are updating the license file for a multi-tenant Controller, you must manually edit the accounts to affect the change. For example, if you purchase an additional 200 licenses to distribute across two accounts, you must manually allocate those other units by updating each account.

1. Copy the license file to the Controller installation directory. After copying the license file, allow up to five minutes for the license change to take effect.
2. Log in with root user at `http://<controller_host>:<port>/controller/admin.jsp`.
3. Click **Accounts** and select the account you want to apply the license to.
4. Update the license expiration date and license units based on license.lic.
5. Save the changes.

Update a License

1. Rename your old license file to something other than license.lic (so that the Controller ignores it). After copying the license file, allow up to five minutes for the license change to take effect.

   If the license has not been detected after five minutes, try the following:

   - Linux: Execute the following command:

     ```
     touch license.lic
     ```

   - Windows: Replace the license file again.

2. To verify that your license file is applied, check that the license you purchased appears in the **Account Usage** tab of the **License** page.

Update License Rules

If you are using license rules, you must update them to account for the additional units you purchased after updating your license. For example, if you buy another 200 licenses to distribute across two license rules, you must manually allocate those additional units by updating the rules.

Update MAC Address

AppDynamics licenses for on-premises Controllers are tied to the machine's MAC address on which you installed the Controller. The original MAC address set up must be done by AppDynamics Operations when a new on-premises license is first provisioned. Afterward, you can update your MAC address manually.

You can change the MAC address up to twelve times within a 12-month period using these steps. If you need to change it again, contact the AppDynamics Licensing Team by emailing licensing-help@appdynamics.com.

To update the MAC address on a permanent license:

All License Admins that have access privileges for a permanent on-premises license can update MAC addresses. To view a list of these users, navigate to the **Subscriptions** page under **Account**.

1. Navigate to www.appdynamics.com.
2. Log in with your AppDynamics user credentials.
3. Ensure that your account is set up with **License Admin** permissions.
4. On the main navigation bar, click **Account >Subscriptions**.
5. Click the [Controller] **Name** or **Actions**.
6. Click **Edit** next to the MAC address.
7. Enter the new hexadecimal MAC address.
8. Click **Save**.

Sensitive Data Collection and Security

AppDynamics collects data on application performance, health, and resources, in addition to the application components (transactions, code libraries) and related infrastructure (nodes, tiers). Before deploying AppDynamics, you should ensure that your application is secure and does not expose sensitive data. This page describes how to prevent sensitive data collection in your application environment.

Prevent Sensitive Data Collection

If your environment contains sensitive data that should not be processed by an AppDynamics product or sent to the AppDynamics SaaS environment, you should avoid:

- Applications that transmit sensitive data in URL query parameters
- Enabling HTTP request parameter capture
- Enabling bind variable capture
- Applications that send sensitive data in error logs and log files
- Allowing method invocation data collection
- Log captures

If you do capture logs, ensure that you mask values in those logs. The following sections cover additional measures that you can take to ensure you do not expose sensitive data.

SaaS Deployment

AppDynamics supports encryption at rest in its SaaS deployments on personally identifiable information and sensitive business data.

On-premises Deployment

AppDynamics offers an on-premises solution for customers who want to maintain full control over their deployment of the software. With this type of implementation, AppDynamics has no access to the software or the data it collects and processes. Customers subject to strict regulatory requirements for data security may want to consider an on-premises solution. On-premises customers are responsible for encrypting their data by either using self-encrypting drives or other non-product solutions.

Role-based Access Control

You can use role-based access controls (RBAC) to limit the number of users who can access data collection features. The controls let you restrict a user's access to specific functions, data, analytics queries, and APIs.

You can control user access to data by specifying permissions for each user role. To configure user access, navigate to **Settings** > **Administration**. See Analytics and Data Security and Create and Manage Custom Roles.

Suppress Raw SQL Capture

Application Monitoring collects raw SQL as prepared statements captured with dynamic parameters bound to runtime values.

You can disable the capture of raw SQL if it contains sensitive data. When you disable raw SQL capture, the SQL call appears in its original form, but with question mark parameters in place of sensitive data.

To disable the capture of raw SQL for an application, navigate to **Configuration** > **Instrumentation** > **Call Graph Settings** > **SQL Capture Settings**. Uncheck **Capture Raw SQL**.

You can also disable bind variable capture. Bind variables are placeholders for literal data in your SQL statements. When you disable bind variable capture, the values of bind variables are not displayed. For more information, see Call Graph Settings.

Hide Query Literals

Database Visibility hides query literals by default since queries can contain sensitive user data.

To verify that query literals are hidden for a database, navigate to **Configuration**. In the Security section, ensure that you have chosen **Remove literals from the queries**. See Configure Query Literals Security.

You may also want to use bind variables as placeholders for literal data in your SQL statements.

Exclude Error Logs

Application Monitoring logs exceptions and errors that match parameters you specify in your custom logger. You may want to exclude sensitive payload data so that it does not show up in error logs.

To exclude a class in your application:

1. Navigate to **Tiers & Nodes** > **Actions** > **Configure App Server Agent**.
2. Select **Use Custom Configuration**.
3. Click (**+**) to create a new agent property.
4. Set the agent property name to exceptions-to-ignore.
5. Set the agent property value to the name of the class you want to exclude.

See Error Detection.

Mask Log Analytics Values

When configured, Application Analytics collects performance data from your app server agents, data from your log files, and performance and sessions data from End User Monitoring. You can mask sensitive information in your log analytics data.

To mask log analytics data:

1. Navigate to **Analytics** > **Configuration** > **Log Analytics** > **Source Rules**.
2. Click the source rule that you want to specify masking for.
3. In the Field Management tab, next to ThreadName, you can specify the starting and ending position of the data you want to mask, and the character to use as the masking value.

See Configure Log Analytics Using Source Rules.

Disable the Data Collector

You can suppress data collection of HTTP request payloads, raw SQL, and other user data.

For the Java Agent, configure the `disabled-features` node property in the Controller UI.

For the .NET Agent, edit the `config.xml` file and set the `disabled-features` property to the names of features that you want to disable.

```
<property name="disabled-features" value="RAW_SQL,LOG_PAYLOAD,METHOD_INV_DATA_COLLECTOR,HTTP_DATA_COLLECTOR,
CUSTOM_EXIT_SNAP_DATA"/>
```

See App Agent Node Properties Reference and .NET Agent Configuration Properties.

Filter Sensitive Data in Environment Variables

You can mask sensitive data found in Java environment variables and system properties. To mask sensitive data, add the `sensitive-data-filter` property to app-agent-config.xml. The valid attributes are `applies-to`, `match-type`, and `match-pattern`. See Filter Sensitive Data.

Data Privacy Policy Dialog

Data collection has regulatory, legal, and customer-defined policies that you must follow. AppDynamics provides a data privacy policy reminder, in the form of a UI dialog, when you or your users configure parts of the AppDynamics products that could be used to collect regulated or other protected information.

This customizable statement is present in all areas of the AppDynamics UI where you can configure data collection. AppDynamics displays a default message if you have not made any customizations.

AppDynamics logs an event when it displays the data privacy policy dialog to you or another user.

Data Collection Dashboard

This page describes the Data Collection Dashboard, which displays all collected and processed data by AppDynamics.

About the Data Collection Dashboard

AppDynamics allows you to customize most data configurations. Due to the various AppDynamics agent types and products, you can change the configuration in many different places. This flexibility can make it challenging to keep track of all of your settings.

The Data Collection Dashboard addresses this challenge by providing a single, easy-to-access view of the current state of all of your configuration parameters that may affect your organization's security and privacy. The Data Collection Dashboard should help you understand what data AppDynamics is collecting for your configured application instance at any point in time.

The unified view is only available to admins.

To view the Data Collection Dashboard, click **Settings**  **> Data Collection Dashboard**.

The dashboard provides a read-only view. You can change the configuration values in the respective places of the platform.

You can find online help for a data collector type. To do so, navigate to **Applications**, choose an application, and then click **Configuration > Instrumentation > Data Collectors**. Click **Help**  in the data collector box of interest to view more information on that particular data collector and how to configure it.

You can click **Export** to save a raw JSON file of the selected data collection settings and share with non-admins, auditors, and sales.

Data Collector Types

This table lists all of the data collector types that you can view on the Data Collection Dashboard.

Data Collector Type	Description
Agents Logging Session	Agents Logging Sessions log information with a specific type of logging information, such as application configuration changes, metric data, and system agent registrations.
Agents Metadata Default Config	Agents Metadata Default Configs collect app agents transaction metadata.
Browser Settings Config	Browser Settings Configs collect Browser RUM and Synthetic metrics.
Custom Data Browser	Custom Data Browsers customize the agent to collect custom metrics for the browser.
Custom Data Mobile	Custom Data Mobiles customize the agent to collect custom data types such as information points, custom timers, and custom metrics.
DB Collector	Database Collectors run within the Database Agent to collect performance metrics about your database instances and servers.
DB Security Config	Database Security Configs collect literals, which can contain sensitive user data, retrieved from the database.
EUM Settings Config	EUM Settings Configs collect EUM metrics, custom data, and Analytics data.
GUID Injection	GUID Injections correlate logs and transaction analytics data to specific business transaction requests.
HTTP Request	HTTP Requests capture the URLs, parameter values, headers, and cookies of HTTP messages exchanged in a business transaction.
Information Points	Information Points reflect the data state across all invocations of a method, independently of business transactions.
Log Analytics Source Rule	Log Analytics Source Rules log data sources using job files to collect log analytics data.
Method Invocation	Method Invocations capture code data, such as method arguments, variables, and return values.
Mobile Settings Config	Mobile Settings Configs collect Mobile RUM metrics.
SIM HTTP User Service Config	SIM HTTP User Service Configs collect Standalone Machine Agent data using HTTP listeners.
SQL Data Collector	SQL Data Collectors collect the business data from SQL parameters for use in transaction analytics.

Metrics and Graphs

This page describes metrics and graphs in AppDynamics.

AppDynamics reports metrics for monitored systems over time. The Controller provides a variety of graphs and tools that visualize metric data so you can effectively analyze your application's performance.

Metric Visualization Tools

While metric data appears throughout the Controller, the Metric Browser is the main interface for you to inspect metrics. The Metric Browser displays all the metric values collected over the selected time period as an expandable tree.

In the browser, you can drag and drop metrics from the tree onto graphs to compare metrics and analyze patterns.

Metrics

AppDynamics includes a wealth of metrics that reflect your application's performance. It also lets you extend and adapt metrics in specific ways. Information points, for example, allow you to create metrics based on data collected from application values, such as method parameters or return values. Percentile metrics let you configure metrics at deviation points that are useful to you.

You can create a metric from any application parameter using custom metrics. See Extensions and Custom Metrics for information on custom metrics.

Metric Browser

This page describes how the Metric Browser provides a metrics-oriented view of the monitored environment for a selected time period.

In the Metric Browser, you can see metrics by nodes, business transactions, the overall application, and more. You can also compare two or more metrics and view metrics relative to a baseline. Use this tool when you want to drill into and analyze the details for a specific metric.

1. Click **Metric Browser**.
2. Expand a metric group and double-click an option for that metric to display in the graph. Add multiple metrics in the same manner to compare them.
3. Mark the checkbox for the type of data you want to see for each metric.
4. Select **Filters** ( ) and choose how you want to view data:
    - **Show Tiers / Nodes with performance data** to show entities and metrics for live entities over the specified time range.
    - **Show Tiers / Nodes with no performance data** to show metrics/entities of entities that are not alive over the specified time range.
    - Select both checkboxes to show metrics and entities without checking the liveness of entities.
5. Select **Plot Points** to enable the plot points on the graph line.
6. Select a **Scale** option to adapt the scale of the y-axis unit according to the values in the graph.

To remove a single metric from the graph, click **X** or click **Clear** at the top of the metric browser to remove all metrics.

Metric Data Point Details

For most types of metrics in the browser, you can click any of the points in the graph to view more information about the metric observed at that point in time. The information shown includes the metric identifier, date and time of the observation and any of the following values relevant to the metric:

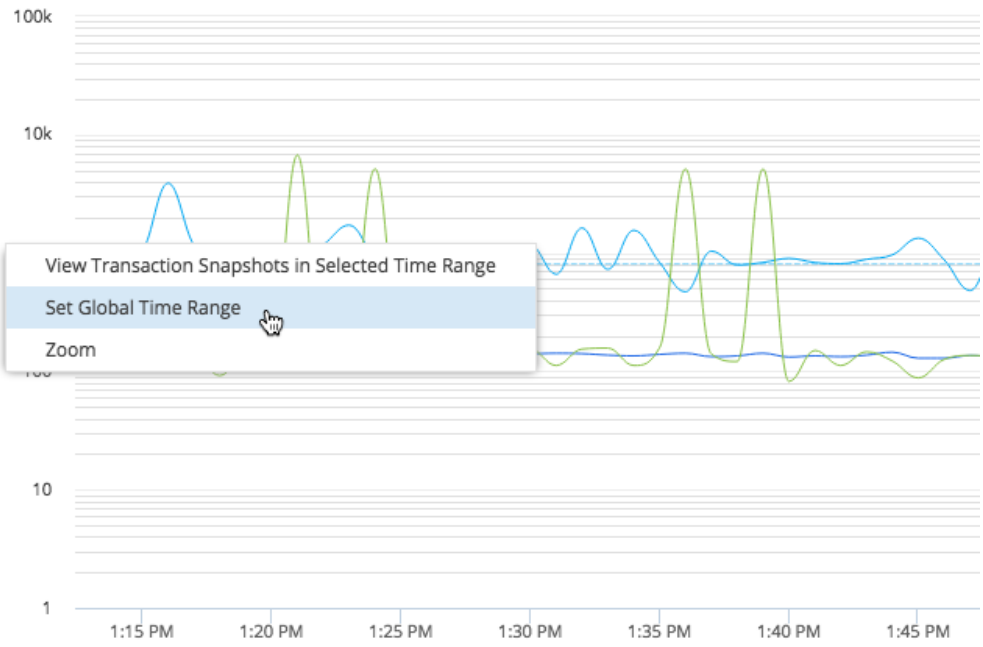
- **Obs**—The observed average of all data points seen for that interval. For the Percentile Metric for the App Agent for Java, this is the percentile value. For a cluster or a time rollup, this represents the weighted average across nodes or over time.
- **Min**—The minimum data point value for that interval.
- **Max**—The maximum data point value for that interval.
- **Sum**—The sum of all data point values for that interval. For the Percentile Metric for the App Agent for Java, this is the result of the percentile value multiplied by the Count.
- **Count**—The number of observations aggregated in that one point. For example, a count of five indicates that there were five one-minute data points aggregated into one point.
- **Base**—The baseline data for the metric. See Viewing Metrics Relative to Baselines.

By selecting the checkbox that corresponds to the metric information at the bottom of the graph, such as Min, Max, Count, or Sum, you can graph that information for the metric over the time range shown in the browser. Checkboxes for only the types of information relevant to the metric can be selected.

> For node-level metrics only, min and max values are available for any count- or sum-based metrics and for when the metric is rolled up to hourly or daily data points. Min and max values are not visible for tier- and application-level metrics. Count- and sum-based metrics include errors per minute, calls per minute, and so on.

Analyze Performance for a Time Range

Once you identify a time frame of interest in the metric browser, you may want to see snapshots in that time range. To analyze snapshots during a time range in the browser, click and hold the start of the time range in the graph and drag to the end of the time range. Release to view the **Time Range** pane.

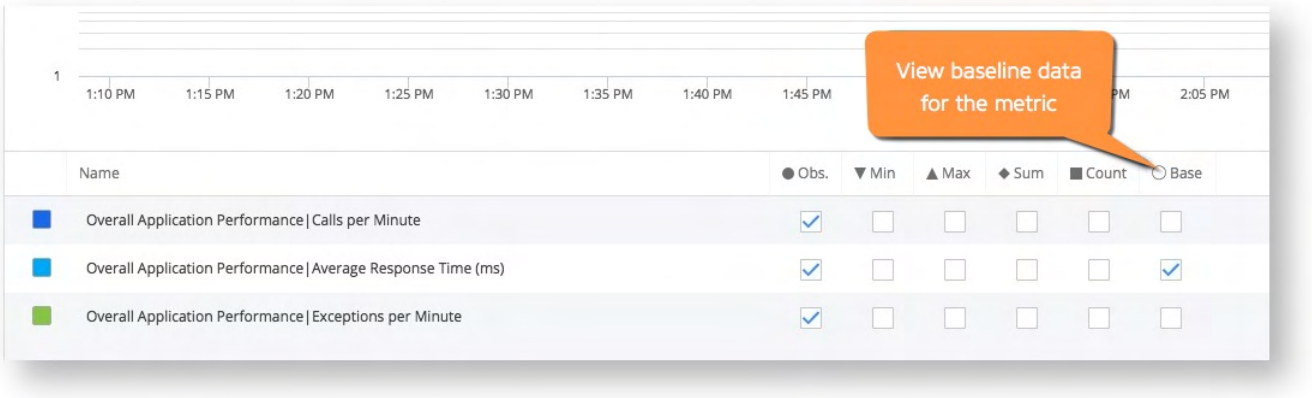


If you do not see a metric in the browser, you can add certain metrics by editing node properties. See App Agent Node Properties.

View Metrics Relative to Baselines

You can see how data in the Metric Browser compare to baseline values. Monitoring baseline deviation exposes performance metrics that might be violating your Service Level Agreements (SLAs). See Dynamic Baselines.

To display baseline patterns, select the baseline option for the metric in the view options for the metric.



Export Metric Data

You can export metric data that is currently displayed to a comma-separated values (CSV) file.

Exported data includes all the data displayed, even if there is a time range selected. To export data only for a specific time range, first set a global time range and then export.

To export all data:

1. Select **Actions** > **Export Data**.
2. In the popup, you can either copy the data to your clipboard or click **Export Data** to download the data as a CSV file.

To export data for an individual metric in the graph, click **Download** at the bottom of the graph.

Metric Data Resolution over Time

You can view metrics in the Controller for different time intervals or resolutions.

Metric Generation and Time Zone Differences

In a distributed system, the metrics that the Controller collects may come from different time zones.

Before sending metrics to the Controller, agents normalize the times associated with the metrics by applying a UTC (Coordinated Universal Time) offset appropriate to their region.

In the Controller UI, all times display in the Controller user's local time. To render local times, the UI applies the offset needed to render UTC time into the user's time zone as configured on the device on which the browser is running.

While the time zone used in the UI is taken from the local environment and cannot be modified apart from changing the computer's system time, each UI user can adjust the format of the time display, as described in User Preferences.

While the Controller UI displays times in the local system time, specific timestamps generated by AppDynamics reflect the Controller system time. These include, for example, event timestamps in notification messages, which indicate the Controller system time.

> SaaS Controllers use Pacific Time (PT) as their system time.

Metrics Data Resolution

Metrics Data Retention and Deletion

View Rolled Up Data in Graphs

Depending on your environment, AppDynamics measures hundreds, thousands, or even tens of thousands of metric values each minute. To display these results over time in a meaningful way, AppDynamics rolls up the data at regular intervals (for example, at one-minute, ten-minute, and one-hour intervals) based on how long ago the metric was measured.

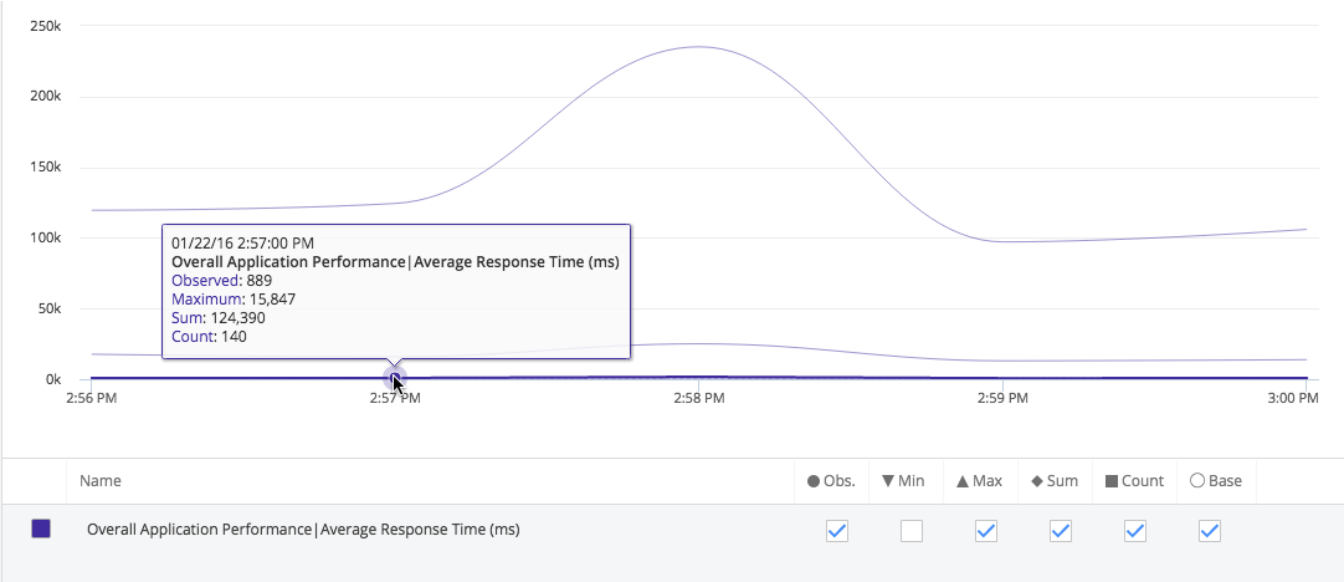
In graphs throughout the Controller UI, you can hover over a point on the chart to get the value of the metric data measured during the interval represented by the point. The point represents the beginning of the interval. The length of the interval depends on the selected time range. Admin-level users can adjust the amount of time data is retained at each resolution, although we do not generally recommend keeping data for longer than these default values. Default settings are listed below.

Rolled Up Data in Graphs

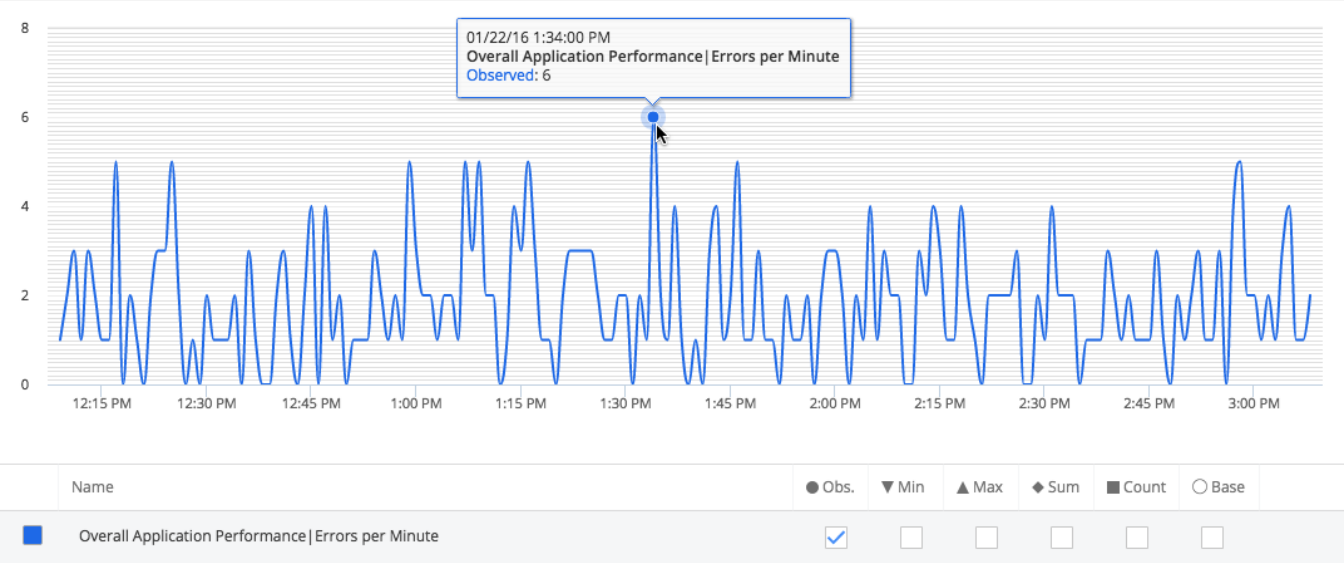
One-Minute Resolution

Each point on the graph represents a roll-up of all the values measured during one minute. For example, if AppDynamics measures the response time for an application for 300 calls in a minute, the response time values of those 300 calls are averaged to present the Average Response Time (ART) for that minute. When you hover over a point on the chart, the Count figure represents how many times the metric data was measured during that minute.

In this graph, response time was measured 140 times between 2:57 and 2:58 PM and the ART for that minute was 889 ms.



In this chart, the time range was set for the last three hours, so the interval is one minute. The errors per minute value displayed at the 1:34 PM point represents the number of errors recorded during the minute from 1:34 PM to the millisecond just before 1:35 PM.



Ten-Minute Resolution

Metric data is displayed at a 10-minute resolution, and each point on the graph represents a roll-up of the values measured during that 10-minute period.

The ART for the rolled-up time period represents the true average. The ART aggregates the sum and count of the total requests for the period; it does not average the ARTs calculated for the previous roll-up granularity.

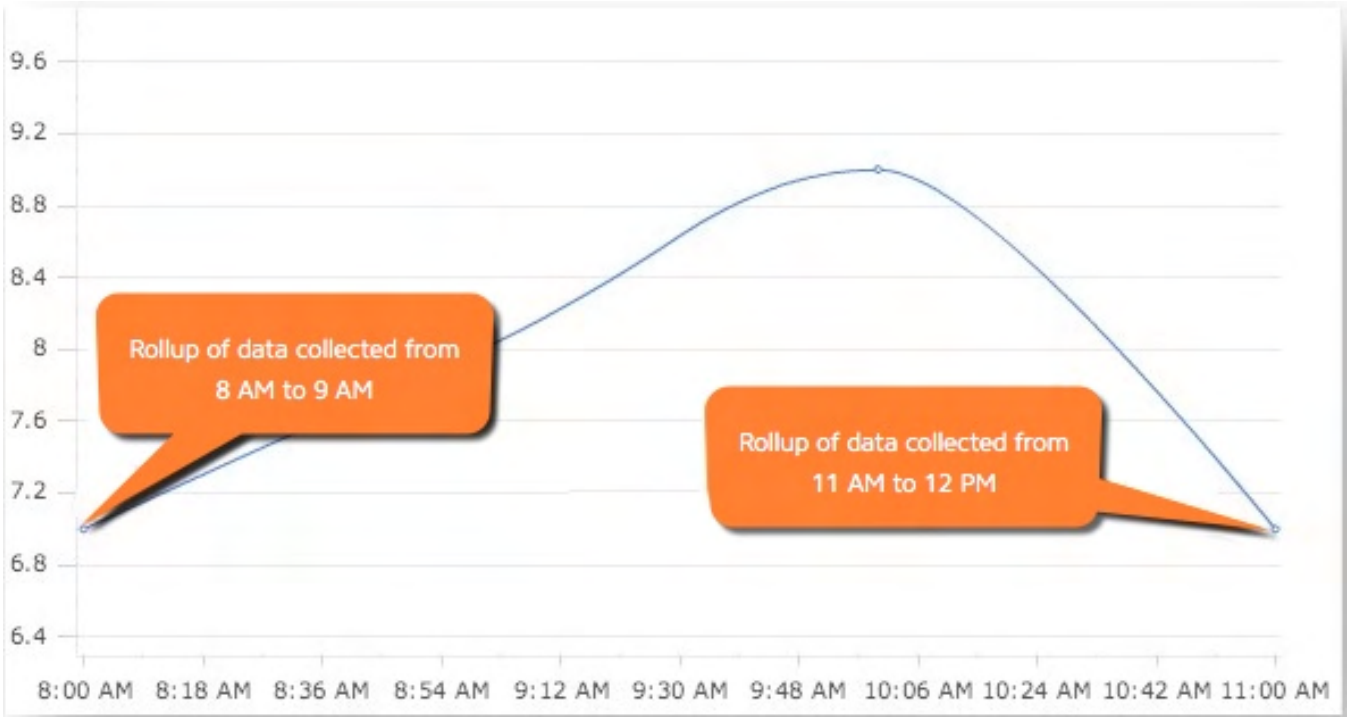
One-Hour Resolution

Metric data is displayed at a 1-hour resolution, and each point on the graph represents a roll-up of the values measured during that 60-minute period.

How the Last Interval Displays in a Chart

Because of the way rolled-up data displays, it may appear as if data is not included for the last increment in the time range. However, the last data point represents the last interval in the range.

For example, when you set a time range from 8 AM to 12 PM for a day that has been rolled up into 1-hour data points, you might expect to see five data points, one for 8 AM, 9 AM, 10 AM, 11 AM, and 12 PM. However, the returned data consists of four data points for the hours 8 AM, 9 AM, 10 AM, and 11 AM. The first data point, 8:00 AM, represents data collected from 8 AM to the millisecond just before 9 AM. The last data point, 11:00 AM, represents data collected from 11 AM to the millisecond just before 12 PM. So you are actually seeing data from 8 AM to 12 PM, even though the last data point in the graph is 11 AM.



View Charts for Long Time Ranges

For time ranges that are greater than three weeks, graphs in the UI such as the Slow Transaction graph show the 14th of each month as the time point on the x-axis. For a time range that is less than two full months, this means that the graph may only have a single point, the middle of the month encompassed by the time range.

View Details about Older Data

To review details for an issue that occurred during a period for which you have only 10-minute or 1-hour data, AppDynamics provides access to diagnostic data through Transaction Snapshots. For example, suppose you were seeing values for 2 AM and 3 AM three days ago, but you need to examine details about a problem you were alerted about that occurred at 2:15 AM that morning. You view Transaction Snapshots for the hour between 2 AM and 3 AM to see details about what happened during that time period. By default, snapshots are retained for two weeks, and individual snapshots can be archived. See Transaction Snapshots.

If you are viewing a graph in the Metric Browser, you can select a specific period in the chart and drill down to see Transaction Snapshots and other information. See Analyzing Performance for a Specific Time Range.

Percentile Metrics

This page describes percentile metrics and how to modify default percentile collection.

A percentile is a value below which a given percentage of measurements in a set falls. For example, a 95th percentile value of 150 ms means that 95% of all values are 150 ms or less. See Quick Tour of Percentile Metrics for Java.

Disable Percentile Metric Collection

The Java Agent and the .NET Agent capture percentile metrics by default. You can disable percentile metric collection in the Configure Percentile Metrics panel on the **Configuration > Slow Transaction Thresholds** page. Alternatively, manually setting the disable-percentile-metrics node property to `True` prevents the agent from collecting percentile metrics.

Specify Percentiles to Collect

By default, app agents capture the 95th percentile. You can indicate 5 whole numbers between 1 and 99 as Percentiles to Collect on the **Configuration > Slow Transaction Thresholds** page. You can apply the configuration changes to all existing business transactions or only to new transactions discovered after the configuration change.

Modify the Percentile Metric Algorithm

The agent uses one of the following methods to calculate percentile metrics:

- P Square algorithm (default): This option consumes the least amount of storage and incurs the least amount of CPU overhead. The accuracy of the percentile calculated varies depending on the nature of the distribution of the response times. You should use this option unless you doubt the accuracy of the percentiles presented.
- Quantile Digest algorithm: This option consumes slightly more storage and CPU overhead for the machine where the agent is running, but may offer better percentiles depending on how the response times are distributed.

To adjust the algorithm option, set the App Agent Node Property to percentile-method-option.

Monitor Events

Related Pages:

- Create a Custom Event
- Transaction Thresholds
- Virtual War Rooms

This page describes event monitoring in AppDynamics.

In the AppDynamics model, events represent a change in the state of a monitored application. An event can represent an error/exception generated by the application, the crossing of a performance threshold, or an operational change in the application (such as a JVM restart).

You can use events to investigate application issues underlying performance problems. You can define policies that generate notifications or perform actions when an event occurs. See Alert and Respond.

View and Monitor Events

You can view and analyze events in the Controller UI. From the Database Visibility, Server Visibility, or user experience application pages in the UI, you can view events by clicking **Events** from the left navigation tree.

> The Controller displays a subset of the types of events generated in AppDynamics. You can access additional event types through the REST API. See AppDynamics APIs.

To view events for application monitoring:

1. Navigate to **Events** on any business application, business transaction, tier or node dashboard to view events generated in the selected time frame.
2. Click **Filters** and select the type of event to display.
3. Review information about an event
   If the event is an application change, such as an application restart, the details contain a description of the event.
   If the event is a slow transaction or an error, the details are transaction snapshots from which you can drill down to the root cause of the problem.
4. From the **Event Details** dialog, right-click an event to start a war room, delete or archive the event, or test action execution.

Archive Events

An event is purged after two weeks, at which point it is no longer accessible. To save an event beyond the default snapshot lifespan for future analysis, you can archive the event.

Event Purging in Ephemeral Environments

Events associated with nodes that are deployed to machine instances in an ephemeral environment are subject to being purged. This occurs because the node associated with the events is identified by a label consisting of two discrete parts: a node name and a machine instance name. The node name is always fixed, but the machine instance name changes in ephemeral environments. Thus, when a node is redeployed to a new machine instance, the label identifying the node changes.

For example, the label for a node in an ephemeral environment might consist of the fixed node name `"Node1"` and the machine instance name `"INSTANCE-M-T1": NodeINSTANCE-M-T1`. When the node is redeployed to another machine instance, the label for the node could change because of the new machine instance to something such as "Node1INSTANCE-M-T2". If this occurs, the events associated with the first label `"NodeINSTANCE-M-T1"` could potentially be purged later.

Events Archived by Default

AppDynamics archives application configuration change events by default. In the Controller UI, these events display as **Application Configuration Change**. You can view the `APPLICATION_CONFIG_CHANGE` event type in the Controller UI or retrieve it with the AppDynamics REST APIs.

> Once `APPLICATION_CONFIG_CHANGE` events are archived or deleted, you can no longer retrieve them through the AppDynamics REST APIs.

Manually Archive Events

To manually archive events:

1. Select the event from the events list.
2. Click **Actions > Archive Event**.
3. Click **Archive** to confirm the action.

This indicates that an event has been archived. You can click the column heading to sort by archived events.

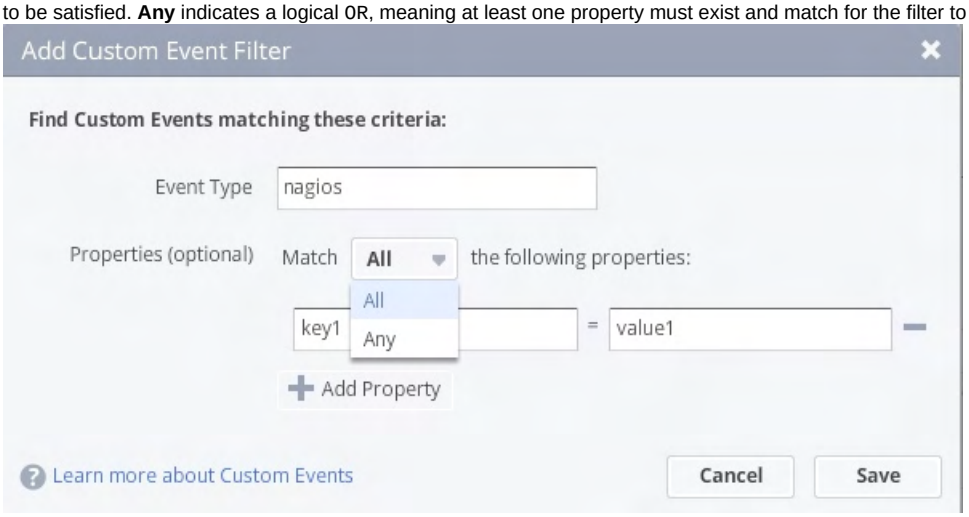
Modify Event Retention Period (On-Premises Controllers)

For on-premises Controllers, administrators can modify the default two-week period by configuring the events.retention.period property in **Administration > Controller Settings**.

Filter Custom Events

The AppDynamics REST API enables you to define custom events. You can then filter on those events by creating a custom filter:

1. Open the filter options on the **Events** page.
2. Navigate to **Filter by Custom Events** and click **Add Property** (**+**).
3. In the **Add Custom Event Filter** dialog, enter the name of the custom event type.
4. Optionally, specify filtering event properties as key/value pairs. The custom event needs to contain the property values you specify to satisfy the filter. If you add multiple properties, a matching value of **All** applies an `AND` operator to the list, meaning all properties need to match for the filter to be satisfied. **Any** indicates a logical `OR`, meaning at least one property must exist and match for the filter to be satisfied.



Monitor Application Change Events

Related Pages:

- Policies
- Monitor Events

This page describes monitoring application change events.

When investigating an application error or poor performance, you may want to see whether any application changes occurred at the same time as the error or issue. AppDynamics automatically detects and reports common application change events, such as app server restarts or changes to application environment variables. You can supplement automatically detected change events with custom application change events generated through the Controller or the REST API.

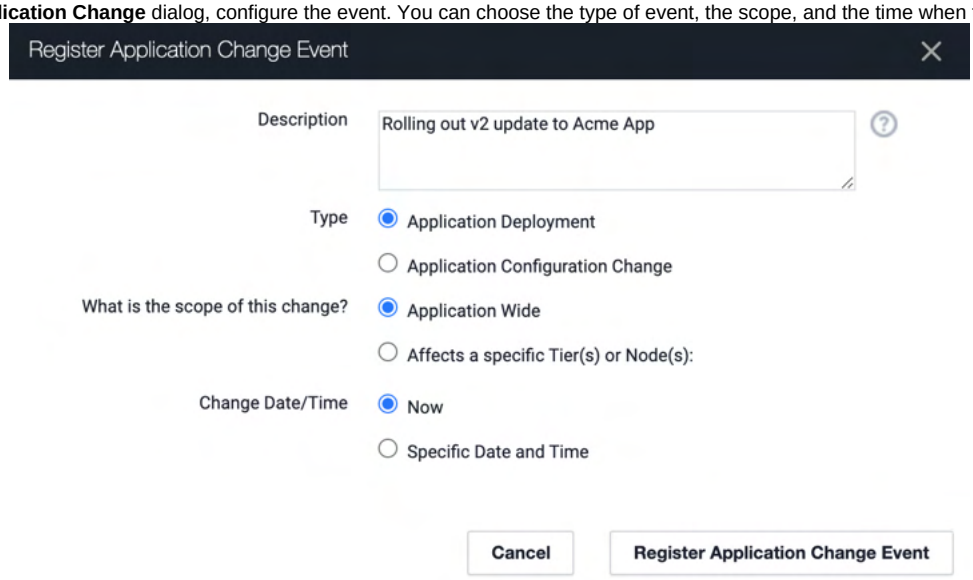
Application changes appear alongside other events in the Events List. Similar to other types of events, you can view details for the event by double-clicking the event or view the details in the context of other performance indicators. They can also serve as triggering conditions for policies.

Manually Register Application Changes

You can manually add an application change event to AppDynamics. This capability enables you to log a change event which might otherwise not be detected by AppDynamics. Controller users can then correlate the event to performance data. For example, you may want to create an event when you push an updated version of the application.

Create an Application Change Event Manually

1. From the **Events** page, expand the **Actions** menu and choose **Register Application Change Event**.
2. In the **Register Application Change** dialog, configure the event. You can choose the type of event, the scope, and the time when the event



should be registered.

3. Click **Register Application Change Event**. If **Now** is selected as the change time, the event appears in the events list immediately.
4. Double-click the event in the list to view more details about the event, including any actions triggered by the event as a result of a policy.

Automatically Register Application Changes

The AppDynamics REST API provides an alternative to creating application change events in the UI.

The API lets you automate event registration so that external systems can generate events based on their activities or conditions. For example, you can configure your release management system to generate an event in AppDynamics when it deploys an update to the application code, allowing you to correlate that event with application performance.

For information on creating a custom event through the REST API, see the Create a Custom Event method in AppDynamics APIs.

Correlate Application Changes with Other Events

You can view application changes in the context with other events and performance indicators in various Controller UI pages.

View Application Changes with Other Events in Transaction Score Histogram

1. In the Application Dashboard, click the Transaction Score tab.

2. Scroll down to the **Events** histogram.
3. Click **Add Criteria**.
4. From the dropdown, select **Event Types**.
5. Unselect all of the criteria except **Application Changes**.

Application change events appear in the **Events** histogram, where they appear in the context of other performance indicators.

Events Reference

Related pages:

- Monitor Events
- AppDynamics APIs

This page lists these types of events generated in AppDynamics:

- Event types that are common to the platform, such as UI, configuration, and licensing-related events
- Events generated by the alert and response system
- Application monitoring events

Information on event types that are specific to a platform module, if available, appears with the module-specific documentation. For example, Database Agent events are listed on Database Agent Events Reference.

Event types listed here that are not visible in the UI can be retrieved using the AppDynamics REST API.

ACTIVITY_TRACE

Description: The agent sent an internal event containing activity traces.
Activity traces enable the tracing of a code path that passes through a specified class/method. The App Agent for Java uses them to provide object instance tracking (OIT) and automatic leak detection (ALD)

Category: AppDynamics Data

Visible in UI: No

Learn More: Java Memory Leaks, Object Instance Tracking for Java

ADJUDICATION_CANCELLED

Description: A designated approver has canceled a thread dump or remediation action that was triggered by a policy.

Visible in UI: No

Learn More: Actions Requiring Approval, Policies

AGENT_ADD_BLACKLIST_REG_LIMIT_REACHED

Description: The agent Application Diagnostic Data (ADD) blacklist registration limit has been reached.

Visible in UI: No

Learn More: Customize System Notifications

AGENT_ASYNC_ADD_REG_LIMIT_REACHED

Description: The agent async Application Diagnostic Data (ADD) registration limit has been reached.

Visible in UI: No

AGENT_CONFIGURATION_ERROR

Description: An agent configuration error has been detected.

Category: AppDynamics Configuration Warnings

UI Display name: Agent Configuration Error

Visible in UI: Yes

Learn More: Troubleshoot .NET Agent Issues

APPLICATION_CRASH

Description: A crash has been detected for a JVM. The crash log file is updated.

Category: Application Changes

UI Display name: Application Crash

Visible in UI: Yes

AGENT_DIAGNOSTICS

Description: Diagnostic information concerning agent activity, such as business transaction overflow or HTTP error code diagnostics, has been sent.

Category: AppDynamics Data

UI Display name: Agent Diagnostics

Visible in UI: No

AGENT_ERROR_ADD_REG_LIMIT_REACHED

Description: The agent error Application Diagnostic Data (ADD) registration limit has been reached.

Visible in UI: No

AGENT_EVENT

Description: Generic internal event. This event type is also used for all Database Agent events with a case-specific message attached.

Category: AppDynamics Internal Diagnostics

UI Display name: Agent Event

Visible in UI: No

Learn More: Database Agent Events Reference

AGENT_METRIC_BLACKLIST_REG_LIMIT_REACHED

Description: The agent metric blacklist registration limit has been reached.

Visible in UI: No

Learn More: Metrics Limits, Customize System Notifications

AGENT_METRIC_REG_LIMIT_REACHED

Description: The agent metric registration limit has been reached.

Visible in UI: No

Learn More: Metrics Limits, Customize System Notifications

AGENT_STATUS

Description: The agent has been enabled or disabled.

Category: AppDynamics Internal Diagnostics

UI Display name: Agent Enabled / Disabled

Visible in UI: No

Learn More: Manage App Agents

ALREADY_ADJUDICATED

Description: A designated approver has previously approved or canceled a thread dump or remediation action that was triggered by a policy.

Visible in UI: No

Learn More: Actions Requiring Approval, Policies

APPDYNAMICS_DATA

Description: Events used to send data from the agent to the Controller, then to the UI. Events such as BTOverflowDetails and MemoryLeakDiagnostics should be moved into this bucket.

Category: AppDynamics Data

Visible in UI: No

APPDYNAMICS_INTERNAL_DIAGNOSTICS

Description: Internal diagnostics events.

Category: AppDynamics Internal Diagnostics

Visible in UI: No

APPLICATION_CONFIG_CHANGE

Description: Application configuration has been changed interactively by the user or through the REST API.

Category: Application Changes

UI Display name: Application Configuration Change

Visible in UI: Yes

> This event type is automatically archived.

APPLICATION_DEPLOYMENT

Description: An application has been deployed.

Category: Application Changes

UI Display name: Application Deployment

Visible in UI: Yes

APPLICATION_DISCOVERED

Description: A new application has been added to the Controller.

Category: Discovery

UI Display name: New Application Discovered

Visible in UI: Yes

Learn More: Policies

APPLICATION_ERROR

Description: An application error has been detected.

Category: Errors

UI Display name: Application Server Exception

Visible in UI: Yes

APP_SERVER_RESTART

Description: An application server has been restarted.

Category: Application Changes

UI Display name: App Server Restart

Visible in UI: Yes

AZURE_AUTO_SCALING

Description: An internal event that reports Azure auto-scaling progress to the UI. Seen at the bottom of the Azure auto-scaling screen.

Category: AppDynamics Data

Visible in UI: No

BACKEND_DISCOVERED

Description: A new backend has been added to the application.

Category: Discovery

UI Display name: New Backend Discovered

Visible in UI: Yes

Learn More: Policies

BT_DISCOVERED

Description: A new business transaction has been added to the application.

Category: Discovery

UI Display name: New Business Transaction Discovered

Visible in UI: Yes

Learn More: Policies

BUSINESS_ERROR

Description: A business error has been detected.

Category: Errors

UI Display name: Business Error

Visible in UI: Yes

CLR_CRASH

Description: A CLR crash has occurred.

Category: AD Infrastructure events

UI Display name: CLR Crash

Visible in UI: Yes

CONTROLLER_AGENT_VERSION_INCOMPATIBILITY

Description: The agent version is newer than the Controller version.

Category: AppDynamics Configuration Warnings

UI Display name: Agent Version is newer than Controller version

Visible in UI: Yes

Learn More: Agent and Controller Compatibility, Install the Controller

CONTROLLER_ASYNC_ADD_REG_LIMIT_REACHED

Description: The Controller limit for registering async Application Diagnostic Data (ADDs) for this account has been reached.

Visible in UI: No

Learn More: Customize System Notifications

CONTROLLER_COLLECTIONS_ADD_REG_LIMIT_REACHED

Description: The Controller COLLECTIONS ADD registration limit for the account has been reached.

Visible in UI: No

Learn More: Customize System Notifications

CONTROLLER_ERROR_ADD_REG_LIMIT_REACHED

Description: The limit for registering error Application Diagnostic Data (ADDs) for the account has been reached.

Visible in UI: No

Learn More: Customize System Notifications

CONTROLLER_EVENT_UPLOAD_LIMIT_REACHED

Description: The limit on the number of events per minute that can be uploaded to the controller for the account has been reached.

Visible in UI: No

Learn More: Customize System Notifications

CONTROLLER_MEMORY_ADD_REG_LIMIT_REACHED

Description: The Controller MEMORY ADD registration limit for the account has been reached.

Visible in UI: No

Learn More: Metrics Limits, Customize System Notifications

CONTROLLER_METADATA_REGISTRATION_LIMIT_REACHED

Description: The Controller metadata registration limit for the account has been reached.

Visible in UI: No

Learn More: Business Transactions, Organize Business Transactions, Customize System Notifications

CONTROLLER_METRIC_DATA_BUFFER_OVERFLOW

Description: The Controller metric data buffer has overflown. Now dropping metric data.

Learn More: Metrics Limits, Customize System Notifications

CONTROLLER_METRIC_REG_LIMIT_REACHED

Description: The limit for registering metrics for the account has been reached.

Visible in UI: No

Learn More: Metrics Limits, Customize System Notifications

CONTROLLER_PSD_UPLOAD_LIMIT_REACHED

Description: The PSD limit for the account has been reached.

Visible in UI: No

Learn More: Customize System Notifications

CONTROLLER_RSD_UPLOAD_LIMIT_REACHED

Description: The request segment data (RSD) limit for the account has been reached.

Visible in UI: No

Learn More: Customize System Notifications

CONTROLLER_SEP_ADD_REG_LIMIT_REACHED

Description: The limit for the Controller SERVICE_ENDPOINT ADD registration for the account has been reached.

Visible in UI: No

Learn More: Customize System Notifications

CONTROLLER_STACKTRACE_ADD_REG_LIMIT_REACHED

Description: The limit for registering StackTrace ADDs for the account has been reached.

Visible in UI: No

Learn More: Customize System Notifications

CONTROLLER_TRACKED_OBJECT_ADD_REG_LIMIT_REACHED

Description: The Controller TRACKED_OBJECT ADD registration limit for the account has been reached.

Visible in UI: No

Learn More: Customize System Notifications

CUSTOM

Description: These are custom events thrown by REST API calls or Machine Agent API calls.

Category: Custom Event

UI Display name: Depends on the event.

Visible in UI: Depends on the event.

Learn More: Create Events

CUSTOM_ACTION_END

Description: A custom action has ended.

Visible in UI: No

Learn More: Custom Actions, Policies

CUSTOM_ACTION_FAILED

Description: A custom action has failed.

Visible in UI: No

Learn More: Custom Actions, Policies

CUSTOM_ACTION_STARTED

Description: A custom action has started.

Visible in UI: No

Learn More: Custom Actions, Policies

CUSTOM_EMAIL_ACTION_END

Description: A custom email action ended.

Visible in UI: No

Learn More: Diagnostic Actions, Policies

CUSTOM_EMAIL_ACTION_FAILED

Description: A custom email action failed.

Visible in UI: No

Learn More: Diagnostic Actions, Policies

CUSTOM_EMAIL_ACTION_STARTED

Description: A custom email action started.

Visible in UI: No

Learn More: Diagnostic Actions, Policies

DB_SERVER_PARAMETER_CHANGE

Description: The DBMS server parameters have been changed.

Category: DB Agent Event

DEADLOCK

Description: The agent has detected code deadlock.

Category: Code Problems

UI Display name: Code Deadlock

Visible in UI: Yes

Learn More: Code Deadlocks for Java

DEV_MODE_CONFIG_UPDATE

Description: The Dev Mode Config has been updated. This is fired from the agent and the Controller.

DIAGNOSTIC_SESSION

Description: A diagnostic session has started.

Category: AppDynamics Internal Diagnostics

UI Display name: Diagnostic Session

Visible in UI: No

Learn More: Diagnostic Sessions

DISK_SPACE

Description: The controller is running out of disk space.

Category: AppDynamics Configuration Warnings

UI Display name: Controller Disk Space Low

Visible in UI: Yes

Learn More: Controller System Requirements, Controller Disk Space and the Database, Database Size and Data Retention

EMAIL_ACTION_FAILED

Description: Email action has failed.

Learn More: Notification Actions

EMAIL_SENT

Description: Email was sent to notify the recipient of an event.

Visible in UI: No

Learn More: Notification Actions

EUM_CLOUD_BROWSER_EVENT

Description: A browser snapshot was stored in the database.

Visible in UI: No

Learn More: End User Monitoring, Browser Snapshots

EUM_CLOUD_SYNTHETIC_BROWSER_EVENT

Description: A synthetic browser snapshot was stored in the database.

Visible in UI: No

Learn More: End User Monitoring, Browser Snapshots

EUM_CLOUD_SYNTHETIC_ERROR_EVENT

Description: A synthetic error event will occur on the first occurrence of an error session for a synthetic job.

Visible in UI: No

Learn More: End User Monitoring

EUM_CLOUD_SYNTHETIC_CONFIRMED_ERROR_EVENT

Description: A synthetic error event is confirmed. When we see an error session again when we retry the job.

Visible in UI: No

Learn More: End User Monitoring

EUM_CLOUD_SYNTHETIC_ONGOING_ERROR_EVENT

Description: A synthetic ongoing error event was detected when the error was confirmed. Sessions will be in error states.

Visible in UI: No

Learn More: End User Monitoring

EUM_CLOUD_SYNTHETIC_HEALTHY_EVENT

Description: A synthetic event was reported healthy when a job is transited from any other state to a healthy state.

Visible in UI: No

Learn More: End User Monitoring

EUM_CLOUD_SYNTHETIC_WARNING_EVENT

Description: When a synthetic job session status is WARNING, the detected warning is reported.

Visible in UI: No

Learn More: End User Monitoring.

EUM_CLOUD_SYNTHETIC_CONFIRMED_WARNING_EVENT

Description: A synthetic warning was confirmed. When we see a warning session again when we retry the job.

Visible in UI: No

Learn More: End User Monitoring

EUM_CLOUD_SYNTHETIC_ONGOING_WARNING_EVENT

Description: A synthetic ongoing warning detection is reported when the warning is confirmed.

Visible in UI: No

Learn More: End User Monitoring

EUM_CLOUD_SYNTHETIC_PERF_WARNING_EVENT

Description: A synthetic ongoing warning detection is reported when performance breaches/exceeds the threshold.

Visible in UI: No

Learn More: End User Monitoring

EUM_CLOUD_SYNTHETIC_PERF_CONFIRMED_WARNING_EVENT

Description: A synthetic performance warning detection is reported when we see a warning session again and perform a retest or see consecutive failures of the job (This depends on the configurations).

Visible in UI: No

Learn More: End User Monitoring

EUM_CLOUD_SYNTHETIC_PERF_ONGOING_WARNING_EVENT

Description: A synthetic ongoing warning event is reported and confirmed.

Visible in UI: No

Learn More: End User Monitoring

EUM_CLOUD_SYNTHETIC_PERF_HEALTHY_EVENT

Description: A synthetic healthy performance event was detected and is reported when a job is transited from any other state to a healthy state.

Visible in UI: No

Learn More: End User Monitoring

EUM_CLOUD_SYNTHETIC_PERF_CRITICAL_EVENT

Description: A synthetic critical performance event was detected and is reported when performance for critical event breaches/exceeds the threshold.

Visible in UI: No

Learn More: End User Monitoring

EUM_CLOUD_SYNTHETIC_PERF_CONFIRMED_CRITICAL_EVENT

Description: A synthetic critical performance event was detected and is reported when we see a critical session again and retest or see consecutive failures of the job (This depends on the configurations).

Visible in UI: No

Learn More: End User Monitoring

EUM_CLOUD_SYNTHETIC_PERF_ONGOING_CRITICAL_EVENT

Description: A synthetic ongoing critical event was detected and the event is confirmed.

Visible in UI: No

Learn More: End User Monitoring

EUM_INTERNAL_ERROR

Description: An internal EUM error has occurred.

Visible in UI: No

Learn More: End User Monitoring

HTTP_REQUEST_ACTION_END

Description: An HTTP request action ended.

Visible in UI: No

Learn More: Diagnostic Actions, Policies

HTTP_REQUEST_ACTION_FAILED

Description: An HTTP request action failed.

Visible in UI: No

Learn More: Diagnostic Actions, Policies

HTTP_REQUEST_ACTION_STARTED

Description: An HTTP request action started.

Visible in UI: No

Learn More: Diagnostic Actions, Policies

INFO_INSTRUMENTATION_VISIBILITY

Description: Information was written to the Bytecode Transformer Log. This log contains information associated with the AppDynamics bytecode instrumentation (BCI) engine.

Category: AppDynamics Internal Diagnostics

UI Display name: Bytecode Transformer Log

Visible in UI: No

Learn More: Request Agent Log Files, App Agent Node Properties Reference

INTERNAL_UI_EVENT

Description: These are the XResponder.handleGeneralServerFaultEvent events.

Category: AppDynamics Internal Diagnostics

UI Display name: Controller API Call Threw an Exception

Visible in UI: No

KUBERNETES

Description: The two Kubernetes event types, normal events and warning events, thrown by the Cluster Agent.

Category: Kubernetes Events

UI Display name: Cluster Event

Visible in UI: Yes

Learn More: Monitor Kubernetes Events

LICENSE

Description: The AppDynamics license has expired.

Category: AppDynamics Configuration Warnings

UI Display name: License Expired

Visible in UI: Yes

Learn More: License Information

MACHINE_AGENT_LOG

Description: The Machine Agent is now logging information.

Category: AppDynamics Data

MACHINE_DISCOVERED

Description: A new machine has been added to the application.

Category: Discovery

UI Display name: New Machine Discovered

Visible in UI: Yes

Learn More: Policies

MEMORY

Description: Events for automatic leak detection and custom memory structures.

Category: AppDynamics Data

UI Display name: AppDynamics Data

Visible in UI: No

MEMORY_LEAK_DIAGNOSTICS

Description: The agent sends this internal event with memory leak data. The UI uses this on the memory monitoring screens in the node dashboards.

Visible in UI: No

MOBILE_CRASH_IOS_EVENT

Description: An iOS mobile application crash has arrived at the Controller.

Visible in UI: No

Learn More: Crashes

MOBILE_CRASH_ANDROID_EVENT

Description: An Android mobile application crash has arrived at the Controller.

Visible in UI: No

Learn More: Crashes

MOBILE_NEW_CRASH_EVENT, SLOW, VERY_SLOW, STALL

Description: An Android mobile application crash has arrived at the Controller with slow, very slow, or stalled status.

Visible in UI: No

Learn More: Crashes

NETWORK

Description: The log data provided by the NPM Agent has been logged by the NPM Dynamic service.

UI Display name: Network

Visible in UI: Yes

NODE_DISCOVERED

Description: A new node has been added to the application.

Category: Discovery

UI Display name: New Node Discovered

Visible in UI: Yes

Learn More: Policies

NORMAL

Description: A business transaction is normal (not slow, very slow or stalled).

Visible in UI: No

Learn More: Transaction Thresholds, Dynamic Baselines

OBJECT_CONTENT_SUMMARY

Description: The agent sent an internal event with object content summary for collections, caches, etc.

Category: AppDynamics Data

Visible in UI: No

POLICY_CANCELED_CRITICAL

Description: A health rule violation, based on a status of critical, was canceled.

Category: Policy Violations

UI Display name: Health Rule Violation Canceled - Critical

Visible in UI: Yes

Learn More: Health Rules, Policies

POLICY_CANCELED_WARNING

Description: A health rule violation, based on a status of warning, was canceled.

Category: Policy Violations

UI Display name: Health Rule Violation Canceled - Warning

Visible in UI: Yes

Learn More: Health Rules, Policies

POLICY_CLOSE_CRITICAL

Description: A health rule violation, based on a status of critical, ended.

Category: Policy Violations

UI Display name: Health Rule Violation Ended - Critical

Visible in UI: Yes

Learn More: Health Rules, Policies

POLICY_CLOSE_WARNING

Description: A health rule violation, based on a status of warning, ended.

Category: Policy Violations

UI Display name: Health Rule Violation Ended - Warning

Visible in UI: Yes

Learn More: Health Rules, Policies

POLICY_CONTINUES_CRITICAL

Description: After the initial POLICY_OPEN_CRITICAL event, an event generated to indicate the continuation of the health rule violation at the critical level.

Category: Policy Violations

UI Display name: Health Rule Violation Continues - Critical

Visible in UI: Yes

Learn More: Health Rules, Policies

POLICY_CONTINUES_WARNING

Description: After the initial POLICY_OPEN_WARNING event, an event generated to indicate the continuation of the health rule violation at the warning level.

Category: Policy Violations

UI Display name: Health Rule Violation Continues - Warning

Visible in UI: Yes

Learn More: Health Rules, Policies

POLICY_DOWNGRADED

Description: A health rule violation was downgraded from critical to warning.

Category: Policy Violations

UI Display name: Health Rule Violation Downgraded - Critical to Warning

Visible in UI: Yes

Learn More: Health Rules, Policies

POLICY_OPEN_CRITICAL

Description: A critical health rule was violated.

Category: Policy Violations

UI Display name: Health Rule Violation Started - Critical

Visible in UI: Yes

Learn More: Health Rules, Policies

POLICY_OPEN_WARNING

Description: A warning health rule was violated.

Category: Policy Violations

UI Display name: Health Rule Violation Started - Warning

Visible in UI: Yes

Learn More: Health Rules, Policies

POLICY_UPGRADED

Description: A health rule violation was upgraded from warning to critical.

Category: Policy Violations

UI Display name: Health Rule Violation Upgraded - Warning to Critical

Visible in UI: Yes

Learn More: Health Rules, Policies

RESOURCE_POOL_LIMIT

Description: A resource pool limit, such as a thread pool or connection pool, has been reached.

Category: Code Problems

UI Display name: Resource Pool Limit Reached

Visible in UI: Yes

Learn More: Remediation Actions, Policies

RUNBOOK_DIAGNOSTIC SESSION_END

Description: A diagnostic session that was started by a diagnostic action triggered by a policy, has ended.

Visible in UI: No

Learn More: Diagnostic Sessions, Diagnostic Actions, Policies

RUNBOOK_DIAGNOSTIC SESSION_FAILED

Description: A diagnostic session that was started by a diagnostic action triggered by a policy failure.

Visible in UI: No

Learn More: Diagnostic Sessions, Diagnostic Actions, Policies

RUNBOOK_DIAGNOSTIC SESSION_STARTED

Description: A diagnostic session that was started by a diagnostic action triggered by a policy, session started.

Visible in UI: No

Learn More: Diagnostic Sessions, Diagnostic Actions, Policies

RUN_LOCAL_SCRIPT_ACTION_END

Description: A local script that was started by a remediation action triggered by a policy, has ended.

Visible in UI: No

Learn More: Remediation Actions, Policies

RUN_LOCAL_SCRIPT_ACTION_FAILED

Description: A local script that was started by a remediation action triggered by a policy, has failed.

Visible in UI: No

Learn More: Remediation Actions, Policies

RUN_LOCAL_SCRIPT_ACTION_STARTED

Description: A local script that was started by a remediation action triggered by a policy, has started.

Visible in UI: No

Learn More: Remediation Actions, Policies

SERVICE_ENDPOINT_DISCOVERED

Description: A new service endpoint has been added to the application.

Category: Discovery

UI Display name: New Service Endpoint Discovered

Visible in UI: Yes

Learn More: Policies

SLOW

Description: A business transaction is now slow.

Category: Slow Transactions

UI Display name: Slow Transactions

Visible in UI: Yes

Learn More: Troubleshoot Slow Response Times, Transaction Thresholds

SMS_SENT

Description: An SMS was sent to notify the recipient of an event.

Visible in UI: No

Learn More: Notification Actions

STALL

Description: A business transaction has stalled.

Category: Slow Transactions

UI Display name: Transaction Stall

Visible in UI: Yes

Learn More: Troubleshoot Slow Response Times, Transaction Thresholds

SYSTEM_LOG

Description: Thrown when events occur during workflow execution.

Category: AppDynamics Data

UI Display name: Automation Event

Visible in UI: Yes

THREAD_DUMP_ACTION_END

Description: A thread dump action ended.

Visible in UI: No

Learn More: Diagnostic Actions, Policies

THREAD_DUMP_ACTION_FAILED

Description: A thread dump action failed.

Visible in UI: No

Learn More: Diagnostic Actions, Policies

THREAD_DUMP_ACTION_STARTED

Description: A thread dump action started.

Visible in UI: No

Learn More: Diagnostic Actions, Policies

TIER_DISCOVERED

Description: A new tier has been added to the application.

Category: Discovery

UI Display name: New Tier Discovered

Visible in UI: Yes

Learn More: Policies

VERY_SLOW

Description: A business transaction is now very slow.

Category: Slow Transactions

UI Display name: Very Slow Transactions

Visible in UI: Yes

Learn More: Troubleshoot Slow Response Times, Transaction Thresholds

WARROOM_NOTE

Description: A War Room Note has been added.

Learn More: Virtual War Rooms

Monitor Infrastructure

This page describes infrastructure monitoring in AppDynamics.

While business transaction performance is the main focus of a performance monitoring strategy, infrastructure performance provides additional insight into the underlying factors of business transaction performance. Infrastructure Visibility provides visibility into the underlying OS infrastructure and networks on which your applications run. These agents enable you to isolate, identify, and troubleshoot infrastructure problems that can affect application performance such as resource-hogging tiers.

AppDynamics provides preconfigured application infrastructure metrics and default health rules to enable you to discover and correct infrastructure problems. You can also configure additional persistent metrics to implement a monitoring strategy for your application architecture and business needs.

Additionally, you can view infrastructure metrics in the Metric Browser. **Correlation Analysis** and **Scalability Analysis** charts in the browser are useful for understanding how infrastructure metrics correlate to business transaction performance.

See Overview of Infrastructure Visibility.

Alert and Respond

Related Pages:

- Create and Manage Custom Roles
- Example Use Cases

This page describes the automated alert and response capabilities in AppDynamics.

Alert and Respond in AppDynamics

AppDynamics can generate notifications or invoke actions based on conditions or events you configure. Using the alert and respond feature, you can determine problems as they occur, or even before they occur.

Policies serve as central configuration artifacts for the alert and respond feature. A policy allows you to define the remedial actions that are invoked when one or more conditions are met or an event occurs. A health rule defines the performance condition or an event.

AppDynamics provides several pre-configured health rules that give you a head start to create your own health rules. You can also use built-in examples and customize them to match your application performance requirements. For example, the built-in health rules test for whether the *Business Transaction error rate is much higher than normal* or *CLR Garbage Collection Time is too high.* See Default Health Rules for more examples.

Actions automate the response to an event, such as, sending an alert, or performing diagnostic, or performing remediation actions. See Alert and Respond API to learn how to create custom URLs for notifications.

> Notification actions that use email or SMS and email digests require that the SMTP server be configured for the Controller. See Enable an Email Server.

Email digests generate email messages about the conditions and events in a system based on a pre-defined schedule.

Permissions

Different users with different roles typically set up and use the various alert and respond features. Some permissions can be granted at the account level and some can be granted at the application or even the tier level.

Account Level Permissions

Alerting templates, Email templates, HTTP request templates, and Email/SMS configuration are account-level features. The scope for these features is the entire AppDynamics account and all the applications in that account. Users who have account-level roles that include the necessary permissions can create and manage account-level templates and configuration. These permissions belong to the predefined Account Owner role. The Account Owner can also create custom roles that include some of these permissions. For example, an account owner can create an email template manager role and assign that role to users, who can then create and modify email templates.

To create, manage, and configure these account-level features, you need the following permissions respectively:

- Configure Alerting Templates
- Configure Email Templates
- Configure HTTP Request Templates
- Configure Email / SMS

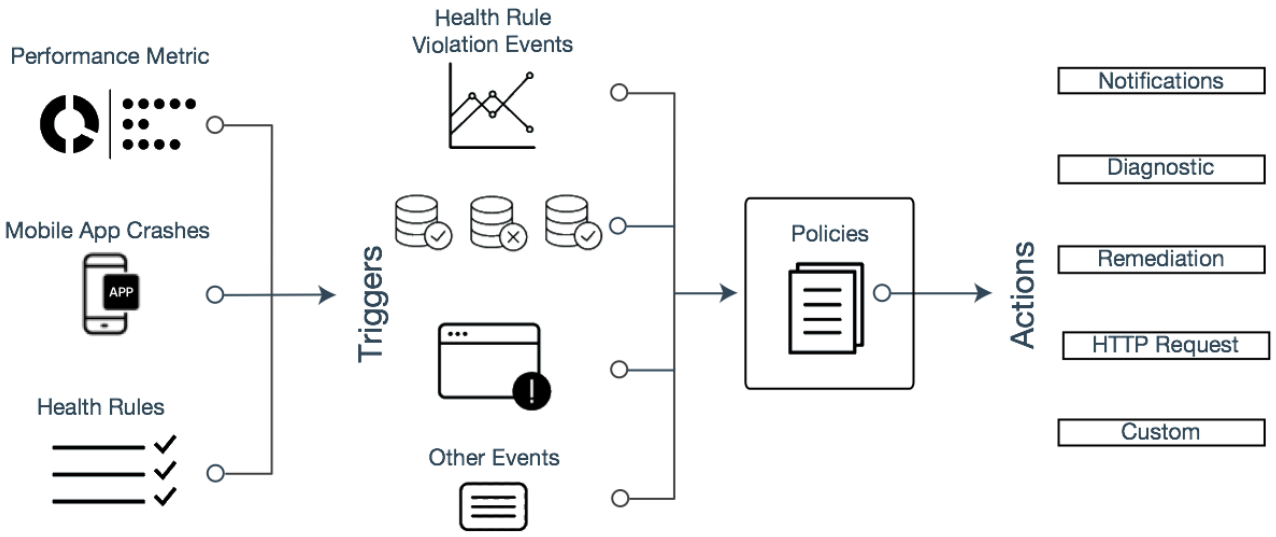
Application and Tier-Level Permissions

Policies, health rules, actions, and email digests are application-level or tier-level features. The scope of these features is the application or tier in which they were created. To create, manage, and configure these features, you need the following permissions respectively:

- Configure Policies: Create, edit or delete policies
- Configure Health Rules: Create, edit, or delete health rules
- Configure Actions: Create, edit, or delete actions on agent properties UI
- Configure Actions: Create, edit, or delete email digests

Alert and Respond Policy Structure

As shown in the following diagram, policies match event triggers with the action to be taken in response to those triggers.



Alert and Respond Across the Platform

The alert and respond features work across AppDynamics products, including Infrastructure Visibility, Analytics, EUM, and Application Monitoring. Unless otherwise noted, this section describes the features in the context of Application Monitoring, which, by its nature, offers the broadest range of configuration and use case options. Specific features as described may not apply to other AppDynamics products.

Additional usage notes include:

- Policy triggers for applications can be health rule violation events or other types of events. Policy triggers for databases and analytics must be health rule violation events.
- The types of actions that you can create for an application include notifications, diagnostics, remediation, HTTP requests, custom actions, and cloud auto-scaling. The types of actions that you can create for a database or analytics are limited to notifications, HTTP requests, and custom actions.
- The types of entities affected by a health rule are more limited for databases and analytics than for applications.
- For information on using policies triggered by browser synthetic events, see 'Alerting and Synthetics' in Browser Synthetic Monitoring.

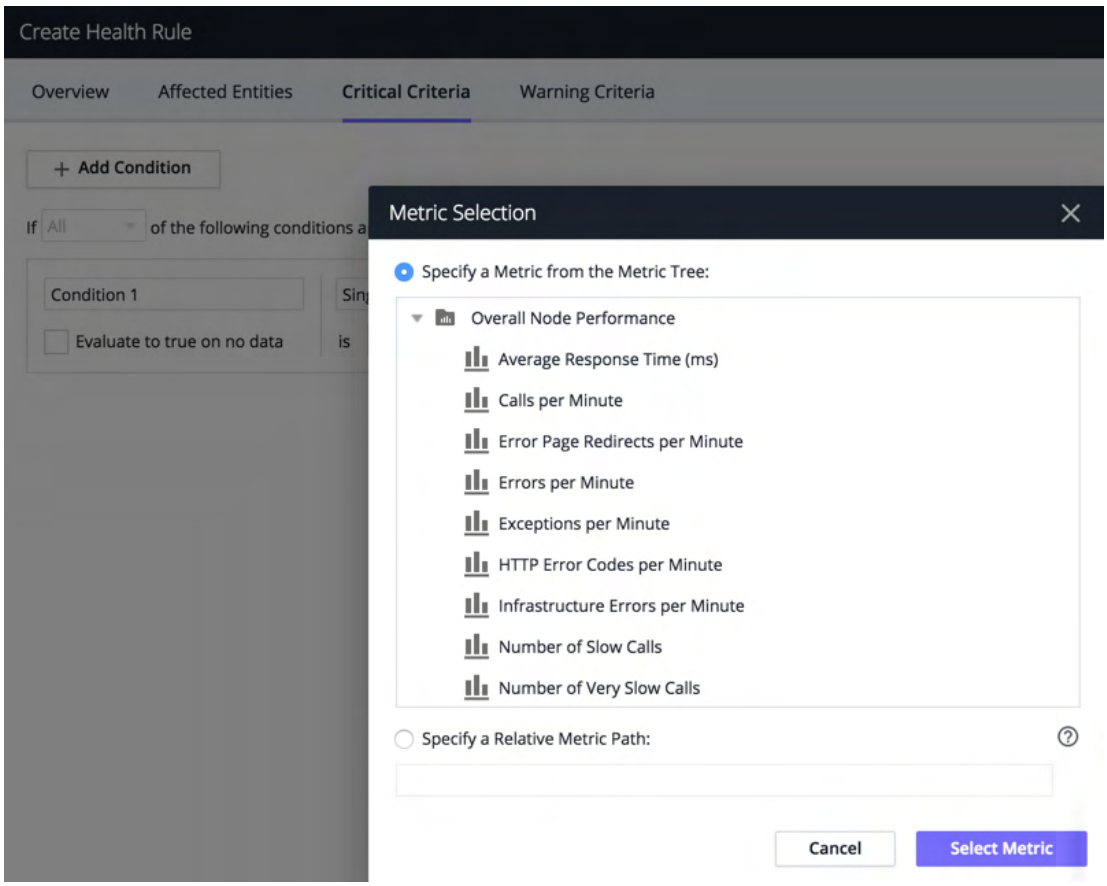
Example Use Cases

Related Pages:

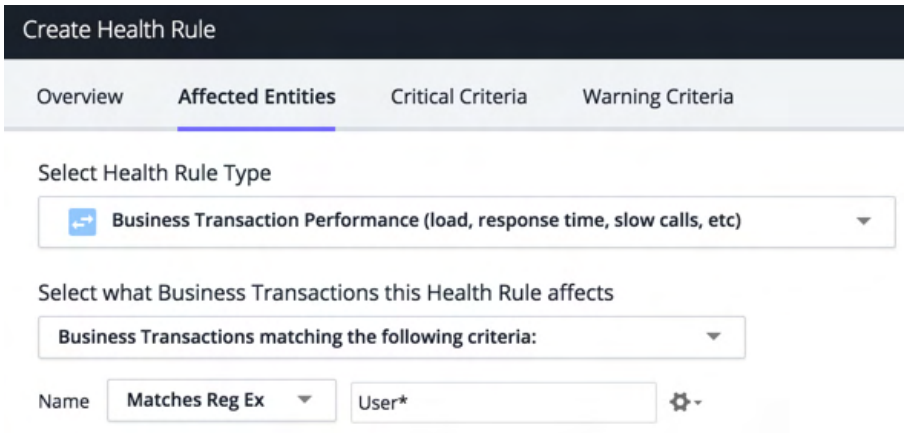
- Alert and Respond
- Policies
- Health Rules

This page provides use case examples for alert and response features:

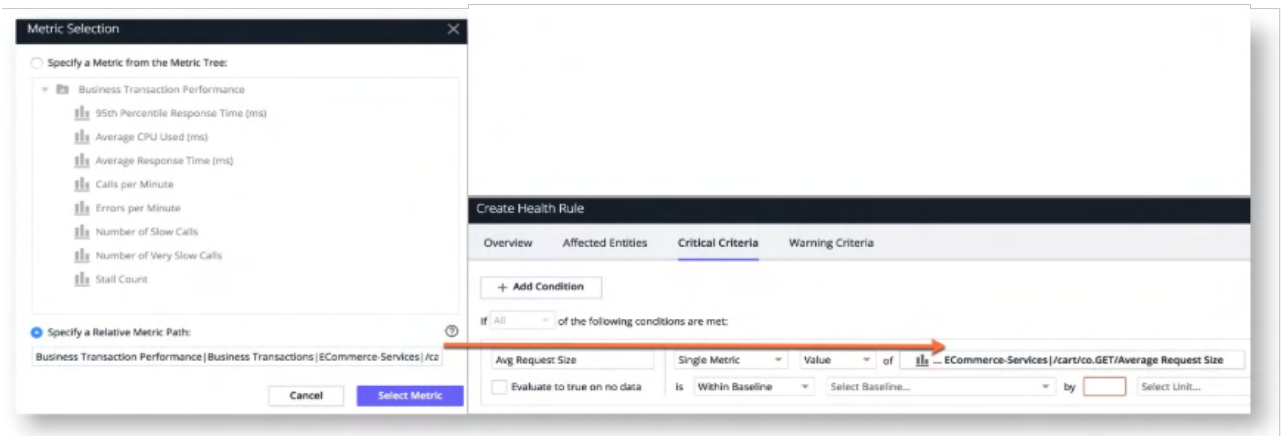
- Define health rules that apply to specific tiers or nodes. Instead of choosing specific nodes, you can trigger a rule when more than a certain percentage of nodes are unhealthy, say 20%.
- Start a diagnostic action for a business transaction.
- Alert when an app agent stops reporting to the Controller.
  Create a node health rule based on the value of the Availability metric reported by the agent. If Availability is less than one, the agent is not reporting.



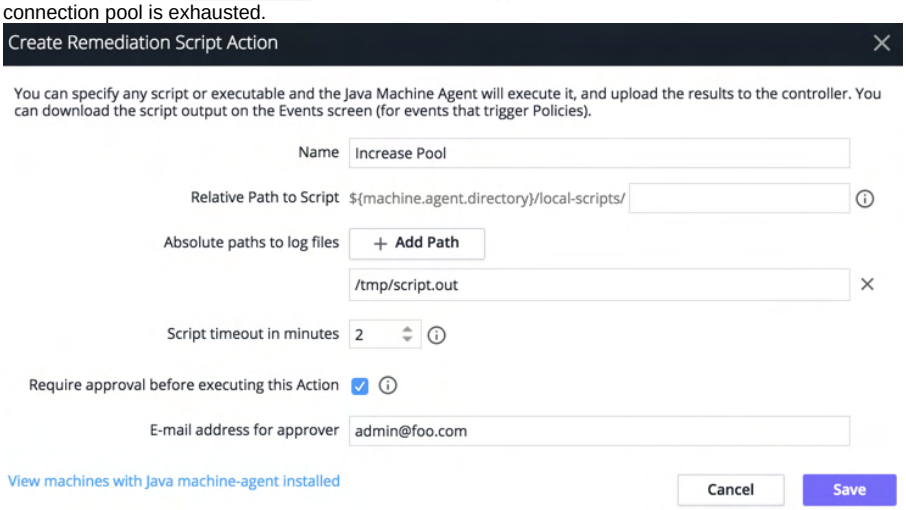
- Alert when the 95th percentile metrics for specific business transactions reach a certain value. You want to apply this rule only to business transactions with names beginning with *User*.



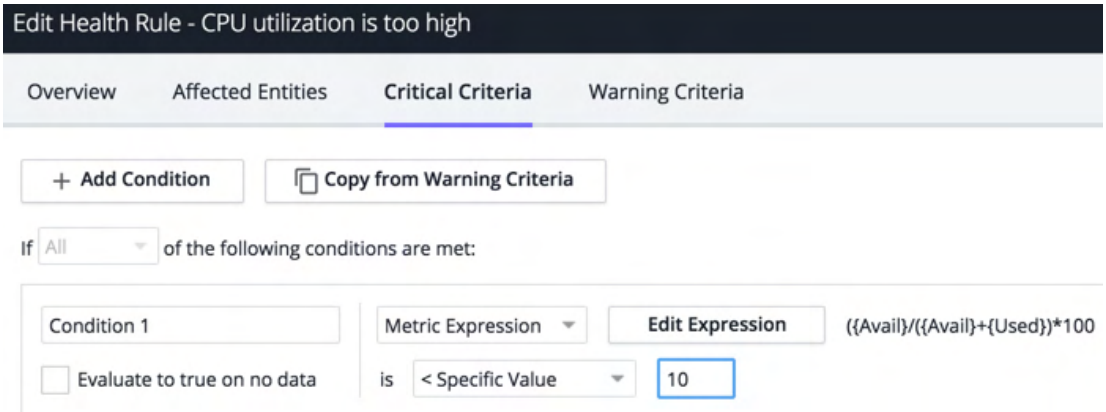
- You can generalize a health rule by specifying a relative metric path, rather than a specific metric. The health rule is evaluated for each of the affected business transactions. Use a relative metric path when you need to evaluate a single metric for multiple entities.



- You have a large operation with several development teams, each responsible for a different service. You create a health rule for one service and then copy it. Then create different policies in which you can pair each copy of the health rule to an alert addressed to the appropriate team.
- Start a script to change the size of the connection pool. You have an application that performs well for a normal load. However, peak loads can cause the application to slow. During peak load, the AppDynamics not only detects the connection pool contention but also allows you to create a remediation script that can automate increasing or decreasing the size of the connection pool. You can require human approval to run this script or just configure it to automatically execute when it is triggered. Create a runbook and associate it with a policy so that it will run when the connection pool is exhausted.

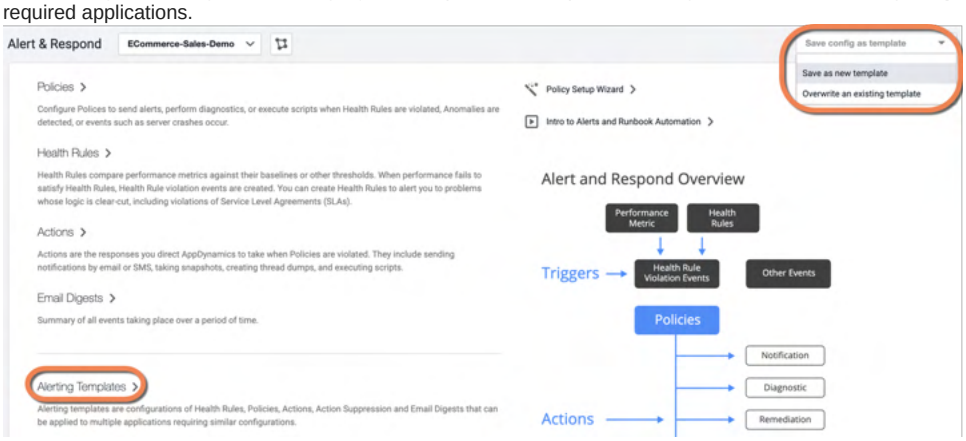


- Alert when the available disk volume is low. Use an expression including two metrics – available and used disk space – to be alerted when disk volume is low.



- When configuring **Alert and Respond** features for an application, you can select **Save Config as Template** to save the application configuration as a template. You can then apply this template to multiple applications to ensure consistency.

To make a global configuration change, you can update the template selecting **Overwrite an existing template** and then re-apply it to the required applications.



- To avoid receiving false alerts and to ensure that you calibrate the sensitivity of health rules appropriately, you can use Alert Sensitivity Tuning (AST). AST helps you visualize the impact of the alerting configuration as and when you configure. AST provides instant insights in a graphical format that helps you fine-tune the alerts.
  Depending on the baseline configuration you define, a graphical view of the metric data for the given baseline configuration displays. The graphical view is instantly updated when you update any configuration. You can also view granular details by modifying the graphical view. The metric data presented in the graph helps you calibrate the sensitivity of the metric evaluation and the health rule.

Policies

Related Pages:

- Health Rules
- Actions
- Configure Policies

This page provides an overview of policies in AppDynamics.

A policy consists of a trigger, which is based on one or more events, and an action, directly in response to the trigger. You can use policies to automate monitoring, alerting, and problem remediation.

View Policies

To view and create policies, access the Policies UI by selecting **Alert & Respond > Policies**.

The policy list displays all the policies created for your application, with its triggers and actions taken. You can modify a policy by double-clicking it in the policy list.

Policy Triggers

Policy triggers are events that cause the policy to fire. The events can be health-rule violation events or other types of events, such as hitting a slow transaction threshold or surpassing a resource pool limit. See Health Rules, Troubleshoot Health Rule Violations and Monitor Events.

The triggering events can be broad, affecting any object in the application, or very narrow, affecting a specific object. For example, you can have a broadly-defined policy that fires whenever a resource pool limit (such as >80% usage of EJB pools, connection pools and/or thread pools) is reached for any object in the application. Additionally, a narrowly-defined policy is one that would fire only when existing health rules on memory utilization or JVM garbage collection time are violated.

A policy is triggered when at least one of the specified triggering events occurs on at least one of the specified objects.

Policy Actions

You can assign the actions that are taken when a policy is triggered, such as an email or SMS notification. Other types of actions do more than just notify. For example, a resource pool violation can trigger a script that increases the pool size.

Other common actions include restarting an application server if it crashes, purging a message queue that is blocked, or triggering the collection of transaction snapshots. You can also trigger a custom action to invoke third-party systems.

Because the definition of health rules is separate from the definition of actions, you can specify different actions for an event in different contexts, such as a threshold crossing in either a tier or node context.

See Build a Custom Action for information about custom actions. See Actions for more information about the different types of actions.

Policy Actions in Batch

You can configure a policy to execute its actions either:

- Immediately for every triggering event.
  For example, if in a two-second period a policy matched 100 events, it would start its actions 100 times as soon as each event occurred.
- Once a minute for all the events that triggered over the last minute. This is the batch option. The **Execute actions in batch** checkbox is selected by default.
  For example, if in a two-second period, a policy matched 100 events and then no triggering events occurred for the next 58 seconds, the policy would start each action just once. The context for the actions would be all 100 events.

Which you choose depends primarily on the type of action. For a notification action, it probably doesn't make sense to send 100 emails or SMS messages in a few seconds. In this case, it makes sense to batch the actions with a summary of the events occurring during the last minute. This can be easily accomplished using an email template that iterates through the event list. See the example in Predefined Templating Variables.

However, if the actions are thread dumps, there is no reason to expect that all 100 events are on the same node. They might be on different nodes. For that kind of action, you probably want the thread dump to be taken for each event and also, not to wait another 58 seconds before taking the thread dump.

Configure Policies

Related Pages:

- Monitor Events
- Health Rules
- Troubleshoot Health Rule Violations
- Actions
- Enable an Email Server

This page provides instructions on configuring policies in AppDynamics. You can configure policies using these two methods:

- Use the Policy Setup Wizard for simple policies that send an email notification when a health rule is violated.
- Configure the policy manually for anything more complicated.

> To configure policies, you need the Configure Policies permission.

Policy Setup Wizard

For simple policies in the Controller, select **Alert & Respond > Policies > Policy Setup Wizard**.

To create a notification policy, you must set up an SMTP server and supply an email address. The policy is then created.

To modify the policy later, select the policy and click **Edit**.

Create Policy Manually

To create a policy manually in the Controller, select **Alert & Respond > Policies > Create Policy Manually**.

The **Create Policy** page contains the following panels:

- **Trigger**: Sets the events that trigger the policy, entities that are affected by the policy.
- **Health Rule Scope**: Defines the Health Rules that trigger the policy.
- **Object Scope**: Defines the objects that trigger the policy for application and user experience contexts.
- **Actions**: Set the actions to take when the policy is triggered.

In either panel, you can configure the policy name, enabled status, and whether to batch the actions executed by the policy.

Create Policies

1. Click **Alert & Respond**.
2. Click **Policies** either in the right panel or the left navigation pane.
3. Select the context for the policy (specific Application, User Experience, Databases, Servers, or Analytics) from the dropdown.
4. To create a new policy, click **Create Policy** (**+**).

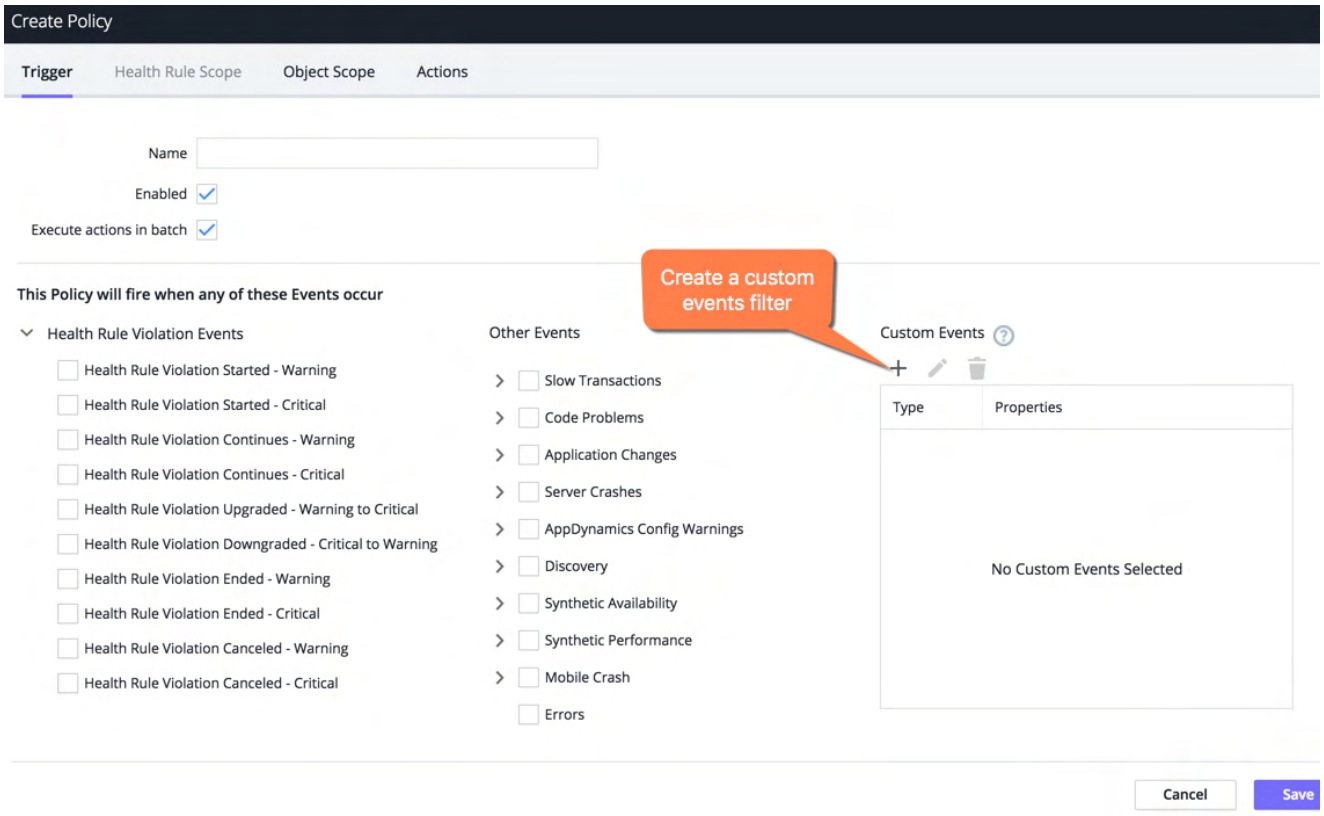
You can edit, copy, enable or disable, and delete a policy by clicking the appropriate button.

Configure Policy Triggers

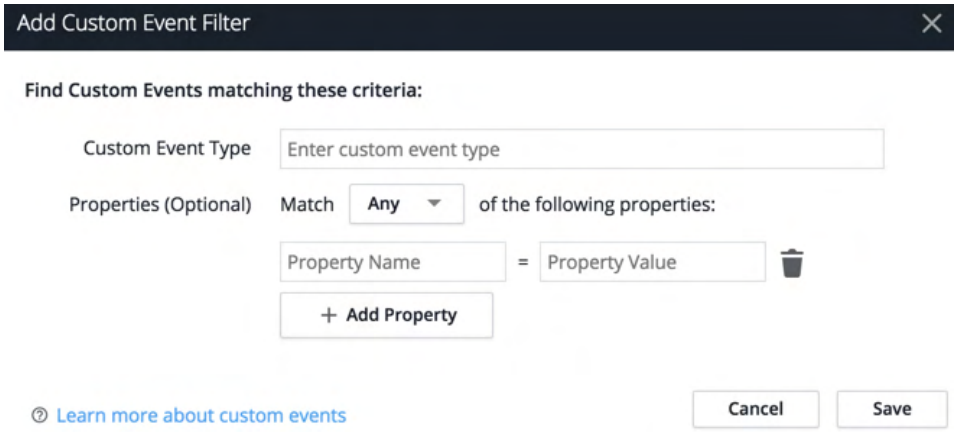
The policy trigger panel defines the events and objects that cause the policy to run and invoke its actions.

For policy triggers that depend on health violation events, you must create the health rules before you can create a policy that uses them. See Health Rules and Troubleshoot Health Rule Violations.

1. Click **Create Policy (+)** or select a policy and click **Edit** (pencil button).
2. Enter a name for the policy in the **Name** field.
3. To enable the policy, select the **Enabled** checkbox. To disable the policy, clear the **Enabled** checkbox.
   You can also enable or disable a policy by selecting it from the policy list and clicking **Enable** or **Disable**.
4. If you clear the **Execute actions in batch** checkbox, the policy starts its actions immediately for every triggering event. If you check this checkbox, the policy starts its actions once for all the triggering events that occurred in the last minute. See 'Policy Actions in Batch' in Policies.
5. Select the **Trigger** panel if it is not already selected.
6. Check one or more types of events that should trigger the policy. You may need to click the arrow to display specific events within an event category.
   If you check at least one health rule violation event, you can choose whether any or all health rule violations, or only specific health rule violations will trigger the policy.



If your AppDynamics environment includes browser synthetic monitoring, other additional events display for Synthetic Availability and Synthetic Performance. See 'Alerting and Synthetics' in Browser Synthetic Monitoring.
You can optionally designate specific custom events to trigger the policy, using the **Custom Events** panel in the lower right corner. Click **+**.



Specify the custom event type. Optionally, add particular properties on that event as key/value pairs.

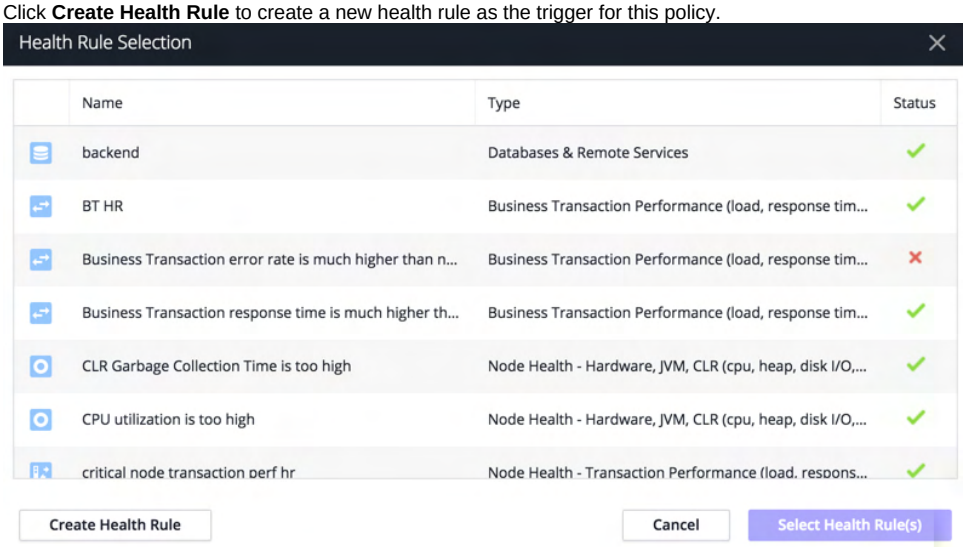
- For **Any**, at least one property must exist and match.
- For **All**, all properties must exist and match.

Configure Health Rule Violation Triggers

The **Health Rule Scope** panel determines the scope of the health rule violations that trigger the policy. If you have not created a health rule, you can create one by selecting **Create Health Rule** in this panel.

1. If you have not already done so, edit the policy to which you want to add Health Rule triggers.
2. Select the **Health Rule Scope** panel, then select **Any Health Rules** if you want the policy triggered by a violation of any health rule.

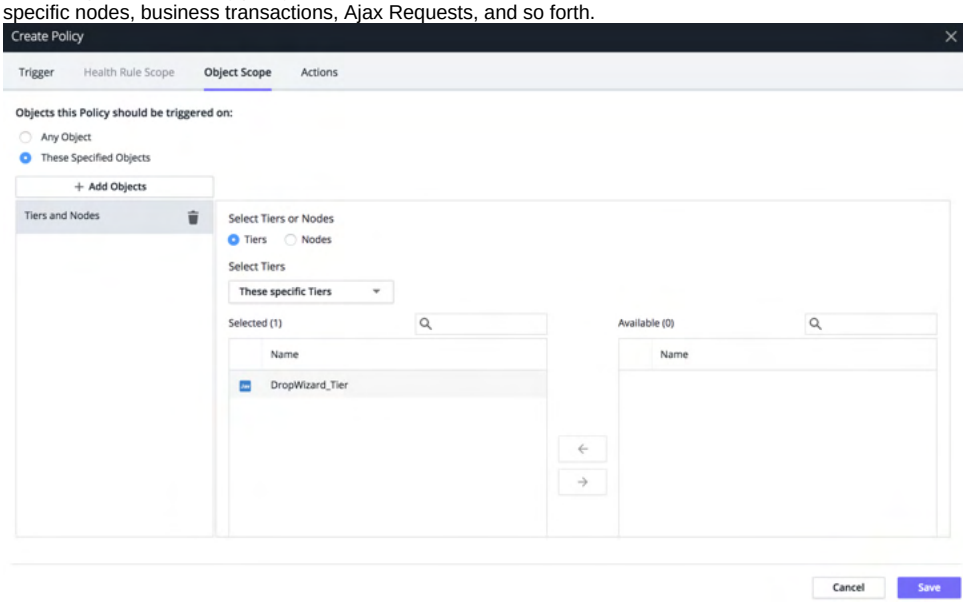
3. To designate specific health rule violations to trigger the policy, click **+**, and then select the health rules from the embedded health rule browser. Click **Create Health Rule** to create a new health rule as the trigger for this policy.



Configure Object Scope

The **Object Scope** panel defines the objects the policy is triggered on. The object scope applies only to business transaction performance type health rules and node health type health rules which comprise tiers and nodes.

1. When you have finished selecting the health rule violations that trigger the policy, click the **Object Scope** panel to select objects to be monitored in case of an event. If you select **Any Object**, the policy will be triggered by the configured events when they occur on any object in your application.
2. To restrict the policy to specific objects, select **These Specified Objects** and then choose the objects.
   For example, the following policy starts when selected events occur on the ECommerce Server tier. You can similarly restrict the objects to specific nodes, business transactions, Ajax Requests, and so forth.



   You can restrict the affected nodes on the node name, on the type of node (such as Java and .NET) on nodes in certain tiers, or on criteria such as meta-info, environment variables, and JVM system environment properties. Meta-info includes key-value pairs for:
   - key: `supportsDevMode`
   - key: `ProcessID`
   - key: appdynamics`.ip.addresses`
   - any key passed to the agent in the `appdynamics.agent.node.metainfo` system property

   To trigger by Health Rule Violation Events, leave the selection at **Any object**. Selecting **These specified objects** means that events occurring within the selected objects, for example, slow transactions, errors, and so forth, trigger the policy.
3. Click **Save**.

Configure Policy Actions

The policy actions panel binds an action to the trigger. It defines which actions the policy automatically initiates when the trigger causes the policy to start.

The actions must be created before you can create a policy that starts them. See Actions and the documentation for individual types of actions (for example, notification actions, remediation actions) for information on creating an action.

1. If you have not already done so, edit the policy to which you want to add actions. See To Access the Policy Wizard.
2. Select the **Actions** panel.
3. Click **+**. The list of existing actions displays. The available actions vary depending on the product area selected for the policy, such as Applications, Servers, Databases and so on.
   You can filter the list by selecting the checkboxes for the types of actions you want to see.
4. In the list of actions, select the action that you want this policy to execute and click **Select**.
   If you do not see an appropriate action for your needs, click **Create**. For information on creating actions, see Actions. After you have created the action, select it here to assign it to the policy that you are configuring.
5. Click **Save**.

You can optionally test whether your action will be fired using the Test Action Execution. Ensure that the policy you are testing is enabled.

Test Action Execution

Related Pages:

- Alert and Respond
- Monitor Events
- Configure Policies
- Actions

This page provides an overview of events actions in AppDynamics. An event represents an error or exception generated by the application, the crossing of a performance threshold, or an operational change in the application, such as a JVM restart.

You can view and analyze events in the Controller UI. From the **Database Visibility**, **Server Visibility**, or **User Experience** pages, you can view events by selecting **Events** from the left navigation panel. The events list also provides information on which policy actions are configured to start when a specific event occurs. It does not execute the actual actions configured for the policy.

This information provides answers to these questions:

- Why didn't my policy action execute? Perhaps policies are not actually configured to respond to an event as you thought they were.
- Why am I getting multiple actions, such as multiple emails, for the same event? Perhaps more than one policy is configured to perform the same action.

To see the actions that are triggered for an event:

1. In the events list, select the event for which you expect actions to start.
2. From the Actions menu, select **Test Action Execution**.

Anomaly Detection

Deployment Support

Related Pages:

- Enabling and Configuring Anomaly Detection
- Troubleshooting Anomalies

The AppDynamics Anomaly Detection and Automated Root Cause Analysis are two features designed to reduce Mean Time To Resolution (MTTR) for application performance problems.

Anomaly Detection automatically determines whether every Business Transaction in your application is performing normally. Automated Root Cause Analysis helps you quickly determine the root cause for problems revealed by Anomaly Detection.

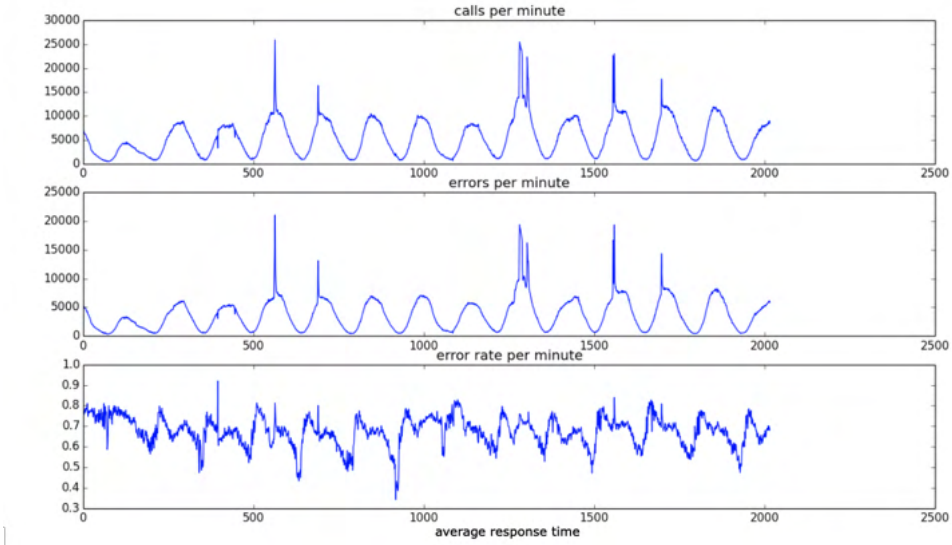
How Does Anomaly Detection Work?

Anomaly Detection uses machine learning capabilities to reduce the `Mean Time to Detect` (MTTD) when an anomaly occurs in a business transaction. It uses a specially designed algorithm that does not require you to configure anything. The Anomaly Detection algorithm works as follows:

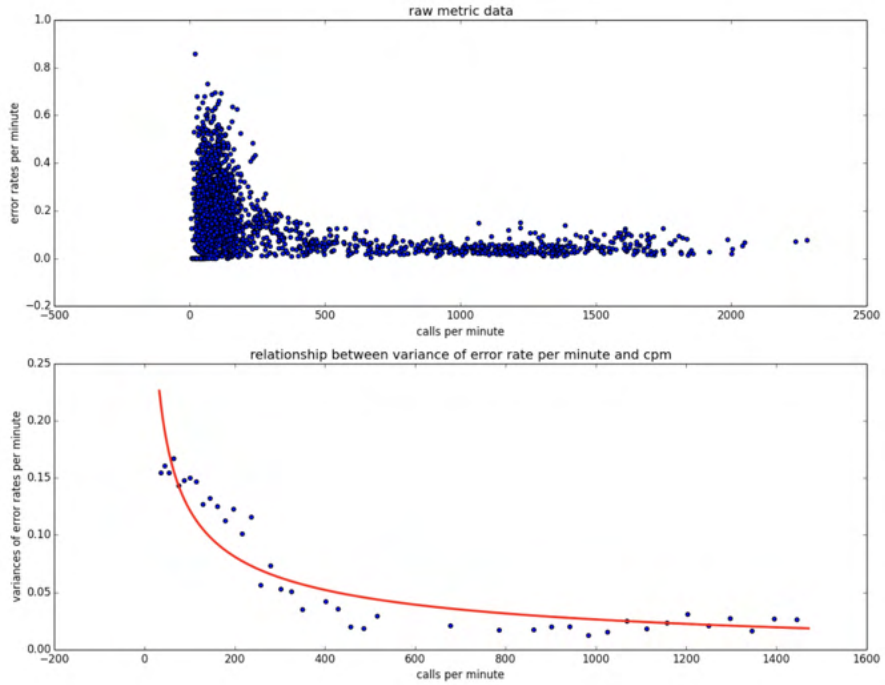
1. It detects if any abnormal reading is reported for the **Errors per minute** (EPM) metric.
2. It detects if any abnormal reading is reported for the **Average Response time** (ART) metric.
3. It then combines the data it learned from these metric readings using heuristics that are designed to reduce alert noise.

Anomaly Detection employs multiple techniques to ensure that the metric data it collects is accurate:

- It disregards any temporary spikes and periods of no data
- It normalizes the metric data. For example, when determining the EPM metric data, any spikes may not indicate a real problem unless there is a corresponding increase in **Calls per Minute** (CPM). EPM data may not be useful in itself, hence, Anomaly Detection uses Error Rate (EPM/CPM).



- It does not apply traditional seasonal baselines. Instead, it correlates the variance of EPM and ART to CPM to obtain reliable results.



What is Root Cause Analysis (RCA)?

When an entity in your application has an anomaly, you will want to know why. Anomaly Detection uses AI capabilities to enable Automated Root Cause Analysis to monitor the health of all the entities in your application, and show you the suspected causes for every anomaly. You can confirm or negate the suspected causes with a brief look, and drill down into deviating metrics and snapshots, as desired. Thus, you can quickly determine the root cause of any anomaly in the application.

How Does RCA Work?

RCA reduces the Mean Time to Identification (MTTI) by automatically pointing to the source of the problem. RCA considers metrics to identify the fault domain, and snapshots and events from the entire application, to find and surface suspected problems. This holistic approach is performed in two parts:

1. Fault domain isolation—Identification of the exact location of the problem in the system
2. Impacted component analysis—Analysis of logs, snapshots, trace, infrastructure, and so on to determine the affected components

How is Anomaly Detection Different from Health Rules?

While both Anomaly Detection and health rules alert you to performance problems in your application, Anomaly Detection provides powerful insights that would be difficult to obtain using health rules.

Anomaly Detection	Health Rules
Anomaly Detection uses machine learning to discover the normal ranges of key business transaction metrics and alerts you when these metrics deviate significantly from expected values. This enables Anomaly Detection to identify a wider range of problems than a person could capture in Health Rules.	Health rules are manually created to apply logical conditions that one or more metrics must satisfy. For example, you could monitor the **Average Response Time** (ART) to check if this metric deviates from the configured baseline.
Anomaly Detection requires no configuration except when you want to limit anomaly alerting.	AppDynamics provides a default set of health rules and you create additional health rules manually as required, configuring time periods, trends, and schedules.
Anomalies are associated with business transactions.	Health rules apply to any entity, for example, business transactions, service endpoints.

Enabling and Configuring Anomaly Detection

Related topics:

- Anomaly Detection
- Troubleshooting Anomalies

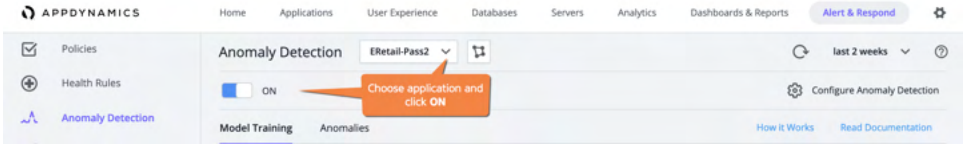
Anomaly Detection must be enabled, but requires no configuration except to limit anomaly alerting. Enabling Anomaly Detection also enables Automated Root Cause Analysis.

> Anomaly Detection and Automated Root Cause Analysis are available to SaaS customers only.

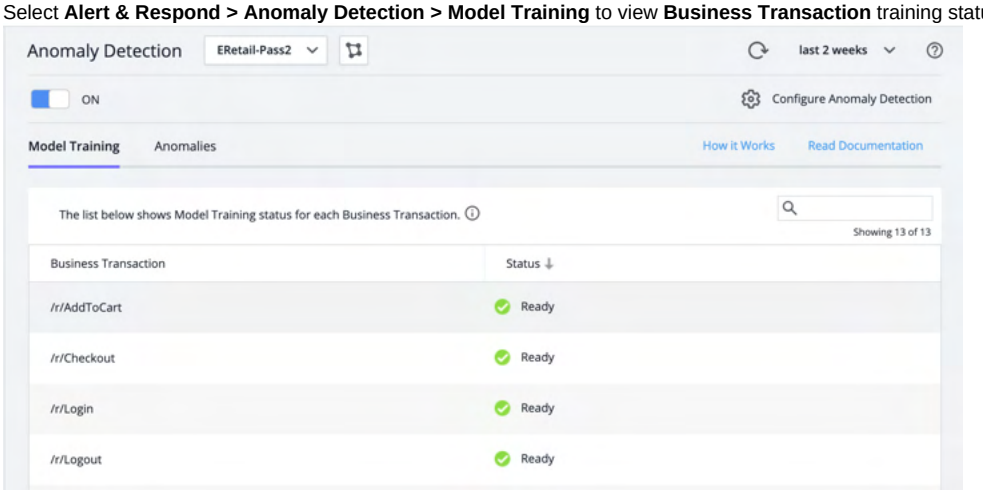
Enable Anomaly Detection

Enable Anomaly Detection separately for each application.

1. In **Alert & Respond > Anomaly Detection**, choose the desired application from the dropdown, and toggle Anomaly Detection **ON.**



   After you enable Anomaly Detection, it takes 48 hours for Anomaly Detection and Automated Root Cause Analysis to become available. During that time, the machine learning models train on the business transactions in your application.
2. Select **Alert & Respond > Anomaly Detection > Model Training** to view **Business Transaction** training status.



   This table explains the training statuses of a **Business Transaction**.

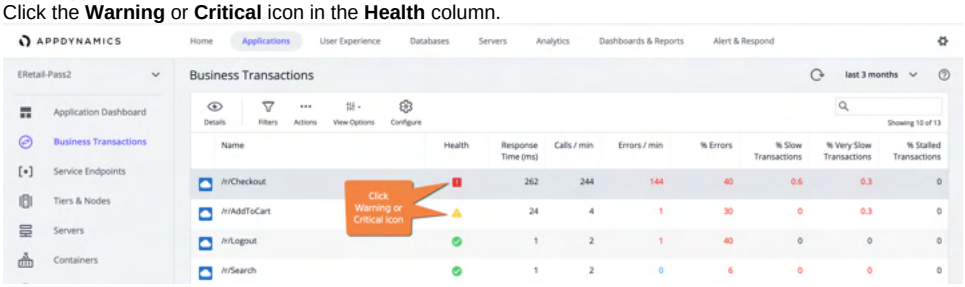
Status	Meaning
In Training	Model training is in progress for the Business Transaction.
Ready	Model training is complete, and the Business Transaction is healthy.
Warning	Model training is complete, but the Business Transaction has experienced one or more Warning level anomalies during the training period.
Critical	Model training is complete, but the Business Transaction has experienced one or more Critical level anomalies during the training period.
Not Available	Model training is incomplete, and the Business Transaction is not visible to Anomaly Detection.

   The models continue training as long as Anomaly Detection is enabled. If traffic to a **Business Transaction** is interrupted for long enough to prevent training that day, Anomaly Detection continues to function using the models from the previous seven days.

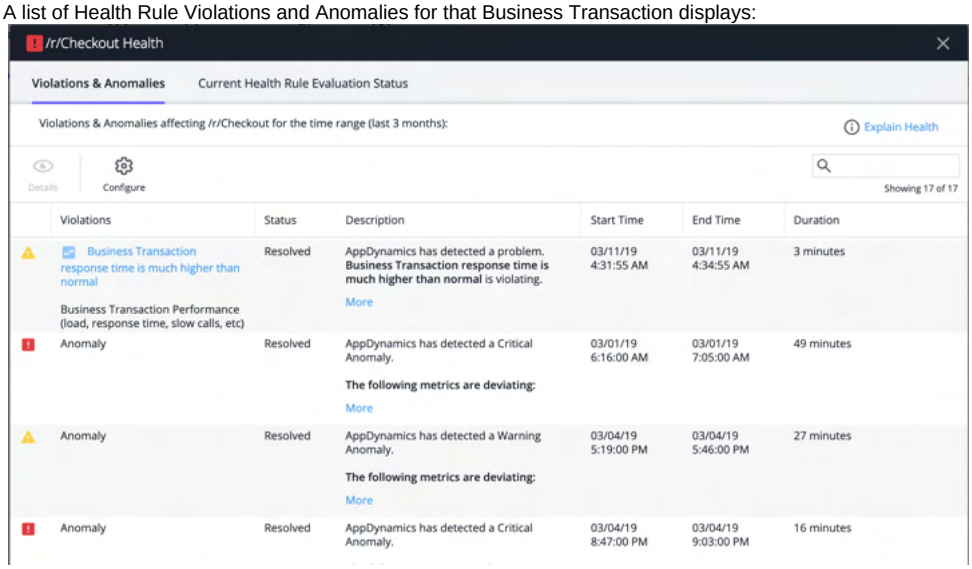
> No machine learning models are trained for **Business Transactions** which have very low CPM, because the sample size would be so small that the resulting model would be unreliable.

Monitor Anomalies

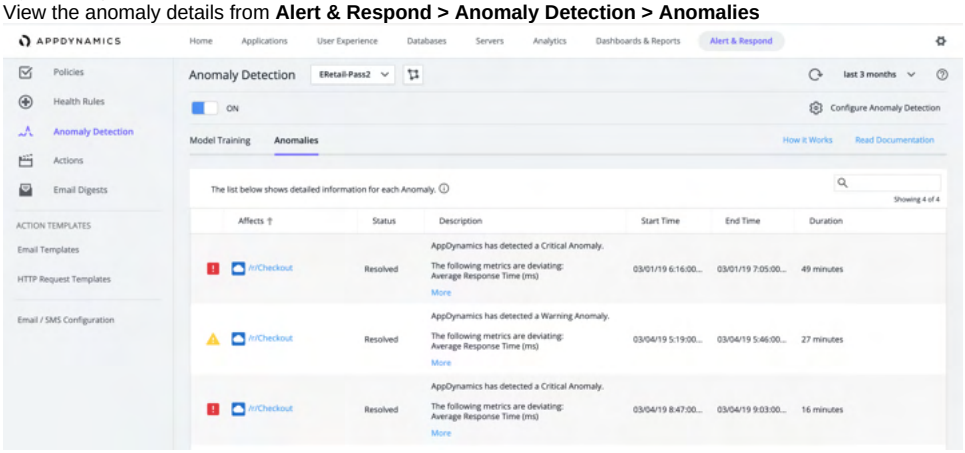
1. From **Applications > Business Transactions**, select any **Business Transaction** of interest.
2. Click the **Warning** or **Critical** icon in the **Health** column.



A list of Health Rule Violations and Anomalies for that Business Transaction displays:

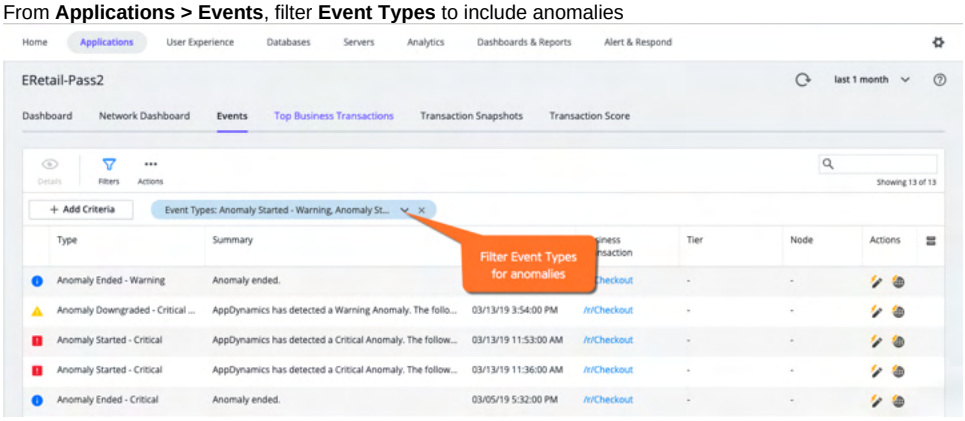


3. You can view lists of anomalies in multiple ways. Monitoring anomalies can reflect how you work with AppDynamics. Choose any of these options to open a detailed view that includes the results of Automated Root Cause Analysis.
    - If you set up and validate tools for a tools team:
      View the anomaly details from **Alert & Respond > Anomaly Detection > Anomalies**

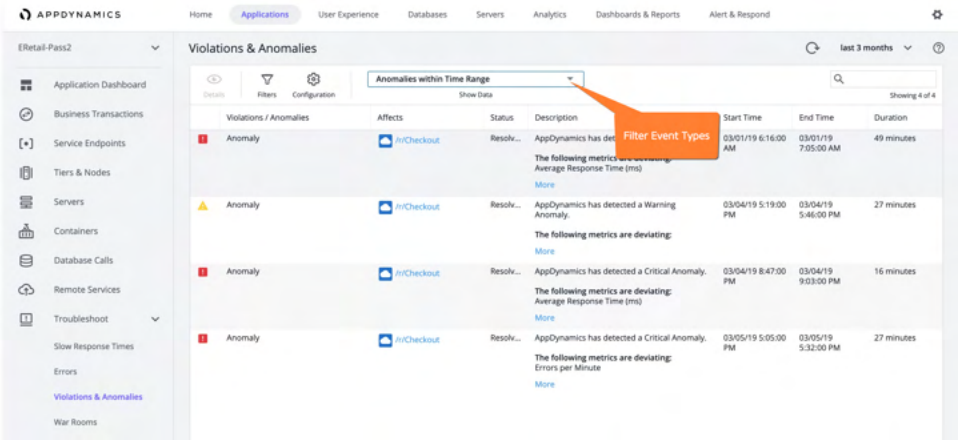


    - If you monitor applications for an application ops team:

- From **Applications > Events**, filter **Event Types** to include anomalies



- From **Applications > Troubleshoot > Violations & Anomalies**, filter **Event Types** to include anomalies



You can configure policies to be triggered by anomalies, similar to the way you configure policies to be triggered by Health Rules.

Configure Anomaly Detection

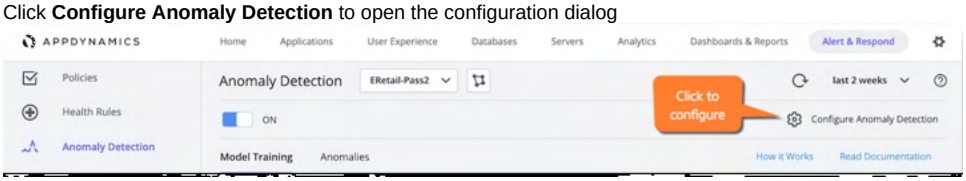
By default, Anomaly Detection alerts you about all Anomalies for all Business Transactions in your application.

You can configure Anomaly Detection to surface only those Anomalies within the combination of Business Transactions and severity levels that you specify. Do this if you prefer to see fewer, more narrowly focused alerts.

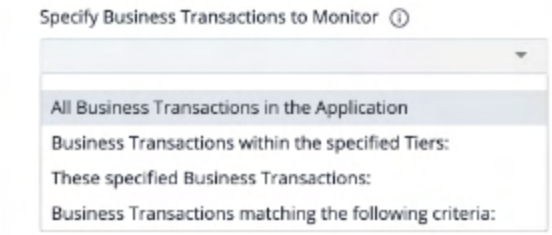
You do not need to configure or enable Automated Root Cause Analysis.

To configure Anomaly Detection:

1. Click **Configure Anomaly Detection** to open the configuration dialog



2. If desired, limit the **Business Transactions** on which you want Anomaly Detection to alert
    - Default (all Business Transactions) is shown



3. If desired, limit alerts to either **Warning** or **Critical** severity only

- Default (All Severities, meaning both Warning and Critical) is shown

Troubleshooting Anomalies

Related topics:

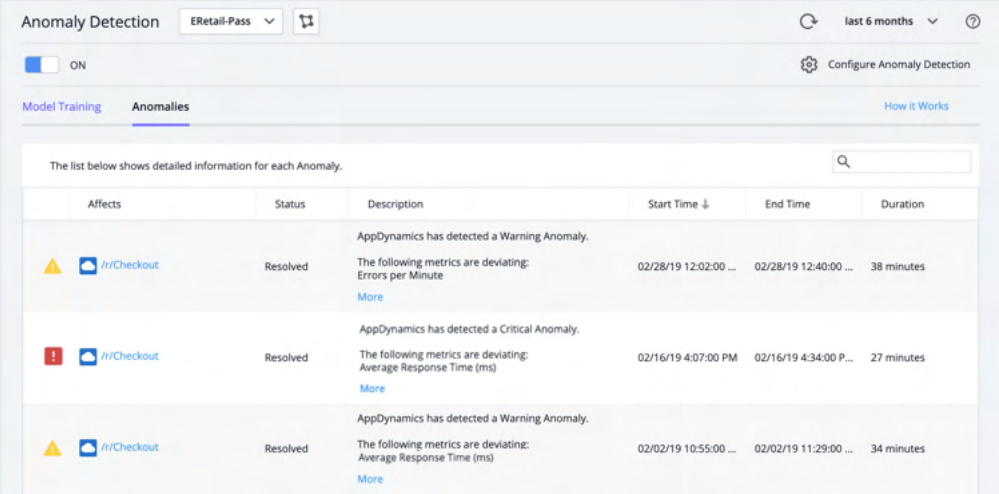
- Anomaly Detection
- Enabling and Configuring Anomaly Detection

To demonstrate techniques for using Anomaly Detection and Automated Root Cause Analysis effectively, this example follows an anomaly from the moment it surfaces until its root cause is confirmed. The troubleshooting process can begin with any of the multiple ways to view anomalies. This example assumes that you start with the **Alert & Respond > Anomaly Detection** page.

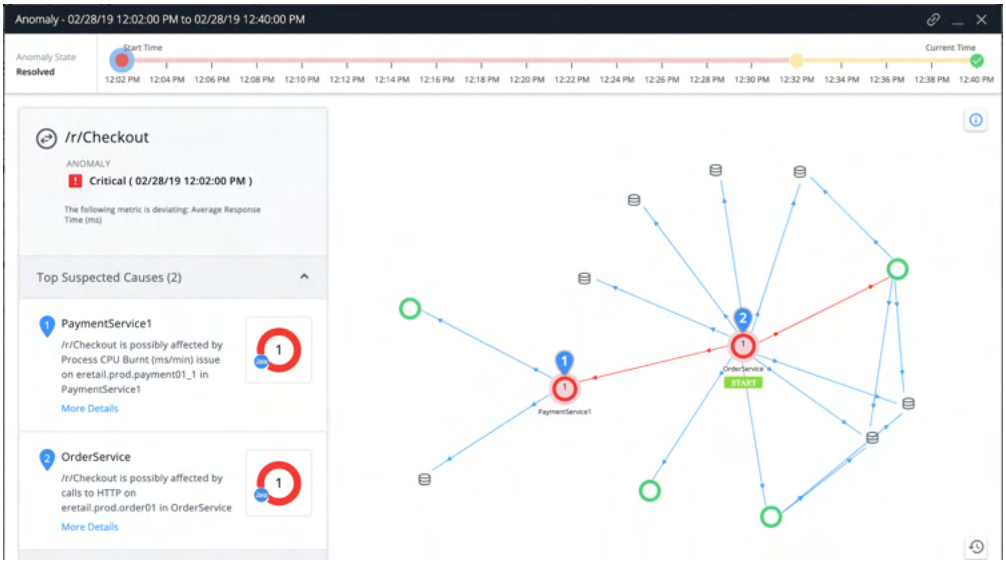
> Anomaly Detection and Automated Root Cause Analysis are available to SaaS customers only.

Drill Into an Anomaly

- **In Alert & Respond > Anomaly Detection**, view the Anomalies tab.



- Double-click an anomaly to open the detailed view.



Initially, the page describes everything that is occurring during the anomaly's Start Time. To review how things change later in the anomaly's lifecycle, click events further along its timeline.

Examine the Anomaly Description

The anomaly description describes the anomaly in relation to its named Business Transaction, the severity level of the selected state transition event, and the top deviating Business Transaction metrics.

In this example, these are:

- Business Transaction: `/r/Checkout`
- Severity Level: Critical
- Top Deviating Metrics: Average Response Time

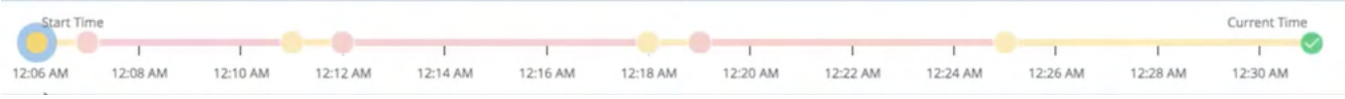
The deviating metric is Average Response Time indicating that checkout responding slowly is the problem.

Examine the Timeline

The state transition events mark the moments when the anomaly moves between **Warning** and **Critical** states.

- The timeline in this example begins in the **Critical** state, followed 30 minutes later by a transition to the **Warning** state, which lasts only eight minutes.
- Because this simple anomaly starts in the **Critical** state and remains there for most of its lifecycle, we can probably learn all we need to know from the initial event

By contrast, patterns that appear in more complicated timelines may help you to understand anomalies. For example, this timeline from a different anomaly repeatedly toggles from a brief **Warning** state to a longer **Critical** state:



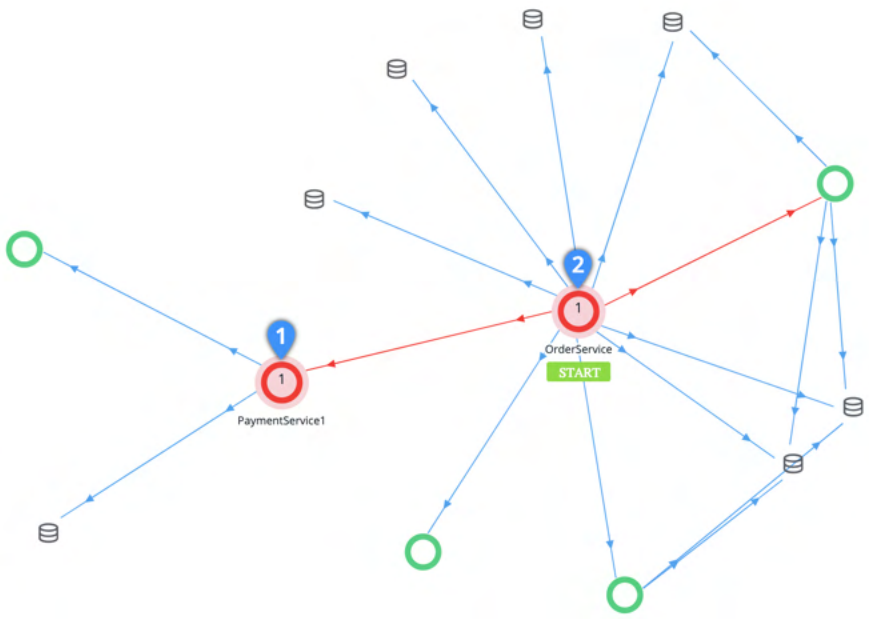
In this case, you should examine several state change events to determine what clues toggling between states offers about problems in your application.

Examine the Flow Map

The example flow map contains:

- The **START** label shows that the Business Transaction begins with the OrderService tier

- Between the OrderService tier and its numerous dependencies, two tiers are red—these are the tiers where the system has found Suspected Causes



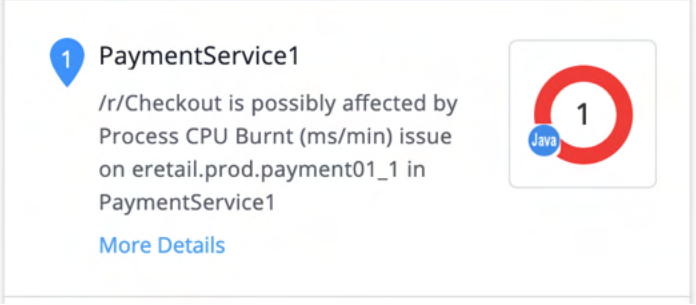
You can now focus on determining which of the red tiers contains the root cause of the anomaly.

> **Anomaly Detection flow maps are different**
>
> There are two types of flow maps in AppDynamics:
>
> - Anomaly Detection and Automated RCA flow map (described on this page)
> - Business Transaction flow map
>
> Each of these flow maps detects deviating or unhealthy entities in its own way. Therefore, you will see some differences:
>
> - The two flow maps may show a different health status (as represented by color) for the same entity because each one uses its own algorithm to determine health
> - User preferences for positioning or hiding entities saved for the Business Transaction flow map have no effect on the Anomaly Detection flow map
> - Some links between tiers might be shown in one type of flow map but hidden in the other
>   - For example, when no data is flowing through a tier or between tiers:
>   - Business Transaction flow map may hide them, as 'inactive' tiers or links
>   - Anomaly Detection flow map may show them in order to represent the application topology completely

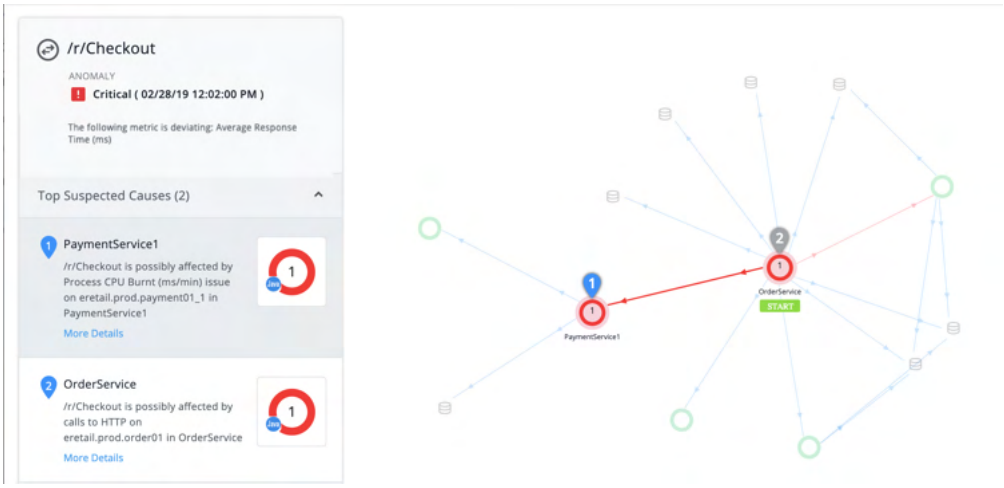
Examine the Top Suspected Causes

The Top Suspected Causes show likely root causes of a Business Transaction performance problem. In this case, we want to know why Checkout is responding slowly.

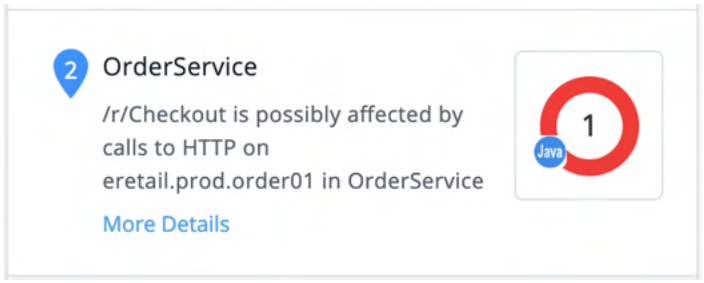
The first Suspected Cause is a Process CPU Burnt issue on the eretail.prod.payment01_1 node of the PaymentService1 tier:



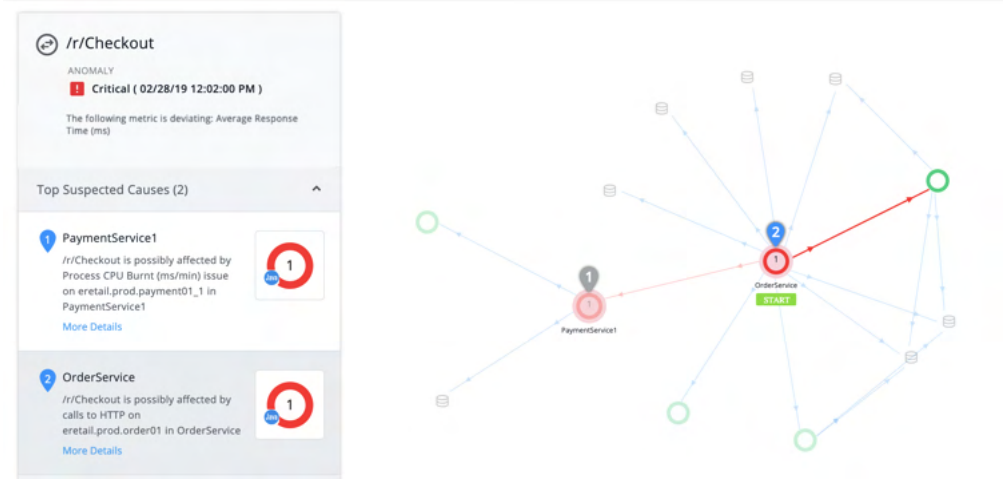
Hover over the Suspected Cause to highlight the relevant entities in the flow map. Everything but the critical path fades away, revealing that OrderService, where the Business Transaction starts, and which had a degraded response time, relies on PaymentService1:



The second Suspected Cause is an HTTP call on OrderService itself.



Hover to highlight the affected entities:



Which Suspected Cause is the root cause? Which is only a symptom of the overall problem?

- We have a plausible root cause in the Process CPU Burnt issue on PaymentService1 tier, which is ranked likeliest by the system.
- Meanwhile, the HTTP call on OrderService bears some analysis:
  - An HTTP call includes both a request and a response
  - We know that the tier on the other end, PaymentService1, has its own problem
  - Therefore, we can infer that the HTTP response from PaymentService1 is what makes the call slow

Now we see that both Suspected Causes originate with PaymentService1, and the HTTP call issue is really a side-effect of the Process CPU Burnt issue. The system's ranking makes sense.

As we continue to investigate, if we decide that the Process CPU Burn issue is not the root cause, we can reconsider the HTTP call.

There can be anywhere from zero to three Top Suspected Causes. For example, if ART is high but every entity connected with ART is behaving normally, there are zero suspected causes because no suspected cause can be identified.

Drill Into a Suspected Cause

Click **More Details** for the Suspected Cause to review:

- Simplified timeline
- Metrics graphed over time

Two types of graphed metrics display:

- **Top Deviating Metrics** for the Business Transaction
- **Suspected Cause Metrics**

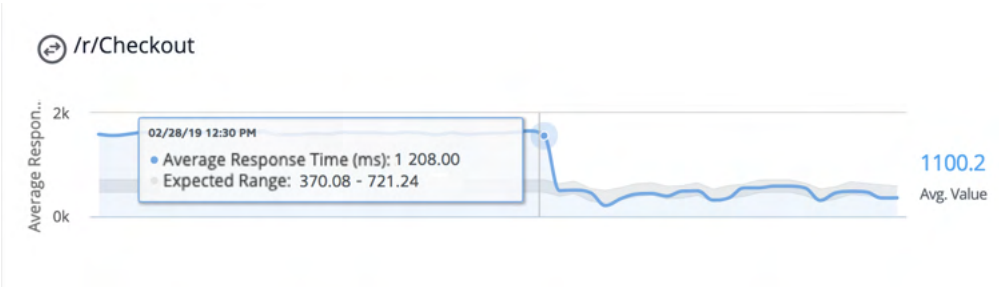
Examine Top Deviating Metrics for the Business Transaction

Deviating Business Transaction metrics can indicate why an anomaly was important enough to surface. (The system does not surface anomalies for every transitory or slight deviation in metrics. Such anomalies would be of dubious value, since their customer impact is minimal. For the same reason, anomalies are surfaced for Business Transactions which have a CPM of under 20.)

Each deviating metric is shown as a thin blue line (the metric's value) against a wide gray band (the metric's Expected Range).

You can:

- Scroll along the graph to compare a metric's value with its Expected Range at any time point
- Hover over a time point to view the metric's value and Expected Range in numerical form



> Hovering to view values in numerical form is essential when the Expected Range is so much smaller than the deviating metric values that the gray band "disappears" into the X axis.

In this example:

- The deviating metric spiked remained elevated for about 30 minutes, then subsided back into Expected Range
- Seven minutes after the metric returned to its Expected Range, the Severity Level changed from **Critical** to **Warning**, and eight minutes after that, to **Normal**

Hovering over time points tells us that for the period of deviation: the Average Response Time was around 1200 ms and above, while its Expected Range was from 370.08 to 721.24 ms

With a key metric elevated by this large amount, it made sense for the system to surface this anomaly.

> Because Top Deviating Metrics are at the Business Transaction level, they are the same for all Suspected Causes.

Examine Suspected Cause Metrics

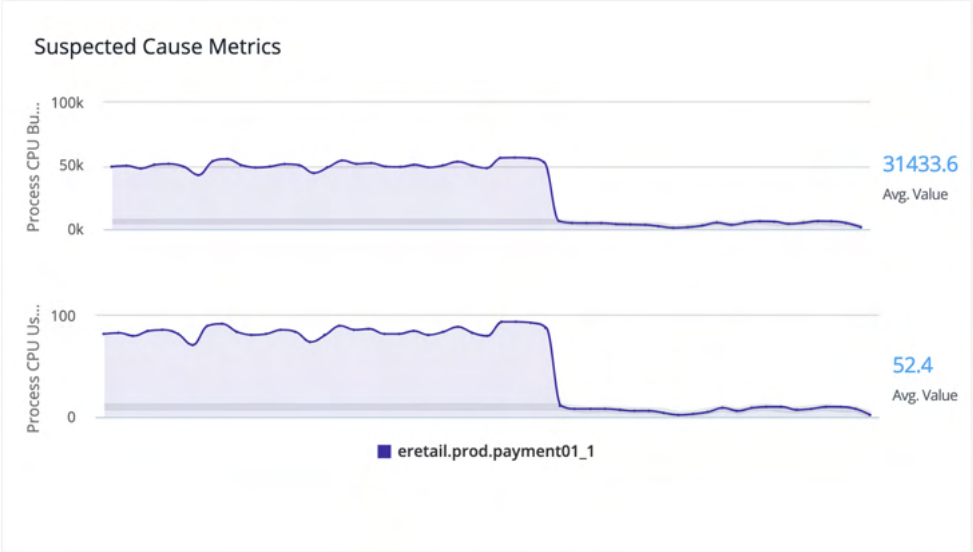
You view, scroll through, and hover over Suspected Cause Metrics similar to Top Deviating Metrics.

Top Deviating Metrics describe the Business Transaction, while Suspected Cause Metrics describe tiers or nodes further down in the entity tree. This means that if you have ART as a Top Deviating Metric and as a Suspected Cause Metric, *those are two different metrics*. Likewise, EPM as a Top Deviating Metric and as a Suspected Cause Metric are also two different metrics.

While the Suspected Cause Metric likely contributes to the way the Top Deviating Metric behaves, the values of the two metrics will differ.

In this example,

- Suspected Cause Metrics are shown for the eretail.prod.payment01_1 node within the ProcessPayment1 tier
- That is the *only* node the tier has—if the tier had multiple nodes, metrics could be viewed separately for each node
- The pattern of elevation in the Process CPU Burnt and Process CPU Used metrics perfectly matches the pattern we saw in the Business Transaction metrics



The hypothesis is now confirmed:

1. CPU usage spiked on ProcessPayment1, a tier that is downstream from the tier where the Business Transaction starts
2. This slowed down response time on ProcessPayment1, including its HTTP response to the HTTP request from OrderService
3. The slow HTTP call, in turn, slowed response time on OrderService
4. Since OrderService is where the Checkout Business Transaction starts, Checkout has a slow response time anomaly
5. Since the Process CPU Burn issue on ProcessPayment1 is the Suspected Cause that's deepest in the entity tree, that is the root cause of the anomaly

Results

We used Anomaly Detection and Automated Root Cause Analysis to quickly find the root cause of a Business Transaction performance problem. What kind of time and effort did Anomaly Detection and Automated Root Cause Analysis save us in this case?

Recall that the tier where the Business Transaction started, OrderService, has multiple dependencies including other services and datastores. Anomaly Detection and Automated Root Cause Analysis eliminated (1) all but two tiers as origins of the slow response time on OrderService, and (2) all but the most relevant of the many metrics on those tiers.

You were spared the tedious process of investigating multiple metrics on each dependency in turn. Instead, you confirmed or negated Suspected Causes with a quick glance at timelines, flow maps, and metrics. Anomaly Detection and Automated Root Cause Analysis performed the vast majority of the work in root cause analysis, presenting you with the information you needed to quickly form and verify a hypothesis.

(Optional) Inspect Snapshots and Execute Actions

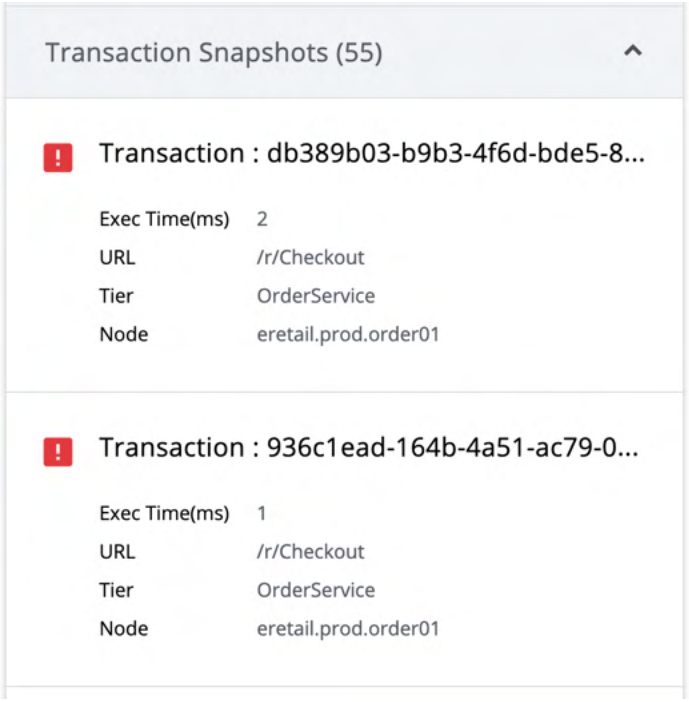
When you view an anomaly, you can inspect

- Business Transaction snapshots from the time period of the anomaly
- Actions executed as the result of policies you have configured for the Business Transaction

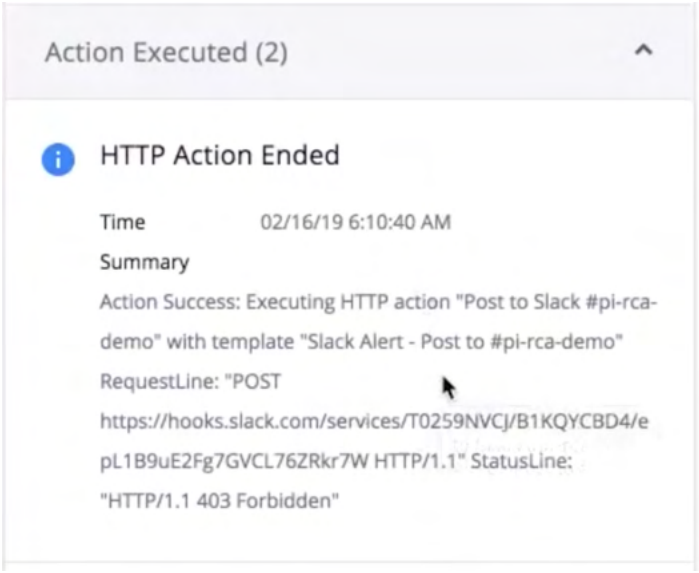
These are options of the standard troubleshooting flow. They are typically done as follow-up.

In this example:

- Suppose we want more context for the Process CPU Burnt issue on PaymentService1. We can view snapshots of transactions affected by that issue. Double-click a snapshot in the list to open it in its own pane, and if desired, drill down into details and call graphs.



- It is common to send messages to a ticketing system when an anomaly occurs. In this case, we posted to Slack, for our Ops team to see that on their phones.



Health Rules

Related pages:

- Configure Health Rules
- Troubleshoot Health Rule Violations
- Policies
- Actions

This page provides an overview of health rules, the policy statements that define triggers in AppDynamics policies.

What is a Health Rule?

Health rules let you specify the parameters that represent what you consider normal or expected operations for your environment. The parameters rely on metric values, for example, the average response time for a business transaction or CPU utilization for a node.

When the performance of an entity affected by the health rule violates the rule's conditions, a health rule violation occurs. The health statuses are represented as critical, warning, normal, and unknown.

When the health status of an entity changes, a health rule violation event occurs. Examples of a health rule violation:

- starting
- ending
- upgrading from warning to critical or
- downgrading from critical to warning

The health statuses of entities and health rule violations are surfaced in the controller user interface. A health rule violation event can also be used to trigger a policy, which can initiate automatic actions, such as, sending alerting emails or running remedial scripts.

You create health rules using the health rule wizard, described in Configure Health Rules. The wizard groups commonly-used system entities and related metrics to simplify setting up health rules. You can also use the default health rules provided by AppDynamics as-is, or modify them.

Default Health Rules

AppDynamics provides a default set of health rules for some products, such as applications and servers. These default health rules vary depending on the entity. To see the default rules, before any health rules have been added to your AppDynamics installation:

1. From the Alert & Respond tab, click **Health Rules**.
2. Select the entity.
   The default health rules for the selected entity are displayed.

If any of these predefined health rules are violated, the affected entities are marked in the UI as yellow-orange if it is a Warning violation and red if it is a Critical violation.

In many cases, the default health rules may be the only health rules that you need. You can edit and customize the health rules to suit your application. You can also disable the default health rules.

Health Rule Scopes

The health rule scope determines the set of default health rule types. You can choose the scope to get a set of default health rule types for applications, servers, or databases. For example, when you define a mobile application as the scope, the default health rules such as crash rates and HTTP/network error rates are displayed. Similarly, if you define the health rule scope for an application, the health rules would be for business transactions, CPU/memory utilization, and so on.

From **Alert & Respond > Health Rules**, you can select one of the following health rule scopes from the drop-down list:

- Applications
- User Experience: Browser Apps
- User Experience: Mobile Apps
- Databases
- Servers
- Analytics

You can also create new health rules to add to the default set for each scope. You may want to add the health rule *app starts* to your mobile application. This health rule is not part of the default set of health rules in the mobile app scope, so you would just need to add a new health rule.

Heath Rule Types

The health rule wizard groups health rules into types that are categorized by the entity that the health rule covers. This allows the wizard to display appropriate configuration items during the health rule creation.

The health rule types are:

- Transaction Performance
  - Overall Application Performance: Groups metrics related to load, response time, slow calls, stalls, with applications.
  - Business Transaction Performance: Groups metrics related to load, response time, slow calls, stalls, so on, with business transactions.
- Node Health
  - Node Health-Hardware, JVM, CLR: Groups metrics like CPU and heap usage, disk I/O, so on, with nodes.
  - Node Health-Transaction Performance: Groups metric related to load, response time, slow calls, stalls, so on, with nodes.
  - Node Health-JMX: Java only, groups metrics related to connection pools, thread pools, so on, with specific JMX instances and objects in specific nodes and tiers.
- User Experience-Browser Apps
  - Pages: Groups metrics like DOM building time, JavaScript errors, so on, with the performance of application pages for the end-user.
  - IFrames: Groups metrics like first-byte time, requests per minute, so on, with the performance of iFrames for the end-user.
  - AJAX Requests: Groups metrics like Ajax callback execution time, errors per minute, so on, with the performance of Ajax requests for the end-user.
  - Virtual Pages: Groups metrics like End User Response Time, Digest Cycles, HTML Download Time, DOM Building Time, etc. for virtual pages created with Angular. See AngularJS Support for information on what these metrics mean in the context of virtual pages.
- User Experience-Mobile Apps
  - Mobile Apps: Groups metrics related to mobile app crashes, starts, and server calls as well as network requests and errors.
  - Network Requests: Groups metrics like HTTP and network errors, request time, and requests per minute with network requests.
- Servers: Groups metrics related to hardware resources.
- Databases & Remote Services: Groups metrics related to response time, load, or errors with databases and other backends.
- Advanced Network: Groups metrics related to Network Visibility, such as PIE (performance impact events), zero window, data retransmission, and errors.
- Error Rates: Groups metrics related to exceptions, return codes, and other errors with applications or tiers.
- Information Points: Groups metrics like response time, load, or errors with information points.
- Service Endpoints: Java and .NET only; groups metrics like average response time, calls per minute, and errors per minute with service endpoints.
- Custom: Presents all the metrics collected by the agent that could affect a single business transaction, a single node or overall application performance. Use this type to create rules that evaluate custom metrics.

When you select one of these health rule types, the wizard offers you the metrics commonly associated with that type in an embedded browser.

How to Set Up Health Rules?

AppDynamics recommends the following process to set up health rules for your application:

1. Identify the key metrics (performance indicators) on the key entities that you need to monitor.
2. Click **Alert & Respond > Health Rules** to examine any default health rules that are provided by AppDynamics.
   - Compare your list of metrics with the metrics configured for the default rules.
   - You can view the list of affected entities for each of the default health rules and modify them. See Configure Affected Entities.
   - If the default health rules cover all the key metrics you need, determine if the pre-configured conditions are applicable to your environment. If required, modify the conditions.

     Define a metric expression to evaluate complex criteria for a condition.

     Define a boolean expression to evaluate multiple health rule conditions.
3. If default health rules do not cover all your requirements or if you need finely-applied health rules to cover specific use cases, create new health rules.
   a. Identify the type of health rule that you want to create. See Health Rule Types.
   b. Decide which entities are affected by the new rule. See Entities Affected by a Health Rule.
   c. Define the conditions to monitor. See Create and Configure Conditions.
4. If you want the health rules to be evaluated according to a pre-defined time schedule, create a health rule schedule. In some situations, a health rule is more useful if it is evaluated at a particular time. See Health Rule Schedules.

After you set up health rules you must configure policies and actions to be executed when health rules are violated. See Policies and Actions.

Additional Considerations

- Your application status is based on health rules for the current time range. If you disable old health rule policies or enable new ones, you might see errors in red in your application status, even if there are no current critical events based on the new policies. To verify that your new or disabled health rule policies have taken effect, change the time range in your dashboard to a smaller, more recent time frame.

- When you are configuring health rules for business transactions with baseline selected in the configured condition with a very fast average response time (ART) such as 25 ms, using standard deviation as a criterion can cause the health rule to be violated too frequently. The health rule may violate too frequently because a tiny increase in response time can represent multiple standard deviations. In this case, consider adding a second condition that sets a minimum ART as a threshold. For example, if you do not want to be notified unless ART is over 50 ms, you could set your threshold as ART > 2 Standard Deviations and ART > 50 ms.

- Similarly, when configuring health rules for calls-per-minute (CPM) metrics with baseline selected in the configured condition, the health rule may never be violated if the condition is using standard deviations, and the resulting value is below zero. In this case, consider adding a second condition that checks for a zero value, such as CPM < 2 Standard Deviations and CPM < 1.

Health Rule Schedules

Related Pages:

- Health Rules
- Configure Health Rules
- Troubleshoot Health Rule Violations
- Policies
- Actions

This page provides an overview of health rule schedules in AppDynamics.

The metrics associated with a health rule are evaluated according to a schedule that you control. You can configure:

- when a health rule is in effect
- which data set should be used, based on time
- what special rules should be in place during a violation event

Time evaluation for health rule schedules is based on the time zone you specify, regardless of the time zone of the Controller.

> For schedules created in AppDynamics controller version 4.5.15 or earlier, all schedules are executed based on the controller time zone. For schedules created in version 4.5.16 or later, schedules are executed based on the time zone you specify, irrespective of the controller time zone.

Health Rule Enabled Schedule

By default, health rules are always enabled. You can define schedules for evaluation of the health rules.

Built-in schedules exist for:

- End of business hours
- Weekday lunch
- Weekday mornings
- Weekdays
- Weekends
- Weeknights

You can also configure custom health rule schedules. The custom schedule allows you to define a time zone specific to your application, independent of the controller time zone. This eliminates the need to adjust the time zone manually if your application is in a separate time zone than that of the controller. For more details, see Create and Manage Health Rule Schedules.

Health Rule Evaluation Time Frame

The health rule evaluation time frame is the period of time over which the data used to evaluate the health rule is collected.

Different kinds of metrics provide better results using different sets of data. You can manage how much data AppDynamics uses when it evaluates a particular health rule by setting the data collection time period. You can define an evaluation time frame between 1 minute to 120 minutes. The default value is 30 minutes. You can select the following values in the **Use data from last** drop-down:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120

> You can define a persistence threshold for a condition only if you have defined the evaluation time frame of 30 mins or less.

Health Rule Wait Time After Violation

The health rule wait time setting lets you control how often an event is generated while the conditions found to violate a health rule continue. If the controller determines that a health rule has been violated, with a status of either Critical or Warning, an Open Critical or Open Warning event is generated. This event is used to trigger any policies that match the health rule and initiate any actions that the policies require.

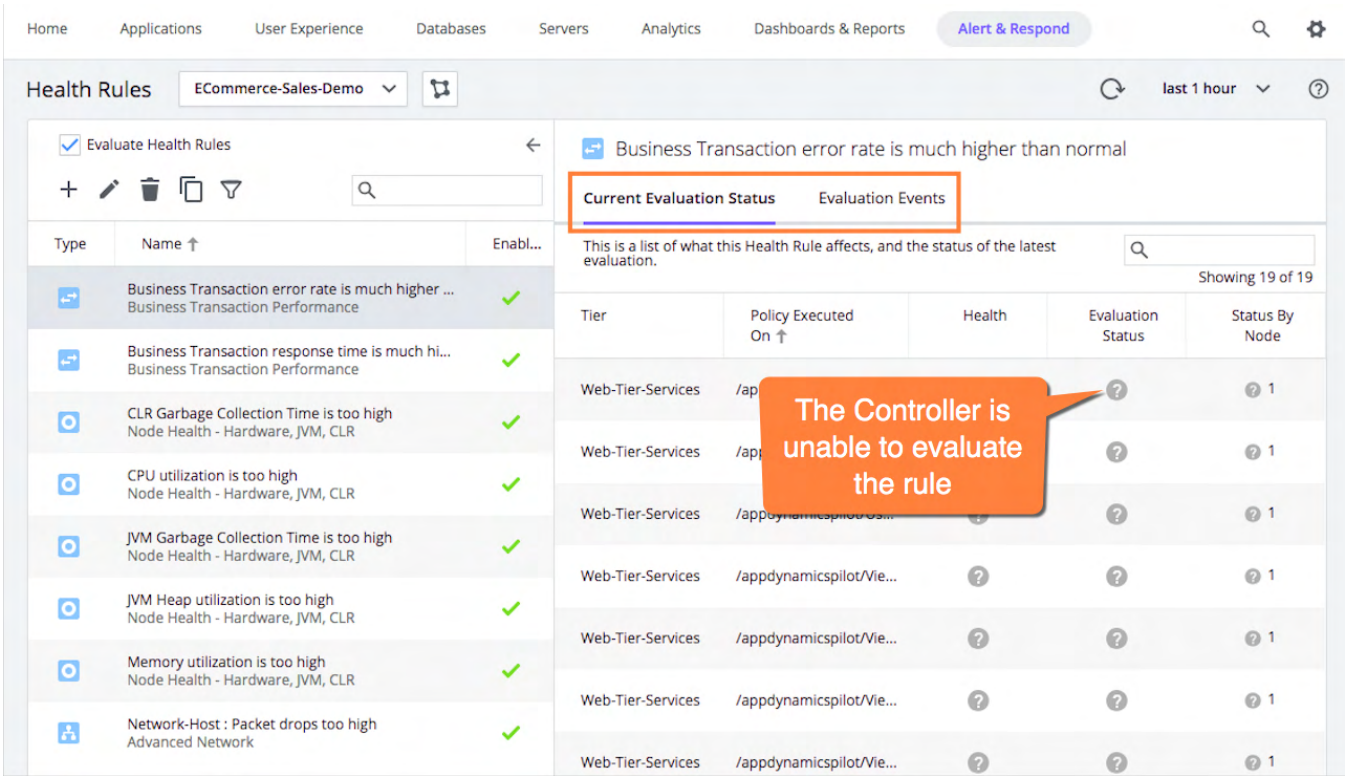
Once an Open event has occurred, the controller continues to evaluate the status of the health rule every minute. If the controller continues to detect the same violation, the violation remains open with the same status. A corresponding **Continues Critical** or **Continues Warning** event may be generated to link to any related policy.

A **Continues** event every minute might be too noisy for your health rule. The health rule's **Wait Time after Violation** setting is used to throttle how often these Continues events are generated for continuing health rule violations. The default is every 30 minutes.

To use Continues Critical and Continues Warning events, adjust the default **Wait Time after Violation** value to the desired frequency. Then configure a policy matching the health rule with the *Health Rule Violation Continues - Warning and/or Health Rule Violation Continues* - Critical events selected in the **H ealth Rule Violation Events** section of the policy settings.

The violations displayed in the **Health Rules Violations** page, under **Troubleshoot**, are updated only when a health rule violation event is triggered.

If the Controller is unable to evaluate the rule—for example, if a node stops reporting—the Evaluation Status of the health rule is marked as a grey question mark or Unknown in the **Current Evaluation Status** tab in the right panel of the health rules list. The current violation event remains open until the Wait Time after Violation period has elapsed, at which point the violation event is closed and a new event is triggered, causing the Health status itself of the rule to display as *Unknown*.



Health Rule Entities

Related Pages:

- Health Rules
- Configure Health Rules
- Troubleshoot Health Rule Violations
- Policies
- Actions

This page provides an overview of health rule entities in AppDynamics.

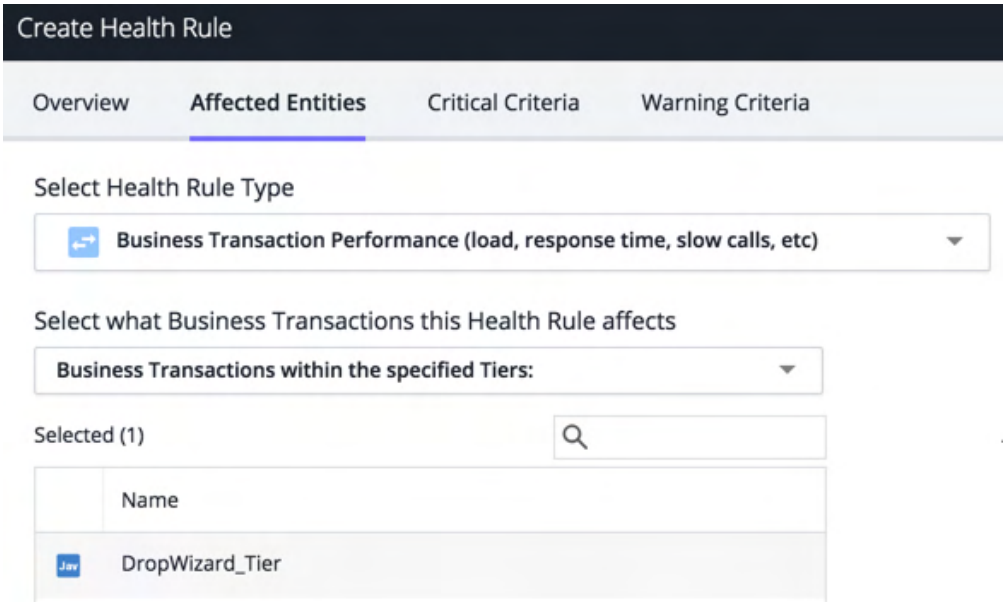
A health rule can evaluate metrics associated with an entire application or a limited set of entities. For example, you can create business transaction performance health rules that evaluate certain metrics for all Business Transactions in the application or node health rules that cover all the nodes in the application or all the nodes in specified tiers. The default health rules are in this category.

You can also create health rules that are applied to a limited set of entities in the application, or even a single entity such as a node or a JMX object or an error. For example, you can create a JMX health rule that evaluates the initial pool size and number of active connections for specific connection pools in nodes that share certain system properties.

Monitoring Serverless Entities

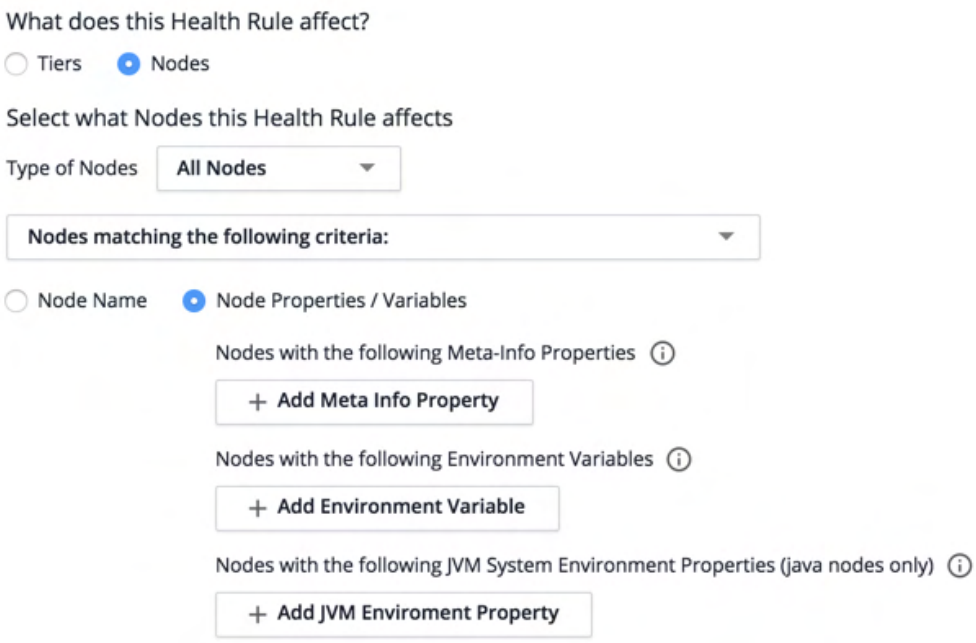
Serverless functions are tracked at the tier level. A serverless function is indicated by a lambda () icon inside each tier. When you configure a health rule for an application comprising serverless entities, you can choose to monitor the serverless tiers in the **Affected Entities** tab. For information on how various health rules are evaluated for serverless entities comprising tiers for AWS Lambda, see Evaluating Serverless Tiers.

The health rule wizard lets you specify which entities the health rule affects, enabling the creation of very specific health rules. For example, for a Business Transaction, you can limit the tiers that the health rule applies to, or limit the health rule application to specific business transactions by name or by names that match certain criteria.



For node health rules, you can specify the type of the node, such as Java, .NET, PHP, and so on.

You can specify that a health rule applies only to nodes that meet certain criteria.

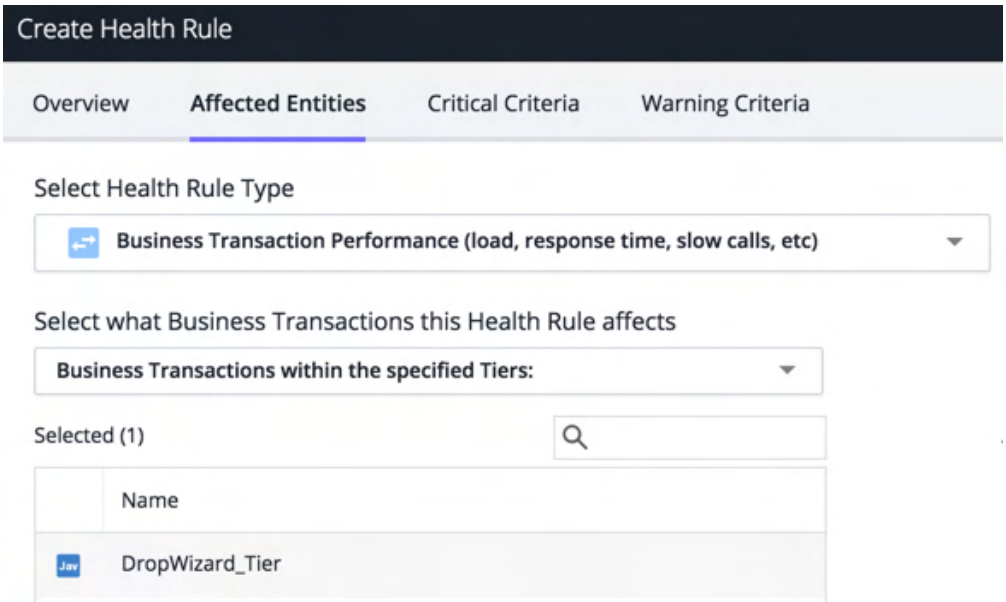


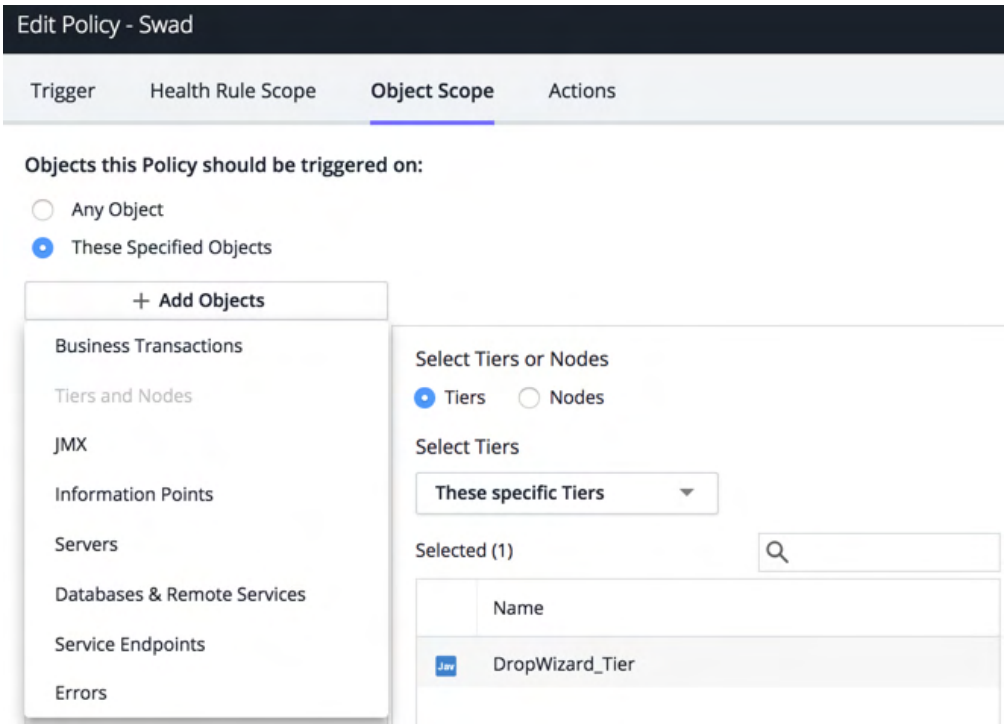
The **Type of Node** pulldown menu does not allow you to specify Node.js, Python, or Web Service nodes. To restrict a health rule to these types of nodes, you can specify the affected entity as a tier and then select only Node.js or Python or Web Service tiers as needed. Or to more finely-tune the affected nodes, use the **Nodes matching the following criteria** menu item to specify node names or matching environment variables or meta-info to restrict the health rule to the nodes you want.

Entities Affected by a Health Rule

For an **Overall Application Performance Health Rule** type, the health rule applies to the entire application, regardless of the business transaction, tier, or node.

If you configure your Health Rule to work with tiers, you must also configure the parallel policy to work with tiers. However, if you configure your Health Rule to work with tiers, but your policy is configured with nodes first, you will not trigger any actions or notifications. The inverse is also true. The following screenshots show examples of a health rule and a policy created in the correct order.





The following table lists the entities that you can apply health rules to.

Health Rule Type	Applicable Entities
Business Transaction Performance	• All Business Transactions in the application • All Business Transactions within tiers that you select • All Business Transactions within serverless tiers that you select for a serverless platform* • Individual Business Transactions that you select • Business Transactions with names that have patterns matching criteria that you specify (such as all Business Transactions with names that start with "INV")
Custom	• A business transaction that you specify • A node that you specify • Overall application performance
Databases & Remote Services health	• All databases and remote services in the application • Individual databases and remote services that you specify • Databases and remote services with name matching criteria that you specify
Error Rates	• All Errors in the application • Specific error types that you select • Errors with the specified tiers • Errors within the specified serverless tiers for serverless platform* • Errors with names that have patterns matching criteria that you specify
Information Points	• All servers in the application • Information points that you specify • Information points with names matching criteria that you specify

Node Health—JMX	• All JMX instance names (MBeans) in the application • Specific MBeans • MBeans on certain nodes • Specific JMX objects • All nodes in the application • Nodes within specified tiers • Individual nodes that you specify • Nodes with names, meta-data, environment variables or JVM system environment properties with matching criteria that you specify • Nodes with certain MBeans
Node Health—Transaction Performance or Node Health—Hardware, JVM, CLR	• All tiers in the application • Individual tiers that you specify • All serverless tiers in the application for a serverless platform* • Individual tiers that you specify for a serverless platform* • All nodes in the application • Nodes types, such as Java nodes, PHP nodes, and so on • Nodes within specified tiers • Individual nodes that you specify • Nodes with names, meta-data, environment variables or JVM system environment properties with matching criteria that you specify The performance of serverless tiers is not evaluated for **Tier/Node Health (Hardware)** health rules. AWS does not offer node-level dashboards or metrics because the serverless platform runtime instances spin up and down on demand.
Server health	• All servers in the application • Servers that you specify • Servers in specific tiers
Service Endpoint	• All service endpoints in the application • Service endpoints that you specify • Service endpoints with names matching criteria that you specify • Service endpoints within specified tiers • Service endpoints within specified serverless tiers for serverless platform*
User Experience - Mobile Apps	• All mobile apps with the specified app key • The specified mobile apps • Mobile apps matching the given criteria
User Experience - Mobile Network Requests	• All network requests in the application • Network requests that you specify • Network requests with names matching criteria that you specify • Network requests of the specified mobile apps
User Experience - Browser Apps—Pages, iframes, Ajax Requests, Virtual Pages, Synthetic jobs	• All such entities • Entities that you specify • Entities with names matching criteria that you specify

Health Rule Conditions

Related Pages:

- Health Rules
- Configure Health Rules
- Troubleshoot Health Rule Violations
- Policies
- Actions

You define the acceptable range for a metric by establishing health rule conditions. A health rule condition sets the metric levels that constitute a **Warning** s tatus or a **Critical** status.

A condition consists of a Boolean statement that compares the current value of a metric against one or more static or dynamic thresholds based on a selected baseline. If the condition is true, the health rule violates. The rules for evaluating a condition using multiple thresholds depend on configuration.

Static thresholds are straightforward. For example, is a business transaction's average response time greater than 200 ms? The condition is evaluated to 'true' if the average response time is greater than 200 ms and the health rule violates.

Dynamic thresholds are based on a percentage in relation to, or a standard deviation from, a baseline built on a rolled-up baseline trend pattern. A daily trend baseline rolls up values for a particular hour of the day during the last thirty days, whereas a weekly trend baseline rolls up values for a particular hour of the day, for a particular day of the week, for the last 90 days. For more information about baselines, see Dynamic Baselines.

You can define a threshold for a health rule based on a single metric value or on a mathematical expression built from multiple metric values.

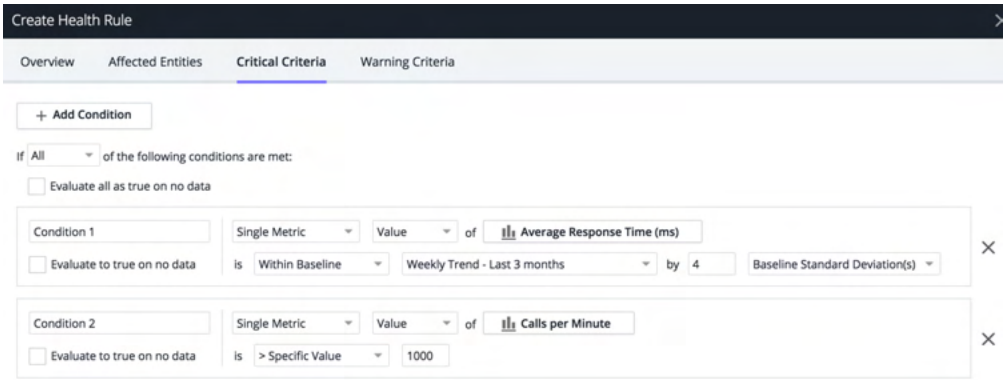
The following are typical health rule conditions:

- If the value of the Average Response Time is greater than the default baseline by 3 X the Baseline Standard Deviation . . .
- If the count of the Errors Per Minute is greater than 1000 . . .
- If the number of MB of Free Memory is less than 2 X the Default Baseline . . .
- If the value of **Errors per Minute/Calls per Minute** over the last 15 days > 0.2 . . .
  This example combines two metrics in a single condition. You can use the expression builder embedded in the health rules wizard to create conditions based on a complex expression comprising multiple interdependent metrics.
- If the (average response time > baseline OR errors per minute > baseline) **AND** (calls per minute > the defined threshold) . . .
  This example uses multiple conditions to evaluate the health rules. You can use the 'CUSTOM' option to define a boolean expression to evaluate the conditions.

Critical and Warning Conditions

Conditions are classified as either critical or warning conditions.

Critical conditions are evaluated before warning conditions. If you have defined a critical condition and a warning condition in the same health rule, the warning condition is evaluated only if the critical condition is not true.



The configuration procedures for critical and warning conditions are identical, but you configure these two types of conditions in separate panels. You can copy a critical condition configuration to a warning configuration and vice-versa and then adjust the metrics in the copy to differentiate them. For example, in the **Critical Condition** panel you can create a critical condition based on the rule:

- If the Average Response Time is greater than 1000

Then from the Warning Condition panel, copy that condition and edit it to be:

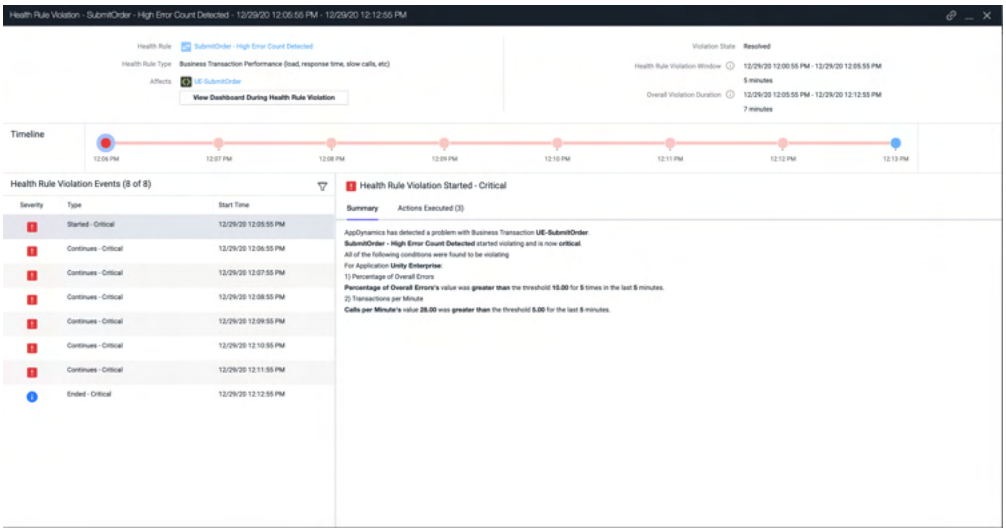
- If the Average Response Time is greater than 500

As performance changes, a health rule violation can be upgraded from warning to critical if performance deteriorates to the higher threshold or downgraded from critical to warning if performance improves to the warning threshold.

Health Rule Violation Event

When metric levels exceed the acceptable range, conditions violate, a health rule violation event occurs. The details of the violation event are displayed on the Health Rule Violation window. This window displays the following details:

- Number of violation events
- Summary of each violation event
- Details of actions initiated in response to the violation event
- Timeline of each violation event

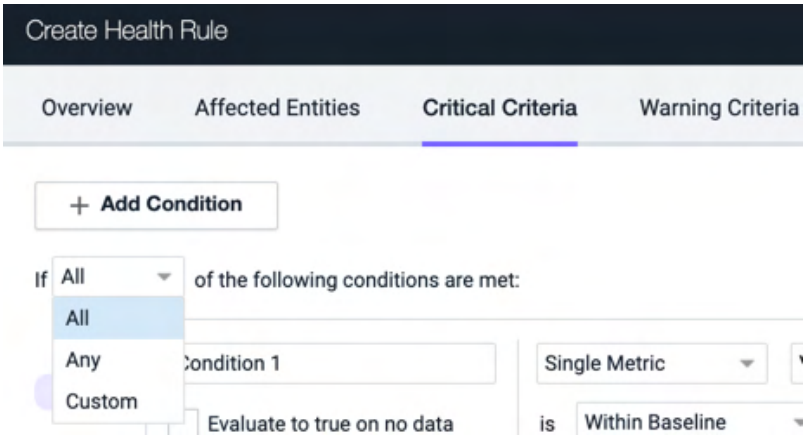


When you define a trigger for the condition, the health rule violation event summary does not display any metric value as there is no single value. However, when you do not define any trigger for the condition, the health rule violation event summary displays the metric value.

Evaluation Criteria

When you define multiple conditions for a health rule, they are evaluated based on the criteria you define. You can use the following options to define the evaluation criteria:

- All: the health rule violates if all the conditions defined in the criteria evaluate to 'true'.
- Any: the health rule violates if one of the conditions defined in the criteria evaluates to 'true'.
- Custom: the health rule violates if the boolean expression with multiple conditions evaluates to 'true'.



For information on how to configure evaluation criteria, see Configure Health Rule Evaluation Criteria.

The following table uses examples to illustrate how a health rule is evaluated based on the criteria and when is it considered to violate.

Health Rule Configuration	Evaluation	Example
Single condition	the condition evaluates to 'true'	A health rule that compares 'average response time' with a defined baseline.

Multiple conditions with 'ANY' evaluation criteria	one of the health rule conditions evaluates to 'true'	A health rule that monitors the health of business transaction may measure any of the following performance metrics: • average response time or • errors per min
Multiple conditions with 'ALL' evaluation criteria	all of the health rule conditions evaluate to 'true'	A health rule that monitors the health of business transaction measures all of the following metrics: • response time • average response time greater than a baseline value, correlated with the application load For example, 50 concurrent users on the system. A policy is defined such that a remedial action is initiated only if the load (calls per minute) is high although the response time threshold is reached. The first part of the condition evaluates the response time and the second part ensures that the health rule is violated only when there is sufficient load.
Multiple conditions with 'CUSTOM' evaluation criteria	the boolean expression with multiple conditions evaluates to 'true' ⓘ The condition is evaluated only if a valid combination of conditions using AND and OR operators is entered, else the evaluation fails.	A health rule that monitors the health of a Business Transaction, measures the performance based on the following conditions: • (average response time greater than baseline OR errors per min greater than baseline) AND • (calls per min greater than threshold)

Persistence Thresholds

Temporary spikes in metric performance data is a major cause of false alerts. Persistence thresholds allow you to define a 'sensitivity level' for a health rule and thereby reduce the number of false alerts. You can define the 'number of times metric performance data should exceed the defined threshold during the evaluation time frame' to constitute a violation and subsequently trigger an alert.

ⓘ You can define a persistence threshold for a condition only if you have defined an evaluation time frame of 30 minutes or less.

For example, when monitoring the CPU utilization, you would not want to be reported of a single violation (section A in the figure) of the threshold. However, if the violation of threshold continues to occur multiple times (section B in the figure) during the evaluation time period, you would want to be alerted.

Alert Sensitivity Tuning

> AST feature is enabled only if you create a health rule to monitor a business transaction, a service endpoint, or a remote service.

SaaS

It is important that you configure conditions appropriately to ensure that you do not miss any alerts or receive false alerts instead. With 'Alert Sensitivity Tuning' (AST), you can view historical data for metri and baselines when you configure conditions. This data helps visualize the impact of the configuration you define and assists in fine-tuning the configuration.

You can view a graphical representation of the metric data, threshold value, standard deviation, and the baseline. The graphical view is instantly updated when you update any configuration. You can also view granular details by modifying the graphical view. To view granular details, you can:

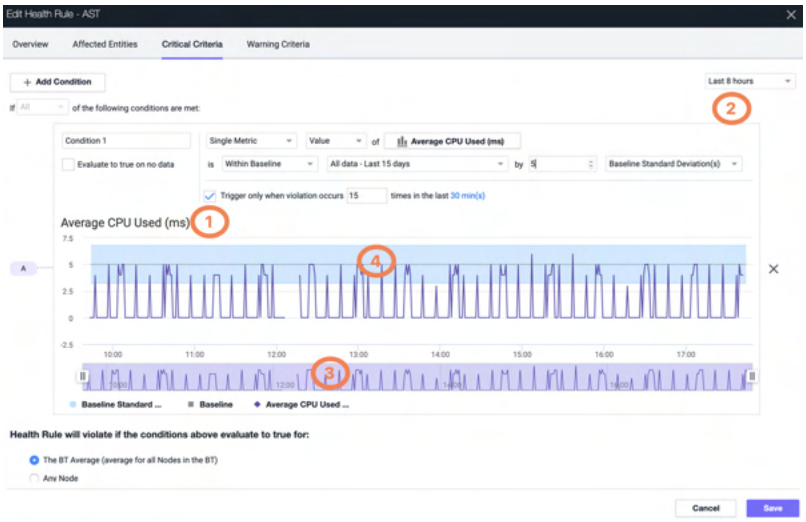
- increase the time period of the data capture to 1 day or 3 days.

  > If data is not available for the selected time period, only available data is presented.

- adjust the time range in the graph to view the metric data details.
- hover over the metric data to view metric and baseline values at any given time.

You can analyze the data presented and then make adjustments to your configuration accordingly. For more information on fine-tuning a condition, see Cre ate a BT Health Rule and Fine-tune Metric Evaluation.

For example, if you select **Average CPU Used (ms)** (1) as a metric to be monitored for a health rule condition, you can view the past metric data for a time period of 8 hours (2). The graphical representation of the metric data indicates the baseline (3) and the baseline standard deviation (4). Based on this data, you can fine-tune the evaluation of the condition so that you avoid false alerts and receive the alerts only when the health rule violates the conditions you define.



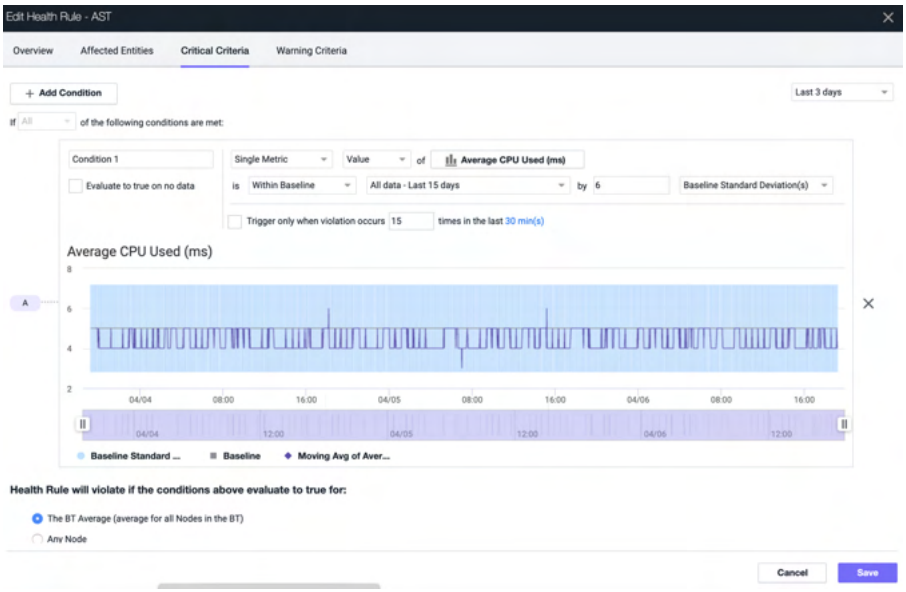
Moving Average

If you define a persistence threshold to evaluate a condition, the metric data for every minute is compared to the baseline and plotted on the AST graph. However, if you do not define the persistence threshold, a 'moving average' for the selected metric is plotted as follows:

1. Depending on the '**Use data from last**' value 'X', the metric data for 'X' minutes is considered.
2. On the 'X+1'th minute, the average of the past 'X' minutes is computed and denoted as the first point on the graph.
3. Similarly, the average for the following 'X' minutes is computed and points are denoted on the graph for the rest of the time range.

Why Use Moving Average?

Unless persistence thresholds are used, health rules compare the moving average of a metric to a threshold or a baseline. Thus, representing the moving average in the graph is appropriate. For more information, see Create a Health Rule and Fine-tune Metric Evaluation.



How are Conditions Evaluated if No Data is Reported?

The Evaluate to true on no data option controls the evaluation of the condition in cases where any metric on which the condition is based, does not return any data. The condition evaluates to 'unknown' (default) when no data is returned. If the health rule is based on all the conditions evaluating to true, having no data returned may affect whether the health rule triggers an action.

When you define a health rule evaluation time frame, reference data is collected for each data point. If the configured metric fails to report data during the time frame, the health rule condition is evaluated as follows:

Evaluate to true on no data	Trigger only when violation occurs x times in the last y min (s)	Condition Evaluation
Enabled	Enabled	The condition is evaluated for each data point in the evaluation time frame. The condition evaluates to 'true' when metric fails to report any data for a given data point. For example, when you set the persistence threshold, X = 3 for an evaluation time frame, Y = 5. This means that 5 data points are required to evaluate the condition. Data is reported for 4 data points, no data is reported for 1 data point and the metric exceeds the threshold twice. The condition evaluates to 'true' for the minute when no data is reported.
Enabled	Disabled	The condition does not evaluate to 'true' if a metric fails to report data for any data point during the evaluation time frame.
Disabled	Disabled	The condition does not evaluate to 'true' if the configured metric fails to report data for any data point during the evaluation time frame.

Custom Boolean Expression

A condition consists of single or multiple statements that evaluate different metrics. You can define a single condition or multiple conditions to evaluate the performance metrics of your application. When you define multiple conditions, you may want to define an evaluation criteria using a boolean expression.

Advantages of using a boolean expression are:

- eliminates the need to create multiple health rules to monitor various performance metrics. Using a boolean expression allows you to evaluate complex criteria for multiple conditions in one go.
- well-calibrated boolean expression ensures reduced false alerts.
- easy to create and maintain health rules with complex evaluation criteria using simple condition names. Conditions are named as **A**, **B**, **C** and so on.
- allows the use of `AND` and `OR` operators to define a highly complex boolean expression. You can use a maximum of 8 operators in your boolean expression.

Evaluation Scope

The health rule evaluation scope defines how many nodes in the affected entities must violate the condition before the health rule is considered violated.

Evaluation scope applies only to business transaction performance type health rules and node health type health rules in which the affected entities are defined at the tier level.

For example, you may have a critical condition in which the condition is unacceptable for any node, or you may want to consider the condition a violation only if the condition is true for 50% or more of the nodes in a tier.

Options for this evaluation scope are:

- The tier average: Evaluation is performed on the tier average instead of the individual nodes.
- Any node: If any node exceeds the thresholds, the rule is violated.
- Percentage of the nodes: If x% of the nodes exceed the thresholds, the rule is violated.
- Number of nodes: If x nodes exceed the thresholds, the rule is violated.

Health Rule Management

Related Pages:

- Health Rules
- Configure Health Rules
- Troubleshoot Health Rule Violations
- Policies
- Actions

This page provides an overview of health rule management in AppDynamics.

To view current health rules, including the default health rules, and to access the health rule wizard, click **Alert & Respond > Health Rules**. Then choose the type of entity for which you want health rules from the pulldown menu at the top.

Current health rules are listed in the left panel. If you click one of these rules, a list appears in the right panel showing which entities this selected health rule affects and what the status of the latest evaluation is. You can also select the **Evaluation Events** tab to see a detailed list of evaluation events.

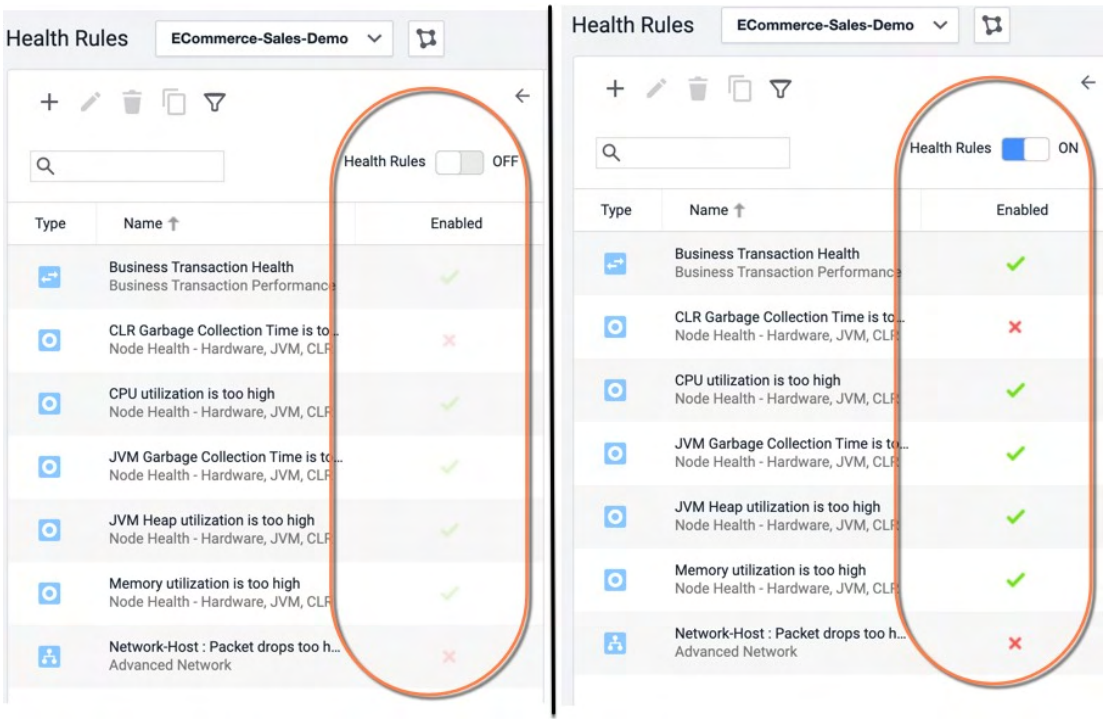
In the left panel, you can directly delete or duplicate a health rule. From here you can also access the health rule wizard to add a new rule or edit an existing one.

You can turn off the evaluation of all health rules in the selected entity by clearing the **Evaluate Health Rules** checkbox. Check it when you want health rule evaluation to start again.

See Configure Health Rules for details on using the health rule wizard.

Disable All Health Rules

You can disable the evaluation of all the health rules for a selected entity by setting the **Health Rules** toggle **OFF**. The **Health Rules** UI indicates the status of all the health rules.



To resume the health rule evaluation, set the toggle **ON**.

- You can disable the evaluation of an individual health rule in the **Create Health Rule** or **Edit Health Rule** dialog box by clearing the **Enabled** chec kbox.
- When you disable the evaluation of all health rules, individual health rules are not evaluated even if you select the **Enabled** checkbox in the **Creat e Health Rule** or **Edit Health Rule** dialog box.

See Configure Health Rules for details on using the health rule wizard.

View Health Rule Status

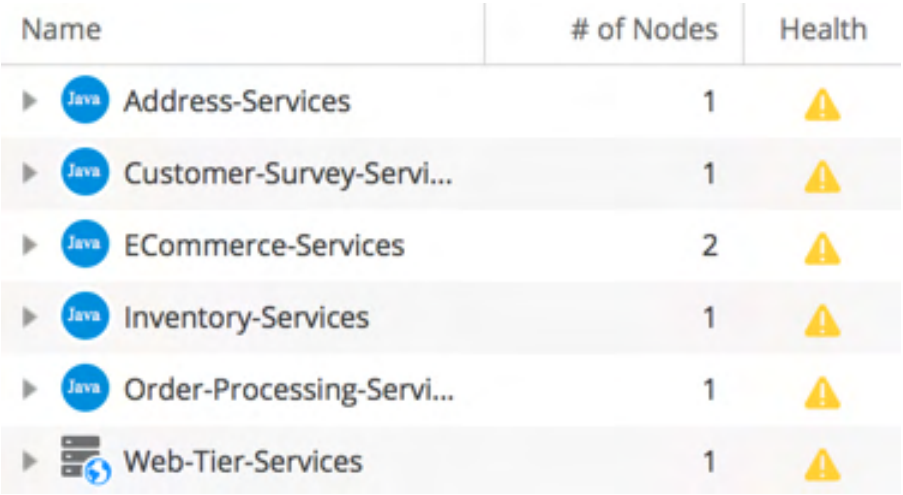
Across the UI, health rule status is color-coded:

- Green is healthy
- Yellow/orange is a warning condition
- Red is a critical condition
- Grey indicates that the status of the health rule is unknown (for example, if the Controller cannot gather the data necessary to evaluate the rule)

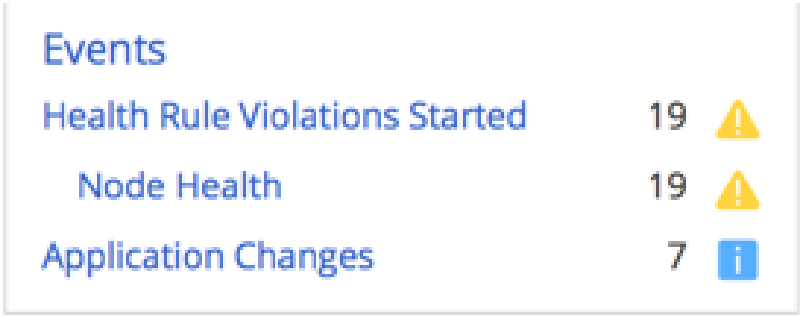
If you see a health rule violation reported in the UI, you can click it to get more information about the violation.

Here are the health summary bars on the built-in dashboards:

A health column is displayed in various lists, such as the tier list below:



In the dashboards, health rule violations are displayed in the Events panel.



Configure Health Rules

Related pages:

- Health Rules
- Configuration Import and Export API
- Metric Browser

This topic describes the detailed steps for configuring health rules using the health rule wizard. For more information on these settings, see Health Rules.

Permissions

To create, edit, or delete health rules, you need the Configure Health Rules permission. For more information, refer to Application Permissions.

Structure of the Health Rule Wizard

The health rule wizard contains four panels:

- Overview: Sets the health rule name, enabled status, the health rule schedule, evaluation period of the health rule data, and wait time post violation.
- Affected Entities: Sets the entities evaluated by the health rule. The options presented vary according to the health rule type you have defined.
- Critical Criteria: Sets the conditions, whether all or any of the conditions need to be true for a health rule violation to exist, and the evaluation scope—business transaction and node health policies defined at the tier level only—it also includes an expression builder to create complex expressions containing multiple metrics.
- Warning Criteria: Settings are identical to Critical Criteria, but configured separately.

You can navigate among these panels using the Back and Next buttons at the bottom of each panel or by clicking the panels in the wizard. You should configure the panels consecutively because the configuration of the health rule type determines the available affected entities in the Affected Entities panel as well as the available metrics in the Criteria panels.

> **Important**
>
> You must configure the panels consecutively because the configuration of the health rule type determines the available affected entities in the **Af fected Entities** panel as well as the available metrics in the **Criteria** panels.

Create and Configure a Health Rule

The following table outlines the steps required to create and configure a health rule:

1	Create a health rule.
2	Configure affected entities.
3	Configure health rule evaluation criteria.
4	Create and configure Conditions.

Create a Health Rule

Related Pages:

- Health Rules
- Configuration Import and Export API
- Metric Browser

You can use the default health rules provided by AppDynamic, modify them to map to your requirements, or define a custom health rule.

Access the Health Rule Wizard

1. Click **Alert & Respond** in the menu bar.
2. Click **Health Rules** either in the right panel or the left navigation pane.
3. Select the context for the health rule from the pulldown menu.
4. Do one of the following:
    - To create a new health rule, click the **+** icon.
    - To edit an existing health rule, select the health rule and click the **Edit** (pencil) icon.
    - To remove an existing health rule, select the health rule and click the **Delete** icon.

Configure Heath Rule Details

You configure health rule details in the **Overview** panel.

1. Enter a name. If a name already exists, you can change it.
2. Check **Enabled** to enable the rule, clear the checkbox to disable it.
3. The Always option is pre-selected in the **When is the rule enabled?** drop-down list. If the health rule is enabled only at certain times, select other predefined schedules from the **When is the rule enabled?** drop-down list.

    > To define a custom health rule schedule or modify the predefined time intervals, click **Manage Health Rule Schedules**. See Create and Manage Health Rule Schedules.

4. Click the drop-down list **Use data from last <> min(s)** and select a number between 1 and 120 minutes. The value you specify is the latest time interval during which data is collected to determine if there is a health rule violation. This value applies to both critical and warning criteria. See Health Rule Evaluation Time Frame.

    > If you have defined a persistence threshold for the health rule condition, ensure that you define an evaluation time frame of 30 mins or less.

5. In the **Wait Time after Violation** field, enter the number of minutes to wait before re-evaluating the rule for the same affected entity in which the violation occurred. See Health Rule Wait Time After Violation.

After you configure the Health Rule details in the **Overview** panel, proceed with the following steps.

1. Configure the entities affected for the health rule, see Configure Affected Entities.
2. Define the evaluation criteria for the health rule, see Configure Health Rule Evaluation Criteria.
3. Save your configuration.

Create and Manage Health Rule Schedules

1. In the Overview panel of the Create Health Rule wizard, click **Manage Health Rule Schedules**. The **Manage Health Rule Schedules** window lists all the predefined time intervals.
2. Select a predefined schedule in the **Manage Health Rule Schedules** window and click the **Edit** icon to see the details of the predefined schedule.
3. To edit a predefined schedule for health rule evaluation:
    a. In the **Edit Schedule** window, make necessary changes.
    b. Click **Save**.

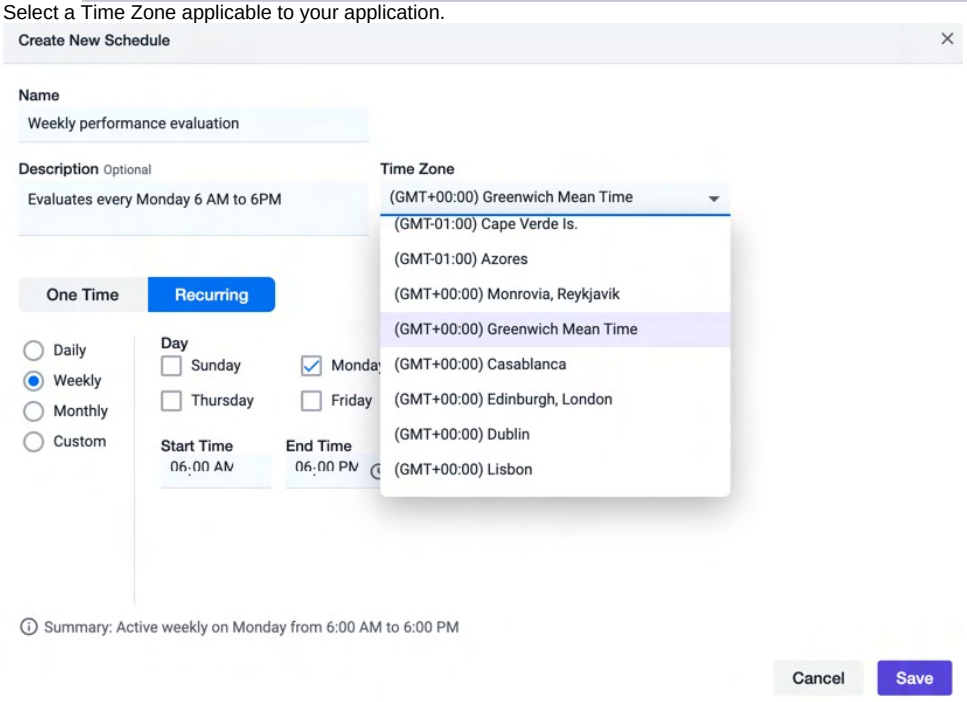
    > Schedules created in version 4.5.15 or earlier are set to 'Custom' type by default and the time zone is automatically set to the controller time zone.

4. To create a new health rule schedule:
    a. Click the **+** icon. The **Create New Schedule** window appears.
    b. Enter following details:
        i. a name for the schedule.
        ii. an optional description of the schedule.
    c. If you want to evaluate the health rule only once,
        i. Select One Time.
        ii. Enter the Start Date and Start Time.
        iii. Enter the End Date and End Time for the schedule.

d. If you want to define a recurring schedule to evaluate the health rules,
   - Select how often you want to evaluate the health rules:

Occurrence	Additional steps
Daily	i. Enter the Start Time and End Time.
Weekly	i. Select a specific day to execute the schedule. ii. Enter the Start Time and End Time.
Monthly	If you select a specific date of the month to execute the schedule: i. Enter the Start Date and Start Time. ii. Enter the End Date and End Time. For example, you might want to evaluate the health rules on the 1st of every month from 6.00 AM to the 5th until 6.00 PM. If you select a specific day of the month to execute the schedule: i. Select an occurrence and the Day. ii. Enter the Start Time and End Time. For example, you might want to evaluate the health rules on the first Monday of every month from 6.00 AM to 6.00 PM.
Custom	i. Enter a Start Date and Start Time for the schedule as a cron expression. ii. Enter an End Date and End Time for the schedule as a cron expression. For example, the following custom schedule specifies: a start time value of 0 0 13 ? * 2-6 and end time of 0 0 15 ? * 2-6 directing the health rule to be evaluated from 1 pm to 3 pm, Monday through Friday:

e. Select a Time Zone applicable to your application.



5. Save your configuration.

Delete Health Rule Schedule

To delete a health rule evaluation schedule, select the schedule in the **Manage Health Rule Schedules** window and click **Delete** – the bin icon at the top. Click **OK** to confirm the deletion.

Create a Health Rule and Fine-tune Metric Evaluation

To ensure that you do not miss alerts or receive false alerts, you must configure alerts appropriately. Alert Sensitivity Tuning (AST) helps configure alerts with appropriate sensitivity by providing historical data for the metric or the baseline being configured. AST also visualizes the impact of the alert configuration.

Create a Health Rule and Fine-tune Metric Evaluation

You can create a health rule to monitor the parameters of an application entity and fine-tune the sensitivity of a health rule using Alert Sensitivity Tuning (AST).

- You must be a SaaS customer.
- You must be monitoring a business transaction, a service endpoint, or a remote service (affected entity).

1. In the entity (BT, SEP, or remote services) UI, select an entity from the list.
2. Right-click the entity and click **Create new Health Rule**. The **Create Health Rule** UI is displayed.
3. Configure the health rule overview details. For more information, see Configure Heath Rule Details.
4. In the **Affected Entities** panel, select the entities to be monitored.
5. In the **Critical Criteria** panel, select **+Add Condition**.
6. Configure the health rule condition.
7. If required. select the **Warning Criteria** panel and define warning conditions as described in step 6.
8. Click **Save.** The health rule is saved.

Configure the Health Rule Condition

1. Enter a name for the condition in the first field of the condition row.
   This name is used in the generated notification text and in the AppDynamics console to identify the violation.
2. From the drop-down list next to the condition name field, select the single metric or metric expression to evaluate the condition.
   a. Single Metric:

Qualifier Type	Description
Minimum	The minimum value reported across the configured evaluation time length. Not all metrics have this type.
Maximum	The maximum value reported across the configured evaluation time length. Not all metrics have this type.
Value	The arithmetic average of all metric values reported across the configured evaluation time length. This value is based on the type of metric.
Sum	The sum of all the metric values reported across the configured evaluation time length.
Count	The number of times the metric value has been measured across the configured evaluation time length.
Group Count	The number of nodes contributing to a metric value, generally relevant for application or tier level metrics.
Current	The value for the current minute.

   i. From the **Value** drop-down list, select a qualifier for the metric from the following options:
   ii. Click **Select a Metric. Metric Selection** dialog box appears. The metric browser in the **Metric Selection** dialog box displays metrics appropriate to the health rule.
   iii. Select a metric from the list for the business transaction. Click **Select Metric**.
   A graphical view of the metric data for the last 8 hours appears.

   If you want to view the metric data for 1 day or 3 days, you can select the time period using the drop-down list.

   iv. From the drop-down list after the metric, select the type of comparison to evaluate the metric.

- To limit the effect of the health rule to conditions during which the metric is within a defined range—standard deviations or percentages—from the baseline, select **Within Baseline** from the drop-down list.
  To limit the effect of the health rule to when the metric is not within that defined range, select **Not Within Baseline**.
  Then select the following:
  -baseline to use
  -numeric qualifier of the unit of evaluation
  -unit of evaluation
  For example:

  ```
  Within Baseline of the Default Baseline by 3 Baseline Standard Deviations
  ```

- To compare the metric with a static literal value, select **< Specific value** or **> Specific Value** from the menu, then enter the specific value in the text field. For example:

  ```
  Value of Errors per Minute > 100
  ```

- To compare the metric with a baseline, **select < Baseline** or **> Baseline** from the drop-down list, and then select the following:
  - baseline to use
  - numeric qualifier of the unit of evaluation
  - unit of evaluation.
  You can use the Baseline Standard Deviation or Baseline Percentage as the unit of evaluation. For example:

  ```
  Maximum of Average Response Time is > Baseline of the Daily Trend by 3 Baseline
  Standard Deviations
  ```

  See Dynamic Baselines for information about the baseline options.

  > **Baseline Percentages**
  >
  > The *baseline percentage* is the percentage above or below the established baseline at which the condition will trigger. For example, if you have a baseline value of 850 and you have defined a baseline percentage of > 1%, the condition is true if the value is > `[850+(850x0.01)]` or `859`.

  To prevent health rule violations from being triggered when the sample sets are too small, these rules are not evaluated if the load—the number of times the value has been measured—is less than 1000. For example, if a very brief time slice is specified, the rule may not violate even if the conditions are met, because the load is not large enough.
  Depending on the baseline configuration you define, a graphical view of the metric data for the given baseline configuration is displayed. The graphical view is instantly updated when you update any configuration. You can also view granular details by zooming in on the graphical view. The metric data presented in the graph helps you calibrate the sensitivity of the metric evaluation.

b. Metric Expression:
   i. Select the **Metric Expression** option from the drop-down list and click **Add Expression**. The **Metric Expression** window appea rs that allows you to construct a mathematical expression to use as a metric. For example, the following expression is created to measure the number of calls per CPU: `Value of {calls} / {cpu} > 1`.
   ii. In the **Variable Declaration** panel of the Mathematical Expression builder, click **+ Add variable** to add a variable.
   iii. In the **Variable Name** field, enter a name for the variable.
   iv. From the drop-down list next to the **Variable Name** field, select the qualifier for the metric from the following options:

Qualifier Type	Description
Minimum	The minimum value reported across the configured evaluation time length. This qualifier is not available for all the metrics.
Maximum	The maximum value reported across the configured evaluation time length. This qualifier is not available for all the metrics.

Value	The arithmetic average of all metric values reported across the configured evaluation time length. This value is based on the type of the metric.
Sum	The sum of all the metric values reported across the configured evaluation time length.
Count	The number of times the metric value has been measured across the configured evaluation time length.
Group Count	The number of nodes contributing to a metric value, generally relevant for application or tier level metrics.
Current	The value for the current minute.

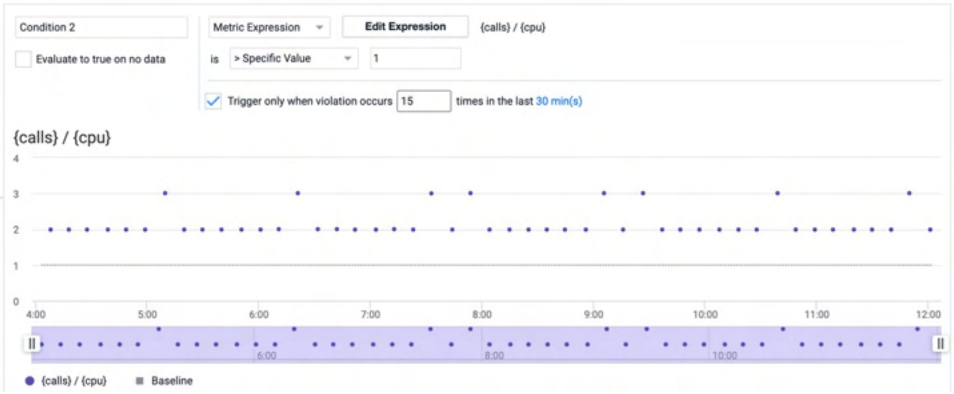
v. Click **Select a metric** to open an embedded metric browser. To avoid erroneous evaluations, if any metric in the expression has a null value, the health rule is not evaluated.

Expression	Null Value	Evaluation
a-b-c	a	entire expression is evaluated negative
a/b	b	the number 'a' is divided by zero, evaluates to an error
a*b	a or b	entire expression is evaluated as zero

vi. Repeat Steps I through V for every metric that you use in the expression. You can remove a variable by clicking the **Delete** icon.
vii. In the **Expression** pane, build the expression by clicking **Insert Variable** to insert variables created in the **Variable Declaration** pane along with appropriate mathematical signs.
viii. When the expression is built, click **Save**.
ix. Back on the **Critical Criteria** tab, select **< Specific value** or **> Specific Value** from the menu to compare the value of the metric evaluation with a static literal value, then enter the specific value in the text field. For example:

```
Value of {calls} / {cpu} > 1
```

A graphical view of the metric data for the last 8 hours appears.



You can view the metric data for 1 day or 3 days by selecting the time period using the drop-down menu.

3. If you want the condition to evaluate to true when a configured metric does not return any data during the evaluation time frame, select the **Evaluate to true on no data** option.
   This option does not affect the evaluation of the unknown in the case where there is no enough data for the rule to evaluate. For example, if the health rule is configured to evaluate the last 30 minutes of data and a new node is added, the condition evaluates to unknown for the first 30 minutes even if the **Evaluate to true on no data** option is selected.
4. If you want to define a 'Persistence Threshold' for the condition to reduce false alerts:

   a. Select **Trigger only when a violation occurs __ times in the last __ min(s).**
   b. Define the number of times metric performance data should exceed the defined threshold to constitute a violation.
   c. If required, adjust the evaluation time frame by setting an alternate evaluation time frame.

- You can define a persistence threshold for a condition only if you have defined an evaluation time frame of 30 minutes or less.
- If you define a persistence threshold for a condition, the metric data is plotted directly on the AST graph. If you do not define a persistence threshold, a 'moving average' for the selected metric is plotted. For more information, see Why Use Moving Average.

5. Specify the evaluation scope in the **Critical Criteria** and **Warning Criteria** panels, select the average of all nodes**.**

AST graphical view is displayed only when you set the health rule evaluation scope to the average for all nodes.

Configure Affected Entities

Related Pages:

- Health Rules
- Configuration Import and Export API
- Metric Browser

This page provides an overview of affected entities in AppDynamics.

The **Affected Entities** panel lets you define what entities your health rule affects. The health rule type you select determines the metrics that are offered for configuration in subsequent panels of the health rule wizard.

To define the affected entities:

1. Select a health rule type from the drop-down list. Depending on the type of the health rule, you can configure the corresponding entities that are affected. See Entities Affected by a Health Rule for information about the types of entities that can be affected by the various health rule types.
2. Use the drop-down list to select the entities affected by this health rule.
3. If you select entities based on matching criteria, specify the matching criteria.
   For example, if you select the Tier/Node Health - Transaction Performance as the health rule type, and if the health rule affects the nodes, you can restrict the health rule evaluation on the types of nodes or criteria such as meta-info, environment variables, and JVM system environment properties. Meta-info includes key-value pairs for:
   - `key: supportsDevMode`
   - `key: ProcessID`
   - `key: appdynamics.ip.addresses`
   - any key passed to the agent in the `appdynamics.agent.node.metainfo` system property

Configure Health Rule Evaluation Criteria

Related Pages:

- Health Rules
- Configuration Import and Export API
- Metric Browser

This page provides an overview of evaluation criteria for health rules in AppDynamics.

After configuring the entities affected by the health rule, you must define the evaluation criteria.

The high-level process for configuring the criteria is:

1. Determine which type of evaluation criteria for the health rule:
    - **Critical Criteria**
    - **Warning Criteria**

    Though the configuration processes for critical and warning conditions are identical, critical conditions are evaluated before warning conditions. If you have defined a critical condition and a warning condition in the same health rule, the warning condition is only evaluated if the critical condition is not true.

    You can copy the settings between Critical and Warning condition panels and edit the fields, if required. For example, if you have already defined a critical condition and you want to create a warning condition that is similar, in the **Warning Condition** window click **Copy from Critical Condition** to populate the fields with settings from the Critical condition.

2. Determine the number and kind of metrics the health rule should evaluate. For each performance metric you want to use, create a condition using one of the following methods:
    - Use a single metric
    - Use a complex metric expression

    For more information, see Configure a Condition.
3. If you have defined multiple conditions, check if the health rule violates by selecting one of the following options:
    - **All**: all conditions evaluate to true.
    - **Any**: one of the conditions evaluates to true.
    - **Custom**: if a boolean expression comprising all conditions evaluate to true.
4. For business transaction performance health rules and node health rule types that specify affected entities at the tier level, decide how many of the nodes must be violating the health rule to produce a violation event. See Health Rule Evaluation Scope.

Create and Configure Conditions

Related Pages:

- Health Rules
- Configuration Import and Export API
- Metric Browser

This page provides an overview of conditions in AppDynamics.

You can define a condition or a set of conditions to evaluate the performance metrics of your application.

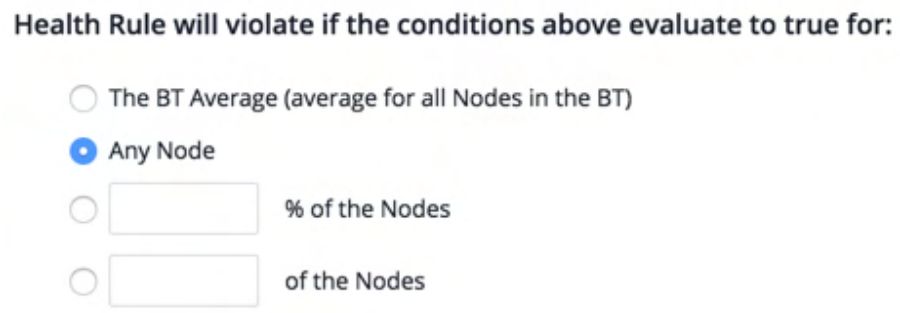
Use the following options to evaluate the conditions:

- Expression builder embedded in the health rules wizard to create a condition based on a complex expression comprising multiple interdependent metrics.
- Custom boolean expression to evaluate multiple conditions within a health rule. You can use the `AND` and `OR` operators in a boolean expression.

Create a Condition

1. In the **Critical Condition** or **Warning Condition** window, click **+ Add Condition** to add a new condition component. A row defining the condition is displayed.
2. Configure the Condition as required.
3. Continue to add conditions as required. You can add a maximum of 8 conditions. Conditions are designated as A, B, C, and so on.
4. From the drop-down list above the conditions, select:
    - **All:** if all of the conditions must evaluate to true to constitute rule violation.
    - **Any:** if any of the conditions must evaluate to true to constitute rule violation.
    - **Custom:** if a combination of conditions defined in a boolean expression must evaluate to true to constitute rule violation. For more information on how to create a custom boolean expression, see Create a Custom Boolean expression.
5. For health rules based on the following health rule types, select:
    - Business Transaction
    - Node health-hardware
    - Node health-transaction performance

    You must specify the evaluation scope in the **Critical Criteria** and **Warning Criteria** panels:

    You can use Alert Sensitivity Tuning to configure health rule conditions for a business transaction, a service endpoint or a remote service only. You must set the health rule evaluation criteria to 'average of all nodes'. For more information, see Alert Sensitivity Tuning.

    If you select the percentage of nodes, enter the percentage. If you select the number of nodes, enter an absolute number.

Evaluating Serverless Tiers

When you monitor serverless entities comprising tiers for AWS Lambda, the health rules are evaluated as described below.

Health Rule Type	Affected Entities	Condition Evaluation Criteria	Evaluation
• Business transactions • Service endpoints • Error rates	serverless tier(s)	The BT Average	Metrics are aggregated at the tier level.
	serverless node(s)	• Any node • % of the Nodes • Number of the Nodes	Metrics for serverless tiers are aggregated at the tier level, while the metrics for other tiers are evaluated as per the defined criteria.

Tier/Node Health (Transaction Performance)	serverless tier(s)	• The Tier Average (average for all Nodes in the Tier) • Any node • % of the Nodes • Number of the Nodes	Metrics for serverless tiers are aggregated at the tier level, regardless of the evaluation criteria defined.
	serverless node(s)	• The Tier Average (average for all Nodes in the Tier) • Any node • % of the Nodes • Number of the Nodes	The performance of serverless tiers is not evaluated for **Tier/Node Health (Hardware)** health rules. AWS does not offer node-level dashboards or metrics because the serverless platform runtime instances spin up and down on-demand.
Tier/Node Health (Hardware)	• serverless tier(s) • serverless node(s)	-	The performance of serverless tiers is not evaluated for **Tier/Node Health (Hardware)** health rules. AWS does not offer node-level dashboards or metrics because the serverless platform runtime instances spin up and down on-demand.

Configure a Condition

1. In the first field of the condition row, enter a name for the condition.
   This name is used in the generated notification text and in the AppDynamics console to identify the violation.
2. From the drop-down list below the **Add Condition** button, define metrics to evaluate the condition, select:
   - a. From the Value drop-down list, select a qualifier for the metric from the following options:

Qualifier Type	Description
Minimum	The minimum value reported across the configured evaluation time length. Not all metrics have this type.
Maximum	The maximum value reported across the configured evaluation time length. Not all metrics have this type.
Value	The arithmetic average of all metric values reported across the configured evaluation time length. This value is based on the type of the metric.
Sum	The sum of all the metric values reported across the configured evaluation time length.
Count	The number of times the metric value has been measured across the configured evaluation time length.
Group Count	The number of nodes contributing to a metric value, generally relevant for application or tier level metrics.
Current	The value for the current minute.

   b. To specify a simple metric, click **Select a Metric.** Metric Selection window is displayed. The metric browser in the **Metric Selection** window displays metrics appropriate to the health rule type. Alternatively, you can define a relative metric path..

   c. Select a metric to monitor and click **Select Metric**.

   > You can use Alert Sensitivity Tuning to fine-tune metric evaluation for a health rule (that monitors BT, service endpoint or remote service). You must select a single metric to evaluate the condition. See Create a Health Rule and Fine-tune Metric Evaluation for more information.

   or
   - Metric Expression to build a metric expression.
3. From the drop-down list after the metric, select the type of comparison to evaluate the metric.

   - To limit the effect of the health rule to conditions during which the metric is within a defined range—standard deviations or percentages—from the baseline, select **Within Baseline** from the menu. To limit the effect of the health rule to when the metric is not within that defined range, select **Not Within Baseline**. Then select the baseline to use, the numeric qualifier of the unit of evaluation and the unit of evaluation. For example:

```
Within Baseline of the Default Baseline by 3 Baseline Standard Deviations
```

- To compare the metric with a static literal value, select **< Specific value** or **> Specific Value** from the menu, then enter the specific value in the text field. For example:

```
Value of Errors per Minute > 100
```

- To compare the metric with a baseline, **select < Baseline** or **> Baseline** from the drop-down list, and then select the baseline to use, the numeric qualifier of the unit of evaluation and the unit of evaluation. You can use the Baseline Standard Deviation or Baseline Percentage as the unit of evaluation. For example:

```
Maximum of Average Response Time is > Baseline of the Daily Trend by 3 Baseline Standard Deviations
```

See Dynamic Baselines for information about the baseline options.

> **Baseline Percentages**
>
> The *baseline percentage* is the percentage above or below the established baseline at which the condition will trigger. For example, if you have a baseline value of 850 and you have defined a baseline percentage of > 1%, the condition is true if the value is > `[850+(850x0.01)]` or `859`.
>
> To prevent health rule violations from being triggered when the sample sets are too small, these rules are not evaluated if the load—the number of times the value has been measured—is less than 1000. For example, if a very brief time slice is specified, the rule may not violate even if the conditions are met, because the load is not large enough.

4. If you want the condition to evaluate to true whenever a configured metric does not return any data during the evaluation time frame, check the **Evaluate to true on no data** option.
   This option does not affect the evaluation of unknown in the case where there is no enough data for the rule to evaluate. For example, if the health rule is configured to evaluate the last 30 minutes of data and a new node is added, the condition evaluates to unknown for the first 30 minutes even if the **Evaluate to true on no data** box is checked.
5. If you want to define a 'Persistence Threshold' for the condition to reduce false alerts:

   a. Select 'Trigger only when a violation occurs __ times in the last __ min(s)'.
   b. Define the number of times metric performance data should exceed the defined threshold to constitute a violation.
   c. If required, adjust the evaluation time frame by setting an alternate evaluation time frame.

> You can define a persistence threshold for a condition only if you have defined an evaluation time frame of 30 minutes or less.

6. Click **Save** when done.

Evaluation of Agent Availability Metrics

Using Health Rule Conditions to evaluate agent availability metrics can result in false positives. For example:

- Agents may not be connecting with controllers due to communication errors for a couple of minutes.
- Data may be delayed for a couple of minutes due to latency issues.

You can configure the health rules such that you get notified of any occasional one-to-two minute metric loss due to network issues, or when an agent fails to communicate to the controller. Configure your Health Rule as follows:

1. Select Nodes that the health rule affects. You can also set Tiers, however, it is recommended that you set Nodes.
2. Select Node Health - Hardware, JVM, CLR as the health rule **Type**.
3. Use the last five minutes, with a wait time of ten minutes.
4. Set your condition to be the Sum of < Specific value of three.

This configuration generates a violation when the agent is down for more than two minutes during the last five minutes.

Build an expression

To access the expression builder to create a complex expression as the basis of a condition, select the **Metric Expression** option from the drop-down list and click **Add Expression**. The **Metric Expression** window is displayed that allows you to construct a mathematical expression to use as a metric.

For example, the following expression is created to measure the percent of slow business transactions. See the screenshot that follows for the UI location where each step is performed.

1. In the **Variable Declaration** pane of the Mathematical Expression builder, click **+ Add variable** to add a variable.
2. In the **Variable Name** field enter a name for the variable.
3. From the drop-down list, select the qualifier for the metric from the following options:

Qualifier Type	Description

Minimum	The minimum value reported across the configured evaluation time length. This qualifier is not available for all the metrics.
Maximum	The maximum value reported across the configured evaluation time length. This qualifier is not available for all the metrics.
Value	The arithmetic average of all metric values reported across the configured evaluation time length. This value is based on the type of the metric.
Sum	The sum of all the metric values reported across the configured evaluation time length.
Count	The number of times the metric value has been measured across the configured evaluation time length.
Group Count	The number of nodes contributing to a metric value, generally relevant for application or tier level metrics.
Current	The value for the current minute.

4. Click **Select a metric** to open an embedded metric browser.

Health Rule Evaluation Condition

A health rule is not evaluated if any metric in the expression has a null value. This is to avoid erroneous evaluations as shown in the following examples:

Expression	Null Value	Evaluation
`a-b-c`	a	`entire expression is evaluated negative`
a/b	b	`the number 'a' is divided by zero, evaluates to an error`
`a*b`	a or b	`entire expression is evaluated as zero`

5. Repeat steps 1 through 4 for each metric that you use in the expression.
   You can remove a variable by clicking the delete icon.
6. In the **Expression** pane, build the expression by clicking **Insert Variable** to insert variables created in the **Variable Declaration** pane along with appropriate mathematical signs.
7. When the expression is built, click **Save**.

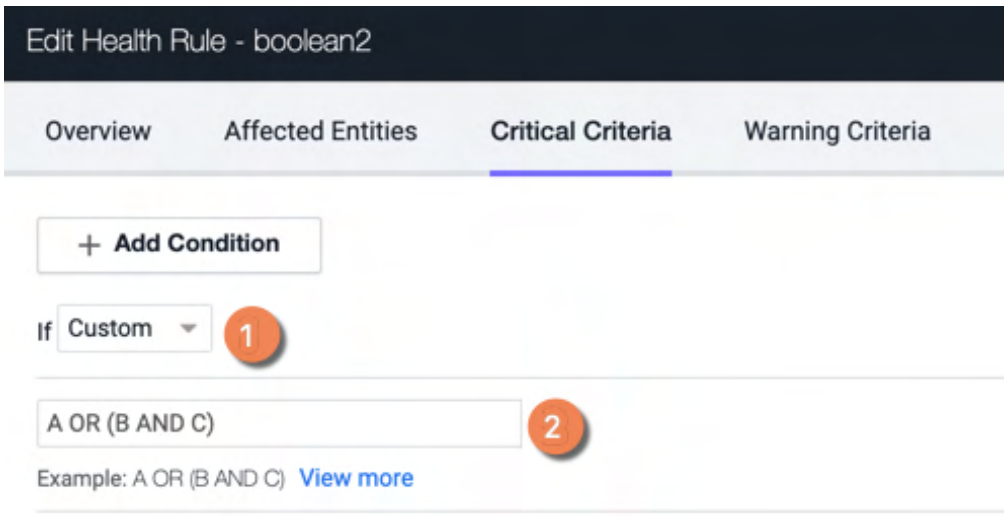
Create a Custom Boolean Expression

Once you define all the conditions required for the health rule, you can create a custom boolean expression to evaluate the health rule.

1. From the drop-down list above the conditions, select **Custom** option.
2. Enter a combination of conditions using `AND` and/or `OR` operators. For example, `(A OR B) AND C`.

Ensure that you enter a valid combination of conditions using `AND` and `OR` operators, else the evaluation fails.

3. Click **Save**.



Modify the Custom Boolean Expression

1. Select the expression in the **Condition Combination** field.
2. Edit the boolean expression as required.
3. Click **Save**.

Delete a condition

Delete a condition component by clicking the delete (**X**) icon.

> If you delete a condition, update the boolean expression accordingly.

Define Custom Metrics for Multiple Entities

Related Pages:

- Health Rules
- Configuration Import and Export API
- Metric Browser

To create a health rule on a custom metric in a single business transaction, node, or overall application performance, you specify the health rule type as *custom* and when you configure the condition component, in the **Select Metric** window, choose to **Specify a Metric from the Metric Tree** and select the metric from the embedded metric browser.

A different use case is to create a rule that evaluates a custom metric that exists across various entities, for example across several nodes. You want to do this with one health rule; you do not want to create a separate health rule for each node. In this case, you need to specify the custom metric using the relative metric path to the metric instead of selecting the metric from the embedded metric browser.

First, get the relative path to the metric and then configure the health rule using that relative path.

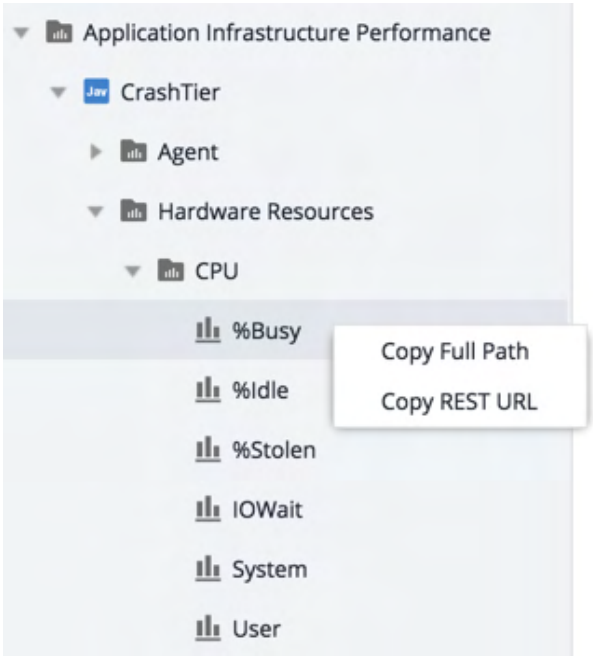
To specify Hardware Resources, JVM, and CLR metrics in multiple entities using a wildcard, you can use the procedure described in Use Wildcards in Metric Definitions.

> The wildcard feature is not supported for custom metrics.

To get the relative metric path for a multi-entity metric:

1. Navigate to the Metric Browser by selecting the **Metric Browser** in the left navigation pane.
2. Select the metric that you want to use for the condition.
3. Right-click and select **Copy Full Path**.
4. Save this value in a file from which you can copy it later.

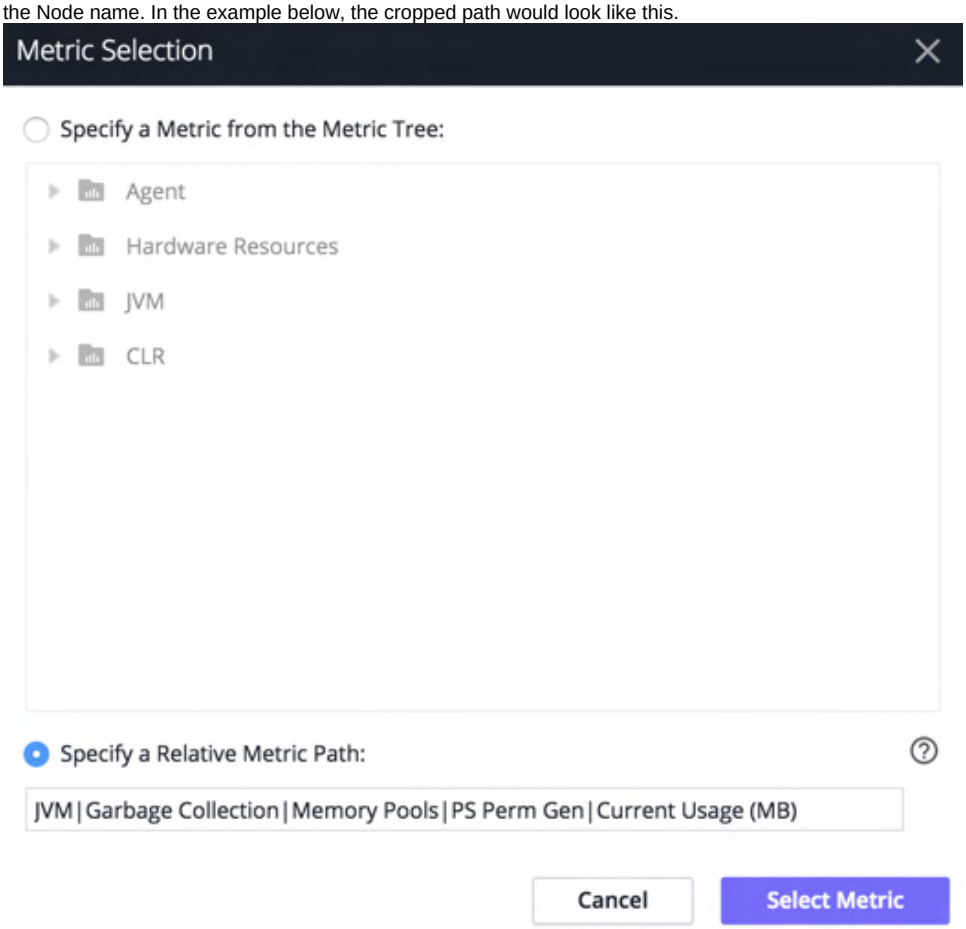
The following example gets the metric path for the CPU %Busy metric for the Inventory Server tier. The CPU %Busy metric would be appropriate to use in a health rule that affects all the nodes in that tier.



To configure a health rule that evaluates the custom metric over multiple entities

1. In the **Overview** panel of health rule wizard choose the health rule type for the kind of entity that you are monitoring.
2. In the **Affected Entities** panel select the affected entity.
3. When you create the condition component that uses the metric, in the Select Metric window choose **Specify a Relative Path Metric**.
4. Crop the relative metric path that you saved from the metric browser by doing one of the following:
    - For all health rule types except Node Health-Hardware, JVM, CLR or Custom, crop the path to use the metric name alone, for example, Average Wait Time (ms).

- For Node Heath-Hardware, JVM, CLR and Custom health rule types, crop the path to use everything after the entity, for example, after the Node name. In the example below, the cropped path would look like this.



5. Paste the cropped relative metric path in the relative metric path field of the **Metric Selection** window.
6. Click **Select Metric**.

JMX Health Rules

Related Pages:

- Monitor JMX
- Configure JMX Metrics from MBeans
- Configure Health Rules

This page provides an overview of JMX health rules, which establish the health status of entities in a monitored Java application based on JMX metrics.

JMX Instance Names

For health rules based on JMX metrics, you can create health rules on a node or on an entity called a JMX instance name. If you create the health rule on the JMX instance name, you have the option of restricting it to specific nodes. If you create the health rule on the node, you have the option of restricting it to specific JMX instance names.

A JMX metric is identified by its JMX metric path, its JMX instance name, and its metric name. These are determined by the **Metric Path**, **Object Name Match Pattern**, and **Metric Name** fields specified in the AppDynamics **JMX metric MBean Configuration** screens. The instance name is derived from the value returned by the object match pattern. Identical JMX instance name values are considered distinct when their metric paths differ.

JMX instance names appear in the Metric Browser under the appropriate JMX metric path. Specific JMX metrics are reported under each JMX instance name.

For details about how JMX metrics are configured, see Configure JMX Metrics from MBeans.

Creating JMX Health Rules

To create a health rule on one or more JMX metrics, in the **Affected Entities** panel of the health rule wizard, set the type of the rule to Node Health-JMX–connection pools, thread pools, and so on.

Determine what JMX objects the health rule will be evaluated on. In some IT organizations, different teams are responsible for different MBeans. In other organizations, different teams are responsible for different nodes or tiers.

The way your organization is set up determines how you think about JMX health, especially where it intersects with node health.

- As a team member responsible for a node, do you consider your node unhealthy because one or more of its MBeans is unhealthy? If so, which nodes or how many?
- As a team member responsible for your JMX infrastructure, are you primarily interested in the health of your JMX data regardless of the nodes that use it? Or are you interested in just specific JMX metrics in specific nodes? Which ones?

> It is recommended to have the JMX rule with instance identifier because it helps in selecting the metric under category in the metric select tree of the UI.

For the purpose of configuring JMX health rules and using them in policies, the ultimate question is: who will receive the alert when the agent reports unhealthy performance detected by JMX metrics? The flexibility of the AppDynamics health rules lets you fine-tune your JMX health rules so that the right people are alerted for the code for which they are responsible.

Select whether the affected entity should be the JMX instance name or the node. In either case, you can configure the rule to cover one of the following scopes:

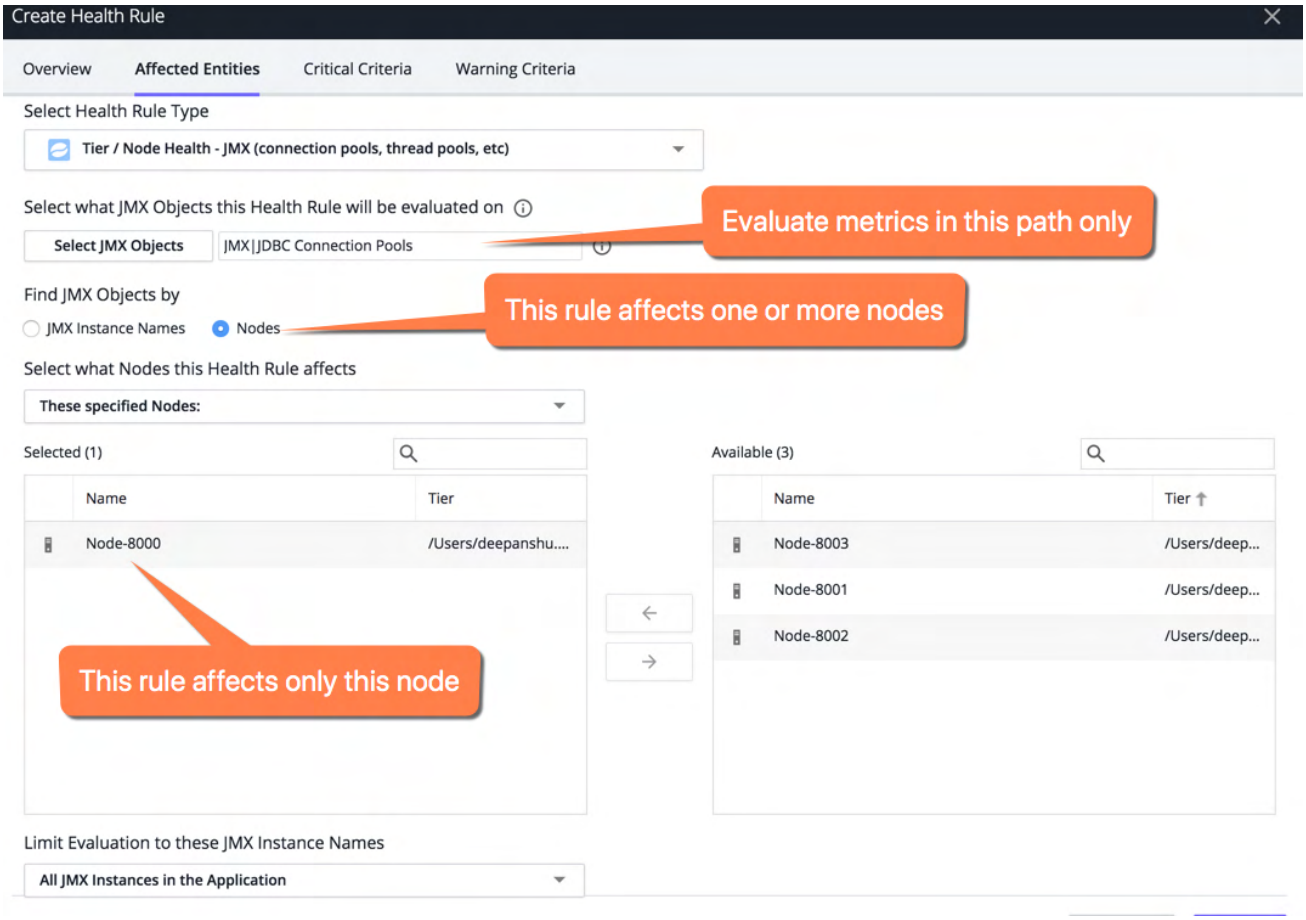
- All JMX instance names in the application
- Specific JMX instance names
- All nodes in the application
- Specific nodes in the application
- Nodes within the specified tiers
- Nodes matching a given criteria

You can limit the evaluation of the health rule to either the specified JMX instance names or nodes depending on the evaluation scope.

JMX Health Rules Affecting a Node

In one organization, teams are responsible for nodes. Mark is responsible for `WEB1_NODE`, Tao for `WEB2_NODE`, and so on.

If an MBean in `WEB1_NODE` generates JMX metrics that violate a critical condition, and a health rule is configured to evaluate a JMX object in that node, Mark or someone on Mark's team will get an alert. The configuration of the health rule would be:



If different people on Mark's team are responsible for different MBeans used in `WEB1_NODE`, they could refine the rule further by selecting specific JMX instance names for evaluation. For example, the last decision of this configuration can restrict the rule to evaluate only metrics in the `jdbc/ECommerceDB` JMX instance name.

Mark's team could create a similar rule for the remaining JMX instance names to use in a policy that alerts whoever on Mark's team is responsible for the `jdbc/OracleECommerceD` JMX instance names.

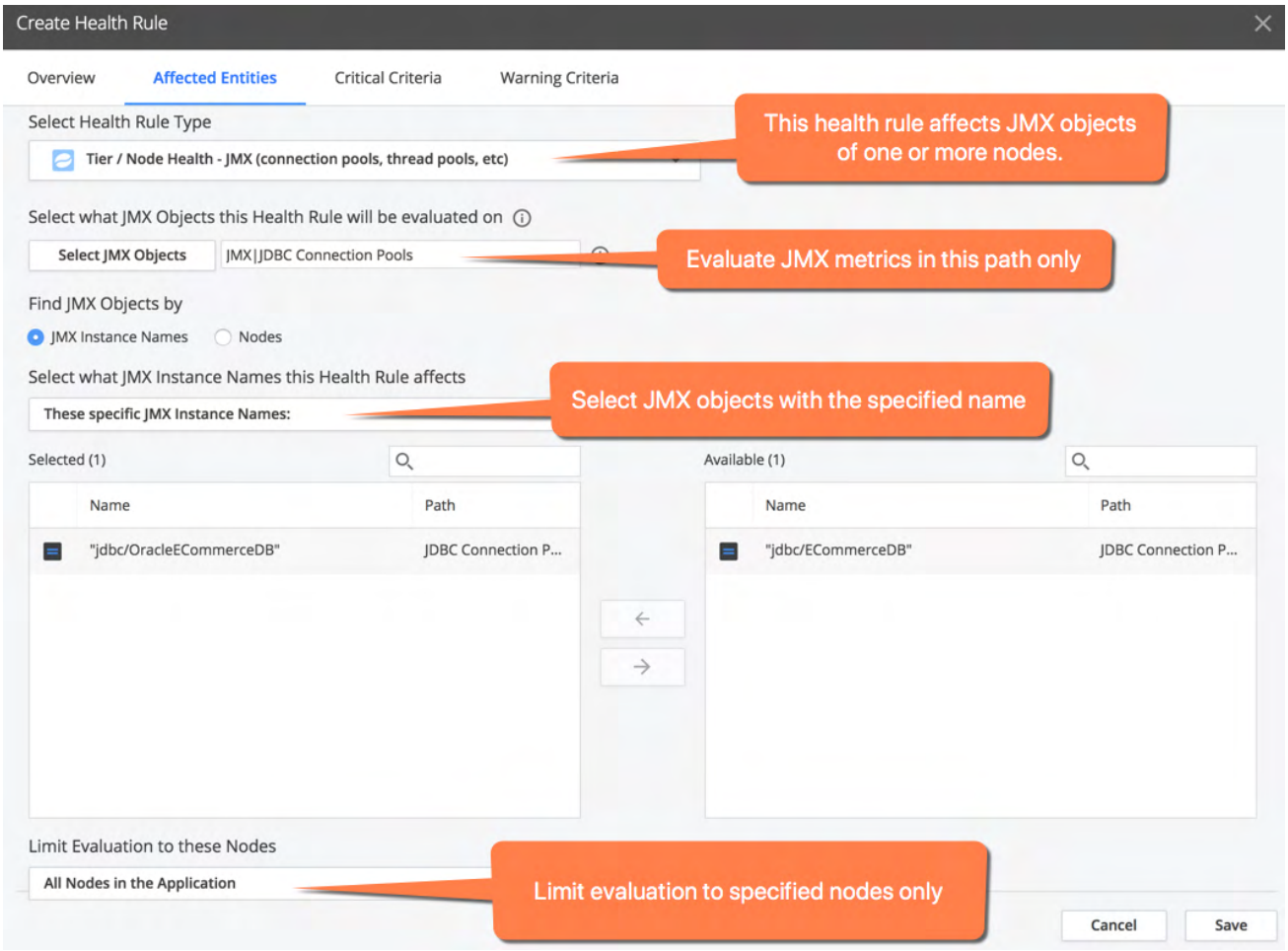
Mark's team could also create different rules that evaluate a different set of JMX metrics by choosing a different metric path in the Select JMX Objects field. For example, they could select **All Web Container Runtimes**.

For problems on `WEB2_NODE`, they would create a different health rule with `WEB2_NODE` as the affected node, so that the alerts for those problems go to Tao's team.

JMX Health Rules Affecting a JMX Instance Name

In another organization, teams are responsible for various parts of the JMX infrastructure. Mary is responsible for `jdbc/OracleECommerceDB`, Meera for `jdbc/ECommerceDB`, and so on, regardless of which nodes use these MBeans.

So if metrics in `jdbc/OracleECommerceDB` violate a critical condition, they want Mary to get the alert because her team is responsible. The configuration of the health rule would be:



The rule could be refined to evaluate the JMX metrics in the specified JMX instance names in all nodes in the application or only in specific nodes.

Troubleshoot Health Rule Violations

Related Pages:

- Health Rules
- Configure Health Rules
- Policies
- Alert and Respond
- Business Transactions

This page provides an overview of health rule violations in AppDynamics.

A health rule is violated when the health rule processor detects that the health rule's critical or warning condition is true. In this case, a health rule violation is created with a status of Open, and a Health Rule Violation Started - Critical event or a Health Rule Violation Started - Warning event is generated.

A health rule violation ends when it is either: resolved (the agent reports metrics that indicate that that the violated condition is no longer true) or canceled (t he health rule processor can no longer accurately assert that the health rule violation continues to violate or that it has ended).

When the violation status of a health rule becomes resolved, a *Health Rule Violation Canceled - Critical* event or a *Health Rule Violation Ended - Warning* event is generated.

The health rule violation status is canceled when:

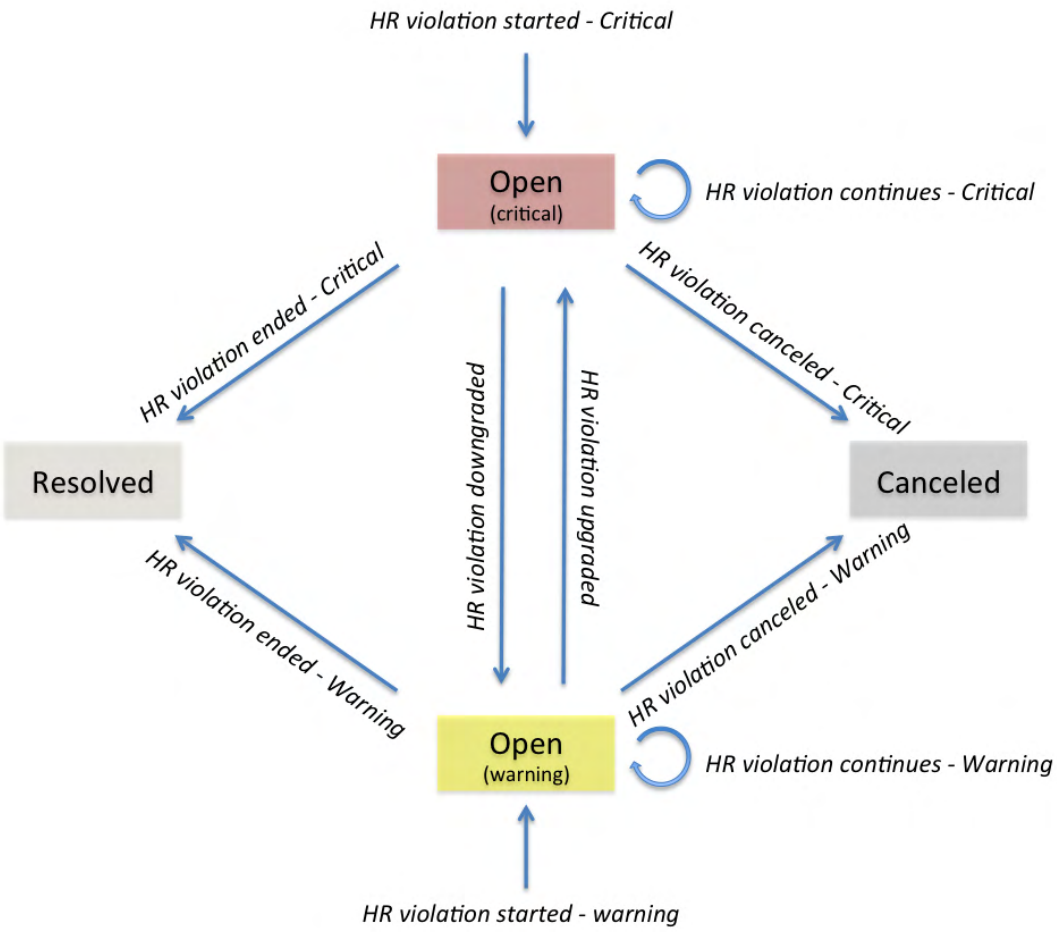
- The health rule is edited
- The health rule is disabled
- Affected entities or evaluation entities on which the health rule is based have been added or removed
- The metric values on which the health rule violation is based have become `UNKNOWN`

When the violation status of a health rule becomes *Canceled*, a *Health Rule Violation Ended - Canceled* event or a *Health Rule Violation Canceled - Warning* event is generated.

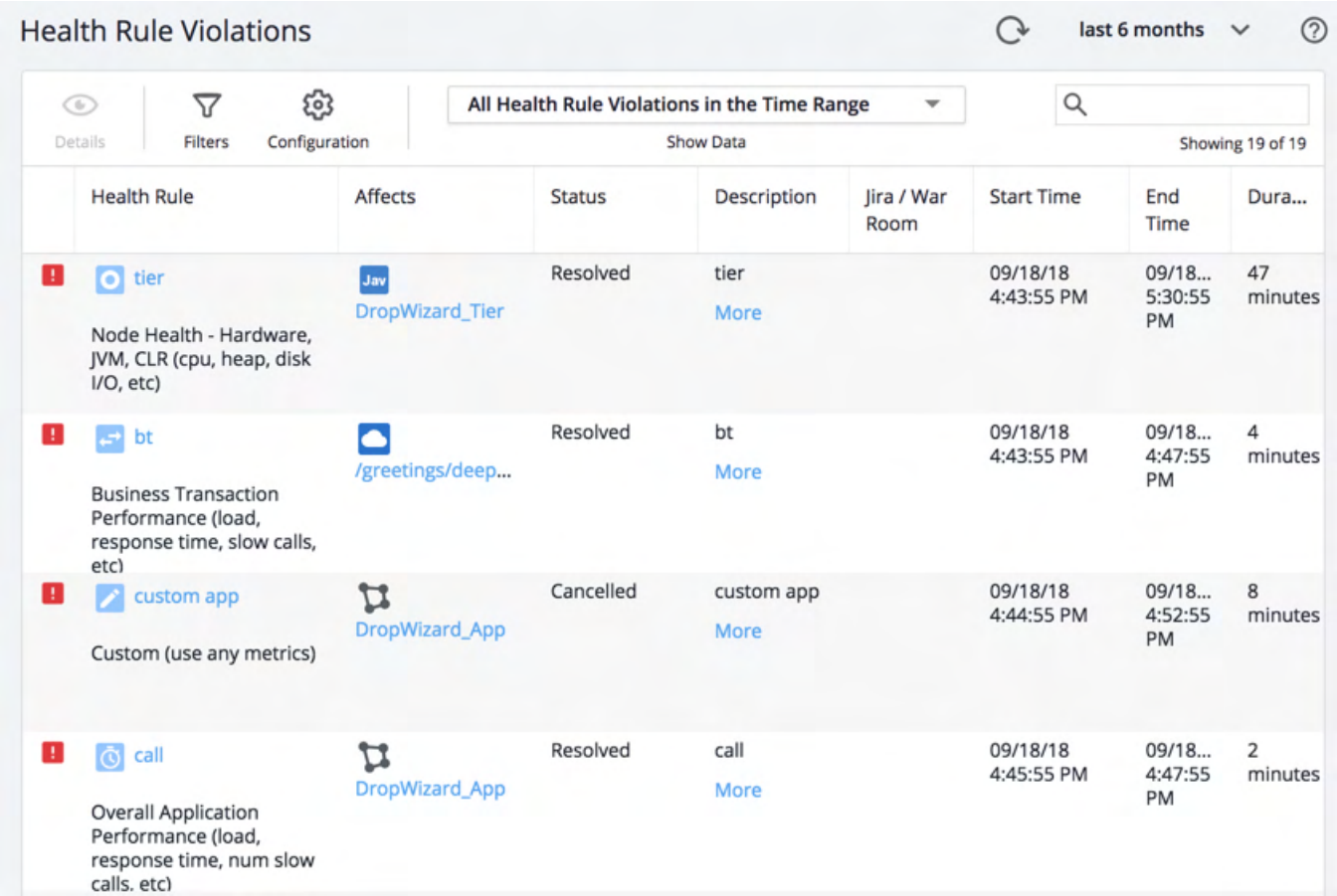
If the same health rule is violated after a violation of it has been resolved or canceled, a new health rule violation is started.

During the life of a single health rule violation, there may be other types of health rule violation events such as Health Rule Violation Ungraded/Downgraded /Continues events.

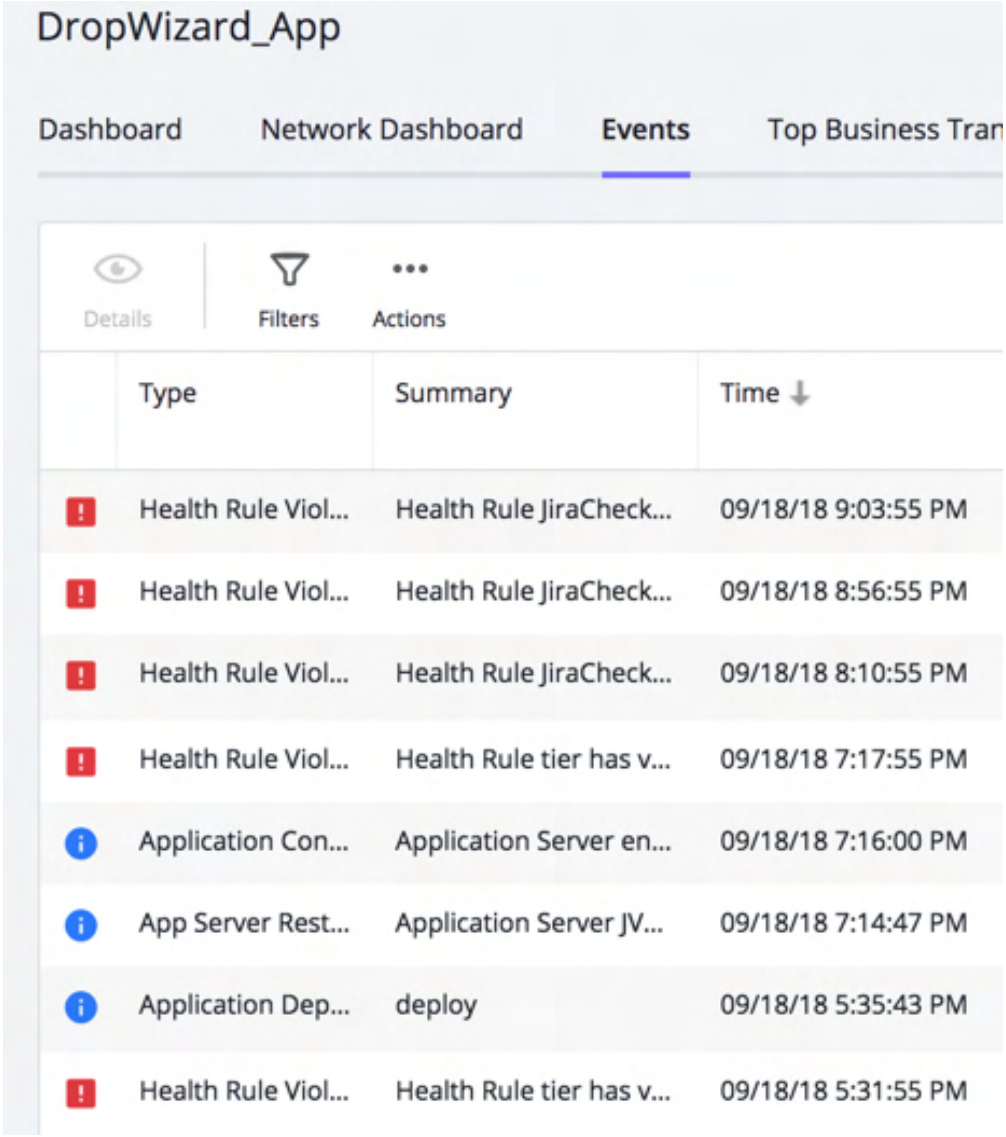
The figure below illustrates the health rule violation life cycle.



The boxes represent the health rules violation statuses that you see in the health rule violations list in **Troubleshoot > Health Rule Violations.** To get more information about a particular violation, select the violation in the list and click **Details**. You can also view the health rule violations in the Controller UI.



Health rule violation events are listed in the **Events** tab of various dashboards.



Because there is a set of default health rules, you may see health rule violations reported for your application even if you have not set up your own health rules. Violations reported for the `APPDYNAMICS_DEFAULT_TX` business transaction are for default health rule violations in the All Other Traffic business transaction.

Find Heath Rule Violations

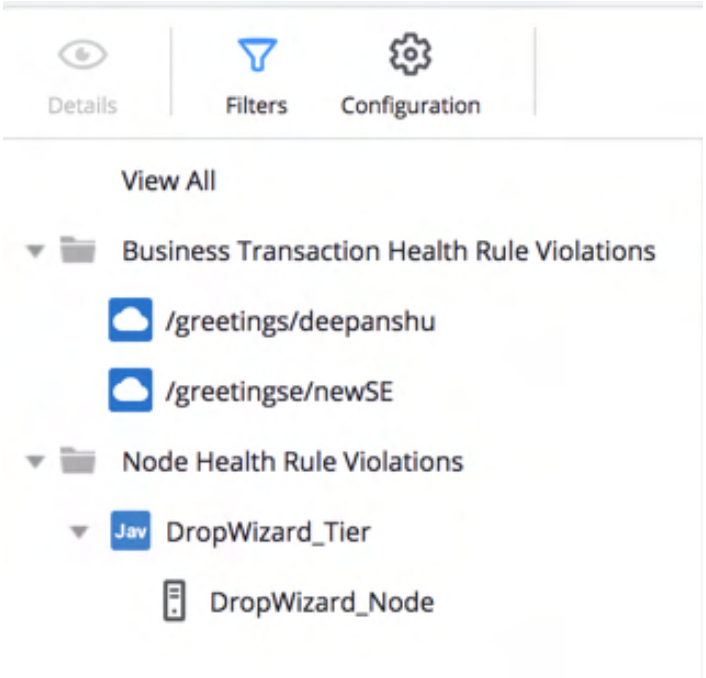
To find all health rule violations:

1. In the left navigation pane, click **Troubleshoot > Health Rule Violations**.
   The list of health rule violations displays.

   > You can also access this screen directly from the left navigation pane of an EUM application.

2. Select **All Health Rule Violations in the Time Range** or **Only Health Rule Violations Open Now** from the drop-down list.
   It is possible that health rule violations that were reported are no longer open because remedial action has been taken or performance has improved on its own.
3. You can filter the list. To see the filters, click **Filters**. To hide them, click **Filters** again.
   With the filters showing in the left filters panel, you can select the health rule violations that you want to troubleshoot.

You can view all health rule violations or expand the nodes in the tree to select by health rule type or affected entity, such as business transaction, tier or node.



You can filter health rule violations by entering the name of the health rule in the search field on the upper right.
The health rule violations are displayed in the right panel, with their affected entity, status, description, start time, end time and duration, if ended.

Examine a Health Rule Violation

To view details about a particular health rule violation:

1. Select the health rule violation row in the list.
2. Click **View Health Rule Violation Details**.

In the **Health Rule Violation** summary window, you can click the **Affects** link to see the dashboard of the entity affected by the health rule. Alternately can click the **View Dashboard During Health Rule Violation** button to view the dashboard at the time the violation occurred.

You can also access details of a health rule violation from the health indicators in the UI. For example, if you see the red indicator on the dashboard indicating that Business Transaction Health or Server Health is critical, click it to get the list of business transactions or tiers and then click the icon in the **H ealth** column of the list.

View Actions Triggered by a Health Rule Violation

For health rules that trigger actions configured by policies, you can get information about the action that was executed.

To view the action triggered by a health rule violation:

1. In the dashboard of the entity affected by the health rule violation, click the **Events** tab.
2. In the events list, locate the health rule violation that you are interested in examining. If an action was triggered by the health rule violation event, you will see an event icon in the **Actions** column of the list.
3. Select the row for the event.
4. Click **Details**.
5. In the health rule violation window, click the **Actions Executed** tab.

Alert Sensitivity Tuning Questions and Answers

As an on-prem customer, can I fine-tune a health rule using AST?

No, AST is available for SaaS customers only.

Can I use AST to fine-tune a health rule that monitors any entity?

You can use AST to fine-tune health rules that monitor business transactions, service endpoints, or remote services only. You must set the health rule evaluation scope to 'average for all nodes' of the affected entity.

How is the metric data presented in the AST graph?

If you define a persistence threshold for a condition, the metric data is plotted directly on the AST graph. If you do not define a persistence threshold, a 'moving average' for the selected metric is plotted. For more information, see Why use 'moving average'?.

What is a moving average? How is it helpful?

Unless persistence thresholds are used, health rules compare the moving average of a metric to a threshold or a baseline. Thus, representing the moving average in the graph is appropriate.

Can I define 'persistence threshold' for a heath rule if I plan to use AST?

Yes, you can define 'persistence threshold' for a health rule. However, you will not be able to use the moving average for the metric. For more information, see Moving Average.

How does AST help calibrate the sensitivity of the health rule?

Depending on the baseline configuration you define, AST presents a graphical view of the metric data for the given baseline configuration. The graphical view is instantly updated when you update any configuration. You can also view granular details by modifying the graphical view. The historical metric data presented in the graph helps you calibrate the sensitivity of the metric evaluation (thus the sensitivity of the health rule).

I am not able to see the AST graphical view, how can I get the graphical view?

The graphical view may not be displayed due to multiple reasons. Check if you have:

- created the health rule for a business transaction, a service endpoint, or a remote service.
- set the health rule evaluation scope to 'average of nodes'.

Can I use AST to configure 'Critical' and 'Warning' criteria?

Yes, you can use AST to configure conditions for both critical and warning criteria.

Will AST impact the evaluation of boolean expression for health rule conditions?

There is no change in the evaluation of boolean expressions for health rule conditions.

Can I use AST to edit an existing health rule?

Yes, you can use AST when editing an existing health rule.

Actions

This page provides an overview of actions in AppDynamics.

An action is a predefined, reusable, automated response to an event. You can use actions to automate your runbooks.

A policy can trigger an action in response to any event. You configure which actions are triggered by which events when you configure policies.

Types of Actions

You can create these types of actions:

- Notification
- Diagnostic
- Remediation
- JIRA
- HTTP Request
- Custom

Not all actions are applicable to all application environments or to all situations. Below are some general guidelines concerning different types of actions. For more details, see the pages on the specific actions before you assign an action to a policy.

- The diagnostic thread dump actions can be performed only on nodes running a Java agent.
- The diagnostic session actions can be triggered only by violations of business transaction performance health rules or by slow or stalled transaction events since these are the events that produce a view into transaction snapshots.
- Remediation actions run a local script in a node and are available on nodes running on machines that have an installed Machine Agent.
- Custom Actions require a dedicated Controller, deployed using either the on-premises or SaaS option. This feature is not supported for accounts on multi-tenant SaaS controllers.

Actions Limits

The Controller limits the actions invoked based on the number of triggering events per event type. There is a maximum of ten events for any single event type that can trigger actions in a given minute. If the number of triggering events per type exceeds the limit, the actions that would have been triggered by the excess events are not started. You will not see a visual indication that these actions are not being started.

For example, your application can have up to ten Health Violation Started events triggering actions and up to ten Resource Pool Limit Reached events triggering actions within the same minute. But if you have eleven Health Violation Started events firing, the action that would be triggered by the eleventh event is not started.

To reduce unnecessary actions, there is a limit on the number of diagnostic and remediation actions that AppDynamics will invoke. The default limit is five actions per minute per machine for each type of action.

If, for example, a policy is configured on all the nodes where there are 100 nodes triggering actions, AppDynamics randomly selects five of the actions to execute.

To avoid exceeding the limits, design your policies so that they do not trigger an excessive number of actions for any particular event. You can generate fewer events by configuring the affected entities of your health rules at the tier level. See entities affected by a health rule and by health rule types in Health Rules.

Actions Requiring Approval

For actions that take thread dumps or run a local script, you can optionally require email approval to run the action whenever it is triggered. If you check this option, human intervention is required before the automated action actually starts.

If you specify the approval required option when you configure the action, when the action is triggered an email containing a link is sent to the configured email address. The link presents a login screen (if the user is not already logged in to AppDynamics) and after the user logs in, a dialog requesting approval to take the thread dump or run the script. The user can click in this dialog to approve and start the action or cancel the action.

If you do not check the **Require Approval** option before executing the **Action** checkbox, the action will start automatically with no human intervention.

Create and Modify Actions

To configure actions, you need the Configure Actions permission.

To access the Actions configuration panels:

1. Click **Alert & Respond** in the menu bar.
2. Click **Actions** either in the right panel or the left navigation pane.

3. Select the context for the action (Application, User Experience Browser App, Databases, Servers, Analytics) from the pulldown menu.
   The list of configured actions displays. You can filter the types of actions by checking the type in the pulldown filter. If no action types are checked, there is no filtering; however if at least one type is checked, then only actions of that type display.
4. Do one of the following:
   - To create a new action, click the **Create Action (+)** icon. After you have clicked **Create Action**, the instructions provided depend on the type of action you are creating.
   - To edit an existing action, select the action and click the **Edit** (pencil) icon.
   - To remove an existing action, select the action and click the **Delete** (-) icon.

View Actions

While actions are being triggered by events, you can get a summary of their execution in the **Actions** column in the **Events** list. You can access the **Events** list from the Events tab of the application, tier, or node dashboard.

The icon type indicates action type; an icon in the **Actions** column indicates that an action was triggered for this event. If the action icon is greyed out, the action is either still executing or failed to complete. If the icon fully displays, then the action executed successfully.

Action Suppression

You can prevent policies from firing actions for a configurable time period.

Notification Actions

Related Pages:

- Alert and Respond
- Policies
- Email Templates
- Enable an Email Server

This page provides an overview of notification actions in AppDynamics. A notification action sends an email or SMS to a recipient or recipient list. If you are using a SaaS Controller, all notification timestamps are in Pacific Time (PST).

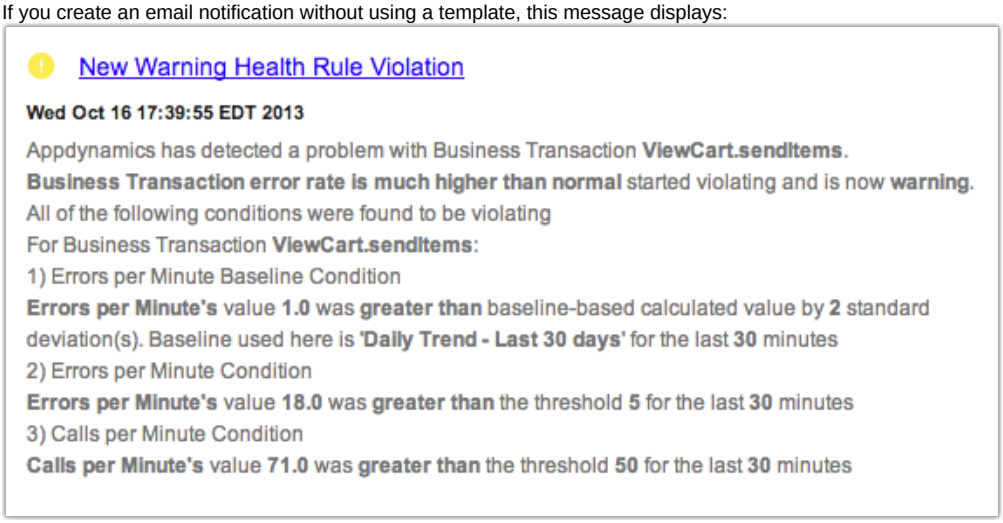
Email Notifications

You can create an email notification with or without an email template.

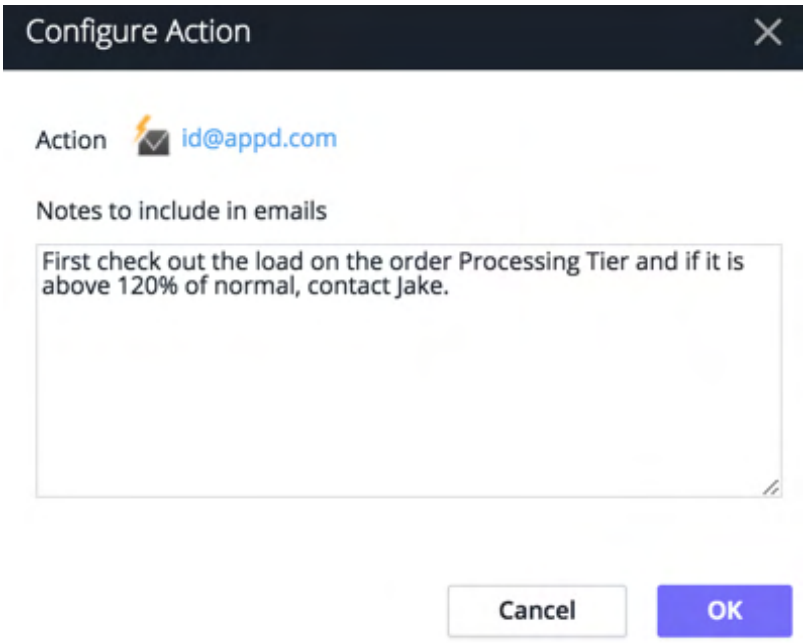
Create Email Notification Without a Template

If you create an email notification without a template, you need to provide the email address of the notification recipient. The name of the action is the name of the recipient. The contents of the email are automatically generated by the policy that triggers the action. The generated message contains a link directly to the Controller panel that details the triggering event, which is the place for the recipient to start troubleshooting the problem.

If you create an email notification without using a template, this message displays:



When you configure a policy to start an email notification action that has not been created with a template, you can add an optional note to the email. This note is applied only when the action is invoked by the particular policy. By adding an optional note, you can customize email notifications for the policies that invoke them.



Create Email Notification With a Template

If you create an email notification with a template, you provide the name of the action and the name of the email template to use. The template provides for addressing multiple recipients and you can add and delete recipients when you create an action using the template.

The template must already exist before you can use it in an action. Email templates are created by users who have account-level permissions to create templates. See Email Templates for information about creating templates.

The body of the message is specified in the email template. It is not generated automatically.

All the text in the message must be created in the template. Customization is accomplished through the use of variables in the message body, which are replaced by actual values when the message is sent.

You can use a template to:

- Reuse the notification in other policies
- Customize the body of the message
- Decorate the notification with your own branding
- Add custom SMTP headers to the email
- Control whether the email is sent once or several times for each event
- Control the maximum number of triggering events listed in the email using a clamp limit
- Integrate with third-party email APIs, such as Remedy

To create an email notification:

1. Access the **Create Action** panel. See Create and Modify Actions in Actions.
2. Under **Notifications**, select **Send an email** in the **Create Action** pane. To use a template to create the email notification, check **Use template?**
3. Click **OK**.
4. Do one of the following:
    - If you are not using a template, enter the email address to which to send the notification.
    - If you are using a template, name the action and select the appropriate template from the dropdown.
      Unless disallowed by the template, you can add recipients at this time but you cannot remove any of the required recipients that were configured in the email template.
5. Click **Save**.

> If email and SMS settings have not been configured for AppDynamics, configure them now. See Configure the SMTP Server.

SMS Notifications

The content of the SMS is automatically generated. It contains:

- Notification header
- Application name
- Triggered time

Notifications of health rule violations also include: Name of health rule violated

Event notifications also include:

- Event notification configuration name
- Map of event types to the number of these events

An SMS notification configuration specifies the phone number of the recipient.

To create an SMS notification:

1. Access the **Actions Configuration** panel. See Create and Modify Actions in Actions.
2. Under **Notifications**, select **Send an SMS message** in the **Create Action** panel.
3. Enter the phone number to send the notification.
4. Click **OK**.

If you have not configured email and SMS settings for AppDynamics, configure them now. See Enable an Email Server.

Email Templates

This page provides an overview of email templates in AppDynamics. Email templates are optional for email notification actions.

You must have account-level Configure Email Templates permission to create an email template. See Alert and Respond.

Access Email Templates

Although the configuration settings are self-explanatory, some template configuration features and options require additional information.

1. Click **Alert & Respond** > **Email Templates**.
2. Click an existing template to view, edit, or delete it or lick **New +** to create a template.
3. Edit or define the template.
4. Click **Save**.

Custom Templating Variables

You can use variables that replace values in the message body when the email is sent.

The template recognizes a set of predefined variables. Review Predefined Templating Variables to verify the variable you want does not already exist.

If the predefined variables do not meet your needs, you can optionally configure custom variables. When a predefined variable and a custom variable are both configured with the same variable name, the template uses the predefined variable.

The template uses Apache Velocity version 1.7 to process the variables. See the Velocity User Guide.

Admin Email Recipients

To add a recipient, enter an email address and click **+ Add**. To remove a recipient, click **–**.

If the **Allow custom email recipients** checkbox is selected in the template, users of the email template can add additional email recipients when they create the email notification. However, they cannot delete any of the recipients configured by the template. Clear this checkbox if you do not want to allow template users the ability to modify the recipient list.

The only way to remove a required recipient is by modifying the template.

Email Template

Enter the subject line and the body of the message. AppDynamics recommends including both an HTML version and a plain text version. You can use any predefined or custom templating variables in both the HTML and plain text message bodies. Use `background-color` instead of `background` for HTML email templates.

For-each loops are supported. See Predefined Templating Variables.

Custom Email Headers

Use this section to add custom SMTP headers to the message.

One Email Per Event

Several separate events, or separate occurrences of the same event, could potentially invoke the same notification action.

The **One Email Per Event** setting controls whether the action bundles the emails triggered by those events. The effect is:

- If selected, the email is sent once every minute, no matter how many events triggered the notification within that minute. This is the default.
- If not selected, an email is sent every time an event triggers it. In this case, if ten events trigger the action, the email is sent ten times, even if all those events occurred within a single minute.

Event Clamp Limit

If you have not selected the **One Email Per Event** checkbox, you might want to limit the display of the events that triggered the action in the email, especially if the potential list of events could be long. The clamp limit is the number of most recent triggering events to be shown in the email. This setting is disabled if **One Email Per Event** is selected.

Test Email Template

After you save the email template, you can test the template by sending an email.

Test Email Templates

This page provides instructions on testing email templates in AppDynamics.

After you save the email template, you can test the template by sending an email.

1. In the **Create Email Template** dialog, click **Test**.
2. In the **Email Template Test** dialog, specify the test variables to use for the test. These variables can be different from those in the real emails that are sent automatically.
3. Set the log level or the amount of detail that you want to see: **INFO**, **WARN**, **ERROR**, **TRACE**, and **DEBUG**.
   The log level defaults to **DEBUG**, but you can change this using the **Log Level** dropdown.
4. Specify the type of events that trigger the email action along with the count. You can add more triggers if required.
5. Click **Save**.

Run Test

When you run the test, an email is sent when the action is triggered. You also receive a transcript of the test.

If the results are not what you expect, you may need to modify the email template, the event triggers, or both, before you use the template in an action.

Diagnostic Actions

This page provides an overview of diagnostic actions in AppDynamics.

A diagnostic action can:

- Start a diagnostic session to collect snapshots
- Take a thread dump for Java only

When performance is slow or your application is experiencing a lot of errors, you can start a diagnostic action to find the root cause.

> ⚠ You can start a diagnostic session only for a health rule that monitors business transactions.

A diagnostic session provides a view into captured transaction snapshots with full call graphs. These snapshots help you diagnose violations of business transaction performance health rules or slow or stalled transaction events. The affected entity of the event triggering a diagnostic session must be a business transaction.

A thread dump is a general-purpose snapshot of the state of all threads that are part of a JVM process. The state of each thread is presented with a stack trace that shows the contents of each thread's stack. Thread dumps are used for diagnosing JVM performance problems, such as code deadlocks.

Diagnostic Session Actions

A diagnostic session is always associated with a business transaction. It shows transaction snapshots with full call graphs to help you drill down to the root cause of a problem.

To create a diagnostic session action:

1. Access the **Actions Configuration** pane. See Create and Modify Actions in Actions.
2. Under **Diagnostics**, select **Start a Diagnostic Session on the selected Business Transactions** in the **Create Action** pane and click **OK**.
3. After entering a name for the action, the duration of the diagnostic session in minutes and the number of snapshots to take per minute, select whether a diagnostic session will be started for any business transaction affected by the event or for specific business transactions.
   If you choose specific business transactions, specify the business transactions that will trigger the diagnostic session by moving them from the available list to the selected list. The business transactions that you can specify are not limited to those that triggered the action.
4. Click **OK**.

Diagnostic Action Results

The results of a diagnostic action that has executed are available in the events list for the event that triggered the action. To get the details of a diagnostic session or a thread dump that has been initiated by an action:

1. In the **Events** list, locate the row for the event that triggered the action for which you want to see the results.
2. In the **Actions** column:
   - Click  to see the details of a thread dump action.
   - Click  to see the details of a diagnostic session action.

On disk, the thread dumps are stored in the `app_agent_operation_logs` directory in the Controller Home. The files are named based on the ID in the a `pp_agent_operation` table.

Threads identified by `AD Thread` are threads initiated by the AppDynamics app agent code.

Thread Dump Actions 

You can direct the Java Agent to take a thread dump for a specified number of samples (maximum of 50) with each sample lasting for a specified number of milliseconds—a maximum of 500 ms. The thread dump is executed on the node monitored by the agent.

Users need the Agent Advanced Operation permission to request a thread dump.

Agent Limit on Thread Dumps

One thread dump operation is executed at a time. They are not executed in parallel. If additional thread dump requests are received while one is being executed, they are queued with a limit of five per agent.

If the five thread dumps per agent limit is exceeded, the console shows an event with a thread dump operation that was skipped because of the limit and the associated action dialog for the executed policy links to this event.

To create a thread dump action:

1. Access the Create Action pane.
2. Under **Diagnostics**, select **Take a thread dump** in the **Create Action** pane and click **OK**.
3. Enter a name for the action, the number of samples to take and the interval between the thread dumps in milliseconds. If you want to mandate an approval before the thread dump action can be started, check the **Require approval before executing this Action** checkbox and enter the email address of the individual or group that is authorized to approve the action. See Actions Requiring Approval for more information.
4. Click **OK**.

When a thread dump action is triggered by a backend discovery event, if the backend is not resolved quickly the policy will not start the thread dump.

Remediation Actions

A remediation action runs a local script in a node. The script executes on the machine from which it was invoked or on the node specified by the remediation action configuration. You can use this type of action to automate your runbook procedures. You can optionally configure the remediation action to require human approval before the script is started. See Actions Requiring Approval.

> Remediation actions are not available for servers. Select an **Application** and then create **Actions** to view the **Remediation Action** option.

Prerequisites for Local Script Actions

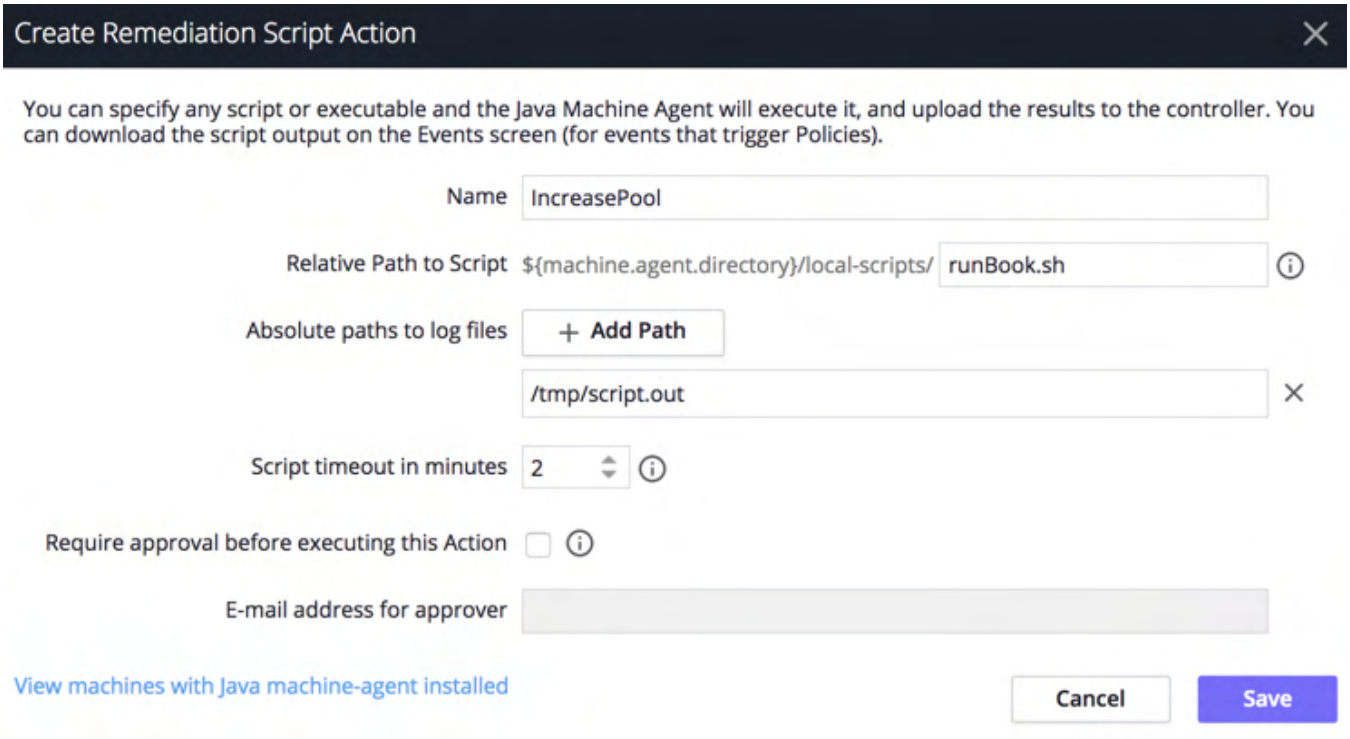
- The Standalone Machine Agent must be installed running on the host on which the script executes. To see a list of installed Machine Agents for your application, click **View machines with machine-agent installed** in the bottom left corner of the remediation script configuration window.
- To be able to run remediation scripts, the Machine Agent must be connected to an on-premises Controller or to a SaaS Controller via SSL. Remediation script execution is disabled if the Machine Agent connects to a SaaS Controller on an unsecured (non-SSL) HTTP connection.
- The Machine Agent OS user must have full permissions to the script file and the log files generated by the script and/or its associated child processes.
- The script must be placed in `<agent install directory>\local-scripts`.
- The script must be available on the host on which it executes.
- Processes spawned from the scripts must be daemon processes.

Remediation Scripts

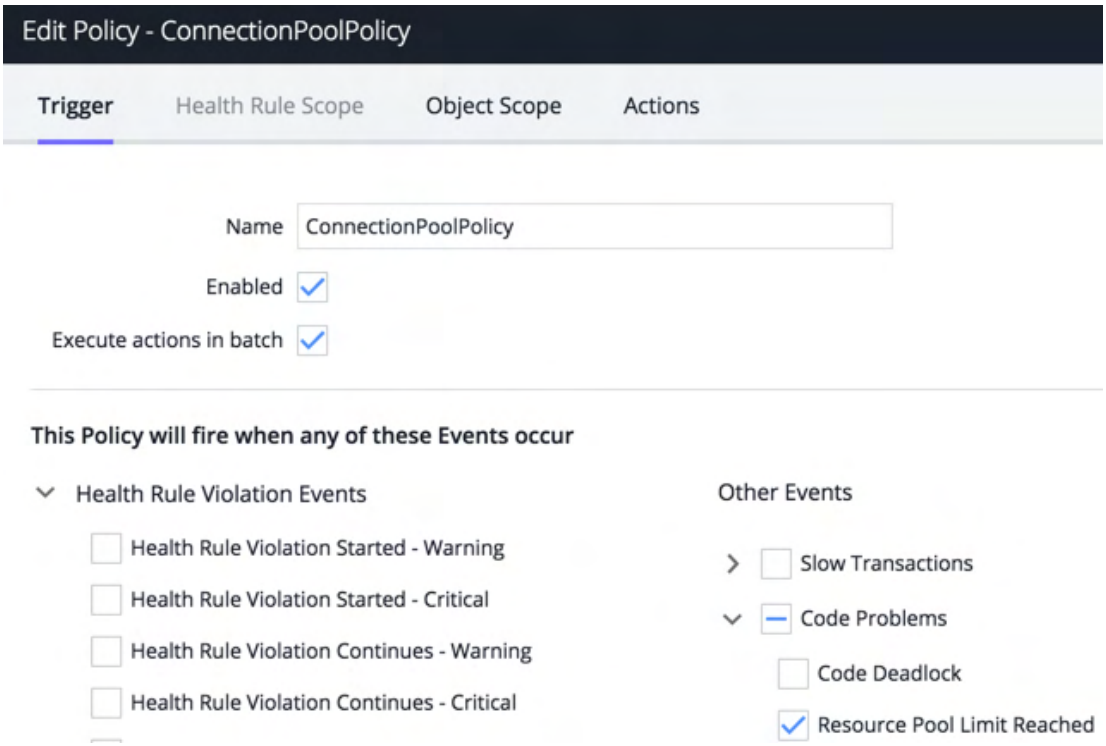
A remediation script is run on the machines that you specify in the remediation script configuration. You can run the script from the machine affected by the violation that triggered the action or from a central management server. It is not necessary for an app agent to be running on the machine on which the script executes; just a Machine Agent.

Remediation Example

The remediation action, named `increasePool`, executes a local script named `runbook.sh`, which increases the size of the connection pool on the JVM.



A policy named **ConnectionPoolPolicy** triggers this action when the Resource Pool Limit Event starts:



Create a Local Script (Remediation) Action

To create a remediation action:

1. Access the **Create Action** pane. See Create and Modify Actions in Actions.
2. Select **Run a script or executable on problematic Nodes** in the **Create Action** pane and click **OK**.
3. After entering a name for the action, in the field that terminates the Relative path to script entry, enter the rest of the path to the executable script. Remediation scripts must be stored in a sub-directory of the machine agent installation. The sub-directory must be named `local-scripts`. These paths are all valid:

```
${machine.agent.directory}/local-scripts/runMe.sh
${machine.agent.directory}/local-scripts/johns_scripts/runMe.sh
${machine.agent.directory}/local-scripts/ops/johns_scripts/runMe.sh
```

4. Click the **+** to enter the absolute paths of any log files that the script writes to that you want to be included in the script output.
5. Enter the timeout period for the script process in minutes.
6. To mandate an approval before the script action can be started, select the **Require approval before executing this Action** check box and enter the email address of the individual or group that is authorized to approve the action. See Actions Requiring Approval for more information.

Specify the Nodes on Which the Action Will Run

When you bind the action to a policy, you specify the nodes on which the script should execute. You can configure the number of nodes or the percentage of nodes or you can configure a specific node. This flexibility allows you to configure scripts to run from a central management server, not just the node on which the violation occurred.

In the **Configure Action** pane of the policy actions to execute, either:

- Select **Execute Action on Affected Nodes** and the percentage of the nodes or the number of nodes on which to run the script.
- To designate the specific node on which to run the script, select **Execute Action on Specified Node,** and select the node on which the script should run from the popup node browser. You can either save the configuration or change it to designate a different node.

Review the Local Script Output

1. In the **Events** list, locate the row for the event that triggered the action for which you want to see the results.
2. In the **Actions** column of the selected row, click the remediation script icon.
3. In the script results list, select the script output that you want and click **Download Local Script Result**.

When a remediation action is triggered by a backend discovery event, if the backend is not resolved quickly the policy will not start the local script.

Remediation Scripts

A remediation script is run on the machine that you specify in the remediation script configuration. You can run the script from the machine affected by the violation that triggered the action or from a central management server. It is not necessary for an app agent to be running on the machine on which the script executes; just a Machine Agent.

Guidelines for Remediation Scripts

By default, the script is a shell script in /bin/sh invoked with the `-ex` option, unless the script has a header, in which case the interpreter in the header is used. For example, if the script header is `#!/bin/`perl, the Perl interpreter is invoked.

A process exit code of zero indicates that the script execution succeeded. A non-zero exit code indicates that it failed.

The script should be written as generically as possible to allow it to run on any of the nodes for which is it invoked. AppDynamics exports the following environment variables to the script runtime to provide context regarding the environment and the event that triggered the action.

Environment Variable	Cardinality (1 or *M*)	Notes
`APP_ID`	1	Name of the Application
`EVENT_TIME`	1	Timestamp of the event
`EVENT_ID`	1	Event ID
`EVENT_TYPE`	1	type of event, such as: `ERROR, APPLICATION_ERROR, APPLICATION_INFO, STALL, BT_SLA_VIOLATION, DEADLOCK, MEMORY_LEAK, MEMORY_LEAK_DIAGNOSTICS, LOW_HEAP_MEMORY, ALERT, CUSTOM, APP_SERVER_RESTART, BT_SLOW, SYSTEM_LOG, INFO_INSTRUMENTATION_VISIBILITY, AGENT_EVENT, INFO_BT_SNAPSHOT, AGENT_STATUS, SERIES_SLOW, SERIES_ERROR, ACTIVITY_TRACE, OBJECT_CONTENT_SUMMARY, DIAGNOSTIC_SESSION, HIGH_END_TO_END_LATENCY, APPLICATION_CONFIG_CHANGE, APPLICATION_DEPLOYMENT, AGENT_DIAGNOSTICS, MEMORY, LICENSE`
`ENV_STARTUP_ARGS`	1	Process args
`ENV_SYSTEM_PROPERTIES`	1	JVM System Props, when Java
`AFFECTED_ENTITY`	1	Affected Entity that triggered the event

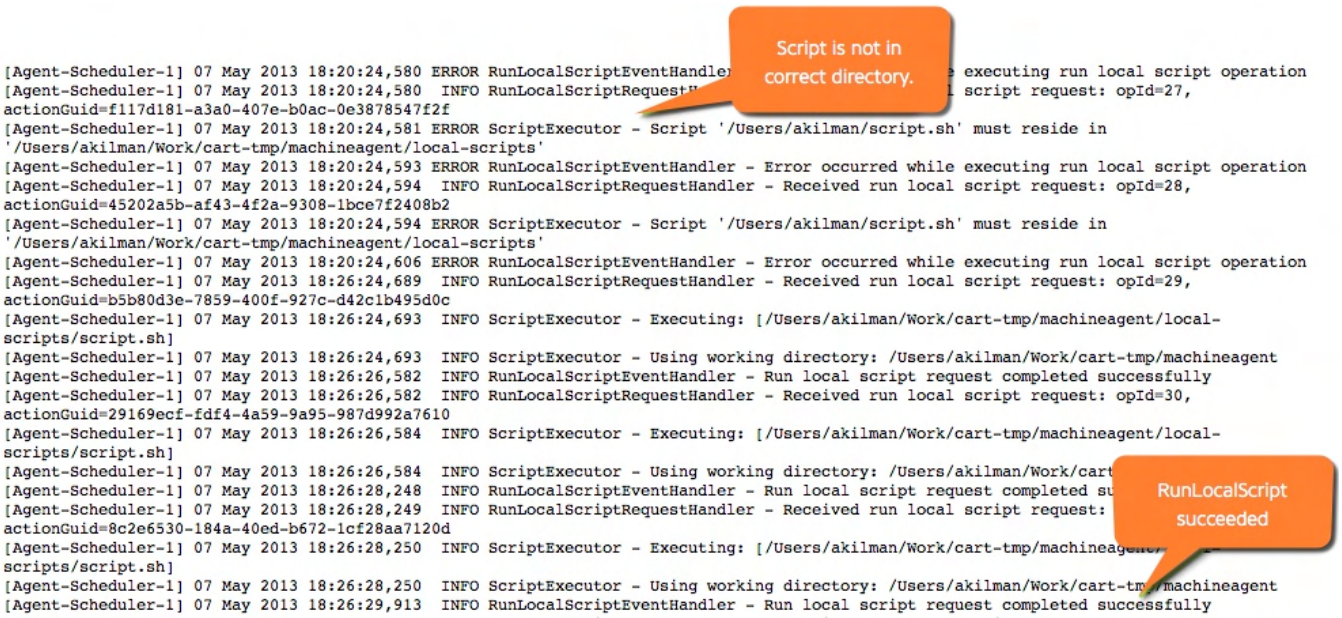
Remediation scripts must be stored in a sub-directory of the machine agent installation. The sub-directory must be named `local-scripts`. The following paths are all valid.

```
${machine.agent.home}/local-scripts/runMe.sh
${machine.agent.home}/local-scripts/johns_scripts/runMe.sh
${machine.agent.home}/local-scripts/ops/johns_scripts/runMe.sh
```

Troubleshooting Remediation Scripts

To troubleshoot your remediation script, look for the process in the Machine Agent log. The log location path is: `<machine_agent_home>/logs/machine-agent.log`

This snippet from the Machine Agent log shows both error and success messages from running a local script named `script.sh`.




```
[Agent-Scheduler-1] 07 May 2013 18:20:24,580 ERROR RunLocalScriptEventHandler  executing run local script operation
[Agent-Scheduler-1] 07 May 2013 18:20:24,580  INFO RunLocalScriptRequestH  l script request: opId=27,
actionGuid=f117d181-a3a0-407e-b0ac-0e3878547f2f
[Agent-Scheduler-1] 07 May 2013 18:20:24,581 ERROR ScriptExecutor - Script '/Users/akilman/script.sh' must reside in
'/Users/akilman/Work/cart-tmp/machineagent/local-scripts'
[Agent-Scheduler-1] 07 May 2013 18:20:24,593 ERROR RunLocalScriptEventHandler - Error occurred while executing run local script operation
[Agent-Scheduler-1] 07 May 2013 18:20:24,594  INFO RunLocalScriptRequestHandler - Received run local script request: opId=28,
actionGuid=45202a5b-af43-4f2a-9308-1bce7f2408b2
[Agent-Scheduler-1] 07 May 2013 18:20:24,594 ERROR ScriptExecutor - Script '/Users/akilman/script.sh' must reside in
'/Users/akilman/Work/cart-tmp/machineagent/local-scripts'
[Agent-Scheduler-1] 07 May 2013 18:20:24,606 ERROR RunLocalScriptEventHandler - Error occurred while executing run local script operation
[Agent-Scheduler-1] 07 May 2013 18:26:24,689  INFO RunLocalScriptRequestHandler - Received run local script request: opId=29,
actionGuid=b5b80d3e-7859-400f-927c-d42c1b495d0c
[Agent-Scheduler-1] 07 May 2013 18:26:24,693  INFO ScriptExecutor - Executing: [/Users/akilman/Work/cart-tmp/machineagent/local-
scripts/script.sh]
[Agent-Scheduler-1] 07 May 2013 18:26:24,693  INFO ScriptExecutor - Using working directory: /Users/akilman/Work/cart-tmp/machineagent
[Agent-Scheduler-1] 07 May 2013 18:26:26,582  INFO RunLocalScriptEventHandler - Run local script request completed successfully
[Agent-Scheduler-1] 07 May 2013 18:26:26,582  INFO RunLocalScriptRequestHandler - Received run local script request: opId=30,
actionGuid=29169ecf-fdf4-4a59-9a95-987d992a7610
[Agent-Scheduler-1] 07 May 2013 18:26:26,584  INFO ScriptExecutor - Executing: [/Users/akilman/Work/cart-tmp/machineagent/local-
scripts/script.sh]
[Agent-Scheduler-1] 07 May 2013 18:26:26,584  INFO ScriptExecutor - Using working directory: /Users/akilman/Work/cart
[Agent-Scheduler-1] 07 May 2013 18:26:28,248  INFO RunLocalScriptEventHandler - Run local script request completed su
[Agent-Scheduler-1] 07 May 2013 18:26:28,249  INFO RunLocalScriptRequestHandler - Received run local script request:
actionGuid=8c2e6530-184a-40ed-b672-1cf28aa7120d
[Agent-Scheduler-1] 07 May 2013 18:26:28,250  INFO ScriptExecutor - Executing: [/Users/akilman/Work/cart-tmp/machineagent/
scripts/script.sh]
[Agent-Scheduler-1] 07 May 2013 18:26:28,250  INFO ScriptExecutor - Using working directory: /Users/akilman/Work/cart-tmp/machineagent
[Agent-Scheduler-1] 07 May 2013 18:26:29,913  INFO RunLocalScriptEventHandler - Run local script request completed successfully
```

HTTP Request Actions and Templates

An HTTP request action sends an HTTP request in response to an event. These types of actions allow you to integrate AppDynamics' policies with third-party HTTP APIs.

You can configure an HTTP/HTTPS proxy for an on-premise Controller to filter access to the web server. When you configure a proxy, the HTTP request actions are routed through the HTTP proxy or HTTPS proxy, depending on the endpoint. See Configure HTTP/HTTPS Proxy.

You create an HTTP request action using an HTTP request template. The template describes the HTTP request and is reusable by different HTTP request actions within an AppDynamics account.

After you create a template, users in the Controller UI can create actions that use it. The template displays as an option after the user selects the **Make an HTTP Request** option in the **Create Action** pane and accesses the **Configure Action** pane.

Required Permissions

To create or modify HTTP Request Templates, users need the account-level Configure HTTP Request Templates permission.

Access HTTP Action Templates

To access existing templates or to create new ones, click **Alert & Respond** in the menu bar, and then select **HTTP Request Templates**.

You can either:

- Click an existing template from the HTTP Request Templates list to view, edit, or delete it.
- Click the **New +** icon to create a template.

Create or Modify an HTTP Request Template

When creating or modifying templates, click the **Info** icons in the template to get information about how to complete the configuration.

After you have created a template, click **Save** at the bottom to save it.

Custom Templating Variables

You can use variables that replace values in the URL path and payload when the HTTP request is sent. For-each loops are supported.

The template already knows a set of predefined variables, which are described in Predefined Templating Variables. Check this list before you create any custom templating variables. Chances are the variable you want to use has already been defined.

You can optionally configure custom variables if the predefined variables do not meet all your needs. When a predefined variable and a custom variable are both configured with the same variable name, the template uses the predefined variable.

The template uses Apache Velocity version 1.7 to process the variables. See the Velocity User Guide for details about usage.

Request URL

Enter the URL for the request in the Raw URL field and select the URL encoding scheme from the pulldown menu. Only UTF-8 and ISO_8858-1 are supported. Verify that the Encoded URL is the exact request to send.

Authentication

You can define basic authentication for your HTTP request. Select **BASIC authentication type** from the **Type** dropdown only if the communication is encrypted. If the communication is not encrypted, we recommend not to use any authentication. After selecting the authentication type, specify the authentication username and password.

Custom Request Headers

You can define custom headers for the HTTP requests. Custom Request Headers can contain custom templating variables or predefined templating variables encoded as $(VARIABLE_NAME).

For a complete list of predefined variables, see Predefined Templating Variables. Check this list before you create any custom templating variable. See the Velocity User Guide for details.

Payload

To include a payload in your HTTP request, define the **MIME Type** from the dropdown. You can encode the payload using UTF-8 or ISO-8859-1 encoding.

Response Handling Criteria

Success or Failure

If you do not specify any response-handling criteria for either success or failure, the HTTP response is always success.

Otherwise:

- Failure criteria are evaluated before success criteria. As soon as a match is found on a fail criterion, the response is set to failure and the remaining fail match conditions, if there are any, are not evaluated.
- If no failure criteria are matched, the success criteria are then evaluated.

If no success criteria are configured, the response will always be success as long as no fail criteria were previously matched.

Otherwise:

- If you specify at least one success criterion, there must be a match on a success criterion for the response to be success. In other words, if any success criteria are configured but not matched, the response will be failure.

Both failure and success criteria are considered in the order in which they appear in the template.

Criteria With and Without Payload

You can set separate criteria for the same status code. For example, you can set one criterion on a status with payload as success and another criterion on the same code without payload as failure.

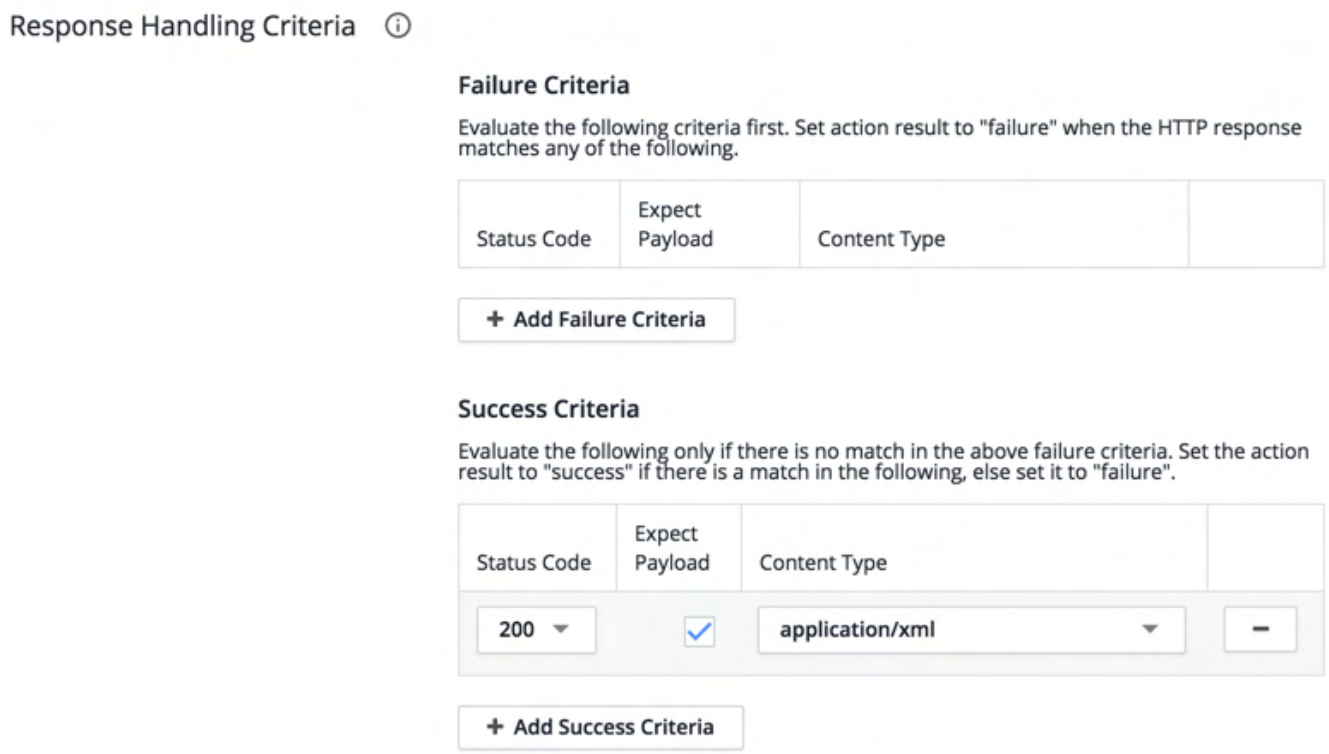
Or, you can set separate criteria for the same status code with payload of different content types. For example, payload of type `application/xml` and `application/xhtml+xml` might be success and payload of other types could be failure.

Content Types for Payload

You can specify the content type for the payload using the dropdown that displays for requests for which the **Expect Payload** checkbox is selected.

If you do not know the content type but the request expects payload, use `*/*` to specify all content types.

This configuration returns a success response for a 200 status code with an XML payload. Any other code will cause the request to return a failure response.



One Request Per Event

Several separate events, or separate occurrences of the same event, could potentially invoke the same HTTP action.

The **One Request Per Event** setting controls whether the action bundles the HTTP requests.

If this checkbox is clear, the HTTP request is sent once every minute, no matter how many events triggered the action within that time frame. This is the default.

If this checkbox is checked, the request is sent every time an event triggers it. In this case, if ten events trigger the action, the HTTP request is sent ten times, even if all those events occurred within a single minute.

You might want to send one request per event, checkbox checked, if you are trying to log tickets and your ticketing system API does not support bulk create. However, if it does, you probably want a single bulk request every minute, checkbox clear.

Event Clamp Limit

If you have not selected the **One Request Per Event** checkbox and the list of triggering events could be large, you might want to limit the display of the events that triggered the action in the response. The clamp limit is the *n* most recent triggering events. The default -1 indicates no limit. This setting is ignored if **One Request Per Event** is selected.

Timeouts and Redirects

- **Connect Timeout**: The maximum number of milliseconds to wait for the request to reach the server.
- **Socket Timeout**: The maximum number of milliseconds to wait to receive the response.
- **Max Redirects**: The maximum number of times that a single request can redirect.

Test the Template

To test the template, click **Test** at the bottom of the template configuration.

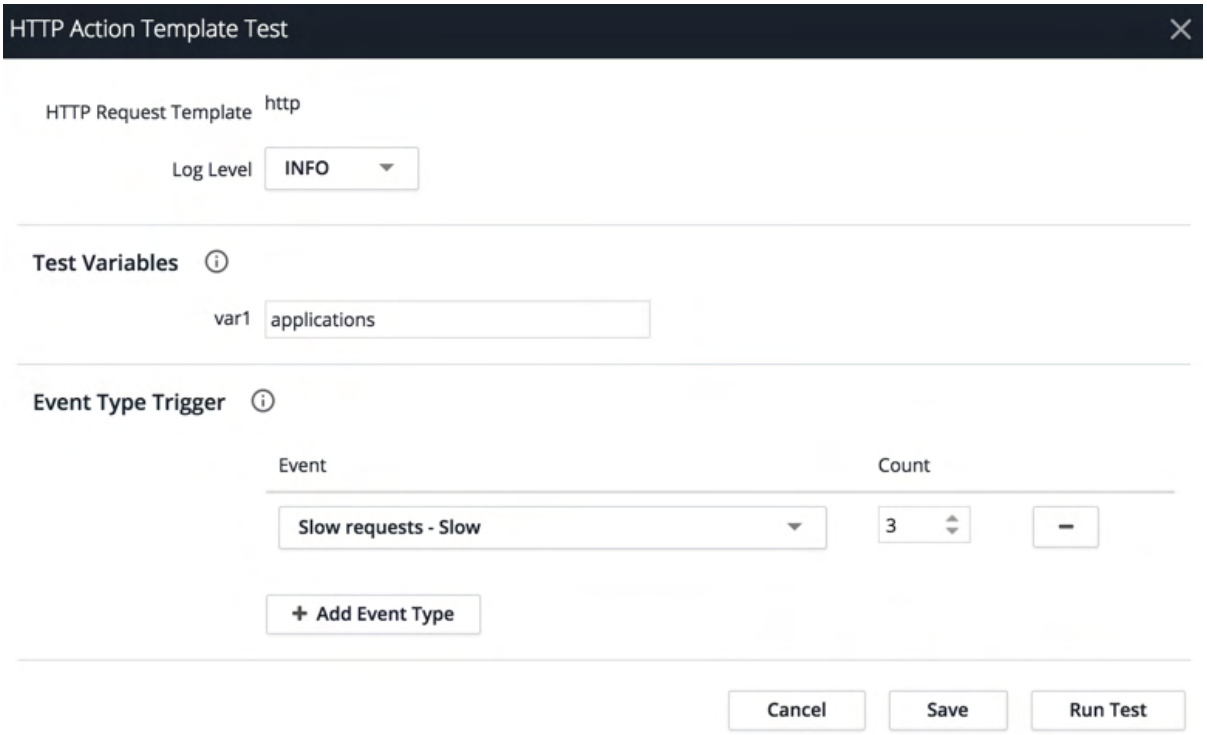
Configure Test

In the **Template Test** configuration pane, specify the test variables to use for the test. These variables may be different from those that will be used in the real requests that will be automatically sent when the HTTP action is live.

Also, specify the type of events that trigger for the test.

The log level defaults to **INFO**, but you can change this using the **Log Level** dropdown.

This sample test template configuration simulates three `POLICY OPEN CRITICAL` events triggering this action. More triggers could be added.



Run Test

When you run the test, the request is sent when it is triggered. You receive a transcript of the run.

If the results are not what you expect, you may need to modify the template test configuration, the HTTP action template, or both, before you use the template in an action.

JIRA Actions

A JIRA action allows AppDynamics to automatically create a JIRA issue when an AppDynamics event is triggered. The event triggers an HTTP custom action for creating a JIRA issue.

You can configure an HTTP/HTTPS proxy for an on-premise controller to filter access to the web server. When you configure a proxy, the JIRA actions are routed through the HTTP proxy or HTTPS proxy, depending on the endpoint. See Configure HTTP/HTTPS Proxy.

JIRA Action Details

The JIRA action is based on templates where each template can be preconfigured with the project, issue type, assignee, and priority. When a JIRA action is triggered, it creates a ticket with those fields in it.

There is a rate limit for the action, which is set to 100 by default. This means the JIRA action will not create more than 100 tickets a minute for an account. Administrators on on-prem Controllers can modify the default by configuring the flag, `max.jira.actions.per.min`, in the **Controller Settings** section of the Administration Console.

This new action needs to be created similar to the rest of the actions, and associated with a health rule for all transition events of the rule (OPEN, CONTINUES, UPGRADE, DOWNGRADE, CLOSE, CANCEL).

This table describes the JIRA action and transitions.

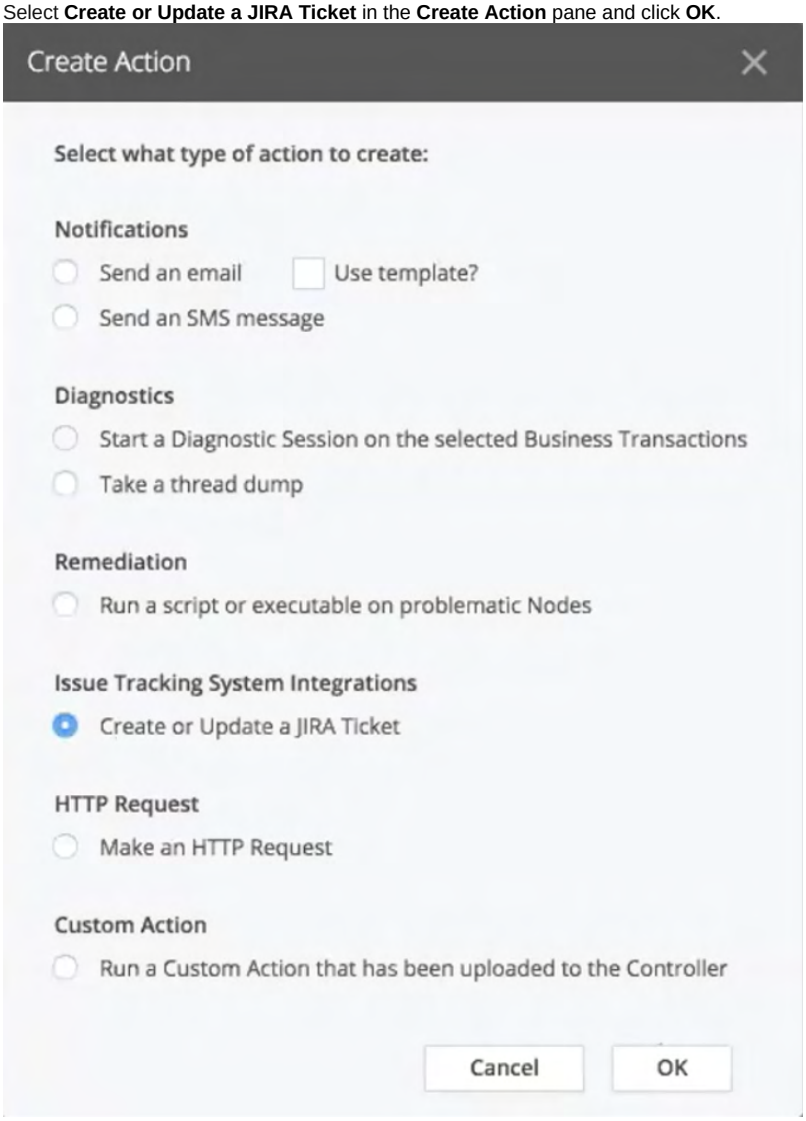
Health Rule Transition (Event)	Reason for Transition	JIRA Action
OPEN (CRITICAL or WARNING)	Health rule violation started.	Creates a JIRA issue. Response of this action provides the JIRA ID which is persisted against the incident.
CONTINUES (CRITICAL or WARNING)	Health rule violation continues.	Fetches the JIRA issue that corresponds to the health rule / affected entity, and updates the JIRA with a comment with the latest violation summary.
UPGRADE (WARNING CRITICAL)	Health rule violation now critical.	Fetches the JIRA issue that corresponds to the health rule / affected entity, and updates the JIRA with a comment with the latest violation summary. May change the priority of the issue.
DOWNGRADE (CRITICAL WARNING)	Health rule violation now warning.	Fetches the JIRA issue that corresponds to the health rule / affected entity, and updates the JIRA with a comment with the latest violation summary. May change the priority of the issue.
CLOSE	Health rule thresholds back within limits, incident closed.	Fetches the JIRA issue that corresponds to the health rule / affected entity, and resolves the JIRA.
CANCEL	Health rule configuration has changed or system lifecycle (restart, entity deleted) caused incident to be cancelled.	Fetches the JIRA issue that corresponds to the health rule / affected entity, and resolves the JIRA.

Create a JIRA Action

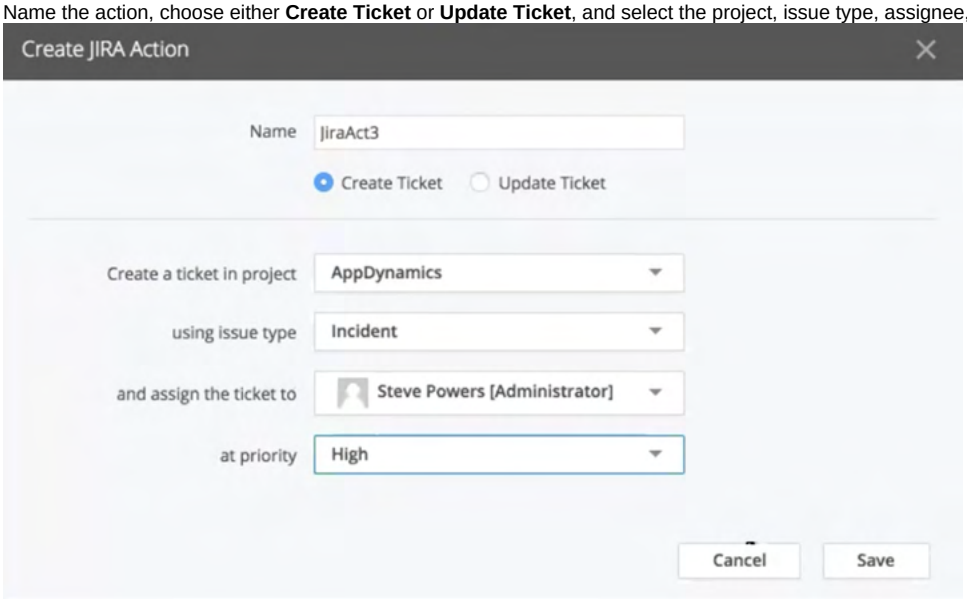
To create a JIRA action:

1. Access the **Actions Configuration** pane. See Create and Modify Actions in Actions.

2. Select **Create or Update a JIRA Ticket** in the **Create Action** pane and click **OK**.



3. Name the action, choose either **Create Ticket** or **Update Ticket**, and select the project, issue type, assignee, and priority from the dropdowns.



4. Click **Save**.

You can now associate a health rule with a policy that executes the JIRA action. The Atlassian JIRA integration must be set up for the alerts and policies to run.

Configure HTTP HTTPS Proxy

You can configure an HTTP/HTTPS proxy for an on-premise Controller. When you configure a proxy, the HTTP request actions or JIRA actions are routed through the HTTP proxy or the HTTPS proxy, depending on the endpoint.

Add HTTP Proxy Details to the Configuration File

The `domain.xml` configuration file is the global configuration for managed hosts. Add the HTTP proxy details to the `domain.xml` file as follows:

1. Locate and edit `domain.xml` for the domain.
   This is usually located under `$JBOSS_HOME/domain/configuration/`.
2. Add the following JVM option to enable the HTTP proxy support for alerting actions:

   ```
   <jvm-options>-Dappdynamics.alerting.actions.http.enable.proxy=true</jvm-options>
   ```

   The HTTP proxy support for alerting actions is disabled by default.

   > You can also enable the HTTP proxy support for alerting actions at, `/private/ControllerFlagsServlet` endpoint. You must have admin credentials to enable this option.

3. Add the following JVM options to configure the HTTP and HTTPS proxy authentication parameters, namely, host, port, user, and password.

   ```
   <jvm-options>-Dhttp.proxyHost=$HOST</jvm-options>
   <jvm-options>-Dhttp.proxyPort=$PORT</jvm-options>
   <jvm-options>-Dhttp.proxyUser=$USER</jvm-options>
   <jvm-options>-Dhttp.proxyPassword=$PASSWORD</jvm-options>
   <jvm-options>-Dhttps.proxyHost=$HOST</jvm-options>
   <jvm-options>-Dhttps.proxyPort=$PORT</jvm-options>
   <jvm-options>-Dhttps.proxyUser=$USER</jvm-options>
   <jvm-options>-Dhttps.proxyPassword=$PASSWORD</jvm-options>
   ```

   Ensure that you provide a plain text password only.
   Proxy parameter changes are not hot-swappable. This means that you need to restart the Controller to ensure that the changes are reflected.

Custom Actions

Deployment Support

This page provides an overview of custom actions in AppDynamics. A custom action is typically used to integrate third-party alerting and ticketing systems. A custom action differs from other actions on its singular execution on the controller instance.

The custom action is made up of a custom action script and a custom.xml file (you must create the file before you create an action that uses them). The custom action scripts include parameters for specifying the affected entity, for example, the tier, node, business transaction, and so on. See Build a Custom Action for details on how to create the custom action script and XML file.

Custom actions are commonly used when you want to trigger a human workflow or leverage an existing alerting system that is external to AppDynamics. For example, you could use a custom action to file a JIRA ticket when AppDynamics reports that a connection pool is near saturation.

Create a Custom Action

After the custom action script and custom.xml files have been tested manually and installed on the Controller, restart the Controller. Then, you can create the custom action.

To create a custom action:

1. Access the **Create Action** pane. See Create and Modify Actions in Actions.
2. Select **Run a Custom Action that has been uploaded to the Controller**.
3. Click **OK**.
4. Enter a name for the action.
5. Select the custom action from the dropdown. All the custom actions you have created are listed here.
6. Click **Save**.

The custom action is now available for assignment to a policy.

Disable Custom Actions on a Multi-Tenant Controller

Because a custom action script has access to the Controller file system, you may decide to disable this functionality in a multi-tenant environment for security reasons.

To disable custom actions:

1. Open the Controller admin console.
2. Go to **Controller Settings**.
3. Search for the `aas.multitenant.custom.action.local.execution.enabled` property.
4. Set to `false`.
5. Click **Save**.

Build a Custom Action

This page provides instructions on custom actions in AppDynamics.

You can set up a custom action on the Controller instance to integrate notification of AppDynamics health rule violations and events with an alerting or ticketing system. Use a push approach by creating custom notifications that pass the information to your alerting system.

Custom Notifications and Custom Actions

A custom notification lets you integrate alerts about AppDynamics health rule violations and events into your own alerting system. This integration extension requires:

- A `custom.xml` file that provides information about the custom notification.
- An executable script that accepts parameters from AppDynamics about the events and health rule violations that trigger an alert.
- Configuring AppDynamics events or policies to trigger the custom notification via a custom action.

Create a Custom Action

Create the Script

For each custom action that you want to implement, create an executable script (`.bat` extension for Windows, `.sh` extension for Linux) that can accept and process the parameters passed to it by AppDynamics. See Information Passed to the Custom Action Script from AppDynamics for details on the parameters.

For each script:

- Set correct executable permissions for the shell scripts in a Linux environment. For example, chmod `770 script1.sh`.
- Ensure that the script file has the correct character encoding. This is especially important when creating a Unix shell script on a Windows machine.

Install the Script on an On-Premises Controller

To install the script on an on-premises Controller:

1. At the top level of the Controller installation directory, create a directory named `custom` with a sub-directory named `actions`.

   ```
   <controller_home>/custom/actions
   ```

2. In the `<controller_home>/custom/actions` directory, create a subdirectory for each custom action script that you will install. For example, for an action that interfaces with a JIRA system.

   ```
   <controller_home>/custom/actions/jira
   ```

3. Move the script to the appropriate subdirectory that you created.

Create the XML File

1. Create a `custom.xml` file that describes the location and name of your custom action script. See Contents of the custom.xml File.
2. For an on-premises Controller, move the file to the `<controller_home>/custom/actions` directory. For a SaaS Controller, contact your AppDynamics account representative for instructions.

Verify the Script on an On-Premises Controller

1. After you have installed the script and the custom.xml file, restart the Controller.
2. Verify the script manually. To verify the script:
   a. Open a command-line console on the Controller host machine.
   b. Execute the script file from the command line console.

Create the Custom Action

To arrange how a custom action will be triggered, see Custom Actions.

Contents of the custom.xml File

The `custom.xml` file has an `<actions>` element for every custom action on the Controller.

The `<type>` element contains the subdirectory that contains the script file.

The <executable> element contains the name of the script.

Sample custom.xml File

```
<custom-actions>
  <action>
     <type>jira</type>
     <executable>script1.bat</executable>
  </action>
  <action>
    <type>bugzilla</type>
    <executable>script2.sh</executable>
  </action>
</custom-actions>
```

Information Passed to the Custom Action Script from AppDynamics

The custom action script must handle the parameters that the Controller passes from the health rule violation or other event. The parameter values are passed as an array of strings.

The parameters are passed as $0 for the script name, then $1, $2, . . . $*n*. $1 is the first parameter (application name), $2 is the application id, and so on in the order in which they are documented in the sections below.

Health rule violations have a different set of parameters from events.

Parameters Passed by a Health Rule Violation

The parameters describe the violated health rule violation that triggered the action.

The total number of elements in the array depends on the number of entities evaluated by the health rule and the number of triggered conditions per evaluation entity. Examples of evaluation entities are application, tier, node, business transaction, JMX. For each evaluation entity, the script expects the entity type, entity name, entity id, number of triggered conditions, and for each triggered condition, the set of condition parameters.

The parameter values are passed in the order in which they are described.

Structure of Parameters Sent by a Health Rule Violation

- APP_NAME
- APP_ID
- PVN_ALERT_TIME
- PRIORITY
- SEVERITY // INFO, WARN, ERROR
- ACTION_NAME
- HEALTH_RULE_NAME
- HEALTH_RULE_ ID
- PVN_TIME_PERIOD_IN_MINUTES
- AFFECTED_ENTITY_TYPE
- AFFECTED_ENTITY_NAME
- AFFECTED_ENTITY_ID
- NUMBER_OF_EVALUATION_ENTITIES–The following parameters are passed for each evaluation entity:
    - EVALUATION_ENTITY_TYPE
    - EVALUATION_ENTITY_NAME
    - EVALUATION_ENTITY_ID
    - NUMBER_OF_TRIGGERED_CONDITIONS_PER_EVALUATION_ENTITY–The following parameters are passed for each triggered condition for this evaluation entity:
        - SCOPE_TYPE_x
        - SCOPE_NAME_x
        - SCOPE_ID_x
        - CONDITION_NAME_x
        - CONDITION_ID_x
        - OPERATOR_x
        - CONDITION_UNIT_TYPE_x
        - USE_DEFAULT_BASELINE_x
        - BASELINE_NAME_x
        - BASELINE_ID_x
        - THRESHOLD_VALUE_x
        - OBSERVED_VALUE_x
- SUMMARY_MESSAGE
- INCIDENT_ID
- DEEP_LINK_URL
- EVENT_TYPE
- ACCOUNT_NAME

- ACCOUNT_ID
- TAG

Definitions of Parameters Sent by a Health Rule Violation

Health Rule Violation Parameter	Definition
APP_NAME	Name of the business application.
APP_ID	Application ID number.
PVN_ALERT_TIME	Alert time, such as Thu Dec 22 15:03:56 PST 2011.
PRIORITY	Integer designating how urgently a health rule violation should be fixed. The lowest number (0) is the most urgent.
SEVERITY	INFO, WARN, or ERROR. In the Controller UI, they are called Info, Warning, and Critical.
ACTION_NAME	Name of the action to be invoked post a health rule violation.
HEALTH_RULE_NAME	Name of the health rule that was violated.
HEALTH_RULE_ ID	Health rule ID.
PVN_TIME_PERIOD_IN_MINUTES	Health rule violation time period in minutes.
AFFECTED_ENTITY_TYPE	APPLICATION, APPLICATION_COMPONENT (aka Tier), APPLICATION_COMPONENT_NODE, BUSINESS_TRANSACTION, APPLICATION_DIAGNOSTIC_DATA (aka Error).
AFFECTED_ENTITY_NAME	The affected entity name.
AFFECTED_ENTITY_ID	The affected entity ID.
NUMBER_OF_EVALUATION_ENTITIES	Number of entities (Business Transactions, Applications, Tiers, Nodes, Errors, JMX counters, and so on) violating the health rule conditions
EVALUATION_ENTITY_TYPE	APPLICATION, APPLICATION_COMPONENT (aka Tier), APPLICATION_COMPONENT_NODE, BUSINESS_TRANSACTION, APPLICATION_DIAGNOSTIC_DATA (aka Error), JMX.
EVALUATION_ENTITY_NAME	The evaluation entity name (for JMX it is the counter name).
EVALUATION_ENTITY_ID	The evaluation entity ID or <NULL> for JMX.
NUMBER_OF_TRIGGERED_CONDITIONS_PER_EVALUATION_ENTITY	Number of times to loop through the triggered condition parameters for each evaluation entity; if more than one condition is triggered, the parameters repeat for each triggered condition, where x is the position of the condition.
SCOPE_TYPE_x	The scope of the parameter, whether the scope is the application, tier, or node: APPLICATION, APPLICATION_COMPONENT, APPLICATION_COMPONENT_NODE.
SCOPE_NAME_x	The name of the scope, such as ACME Book Store Application.
SCOPE_ID_x	The scope ID.
CONDITION_NAME_x	The health rule condition name.
CONDITION_ID_x	The health rule condition ID.
OPERATOR_x	Allowed operators: LESS_THAN, LESS_THAN_EQUALS, GREATER_THAN, GREATER_THAN_EQUALS, EQUALS, NOT_EQUALS.
CONDITION_UNIT_TYPE_x	The condition for the threshold parameter: ABSOLUTE, BASELINE_STANDARD_DEVIATION, BASELINE_PERCENTAGE, BASELINE_PERCENTILE.
USE_DEFAULT_BASELINE_x	A Boolean parameter (true or false) applies only when the condition unit type is one of the BASELINE_ types.
BASELINE_NAME_x	Applicable only when the condition unit type is one of the BASELINE_ types and the use *default baseline* parameter is false.
BASELINE_ID_x	Applicable only when the condition unit type is one of the BASELINE_ types and the use *default baseline* parameter is false.
THRESHOLD_VALUE_x	Health rule threshold setting.
OBSERVED_VALUE_x	Value that violated the health rule threshold.

SUMMARY_MESSAGE	Summary of the notification, such as Health rules have been violated.
INCIDENT_ID	The incident identifier number for this health rule violation. Incident ID is unique within the Controller. The field is defined as int(11) which means it takes four bytes of space that is 32 bits of space with 2^(31) - 1 = 2147483647 max value and -2147483648 min value. One bit is for sign.
DEEP_LINK_URL	Controller deep link URL, such as: `http://<controller-host-url>/#location=APP_INCIDENT_DETAIL&incident=<incident-id>` Append the incident ID to the URL to provide a link to the Controller UI for this policy violation.
EVENT_TYPE	POLICY_OPEN_WARNING, POLICY_OPEN_CRITICAL, POLICY_CLOSE_WARNING, POLICY_CLOSE_CRITICAL, POLICY_UPGRADED, POLICY_DOWNGRADED, POLICY_CANCELED_WARNING, POLICY_CANCELED_CRITICAL, POLICY_CONTINUES_CRITICAL, and POLICY_CONTINUES_WARNING.
ACCOUNT_NAME	Name of the account in which the action was triggered.
ACCOUNT_ID	ID of the account in which the action was triggered.
TAG	Tag specified by the user or the empty string if no tag was specified.

Parameters Passed by an Event

The parameters describe the event that triggered the action.

The total number of elements in the array depends on the number of event types and event summaries that triggered the action.

The parameter values are passed in the order in which they are described.

Structure of Parameters Sent by an Event

- APP_NAME
- APP_ID
- EN_TIME
- PRIORITY
- SEVERITY
- EN_NAME
- EN_ ID
- EN_INTERVAL_IN_MINUTES
- NUMBER_OF_EVENT_TYPES
  The following parameters are passed for each event type:
  - EVENT_TYPE_x
  - EVENT_TYPE_NUM_x
- NUMBER_OF_EVENT_SUMMARIES
  The following parameters are passed for each event summary:
  - EVENT_SUMMARY_ID_x
  - EVENT_SUMMARY_TYPE_x
  - EVENT_SUMMARY_SEVERITY_x
  - EVENT_SUMMARY_STRING _x
- DEEP_LINK_URL
- ACCOUNT_NAME
- ACCOUNT_ID
- TAG

Definitions of Parameters Sent by an Event

Event Notification Parameter	Definition
APP_NAME	Name of the business application.
APP_ID	Application ID number.
EN_TIME	Event notification time, for example, Wed Jan 04 09:36:55 PST 2012.
PRIORITY	Integer designating how urgently a health rule violation should be fixed, with the lowest number (0) the most urgent.
SEVERITY	Allowed values: INFO, WARN, or ERROR. In the AppDynamics UI they are called Info, Warning, and Critical.
EN_NAME	Name of the event notification.
EN_ID	Event notification ID number.
EN_INTERVAL_IN_MINUTES	Event notification interval in minutes.
NUMBER_OF_EVENT_TYPES	Determines how many times to loop through the event type map parameters.

EVENT_TYPE_x	If there is more than one event type, the parameters repeat for each event type, where x increments the number representing the event type.
EVENT_TYPE_NUM_x	Number of events of this type.
NUMBER_OF_EVENT_SUMMARIES	Number of event summaries in the notification that determines how many times to loop through the event summary parameters.
EVENT_SUMMARY_ID_x	Event summary ID number.
EVENT_SUMMARY_TIME_x	Event summary time, for example: Wed Jan 04 09:34:13 PST 2012.
EVENT_SUMMARY_TYPE_x	Type of event, such as: APPLICATION_CONFIG_CHANGE, APP_SERVER_RESTART, DIAGNOSTIC_SESSION, STALL.
EVENT_SUMMARY_SEVERITY_x	Event severity, such as: INFO, WARN or ERROR. In the Controller UI, they are called Info, Warning, and Critical.
EVENT_SUMMARY_STRING_x	Event summary string, such as: Application Server environment variables changed.
DEEP_LINK_URL	http://<controller-host-url>/#location=APP_EVENT_VIEWER_MODAL&eventSummary= Append each event summary ID to the URL to provide a link to the Controller UI for this event.
ACCOUNT_NAME	Name of the account in which the action was triggered.
ACCOUNT_ID	ID of the account in which the action was triggered.
TAG	Tag specified by the user or the empty string if no tag was specified.

Action Suppression

You can temporarily suppress a policy's automatic invocation of actions and alerts. You may want to do this while you are performing maintenance on or troubleshooting a component.

To see action suppression configurations created for an application, click **Alert & Respond** > **Actions** > **Action Suppression**.

Objects Affected by Action Suppression

You can configure action suppression for a specific time period to apply to a specific object or several objects. These entities can be the objects of action suppression:

- Application
- Business Transaction
- Tier
- Node
- Server

Within the time period configured for the action suppression, no policy actions are invoked for health rule violation events that occur on the specified object (s).

You can disable the reporting of metrics for agents associated with the object for which actions are suppressed. This option can cause reported metrics to change without notice.

If you see a sudden unexpected change in reported metrics for an object, review the action suppression configurations list to determine whether action suppression with reporting disabled is currently active for that object.

You can configure the entities that the action suppression affects for each object. This table lists the entities for which you can suppress the actions, depending on the configured object scope.

Object Scope	Affected Entities
Application	All entities within the application.
Business Transaction	• All business transactions in the application. • All business transactions within specific tiers. • Specific business transactions. • Business transactions with names having patterns that match certain criteria (such as all business transactions with names that start with "XYZ").
Tiers	• All tiers in the application. • Specific tiers. In a tier-level configuration, the suppression affects: • all the nodes in the specified tier(s) • individual node health rules as well as tier-level health rules For example, if a tier-level health rule is configured to invoke an action when a percentage of the nodes violates the condition, and then action suppression is configured on certain nodes in that tier, those nodes are still evaluated by the tier-level health rule.
Nodes	• All nodes • Java nodes • .NET nodes • PHP nodes Within these node types you can suppress actions for: • All nodes • Nodes in specific tiers • Specific nodes • Nodes with names, meta-info, environment variables or JVM system environment properties with specified matching criteria In a node-level configuration, the suppression affects node health rules only.
Servers	• All servers • Specific servers

Health Rules Affected by Action Suppression

By default, an action suppression configuration applies to actions triggered by all events that are generated by the configured objects.

You can refine the configuration to apply only to actions triggered by specific health rule violations. For example, if an application contains HealthRuleA, HealthRuleB and HealthRuleC, but only HealthRuleC is configured for action suppression, actions will continue to be invoked for violations of HealthRuleA and HealthRuleB during the configured time period.

Disable Health Rule Evaluation

Suppressing an action does not suppress the evaluation of any health rules linked to the action through a policy. Health rule violation events continue to be raised even when actions are suppressed. You can turn off the evaluation of a specific health rule or all health rules. See Disable All Health Rules.

Configure and Manage Action Suppressions

You can create, edit, or delete an action suppression in the Action Suppression tab. To access the action suppression configuration, click **Alert & Respond** > **Actions** > **Action Suppression**.

Create an Action Suppression

> When you create a new configuration, configure the panels in the same order as in the interface, because the configuration of the object scope determines the available affected entities presented in the object scope panel.

1. In the **Action Suppression** panel, click the **Create +** icon. The **Create Action Suppression** wizard appears.
2. In the **Overview** panel, enter a name for the action suppression. You can use alphanumeric characters to specify a name. Ensure that you specify a unique name.
3. To prevent the agents defined in the object scope from reporting metric data during the action suppression active time frame, select **Disable Agent Reporting**.
4. Select a time zone from the **Time Zone** dropdown.
5. To suppress actions only once, configure these **One Time** settings:
    a. The time to start action suppression. You can choose:
        - **Now**—to suppress actions immediately.
        - **After**—to suppress actions after the defined number of minutes.
        - **Date**—to suppress actions on a date, at a defined time.
    b. The time to stop action suppression. You can choose:
        - **After**—to stop suppressing the actions after the defined number of minutes.
        - **Date**—to stop suppressing actions on a date, at a defined time.
6. To suppress actions on a recurring basis, configure these **Recurring** settings:
    - **Daily**—to suppress actions every day from the defined **Start Time** to the **End Time**.
    - **Weekly**—to suppress actions on a specific day of a week from the defined **Start Time** to the **End Time.**
    - **Monthly—**to suppress actions on a specific date or a day of the month.
        - **Specific date—**to suppress actions starting on a specified **Start Date** and **Start Time** through the **End Date** and **End Time**.
        - **Specific day**—to suppress actions starting on a specified day and **Start Time** through the **End Time**.
7. In the **Object Scope** panel, select what entities that the action suppression affects.
   Depending on the type of object, you can configure the corresponding entities that are affected. See Entities Affected for information about the type of entities that can be affected by various objects.
8. To suppress actions only for the violations caused by specified health rules:
    a. In the **Health Rule Scope** panel, select **Suppress Action execution for these specified Health Rules.**
    b. Click **Add**. The **Health Rule Selection** dialog appears.
    c. Select the health rules and click **Select Health Rule(s)**.
9. Click **Create**.

Edit an Action Suppression

To modify an action suppression configuration, click the **Edit** icon.

Delete an Action Suppression

To delete a configured action suppression, click the **Delete** icon. A dialog asks you to confirm the deletion.

Predefined Templating Variables

This page describes the predefined variables that can be used in an HTTP request template or in an email template. The policy engine substitutes the value of the variable in the context of the triggering event(s) when it sends the actual request or email.

Form Variable Names

To form a predefined variable name, combine the base name of the variable with a field of its corresponding info class. You can use the base name with the corresponding info class only for example, you can use the base name action with info class `ActionInfo only`.

If an info class has no fields associated, use the base name as the variable. For example, to use the controller URL as a variable in a template, use `${controllerUrl}`.

If an info class comprises multiple fields, select a single field and associate it with the base name. For a list of fields associated with an info class, see Info Class Fields.

For example:

- To use the account name as a variable in a template, combine the base name `account` with the `name` field of the `EntityInfo` class to form the variable `${account.name}`.
- To use the trigger time of an action as a variable, combine the base name `action` with the `triggerTime` field of the `ActionInfo` class to form the variable `${action.triggerTime}`.

You can chain the segments of a variable name where the info class field type is yet another info class.

For example, to use the name of an application in which the latest triggering event occurred:

1. Combine the latestEvent base name with the `application` field of the `EventInfo` class.
2. The field type of the `application` field is `EntityInfo` which is yet another info class, hence select the name field of the `EntityInfo` class.
3. Form the variable name by appending the segments:

   ```
   { <base name>.<field name of info class1>.<field name of info class2>}

   ${latestEvent.application.name}.
   ```

Base Names

This tables describes the base names for the predefined variables with their corresponding info classes.

Base Name	Description	Info Class
`account`	Account in which the action was triggered.	`EntityInfo`
`policy`	Policy that triggered the action.	`PolicyInfo`
`action`	Triggered action.	`ActionInfo`
`topSeverity`	`INFO`, `WARN` or `ERROR`.	`NotificationSeverity`
`topSeverityImage`	Severity image or icon.	`ImageInfo`
`notificationConfigText`	Email actions only.	`String`
`controllerUrl`	URL of the controller.	`String`
`appDynamicsIcon`	-	`ImageInfo`
`appDynamicsLogo`	-	`ImageInfo`
`latestEvent`	Most recent triggering event.	`EventInfo`
`fullEventList`	List of events that triggered the action.	`List<EventInfo>`
`fullEventsByTypeMap`	List of events that triggered the action, grouped by type.	`Map<String, List<EventInfo>>`
`clampLimit`	Optional setting, limit on how many triggering events to display.	`int`
`clamped`	True if clamp limit is set.	`boolean`
`clampedEventList`	*n* most recent triggering events if clamped.	`List<EventInfo>`
`clampedEventsByTypeMap`	*n* most recent triggering events if clamped, grouped by type.	`Map<String, List<EventInfo>>`
`fullEventsNodeMap`	Node level variables to be used in email and HTTP templates.	`Map<EventInfo, NodeTemplateVariables>`

Info Class Fields

From the previous Base Names table, use the fields in the appropriate class to form the variable to use in a template.

```
class EventInfo {
  EventType eventType
  long id
  String guid
  String eventTypeKey
  Date eventTime
  String displayName
  String summaryMessage
  String eventMessage
  EntityInfo application
  EntityInfo tier
  EntityInfo node
  EntityInfo db
  List<EntityInfo> affectedEntities
  boolean healthRuleEvent
  EntityInfo anomaly // * Only defined in case of anomaly event
  EntityInfo healthRule // * Only defined when healthRuleEvent == true
  EntityInfo incident // * Only defined when healthRuleEvent == true
  boolean healthRuleViolationEvent
  NotificationSeverity severity
  ImageInfo severityImage
  boolean btPerformanceEvent // * true when eventType matches one of the BT performance event types
  String deepLink
}

class ImageInfo {
  String name
  String fileName
  String mimeContentRef
  String deepLink
}

class EntityInfo {
  EntityType entityType
  String entityTypeDisplayName
  long id
  String name
}


class ActionInfo extends EntityInfo {
 Date triggerTime
}


class PolicyInfo extends EntityInfo {
 boolean digest
 digestDurationInMins
}

class PolicyInfo {
  EntityType entityType
  String entityTypeDisplayName
  long id
  String name
  boolean digest
  int digestDurationInMins
}

class ActionInfo {
  EntityType entityType
  String entityTypeDisplayName
  long id
  String name
  Date triggerTime
}
```

```
enum NotificationSeverity { INFO, WARN, ERROR }

class NodeTemplateVariables {
    private long id;
    private String name;
    private long tierId;
    private String tierName;
    private long machineId;
    private String machineName;
    private boolean machineAgentPresent;
    private String machineAgentVersion;
    private boolean appAgentPresent;
    private String appAgentVersion;
    private String ipAddresses;
    private AgentType agentType;
}
```

The `private String ipAddresses` field contains hostnames instead of IP addresses.

Examples

HTTP request example for most recent triggering event:

```
http://myController:8080/controller/rest/applications/${latestEvent.application.name}/nodes/${latestEvent.node.name}
```

HTTP request example iterating through a list of triggering events:

```
#foreach(${event} in ${fullEventList})
  http://myController:8080/controller/rest/applications/${event.application.name}/nodes/${event.node.name}
#end
```

Email with dynamically varying subject example:

If you have several health rules to monitor your application and one policy to send email notification for all the health rules, you might find it difficult to search for the correct information in the notification. You can define a variable to dynamically update the email subject based on the event that triggered the notification.

Ensure that you define the subject variable in the email body and include it in the subject field.

```
Subject: $subject
Body:
...
 <!-- Subject changes dynamically based on the triggering event -->
#set($subject = #foreach(${events} in ${fullEventList}) ${events.eventType} #end )}
...
```

Email HTML body example:

AppDynamics uses this email template to notify the details of the events occurring on an application. You can customize the required event details you want notified by enabling the event types.

```
<!-- Default AppDynamics email template -->

#if( $!{notificationConfigText} != '' )
$!{notificationConfigText}
#else
AppDynamics
#end
Event Notification for the "${latestEvent.application.name}" Application
Event Notification Name: ${policy.name}
Event Notification Severity: ${topSeverity}

Summary of events occurring during the ${policy.digestDurationInMins}+ minute(s) prior to ${action.triggerTime}:
## Summary table
|| Event Type | Count ||
## Copy this row for each event type and count
#foreach($eventTypeEntry in $fullEventsByTypeMap.entrySet())
|| ${eventTypeEntry.getKey()} | ${eventTypeEntry.getValue().size()} ||
#end

## Full List of Events
The following events occurred during the time frame:
#if ($clamped)
Warning: The event list has been clamped at ${clampLimit} results! Please see the event and/or request snapshot
viewers for the full list of events.
#end
|| Event Time | Event Type | Severity | Tier | Node | Summary ||

#foreach($eventList in $clampedEventsByTypeMap.values())
  #foreach($event in $eventList)
    || ${event.eventTime} | ${event.displayName} | ${event.severity} | $!{event.db.name} | $!{event.tier.name}
| $!{event.node.name} | ${event.summaryMessage} ||
  #end
#end
Event Notification from AppDynamics.
This is an auto-generated email summarizing events on the "${latestEvent.application.name}" application. You
are receiving this because you are configured as a recipient on the "${policy.name}" event notification. This
is not necessarily an exhaustive list of all events during this time frame. Only those event types enabled in
the notification will appear in this message.
```

Example to extract event details using iteration:

These examples iterate through the `eventList` to extract the details of each event occurring during the specified time.

```
<h1>Summary of events occurring during the ${policy.digestDurationInMins}+ minute(s) prior to ${action.
triggerTime}:</h1>

<table>

#foreach(${eventList} in ${fullEventsByTypeMap.values()})
  #foreach(${event} in ${eventList})

    <tr>
        <td>
            <!-- Event icon -->
            <img src="${event.severityImage.mimeContentRef}" alt="${event.severity}" />
        </td>
        <td>
            <!-- Event name with event link -->
            <a href="${event.deepLink}">${event.displayName}</a>
        </td>
        <td>
            <!-- Event message -->
            ${event.eventMessage}
        </td>
    </tr>

  #end
#end

</table>
```

Example code to list all variables:

```
<h1>variables below </h1>

${fullEventsNodeMap}

<h2>map elements below</h2>

#foreach(${value} in ${fullEventsNodeMap.values()})
${value}
${value.name}
#end
```

If these predefined variables do not meet your needs, you can define your own custom variables for use in these templates in the **Create Template** pane.

Do not use any of the predefined variable names for your custom variables.

Email Digests

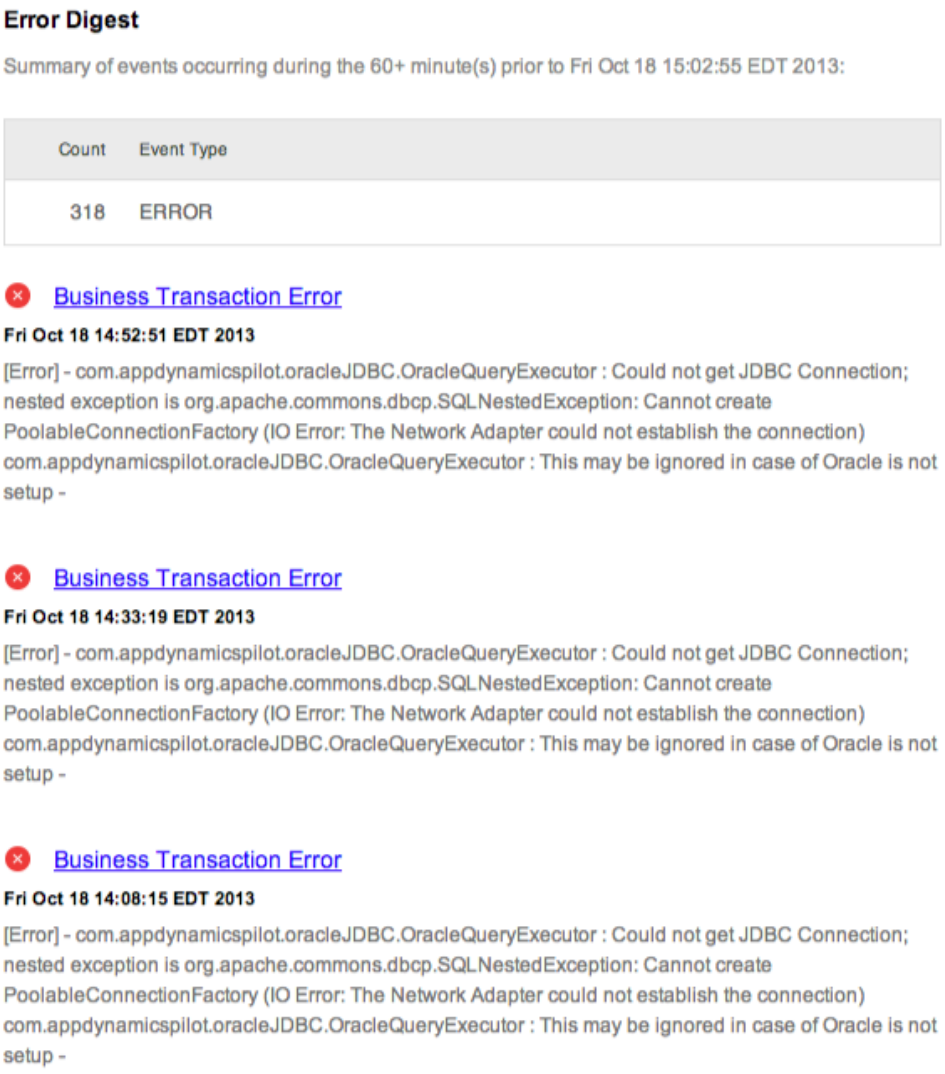
This page provides an overview of email digests in AppDynamics. In addition to specific actions that are triggered by specific events, you can create an email digest that reports a summary of specific events to a recipient list on a schedule.

Permissions

To create, edit, or delete email digests, a user must have the Configure Actions, Configure Health Rules, and Configure Policies permissions.

Sample Email Digest

This is a sample of an email digest that is sent every hour:



Error Digest

Summary of events occurring during the 60+ minute(s) prior to Fri Oct 18 15:02:55 EDT 2013:

Count	Event Type
318	ERROR

Business Transaction Error

Fri Oct 18 14:52:51 EDT 2013

[Error] - com.appdynamicspilot.oracleJDBC.OracleQueryExecutor : Could not get JDBC Connection; nested exception is org.apache.commons.dbcp.SQLNestedException: Cannot create PoolableConnectionFactory (IO Error: The Network Adapter could not establish the connection) com.appdynamicspilot.oracleJDBC.OracleQueryExecutor : This may be ignored in case of Oracle is not setup -

Business Transaction Error

Fri Oct 18 14:33:19 EDT 2013

[Error] - com.appdynamicspilot.oracleJDBC.OracleQueryExecutor : Could not get JDBC Connection; nested exception is org.apache.commons.dbcp.SQLNestedException: Cannot create PoolableConnectionFactory (IO Error: The Network Adapter could not establish the connection) com.appdynamicspilot.oracleJDBC.OracleQueryExecutor : This may be ignored in case of Oracle is not setup -

Business Transaction Error

Fri Oct 18 14:08:15 EDT 2013

[Error] - com.appdynamicspilot.oracleJDBC.OracleQueryExecutor : Could not get JDBC Connection; nested exception is org.apache.commons.dbcp.SQLNestedException: Cannot create PoolableConnectionFactory (IO Error: The Network Adapter could not establish the connection) com.appdynamicspilot.oracleJDBC.OracleQueryExecutor : This may be ignored in case of Oracle is not setup -

Create an Email Digest

To create an email digest, navigate to **Alert & Respond > Email Digests** on either the right panel or in the left navigation pane. Follow the steps in the wizard to create the digest.

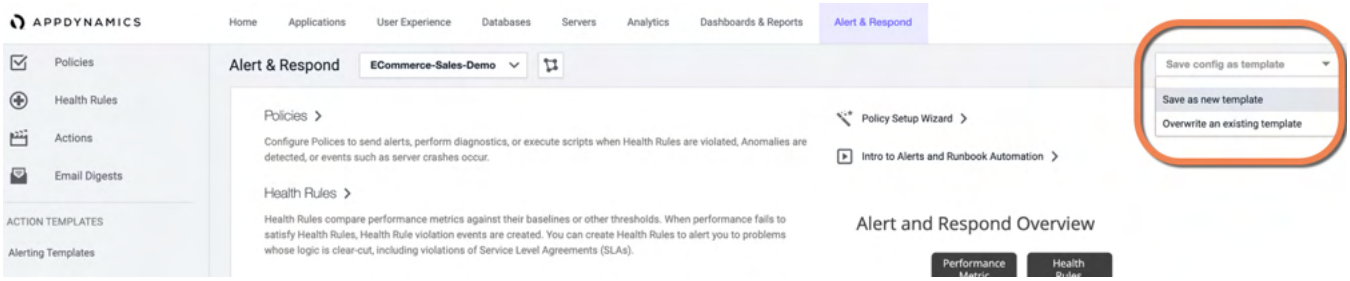
Alerting Templates

An alerting template is a configuration of features such as Health Rules, Policies, Actions, Action Suppression, Schedules, and Email Digests. You can save the **Alert & Respond** configuration of an existing application as a template. Templates help maintain consistent configuration across various applications regardless of their location, version, or environment.

How to Use an Alerting Template

You can save the **Alert and Respond** configuration of an existing application as a template. You can use the **Save as new template** option. You can apply the template to other applications to replicate the configuration.

After configuring an alerting template, if you require to change the configuration and apply the changes to all the applications, use the **Overwrite an existing template** option. The existing template is overwritten. You can then re-apply the template to the desired applications. For more information, see Configure and Manage Alerting Templates.



Alerting Template Operations

To view alerting templates, select **Alert & Respond > Alerting Templates**. The alerting templates list displays all templates. You can create and manage the templates using the **Alerting Template** interface.

This table lists the template operations:

Operation	Description
	View the configuration details of the selected template(s). For more information, see View Alerting Template Details.
	Create a new template. For more information, see Create an Alerting Template.
	Overwrite a selected template. Any changes you make, overwrite the existing configuration when you save the template. For more information, see Overwrite an Alerting Template. When you edit a template, the updates are not applied to applications using the template. You must re-apply the template to the desired applications.
	Delete an existing template. For more information, see Delete an Alerting Template. The configuration of applications using the template remains intact even after you delete the template.
	Import a saved template configuration. Use this option when you have saved the configured alerting template and want to apply it to other applications.
	Export a template configuration for future use. Use this option when you have configured an alerting template and want to save it for future use.
	Apply a template to the selected application(s). For more information, see Apply Template to Application(s).
	Search for a template.

Configure and Manage Alerting Templates

You can create a new alerting template in **Alert & Respond > Alerting Templates**. Alternatively, you can configure the **Alert and Respond** features in the Controller UI and save the configuration as an alerting template.

> JMX Health Rules are not supported.

View Alerting Template Details

To view the configuration details of an existing alerting template:

1. Select a template.
2. Click the view icon to view the configuration details.
3. To edit the template, view template history, or apply the template to other application(s), click **More**.

Create an Alerting Template

Ensure that you create and configure the health rules, policies, actions, action suppression, and email digests you plan to use to create the alerting template before you create the template.

1. Click the create icon. The **New Template** wizard displays.
2. Enter a name for the template. You can use alphanumeric characters to specify a name. Ensure that you specify a unique name.
3. If required, include a description. Click **Next**.
4. Select an **Application** from the list displayed in **Choose an Application**. The configuration you define for this application is saved as a template. Click **Next**.
5. Select the following feature configuration as required:
    a. Policies
    b. Health Rules
    c. Actions
    d. Action Suppression
    e. Email Digests
6. Click **Next**.
7. Review the configuration settings you have defined. Click **Save**.

Overwrite an Alerting Template

If you make any inadvertent updates to a template, the changes cannot be reverted. You can update the template using the overwrite option, which overwrites the existing configuration with the updates you made when you save the template.

1. Select the template and click the overwrite icon. The **Overwrite Existing Template** wizard displays.
2. Select the application whose configuration you want to save as a template. Click **Next**.
3. Modify the pre-configured features as necessary. Click **Next**.
4. Review the configuration settings you have defined. Click **Overwrite**.

Once you overwrite the configuration of a template and save it, you cannot revert the changes.

Any updates made to the template are not applied to all the applications using the template. You must re-apply the modified template to applications.

Delete an Alerting Template

You can delete a template if you no longer use it. When you delete a template, the applications using the template are not affected. The application configuration also remains intact. You can continue to use the applications as before.

1. Select the template and click the delete icon.
2. Click **Delete**.

> Once you delete a template, you cannot restore it.

Import an Alerting Template

You can import a saved alerting template and apply it to other applications.

1. Click the import icon. The **Import Template** dialog displays.
2. Click and select a file to import.

> Ensure that:
>
> - You select a JSON file to import.
> - The file you want to import has a unique name, otherwise the file is not imported.

3. Click **Import**. The selected file is imported and displays in the **Alerting Templates** UI.

Export an Alerting Template

You can export a saved alerting template and apply it to applications in other accounts.

1. Select a template to export.
2. Click the export icon. The **Template exported successfully** message displays.

Apply Template to Application(s)

After you create an alerting template, you can apply the template to multiple applications.

1. In the **Alert & Respond > Alerting Templates**, select the template and click 

. The **Apply Template** wizard displays.
2. In **Choose Application(s)**, select the applications to apply the template. Click **Next**. Use the **Search** option if the list of templates is too long.
3. To add the template configuration (such as policies, health rules, and actions) to the existing configuration of the selected applications:
    a. In **Choose How to Apply Template**, select **Add/Merge**. Click **Apply**. The **Add/Merge** operation status displays.
    b. The alerting configuration of the template is added to the selected application(s).
    c. If you see an error or warning for an application, refer to **Check the Status of the Applications** to fix it.

    > If the entities (such as Health Rule, Policy, Action, Action Suppression, or Email Digest) exist in the template and the application, the configuration of the template entity replaces the configuration of the application entity.

4. To replace the existing configuration of an application with the configuration of the selected template:
    a. Select **Replace**.
    b. Click **Apply**.
    c. A confirmation dialog displays. Click **Yes** to confirm. The **Replace** operation status displays.

    > This operation replaces the alerting configuration of all the entities of the selected application(s), such as the health rules, policies, and actions, with that of the alerting configuration of the template. The existing alerting configuration of the selected application(s) is deleted permanently. Perform this operation only if you want to replace all the entities you have configured.
    >
    > If you see an error or warning for an application, refer to **Check the Status of the Applications** to fix it.

Check the Status of the Applications

You can check if the template was successfully applied to all the applications.

1. Click the **Status** of the template. The **Template History** page listing the status of each application displays.
2. If an error is listed on the **Template HIstory** page, click the error to determine the cause of the error and the exact location.
3. Fix the error and then re-apply the template to the affected application.

Alerting Template Questions and Answers

This page lists some questions and answers that help you use the alerting template.

As an on-premises customer, will I be able to use the alerting template?

Yes, this feature is available for on-premises users.

Will this feature require an agent upgrade?

This feature does not require any agent upgrade.

Do I need role-based access privileges to use alerting templates?

No, any user can create and manage alerting templates.

When I update a template, will the changes be applied to all applications using the template?

No, when you update a template, you need to re-apply the template to the application(s) to ensure that the updates are reflected.

Where can I view all the templates I created?

You can view the template library in the **Alert & Respond > Alerting Templates** UI. For more details, see View Alerting Template Details.

What are two ways to create a template?

You can save the Alert and Respond configuration of an existing application as a template using the **Save as new template** option.

After configuring an alerting template, if you require to change the configuration, use the **Overwrite an existing template** option.

Can I apply a template without selecting an application?

No, you can apply a template only to an application. Hence, you must select an application.

Where can I track the status of applying a template?

You can check if the template was successfully applied to all the applications. From the **Alert & Respond > Alerting Templates** UI, click the **Status** of the template. The **TEMPLATE HISTORY** page listing the status of each application displays.

Will updates to a template be applied to an application that is being edited by someone?

No, the template history page displays an error in the status column. Complete your edits to the application, then apply the updated template.

When I update a template, will I be able to revert the changes?

No, once you configure a template and save it, you cannot revert the changes. If you make any inadvertent updates to a template, you can overwrite the configuration and save the template again. For more information, see Overwrite an Alerting Template.

Can I edit a template?

You can overwrite the configuration of an existing template. If you make any inadvertent updates to a template, you can overwrite the configuration and save it. For more information, see Overwrite an Alerting Template.

When I delete a template, what happens to the applications using the template?

When you delete a template, the applications using the template are not affected. The application configuration also remains intact. You can continue to use the applications as before.

When I update a feature (for example a Health Rule) and re-apply the template, will the other associated features (like Policies, Actions) get selected by default?

Yes, when you update a feature within a template and re-apply it, all other associated features are selected by default.

Can I restore a deleted template?

Once you delete a template, it is lost, you cannot restore it.

Can I make an update to a single application without affecting the other applications using the template?

1. In the **Alert and Respond** UI, select the application you want to update.

2. Select the feature you want to modify (for example, policy, health rule).
3. Update the required configuration and save.

I made an inadvertent change to a template, how can I correct it?

You can delete the template, create a new one, and apply the template to the required applications.

> Once you delete a template, you cannot restore it.

Can I overwrite a template?

You can overwrite the configuration of a template and save it. You must re-apply the template to all applications. You can either replace an existing application configuration or append to it.

How can I apply a template to an existing application?

You can apply a template to an existing application with two methods:

- **Add/Merge**—The alerting configuration of the template is added to the selected application(s). If the entities (such as Health Rule, Policy, Action, Action Suppression, or Email Digest) exist in both the template and the application, the configuration of the template entity replaces the configuration of the application entity.
- **Replace**—This operation replaces the alerting configuration of all the entities of the selected application(s), such as the health rules, policies, and actions, with that of the alerting configuration of the template. The existing alerting configuration of the selected application(s) is deleted permanently. Only perform this operation if you want to replace all the entities you have configured.

For more information, see Configure and Manage Alerting Templates.

Can I apply a template to multiple applications at a time?

Yes, you can. For more information, see Apply Template to Application(s).

Troubleshoot Alert and Respond Problems

Health Rules

Why do I see consistent false alerts; logs display an error 'Error sending metrics'?

This happens when you configure a highly sensitive condition for a health rule.

When you define a very small evaluation time frame (for example, 3 mins) for a health rule condition, few configured metrics may fail to report data during the short time frame due to reasons such as network errors. This may not be the norm, however, alerts are triggered even if a single data point is not reported leading to consistent false alerts.

AppDynamics recommends that you configure a higher time frame to evaluate a health rule condition.

Dashboards and Reports

This page provides an overview of dashboards and reports in AppDynamics.

Dashboards and Reports are useful for getting a high-level understanding of the health and performance of your system on a regular basis.

Dashboards provide a graphical overview of the selected data made available for quick access. Custom dashboards let you create and arrange widgets to give users a visual overview of the data of interest to them. A War Room is similar to a dashboard, but it is interactive. The chat facility in virtual war rooms enable visually-enhanced collaborative troubleshooting in real-time in response to high priority events.

Reports capture and share AppDynamics data from dashboards to recipients. Reports can be automatically generated on a regular schedule.

Custom Dashboards

Deployment Support

Related Pages:

- Create and Manage Custom Dashboards and Templates
- Virtual War Rooms
- Custom Dashboard Permissions
- Dashboard Recovery

This page provides an overview of custom dashboards in AppDynamics.

A custom dashboard enables you to display a specific set of metrics and data points on one screen. You can display application, server, and database metrics reported by AppDynamics agents. You can also supplement the AppDynamics built-in dashboards (such as the application dashboard, tier dashboard, and so on) and display metrics that are of interest to a particular audience.

You can use custom dashboards to present selected metrics for a user who only needs a relatively narrow or focused view of the data. For example, an executive may only need a high-level view of system performance and activity. You can assign such users to the predefined Custom Dashboard Viewer role. Any user with a custom dashboard viewer role can view custom dashboards.

> Regardless of application-level or other restrictions configured for that data, a custom dashboard viewer can view all data that the dashboard owner is permitted to view on the controller-level custom dashboards.

Instead of using the Custom Dashboard Viewer role, you can share a custom dashboard.

You can use Controller-level custom dashboards to:

- Provide a customized view of application, server, and database performance data.
- Aggregate data from different applications on the same Controller
- Compare data from different applications on the same Controller
- Show a single view of both live and historical data
- Share data with other users and stakeholders

Create Custom Dashboards

To create a Controller-level custom dashboard, navigate to **Dashboards & Reports > Dashboards > Create Dashboard**. In the **Dashboards** panel, click an existing dashboard to edit it.

> To create or edit a custom dashboard, a user needs to have the Can Create Custom Dashboards permission. See Visibility and Permissions.

Custom Dashboard Templates

Custom dashboard templates are an efficient way to display the same type of information for multiple tiers or nodes. You create a dashboard in the scope of a tier or node and then use that dashboard as a template to display the same type of data for other tiers or nodes in the same application. Associating the template with a specific tier or node sets the source of the dashboard's data.

> To create or edit a custom dashboard template, a user needs to have the Configure 'My Dashboards' for Tiers and Nodes permission.

Create a custom dashboard template to:

- Compare overall application or infrastructure performance against individual tier or node performance
- Show a standard set of metrics for multiple tiers and nodes

You create and access dashboard templates from the My Dashboards tab of the built-in tier or node dashboard. From there, you can create a new dashboard template for that tier/node or use dashboard templates that were created for other tiers/nodes in the same application by association.

Although the process for configuring Controller-level and tier or node-level dashboard templates is similar, the two sets of dashboards are separate. In other words, you cannot associate a Controller-level dashboard with a tier/node through the My Dashboards tab, nor can you access from the Controller-level dashboards list any tier and node level dashboards outside the application in which they were created.

The dashboard and template names must be unique Controller-wide. You cannot create a custom dashboard using an existing dashboard template name, nor can you create a dashboard template using an existing custom dashboard name.

You can export custom dashboards and dashboard templates to a JSON file and then import them where you need them.

View Strategies

For AppDynamics Users

If you have set up default view permissions to custom dashboards, any user who is logged in to the Controller can view a custom dashboard.

Users can also be granted view, edit, share, and delete permissions on specific dashboards.

The Dashboards Viewer predefined role has permissions to view custom dashboards. You can grant this role to users or groups who need access to custom dashboards. See Custom Dashboard Permissions for details.

For the Public

Sharing a Controller-level custom dashboard makes the dashboard available to viewers who do not have AppDynamics accounts.

If the dashboard is shared, anyone who has been sent a direct link to the dashboard URL can view it. If someone has the URL and the dashboard is not shared or no longer shared, they see a message that the resource is not available when they try to access that URL.

To control Controller-level custom dashboard sharing, click **Share** on the menu of the custom dashboard configuration.

There is no comparable sharing mechanism for external viewers without Controller credentials to access tier- and node-level custom dashboards based on templates. To access these types of dashboards, the user must be logged in to the Controller and have custom dashboard view permissions on the specific dashboard.

The availability of sharing mechanism is now controlled by the Share permission of the user.

Simplify the Data Displayed

If a chart in a custom dashboard is displaying more data than you want, click the legend for the data that you want to remove from the chart. You can click the greyed-out legend to restore the display.

Widgets

Related Pages:

- Create and Manage Custom Dashboards and Templates
- Use Wildcards in Metric Definitions
- Configuration Import and Export API
- Visualize Analytics Data

This page provides an overview of widgets in AppDynamics. You can use widgets to create visual representations of your data in custom dashboards and war rooms.

If your platform is licensed for Analytics, you can display the results of Analytics searches on custom dashboards as well. See Visualize Analytics Data for more information.

Widget Types

Most widgets types in custom dashboards and war rooms will be familiar to you if you have worked with numeric charts before. This section provides information for certain widget features that may require additional explanation.

Time Series Graph

The standard time-series graph displays one or more values over time in various formats: line, area, column, scatter diagram.

If you configure a widget for a war room that examines metrics on a Java node, consider the streaming graph widget described in Virtual War Rooms. The streaming widget updates more frequently (every five seconds) than the standard time series widget (every 60 seconds) for collaborative real-time troubleshooting.

Events are grouped by time of resolution at the bottom of the graph. You can choose what events to display when you enable **Events Overlay** and configure the event queries for the graph.

Pie Chart

Pie charts or donut charts display multiple values as proportional slices of a whole. The data series should meaningfully total 100% of some series. For example, CPU Busy + CPU Idle + CPU Stolen on a single node or tier.

Gauge Widget

The gauge widget displays the magnitude of a metric value with respect to minimum and maximum values, both by the extent to which the gauge is filled and by the color of the fill.

You can set the minimum and maximum values displayed by the widget. Set the maximum to the highest acceptable maximum or a value close to it so that the gauge is configured for metrics within acceptable values for your site. If you do not set them, default values are used.

This table explains the color coding for gauge widgets.

Percentage	Color	Status
0-10%	Green	OK
10-50%	Gradient from green to yellow	OK trending to WARNING
50-90%	Gradient from yellow to red	WARNING trending to CRITICAL
90-100%	Red	CRITICAL

This gauge example displays the calls per minute metric without any special configuration for minimum and maximum values. The current value is 74 cpm, which is midway between the minimum (0) and maximum (148) for the time period in the WARNING range.

If you set the **Max Value** to 1000, the gauge color changes to green because 74 cpm falls within the OK range.

Streaming Graph Widget

A streaming graph widget is similar visually to a time-series graph, but it is updated every five seconds in contrast to every 60 seconds in the time-series graph.

For Java nodes only, the war room supports the streaming graph widget. It displays data from only a single specified node and only a Java node. This feature is not supported for any of the other agents. To display streaming data from multiple nodes, you need to create multiple widgets.

Log Tail Widget

The war room supports a log tail widget used for Java nodes only to share a log file or parts of one.

The log tail widget displays the contents of the file specified by its Path property. The Path is the absolute path to the log file on the machine on which the selected node is running. If you provide a regular expression in the widget properties, the widget displays only those lines in the log file that match the regular expression. If no regular expression is provided, the widget displays the entire log file. Note that applying a regular expression filter will only adjust lines that come after you have specified the filter. Previous lines will remain filtered based on the previous rule. The widget uses the Java conventions for regular expressions.

If the log file rolls over during the request, the response to the log tail request is undefined.

Issue Tracking Widget

The issue tracking widget displays information about an issue or a list of issues that you have selected. You can choose any detail, people, and date fields that you would like to display. Any updates made to the tracked issue is reflected on the widget.

This widget is only available once you have integrated AppDynamics with Atlassian JIRA.

Metric Value

You can specify a title for the widget as well as a format string that adds the name of the selected metric, the time range, or the name of the function. Alternatively, you can use a separate label widget to explain the metric value.

You have the option of formatting metric values by abbreviating them and removing trailing zeros. Abbreviations only apply to numbers over one million. Metric values are always integers on dashboards with single metrics, therefore decimal formatting does not apply.

Metric Expression

The ability to set the number of decimal places allows you to display decimal values less than one, as well as decimals for larger numbers. If your decimal limit is set to zero, then the remove trailing zeros feature is turned off by default.

Health Status

This is a health bar, filtered table, or status light circle of health statuses for entities, such as applications, business transactions, tiers, nodes, databases, servers, and health rules. After choosing an entity type to show the health for, you have the option to narrow the scope or simply display all.

It is important to note that when selecting tiers as your entity, the color of the health status depends on the statuses of all the health rules that are configured for the tiers. For example, if there are four health rules for the tiers, then the color of the widget will reflect the cumulation of all of their statuses. The widget displays green for normal, yellow for warning, and red for critical status. See Configure Health Rule Conditions to learn how the metrics that define warning and critical health statuses are set.

When using the circle visual, you have the option of manipulating the display by choosing its aggregation type: either **Ratio Graph** or **Most Severe Status**.

Event List

This is a filtered table of events that occurred within the specified time range.

IFrame

You may want to include a shared custom dashboard in another custom dashboard when you have different operators who focus on particular dashboards and you want to roll them all up into a primary dashboard for management.

To display information from another dashboard or from another website inside this custom dashboard, you need the URL. You can get the URL for a pane by clicking **Settings > Copy a link to this screen to the clipboard.**

The IFrame widget is sandboxed. Advanced HTML features, such as forms and Javascript, will not work with the IFrame widget.

Widget Data

Some widgets, such as a time series graph, can display a single value or multiple values. Some widgets, such as a donut, must display multiple values to make sense. Others, such as a gauge or a metric value, display only a single value.

Specify a Function of a Metric Value for a Single Entity

When you specify the data to display in a widget, you use a pulldown menu in the **Add Series** panel to select which function of the value to display:

- **Minimum**: Minimum value, only available for averaged metrics
- **Maximum**: Maximum value, only available for averaged metrics
- **Value**: Contains the average or the sum across the time range depending on the metric
- **Sum**: Aggregated value of the metric over the time range
- **Count**: Count of the observed values over the time range
- **Current**: Sum of the most recent minute's metric data values across all the included nodes

You can also use a metric expression to specify a metric. The behavior is similar to using an expression for a health rule. For more details about metric expressions, see how to build an expression for Health Rules.

> ⚠ Dashboard widget metric expressions currently cannot properly handle fraction or float constants that are less than one. These metrics are treated as zero. They work properly when they are a fraction, or float values greater than one.

Specify Metrics in Multiple Entities Using a Wildcard

You can use a wildcard to specify a metric that evaluates across several similar entities for metrics in these metric hierarchy branches:

- Hardware Resource
- JVM
- CLR
- Custom Metrics

See Use Wildcards in Metric Definitions for instructions.

Empty Data Series

It is possible that no data is available for a particular metric within the specified time range while for other time ranges valid data is reported. When this occurs, the label for the empty data series appears in the legend but no data is displayed. For example, no line in a time-series graph or slice in a pie chart.

Enable Limits on the Number of JMX Instances or Nodes for a Single Entity

You can use account-level configurations to enable dashboard or widget limits on large numbers of JMX instances or nodes instrumented in one application while returning widget data. This also includes threshold checks for business transactions, errors, servers, and backend data. This applies to the JMX type metric in the widget **Type of Nodes** menu. The **All Nodes in the Application** option becomes disabled if the number of nodes exceeds the threshold. An error is returned if the number of any threshold is exceeded on the backend. This should help prevent outages.

Widget Properties

Widget properties vary to some extent based on the type of the widget.

Most of the widget properties in the configuration forms are self-explanatory. This list describes some of the less obvious properties:

- **Metric Display Name**: You can choose the display name for the metrics in a widget. Specify one or more of the following substitution variables to determine the metric display name:
  - ${#}: Number the displayed metrics based on sorting with the rollup value
  - ${m}: Metric name
  - ${e}: Entity name

  For example, if specify ${m}, the widget displays the full metric name.

- **Static Thresholds**: You can create static threshold with a specific value for graph widgets. The threshold appears as a line on the graph.
- **Double click action**: Some widgets have an option to define the result of double-clicking the widget. You can configure the double click action to open the **Metric Browser** or a URL specified by the Drilldown URL value.
- **Drilldown URL**: The Drilldown URL specifies the URL of an AppDynamics panel you want the widget to open when you double-click the widget. For example, if you see an increase in slow requests and want to investigate the cause, you may want to jump to the **App Servers List** of all tiers from your dashboard by double-clicking a widget. To get the URL for a specific panel, click **Settings > Copy a link to this screen to the clipboard**.
- **Time Range**: Widgets can use the global or dashboard time range or a time range that is specific to the widget. If you choose a widget-specific time range, select the widget time range from the pulldown menu.
- **Events Overlay**: You can configure specific events to overlay on the graph widget to show how specific events may correlate with performance. Check the **Show Events** checkbox and create the query to define which events should be overlaid.

Configuring Widgets

Set up or modify widgets with the widget palette in Edit mode.

If you configure widgets for a war room, you will see an additional category for Streaming Data widgets. To add widgets to the war room space the presenter must be in edit mode. To enter edit mode, click **Edit**. If it is not visible and you are the presenter, you are already in edit mode. You can duplicate one or more widgets at a time and delete any widget as required.

To configure a new widget:

1. Click **Edit**.
2. Click **Add Widget**. The widget palette displays.
3. Click the widget in the palette. The configuration form displays on the left side of the configuration panel.
4. Configure both the widget data and the widget properties. After you configure the data, a preview of the widget displays. The preview updates as you set more properties.
5. Click **Save**.

To modify an existing widget:

1. Click **Edit**.
2. Click the widget in the dashboard.
3. Click the settings icon in the upper right corner of the widget.
4. Use the form to modify the widget.
5. Click **Save**.

Use Wildcards in Metric Definitions

Related Pages:

- Configure Health Rules
- Create and Manage Custom Dashboards and Templates
- Metric Browser

This page provides an overview of wildcards in AppDynamics. In a Custom Dashboard widget or Health Rule, you can use a wildcard to specify a metric that evaluates across several entities, such as multiple hardware entities or memory pools.

The wildcard feature is supported for metrics in the Hardware Resources, JVM, and CLR branches of the metric hierarchy, and for custom metrics created with a monitoring extension as described in Extensions and Custom Metrics.

> The wildcard feature is not supported for custom metrics.

To use wildcards:

1. Define the relative metric path to the metric that you want to display. This involves getting the full metric path in the Metric Browser and then trimming it.
2. Modify the relative metric path to apply to multiple entities.
3. Paste the modified relative path in the metric selector for the widget or health rule that you are configuring.

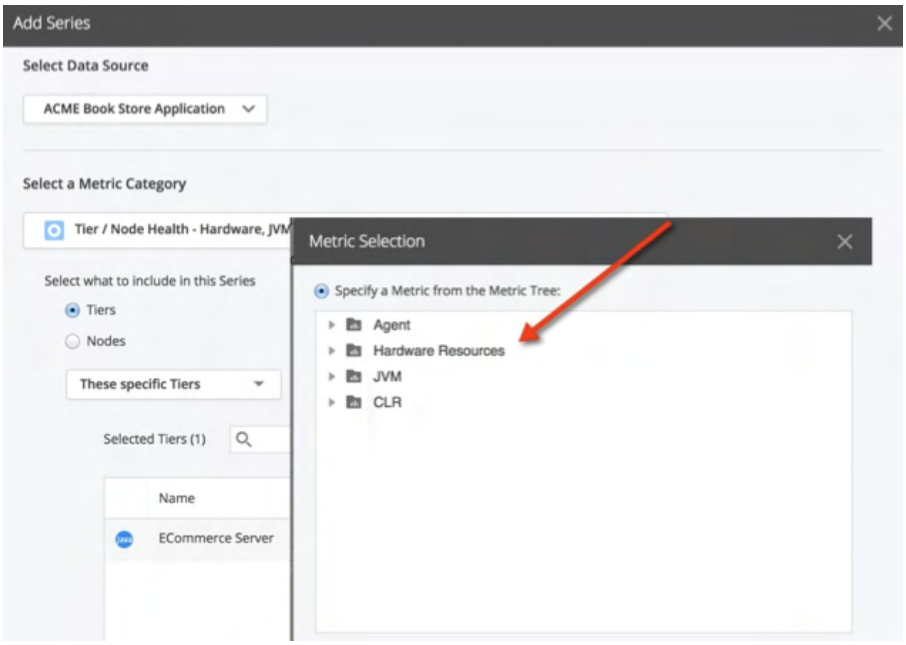
Define the Relative Metric Path

A metric path is a pipe-delineated path to a specific metric. In the Controller UI, hover over a metric in the Metric Browser to get the full metric path. Right-click the metric to view the **Copy Full Path** option.

To define the relative metric path, truncate the leftmost part of the full metric path.

To know how much to truncate for a particular use case, look at the embedded metric browser in the **Metric Selection** panel of the widget or health rule configuration that you are setting up. By the time you have reached this selector, you will have configured the application, tiers, or nodes of the metric that you want to select.

Use the category in the **Metric Selection** panel as the first segment of the relative metric path, truncating everything from the full metric path that comes before that segment. In the example, the first segment would be Agent, Hardware Resources, JVM, or CLR.



As per the preceding example, the full metric path copied from the Metric Browser is Application Infrastructure Performance|ECommerce Server|Hardware Resources|Disks|dev-dm-1|% CPU Time.

Truncate everything to the left of the category selected in the Metric Selection panel that is, Hardware Resources in this example, as the display in the embedded Metric Browser starts with Hardware Resources.

The relative metric path is: Hardware Resources|Disks|dev-dm-1|% CPU Time.

Consider another example where the full metric path copied from the Metric Browser is: Application Infrastructure Performance|ECommerce Server|Individual Nodes|ECommerceAppNode|Hardware Resources|Disks|dev-dm-1|% CPU Time.

The relative metric path after truncating everything to the left of the Hardware Resources is: Hardware Resources|Disks|dev-dm-1|% CPU Time.

Modify the Relative Metric Path

Replace a single segment in the relative metric path with an asterisk to indicate that the metric should be evaluated for all the entities represented by that segment.

> ⚠ Multiple asterisks in a single metric path are not supported.

For example:

The relative metric path is: Hardware Resources|Disks|dev-dm-1|% CPU Time, and you want to display or create a health rule condition on the % CPU time for all of the disks in that tier or node.

Substitute the asterisk for the disk name as: Hardware Resources|Disks|*|% CPU Time.

Configure the Metric for Multiple Entities

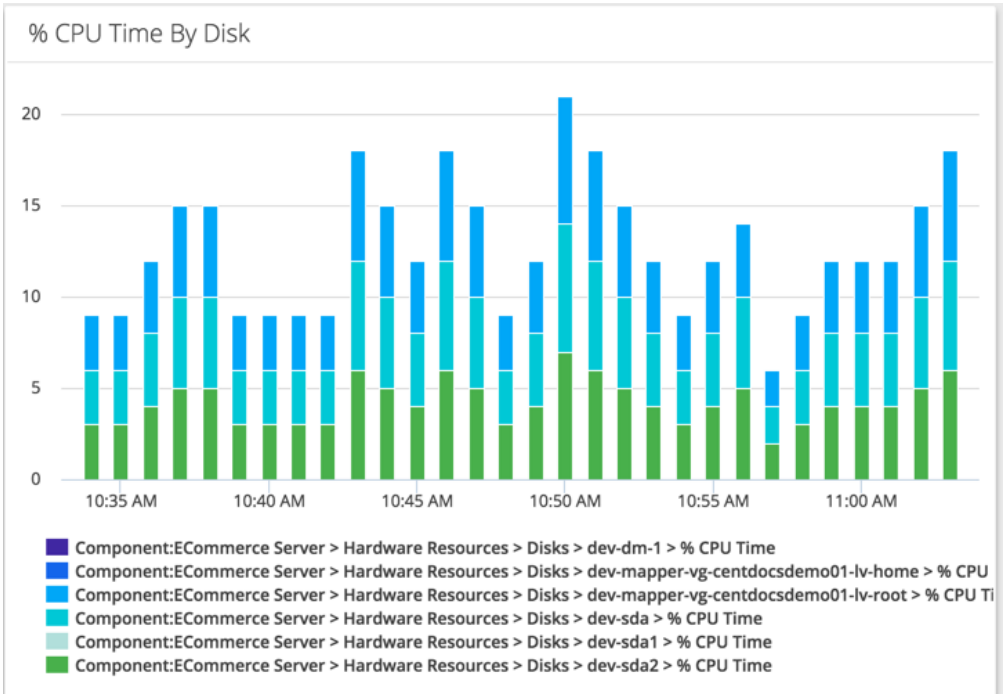
In the **Metric Selection** panel for the dashboard widget or health rule, you can configure metric for multiple entries.

To configure the metric for multiple entities:

1. Select **Specify Relative Metric Path** on the bottom of the **Metric Selection** panel.
2. Paste the modified, wild-carded relative metric path and click **Select Metric**.

Verify the Metric Specification

When you configure a relative metric specification for multiple similar entities in a custom dashboard, multiple metrics display in the widget.



Wildcards Replace Entire Path Segments

An asterisk replaces the entire segment in the path. You cannot use it to replace only a portion of a string because they are used in regular expressions.

For example, the following usage in an attempt to get the ART for all the business transactions beginning with **View** is not valid and not supported:

Business Transaction Performance|Business Transactions|ECommerce Server|View*|Average Response Time (ms).

Specify Average Response Time (ms) to get ART for all the business transactions on the ECommerce Server.

Colons in the Metric Path

The colon, if any, in the metric path is translated into a pipe. Only one segment is allowed between pipes. The colons and pipes can be used interchangeably.

For example:

The colon in the metric path, Application Infrastructure Performance|ECommerce Server|JVM|Memory:Heap|Committed (MB) is translated to Application Infrastructure Performance|ECommerce Server|JVM|Memory|Heap|Committed (MB).

For the relative metric path, to get metrics for both heap committed and non-heap committed, specify JVM|Memory|*|Committed (MB). Do not specify JVM|*|Committed (MB).

Create and Manage Custom Dashboards and Templates

Related Pages:

- Custom Dashboards
- Widgets
- Import and Export Custom Dashboards and Templates Using the UI

This page describes how to create and manage dashboards.

Permissions to create, view, edit, and delete custom dashboards are set through roles. See Custom Dashboard Permissions and Create and Manage Custom Roles.

Access and Create Custom Dashboards

For Controller-level dashboards:

1. Click **Dashboards & Reports** to view existing custom dashboards or to create a new one.
2. If existing dashboard names display, click **Dashboards**.
3. Click **+ Create Dashboard**.
4. Add widgets to the dashboard.
5. Click **OK**.

For tier/node-level dashboard templates:

1. In an application, in the built-in tier or node dashboard in which you want to create or use a custom dashboard template, click **My Dashboards**.
2. If there are no associated custom dashboards for the entity, click **+ Create a new Custom Dashboard**.
3. If you want to associate an existing dashboard with the entity, click **Associate Dashboards** from the dropdown of the dashboard with which you want to associate the entity. See Associate a Custom Dashboard Template with a Tier or Node.

A dashboard has no content until you add widgets to it. See Widgets.

General Dashboard Characteristics

- **Name**—Dashboard names must be unique per Controller.
- Two layout types:
  - **Grid—**This is the default type that gives you a flexible layout that is easy to rearrange on the canvas. The grid layout also scales in size when viewed on mobile devices. You cannot overlap widgets using this layout type.
  - **Absolute**—Use this type to control width and height and the exact placement of widgets on the canvas. With the absolute layout, you can overlap widgets and use **Send to Front** or **Send to Back** actions to control what displays.
- **Configurable auto-refresh Interval**—Sets the data refresh rate in seconds.
- **Dashboard Time Range**—This setting applies only to Controller-level dashboards. From the **Time Range** dropdown, you can set a global time range for the dashboard. When you add widgets, you can specify whether the widgets use this global time range or a widget-specific time range. If you do not specify a widget-specific time range, the global time range is used for all the widgets.

Access the Dashboard Operations

While you are in edit mode, your changes are automatically saved. To save changes manually, disable **Auto-Save** and click **Save**. There is no way to save your changes manually.

If you are accessing a dashboard on a mobile phone, click the **Mobile** icon for a better viewing experience. To access various operations concerning the dashboard, click **Actions**.

Widget Operations

You must be in edit mode to add and modify widgets. See Widgets.

Widget Operation	Procedure
Access the widget palettes	In the dashboard configuration panel, click **+ Add Widget**.
Add a widget to the dashboard	1. In the left panel of the **Add Widget** panel, select the context. 2. Click the widget in the palette on the right.
View and edit the widget properties panel	Click the **Gear** icon in the upper right corner of the widget.

Delete one or more widgets	• Select the widgets and click **Delete**, or; • Right-click the widget and select **Delete**.
Duplicate one or more widgets	• Select the widgets and click **Duplicate**, or; • Right-click the widget and select **Duplicate**.
Copy/paste a widget	• Select the widget and click **Copy**, then click **Paste**, or; • Select the widget, right-click, and select **Copy**. Then, right-click the widget again, and select **Paste**.
Edit the properties of multiple widgets	1. Hold down your command key and select the widgets you want to edit the properties for. 2. Click **Edit**. 3. From the **Edit <number of widgets> Widgets** dialog, select the properties you want to apply to all the widgets, and click **OK**.

Note that not all widgets share the same properties, and Analytics widgets cannot be edited using the multi-widget editing feature.

Delete or Modify a Custom Dashboard

For existing Controller-level custom dashboards:

1. Click **Dashboards & Reports**.
2. Click **Dashboards**.
3. In the Dashboards list, select the dashboard that you want to edit, delete, copy, share, or export, and click the appropriate button.

For existing dashboard templates:

In the My Dashboards tab of an entity associated with the dashboard:

- To modify an existing dashboard template, select it in the list and click **Edit** (pencil).
- To remove an existing dashboard, click **Manage Dashboards** from the dropdown, select the template in the list, and then click **Delete.**

Associate a Custom Dashboard Template with a Tier or Node

The My Dashboards tab of the built-in tier and node dashboards provides access to custom dashboards that are associated with the tier or node.

To view an associated custom dashboard populated with data from the entity, click the dashboard in the list displayed by the My Dashboards tab.

To associate one or more existing dashboard templates with the entity:

1. Click **Associate Existing Dashboard with the Tier/Node** for an entity that has no associated dashboards.
   Alternatively, click **Associate Dashboards** from the dropdown for an entity that already has at least one associated dashboard.
2. In the dashboard picker, move the dashboards to associate from the **Available Dashboards** list to the **Selected Dashboards** list.

To disassociate a dashboard template with an entity:

1. In the dropdown, click **Associate Dashboards**.
2. In the dashboard picker, move the dashboards to disassociate from the **Selected Dashboards** list to the **Available Dashboards** list.

Dashboard Templates Associated Across Applications

All the widgets in a custom dashboard template that is associated with a tier or node in a different application must use the default baseline if they use a baseline.

To set a widget's baseline to the default baseline, or to verify that it is set, in the **Select Baseline** section of the **Advanced** panel of the widget **Edit Series** pane, select **Default Baseline**.

Manage Custom Dashboard Templates

To perform miscellaneous operations on dashboard templates:

1. In the dropdown, click **Manage Dashboards**.
2. From the **Tier** or **Node Dashboards** panel, you can:

    - Delete, copy, import, or export a selected dashboard template.
    - Create a new dashboard template.

- Associate an existing template with tiers or nodes in the application.
- View and edit an existing dashboard template.

Import and Export Custom Dashboards and Templates Using the UI

This page provides instructions on importing and exporting custom dashboards in AppDynamics.

Exporting custom dashboards and templates generates a JSON file that you can import into another AppDynamics Controller.

You can import and export custom dashboards and templates, from the Controller UI, or programmatically with the REST API. See Configuration Import and Export API to import and export programmatically.

Export Custom Dashboards

To export a custom dashboard in the Controller UI:

- From the custom dashboard list, select the dashboard that you want to export and select **Export** in the **Dashboards** menu bar.
- From the dashboard editor of the dashboard that you want to export while in **Edit** mode, select **Export Dashboard** from the **More Actions** dropdo wn.

If you have edit permissions for dashboards, you can use either method. If you do not have edit permissions, use the **Export** option from the custom dashboard list.

The dashboard is exported to a file named `CustomDashboard_<dashboard_name>_<unique_id>.json`. Optionally, you can rename the file before importing it.

Export Custom Dashboard Templates

After creating a custom dashboard, you can export it as a template. Templates enable you to quickly create new dashboards.

To export a custom dashboard template, click **Export** in the toolbar of the custom dashboard.

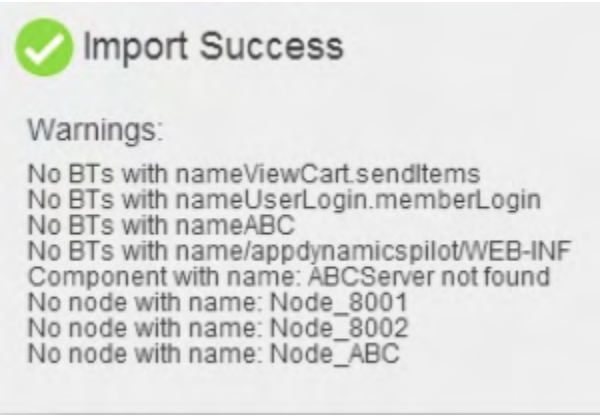
Importing and exporting are for re-using the template in a different application or a different Controller. If you want to use the template in another tier or node in the same application, you do not have to export and import it. You can simply associate it. See Associate a Custom Dashboard Template with a Tier or Node.

Import Custom Dashboards

You can import a dashboard file to create a new dashboard based on a previously exported one.

1. From the custom dashboards list, click **Import** in the menu bar.
2. Click **Choose File** and navigate to the previously exported JSON file that you want to import.
3. Click **Open**.
4. Click **Import**.

If the metrics, nodes, tiers, or applications used in the exported dashboard do not exist in the new Controller environment, warning messages display:



Even when the metrics are not available in the new Controller environment, your import succeeds and saves the widgets and layout of the dashboard. You can modify the widget properties substituting different metrics to create a similar-looking dashboard in the new environment.

Import Custom Dashboard Templates

1. From the pulldown of the My Dashboards tab of the tier or node dashboard from which you want to import a template, click **Manage Dashboards**.
2. In the Tiers/Nodes Dashboards list of dashboard templates, select the dashboard template that you want to import and click **Import**.

Dashboard Recovery

This page provides instructions on recovering custom dashboards in AppDynamics.

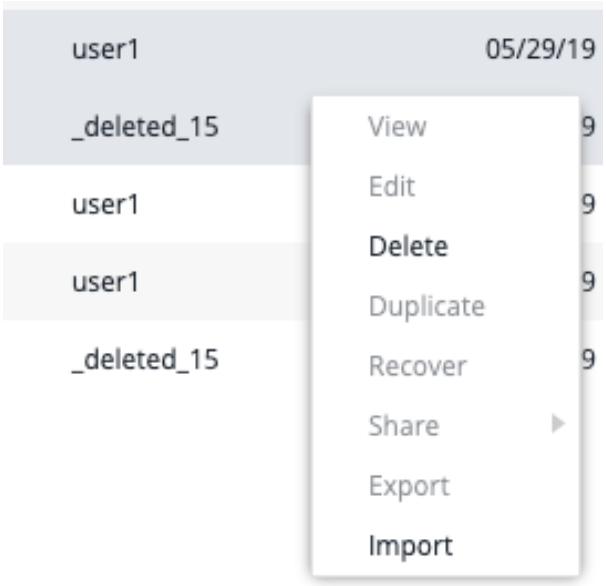
When you delete a user from the user list, any custom dashboard that the user created becomes non-functional with no permissions and is unable to execute properly. This impacts all associated dashboard-based reports and shares. To solve this problem, you can use the **Recover** option.

Recovering a non-functional dashboard reassigns ownership to the recovering user which automatically makes the custom dashboard, its shares, and related reports functional again.

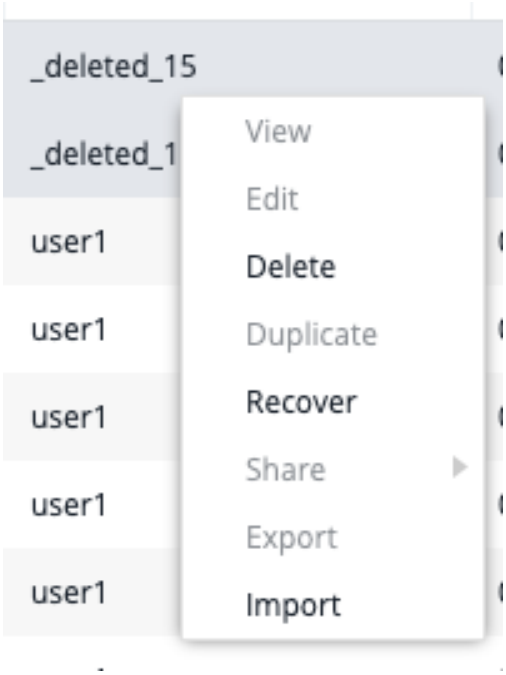
> The **Recover** option is active only when the dashboard owner is a deleted user.

To recover a non-functioning dashboard:

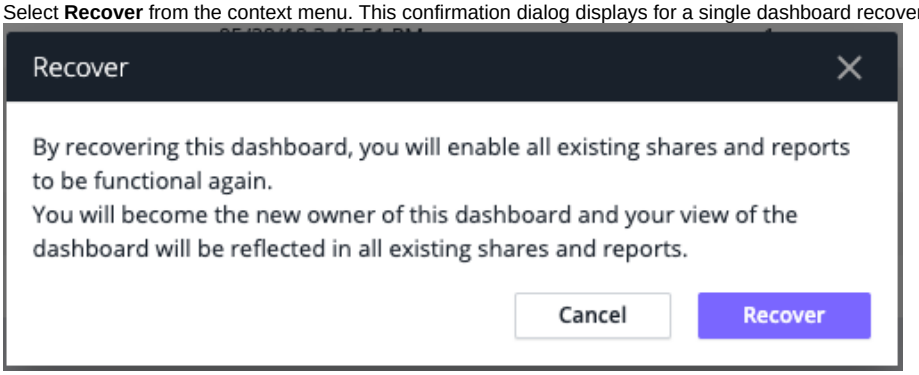
1. Log in to the Controller as a user with VIEW_DASHBOARDS and CREATE_DASHBOARDS permissions.
2. Access the custom dashboards listing page.
3. Select the non-functioning dashboard and right-click to access the context menu. When the owner of the dashboard is active, the **Recover** option remains disabled:



> When the owner is deleted, the **Recover** option is enabled automatically.

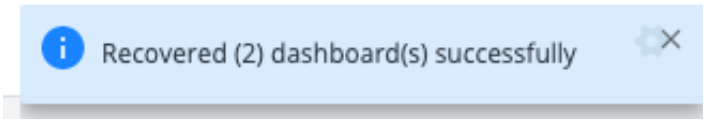


4. Select **Recover** from the context menu. This confirmation dialog displays for a single dashboard recovery:

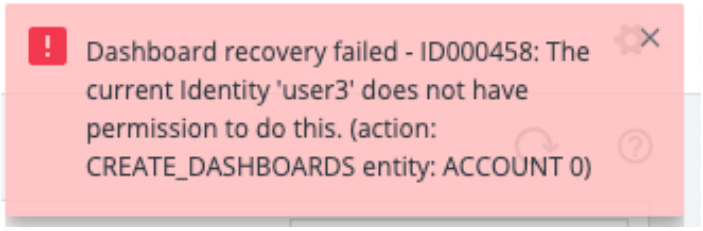


> You can select multiple non-functional dashboards and recover all of them at once. To select multiple custom dashboards, use **Ctrl-Click** for non-contiguous rows and **Shift-Click** for contiguous rows. You must ensure that the owners of all the selected dashboards are deleted users, otherwise the **Recover** option remains disabled.

5. Once the dashboard is recovered successfully, this message popover displays:



To recover a dashboard, you must have the CREATE_DASHBOARD permission. If you execute the **Recover** option without this permission, this message displays:



See Visibility and Permissions.

Duplicating a dashboard creates a new dashboard with the same capabilities as the non-functioning dashboard. However, duplication does not create the shares or the associated reports. The result is that the new dashboard operates using a deleted user's permissions (as with Recover), but you must create the shares and reports manually.

In some cases, duplicating the inactive dashboard helps you regain control over the shares and reports. However, there may be some cases when duplication would not work. You should use the Recover option to retain all existing shares and related reports instead of duplication.

Virtual War Rooms

Related Pages:

- Custom Dashboards
- Monitor Events
- Create and Manage Custom Roles
- Widgets

This page provides an overview of virtual war rooms in AppDynamics.

A war room is a combination of interactive dashboard and notes, integrating a real-time custom dashboard with collaborative troubleshooting in a focused chat room. When troubleshooting an issue, you can add notes and data using a timeline that is visible to all war room participants immediately.

The virtual war room:

- Orients everyone to the same page by viewing the same real-time data.
- Keeps the focus on metrics that translate to the business value the application delivers.
- Fosters communication among a broad audience.
- Identifies resolution criteria and assigns ownership for resolution tasks.

Start a War Room

Start a virtual war room or a collaborative chat room when you want to troubleshoot an issue.

You must have account-level permission (Create War Rooms) to start a war room.

1. Start creating a war room using one of these three options:
    - Navigate to **Applications** > **Troubleshoot** > **War Rooms** > **My Active War Rooms**, click **Start a War Room**.
    - Navigate to **Applications** > **Application Dashboard** > **Events** > **Actions**, click **Start a War Room**. Access the events list through the Events tab available in many of the built-in dashboards.
    - If you integrated Atlassian JIRA, navigate to **Applications > Troubleshoot > Health Rule Violations** and click **Start War Room** under t he JIRA/War Room column.
2. Navigate to **JIRA Issue** > **AppDynamics Incident** and click **Start or Join a War Room associated with this Incident**.
3. Name your war room and click **OK** to create a new room, or select **From Template** to create a new room using available war room templates. See War Room Templates.
4. Click **Add Widget** to add your widgets to the war room. See Widgets.
5. Click **Save**.

You can view all the war rooms you created on the **My Active War Rooms** page, and edit your widgets using the **Actions** menu.

War Room Templates

You can re-use the layout of an existing war room by saving a war room as a war room template. This can save time in creating new war rooms with numerous widgets. You can save an existing war room as a template while you are working on it, or just before you close it.

Create a War Room Template

1. From the Actions menu of the current war room from which you want to create a template, click **Save as Template**. Alternatively, when you end a war room, select **Save as Template**.
2. Provide a name for the template and an optional description.
3. Click **OK**.

The new template is now available to create other war rooms. You can view all your war room templates under **War Rooms > Templates**.

Delete a War Room Template

1. Click **War Rooms > Templates**.
2. From the list of war room templates, select the template to delete.
3. Click **Delete Template** and confirm the deletion.

Manage Presenters

The user who starts a war room is the first presenter.

Only the presenter can:

- Modify the collaboration space to add, delete, and configure widgets.
- Make any other participant a presenter. There can only be one presenter at a time.

- Share and stop sharing the war room with other participants.
- End the war room.

Presenters can also do everything that participants can do.

Change a Presenter

Right-click on any participant name in the participant list and select **Make Presenter**.

The ability to edit and control sharing of the collaborative dashboard and to end the war room passes to the new presenter. Other participants cannot see the war room until the presenter shares it. This enables the presenter to set up widgets privately before showing the war room to others.

End the War Room

Click **End War Room**. When prompted, confirm that you want to end it.

Only the presenter can end the war room. A war room is automatically ended after 60 minutes of inactivity. Inactivity means no participants are viewing the war room pane.

Manage Participants

Everyone who has joined the war room but who is not currently a presenter is a participant. A participant can be:

- An AppDynamics participant is an authenticated UI user with access to the AppDynamics account that is hosting the war room.
- A guest participant does not log in to the AppDynamics UI, but has been invited to join by an AppDynamics participant. This user type functions as an anonymous, read-only participant.

AppDynamics participants can:

- View a war room to which they have been invited.
- Invite other participants.
- Add notes.
- Start a new war room, if the participant has Create War Rooms permission.

Guest participants can view a war room to which they have been invited.

View a War Room

You can view only the war rooms in which you are a presenter or a participant. The list does not display all the war rooms currently in progress on the Controller.

To view a war room, you should have created one, got invited to one, or have access to the shared URL available to anonymous users.

To access a war room, either:

- Click the link in the invitation that you received to join the war room.
- Click **Troubleshoot** > **War Rooms**, and then select the war room to join from the war rooms list.

Invite Another Participant to the War Room

Send a link to invite another participant to the war room:

1. From your active war room, click **+ | Participants**.
2. From the **Invite Users** pane, copy the URL for joining this War Room.
3. Email or SMS the link to the persons whom you are inviting, with instructions to click the link from a browser.
4. Click **OK** to close the **Invite Users** pane.

After clicking the link, the recipient is invited to log in as an AppDynamics participant or as an anonymous guest.

Add a Note

1. From your active war room, click **+ | War Room Notes**.
2. From the **Enter Notes** pane:
    a. Enter your note in the text field.
    b. (Optional) Qualify your note as good or bad.
    c. Click **Post**.

Automatic Save, Synchronize, and Archive a War Room

The war room is automatically saved and all participants' views of it are synchronized whenever the presenter makes a modification to the dashboard. The saving and synchronizing happens continually and is transparent to the participants when it occurs. In addition, the presenter can always force a save and sync manually by clicking **Save** at the top of the war room dashboard.

If you restart the Controller while a war room is in progress, the war room is saved and you can access it after the Controller comes back online.

Reports

This page provides an overview of reports in AppDynamics.

AppDynamics extracts data from dashboards and creates scheduled reports. You can configure automatic report generation for a regular schedule or create reports manually.

Scheduled Reports

Scheduled reports are automatically created on a regular interval. AppDynamics creates reports with the data pulled from Dashboards and sends it to the configured list of email recipients, as scheduled. To deliver a report instantly, select the **Send Report Now** option available for a report.

You must have account-level permissions to view and configure scheduled reports. See account-level permissions.

Report Types and Formats

You can schedule periodic generation and delivery of the these report types to a list of email recipients:

<table>
<tr><th>Report Type</th><th>Captured Information</th></tr>
<tr><td>Application Health Report</td><td>Application health data from the application dashboard that includes:
Business transaction and server health data.
Transaction scorecard.
Events and exceptions.
Load, response time, and errors graphs.</td></tr>
<tr><td>Dashboard Report</td><td>Dashboard data in the selected custom dashboard.</td></tr>
<tr><td>Controller Audit Report</td><td>Audit entries that include:
User logins and information changes.
Controller configuration changes.
Environment properties changes.
Application properties and object changes such as policies, health rules, and the entities listed here:
<table><tr><td>Date and time range</td><td>ObjectType</td></tr><tr><td>UserName</td><td>ObjectName</td></tr><tr><td>Action</td><td>ApiKeyId (if applicable)</td></tr><tr><td>ApplicationName</td><td>ApiKeyName (if applicable)</td></tr></table>AppDynamics supports PDF, JSON, and CSV file formats for this report type.</td></tr>
<tr><td>Home Screen Report</td><td>Overview data from the home page.</td></tr>
<tr><td>All Applications Summary</td><td>Data summary from the applications page.</td></tr>
<tr><td>User Experience: Browser Apps</td><td>Browser RUM data from the Browser App Dashboard Overview page.</td></tr>
</table>

Manage Scheduled Reports

You can create and deliver scheduled reports on a regular interval. To create a report identical to the existing one, you can duplicate the existing scheduled report and modify relevant fields, as required. You can add a set of email recipients for the report delivery.

In addition to the scheduled delivery of reports, use the **Send Report Now** option to send the generated report immediately to the email recipients. From the **Reports** page, select your report and click **Send Report Now** from the right-click menu.

Create a Scheduled Report

You need to have the account-level Configure Scheduled Reports permission to create scheduled reports.

1. Click **Dashboards & Reports > Reports > Add Report**.
2. From the **Create Scheduled Report** page, you can customize your report:
    a. Enter **Report Title** and **Report Subtitle**.

> You can label a report CONFIDENTIAL using **Report Subtitle**.

b. Optionally, select **Show Title Page** to include a title page at the beginning of your report file.
c. Select your **Report Type** from the dropdown. Based on this selection, fields in the **Report Data** tab vary.
d. Make necessary changes in the Schedule, Report Data, and Recipients tabs, and then click **Save**.

Create a Controller Audit Scheduled Report

Use this report to view changes made to the user information, Controller configuration, and application properties.

1. Click **Dashboards & Reports > Reports > Add Report**.
2. Enter **Report Title** and **Report Subtitle**.

   > You can label a report CONFIDENTIAL using **Report Subtitle**.

   Optionally, select **Show Title Page** to include a title page at the beginning of your report file.
3. Select **Report Type** > **Controller Audit** to define the fields in the Reports Data tab.
4. Set the time ranges. You can create and manage custom time range, if required.

   > Custom time range options are available for all the Report Types.

5. Select your report file format as PDF, JSON, or CSV.
   Optionally, uncheck **Show Diff** to remove the *Object Changes* column from your report file.
6. Select the data to include or exclude from the dropdown. Repeat as necessary with these options:

Filter Type	Attribute	Attribute Value	Result
Exclude/Include	Same	Different	Audit entries matching with any of the attributes are excluded/included.
Exclude/Include	Different	Different	Only audit entries matching with all of the attribute values are excluded/included.

7. Enter the attribute value.
8. Click **+Add**.

> **Alternative**
>
> You can use the Controller audit log that replicates the Controller Audit Report. This audit log monitors user activities and configuration changes in the Controller. See Controller Audit Log.

Scheduled Report Management

You can manage scheduled reports from the **Dashboards & Reports > Reports > Scheduled Reports** page. To edit, duplicate, enable, disable, export, import, and delete reports, use the menu bar on the **Scheduled Reports** page, or the right-click menu options available for each report in the list.

Managing a report requires account-level permissions such as View Scheduled Reports and Configure Scheduled Reports.

To create and send the scheduled report on a regular interval, ensure that you have enabled the report. You can disable a scheduled report to put it on hold for a certain length of time. You can verify the current status of a report using the **Enabled** column in the list of reports. For any report in the list, you can use the **Send Report Now** option to generate and deliver it instantly.

Dash Studio

AppDynamics Dash Studio (Preview) is a next generation dashboard designed to make building dashboards faster and easier. It is available in the Dash Studio (Preview) tab under **Dashboards & Reports**. The previously created dashboards are still available unmodified, under the Dashboards tab.

Dash Studio is available as an early preview release and so we seek your feedback through the AppDynamics Community and through the **Feedback** button available in Dash Studio. Dash Studio is delivered through the UI Cloud App to SaaS customers.

> Enhancements and new capabilities will be added to Dash Studio (Preview) periodically.
>
> Dash Studio is available for on-premises customers >= 4.5.16.
>
> SaaS requires 4.5.15.1.

Visual Dashboard Editor

This page provides an overview of the Dashboard Editor in AppDynamics.

The Dashboard Editor is available next to the existing Dashboards tab.

> Dash Studio is not recommended for mission-critical dashboards as it is intended for an early customer preview.

The Dashboard Editor allows you to:

- Quickly align widgets.
- Resize widgets on a dashboard by grabbing the edges of the widget with a click-and-drag to resize.
- Select multiple widgets, copy and paste them using GUI controls, or using command-C or command-V controls, so that you can replicate them on a single dashboard.
- Edit widget titles by editing the widgets directly.
- Edit widget attributes such as: titles, line colors, and labels.
- Access commonly used tools from the toolbar with a single click.

What's New? Button

The **What's New** button provides information about the latest updates in the Dash Studio.

Click the **Show Documentation** button to access the online documentation for Dash Studio.

Dashboard Library

All dashboards created in the Dash Studio are available in the **Dashboard Library**, which provides a list of all Dash Studio dashboards. Select the **Show My Dashboards Only** toggle button to see a list of dashboards created in the Dash Studio.

Dashboard Editor

The new dashboard canvas offers a visually attractive and responsive display in both edit and view modes. The canvas and associated toolbars:

- Allow you to easily edit the dashboards manually.
- Allow widget data binding and editing of some widget properties (such as, titles and graph axes labels) on the canvas directly.
- Are architected to enable future functionality such as, repeating widgets and allowing column width lock in responsive dashboards.

Edit and View Modes

To optimize the Dash Studio (Preview) experience, you can switch between these two display modes:

- **View** mode (default mode for dashboards)
- **Edit** mode

In **View** mode, the toolbar disappears, and the dashboard displays to the largest extent possible in the browser panel in which it is housed.

The **Edit** mode enables you to edit the content of the selected dashboard. To return to the dashboard library from the **Edit** mode, click  on the top-left of the page.

You can apply these layouts to your dashboard:

- Responsive Layout
- Fixed Aspect Ratio

The **View** icon (available on the top-right of the page) allows you to preview the dashboard. On the preview page, click the **Edit** icon to return to the Dashboard Editor.

Feedback Button

Use the **Feedback** button to submit feedback about your experience using Dash Studio. The submitted feedback is sent to the AppDynamics product team.

Auto Refresh Interval

Enable the **Auto Refresh Interval** option under the **Dashboard Properties** panel to refresh the dashboard automatically at specific intervals. Auto-refresh ensures that the dashboard data and layout is refreshed regularly. You can select an option from the **Auto Refresh Interval** dropdown or select **Custom** fr om the list of options to customize the interval value. The default refresh interval is two minutes. The set interval value persists when you switch between edit and view mode, and when you share, import, or export a dashboard.

Help Popover

The **Help** popover appears at various places on the dashboard UI. This popover provides details about features, concepts, and product changes. Click "**Ok ay, got it**!" to close the popover.

Dashboard Sharing

The sharing capability in Dash Studio allows you to share the dashboard with business users and with those who do not have an AppDynamics account. This helps multiple users access the dashboard and enables effective collaboration among the users.

You can also:

- Stop sharing the dashboard.
- Copy the shared URL to the clipboard.
- Open the shared URL in a new panel.

Import and Export Dashboards

This feature allows you to export and import JSON dashboard metadata between Dash Studio instances running in different environments.

Export Dashboard

To export a dashboard in Dash Studio:

1. Select the dashboard to export from the dashboard list.

   > The **Export** icon) is enabled only when you select a dashboard from the dashboard list.

2. Click **Export** when available in the dashboard toolbar.
   The exported dashboard is generated as a JSON file on your system. You can rename the downloaded file later.

Export Dashboard with Edit Permissions

To export the selected dashboard with **Edit** permissions:

1. Right-click on the selected dashboard.
2. Select **Export** icon from the dropdown.
   The exported dashboard is generated as a JSON file on your system.

Import Dashboard

To import a previously exported dashboard file in Dash Studio:

1. Click **Import** to display the **Custom Dashboard Import** dialog.
2. Click **Choose File** and navigate to the previously exported JSON file that you want to import.
3. Click **Open**.
4. Click **Import** to open the imported dashboard and display it with a new name.

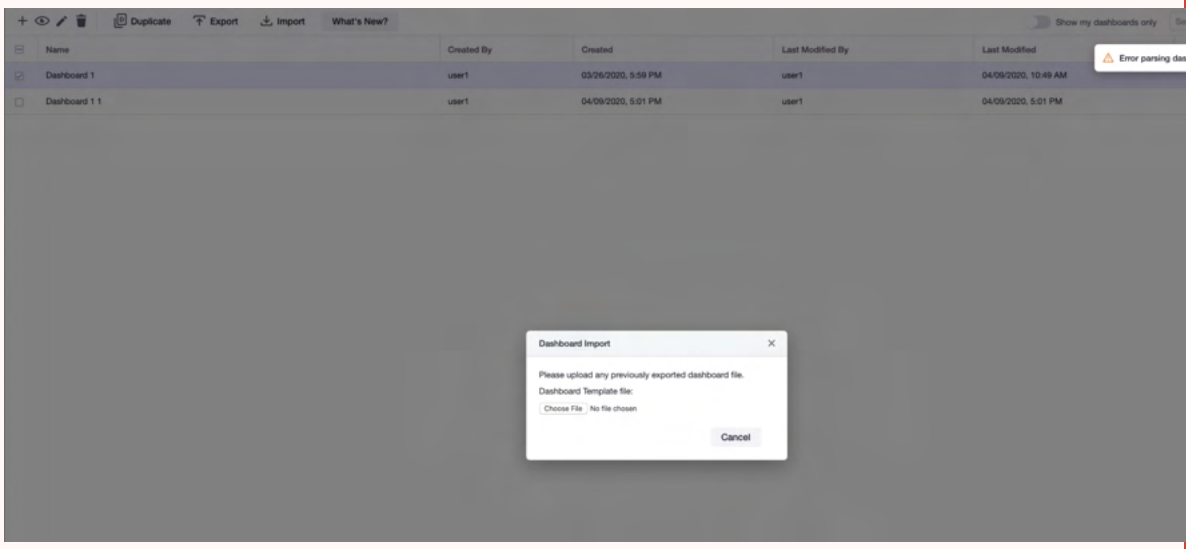
Import Dashboard with Edit Permissions

To import the selected dashboard with **Edit** permissions:

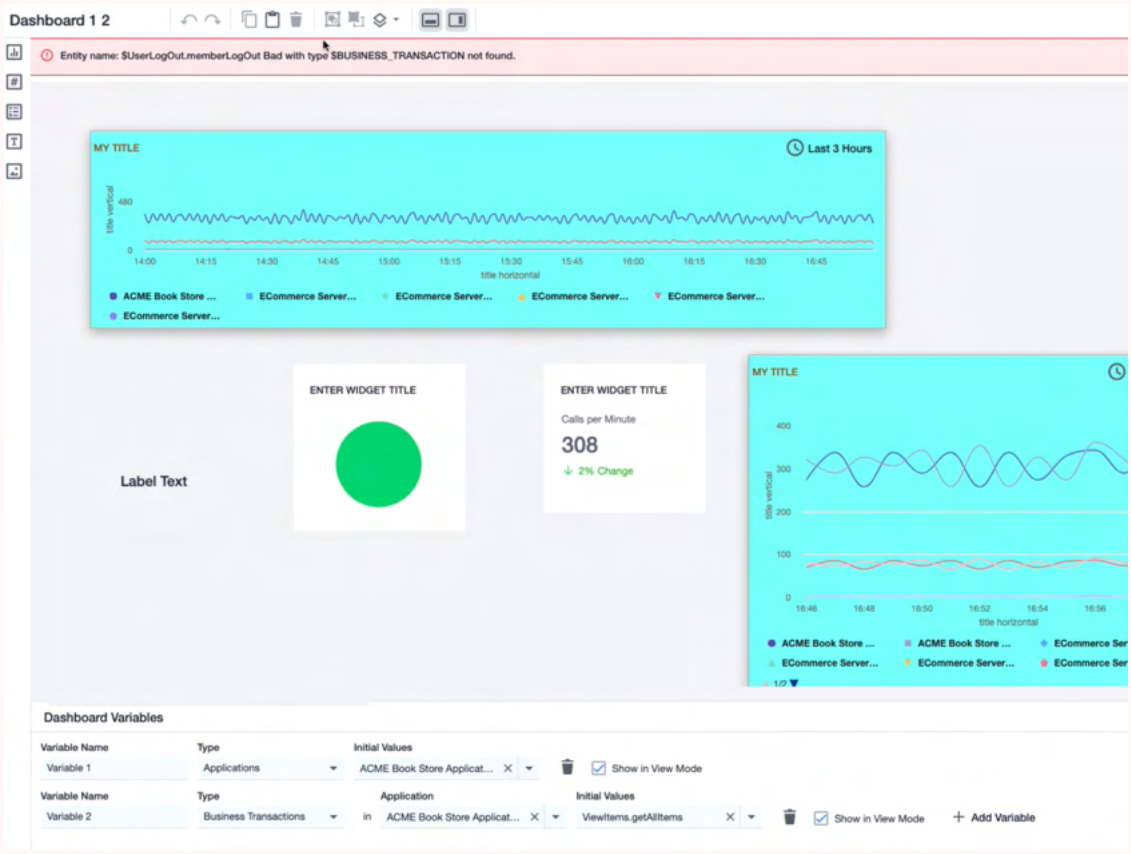
1. Right-click on the selected dashboard.
2. Select **Import** rom the dropdown.
3. Click **Choose File** to navigate to the dashboard file to import.
4. Click **Open**.
5. Click **Import** to open the imported dashboard and display it with a new name.

These errors may display when you try to import a dashboard file:

- Parsing error when you import a bad or malformed dashboard file.



- File Not Found error when you import a dashboard from another Controller.



Dash Studio Layout

This page provides an overview of the Dash Studio layout in AppDynamics.

You can organize the dashboard layout and explore important system data in the Dash Studio. You can find required dashboards, delete dashboards, and create new ones. You can also place widgets on the canvas in the Responsive Mode layout and Fixed Aspect Ratio layout. The canvas is a backdrop where you can drop content in, such as widgets.

Dashboard Organization

Create Dashboards

Dash Studio allows you to create a dashboard without having to name it, or indicate its layout or characteristics. If you accidentally close a dashboard without naming it, the dashboard is auto-saved and named "Untitled-X." X is the next untitled dashboard in the collection of dashboards that you are authorized to access and is saved in the Dashboard Creation Directory. Untitled-X displays as the dashboard title on the page as soon as the new dashboard canvas loads. You can rename the dashboard later.

Auto-Save

Dashboards are auto-saved by default.

Responsiveness and Sizing

You can resize the Dashboard canvas to any size which enables you to switch between editing and viewing mode. By default, the dashboard automatically resizes to fit on the page. The width of some columns in the dashboard are locked so that when you resize a dashboard, a locked column is not resized. The default canvas sizing fits the dimensions of a typical laptop.

Dashboard Editor Actions	Description
Adding Widget	You can drag and drop widgets from the toolbar onto the dashboard canvas.
Alignment and Placement	You can place the widgets anywhere on the dashboard canvas, including placing widgets on top of other widgets or running off the edge of the canvas itself. Use the red alignment lines as a quick visual to check the alignment of the widgets. These lines also indicate when the widgets are at an appropriate distance from other widgets.
Group Creation	You can select multiple widgets by dragging a box over them with a mouse, or holding **Shift** and selecting all widgets with a mouse click.
Selection and Undo/Redo /Copy/Paste	You can select widgets, or groups of widgets by clicking them, or dragging a mouse-generated box over by holding down the left mouse button. You can select multiple widgets by either dragging the box, or by a command-click action. • You can undo or redo an action using a toolbar command, or by command-Z and command-Y functionality on the keyboard. • You can copy, cut or paste groups of widgets using a toolbar command, or by command-C, command-X, or command-V functionality on the keyboard.
Selection in Edit Mode	You can select widgets or groups in edit mode by clicking them, or dragging a mouse-generated box over by holding down the left mouse button. You can select multiple widgets by either dragging the box, or by command-click action. In view mode, data queries are not exposed. To drill down to the next data level, you must double-click. • For Widgets: Single click selects a widget, enables its movement, and exposes an editing icon at the top right of the widget. To edit data, click on the editing icon. Double-click on the widget to drill down to the next data level. Google Data Studio is a good example of this interaction. • For Groups: Single click a group selects that group, enables its movement, and exposes the editing icon. Clicking again selects the widget. Double-clicking a widget in a group has the same effect as clicking once on the group, and then clicking the widget. To drill down on the widget, you must double-click again.

Edit and View Modes

You can view Dashboards in these two modes:

- **Edit** mode
- **View** mode

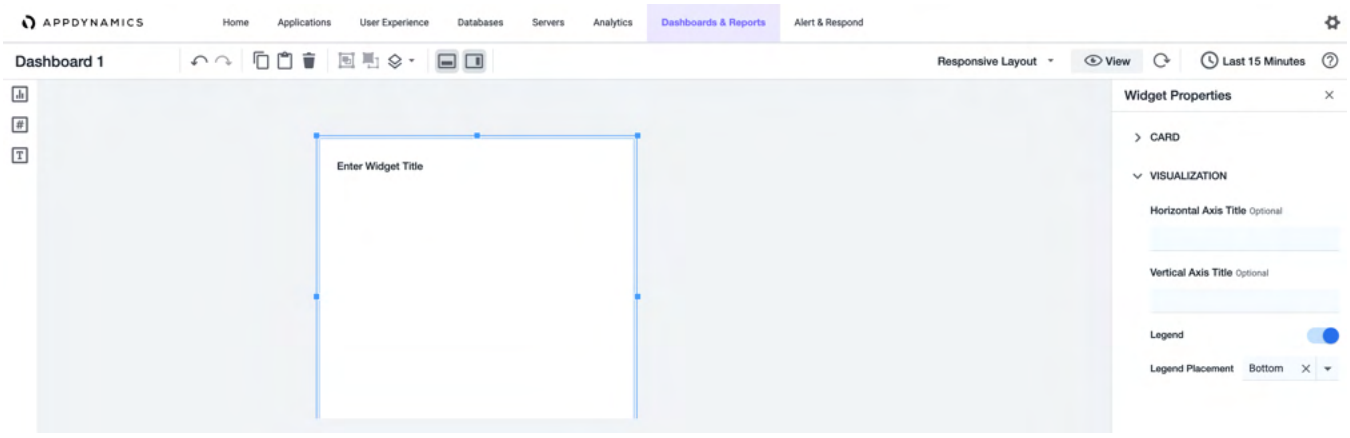
By default dashboards are in **View** mode. The toolbar does not display in **View** mode.

General Layout

Toolbar

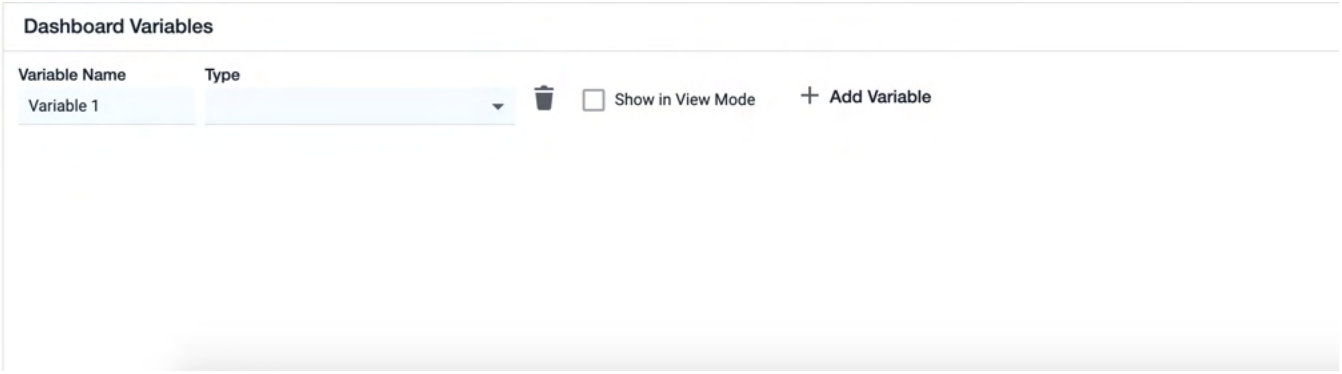
The dashboard editor tools available in the toolbar enable you to perform these actions:

Actions	Description
Undo and Redo	The toolbar has two buttons for undo and redo actions on a dashboard. You can also perform these actions using command+Z and command+Y keyboard functions.
Copy, Paste, and Delete	You can use tools from the editing toolbar to copy, paste, or delete widgets, or containers from a dashboard. You can also perform these actions using command-C, command-V, and the Delete keys on the keyboard.
Dashboard title	You can edit dashboard title. By default, a new dashboard is titled "Untitled Dashboard."



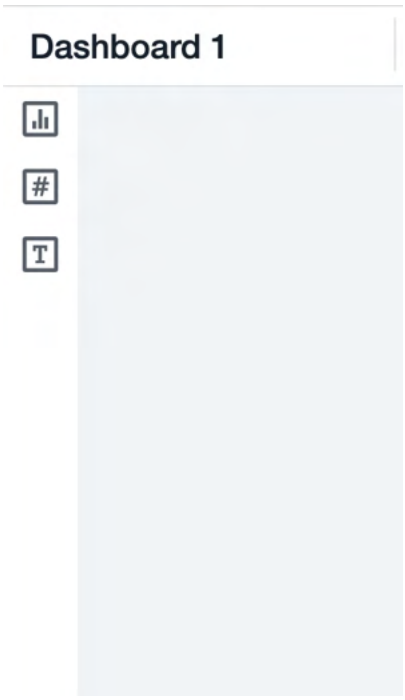
Data Panel

The Data Toolbar at the bottom of the Dashboard Editor enables you to bind containers and visualizations to data. See Data Binding for details.



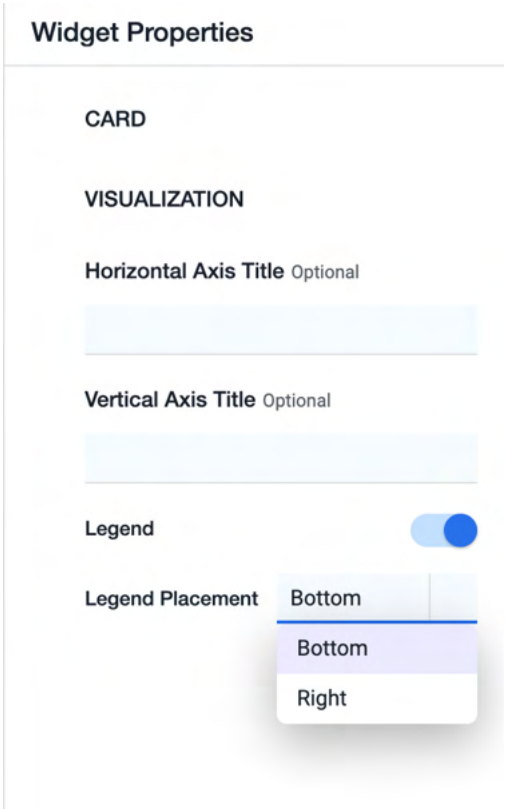
Widget Palette

The **Widget Palette** on the left-side of the Dashboard Editor allows you to drag and drop widgets from the palette on to the canvas.



Widget Properties Panel

The **Properties Panel** on the right-hand side of the Dashboard Editor enables you to set the properties of a widget, such as: background color, shadow, padding, and basic display options.



Dash Studio Widgets

This page provides an overview of existing Dash Studio widgets in AppDynamics. Widgets create a visual representation of your data in the dashboards.

> Only certain widgets of Beta Dash Studio have been included on this page. The page will be updated when new widgets release with the product.

Card Component

Cards are small containers for placing the widgets. Cards have a title, border, and sometimes a footer and actions related to the specific context of the Card. These containers have text labels to clearly describe the information. You can rename or delete them.

Grouped Widgets

The widget grouping functionality enables you to organize two or more widgets on a card component or on their respective cards. You can apply widget properties to the entire group of widgets instead of a single widget.

- Each widget is created on a card by default.
- A card component, for a widget or group of widgets, has a title and a background.

Create a Widget Group

To create a group:

1. Select two or more widgets.
2. Click the group icon  in the top panel.
   Select one of these options:
   - **Group -** Groups the widgets on their respective cards.
   - **Group In Card -** Groups the widgets on a card component.

To separate the widgets in a group, click the ungroup icon  in the top panel.

Group Properties

You can apply these properties to modify the appearance of grouped widgets:

Property	Description
Group In Card	Group all the selected widgets on a card.
Background Color	Set the background color of the widget. You can either leave the the default option as it is or customize it.
Padding	Add extra spaces between the top and bottom of the text within the widget. The default value is 1.
Border	Set the border around the widget.
Shadow	Set a shadow around the widget.
Title	Define a title for the widget.
Title Alignment	Align the title of the widget.
Title Color	Set the color of the widget title.
Subtitle	Define a subtitle for the widget.

See group time range.

Time Ranges

Global Time Range

The **Global Time Range Selector** option on the top right enables you to set a time range for the dashboard. This can be for the last five minutes, hours, weeks, or even months.

Widget Time Range

You can compare widgets across different time ranges by setting a time range that is distinct from the global time range.

To set the time range for a single widget independent of other widgets:

1. Select the widget to set its time range.
2. Go to **Widget Properties** on the left panel and click **Widget Time Range**. Two options display: **Global Time Range** and **Widget Specific Time Range**.
3. Select **Widget Specific Time Range** and set a time range distinct from Global Time Range. Select the **Show Time Range** toggle to display the time range on the top of the selected widget:

> You can enable the **Show Time Range** toggle for a widget to change its time range in **View** mode.

Group Time Range

You can set the time range for a set of grouped widgets using the **Group Time Range** option under **Group Properties** panel. It provides these options:

- **Inherit Time Range** - Set the same time range as the global time range for each widget of the group.
- **Group Specific Time Range -** Set the time range for each widget of the group by selecting an option from the dropdown. You can also click **Cust om** to create a customized time range.

Toggling the **Show Time Range** option hides or displays the time range in the upper-right corner of each widget in the group.

Simple Navigations

Simple navigation actions in Dash Studio enables basic user interactions and helps you to seamlessly navigate to any specific part of the product. The navigation actions allow you to drill down into complex and detailed information about a selected entity. You can utilize this capability as part of the troubleshooting processes and for root cause analysis.

> This feature is available only for Time Series, Pie, Label, Metric Number, and Image widgets.

Navigate with a Single Click

To perform navigation actions with just a click in the view mode, click the widget for which you want to set this action:

1. Click **Single Click Action** option available under the **Widget Properties** panel.
   It displays the following options:
   - **Destination Type** - You can select one of the following:
     - **Metric Browser -** This option is selected by default. Click the widget in the view mode to navigate to the selected metric browser.
     - **URL** - **Destination Link** field displays, where you must enter the desired link. Click the widget in the view mode to navigate to the selected link.
     - **Dashboard** - **Dashboard Name** field displays when you select this option. Enter the name of an existing dashboard or select an option from the dropdown list. The following option displays if the selected destination dashboard contains variables:
       - **Variable Passing 1** - This option allows you to modify the default value of multiple variables in an existing Dashboard.

> For example, if “Dashboard 1” has a variable named “Nodes 1” with value “Node_8000”, you can change the preset value of “Nodes” to any other desired Node value using **Variable Passing 1**. Similarly you can modify a second variable named “Nodes 2” in the same dashboard using the subsequent field, **Variable Passing 2**.

Click the widget in the view mode to navigate to the selected dashboard.

- **Show In** - Select **Current page** or **New Window**.

2. Click the **View** icon at the top right after entering valid details.
   In the View mode, click the widget to navigate to the appropriate destination.

Widget Types

These widget types are available in the Dash Studio.

Time Series Widget

A Time Series widget is a two-dimensional time-based series of data points connected by a line. There can be multiple lines on one chart.

Widget Properties

You can select the following options under the **Widget Properties** panel to modify the visual appearance of the Time Series widget:

Property	Description
Background Color	Set the background color of the widget. You can either leave the the default option as it is or customize it.
Padding	Add extra spaces between the top and bottom of the text within the widget. The default value is 1.
Border	Set the border around the widget.
Shadow	Set a shadow around the widget.
Title	Define a title for the widget.
Title Alignment	Align the title of the widget.
Title Color	Set the color of the widget title.
Subtitle	Define a subtitle for the widget.

Property	Description
Horizontal Axis Title	Set the title of the horizontal axis on the widget. This is optional.
Vertical Axis Title	Set the title of the vertical axis on the widget. This is optional.
Legend	Set the legend.
Legend Placement	Place the legend at the bottom or on the right-side of the widget.

You can define performance baselines in a widget by adding a **Threshold** value. You can add multiple thresholds to a widget, which enables you to set threshold values based on the severity.

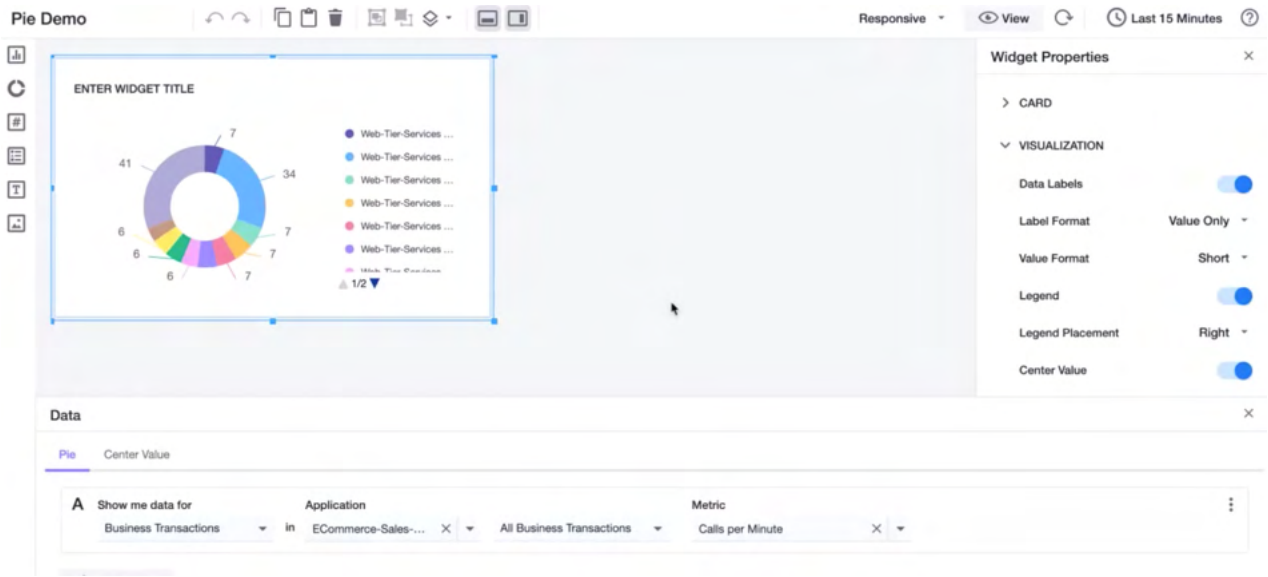
Property	Description
Label	Set name of the threshold.
Value	Set threshold value to be set.
Color	Set color of the threshold line. By default, it is black in color.

See widget time range.

See time range comparisons.

Pie Widget

Pie widget displays single consolidated values for a series of metrics for a given time range as a pie visualization.



Widget Properties

You can select these options under the **Widget Properties** panel to modify the appearance of the Pie widget:

Property	Description
Background Color	Set the background color of the widget. You can either leave the the default option as it is or customize it.
Padding	Add extra spaces between the top and bottom of the text within the widget. The default value is 1.
Border	Set the border around the widget.
Shadow	Set a shadow around the widget.
Title	Define a title for the widget.
Title Alignment	Align the title of the widget.
Title Color	Set the color of the widget title.
Subtitle	Define a subtitle for the widget.

Property	Description
Data Labels	Define the individual data points that you can turn on or off.
Label Format	Set the format of the label. You can select one of the following options: • **Value and Name** • **Value only**
Value Format	Set the format of the label value. You can select one of the following options: • **Long** • **Short** • **Percent**
Legend	Set the legend, which describes the value of each pie slice of the widget.
Legend Placement	Pace the legend on right side or bottom of the widget.
Center Value	Set the unit value.
Center Value Format	Set the format of the center unit value.
Center Value Unit:	Set the unit for the center value. This is optional.

See widget time range.

See single click action.

Metric Number

A Metric Number chart highlights a single value/KPI and is typically created by:

- A single measure which can be specified using the Data Panel.
- Combining or aggregating multiple metrics.

The value display shows:

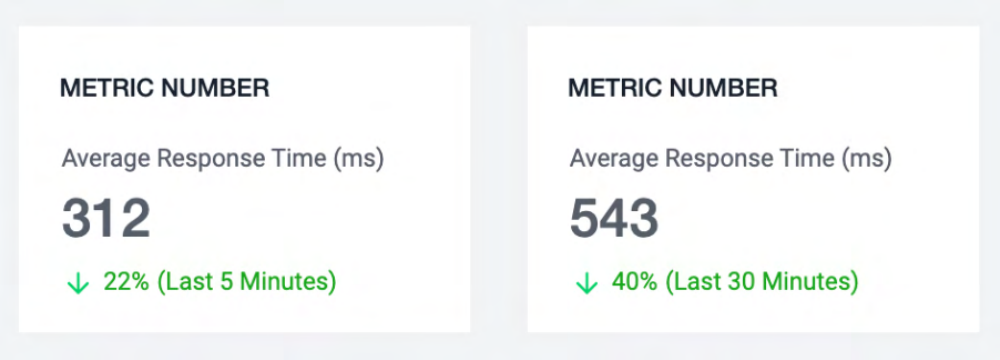
- A single measure or metric value at any instant.
- A comparison of two values (either two different metrics or a metric at different points of time).
- An aggregation of metrics across many objects or a large interval of time.

If the trend of a value over time is concerning, you can enrich these charts with contextual information to better interpret the present value, or the change in value over a time period. These charts can display numbers and text, for example, date and time of an incident.

You can also display the metric value, or a combination of these values in various ways, such as selecting a single metric:

- By name associated with a particular entity, or object that you want to learn more about.
- Aggregated across all entities of a certain type, or a provided sub-list of entities.
- Aggregated for a sub-list of entities which is dynamically determined.

The metric number helps you understand the metric value at the current point in time, and also the pattern of the metric value over a time interval.



Widget Properties

You can select these options under the **Widget Properties** panel to modify the appearance of the Metric Number widget:

Property	Description
Background Color	Set the background color of the widget. You can either leave the the default option as it is or customize it.
Padding	Add extra spaces between the top and bottom of the text within the widget. The default value is 1.
Border	Set the border around the widget.
Shadow	Set a shadow around the widget.
Title	Define a title for the widget.
Title Alignment	Align the title of the widget.
Title Color	Set the color of the widget title.
Subtitle	Define a subtitle for the widget.

Property	Description
Show Metric Name	Hide or show the name of the metric.
Unit Name	Set the unit of widget value. This is optional.

Number Format	Set the formal of the widget value. You can select one of the following options: • Short • Long • Percent
Show Trend	Hide or display the relative trend of the widget value.
Reverse Trend Color	Reverse the color of the relative trend of the widget value.

See widget time range.

See time range comparisons.

See single click action.

Gauge Widget

The Gauge widget enables you to present the metric data on a relative scale. The status is indicated by an arc, where the left end is the minimum value and the right end is the maximum value.

Following are the three types of Gauge widgets:

- Status Gauge - The default Gauge widget is the status Gauge.
- Target Gauge widget - The Gauge widget in the target mode is called the Target Gauge widget. The Target Gauge enables you to set a target value for the metrics. It is set on an arc with a triangular indicator that points to the target value. The threshold values are hidden on this type of widget.
- Threshold widget - This widget displays the threshold values. It provides a visual alert through color changes.

Gauge widget currently supports only two data binding options:

- ADQL Query
- Applications

Widget Properties

You can select these options under the **Widget Properties** panel to modify the visual appearance of the Gauge widget:

Property	Description
Background Color	Set the background color of the widget. You can either leave the the default option as it is or customize it.
Padding	Add extra spaces at the top and bottom of the text within the widget. The default value is 1.
Border	Set the border around the widget.
Shadow	Set a shadow around the widget.
Title	Define a title for the widget.
Title Alignment	Align the title of the widget.
Title Color	Set the color of the widget title.
Subtitle	Define a subtitle for the widget.

Property	Description
Metric Name	Hide or display the name of the metric.
Value	Hide or display the value of the metric.
Value Format	Set the format of the value. You can select one of the following options: • **Long** • **Short** • **Percent**.
Min and Max Values	Hide or display the maximum and minimum value to be displayed on the widget.
Set Value Range	Set the maximum and minimum value to a desired input.

Target	Enable this option to turns the status Gauge into a Target Gauge widget. You can set it to **None**, **Fixed**, or **Percentage.**

If **Target** option is enabled, all the threshold values on the widget are hidden.

Property	Description
Reverse Threshold Order	Reverse the color order for threshold values. By default, the color order is green for normal, yellow for warning, and red for critical but when you enable this toggle button the order changes to red for normal, yellow for warning, and green for critical.
All Threshold Values	Display all threshold values on the widget and the **Target** option gets disabled automatically.
Threshold Starting Values (%)	Set the threshold range between 0 to 100% where: • **Normal** range is between 0 and the defined normal threshold value. • **Warning** range is between the normal threshold value and warning threshold value • **Critical** is when it exceeds the warning threshold value

If threshold option is enabled, **Target** option is hidden from the panel.

See widget time range.

See single click action.

Health Widget

A Health Widget provides different visualizations of the health information of one or more entities for a specified time range. Use this widget to create and view health information of one, or more entities at a time, for Applications, Business Transactions, Tiers, Nodes, and Health Rules. By default, the Health widget displays health information as a ratio pie. The ratio pie visualization provides an additional option of displaying a metric value at the center of the donut pie.

Widget Properties

You can select these options under the **Widget Properties** panel to modify the appearance of the Health widget:

Property	Description
Background Color	Set the background color of the widget. You can either leave the the default option as it is or customize it.
Padding	Add extra spaces between the top and bottom of the text within the widget. The default value is 1.
Border	Set the border around the widget.
Shadow	Set a shadow around the widget.
Title	Define a title for the widget.
Title Alignment	Align the title of the widget.
Title Color	Set the color of the widget title.
Subtitle	Define a subtitle for the widget.

Property	Description

Display as	Display the widget on the dashboard as a:
	• **Ratio Donut**: By default, the Health Widget displays as a Ratio Donut.
	  • **Label Format**: Use to change the format of the label by showing either the **Count and Type** or just the **Count**.
	  • **Center Value Unit**: Use to set the unit value.
	  • **Data Labels**: Use to turn data labels on or off.
	• **Traffic Light**: Displays the health information of critical, warning, and normal status as a traffic light. The health status shown is the overall health status of a list of entities based on the **Aggregation Type**.
	"Worst Health" is the only **Aggregation Type** currently supported for the Health widget. It shows the worst health status in the given list of entities. Additional aggregation types will be included in future releases.
	• **Ratio Bar and List**: Displays health status as a ratio bar with the list of all the entities being monitored (with a health icon next to it). You can modify the size of the bar with **Bar Size** from small to medium. You can use the **Font Weight** option to change the appearance of the font inside the widget to **Light**, **Medium**, and **Strong**.
	• **Ratio Bar**: Displays the health status as a ratio bar.
	• **List:** Displays the health status as a list.

Data Labels	Hide or show the individual data points that you can turn on or off.
Label Format	Set the format of the label. You can select one of the following options: • **Count Only** • **Count and Type**
Center Value Format	Set the format of the center unit value.
Center Value Unit :	Set the unit for the center value. This is optional.

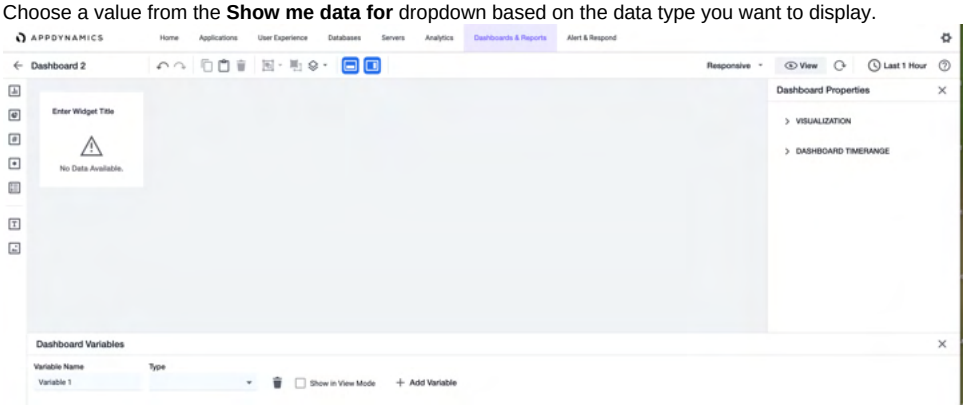
See widget time range.

View Health Information

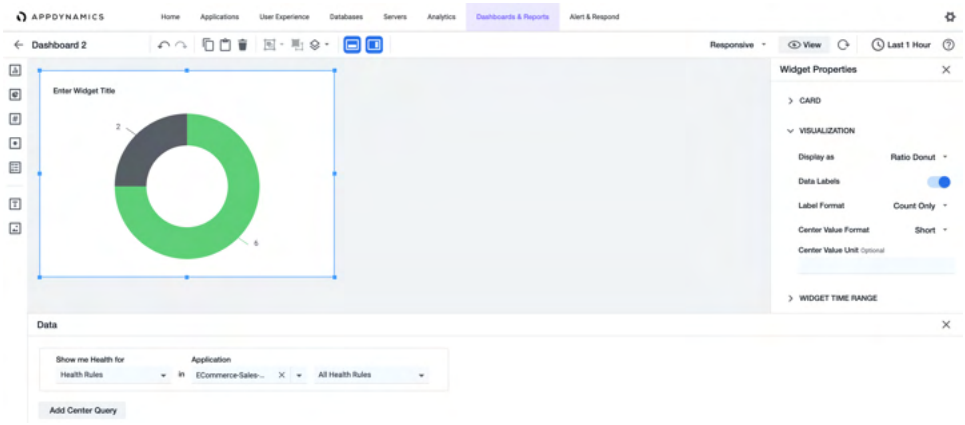
From a Data Panel, you can select the data type whose health information is to display on the Health widget.

For example, supposed you want to know the health status of the ECommerce application.

1. Choose a value from the **Show me data for** dropdown based on the data type you want to display.



   For this example, select Health Rules.
2. The subsequent input fields displayed in the **Data Panel** are specific to the data type selected in the previous step. Click the **Application** dropdown and select the application for which you want to drill down the data.
3. From the next dropdown, select a specific **Named** Health Rule from the popover list or select **All Health Rules**. For example, if you select **All Health Rules**, the Health widget displays as:



After you specify valid inputs for all fields, the widget will update on the canvas and display the results. To view the center value, click **Add Center Query**.

Datagrid Widget

The Datagrid widget provides a quick view of the list of entities associated with all the events on the Dash Studio. You can use this widget to present a collection of data in a tabular format. It improves the troubleshooting capabilities by reducing the number of drill-downs required to perform and navigate to other panels.

Widget Properties

You can select these options under the **Widget Properties** panel to modify the appearance of the Datagrid widget:

Property	Description
Background Color	Set the background color of the widget. You can either leave the the default option as it is or customize it.
Padding	Add extra spaces between the top and bottom of the text within the widget. The default value is 1.
Border	Set the border around the widget.
Shadow	Set a shadow around the widget.
Title	Define a title for the widget.
Title Alignment	Align the title of the widget.
Title Color	Set the color of the widget title.
Subtitle	Define a subtitle for the widget.

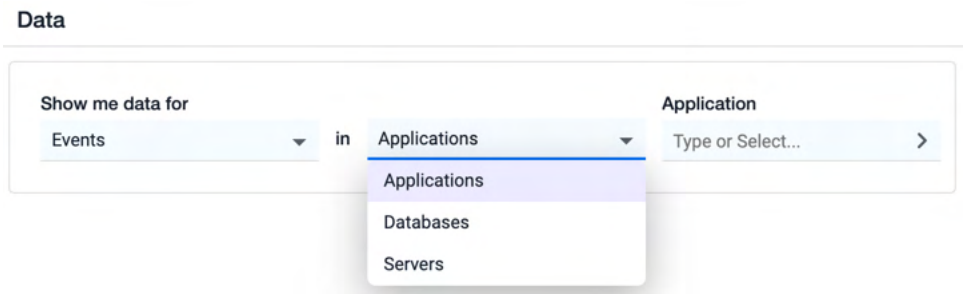
Property	Description
Column sorting	Sort items in the widget columns.
Column resizing	Adjust or resize the width of the widget columns based on the content contained in the viewport of the Datagrid.
Search	Search for objects in the widget.
Grid footer	Hide or display the footer content.
Row height	Adjust the height of the rows.
Filter panel	Display a list of filters.

See widget time range.

View List of Events

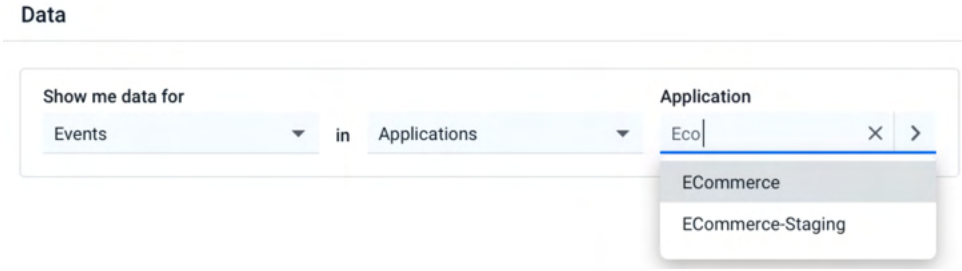
To view a list of events for a given data source, select the Datagrid widget icon from the widget palette:

1. Under **Data** panel, click **Show me data for** and select **Events**.
2. Click the subsequent dropdown and select the data source for which you want to see the list of events. **Applications** is selected in this example.

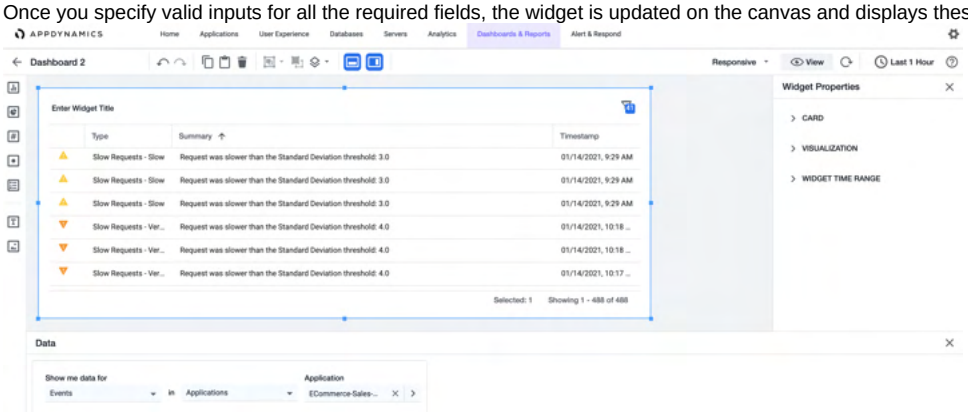


The subsequent inputs shown in the **Data** panel are specific to the data type selected from the previous step.

3. Click **Application** and select one from the list of options. You can also enter the name of the application manually.



Once you specify valid inputs for all the required fields, the widget is updated on the canvas and displays these results:



Click the **Filter Panel** icon on the top-right corner of the Datagrid widget to see the available filters to apply.

To view information such as summary, details, actions executed, and comments about the selected event, double-click the specific row of the widget. Based on your analysis of the event information, you can start a war room, delete the event, archive it, or test the action.

You can also resize the widget in vertical and horizontal directions by dragging the edges.

Label Widget

A Label Widget is a simple text string that enables you to add text on the dashboard canvas.

Widget Properties

You can select these options under the **Widget Properties** panel to modify the appearance of the Label widget:

Property	Description
Background Color	Set the background color of the widget. You can either leave the the default option as it is or customize it.
Padding	Add extra spaces between top and bottom of the text within the widget. The default value is 1.
Border	Set the border around the widget.

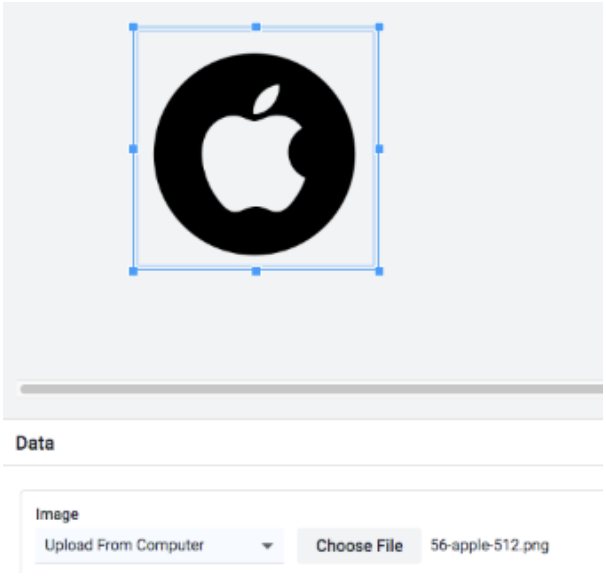
Shadow	Set a shadow around the widget.

Property	Description
Font Color	Specify the color of the text inside the widget.
Font Size	Specify the size of the text within the widget.
Horizontal Alignment	Define horizontal alignment of the text in the widget.
Vertical Alignment	Define vertical alignment of the text in the widget.

See single click action.

Image Widget

You can use the Image Widget to add images to a new dashboard.



Widget Properties

You can select these options under the **Widget Properties** panel to modify the appearance of the Image widget:

Property	Description
Background Color	Set the background color of the widget. You can either leave the the default option as it is or customize it.
Padding	Add extra spaces between top and bottom of the text within the widget. The default value is 1.
Border	Set the border around the widget.
Shadow	Set a shadow around the widget.

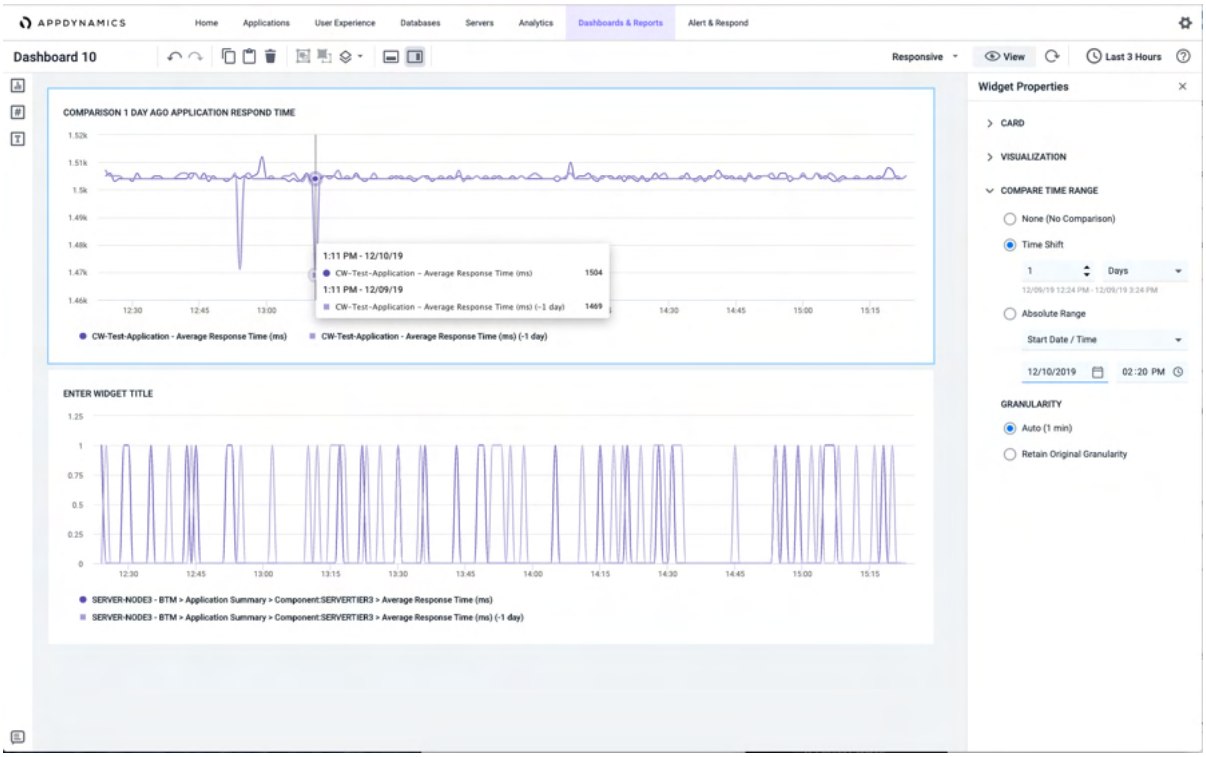
Property	Description
Image Scale	Set the image size. You can select one of the following options: • **Scale to Fit** • **Original Size** • **Stretch**

Time Range Comparisons

This page describes the time range comparison feature in AppDynamics.

You can use the time range comparison in Dash Studio to identify differences in the application performance trends for two time periods. You can align two time periods of the same data and compare. For instance, it might be useful to compare an application's average response time on the previous day with the same day last week.

- Comparisons are currently available only for Time Series widget and Metric Number widget.
- All Data Panel options for Time Series widget and Metric Number widget support comparisons.
- Global time range (for example "Last 4 hours") is selected for the entire dashboard by default.



The default global time range is denoted as T1 and time comparisons are performed using this time range.

Compare Two Time Periods

To compare two time periods on a widget:

1. Select the widget from the widget panel.
2. Expand the **Compare Time Range** section under the **Widget Properties** panel.
3. Set a second time range (denoted as T2) with the same duration as the global time range. You can shift T2 back in time based on the settings in the **Compare Time Range** section.
   You can select one of the following options:
   - **None**: This is the default setting for all the widgets when initially placed on the canvas. Select this option to exclude time comparison.
   - **Time Shift**: This option shifts current widget time range (T1) back by the selected amount of time to become T2. For example, if T1 is "Last 1 hour" and the Time Shift is set to 2 Hours, T2 is 2 hours earlier than T1. When you select this option, you first specify a whole number not less than one as the first input and then select one of the available time options (minutes, hours, days, or weeks) from the dropdown.
   - **Absolute Range**: Use this option to specify the T2 comparison time range by selecting either the **Start Date/Time**, or the **End Date /Time** of the range. For example, if T1 is "Last 1 hour" and you select this option, you can use the date and time inputs to select a start of 8 days ago at 11 am. T2 would thus be 11 am to 12 pm on that specific day. After you set the second or third options, T1 and T2 are superimposed over the same chart space in the Time Series widget.

Granularity

By default, Dash Studio matches granularities if there are differences between T1 and T2 time ranges. AppDynamics stores less recent data at a lower granularity which means fewer data points are displayed on the data series in Dash Studio. To perform an apples-to-apples comparison, Dash Studio is set to **Auto** by default and shows both T1 and T2 time ranges using the same granularity (even if they differ). However, you can change this behavior by selecting the **Retain Original Granularity** option instead of the **Auto** option. When you select **Retain Original Granularity** option, both T1 and T2 show data at their original granularities. This makes comparing behaviors difficult, but the newer data is highly granular.

- Showing the time range comparison doubles the number of data series on the chart. If initially three lines are shown in the widget time range without the comparison, six are shown when the comparison is defined. As new data is received, the widget and its comparisons are updated continuously.
- AppDynamics stores a reduced granularity (fewer data points) for older time ranges.
- Granularity is not available for Metric Number widget.

Time Zone Setting

You can adjust the time zone setting to collaborate and troubleshoot issues with more accuracy. You can align the time zone with other users working in different time zones and eliminate issues such as translating the timestamp of your location to another user's time zone.

To set the time zone:

1. Click the gear icon located at the top right of the page.
2. Select **My Preferences** from the listed options.
   The **My Preferences** page displays an option to select a time zone.
3. Select the **Display Time Zone** dropdown and choose your time zone from the available options. In this example,
4. Click **Reload** to reload the browser and reflect the time zone update.
5. To verify the new time zone setting, open an existing dashboard.

Dashboard Variables

This page provides an overview of dashboard variables in AppDynamics.

You can use dashboard variables to quickly change the data that widgets display without creating a new widget. You can create a series of widgets that show the same metric for a different entity, and view information from multiple applications by simply switching the variable linked to application name. You can also build a dashboard by copying groups of widgets and switching the variable set for each group.

Dash Studio enables you to toggle with ease between variables at the dashboard-level, or group-level. You can define and set variables at various levels of the dashboard, whether for the dashboard at a high level, or for the group at the group level. You can define the variables that you want to set for a group.

Define Variables

You can define variables in a dashboard and across an assembly.

Set Variables

You can set variables to change the values used by a variable across an assembly or within a container. This can occur at the container level and assembly level.

Inherit Variables

Group Variable Inheritance

For group level, variable inheritance is inherited from parents to children. For example, a group inherits its variable definition from the dashboard in which it sits, however the dashboard does not inherit variable definitions from a group. This is because you cannot define variables at the group level.

Assembly Variable Inheritance

For assemblies, the sibling assemblies inherit variable definition from their siblings. For example, an assembly inherits its variable definition from the assembly from which it was made.

Data Binding

You can use the **Data** panel to query and display data for a selected widget. Additionally, you can:

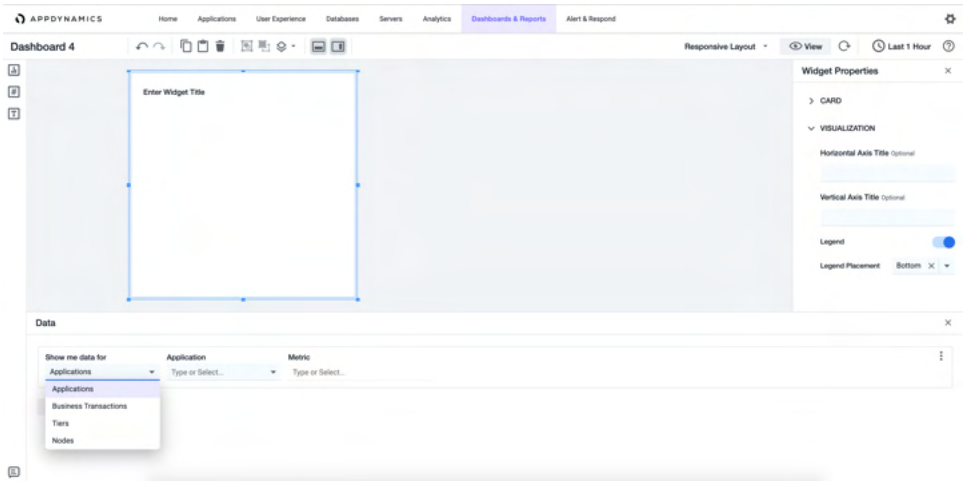
- Construct queries that span your environment.
- Add important attributes and relationships to your queries.
- Quickly access data when you do not know the metric path.
- Construct sophisticated queries.
- Query both metric and analytics data in the same query.

Data Queries

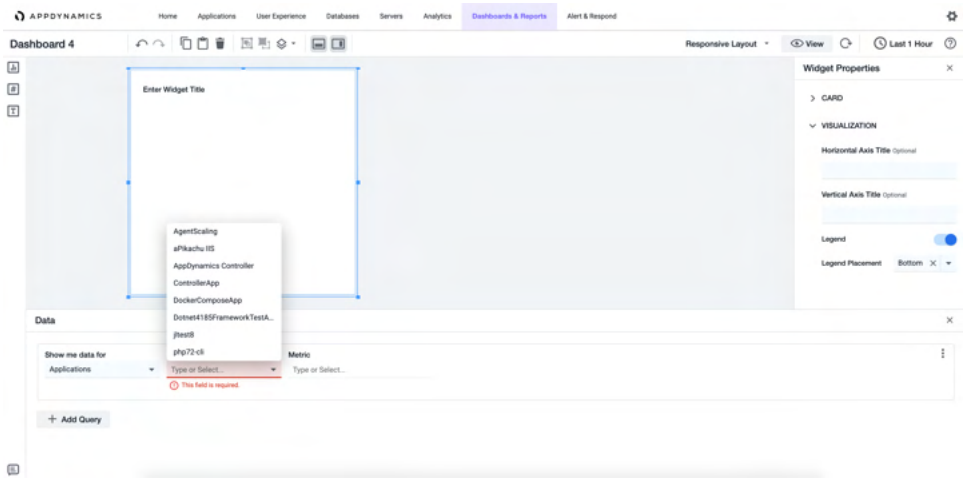
The **Data** panel is structured around queries. Each query follows a sentence-like format, starting with a selection of the data type of the query. Many fields in the **Data** panel support typeahead functionality.

In this example, you want to know errors per minute for the nodes in an application.

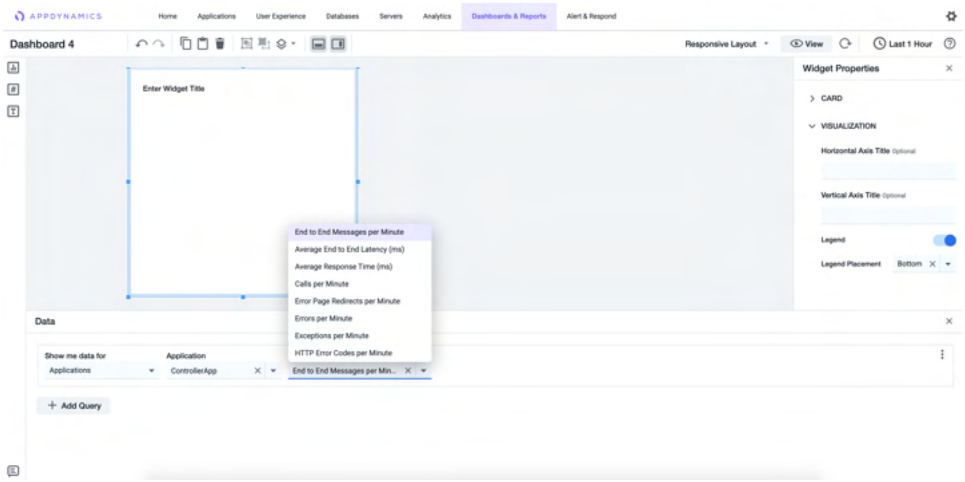
1. Select a value from the **Show me data for** dropdown based on the data type you wish to display.



   In this example, select **Applications**.
2. The subsequent inputs shown in the **Data** panel are specific to the data type you selected in the previous step. In this example, use the **Applicatio** dropdown to select the application for which you want to drill down the data.



3. The next dropdown is **Metric**. The **Metric** dropdown changes based on the data type. For example, the node displays two more inputs before you select the metric.
4. Select the metric you wish to display on the widget. You can select the **More** icon on the right-corner of the **Data Panel** to set the limit and baseline.



After you have specified valid inputs for all fields in the query, the widget is updated on the canvas and shows the query result.

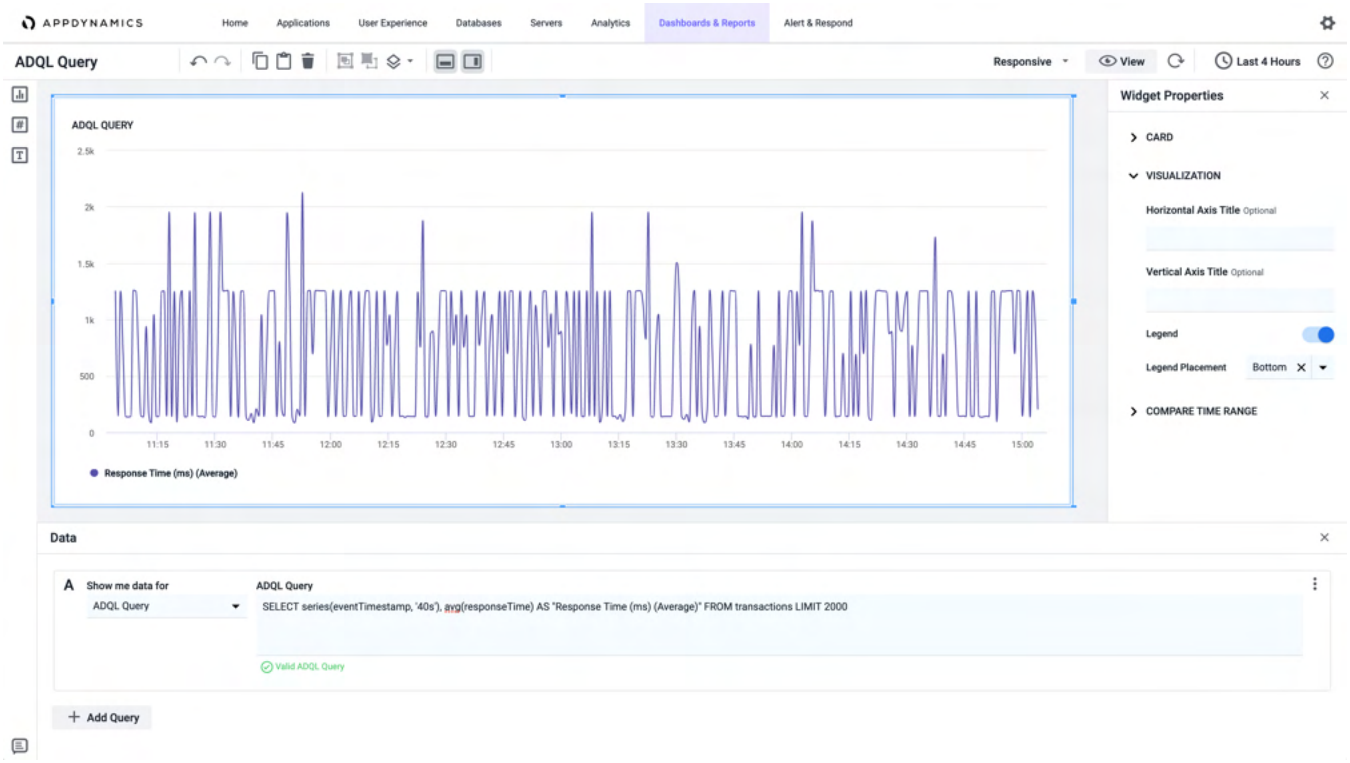
ADQL Queries

ADQL (AppDynamics Data Query Language) Query is a metrics query available in the **Data** panel for dashboards created in Dash Studio. Use this query to filter and create customized metrics for Analytics data.

ADQL Query data binding uses Analytics queries written in ADQL. After selecting the **ADQL Query** option from **Show me data for** dropdown, you can either type or paste an ADQL query. See ADQL Reference for the syntax of these query statements.

> To view data reflected in the Time Series widget in Dash Studio, you must enter a valid ADQL Query for series. You must include the `series` expression in the ADQL query statement otherwise it does not display a series. ADQL Query is available for the **Time Series** widget only.

After you type or paste the query in the input field, moving the cursor away from the input field automatically validates the query. For a **Valid ADQL Query** (displayed in green), the data displays in the widget on the canvas. For any invalid query, an error message displays in red.

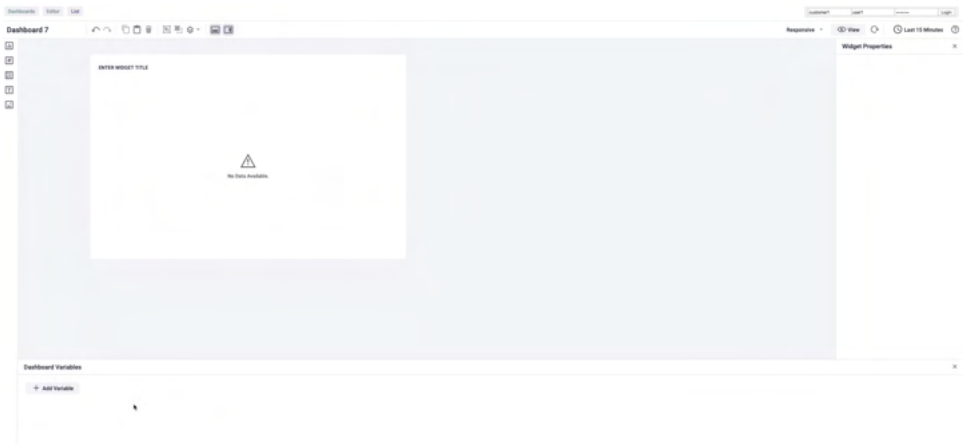


Variables in ADQL Queries

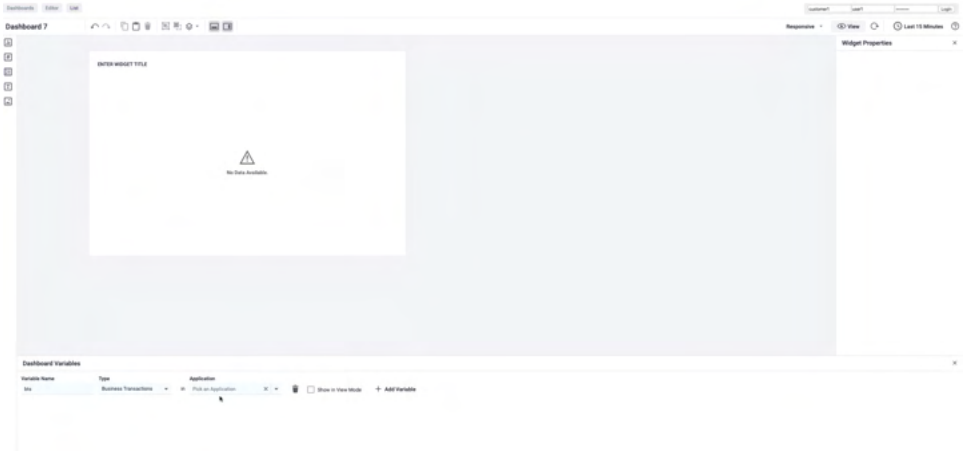
Variables in ADQL queries enable you to dynamically change the values passed into the queries.

For example, you want to use a variable for an application within an ADQL query.

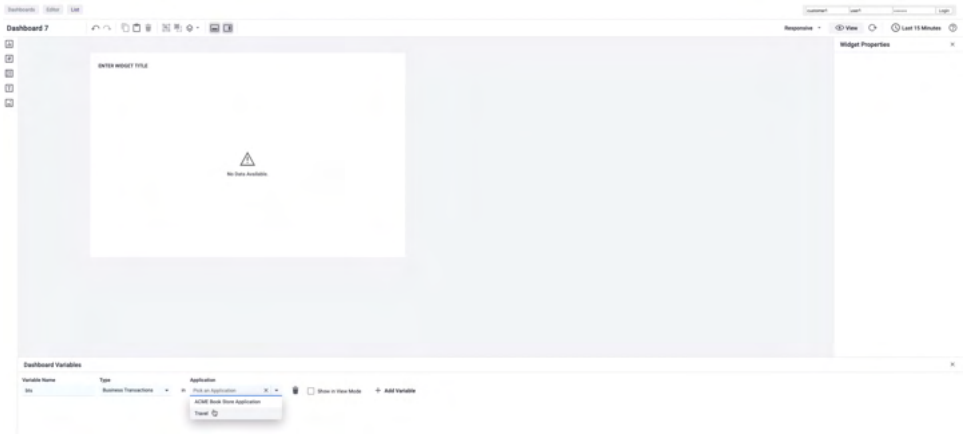
1. Click **+ Add Variable** under the **Dashboard Variables** panel.



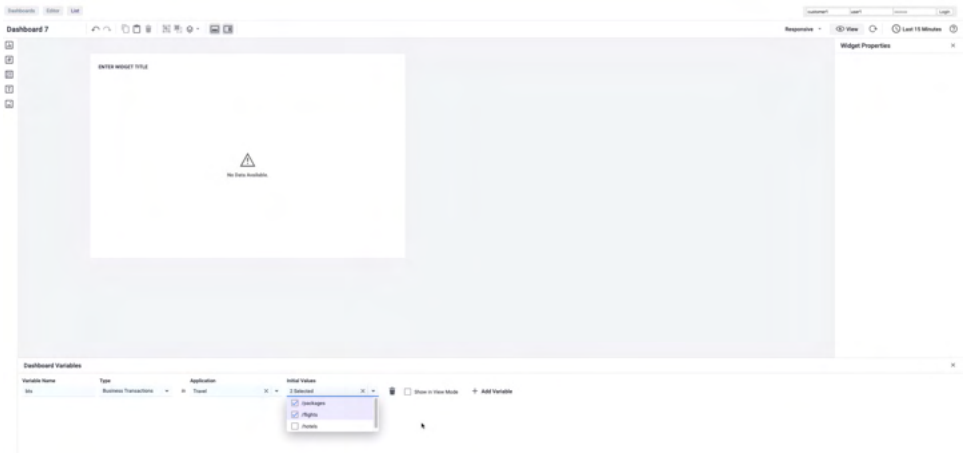
2. Enter a value in the **Variable Name** field and select the variable type in the **Type** dropdown.



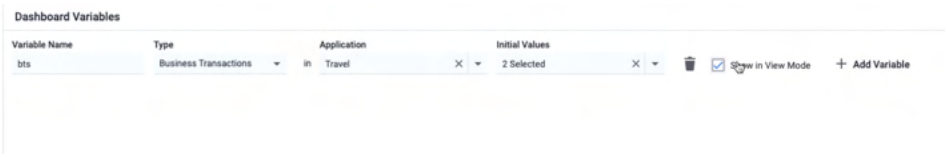
3. Select an application in the **Application** dropdown.



The subsequent inputs shown in the **Dashboard Variables** panel are specific to the data type selected in the previous step. In this example, it is the **Initial Values** dropdown, which enables you to select the business transaction for which you want to drill down the data.

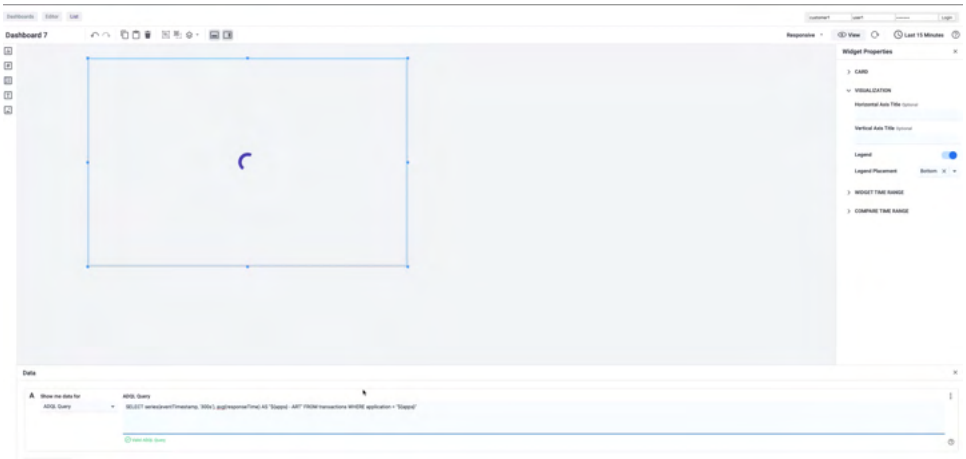


4. Select the **Initial Values**, and then select the **Show in View Mode** checkbox.

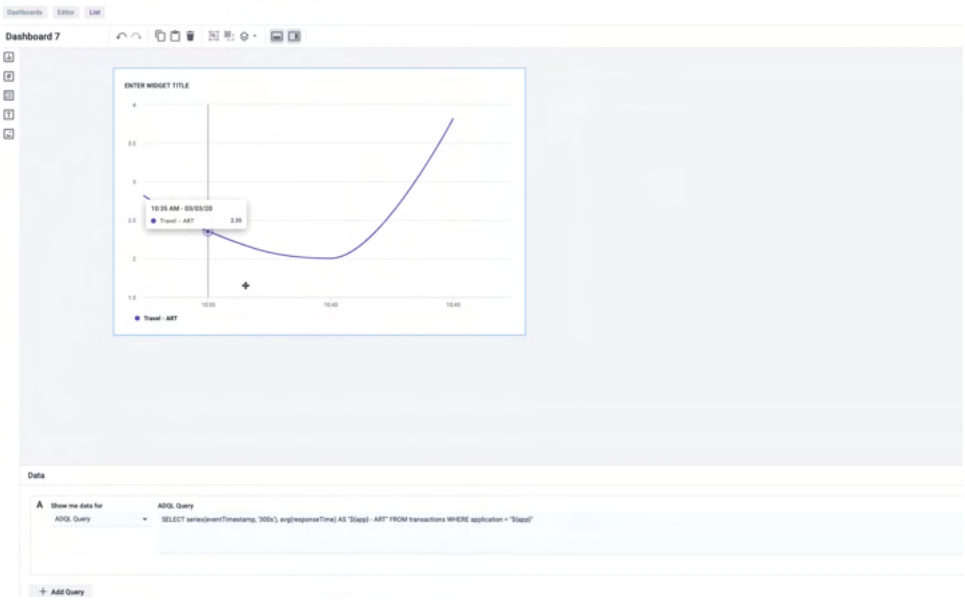


Click the widget to change **Data Panel** from variable data to widget data.

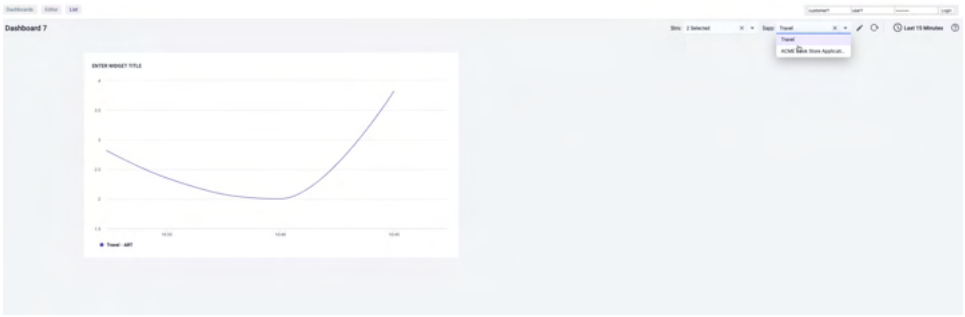
5. Choose a value from the **Show me data for** dropdown.
   In this example, select **ADQL Query**, and then enter the query in the **ADQL Query** field:



After you enter a valid query, the widget displays:



6. Click **View** on the top-right of the page. The subsequent dashboard page enables you to change the application by clicking the $**app** dropdown.

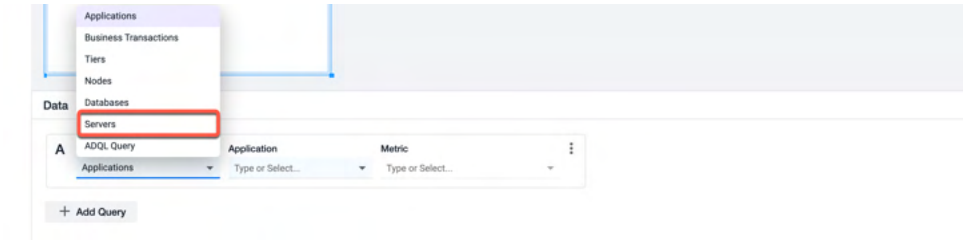


Server Metrics

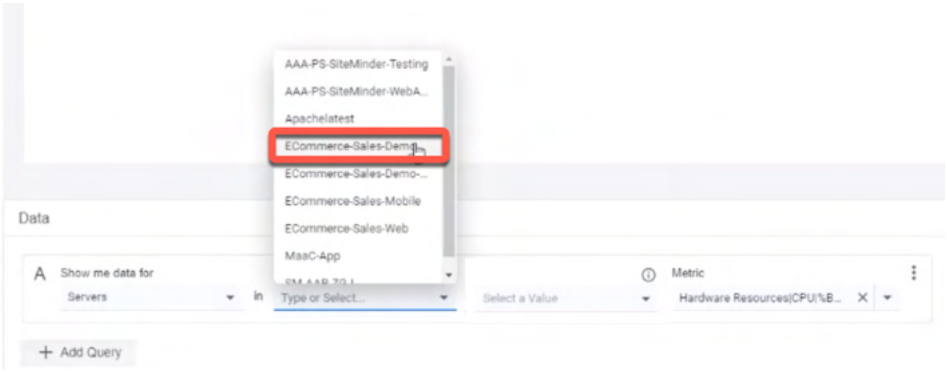
Server Metrics [also known Server Infrastructure Monitoring (SIM) Metrics] is a data binding option available in the **Data** panel of Dash Studio. Use this option to filter and create customized metrics for server monitoring.

To filter and create customized metrics for one or more servers:

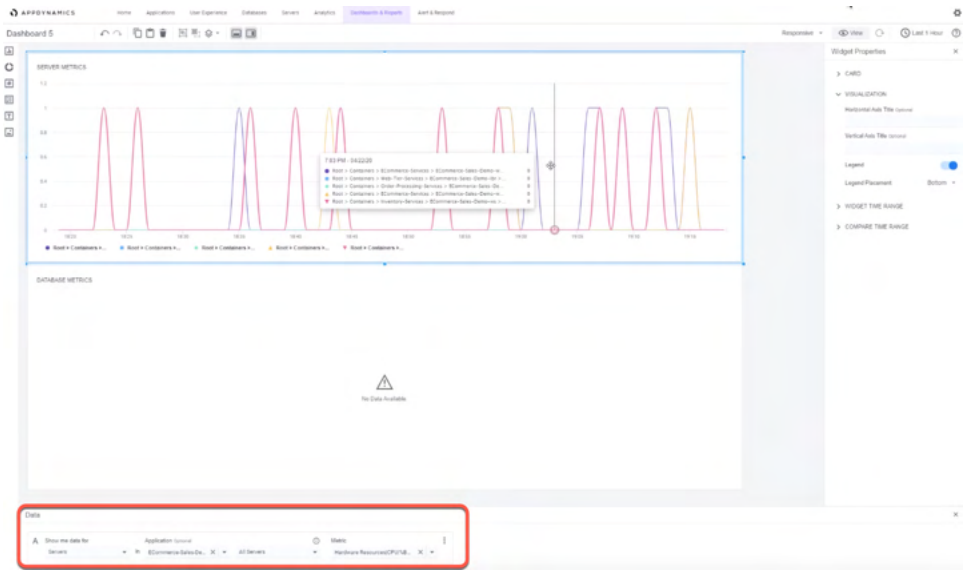
1. Click the **Show me data for** dropdown and select **Servers**.



2. Click the **Application** dropdown and select the application for which you want to drill down the data.

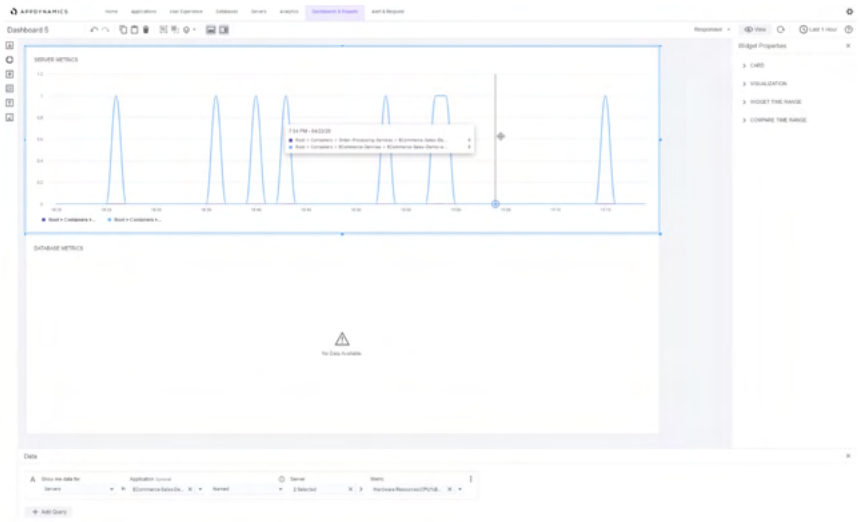


3. Click **Select a Value** and select one of these options:
    - **All Servers:** Select this option to show data across all applications. If you select this option, then the **Application** selected in the previous step becomes optional. If you select **All Servers,** the next input field in the **Data** panel automatically changes to **Metric.** Select the metric value you wish to display on the widget from the dropdown, or by manually entering a value in the input field.



> If none of the applications are selected, then the widget displays data for stand-alone applications only, and not for the applications included in the application tree.

    - **Named:** Select this option to specify individual servers known to the specific application that you chose in step 2. If you select **Named,** the next input field in the **Data** panel changes to **Server**. Select one or more servers from the **Server** dropdown, and the metric value from the **Metric** dropdown.



- **Within:** Select this option to select the tier, or tiers, of the server you wish to display in the server chart.

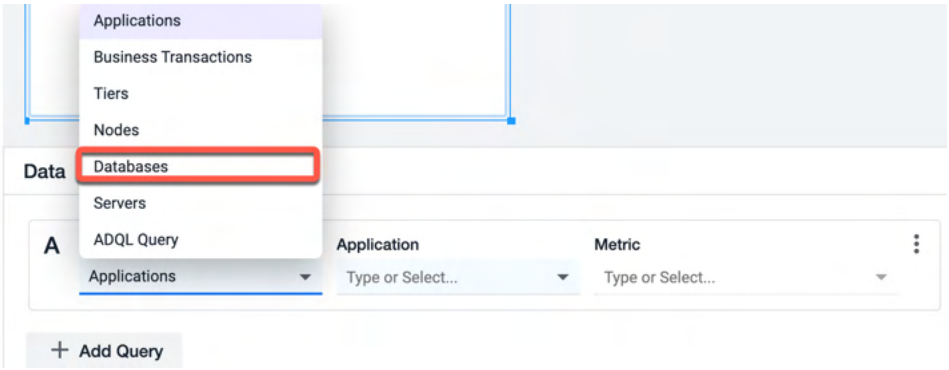
Database Metrics

Database Metrics is a data binding option available in the **Data** panel of Dash Studio. Use this option to filter and create customized metrics for database monitoring.

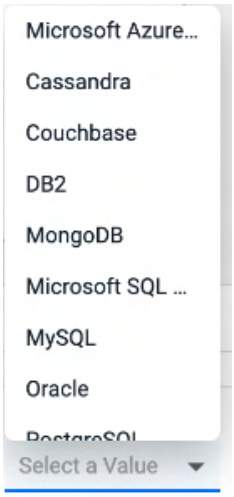
> The Metric Number widget does not support Database metrics.

For example, to monitor a specific database or multiple databases:

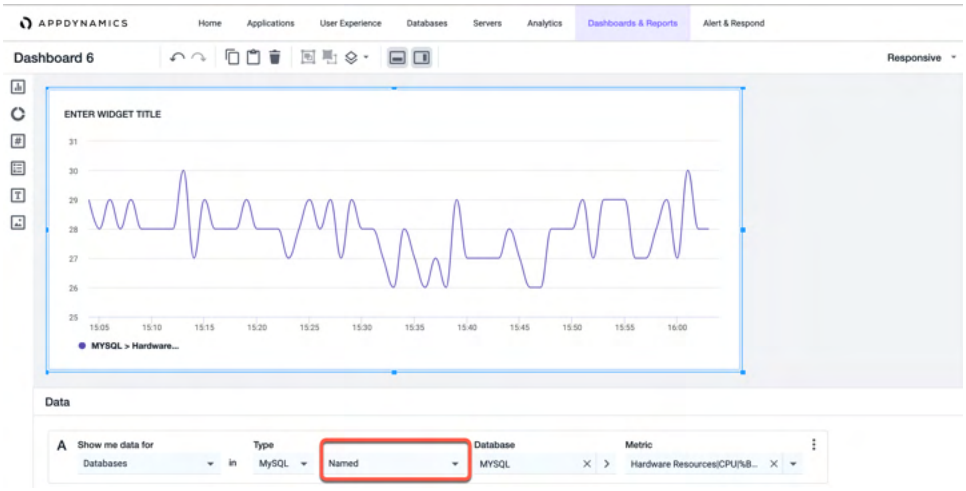
1. Click the **Show me data for** dropdown, and then select **Databases**.



2. Click **Type** and select the required database from the dropdown to display the metrics on the widget.



3. Click **Select a Value** and choose one of these options:
    - **All Databases:** Displays data for all the databases.
    - **Named:** Displays data for a specific database. You can select the required database from the subsequent input field **Database** that displays when you select **Named**.
4. Click **Metric** and select the metric value, or enter a value manually. The database metric chart displays.

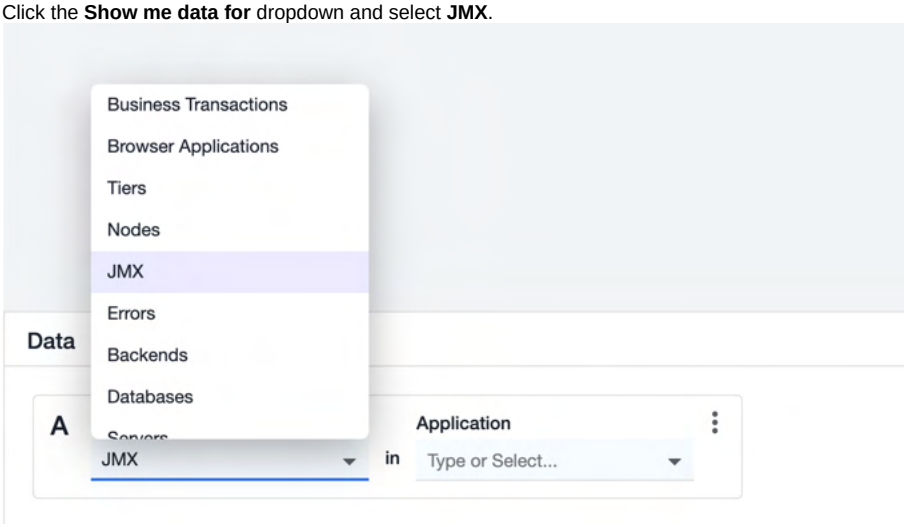


JMX

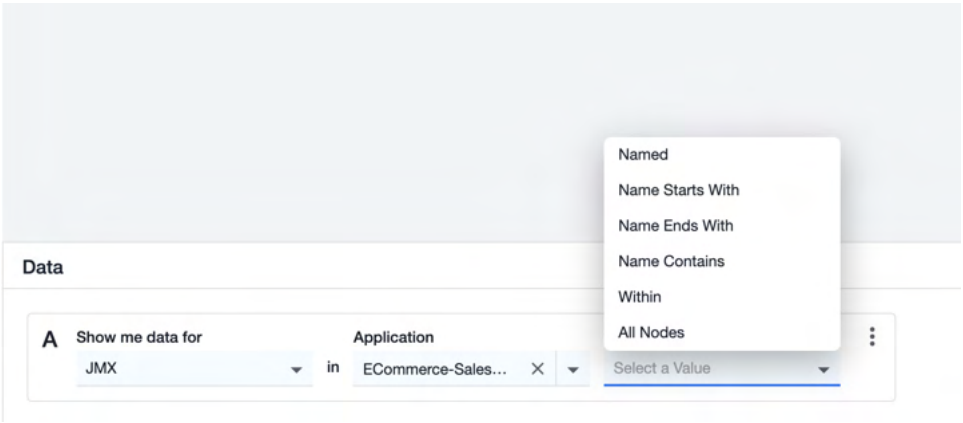
JMX (Java Management Extensions) is a data binding option available in the **Data** panel for dashboards created in Dash Studio. Use this option to filter the entities by nodes and monitor the JMX metrics for the given set of nodes.

For example, to monitor JMX metrics of a node or multiple nodes:

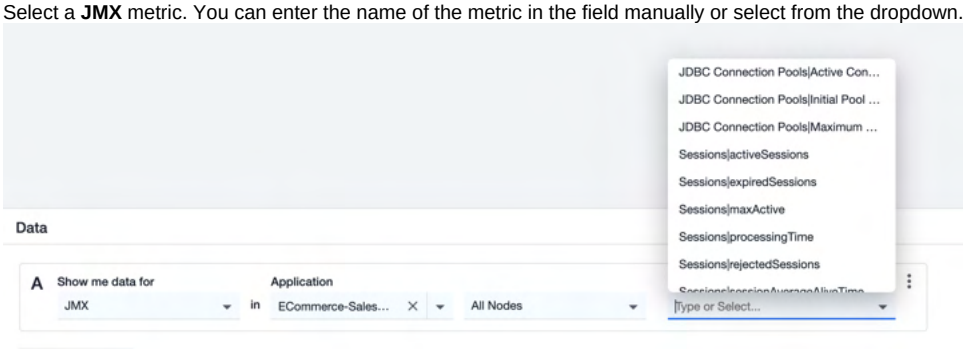
1. Click the **Show me data for** dropdown and select **JMX**.



   The subsequent inputs shown in the **Data** panel are specific to the data type selected in this step.
2. Click the **Application** dropdown and select the application for which you want to drill down the data.
3. Click **Select a Value** and select a node filter from the dropdown.



4. Select a **JMX** metric. You can enter the name of the metric in the field manually or select from the dropdown. This list is dynamic.



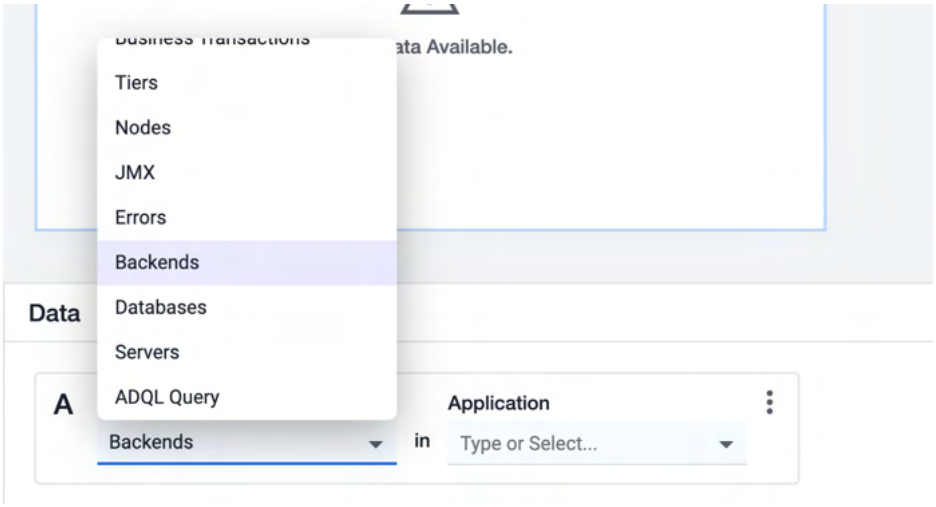
- Entries do not display if the selected entry is not a Java application, or if there are no JMX metrics for the selected application.
- The metric selector parses the Java Application metric tree dynamically for all available JMX metrics and displays them as a flattened list.
- Currently, only the JMX metrics that are 2 levels deep from a JMX folder in the metric tree are available in Dash Studio.

Backends

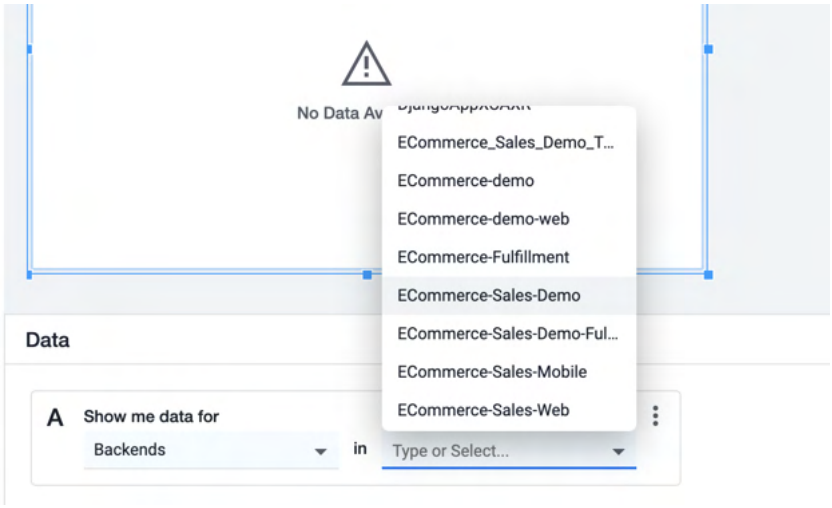
Backends is a data binding option available in the **Data** panel for dashboards created in Dash Studio. Use this option to filter the backend entities.

To monitor **Backends**:

1. Click the **Show me data for** the dropdown and select **Backends**.

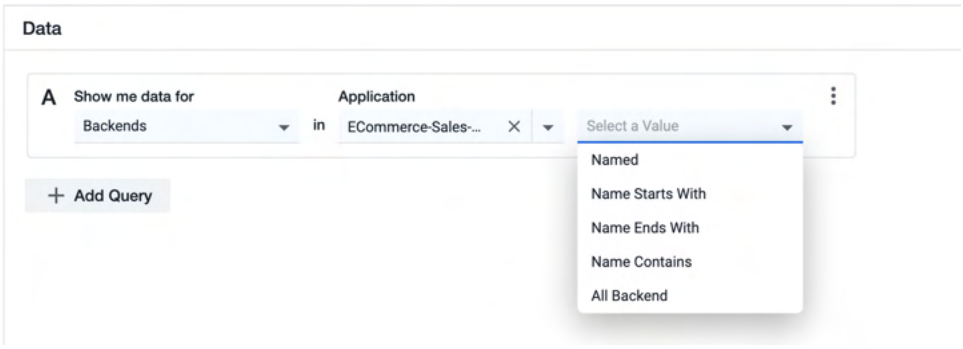


2. Click **Type or Select** and then select the application to display the metrics on the widget.

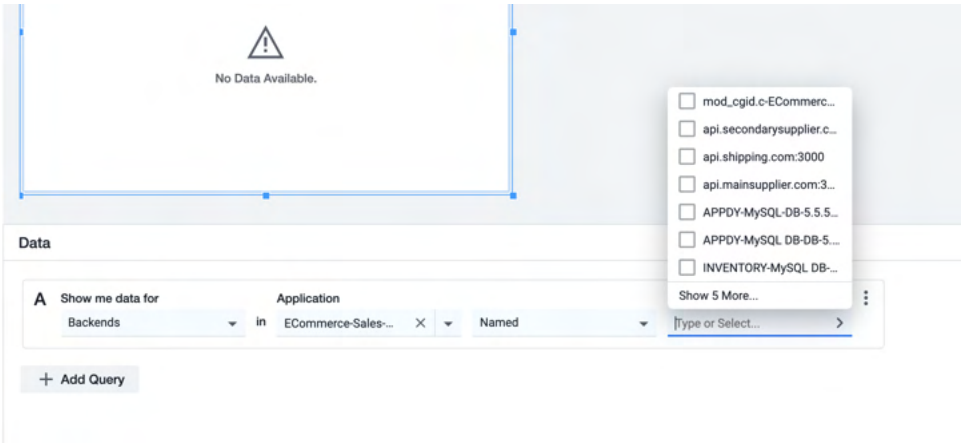


3. Click **Select a Value** and then select the filter you want to apply:

    - Named
    - Name Starts with
    - Name Ends With
    - Name Contains
    - All Backend



4. Click **Type or Select** and then select a named filter from the list of options.



5. Click **Metric** and select the backend metric value.

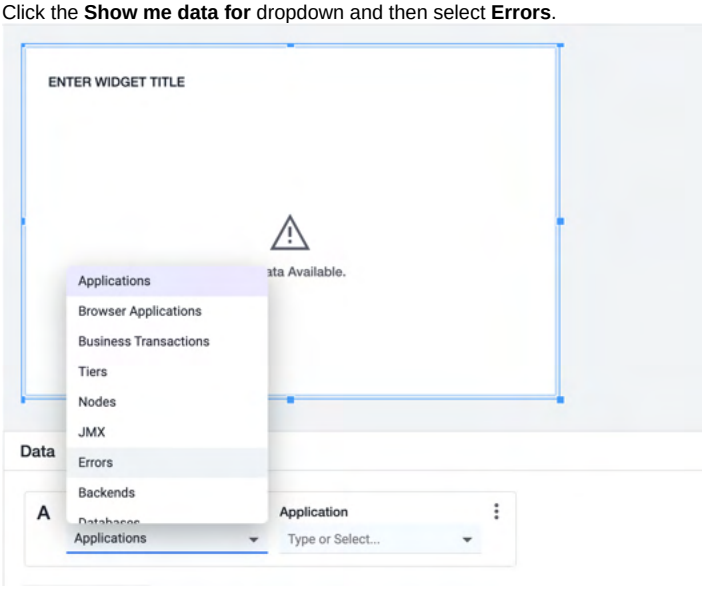
The metric chart displays:

Errors Data Binding

Errors is a data binding option available in the **Data** panel for dashboards created in Dash Studio. Use this option to filter the error entities.

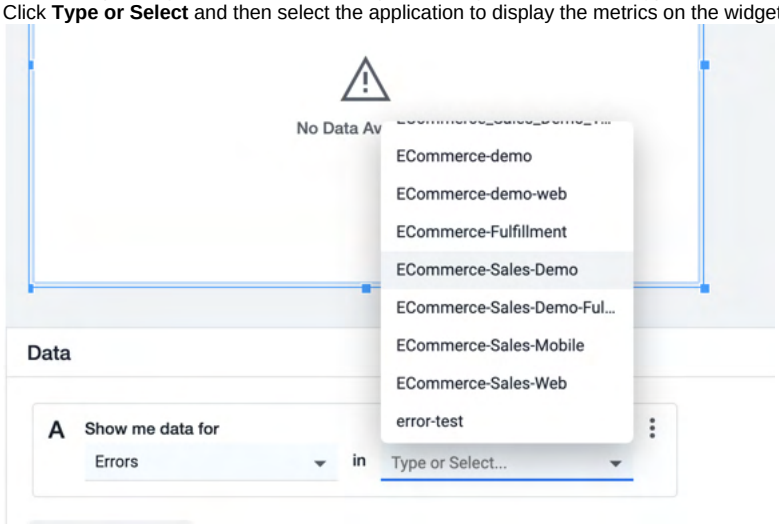
To monitor error entities:

1. Click the **Show me data for** dropdown and then select **Errors**.

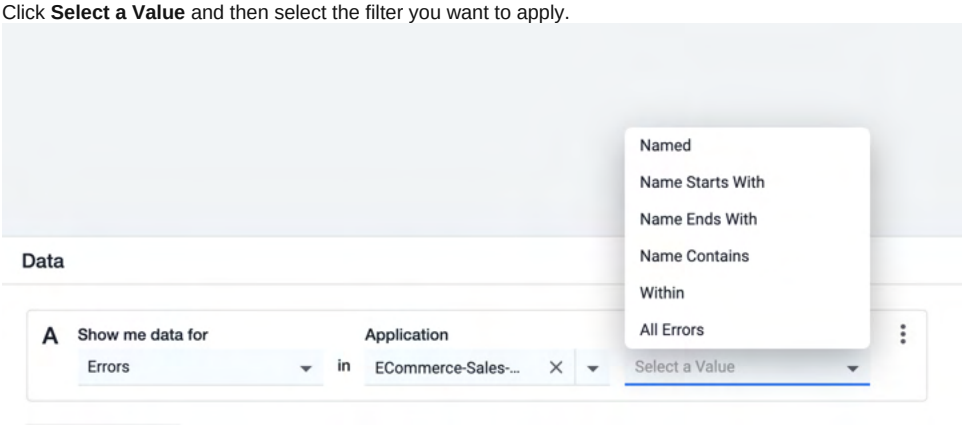


The subsequent inputs shown in the **Data** panel are specific to the data type selected in this step.

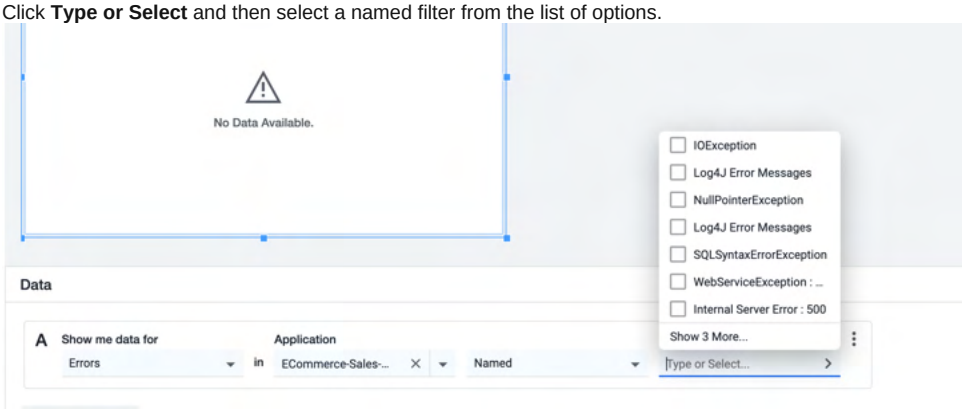
2. Click **Type or Select** and then select the application to display the metrics on the widget.



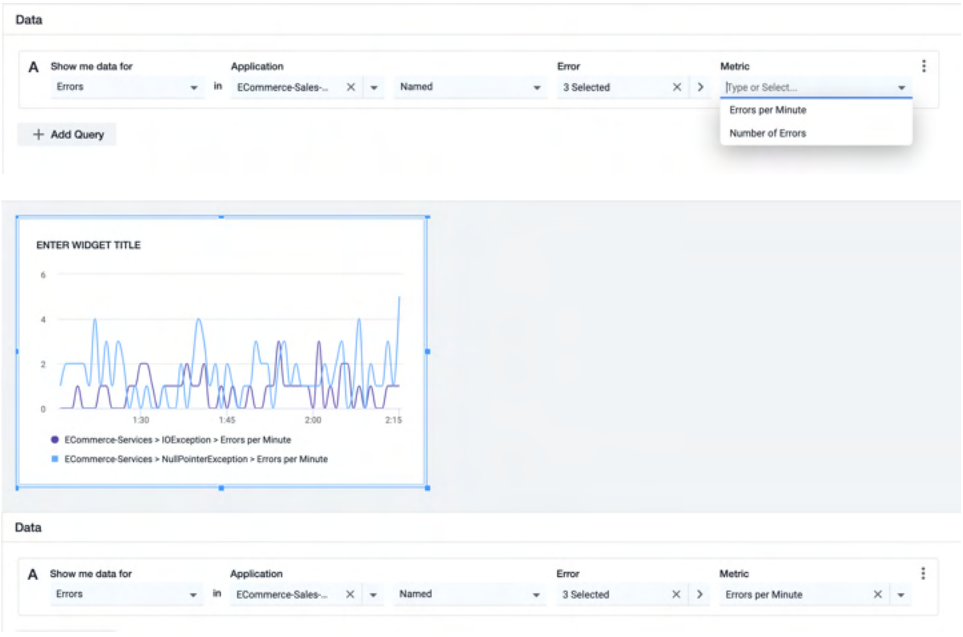
3. Click **Select a Value** and then select the filter you want to apply.



4. Click **Type or Select** and then select a named filter from the list of options.



5. Click **Metric** and select the metric value.



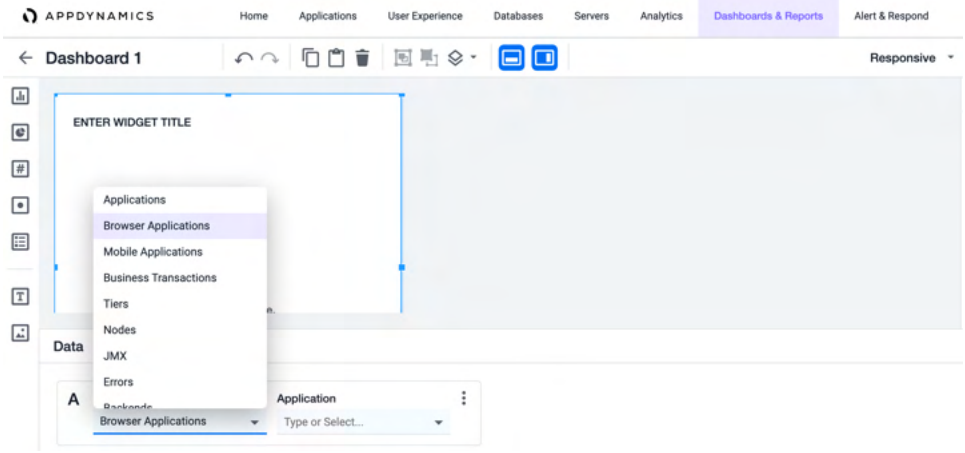
A chart that shows data for All Errors displays:

Browser Applications Metrics

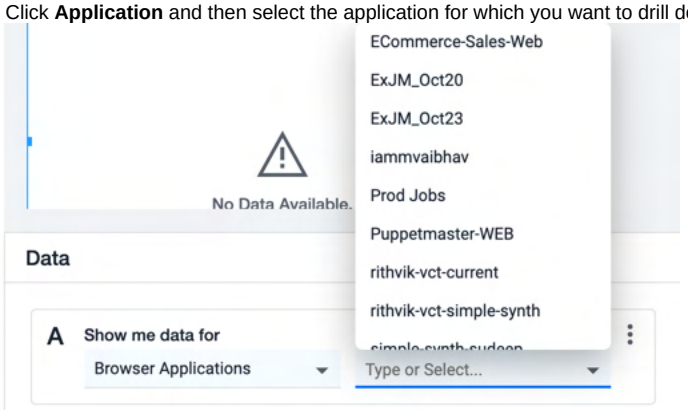
Browser Applications metrics [also known as the Browser Real-User Monitoring (BRUM) metrics] is a data binding option available in the **Data** panel of Dash Studio. Use this option to filter and create customized metrics to view the performance of the web applications.

To filter and create customized metrics for one or more browser applications:

1. Under **Data** panel, click the **Show me data for** dropdown and then select **Browser Applications**.

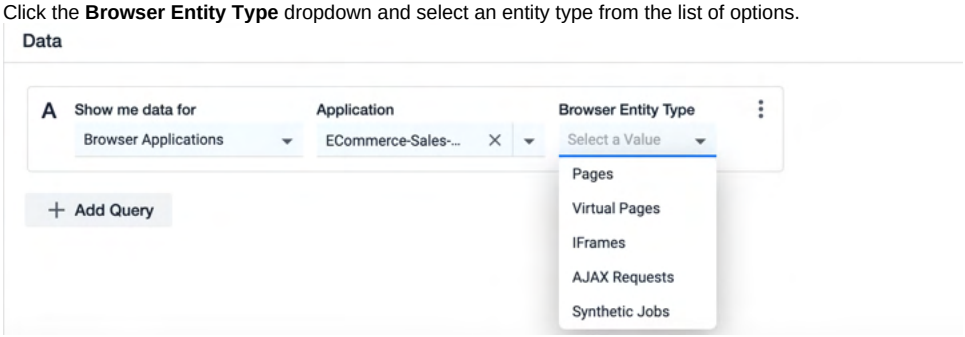


2. Click **Application** and then select the application for which you want to drill down the data.

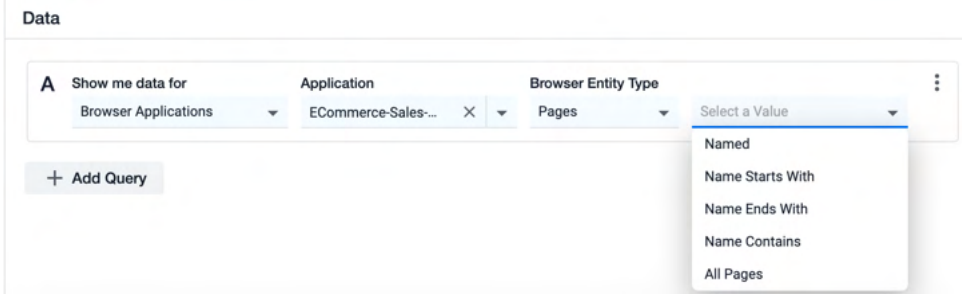


> The subsequent inputs shown in the **Data** panel are specific to the data type selected in the previous step.

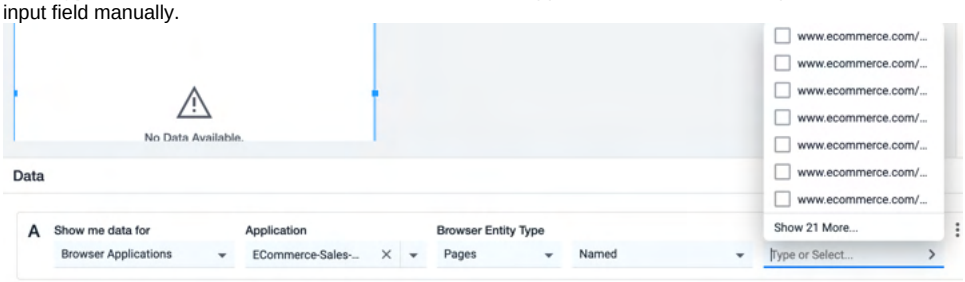
3. Click the **Browser Entity Type** dropdown and select an entity type from the list of options.



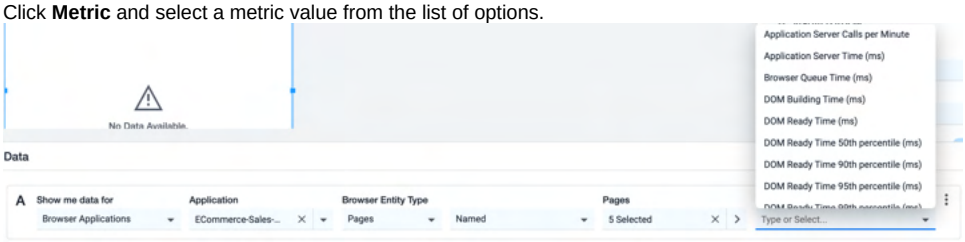
4. Click **Select a Value** and select the filter that you want to apply.



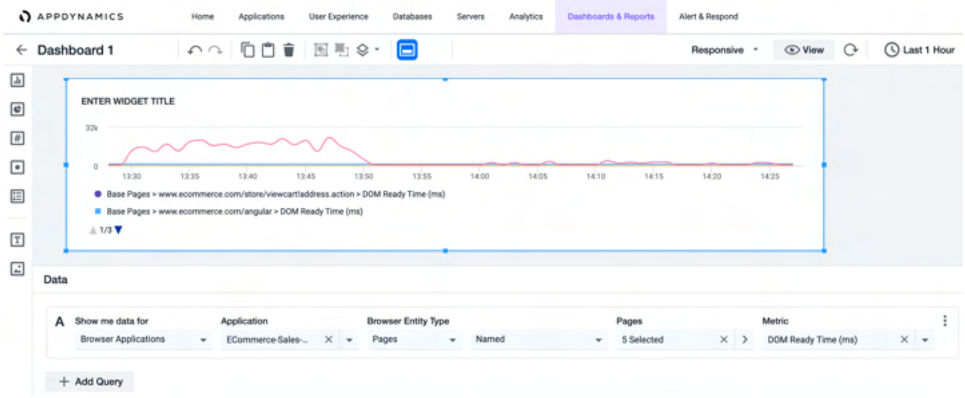
5. Click **Type or Select** and then select one or more browser applications from the list of options, or enter the name of the browser application in the input field manually.



6. Click **Metric** and select a metric value from the list of options.



The Browser Applications metric chart displays:

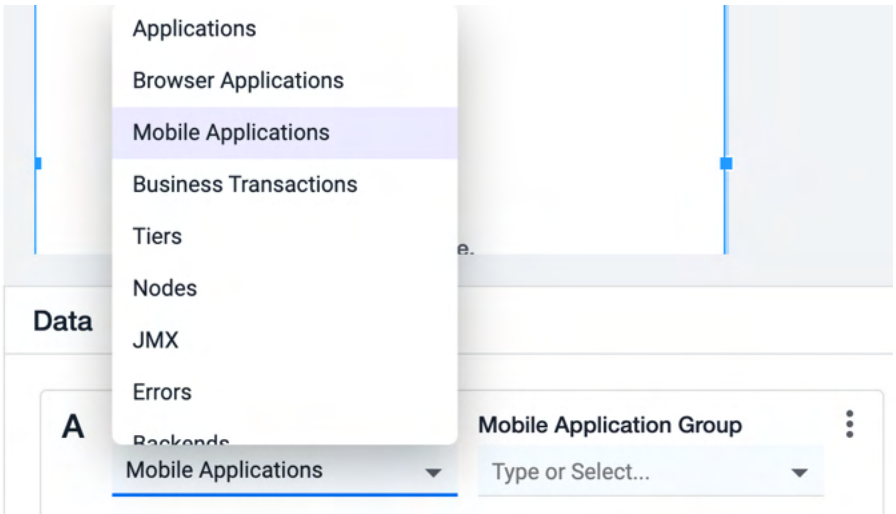


Mobile Applications Metrics

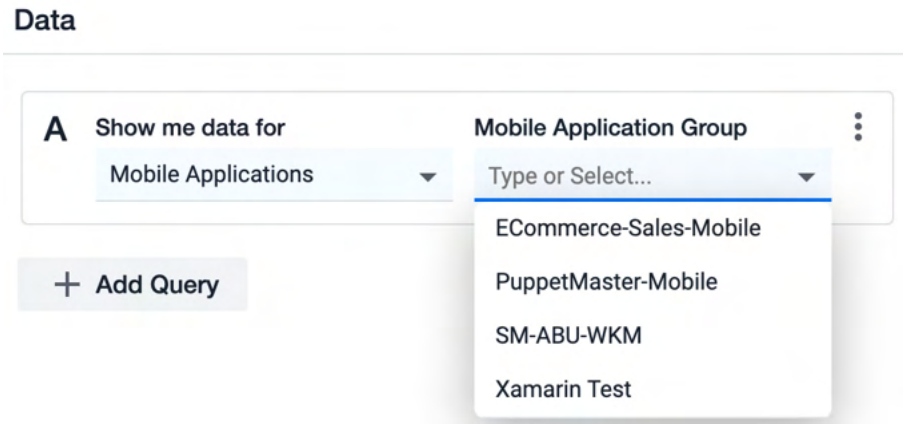
Mobile Applications metrics [also known as the Mobile Real-User Monitoring (MRUM) metrics] is a data binding option available in the **Data** panel of Dash Studio. Use this option to filter and create customized metrics to view the performance of mobile applications.

To filter and create customized metrics for one or more mobile applications:

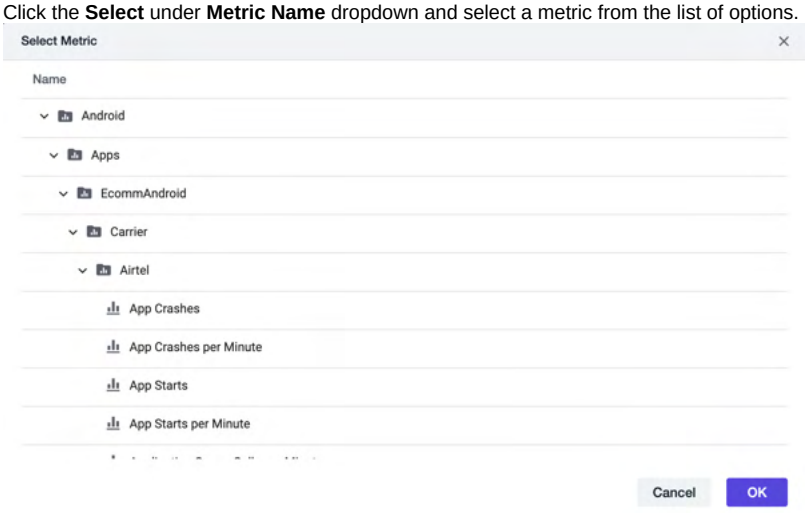
1. Under **Data** panel, click the **Show me data for** dropdown and select **Mobile Applications**.



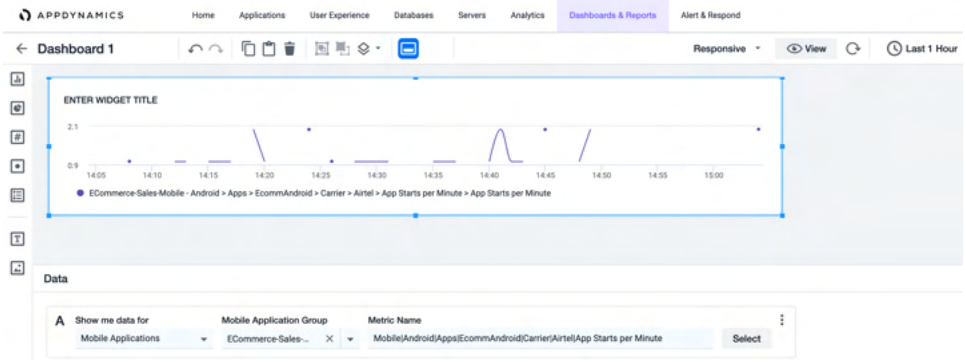
2. Click **Mobile Application Group** and then select the application for which you want to drill down the data.



3. Click the **Select** under **Metric Name** dropdown and select a metric from the list of options.



4. Click **OK**. The Mobile Applications metric chart displays:



ThousandEyes Integration with AppDynamics

ThousandEyes is a network intelligence SaaS platform that provides deep insights about the network and external dependencies. AppDynamics and ThousandEyes integration delivers a full stack solution you can use to correlate your network to the application performance. You can identify network related performance issues to reduce the impact on critical applications.

You can visualize ThousandEyes data along with application data on the dashboard created in Dash Studio.

- Analyze performance bottlenecks in your network at a glance
- Drill down to the root cause of performance issues
- Monitor key metrics
- Reduce Mean Time to Resolution (MTTR)

To configure the ThousandEyes token and to enable ThousandEyes query see:

- Enable the ThousandEyes Token
- Configure the ThousandEyes Dashboard

Configure the ThousandEyes Token

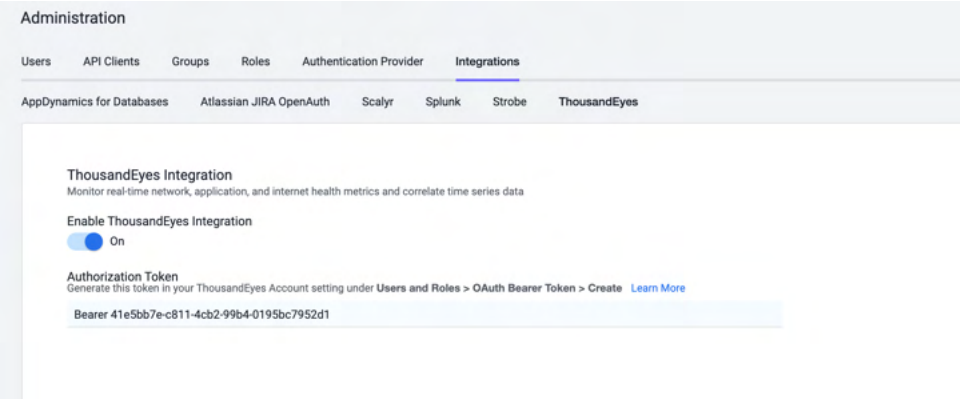
To enable the integration for a controller version lower than 21.5.0, you need to raise a ticket at help.appdynamics.com.

Before you can access the ThousandEyes network data, you must configure the ThousandEyes token in your ThousandEyes Account. See account settings.

To access the ThousandEyes data, configure the ThousandEyes token by performing the following steps:

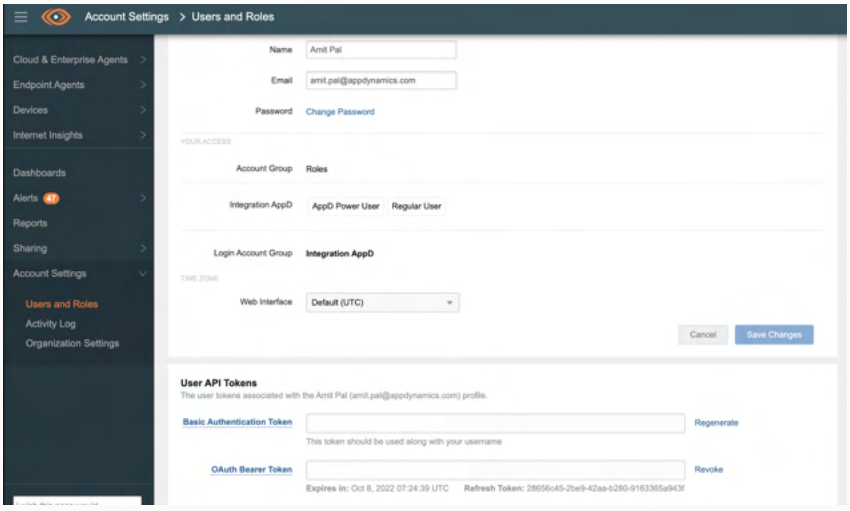
You must have ApDynamics admin privileges access to this page. To request access contact the AppDynamics support.

1. Click the gear icon at the top-right corner of the Home page and select **Administration**.
   The **Administration** page displays.
2. Click the **Integrations** tab.
3. Click the **ThousandEyes** tab. The **ThousandEyes Integration** page displays.
4. Click the **Enable ThousandEyes Integration** toggle.
5. Enter the token value in the **Authorization Token** field manually or you can copy and paste it.



The token must be in the format `Bearer<space><token>.`

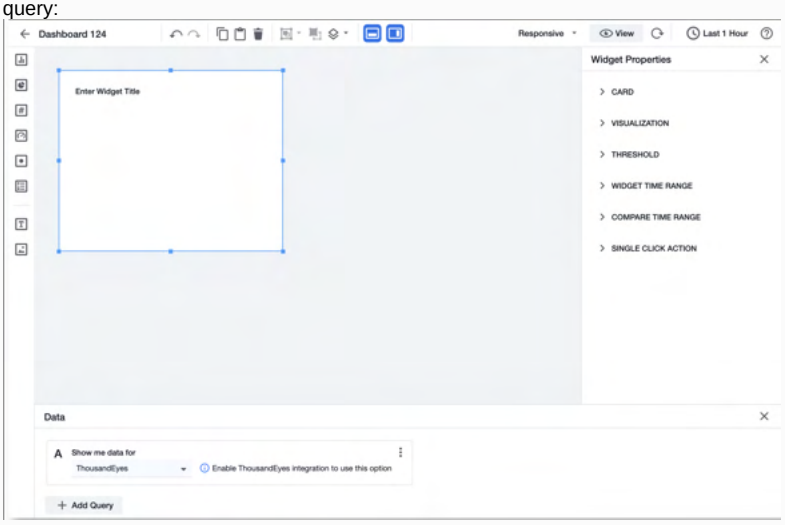
You can retrieve the authorization token by logging in to ThousandEyes Account.



6. Click **Save**.

On disabling the ThousandEyes integration on the **Administration** page:

- If you select ThousandEyes query in **Show me data for** dropdown list, it displays a warning message requesting you to enable the query:



- The existing widgets that use ThousandEyes query do not render ThousandEyes data.

Configure the ThousandEyes Dashboard

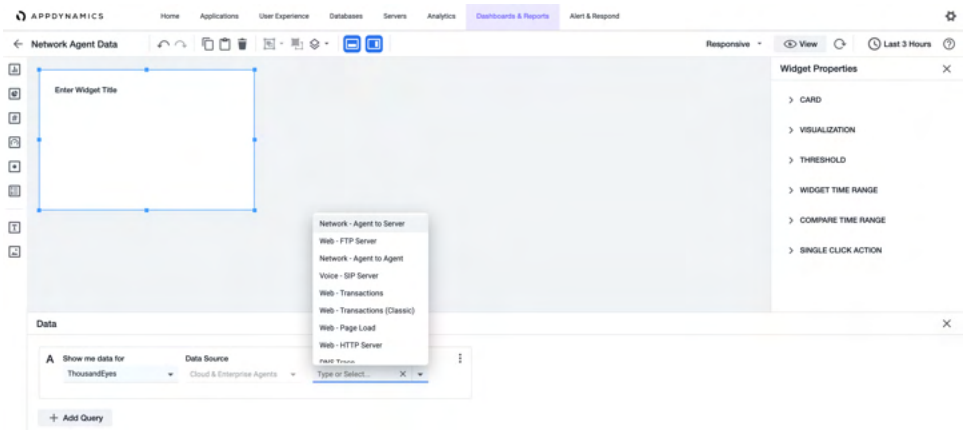
The ThousandEyes query is a metrics query available in the **Data** panel for dashboards created in Dash Studio. Enable this query to troubleshoot network performance issues related to latency, packet loss, and bandwidth utilization. See ThousandEyes platform.

To filter and create customized metrics to analyze the performance of your network:

1. Click **Dashboards and Reports** and select the **Dash Studio (Preview)** tab.
2. Click + to create a new dashboard or you can open an existing dashboard.
3. Select a widget from the widget panel.

> Only the Time Series and Metric Number widget support the ThousandEyes query.

4. Click the **Show me data for** dropdown in the **Data** panel.
5. Select **ThousandEyes** from the dropdown. You can add only one ThousandEyes query per widget. **Data Source** field displays with the **Cloud & Enterprise Agents** option set by default.



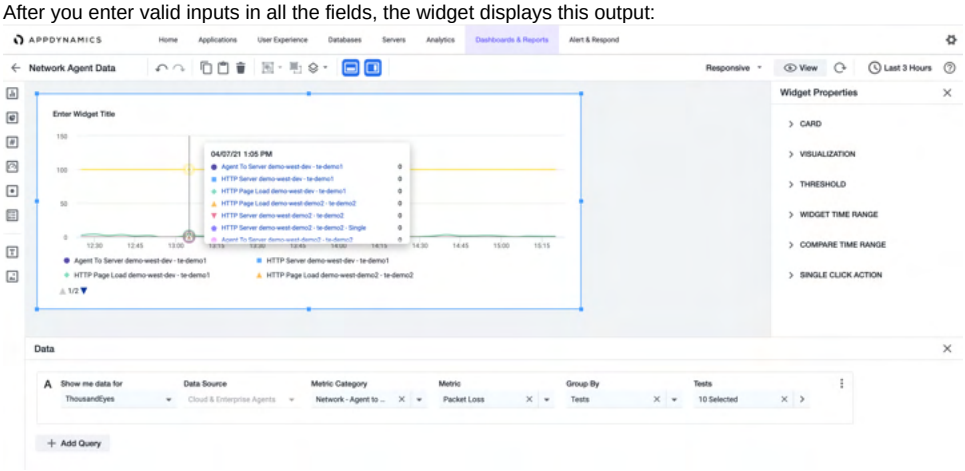
> - The Data Source option allows you to choose an agent type to gain insights about network and application performance. See agent types in ThousandEyes.
> - Dash Studio supports only one agent type, Cloud & Enterprise Agents.

6. Click **Metric Category** and select a category from the dropdown list.
7. Click **Metric** and select a metric value from the dropdown list.
8. Click **Group By** to select how to group the selected metric value. You can group the metrics using one of these options:
    - **Agent Labels** - Logical grouping of agents.
    - **Agents** - Agents installed at various locations.
    - **Tests** - Network assets to be monitored. See tests.
    - **Test Labels** - Logical grouping of multiple tests.

> The **Group By** option is available only for Time Series widget.

9. Click **Tests.**

After you enter valid inputs in all the fields, the widget displays this output:



On completing successful ThousandEyes binding, the **Powered by ThousandEyes** icon displays for Time Series widget.

- ThousandEyes test selection limit is ten per widget.
- Maximum supported time range is eight days, so you can view ThousandEyes data available over the past 8 days only.
- Time Range Comparison is not available for ThousandEyes data.

Application Monitoring

AppDynamics Application Performance Monitoring (APM), a component of the AppDynamics platform, provides end-to-end visibility into the performance of your applications.

AppDynamics works with popular programming languages such as Java, .NET, Node.js, PHP, Python, C/C++, and more, enabling you to:

- Troubleshoot problems such as slow response times and application errors.
- Automatically discover application topology and how components in the application environment work together to fulfill key business transactions for its users.
- Measure end-to-end business transaction performance, along with the health of individual application and infrastructure nodes.
- Receive alerts based on custom or built-in health rules, including rules against dynamic performance baselines that alert you to issues in the context of business transactions.
- Analyze your applications at the code execution level using snapshots.

Agent Installation

- Install the Java Agent
- Install the .NET Agent for Windows
- Install the Node.js Agent
- Install the Python Agent

Configuration

- Transaction Detection Rules
- Backend Detection Rules
- Error Detection
- Data Collectors

Monitoring Applications

- Business Transactions
- Transaction Snapshots
- Tiers and Nodes
- Troubleshooting Applications

Concepts

- Overview of Application Monitoring
- Organize Business Transactions
- Flow Maps
- Remote Services

Overview of Application Monitoring

Related pages:

- Flow Maps
- Metrics and Graphs

After you understand the basics of AppDynamics, you can learn how AppDynamics models application environments. The model serves as the framework around which AppDynamics organizes and presents performance information.

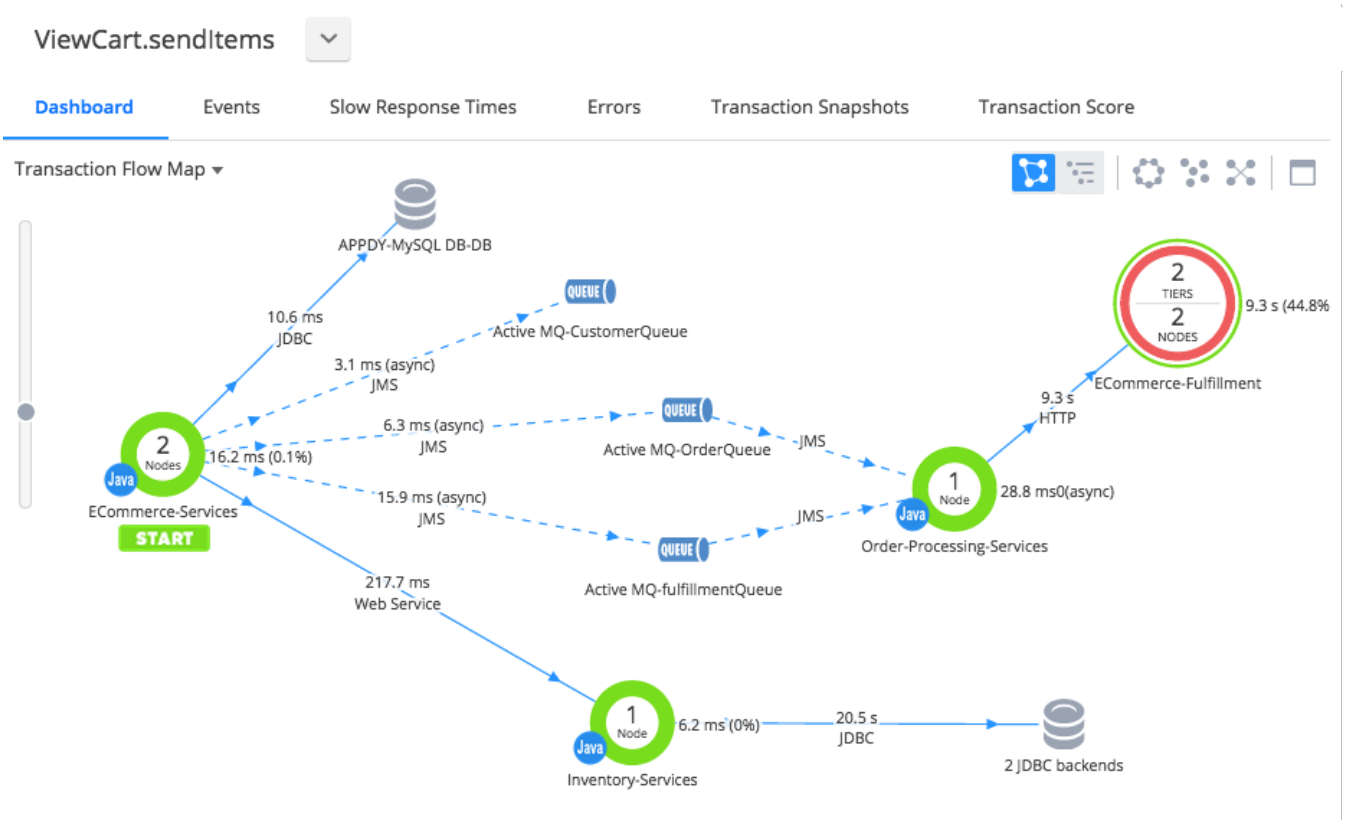
Application Model Overview

A typical application environment consists of different components that interact in a variety of ways to fulfill requests from the application's users:

- Web applications served from an application server
- Databases or other data stores
- Remote services such as message queues and caches

AppDynamics app agents automatically discover the most common application frameworks and services. Using built-in application detection and configuration settings, agents collect application data and metrics to build flow maps.

A flow map visually represents the components of your application to help you understand how data flows among the application components. For example, the business transaction flow map for a simple e-commerce application shows data flowing between web services, message queues, and databases:



Automatic detection lets you start exploring AppDynamics features quickly. As your understanding of AppDynamics matures and you identify areas unique to your environment, you can refine your application model.

Business Transactions

In the AppDynamics model, a business transaction represents the data processing flow for a request, most often a user request. In real-world terms, many different components in your application may interact to provide services to fulfill the following types of requests:

- In an e-commerce application, a user logging in, searching for items or adding items to a cart
- In a content portal, a user requests content such as sports, business, or entertainment news
- In a stock trading application, operations such as receiving a stock quote, buying or selling stocks

AppDynamics app agents discover requests to your application as entry points to a business transaction. Similar requests, such as user login, are treated as multiple instances of the same business transaction. The agents tag the request data and trace the request path as it passes from web servers to databases and other infrastructure components. AppDynamics collects performance metrics for each tier that processes the business transaction.

Because AppDynamics orients performance monitoring around business transactions, you can focus on the performance of your application components from the user perspective. You can quickly identify whether a component is readily available or if it is having performance issues. For instance, you can check whether users able to log in, check out or view their data. You can see response times for users, and the causes of problems when they occur.

Business Applications

A business application is the top-level container in the AppDynamics model. A business application contains a set of related services and business transactions.

In a small AppDynamics deployment, only a single business application may be needed to model the environment. In larger deployments, you may choose to divide the model of the environment into several business applications.

The best way to organize business applications for you depends on your environment. A leading consideration for most cases, however, is to organize business applications in a way that reflects work teams in your organization, since role-based access controls in the Controller UI are oriented by business application.

Nodes

A node in the AppDynamics model corresponds to a monitored server or JVM in the application environment. A node is the smallest unit of the modeled environment. Depending on the agent type, a node may correspond to an individual application server, JVM, CLR, PHP application, or Apache Web server.

Each node identifies itself in the AppDynamics model. When you configure the agent, you specify the name of the node, tier, and business application under which the agent reports data to the Controller.

Tiers

A tier is a unit in the AppDynamics model composed of a grouping of one or more nodes. How you organize tiers depends on the conceptual model of your environment.

Often a tier is used to a group of a set of identical, redundant servers. But that is not strictly required. You can group any set of nodes, identical or not, for which you want performance metrics to be treated as a unit into a single tier.

The single restriction is that all nodes in a single tier must be the same type. That is, a tier cannot have mixed types of agents, such as both .NET and Java nodes.

The traffic in a business application flows between tiers, as indicated by lines on the flow map, which are annotated with performance metrics.

In the AppDynamics model:

- There is no interaction among nodes within a single tier
- An application agent node cannot belong to more than one tier

Entities

An entity is any object that AppDynamics monitors, such as an application, tier, node, or even a business transaction. Entities typically have associated metrics, events, and a health status.

Historical and Live Entity Data

The Controller has an entity liveness module that tracks the "live" or "historical" status of the the four entity types: application, tier, node, and business transaction for 365+ days.

- **Historical:** Oldest time (a year before the latest Controller restart) to the latest Controller restart time
- **Live:** Latest Controller restart time until the current time

Anchor Metrics for Entities

The entities have special metrics called *anchor metrics* that are used to determine the *liveness* of the entity. This table lists the anchor metrics for each of the entities.

Entity	Anchor Metric
Application	Agent | App | Availability
Tier	Agent | App | Availability

Node	Agent | App | Availability
Business Transactions (BTs)	BTM | BTs | BT: %d | Component: %d | Calls per Minute

Liveness Status

The liveness of an entity affects the associated entities as the liveness is rolled up the hierarchy. If the entity type in the table is live, you can determine the liveness of the associated entities in the right column.

Live Entity	Liveness Status
Application	An app is alive if any tiers in this app are alive
Tier	A tier is alive if any nodes in this tier are alive
Node	Any metrics from the particular node
Business Transactions (BTs)	BT metrics, Calls per Minute

How the Controller Displays Live Entities

Based on entity liveness status of the selected time range, the Controller determines whether to count and display entities in these places:

- Flowmap
- Tier and Node list pages. This is also determined by the **Performance Data** checkboxes. See Live Entity Data in Flowmaps.
- Metric tree of the **Metric Browser**
- Custom dashboards
- AppDynamics REST APIs related to topology such as the Application Model API.

Backends

A backend is a component that is not instrumented by an AppDynamics agent but one that participates in the processing of a business transaction instance. A backend may be a web server, database, message queue, or another type of service.

The agent recognizes calls to these services from instrumented code (called exit calls). If the service is not instrumented and cannot continue the transaction context of the call, the agent determines that the service is a backend component. The agent picks up the transaction context at the response at the backend and continues to follow the context of the transaction from there.

Performance information is available for the backend call. For detailed transaction analysis for the leg of a transaction processed by the backend, you need to instrument the database, web service, or other application.

Integration with Other AppDynamics Modules

This section describes how other AppDynamics APM Platform products work with Application Monitoring to provide complete, full visibility on application health and user experience.

Application Monitoring and Infrastructure Visibility

Infrastructure Visibility provides end-to-end visibility into the hardware and networks on which your applications run. You can use Infrastructure Visibility to identify and troubleshoot problems that affect application performance such as server failures, JVM crashes, and hardware resource utilization. There are three classes of Infrastructure Visibility functionality:

- You use the Machine Agent to collect basic hardware metrics. One Machine Agent license is included with each App Agent license that you purchase. You can deploy this Machine Agent only on the same machine where the App Agent is installed. The functionality provided by the Machine Agent includes:
  - Basic hardware metrics from the server OS. For example, %CPU and memory utilization, disk and network I/O
  - Custom metrics passed to the Controller by extensions
  - Run remediation scripts to automate your runbook procedures. You can optionally configure the remediation action to require human approval before the script is started.
  - Run JVM Crash Guard to monitor JVM crashes and optionally run remediation scripts
- If you have a Server Visibility license, the Standalone Machine Agent provides this additional functionality:
  - Extended hardware metrics such as machine availability, disk/CPU/virtual-memory utilization, and process page faults
  - Monitor application nodes that run inside Docker containers and identify container issues that impact application performance
  - Tier Metric Correlator, which enables you to identify load and performance anomalies across all nodes in a tier
  - Monitor internal or external HTTP and HTTPS services
  - Group servers together so that health rules can be applied to specific server groups
  - Define alerts that trigger when certain conditions are met or exceeded based on monitored server hardware metrics
- Network Visibility monitors traffic flows, network packets, TCP connections, and TCP ports. Network Agents leverage the APM intelligence of App Server Agents to identify the TCP connections used by each application. Network Visibility includes this functionality:
  - Detailed metrics about dropped/retransmitted packets, TCP window sizes (Limited / Zero), connection setup/teardown issues, high round trip times, and other performance-impacting issues
  - Network Dashboard that highlights network KPIs (Key Performance Indicators) for tiers, nodes, and network links

- Right-click dashboards for tiers, nodes, and network links that enable quick drill-downs from transaction outliers to network root causes
- Automatic mapping of TCP connections with application flows
- Automatic detection of intermediate load balancers that split TCP connections
- Diagnostic mode for collecting advanced diagnostic information for individual connections

Application Monitoring and Browser Real User Monitoring

When you add End-User Monitoring to Application Performance Management, you can correlate business transaction performance to the user experience for those transactions. See Correlate Business Transactions for Browser RUM.

If server app agents run on the applications that serve your browser applications, you can further configure the app server agents to inject JavaScript agent into the code that runs on the browser. Access the settings to configure injection in the Applications Configuration page. See Automatic Injection of the JavaScript Agent and Assisted Injection.

Application Monitoring and Database Visibility

In Application Monitoring, a database called by an instrumented node is considered a remote service. You can get a significant amount of information on the interaction between the application node and database, but not from the database server perspective. When using Database Visibility with Application Monitoring, you can drill down to detailed database performance information directly from application flow maps. See Access Database Visibility from Application Monitoring Views.

Application Monitoring and Analytics

For those times when tracing application code does not provide enough clues to track down the cause of a problem, AppDynamics provides visibility into the transaction logs that can be correlated to specific business transaction requests. Log correlation visibility requires a license for both Transaction Analytics and Log Analytics. See Business Transaction and Log Correlation.

Install App Server Agents

Related pages:

- Agent and Controller Compatibility
- SELinux Installation Issues

Instrumenting an application adds the AppDynamics application agent (app agent) into the runtime process of the application. This page provides an overview of how to install agents in the application environment.

> All use cases for App Server agents are supported on AWS Outposts similarly as in other deployment options.

Agent Installation Quick Start

The **Getting Started Wizard** in the AppDynamics Controller walks you through the steps to download and configure an agent for your application. This section provides an overview of how to use the wizard.

The wizard generates a fully configured agent, including a node identity. You should run the wizard for each application instance you want to monitor, or until you understand how to customize the configuration manually.

Before Installing an Agent

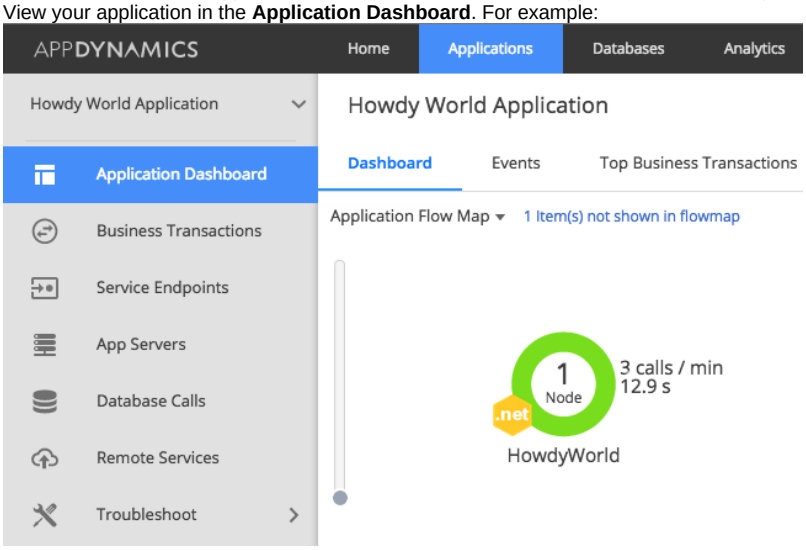
Ensure that:

- The agent supports your application environment. See App Server Supported Environments.
- You can access the application host with a user account that has sufficient privileges to install the agent and—for certain installation types—restart the application.
- The application host has network connectivity to the Controller. Proxies or firewalls on the network between the agent and Controller may require additional configuration.

Use the Getting Started Wizard

After verifying the requirements, follow the workflow as guided by the wizard:

1. Open the wizard from the home page in the Controller UI by clicking **Getting Started**.
2. Enter the configuration values for the application instance as described in the wizard. You name the node and indicate the tier and business application to which it will belong. If you are not sure of the best values to use, you can use temporary names and change them later.
3. When finished, download the agent. How you download the agent varies by agent type. In most cases, the agent comes as a ZIP file that you extract and install in the startup routine of your server. For other types of agents, you may need to modify the instrumented source code, for example, by including the agent library. The wizard walks you through it for each agent type.
4. Install the agent on your app server.
5. Apply load to your application. If you are instrumenting a production application, this will happen with customer interaction. Otherwise, create some test load on your application. The agent instruments the application code and reports metrics back to the Controller.
6. View your application in the **Application Dashboard**. For example:



The wizard makes it easy to perform a basic agent installation with minimally required settings, such as the Controller host and port, SSL, application name, and tier name. For advanced options or more complicated scenarios, you need to perform a manual installation of the agent. For more information, see the agent-specific link in the following section.

Agent Installation by Type

For detailed installation information by agent type, see these topics:

- Java Agent
- .NET Agent
- Node.js Agent
- PHP Agent
- Python Agent
- Serverless APM for AWS Lambda
- Apache Web Server Agent
- C/C++ SDK
- Go Language Agent
- IBM Integration Bus Agent
- Machine Agent
- SAP Monitoring Using AppDynamics
- Getting Started with Ruby Agent

For automated deployment guidelines, see Controller Deployment.

Tier and Node Naming Guidelines

The maximum length of a tier name is 100 characters, and the maximum length of a node name is 225 characters for Linux, and 500 characters for all other operating systems. In your tier, node and application names, you should avoid certain special characters. The characters you can use are listed on the Tiers and Nodes page.

Generally, node names should be unique. However, nodes that reside on different tiers and different machines (hosts) can have duplicate node names.

Within a business application, node names *should always be unique* in the following use cases:

- If the nodes reside on the same tier, but on different machines
- If the nodes reside on the same machine, but on different tiers
- Node names and machine names must unique. When a node is registered to a Controller, it is associated with the machine it is on, and cannot be moved to another machine without changing the node name.

If the nodes names are the same in the aforementioned use cases, the nodes will not register or report successfully.

Nodes on proxy-based agents can have duplicate names on the same tier and same machine.

To rename an application, see Business Applications.

For node naming conventions by agent type, see the installation page for that agent, such as Node.js Agent or PHP Agent.

App Agent Network Bandwidth Usage

These guidelines can help you estimate how much bandwidth overhead will be added to your environment by deploying AppDynamics agents.

Keep in mind that the exact bandwidth required for a deployment varies greatly depending on the nature of your application, the agent configuration, and the AppDynamics features you use. The best way to determine the bandwidth overhead is to test the AppDynamics deployment in a staging environment that mirrors as closely as possible the live operating environment.

1. The approximate bandwidth used by a single Java Agent with the default configuration is five to eight kilobytes per second.
2. Scaling of additional agents is linear. That is, if the average bandwidth usage for an app agent in a given deployment is five kilobytes, adding 10 means that bandwidth usage will be 5 × 10, or 50 kilobytes.
3. While the average bandwidth used is five to eight kilobytes per second, agents send data to the Controller in bursts rather than as a steady stream of data. When testing bandwidth usage, to determine the actual kilobytes per second used by an agent, you need to observe and average out traffic over the course of at least several minutes.
4. When testing bandwidth usage in the environment, keep in mind that different types of tiers will generate a different amount of load. For instance, a database tier tends to generate more traffic between the agent and Controller than an application server tier. For the best possible estimate, the test should take this into account.

Agent License Considerations

For agent-based license units—including APM, database monitoring, and server monitoring—licenses are allocated to the first agents that register with the Controller up to the licensed limit. For example, with five agent licenses, the first five agents that connect to the Controller are licensed.

Agent licenses are not bound to a particular machine or application. Therefore, a transfer of an agent-based license can be accomplished simply by shutting down the application that runs the licensed agent—uninstalling the agent if the application will need to be restarted—and starting up the new application with the newly installed agent. Once the agent disconnects, a license unit is freed for the second agent.

For application monitoring agents (Java, .NET, Node.JS, and so on), a license validation cycle runs every five minutes. It causes the agents to connect and validate that available license units are not exceeded. Historical usage data is captured during this cycle and stored as five-minute usage data. Every hour, the five-minute usage data is rolled up in hour usage data, which includes data on license unit usage. The five-minute data is purged after a few hours.

Container Installation Options

This page describes different options you can use to instrument containerized applications with an AppDynamics App Server Agent.

You can use these on-premises and cloud deployment platforms to deploy an application as a Docker container image:

- Kubernetes
- Cloud containerized platforms, such as Amazon ECS and Azure Container Instances
- Docker Swarm
- Hosts running Docker Engine

Each of these platforms supports different options to instrument a containerized application with an App Server Agent.

Kubernetes Platforms

For applications running on Kubernetes platforms (such as: OpenShift, Rancher, EKS, AKS, and GKE), you can use these options to instrument applications:

- Auto-Instrument Applications with the Cluster Agent
- Use Init Containers to Instrument Applications
- Use a Dockerfile to Instrument Applications

AppDynamics recommends auto-instrumentation because it provides the simplest operational experience. It avoids the need to change individual application images or deployment specs. It centralizes App Agent configuration to a single location and includes the App Server Agent version. This greatly simplifies initial rollouts of App Server Agents and subsequent upgrades. See Auto-Instrument Applications with the Cluster Agent.

If you cannot use auto-instrumentation, we recommend using init containers as the manual option to copy agent files to the application container at deploy time. See Use Init Containers to Instrument Applications.

You can also use Dockerfiles to copy the agent files to the Docker image at build time. See Use a Dockerfile to Instrument Applications.

Non-Kubernetes Platforms

For non-Kubernetes environments (such as: Docker Swarm, Amazon ECS, and Azure Container Instances), using Dockerfiles is the best option. See Use a Dockerfile to Instrument Applications.

Depending on the environment, you may be able to install the App Server Agent without changing the application image. For example, you can use a Docker shared volume to provide a running container access to the App Server Agent files. Amazon ECS supports container dependencies which provide a way to reference a second container at startup. You can then use the second container to copy the App Server Agent files to the application container.

Instrument Kubernetes Applications Manually

When Auto-Instrument Applications with the Cluster Agent is not available, you can manually instrument Kubernetes applications using one of these methods:

- Use Init Containers to Instrument Applications
- Use a Dockerfile to Instrument Applications

To configure agents for manually instrumented applications, see Best Practices to Configure Agents in Kubernetes.

To establish APM correlation between the Cluster Agent and manually instrumented applications, see Manually Configure App Agents to Correlate with the Cluster Agent.

Use Init Containers to Instrument Applications

This page describes how to use the init containers option to copy agent files into a Kubernetes application container at deploy time.

> When the Cluster Agent is installed in a cluster, AppDynamics recommends Cluster Agent auto-instrumentation. See Auto-Instrument Applications with the Cluster Agent.

The init containers option is available in Kubernetes environments used to run additional containers at startup that helps initialize an application. Once the init containers have completed their initialization tasks, they terminate but leave the application container(s) running. For the App Server Agent installation, init containers are used as a delivery mechanism to copy the agent files into the application container at deploy time. See Init Containers Option in Kubernetes.

Copying the agent files at deploy time (instead of at build time) provides advantages over explicitly copying the agent files and dependencies into the application image. The init containers option simplifies the image build process because:

- You do not need to update each application's Dockerfile to copy the agent files.
- You do not need to rebuild the application image for agent upgrades.

See the language-specific container installation page for how to use an init container:

- Install the Java Agent in Containers
- Install the .NET Agent for Linux in Containers
- Install the Node.js Agent in Containers

Kubernetes Init Container Deployment Spec Example

In the Kubernetes deployment spec example for a Java application, the init container is a second Docker image that contains the App Server Agent files. Because this Docker image is built separately from the application image, you do not need to rebuild the application image when you add or upgrade the App Server Agent.

```
kind: Deployment
spec:
  containers:
  - name: java-app
        image: repo/java-app:v1
    volumeMounts:
    - mountPath: /opt/appdynamics
      name: appd-agent-repo
  initContainers:
  - command:
    - cp
    - -ra
    - /opt/appdynamics/.
    - /opt/temp
    name: appd-agent
        image: docker.io/appdynamics/java-agent:20.6.0
    volumeMounts:
    - mountPath: /opt/temp
      name: appd-agent-repo
  volumes:
    - name: appd-agent-repo
      emptyDir: {}
```

Best Practices to Configure Agents in Kubernetes

This page describes best practices to use when configuring App Server Agents in a Kubernetes environment and manually instrumenting applications using init containers or a Dockerfile.

> This page does not apply to applications that use the Cluster Agent auto-instrumentation to configure agents. See Auto-Instrument Applications with the Cluster Agent.

You can configure App Server Agents in Kubernetes using environment variables. Depending on the use case, you have three options to set environment variables:

- Use ConfigMaps to Configure the App Server Agent
- Use Secrets for the Controller Access Key
- Set Application-Specific Configuration in the Deployment Spec

See the language-specific container installation page to implement these best practices:

- Install the Java Agent in Containers
- Install the .NET Agent for Linux in Containers
- Install the Node.js Agent in Containers

Gather Login Credentials

To access AppDynamics, you need your login credentials and the URL from where you can access your organization's AppDynamics Controller(s). If you do not have this information, contact your AppDynamics administrator.

Use ConfigMaps to Configure the App Server Agent

When instrumenting a container running in Kubernetes, each App Server Agent requires that you set environment variables to configure:

- Connection parameters
- Application metadata
- Agent behavior

You can use ConfigMaps to store an application's environment variables in a Kubernetes object within the scope of a namespace. ConfigMaps are an optimal twelve-factor method used to manage agent configuration shared across multiple Kubernetes applications within a namespace.

Controller credentials configuration is stored in a single ConfigMap, and shared with each application in the namespace that reports to that Controller. If you use multiple Controllers across different environments, such as development, test, and production, you can store different ConfigMaps in the namespaces associated with those environments.

Use Secrets for the Controller Access Key

The Controller access key is another configuration that you can pass to App Server Agents as an environment variable. Sensitive data is stored in a Kubernetes namespace using a Secret, rather than a ConfigMap. You can use Kubernetes Secrets to obfuscate values rather than manipulate them in plaintext.

Set Application-Specific Configuration in the Deployment Spec

The AppDynamics Controller application tier name identifies a service that reports under a particular AppDynamics application. An application tier name is an example of agent configuration that is different for each Kubernetes application but typically does not vary across environments and namespaces. Application-specific configuration, similar to the tier name, is easier to define in the Kubernetes deployment spec directly.

Verify the Environment Variables

After you set the agent environment variables, you can verify that they were added to your application.

To verify the environment variables:

1. Create a ConfigMap.
2. Create a Secret.
3. Create an environment variable in the deployment spec.
4. Restart the application container.
5. Log in to the container using `kubectl exec`:

   ```
   kubectl -n ecommerce exec -it <pod name> /bin/sh
   ```

6. List the AppDynamics environment variables available in the container to confirm they are defined and have the correct values:

```
env | grep APPD
APPDYNAMICS_AGENT_APPLICATION_NAME=<value>
APPDYNAMICS_AGENT_ACCOUNT_NAME=<value>
APPDYNAMICS_CONTROLLER_HOST_NAME=<value>
...
```

Use a Dockerfile to Instrument Applications

This page describes how to use a Docker file to copy the App Server Agent files to the Docker image.

> When the Cluster Agent is installed in a cluster, AppDynamics recommends Cluster Agent auto-instrumentation. See Auto-Instrument Applications with the Cluster Agent.

A common method to instrument a containerized application is to include the AppDynamics App Server Agent when building the application image. You add a `COPY` command to the Dockerfile to copy the agent files to the image. The `COPY` command copies the application binary to the Docker container image. If an App Server Agent upgrade is required, you must rebuild the image with the new agent files.

See the language-specific container installation page for instructions on how to use a Dockerfile:

- Install the Java Agent in Containers
- Install the .NET Agent for Linux in Containers
- Install the Node.js Agent in Containers

Dockerfile Example

In this example for a Java application, the Java Agent files in the `AppServerAgent` folder are copied to the application image and the application JAR file. The agent environment variables are set before the start script runs.

```
FROM openjdk:8-jre-slim

COPY myapp.jar /app
COPY AppServerAgent/ /opt/appdynamics

ENV APPDYNAMICS_AGENT_APPLICATION_NAME=<value>
ENV APPDYNAMICS_AGENT_TIER_NAME=<value>
ENV APPDYNAMICS_AGENT_ACCOUNT_NAME=<value>
ENV APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY=<value>
ENV APPDYNAMICS_CONTROLLER_HOST_NAME=<value>
ENV APPDYNAMICS_CONTROLLER_PORT=<value>
ENV APPDYNAMICS_CONTROLLER_SSL_ENABLED=<value>
ENV APPDYNAMICS_JAVA_AGENT_REUSE_NODE_NAME=true
ENV APPDYNAMICS_JAVA_AGENT_REUSE_NODE_NAME_PREFIX=<value>

COPY ./startup.sh /startup.sh
RUN chmod +x /startup.sh
ENTRYPOINT ["/bin/bash", "/startup.sh"]
```

Agent-to-Controller Connections

AppDynamics app agents need to connect to an AppDynamics Controller to retrieve configuration data and send back information about the monitored environment. This page provides information about the connections between the agents and Controller.

Connection Settings

The connection between the agent and Controller is a one-way connection initiated by the agent. Therefore, you only need to configure connection settings in the agent.

To connect an agent to the Controller, you must provide this information:

- Controller host: The hostname of the Controller to connect to. Agents may connect directly to the Controller or through a proxy.
- Controller port: The port on which the Controller listens for agent traffic. Agents use port 443, which is the same port as the browser connection to the Controller UI.
- Account name: The name of the account listed in the Controller. A single tenant Controller has two accounts, a default account name, customer1, and an internal system account. For most connections, use the default account name.
- Global Account name: The Global Account name is used for certain connections, such as to the Events Service or from the Analytics Agent.
- Account access key: A unique key associated with the Controller account. See Find Your Account and Access Key.
- SSL enabled: If the agent should connect using SSL.

If you downloaded the agent through the Getting Started Wizard in the Controller, the Controller host, port, and account settings are already configured.

To connect the agents in your environment to a SaaS Controller or an on-premises Controller through a proxy, configure the agent properties to use the host and port settings of the Controller. For an example, see Java Agent Configuration Properties for proxy settings descriptions.

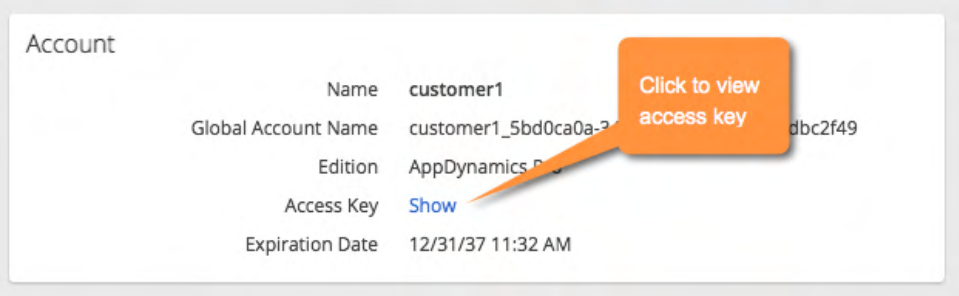
To connect to SaaS Controllers from agents in environments that limit outbound connections, set firewall rules that permit access to AppDynamics SaaS platform components. For a list of SaaS IP addresses, see SaaS Domains and IP Ranges.

Find Your Account Name and Access Key

To configure the agent manually rather than through the Agent Download Wizard, set the Controller host and port, account name, and account access key settings manually. If you have an Admin account, you can find your account name and access key in the AppDynamics Controller UI. If you are not an admin, ask your administrator for your access key.

To view your account name and default access key:

1. Log in to the Controller UI as a user with view license permissions.
2. Click the gear icon and select **License**.
3. Click on the Account tab. The account name appears next to the **Name** label. The account name is customer1 in a single tenant, on-premises Controller; the account name varies on SaaS Controllers.
4. Click **Show** next to the **Access Key** label to reveal the account default access key setting for this instance. Use this value as the Account access key setting in agent settings.



If you create license rules in addition to the default rule, the access key for each rule appears in the Rules tab.

Secure the Connection

The .NET Agent uses the settings in the container to negotiate the SSL protocol with the Controller. Normally you do not need to configure the security protocol for the .NET Agent.

For the Java Agent, see the Agent and Controller Compatibility for a list of the default security protocols for different versions of the Java Agent.

If the default security protocol for your version of the Java Agent is incompatible with the Controller or an intervening proxy, pass the `Dappdynamics.agent.ssl.protocol` system property to set the protocol to one of these security protocols:

- SSL
- TLS
- TLSv1.2
- TLSv1.1

Use the following format:

```
java -javaagent:<agent_home>/javaagent.jar ... -Dappdynamics.agent.ssl.protocol <protocol> ...
```

Controller-specific security considerations vary between SaaS and on-premises Controllers, as described below.

On-premises Controller Secure Connections

An on-premises Controller has both an active secure (HTTPS) port and an HTTP port. Agents can use either port to connect to the Controller. By default, the certificate used for the secure connection is a self-signed certificate. The .NET Agents cannot connect on a secure port that uses a self-signed certificate, so you must apply your own certificate to the secure port. App Agents connecting to an AppDynamics SaaS Controller must use an HTTPS connection.

To implement SSL for the Controller-agent connection:

- Set the application server primary port to the SSL port, 8181 by default. See Port Settings.
- Install a trusted certificate. See Controller SSL and Certificates.
- Configure your agents for SSL. See App Agent Security and Machine Agent Security.

SaaS Controller Secure Connections

SaaS Controllers require the use of SSL. Therefore, you only need to enable SSL in the configuration settings for your agents and connect them to the secure Controller port, 443. See App Agent Security and Machine Agent Security.

Using AWS PrivateLink with a SaaS Controller

Some customers may have a policy where they do not want agent traffic exposed to the public Internet even though the agent traffic is encrypted. AppDynamics customers can leverage Amazon Web Services (AWS) inter-region PrivateLink to privately connect their cloud-based applications between Virtual Private Clouds (VPC) and AppDynamics SaaS and on-premises Controllers. In this scenario, all agent traffic flows across Amazon's private network, even if VPCs reside in different AWS regions. Before you can ingest data from the AppDynamics endpoint to your AWS VPCs and AWS Accounts using AWS PrivateLink, you must complete several set up steps. See How do I use AWS PrivateLink to connect to an AppDynamics SaaS Controller?

App Agent Security

To configure your agents for SSL, set these SSL-related properties:

- Set `controller-ssl-enabled` to `true`.
- Set the `controller-port` to the correct value for either on-premises or SaaS Controller.

In multi-tenant and SaaS environments, App Agents authenticate themselves to the Controller using the required account name and account access key values set in the connection properties configuration file.

Standalone Machine Agent Security

For information on the security settings related to the Machine Agent connection to the Controller, see Machine Agent Configuration Properties.

Verify the Connection

You can verify that an app agent is reporting to the Controller from the **Tiers & Nodes** list in the Controller UI. See Tiers and Nodes. You can also verify the connection from the AppDynamics Agents page, under the gear icon, see Manage App Agents.

In the **Tiers & Nodes** pages, the **App Agent Status** column indicates the status of the agent connection to the Controller. A green arrow icon indicates active connected agents, a red down arrow indicates an agent that has been previously recognized but is not currently connected.

If the agent is not reporting to the Controller, see troubleshooting information:

- Troubleshooting Java Agent Issues
- Troubleshoot .NET Agent Issues
- Dynamic Language Agent Proxy
- Resolve PHP Agent Installation Issues

If traffic is not being properly correlated between tiers, make sure that any network components, such as load balancers or routers that sit between monitored nodes, are preserving the AppDynamics correlation header from HTTP traffic.

Agent-Controller Communication Intervals

Each AppDynamics agent has multiple communication channels for different purposes that initiate connections to the Controller independently, and at different time intervals.

- The agent configuration channel queries the Controller for any new configuration changes, and downloads these changes when available, every 60 seconds.
- The agent metric channel posts all new periodic metrics, including JMX, Windows performance counters, and business transaction metrics to the Controller every 60 seconds.
- If there are new business transactions that have not been seen before by the agent, they are posted to the Controller for registration every 10 seconds.
- If the agent has collected any new snapshots or events, they are posted to the Controller every 20 seconds.

Information Sent to the Controller

This table shows the types of information that is collected by an application agent and sent to the Controller.

Metric	Description
Server Request GUID	A unique GUID identifying a request, known as a Business Transaction, in the form of `xxxxxxxx-xxxx-xxxx- xxxx-xxxxxxxxxxxx` where x is a lower-case hexadecimal digit.
Application Entry Points	Transaction entry points are identified among various frameworks and technologies. This includes Servlet URIs, Strut Action and Method name, Spring Bean Name and Method Name, JMS queue destination or listener name, Web Service/WCF action/operation name, PHP Virtual Name, and more.
Application Exit Points	Transaction exit points are identified among various frameworks and technologies. This includes: HTTP URL end points, JMS queue /destination, type, and vendor; database URL endpoint and vendor/version, web service Service Name, cache name and URL endpoint.
Performance Metrics	In general, for each monitored metric in AppDynamics, a response time, call rate, and error rate are collected. Each of these metrics also have an automatic baseline derived for each respective metric value.
Thread Stack Traces	Code level method execution metrics that comprise the application request are collected. This data includes the class and method that executed and the line number within the source code. Note that less than 5% of transactions will have stack traces collected. Errors /Exceptions and stack trace of error data will be collected.
SQL Query Values	If assigned with administrative permissions, SQL query variables within a query can be enabled, collected, and viewed. Note that the parameter data is collected for less than 5% of transactions. Configuration changes are logged in an audit log that is available for security review.
Data Collectors	If assigned with administrative permissions, data in the form of HTTP values or method payload can be collected and viewed. Note that a specific data collectors and code payload accessors require explicit configuration to be collected. Note, data is collected for less than 5% of transactions. Configuration changes are logged in an audit log that is available for security review.
Hostname	The DNS hostname of the machine (virtual/physical) from where the agent is installed and reporting monitoring data.
IP Address Internet Protocol (IP)	IP Address Internet Protocol (IP) address of the machine (virtual/physical) where the agent is sending monitoring data from.
CPU Usage	The value of CPU that is consumed on the monitored machine/virtual machine.
Memory Usage	The value of physical memory that is consumed on the monitored/virtual machine.
Network I/O Usage	The value of network I/O that is consumed on the monitored machine/virtual machine.
Disk I/O Usage	The value of disk I/O that is consumed on the monitored machine/virtual machine.
JVM Performance	JVM Heap Usage, JVM Memory Pools Settings, Garbage Collection performance, JVM System/Start-up Options, MBean metric values (for example, connection pool names and metric values, such as active connections, maximum connections, and so on)

Blitz Load Profile

Blitz is a horizontally scalable data processing platform for SaaS deployments. It collects metric data from agents, which it then aggregates and stores.

The 10M metrics/min Blitz load profile includes the following agents and churn information:

- Active load 10MM with 24K nodes. Total registered metrics is 40M.
- Node distribution:
    - DotNet = 3K
    - SIM Machine Agents OR Docker Containers = 30K
    - DBMon Data Collectors = 1K

    - Java Nodes = 16K
    - EUM Browser Apps = 60

- Java Node Churn: 1K/hr
- Either SIM or Docker Churn

    - Sim Node Churn = 40/hr (1% of 4K SIM nodes)
    - Docker Churn = 200 containers/hr

- Node purger enabled with hard-limit of 4K and soft-limit of 10K
- SIM node purger enabled with a deletion max limit of 300/hr

Agent Installer

Deployment Support

> The Agent Installer requires microservices configuration performed by AppDynamics.

The AppDynamics Agent Installer simplifies deployment to instrument your applications faster. You can manage applications instrumented with the Agent Installer with minimal code changes in Monitoring Settings.

The Agent Installer:

- Deploys Infrastructure, Java, and Machine Agents, and is compatible with Linux

  > The Infrastructure Agent is in beta release only.

- Automatically instruments applications
- Assigns unique names to tiers and nodes

You can deploy other agents using the Getting Started Wizard.

Agent Installer Overview

The Agent Installer works with these items:

- *Agent Installer* - Executable installer inside the `appdynamics-zero-agent-<version>.zip`. file (representing the Agent Installer Platform). The Agent Installer installs the Agent Installer Platform with Infrastructure, Java, and Machine Agents on a system. You access the installer from the Controller UI.
- *Decorator* - Library responsible for automatically instrumenting new processes with AppDynamics APM agents. The library auto-instruments the application and auto-names tiers and nodes for the Controller.
- *Agent Installer Platform* - Software bundle that manages the Decorator, Infrastructure Agent, Java Agent, and Machine Agent. Enables automatic instrumentation and reports the status of various processes to the Controller UI. Collective term for the decorator, agent `daemon`, agent binaries.
- *Downloader* - Script (`zero-agent.sh`) that downloads individual zip files containing the Infrastructure, Java, and Machine (if applicable) Agents, and Agent Installer Platform.

Agent Installer Requirements

The Agent Installer requires the following components, permissions, and supported environments.

AppDynamics Components

- The Agent Installer requires SaaS Controller => 20.6.0.
- You also need sufficient APM licenses to use the Java and Machine Agents. No additional license is required for the Agent Installer Platform or the Infrastructure Agent.

Configure and Unblock Your Firewall

You may need to configure your firewall rules to allow outgoing traffic to certain URLs including:

- Access to your AppDynamics Controller: `<customername>.saas.appdyanamics.com/*` to allow APM traffic.
- Access to the AppDynamics download files site: `download-files.appdynamics.com` to download the agent binaries.
- (If Infrastructure Agent is installed as a Docker container) Access to Docker containerization software: `*.docker.io`.

> You can run the Infrastructure Agent in a container or directly on the host. If Agent Installer detects an operational Docker runtime, it attempts to install the Infrastructure Agent in the container mode whether or not you have configured the firewall to allow Docker Hub access.

Agent Installer Permission

> To configure access and roles, you must either be the AppDynamics Account owner, or have Administrator permissions. Additionally, Cloud Native Visualization requires configuration changes in the AppDynamics SaaS platform. The configurations began rolling out to SaaS Controllers on October 28, 2020.
>
> To visualize Infrastructure Agent entities, you need **View Infrastructure Entities** account-level permission. See Configure Cloud Native Visualization Permissions.

To use the Agent Installer, you need **Install Agent** account-level permission and at least one of the following:

- Any default role (except for the Analytics Administrator role)
- Any application-level permission
- **Execute Workflow** account-level permission

Install Agent permission is not added to any default role. See Roles and Permissions.

Supported Environments

This table lists the Agent Installer supported environments. JVM and Application Server requirements only apply if you install the Java Agent. The Agent Installer does not support Windows installations.

Agent Installer Supported Environments	
Operating System	Linux (64 bit). Fully tested flavors: • CentOS 7, 8 • Rhel 7, 8 • Ubuntu 18.04 • Fedora 31 • Debian 10 Other Linux operating systems and versions should work but are not certified by AppDynamics.
JVM	• Open => JDK 1.6 • Oracle => JDK 1.7 • IBM JDK => 1.6
Application Server	• Apache Tomcat • Jboss Wildfly • GlassFish • Websphere • Weblogic

See Java Supported Environments and Machine Agent Requirements and Supported Environments for version compatibility.

Agent and Process Limitations

This table lists the active agent registrations, process, and process group limitations for the Agent Installer:

Agent	Maximum Per-Account Limit
Active agent registrations	1000
Process	**Maximum Per-Account Limit**
Managed processes	10,000
Unmanaged processes	10,000
Managed processes in a single installation	100
Unmanaged processes in a single installation	100
Process Group	**Maximum Per-Account Limit**
Managed process groups	1000
Unmanaged process groups	1000

Managed process groups in a single installation	100
Unmanaged process groups in a single installation	100

Available Agents with the Agent Installer

The Agent Installer deploys the Agent Installer Platform, which downloads monitoring agents to your machine. You can install:

- Infrastructure Agent >= 21.1.0
- Java Agent >= 4.5.19
- Machine Agent >= 20.3.0

Use the Agent Installer to Deploy an Agent

To use the Agent Installer to deploy an agent:

1. From the Controller UI, select **Home** > **Agent Installer**.
2. From the **Specify Application to Deploy to** dropdown, select an existing application, or select **New application** and enter its name.

   > Application names cannot contain a single quotation mark (').

3. Download and run the Agent Installer using either an express installation or a custom installation method.

Express Installation Method

> The express installation method is not available for the Infrastructure Agent.

1. The **Express installation** method is selected as the default. With minimal effort, you can download and install the latest version of all available agents.
2. Copy and run the first command to download the latest available agents.
3. Select (top right corner of the second command) to copy and run the second command to install the agents. Additionally, you can select **Show command** to reveal the full command with your access key.
4. Restart the application processes you want to monitor.
5. Click **Next** to continue with the instrumentation.

After you finish installing the agents, you are redirected to the Monitoring Settings tab to view instrumented tiers and nodes.

Custom Installation Method

1. Select **Custom installation**.
2. Select the **Infrastructure**, **Java**, or **Machine Agent**(s) you want to install.

   > You must select at least one agent to proceed with the installation.

3. The default version displays next to the selected agent(s). You can edit the version from the corresponding dropdown. A command based on the agent(s) you selected is generated.
4. Copy and run the command into your terminal or automated tools.
5. Determine how to run the Agent Installer by setting user permissions:
   - Select **Instrument processes for all system users** (requires `sudo` access) to run the Agent Installer with `sudo` permission and enable the Agent Installer Platform for *all* users on the host.
   - Select **Instrument processes for an individual user** to run the Agent Installer without `sudo` permission and enable the Agent Installer Platform for the *current* user only.
     See Sudo vs. Non-Sudo Access for more information.
6. After confirming the user installation permissions, select (top right corner of the command) to copy and run the command. To reveal the full command with your access key, select **Show command**.

   > When you run the command on a designated host, the agent files download onto that host *only*. To install agents on multiple hosts, select Run the Agent Installer on Multiple Hosts.

7. Restart the application processes you want to monitor.
8. Click **Next** to continue with the instrumentation.

After you finish installing the agents, you are redirected to the Monitoring Settings tab to view instrumented tiers and nodes.

Sudo Versus Non-Sudo Access

You can run the Agent Installer with or without `sudo` permission.

> AppDynamics recommends using the `sudo` command to install the agents for all users in your system.

This table describes the differences between `sudo` and `non-sudo` installations:

	sudo Installation	**Non-sudo Installation**
Description	Installs the agents on all supported processes for all users in your system.	Installs the agents for the current user only.
Automatic Instrumentation Process	By default, integrates with `systemd` to ensure the agents always run. See Customize the Agent Installer to override this behavior.	Auto-instruments processes that are started by the installing user, running under a Linux shell such as `bash`.

Run the Agent Installer on Multiple Hosts

You can download the Agent Installer once and run on multiple hosts. To install the agent bundle across multiple hosts, distribute the binaries to all applicable machines.

For example, these steps show how to move files across hosts using the `tar` command.

1. Download the agent bundle (see Use the Agent Installer to Deploy an Agent).
2. Distribute the Agent Installer to multiple hosts.
    a. Make sure you are in the same directory in which you downloaded the agent bundle.
    b. Use the `tar` command to place the script in a single file:

    ```
    tar cvf zero-agent.tar *
    ```

    c. Copy the file to multiple hosts:

    ```
    scp zero-agent.tar <target host>:<directory>
    ```

    d. Extract the archived script in the same directory:

    ```
    tar xvf zero-agent.tar
    ```

3. Complete the Agent Installer deployment using the custom installation method.

Upgrade the Agent Installer

Before you can upgrade the Infrastructure, Java, and Machine Agents (which run through the Agent Installer), you must first uninstall your current agents. In the terminal, navigate to the directory where the Agent Installer Platform is installed and run this command:

```
<install-directory>/bin/zeroctl uninstall
```

You can now install the new versions of the Infrastructure, Java, and Machine Agents with the Agent Installer.

Customize the Agent Installer

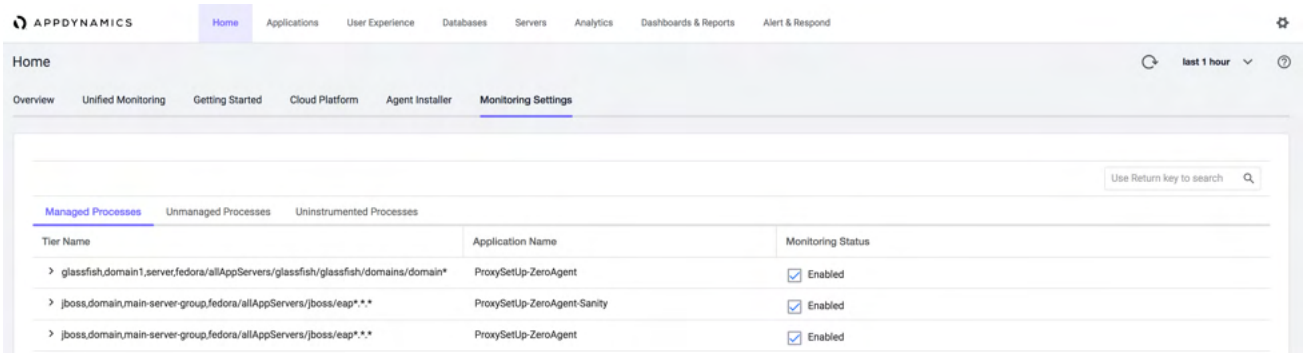
See Customize the Agent Installer to modify the agent configuration.

Monitoring Settings

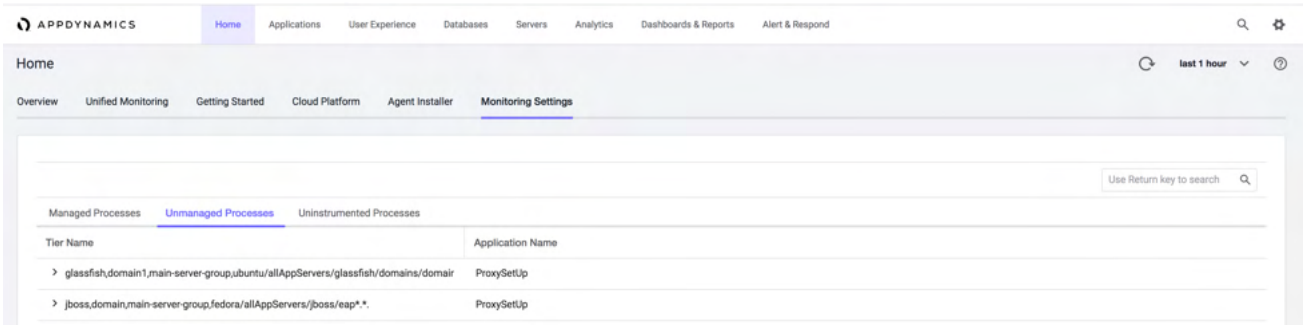
This page describes how to use Monitoring Settings to monitor and update the processes instrumented by the Agent Installer.

In the Controller UI, select **Home** > **Monitoring Settings**. The Monitoring Settings page is an interface that displays tiers, nodes, and applications associated with the Agent Installer. The Monitoring Settings page contains three tabs:

- Managed Processes
- Unmanaged Processes
- Uninstrumented Processes



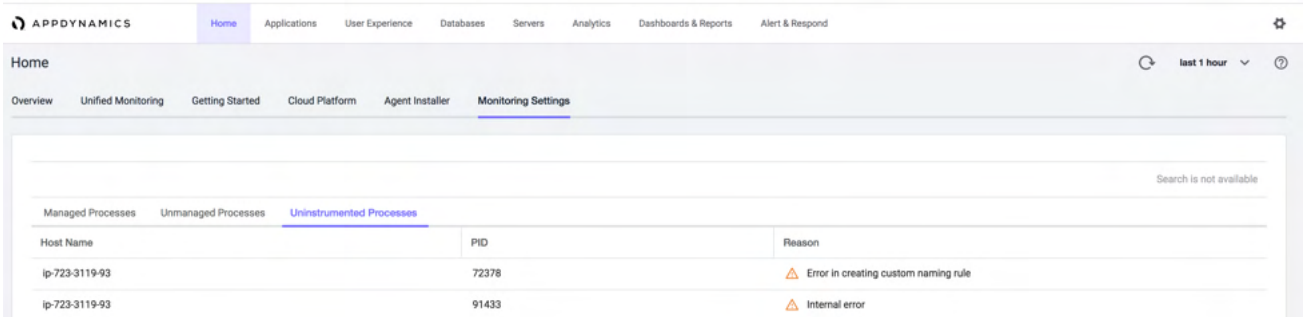
You can view and scroll through pages of processes and tiers where tiers are listed by their associated application. For example, a tier may be monitored by a Java Agent configured with Agent Installer.



From both the Managed Processes and Unmanaged Processes tabs:

- By default, all nodes under a tier are collapsed. To expand the tier and review its nodes, select the arrow next to the **Tier Name**.
- Search on a tier name, application name, or node name by entering the search string in the upper-right corner **Search** box and pressing Return. To facilitate searching, you can right-click an application name, and then select **Copy application name** from the popup to copy that name to the clipboard and then paste into the **Search** box.

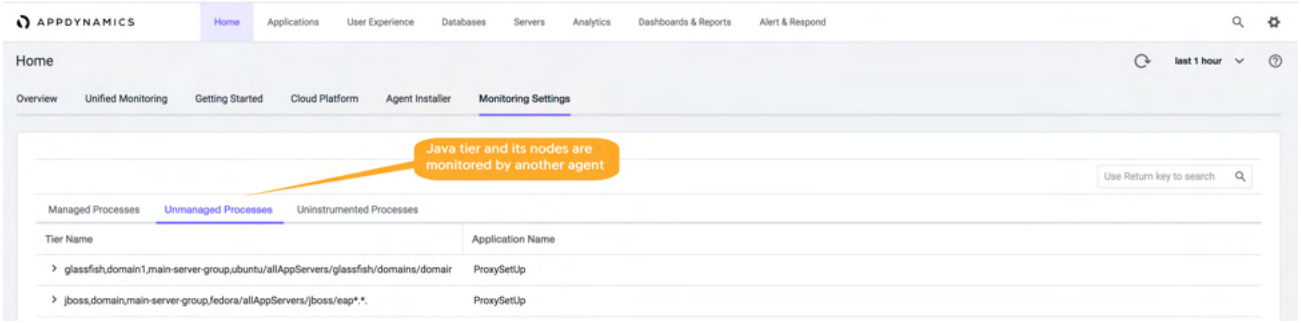
> Search is not available from the Uninstrumented Processes tab.



Managed and Unmanaged Tiers

A tier is considered "managed" when it has been instrumented with the Agent Installer. In the Monitoring Settings tab, managed tiers are those that you can enable or disable in the Controller UI.

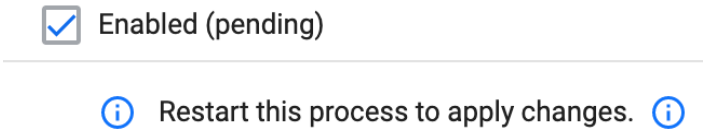
You may see tiers listed on the Monitoring Settings page that you cannot change in the UI. A tier is considered "unmanaged" when it cannot be instrumented by the Agent Installer. Unmanaged tiers include tiers that have been instrumented manually with the Java Agent or managed outside of AppDynamics. To monitor the unmanaged tiers, you must remove your previous configuration and restart the processes.



Change Managed Tiers and Nodes Status

To disable monitoring for specific tiers, uncheck one or more tiers you want to disable.

Restart all affected processes (on all affected hosts). This messages displays in the UI when the process is ready to restart:



When you disable monitoring, the changes are not immediately recognized by the Agent Installer Platform. Wait until you see a message to restart processes in the Monitoring Settings tab. If you restart processes before you see the message, the change does not take effect.

Disable Instrumentation of Specific Hosts

To reduce license consumption, you can disable the instrumentation of application server types that you do not want to monitor.

In the terminal, enter this command: `<install-directory>/bin/zeroctl configure --disable-instrumentors=java,<app-server-type>`

The location `<install-directory>` refers to the directory in which you installed the Agent Installer.

This table lists the supported values for `<app-server-type>`:

App Server	Accepted Value
Tomcat	`tomcat`
JBoss	`jboss`
GlassFish	`glassfish`
WebLogic	`weblogic`
WebSphere	`websphere`
Plain (non-application server) Java applications	`java`
All app server types	`all`

Customize Agent Installer

This page explains how to fine-tune the Agent Installer configuration. Changes apply to the Agent Installer Platform configuration file, `config.json`.

Customize Agent Installer Platform Configuration

The Agent Installer Platform has several commands available to customize the configuration. You can run the commands during or after installation. You can issue the commands to the `zeroctl` executable, a binary located in the Agent Installer Platform directory. The Agent Installer Platform periodically checks the configuration file and applies the changes.

You should only use these commands in the `zeroctl` executable. Do *not* modify the `zero-config.json` file directly as this may result in undefined behavior.

Command Name	Description
`configure`	Modify the Agent Installer Platform configuration file.
`stop`	Stop running the Agent Installer Platform.
`stop <agent>`	Stop running an individual agent on the Agent Installer Platform.
`start`	Start or restart the Agent Installer Platform. Automatically done for `sudo` installations.
`start <agent>`	Start or restart an individual agent on the Agent Installer Platform.
`status`	View or print the status of the managed agents.
`purge`	Remove the Agent Installer Platform, Infrastructure, Java, and Machine Agents and all configuration files from your machine.
`uninstall`	Remove the Agent Installer Platform, Infrastructure, Java, and Machine Agents from your machine, but retain the configuration files.

The Infrastructure Agent is in beta release only.

View Command Flags

To view the available flags for a command, enter:

```
<install-directory> bin/zeroctl <command> -h
```

For example, to view the available flags for the `install` command, enter:

```
<install-directory> bin/zeroctl install -h
```

The output example displays the flags with the data type for the parameters and descriptions for the `install` command:

```
<install-directory> bin/zeroctl install -h
...
Install Agent Installer Platform


Usage:
  zeroctl install [flags]


Flags:
      --log-level string                Logging level (choose from [panic fatal error warning info debug
trace])
      --max-log-rate int                Maximum log msg rate (default -2)
      --disable-instrumentors strings   Instrumentors to disable (choose from [java tomcat jboss glassfish
weblogic websphere all])
      --enable-instrumentors strings    Instrumentors to enable (choose from [java tomcat jboss glassfish
weblogic websphere all])
      --application string              Controller application name
      --account string                  Controller account name
      --access-key string               Account access key
      --service-url string              Service URL
      --javaagent string                Java agent (choose from [dynamic ibm sun])
      --jboss-log-manager-modify        LogManager args for JBoss (choose from [true false])
      --proxy-url string                Proxy URL
      --install-path string             Agent Installer Platform installation path (default "/opt/appdynamics
/zeroagent")
      --systemd                         Install in systemd (default true)
  -h, --help                            help for install
```

Run Commands with Flags During and After Installation

You can run commands in the `zeroctl` binary after installation. To make changes during installation, add flags to the installation script.

For example, to configure the log level to `info`:

- During installation (as a flag), enter:

  ```
  sudo ./zero-agent.sh install --application 'AgentInstallerTestApp' --account 'project-zero' --access-key
  'myaccesskey' --service-url 'https://test.saas.appd-test.com' --log-level=info
  ```

- After installation (using the `zeroctl` executable), enter:

  ```
  <install-directory> bin/zeroctl configure --log-level=info
  ```

Connect the Agent Installer Platform to a Proxy

The Agent Installer supports connections with unencrypted proxies only. To configure the `zeroctl` executable to use a proxy connection, enter:

```
<install-directory> bin/zeroctl configure --proxy-url=http://localhost:1001
```

Change Tier Names

The Agent Installer automatically creates a name for every instrumented tier. To create names, the installer extracts information from the Java process, such as the application server name.

Contact AppDynamics Support to change the name of an instrumented tier. Because an AppDynamics Support representative applies the changes, you need to create a new user named `appd_rule_user`, and then give `appd_rule_user` **Install Agent** permission only. This secures your sensitive data.

When you contact AppDynamics Support, provide the log in credentials for `appd_rule_user`.

Override `Systemd` Integration

If you do not want the Agent Installer to integrate with `systemd` in a `sudo` installation, add `-systemd false` to the install command (step 5a in Use the Agent Installer). For example:

```
sudo ./zero-agent.sh install --application 'AgentInstallerTestApp' --account 'project-zero' --access-key
'myaccesskey' --service-url 'https://test.saas.appd-test.com' --systemd=false
```

Maintain Continuous Operation with `watchdog`

The Agent Installer includes a `watchdog` process to ensure continued operation of these standalone agents and their related processes:

- Agent Installer Platform Daemon
- Infrastructure Agent
- Machine Agent

If you integrate the Agent Installer with `systemd`, a profile is created for the `watchdog` process to ensure that it starts after a planned or unplanned reboot of the system. The `systemd` profile also restarts the `watchdog` process if it was terminated during regular operation.

If you do not integrate the Agent Installer with `systemd` (by setting `--systemd=false`), you can create a `systemd` profile after installation.

If you do not have a `systemd` profile, then you must start the `watchdog` process manually to ensure operation of the standalone agents and their related processes.

> If you enabled the SELinux module on your system, then you must configure it to ensure that the `watchdog` and other Agent Installer Platform processes are not blocked.

To start the `watchdog` process, enter:

```
<install-directory> bin/zeroctl start
```

To stop the `watchdog` process, enter:

```
<install-directory> bin/zeroctl stop
```

See Agent Installer for more information.

Start or Stop Individual Agents

You can use the watchdog process to start and stop individual agents (Infrastructure, Machine, or Zero) on the Agent Installer Platform. You can also view or print the status of the managed agents.

For example, to start the Machine Agent, enter:

```
<install-directory> bin/zeroctl start machine
Starting Machine Agent
...
Machine Agent started successfully
```

For example, to stop the Zero Agent, enter:

```
<install-directory> bin/zeroctl stop zero
Stopping Zero Agent
...
Zero Agent stopped successfully
```

To view or print the status of the managed agents, enter:

```
<install-directory> bin/zeroctl status
```

Secure Agent Installer Platform

This page describes how to secure sensitive information in the Agent Installer Platform configuration file.

How to Secure the Agent Installer Platform Configuration File

The Agent Installer Platform configuration file, `config.json`, contains potentially sensitive information, such as the agent access key used by the Java and Machine Agent to connect to the Controller. By default, the Agent Installer Platform can be read by any user of the system after it is installed. The default permissions allow the Agent Installer Platform to auto-instrument new processes started on your system by any user.

Block Access to the Agent Installer Platform

You can block access to the Agent Installer Platform configuration file and still allow the agent to auto-instrument processes for authorized users or groups.

Change the default permissions:

1. Define an authorized group.

   ```
   groupadd appdynamics-zero-agent
   ```

2. Add users to the authorized group. For example, the `usermod` command adds a user named `tomcat-user` to the `appdynamics-zero-agent` group. Run this command for each user that creates Java processes you want the Agent Installer Platform to auto-instrument.

   ```
   usermod -a -G appdynamics-zero-agent tomcat-user
   ```

3. Navigate to the directory where the Agent Installer Platform was installed; the default directory is */*`opt/appdynamics/zeroagent`.

   ```
   cd <zero-agent-install-directory>
   ```

4. Change directories to the Agent Installer Platform's configuration file directory.

   ```
   cd configs
   ```

5. Change the group ownership of the `config.json` file to the authorized group created in Step 1.

   ```
   chgrp appdynamics-zero-agent config.json
   ```

6. Change the permissions associated with the `config.json` file.

   ```
   chmod 640 config.json
   ```

Configuration File Access

The Agent Installer Platform configuration file has the following access:

- Owning user - read and write
- Authorized group - read only
- All other users - no access

Unauthorized access to the file is not permitted. If a non-authorized user starts a Java process, it is not auto-instrumented by the Agent Installer Platform. However, it does not prevent the new Java process from starting.

Java Agent

Related pages:

- AppDynamics Essentials
- Install the Machine Agent
- Administer App Server Agents

To monitor Java applications in the Controller, you need to install the AppDynamics Java Agent on each server that hosts applications to be monitored. You can use the AppDynamics Agent Installer to streamline the deployment of the Java Agent. The Agent Installer simplifies the agent installation process.

If you downloaded the agent from the AppDynamics Downloads Center, see Install the Java Agent.

Before You Begin

1. Verify support for your application environment at Java Supported Environments.
2. Confirm you have access to a compatible controller. See Agent and Controller Compatibility.
3. Confirm the connection settings to the Controller where your agent will report data:
    - If you use a SaaS Controller, AppDynamics sent you the Controller hostname in your Welcome Email. Use port 443 for HTTPS or port 80 for HTTP.
    - If you use an on-premises Controller, you supplied hostname and port at install time.
4. Verify you have access to the machine where the application runs as a user account with privileges to install the agent software and restart the application.
5. Verify that the machine where the application runs can connect to the Controller. Proxies or firewalls on the network between the agent and Controller may require additional configuration.

Install the Agent

To install the agent using the **Agent Download Wizard** in the Controller UI:

1. Log in to the Controller UI and access the **Getting Started Wizard for Java**.
2. Follow the steps in the wizard to configure and download the agent. The agent guides you through some preliminary configuration steps.
3. As an administrator on the machine running your Java application, unzip the `AppServerAgent.zip` file.
   For example, on Linux unzip the agent to `home/appdynamics`. This is the `<agent_home>` directory.

   ```
   unzip AppServerAgent.zip -d /opt/appdynamics/appagent
   ```

4. Add the Java Agent binary to the application process, typically by passing the agent JAR file as a startup argument to the application. For information by framework and application type, see these pages:

When completed, you can restart your application and view data reported by the agent in the Controller UI. From there, you can install more agents or begin monitoring your application environment.

Tier and Node Naming in the Java Getting Started Wizard

Each monitored JVM must have a unique combination of tier and node name in AppDynamics. Naming the components in the wizard varies between the self-service trial edition of AppDynamics and the non-trial edition:

- For the self-service Pro Trial edition of AppDynamics Pro, the wizard names the application and tier for you using the format described in the following section. You can always change the names later.
- For a non-trial edition, you name the application and tier in the wizard, while the wizard generates the node name.

For self-service trials of AppDynamics Pro, the getting started wizard uses the following naming scheme to identify agents:

- Node name: `<app_server_type>@<hostname>:<port>`
  For example: `JBoss@appserver1.example.com:8080`
- Tier name: `MyTier`
- Application name: `MyApp`

The port number in the node name ensures that each node has a unique name if there is more than one app server on the same machine.

If the application server listens on multiple ports, the node name uses the lowest of the port numbers. Notice that the lowest port number may not be the primary port for the host. For instance, if a server listens for client requests at 8080 but listens for shutdown requests on port 8005, the node will be named with the 8005 port.

The Controller identifies distinct tiers based on traffic flow between nodes. All tiers belong to a single business application, MyApp.

Java Supported Environments

This page lists the application environments and versions supported by the AppDynamics Java Agent.

Java Agent Supported Platforms

In the following tables, note that:

- A dash ("-") in a table cell indicates that this column is not relevant or not supported for that particular environment.
- In cases where no version is provided, assume that all versions are supported. Contact AppDynamics Sales for confirmation.
- For environments that require additional configuration, a separate table describing or linking to configuration information follows the support matrix.
- For environments supported by AppDynamics End User Monitoring, see Supported Environments and Versions - Web EUM.
- For environments supported by AppDynamics Server Visibility, Machine Agent Requirements and Supported Environments.

JVM Support

The AppDynamics Java Agent uses the standard JVM Tool Interface (JVMTI) mechanism allowing it to instrument any software running on a JVM supporting this mechanism.

AppDynamics certifies the successful operation of the basic mechanisms of instrumentation used by the agent on the following Java runtimes. These capabilities are supported on both JRE or full JDK installations.

Where the agent supports the following advanced memory monitoring features, they are listed for the JVM: Object Instance Tracking (OIT), Automatic Leak Detection (ALD), Content Inspection (CI), and Access Tracking (AT).

JVM	OS	Memory Monitoring Features
AdoptOpenJDK 8, 9, 10, 11, 12, 13, 14, 15, 16 (supported for both Hotspot and OpenJ9 JVMs)	Linux, Windows, MacOS	OIT (supported only for Hotspot JVM), ALD, CI, AT
Amazon Corretto 8, 11	Linux, Windows	OIT, ALD, CI, AT
Azul Zing 15.x.	Linux x64	OIT, ALD
Azul Zulu OpenJDK 1.6, 1.7, 1.8, 9, 10, 11, 13, 14, 15, 16 JDK11 is supported from 4.5.6 onwards JDK13 is supported from 4.5.15 onwards JDK14 is supported from 20.4.0 onwards JDK15 is supported from 20.10.0 onwards JDK16 is supported from 21.4.0 onwards	Linux, Windows	OIT, ALD, CI, AT
GraalVM 20.0.0, 20.2.0, 21.1.0	Linux, Windows, MacOS	OIT, ALD, CI, AT
HP OpenVMS		
IBM JVM 1.6.x, 1.7.x, 1.8.x	AIX, HP-UX, Linux, Solaris, Windows, z/OS	ALD, CI Object instance tracking, automatic leak detection, and custom memory structure monitoring are not supported with the AppDynamics IBM Java Agent. IBM JVMs can be instrumented with the AppDynamics Sun Java Agent to work around this limitation; however, this only enables automatic leak detection and custom memory structure monitoring. Object instance tracking is not available. Working around this limitation can result in negative performance impact and is not recommended. In such cases, the IBM JVM needs to be restarted to enable custom memory structure monitoring.
Oracle Rockit JVM 28.1+	Linux Intel 64, Windows	

Oracle/BEA JRockit 1.6		
Oracle/Sun JVM 1.6, 1.7, 1.8, 9, 10, 11, 12, 13, 14, 15, 16 JDK11 is supported from 4.5.6 onwards JDK12 is supported from 4.5.11 onwards JDK13 is supported from 4.5.15 onwards JDK14 is supported from 20.4.0 onwards JDK15 is supported from 20.10.0 onwards JDK16 is supported from 21.4.0 onwards	Solaris Sparc 64, Windows, Linux	OIT, ALD, CI, AT Content Inspection and Access Tracking require a JVM restart.
Open Source OpenJDK 1.7, 1.8, 9, 10, 11, 12, 13, 14, 15, 16 OpenJDK11 is supported from 4.5.6 onwards OpenJDK12 is supported from 4.5.11 onwards JDK13 is supported from 4.5.15 onwards JDK14 is supported from 20.4.0 onwards JDK15 is supported from 20.10.0 onwards JDK16 is supported from 21.4.0 onwards	Solaris Sparc 64, Windows, Linux	OIT, ALD
SAP JDK 6+	Windows, Solaris, Linux, HP-UX, i5/OS, AIX	

JVM Application Server and Framework Support

AppDynamics supports the use of the Java Agent to instrument any application component running on a supported JVM, irrespective of how that component is built. The power of the AppDynamics platform is that it can automatically discover the topology and behavior of complex enterprise applications without requiring deep technical knowledge of the application's underlying code.

Frequently, Java-based systems employ standard framework code to implement business logic. Automatic instrumentation of framework code relies on knowledge of the business logic and programming patterns employed by the framework. AppDynamics instrumentation targets processing hand-offs between components, called entry points and exit points, either within the JVM or between JVMs. This includes hand-offs between frameworks in cases where multiple frameworks are being used together. This section covers the capabilities for frameworks for which AppDynamics provides automatic detection rules.

Monitoring application components built using frameworks not listed here may require custom configuration. The custom configuration may involve, for example, custom POJO entry or exit points. If you understand how the application behaves internally, you can easily configure this type of instrumentation. For more complex configuration tasks, contact your account representative to discuss how to engage the AppDynamics customer success organization.

JVM Language Frameworks Support

No additional configuration is required for these frameworks.

Vendor	JVM Language Framework	Version	Correlation/ Entry Points	Exit Points	Transports	Notes

Open Source	Akka Actor	2.1 – 2.5.x	Yes	Yes	Netty	4.3.1 required for 2.4.x 2.5x support includes Persistence Remoting exit/entry supported
Open Source	Akka HTTP Name: `akka-http-stream-entry-enabled` Type: Boolean Default: False	Akka Actor 2.5.x Akka HTTP upto 10.2.4 Scala 2.11, 2.12	Yes	Yes	HTTP	EUM is supported Support for Non-Route DSL
Open Source	Http4s Blaze Client	Blaze versions: 0.21.1, 0.21.0, 0.20.23, 0.20.5 scala 2.11, 2.12	No	Yes	HTTP	
Open Source	Groovy	-	Yes	Yes		
Open Source	Ktor	1.0.x -1.5.x	Yes (Netty Engine)	-	HTTP	EUM is supported
Open Source	Play for Scala Play for Java	2.1 – 2.8 Scala 2.11, 2.12	Yes	-	HTTP over Netty server Akka HTTP server	Includes framework specific entry and exit points Play EUM-APM correlation supported
Open Source	Scala	2.11.6				
Open Source	Spray toolkit (Spray.io)	1.1.x 1.1.3	Yes	Yes	HTTP	Entry points are detected and configurable as servlet entry point and exit points as HTTP exits
Pivotal	Grails	-	-	-	-	

Java Frameworks Support

The Java Agent supports these Java frameworks. Some require additional configuration as indicated in the Configuration Notes column.

Vendor	Framework	Version	SOA protocol (WebServices)	Auto Naming	Entry Points	Exit Points	Detection	Configuration Notes
Adobe	BlazeDS	-	HTTP and JMS adaptor	-	Yes		-	Example Message Queue Backend Configuration
Adobe	ColdFusion	8.x, 9.x	-	-	Yes	-	Configuration required for transaction discovery	Configuration is required for transaction discovery. See: • Java Business Transaction Detection • Servlet Entry Points
Apache	Cassandra with Thrift framework	-	-	-	Yes	Yes	Apache Thrift Entry and Exit points are detected	
Apache	Struts	1.x, 2.x	-	-	Yes		Struts Actions are detected as entry points; struts invocation handler is instrumented	Struts Entry Points
Apache	Tapestry	5	-	-	Yes	-	Not by default	See: • Java Business Transaction Detection • Servlet Entry Points
Apache	Wicket	-	-	No	Yes	-	Not by default	See: • Java Business Transaction Detection • Servlet Entry Points
Apple	WebObjects	5.4.3	HTTP	Yes	Yes	-	Yes	Apple WebObjects Startup Settings
axonframework.org	Axon	2.x, 3.x	-	-	Commands on the Command Bus continue existing Business Transactions	Correlation for Distributed Command Bus on JGroups and for Spring Cloud Connector transport as an exit		

Open Source	CometD	2.6	HTTP	Yes	Yes	-	-	See also "HTTP Exit Points" on Java Backend Detection.
Open Source	Spring Batch	-						Spring Batch Support
Eclipse	RCP (Rich Client Platform)	-	-	-	-	-	-	
Google	Google Web Toolkit (GWT)	2.5.1	HTTP	Yes	Yes	-	-	
JBoss	JBossWS Native Stack	4.x, 5.x	Native Stack	-	-	-	-	
IBM	IBM-BPM	8.5.7, 8.6	-	Yes	Yes	Yes	Yes	IBM-BPM Support
Open Source	Direct Web Remoting (DWR)	-	-	-	-	-	-	
Open Source	Eclipse Vert.x Core	3.3.3-3.5.4, 3.6.x, 3.7.x, 3.8.x, 3.9.x, 4.0.x	HTTP	Yes	Yes	Yes	Yes	EUM Correlation is supported
Open Source	Enterprise Java Beans (EJB)	2.x, 3.x	-	-	Yes	-	-	EJB Entry Points
Open Source	Grails	-	-	-	Yes	-	Not by default	
Open Source	Hibernate JMS Listeners	1.x	-	-	-	-	-	
Open Source	Java Abstract Windowing Toolkit (AWT)	-	-	-	-	-	-	
Open Source	Java Server Faces (JSF)	1.x, 2.x	-	Yes	Yes	-	-	Java Business Transaction Detection and Servlet Entry Points
Open Source	Java Server Pages	2.x	-	Yes	-	-	Yes	Servlet Entry Points
Open Source	Java Servlet API	2.x, 3.0	-	-	-	-	-	
Open Source	Jersey	1.x, 2.x	REST, JAX-RS	Yes	Yes	No	Not by default	JAX-RS Support and node properties: • `rest-num-segments` • `rest-transaction` • `rest-uri-segment-scheme` See App Agent Node Properties Reference for information on the properties.
Open Source	JRuby HTTP	-	-	-	Yes	-	Not by default	See: • Java Business Transaction Detection • Servlet Entry Points
Open Source	Micronaut	1.1.0, 2.5.3	-	Yes	Yes	Yes	By default	-
Open Source	Netty	3.x, 4.x	HTTP	Yes	Yes	Yes	By default	• Node property to disable Netty Instrumentation: `netty-enabled`, by default it is true
Open Source	Spring Annotated Web Services	2.x+	HTTP	Yes	Yes	No	-	
Open Source	Spring WebFlux	5.0, 5.1, 5.2, 5.3	HTTP	Yes	Spring Boot (Netty, Jetty, Tomcat, Undertow)	WebClient (Reactor Netty, Reactive Jetty)	By default	The node property `enable-webclient`, disables the Netty instrumentation and enables WebClient configuration. This node property should not be enabled unless there is some issue or loss in visibility with the OOTB support. By default, the value of this property is false.
Open Source	Spring Cloud Gateway	2.0.x, 2.1.x, 2.2.x, 3.0.2	HTTP	Yes	Yes	Yes	By default	
Open Source	WebSocket	1.0 (Java EE 7, JSR-356)	-	Yes, BT Naming not configurable	Yes, correlation not supported	Yes	Detection is automatic	Node property: websocket-entry-calls-enabled
Oracle	Coherence with Spring Beans	2.x, 3.x	-	-	-	-	-	
Oracle	Swing (GUI)	-	-	-	-	-	-	

Oracle	WebCenter	10.0.2,10.3.0	-	-	-	-	-	
Spring	Spring MVC	5.3	-	-	Yes	-	Not by default	See App Agent Node Properties Reference.

Application Servers

The Java Agent supports the following application servers. Some require additional configuration. Click the link on the server or OSGi Runtime for information about additional requirements or related configuration topics. The agent usually discovers application servers as an entry point.

Vendor	Application Server / OSGi Runtime	Version	SOA Protocol	RMI Supported	JMX	Entry Points	Configuration Notes
Adobe	Cold Fusion	8.x, 9.x	-	No	-	Yes	Requires configuration for transaction discovery; see Servlet Entry Points
	Equinox	-	-	-	-	Yes	OSGi Infrastructure Configuration
Apache	Felix	-	-	-	-	Yes	OSGi Infrastructure Configuration
Apache	Sling	-	-	-	-	Yes	OSGi Infrastructure Configuration
Apache	Tomcat	5.x, 6.x, 7.x, 8.x, 9, 10	-	-	Yes	Yes	Apache Tomcat Startup Settings
Apache	Resin	1.x - 4.x	-	-	-	-	Resin Startup Settings
Eclipse	Jetty	6.x, 7.x, 8x, 9x	-	-	-	-	Jetty Startup Settings
IBM	InfoSphere	8.x	-	-	-	Yes	IBM WebSphere and InfoSphere Startup Settings
IBM	WebSphere	6.1, 7.x, 8.x, 9.x	JAX-WS	Yes, detect and correlate	Yes for WebSphere PMI	Yes	IBM WebSphere and InfoSphere Startup Settings
Open Source	Liferay Portal	-	-	-	-	-	
Open Source	JBoss EAP	7.1.5 and 7.2.0		Yes		Yes	JBoss and Wildfly Startup Settings
Open Source	JBoss Wildfly (formerly JBoss Application Server)	4.x to 23.x		Yes		Yes	JBoss and Wildfly Startup Settings
Sun/Oracle	GlassFish Enterprise Server	2.x	-	-	Yes	Yes	GlassFish Startup Settings
Oracle	GlassFish Server and GlassFish Server Open Source Edition	3.x, 4.x	-	-	Yes for AMX	Yes	GlassFish Startup Settings
Oracle and BEA	WebLogic Server	9.x+	JAX-WS	Yes, detect and correlate for 10.x To enable correlation using a header transported in the `SOAP:Envelope` set node property enable-soap-header-correlation=true	Yes	Yes	Oracle WebLogic Startup Settings
Software AG	webMethods	9.5, 9.6	-	-	-	Yes	webMethods Startup Settings
Tibco	ActiveMatrix BusinessWorks Service Engine	5.x, 6.x	-	To enable correlation using a header transported in the `SOAP:Envelope` set node property `enable-soap-header-correlation=true`	-	Yes	Tibco ActiveMatrix BusinessWorks Service Engine Settings
	Application Server (OC4J)	-	-	Yes, detect and correlate for 10.x	-	Yes	
-	Grails, with Tomcat 7.x, Glassfish v3, Weblogic 12.1.1 (12c)	-	-	-	-		

Servlet 3.x detection is not supported.

PaaS Providers

PaaS Provider	Buildpack
Pivotal Cloud Foundry	Java Buildpack 3.4 and higher See Using AppDynamics with Java Applications on Pivotal Cloud for more information.
Red Hat Openshift 3	JBoss EAP 6.4 and 7.x WildFly 8.1 Docker images For documentation and download information, see the AppDynamics Java APM Agent page on the Red Hat Customer Portal.

Message Oriented Middleware Support

The Java Agent supports the following message oriented middleware environments. Some require additional configuration as indicated in the Configuration Notes column. Message oriented middleware servers are usually found by the Java Agent as an entry point.

Vendor	Messaging Server	Version	Protocol	Correlation /Entry Points	Exit Points	JMX	Configuration Notes
Amazon	Simple Queue Service (SQS)	-	-	Yes (correlation only)	Yes	-	See "Amazon Simple Queue Service Backends" on Java Backend Detection
Amazon	Simple Notification Service (SNS)	-	-	No	Yes	-	See "Amazon Simple Notification Service Backends" on Java Backend Detection
Apache	ActiveMQ	5.x+	JMS 1.x	Yes	Yes	Yes	
Apache	ActiveMQ	5.x+	STOMP	No	-	Yes	
Apache	ActiveMQ	5.8.x+	AMQP 1.0	No	-	Yes	Example Message Queue Backend Configuration
Apache	Axis	1.x, 2.x	JAX-WS	Yes	Yes	-	Default exclude rules exist for Apache Axis, Axis2, and Axis Admin Servlets. See aslo "Web Service Entry Points" on Java Backend Detection.
Apache	Apache CXF	2.1	JAX-WS	Yes	Yes	-	To enable correlation, set node property `enable-soap-header-correlation=true`.
Apache	Kafka	0.9.0.0 to 2.8.0	-	Yes	Yes	Yes	Kafka consumer entry points are disabled by default. Correlation is supported. See Apache Kafka Consumer Backends.
Apache	Synapse	2.1	HTTP	Yes	Yes	-	To enable correlation, set node property `enable-soap-header-correlation=true`
Fiorano	Fiorano MQ		-	-	-	-	
IBM	IBM Web Application Server (WAS)	6.1+, 7.x	Embedded JMS	-	Yes	-	Example Message Queue Backend Configuration
IBM	IBM MQ (formerly IBM WebSphere MQ)	6+	JMS	Yes	Yes	-	Example Message Queue Backend Configuration
Mulesoft	Mule ESB	3.4, 3.6, 3.7, 3.8, 3.9, 4.1.x, 4.2.0, 4.2.1	HTTP, JMS	Yes	Yes	-	Mule ESB Startup Settings
Open Source	Eclipse Vert.x verticles	3.3.x, 3.4.x, 3.5.0, 3.6.0	-	Yes (correlation only)	Yes	-	The Java Agent detects messaging exit calls between verticles.
Open Source	Open MQ	-	-	-	-	-	
Oracle	Java Message Service	2.0	JMS	Correlation of the listener is disabled by default	Yes		
Oracle	Oracle AQ	-	JMS	-	Yes	-	
Oracle	OSB deployed on WebLogic	12.2.1	HTTP JMS	Yes	Yes		OSB Support
Oracle / BEA	WebLogic	9.x+	JMS 1.1	Yes	Yes	Yes	Oracle WebLogic Startup Settings
Progress	SonicMQ	-	-	-	-	-	
Pivotal	RabbitMQ	-	HTTP	-	Yes	-	See "RabbitMQ Backends" on Java Backend Detection
Rabbit	RabbitMQ Spring Client	-	-	Yes	Yes	-	See "RabbitMQ Backends" on Java Backend Detection
Red Hat	HornetQ (formerly JBoss Messaging and JBoss MQ)	-				Yes	
Red Hat	JBoss A-MQ	4.x+	-	-	-	Yes	
Spring	Spring Integration	2.2.0+, 4.0+	JMS	Yes	Yes	Yes	Spring Integration Support See also "Java Message Service Backends" on Java Backend Detection
WSO2	ESB	4.7.0	-	Yes	Yes	-	EUM Correlation is not supported
WSO2	API Microgateway	3.1.x, 3.2.0, 3.2.1	HTTP1	Yes	Yes	-	See WSO2 API Microgateway Startup Settings

JDBC Drivers and Database Servers Support

The Java Agent supports these JDBC driver and database server environments. AppDynamics can follow transactions using these drivers to the designated database.

JDBC Vendor	Driver Version	Driver Type	Database Server	Database Version

Apache	10.9.1.0	Embedded or client	Derby	-
Apache	-	-	Cassandra	-
Progress	DataDirect	data connectivity for ODBC and JBDC driver access, data integration, and SaaS and cloud computing solutions	-	-
IBM	JDBC 3.0 version 3.57.82 or JDBC 4.0 version 4.7.85	DB2 Universal JDBC driver	DB2	9.x
IBM	JDBC 3.0 version 3.66.46 or JDBC 4.0 version 4.16.53	DB2 Universal JDBC driver	DB2	10.1
IBM	-	Type IV	Informix	-
Maria				1.4.x - 2.6.x
Microsoft	4	Type II	MS SQL Server	2012
Oracle MySQL, MySQL Community	5.x, 6.x, 8.x	Type II, Type IV	MySQL	5.x
Oracle	RAC			
Oracle	9i, 10g 11g, 12c, 18c, 19c	Type II, Type IV	Oracle Database	8i+
Open Source PostgreSQL	42.2.5	Type IV	Postgres	8.x, 9.x, 11x
Sybase	jConnect	Type IV	Sybase	-
Teradata			Teradata	-

Notes:

- Type II is a C or OCI driver
- Type IV is a thin database client and is a pure Java driver

NoSQL/Data Grids/Cache Servers Support

The Java Agent supports these NoSQL, data grids and cache server environments. Some require additional configuration. Click the link on the database, data grid or cache name in the following support matrix for information about additional configuration required or related configuration topics.

Vendor	Database/Data Grid /Cache	Version	Correlation/Entry Points	JMX	Configuration Notes
Amazon	DynamoDB	-	Exit Points	-	See "Amazon Web Services" on Java Backend Detection.
Amazon	Simple Storage Service (S3)	-	-	-	"Amazon Simple Storage Service Backends" on Java Backend Detection.
Apache	Casandra • DataStax drivers • Thrift drivers	1.x, 2.x	Correlation for Thrift drivers only	Yes	• "Cassandra Backends" on Java Backend Detection. • For Cassandra server-side support, see Apache Cassandra Startup Settings
Apache	Lucene - Apache Solr	1.4.1	Entry Points	Yes	Apache Solr Startup Settings
Couchbase	Couchbase	3.x	Exit Points	-	See "Couchbase Backends" on Java Backend Detection
JBoss	Cache TreeCache	-	-	-	JBoss Startup Settings
JBoss	Infinispan	5.3.0+	Correlation	-	-
Open Source	Memcached	-	-	-	Memcached Exit Points
Open Source	MongoDB Async Driver	3.4-3.12	-	-	See "MongoDB Backends" on Java Backend Detection
Open Source	MongoDB Sync Driver	3.1-3.12, 4.0.x	-	-	See "MongoDB Backends" on Java Backend Detection
Open Source	MongoDB Reactive Streams Driver	1.3-1.13, 4.0.x	-	-	See "MongoDB Backends" on Java Backend Detection
Oracle	Coherence	3.7.1	Custom-Exit	Yes	Coherence Startup Settings
Red Hat	JBoss DataGrid	-	-	-	JBoss Startup Settings
	JBoss Cache TreeCache	-	-	-	
	JBoss Infinispan	5.3.0+	Correlation	-	

Terracotta	EhCache	-	-	-	EhCache Exit Points

RPC/Web Services API/HTTP Client Support

The Java Agent supports these RPC, web services or API framework types. Some require additional configuration as indicated in the Configuration Notes column.

Vendor	RPC/Web Services API Framework /HTTP Client Support	Version	SOA Protocol-WebServices	Auto Naming	Correlation/Entry Points	Exit Points	Configurable BT Naming Properties	Detection	Configuration Notes
Apache	Apache CXF	2.1	JAX-WS	Yes	Yes	Yes	Yes	Yes	
Apache	Apache HTTP Client	-	HTTPClient (now in Apache HTTP Components)	Yes	Yes (correlation only)	Yes	-	Yes	See "HTTP Backends" on Java Backend Detection
Apache	Apache Async HTTP Client	4.1.x	-	-	-	-	-	-	-
Apache	Ribbon HTTP Client	2.1.0	HTTP Client	Yes	Yes (correlation) Entry - NA	Yes	NA	Yes	
Apache	Apache Thrift	-	-	Yes	Yes	Yes	Yes	Yes	Binary Remoting Entry Points for Apache Thrift
Eclipse	Jetty	8.x, 9.x	HTTP Client	Yes	Yes (correlation only)	Yes (ART supported)	-	Yes	See "HTTP Backends" on Java Backend Detection
Google	gRPC	1.6.x to 1.37.x	RPC	Yes	Yes (Asynchronous)	Yes (Asynchronous)	ServiceName /MethodName	Yes	See Web Service Backend
IBM	WebSphere	6.x, 7.x, 8.x	JAX-RPC	-	-	-	-	-	IBM WebSphere and InfoSphere Startup Settings; also see Default configuration excludes WebSphere classes
IBM	Websphere	7.x, 8.x	IIOP	-	-	-	-	-	IBM WebSphere and InfoSphere Startup Settings; also see Default configuration excludes WebSphere classes
IBM	Websphere	6.1, 7.x, 8.x, 9.x	JAX-WS	Yes	Yes, detect and correlate. To enable correlation using a header transported in the `SOAP:Envelope` set node property `enable-soap-header-correlation=true`	-	Web Service naming	Yes	-
JBoss		7,8,11,16, and 18	JAX-WS	Yes	Yes, detect and correlate. To enable correlation using a header transported in the `SOAP:Envelope` set node property `enable-soap-header-correlation=true`		Web Service naming	Yes	To detect Web Service entry and to support correlation you must create a Servlet exclude rule. See Web Service Entry Points to exclude a rule for JBoss.
Open Source	`java.net.Http`	-	HTTP	Yes	-	Yes	Yes	Yes	See "HTTP Backends" on Java Backend Detection.
Open Source	HTTPClient	0.3-3	Oracle SOA (and potentially others that embed this library)	-	Correlation: Yes; Entry: No	Yes	-	Yes	Oracle WebLogic Startup Settings; also see Default configuration excludes WebLogic classes

Open Source	Grizzly	Grizzly Async HTTP Client (`com.ning.http-client` 1.8.x, 1.9.x, grizzly-http-client 1.1x) • NingAsyncClient v1 with NettyProvider, GrizzlyProvider • NingAsyncClient v2 with NettyProvider	HTTP	-	Correlation: Yes; Entry:No	Yes	-	Yes	
Oracle	GlassFish Metro	-	JAX-WS	-	-	-	-	-	
Oracle	GlassFish Metro with Grails	-	JAX-WS	-	Yes	-	-	Not by Default	
Oracle	JAX-WS RI	2.3.1	JAX-WS	-	To enable correlation using a header transported in the `SOAP:Envelope` set node property en`able-soap-header-correlation=true`	-	Web Service naming	Yes	-
Spring WS	Web Services	3.x, 4.x, and 5.x	HTTP, SOAP	-	To enable correlation using a header transported in the `SOAP:Envelope set node property enable-soap-header-correlation=true`	-	Web Service naming	Yes	-
Oracle	Oracle Application Server	ORMI	-	no	-	-	-	-	
Oracle	WebLogic	10.x	T3, IIOP	Yes	Correlation: Yes; Entry: No	Yes	-	Yes	
Oracle	WebLogic	9.x, 10.x	JAX-RPC	-	-	-	-	-	
Oracle/Sun	Java	11	-	-	-	Yes (ART supported)	-	Yes	
Oracle/Sun	Sun RMI	-	IIOP	-	Not by Default	-	-	-	
Oracle/Sun	Sun RMI	-	JRMP	-	No	Yes	host/port	Yes	
Red Hat	JBoss A-MQ	4.x+	RMI	Yes	Yes	Yes	Yes	Yes	JBoss and Wildfly Startup Settings
Square	OkHttp	2.x, 3.x, 4.x (upto 4.22)	HTTP	Yes	Correlation: Yes Entry: No	Yes	-	Synchronous (2.x, 3.x, and 4.x upto 4.22) and Asynchronous (3.x and 4.x upto 4.22)	
-	Web Services	-	SOAP over HTTP	-	Yes	Yes	-	-	Create Match Rules for Web Services "Web Service Entry Points" on Java Backend Detection
jersey.github.io	Reactive JAX-RS client API	2.25+	HTTP Client	Yes	Yes (correlation) Entry – NA	Yes	NA	Yes	"Web Service Entry Points" on Java Backend Detection

Business Transaction Error Detection

The Java Agent supports the following logging frameworks for business transaction error detection:

- Apache `Log4j` and `Log4j 2`
- `java.util.logging`
- Simple Logging Facade for Java (SLF4J)
  Support for the following method has been added: `public void error(String format, Object... argArray)`
- Logback

To instrument other types of loggers, see Error Detection.

Install the Java Agent

Related pages:

- Download AppDynamics Software
- Java Agent Configuration Properties
- Enable SSL for the Java Agent
- SELinux Installation Issues

To begin monitoring a Java application using AppDynamics, you install the AppDynamics Java Agent into the application JVM:

1. Download the agent distribution to the machine where your Java application runs.
2. Configure the Java Agent settings. This configuration section of this page describes manual configuration. For more options, see Administer the Java Agent.
3. Add the agent to the JVM process.

Prepare to Install the Java Agent

Consider the following as you decide where and how to deploy the AppDynamics Java Agent:

- The user under which the JVM runs must have write privileges to the conf and logs directories in the Java Agent home (privileges to the conf *(*<JAVA_AGENT_HOME>/ver<VERSION>/*)*. To achieve this, you can install the agent as the same user that owns the JVM or as an administrator on the host machine. You can restrict the remaining contents of the agent directory to read-only access.
- To secure communications between the Java Agent and your Controller, see Enable SSL for the Java Agent.
- If the agent connects to the Controller through a local proxy, you must configure proxy settings for the agent. See "Proxy Properties for the Controller" in Java Agent Configuration Properties.
- The Java Agent supports sharing a single agent between multiple JVMs running on the same machine. If you choose this deployment scenario, you will want to specify settings in a combination of system properties and the versioned configuration file. See Instrument Multiple JVMs on a Single Machine for example configurations.

For cURL installation, see Download AppDynamics Software.

The AppDynamics Java Agent is one type of bytecode injection (BCI) agent. To prevent unforeseen issues with other BCI agents, AppDynamics only supports environments running the Java Agent as the sole BCI agent on the JVM.

You can run multiple BCI Agents on the JVM at your own risk. For Java Agent <= 4.5.16, you can decrease the chances of conflict with other agents by specifying the following command-line option:

```
-Dappdynamics.agent.prefer.retransformClasses=true
```

Java Agent >= 4.5.17 automatically enables this option if they detect the presence of other agents.

Instrument a Custom Java Runtime Image

You can use the Java Agent to instrument an application running on a custom Java runtime image constructed with J-link. To instrument the agent, the custom runtime requires these modules:

- `jdk.management`
- `java.logging`
- `jdk.unsupported`

`jdk.jcmd` is not a required module for the custom runtime. However, it is required for the proper functioning of Object Instance Tracking. See Object Instance Tracking for Java.

AppDynamics assumes that all packages from Java.se are installed.

Java Agent Resource Overhead

The Java Agent typically adds between 0% to 2% additional CPU consumption.

However, certain factors can increase CPU overhead from the agent beyond 2%. These include the use of resource-intensive AppDynamics features, such as asynchronous transaction tracking. Very active environments or configuration settings that result in a high number of metrics or snapshots reported per minute can also affect agent resource consumption.

In all cases, AppDynamics recommends that you test the agent in a staging environment, and monitor resource consumption of your application to ensure that it remains within proper operating parameters.

If your application operates within a small margin of its existing memory resource allocation, you may choose to increase the allocation for the application. AppDynamics recommends allocating the following amounts of additional Heap and PermGen space to accommodate the agent:

- Maximum heap size (`-Xmx`): 100 MB in addition to the amount required by the application

- Maximum PermGen (permanent generation) heap size (`-XX:MaxPermSize`): 20 MB in addition to the amount required by the application. The setting for MaxPermSize is applicable only for Java 6 or Java 7.
- For network bandwidth consumption, see Install App Server Agents.

Download and Unzip the Java Agent Distribution

You can get the agent from the Agent Download Wizard. If you have never installed an agent before, the wizard is a good place to start. The wizard populates the configuration file in the agent you download with Controller connection settings and identifying settings for the agent. After you download the agent, you can install it in the JVM.

Alternatively, you can download the agent manually, as follows:

1. Download the Java Agent ZIP file from AppDynamics Download Center.
2. Extract the ZIP file to the destination directory as the same user or administrator of the JVM. Note the following:
    - Extract the Java Agent to a directory that is outside of your container or application server runtime directories, such as to `\usr\local\appdynamics\appagent`.
    - All files should be readable by the user under which the JVM runs. The user must have write privileges to the `conf` and `logs` directories in the Java Agent home. One way to achieve this is to install the agent as the same user that owns the JVM or as an administrator on the host machine.
    - The application server's runtime directory should be writable by the Java Agent as well.

For information on the contents of the Java Agent home directory, see Java Agent Directory Structure.

Configure the Java Agent

If you downloaded the agent from the Agent Download Wizard in the Controller, you can skip to the next section, as the agent is already configured.

To configure the settings manually (or verify the wizard settings):

1. Edit the versioned configuration file:
   `<`*`agent_home`*`>/<version_`*`number`*`>/conf/controller-info.xml`
   The controller-info.xml is one of several approaches available for supplying configuration settings. For others, see Java Agent Configuration Properties.
2. Modify the connection settings to the Controller:
    - `controller-host`: Set to the IP address or hostname of the Controller. If the agent needs to connect through a proxy, see "Proxy Properties for the Controller" in Java Agent Configuration Properties.
    - `controller-port`: Set to the primary listening port number on the Controller. By default:
        - For a SaaS Controller, use 80 for HTTP or 443 for HTTPS
        - For an on-premises Controller, use 8090 for HTTP or 8181 for HTTPS
3. Direct the agent to connect to the Controller by SSL (HTTPS) by setting the `controller-ssl-enabled` value to true. See Enable SSL for the Java Agent.
4. Identify the business application, tier, and node that this the monitored JVM belongs to in the AppDynamics application model using these settings:
    - `application-name`
    - `tier-name`
    - `node-name`

   In a self-service Trial edition of AppDynamics Pro, the agent uses a default naming scheme, see Java Agent. You can use automatic naming with a standard edition of AppDynamics Pro by adding this property:
   `<auto-naming>true</auto-naming>`
5. If the agents connects to a SaaS Controller or other multi-tenant Controller, configure the Account Name. For all Controllers, configure the Account Access Key.
    - `account-name`
    - `account-access-key`

   This information is provided in the Welcome email from the AppDynamics Team when you acquired the Controller. For a multi-tenant on-premises Controller, you can find this information in
   `<`*`controller_home`*`>/initial_account_access_info.txt`
6. See Java Agent Configuration Properties and configure any additional properties required in your environment.

The following shows a `controller-info.xml` file with sample configuration values:

```
<controller-info>
    <controller-host>192.168.1.20</controller-host>
    <controller-port>8090</controller-port>
    <application-name>ACMEOnline</application-name>
    <tier-name>InventoryTier</tier-name>
    <node-name>Inventory1</node-name>
</controller-info>
```

Load the Java Agent in a JVM

After configuring the agent settings, you can add the agent to the JVM. The exact steps for doing so vary by framework. The general approach involves specifying the agent as a `javaagent` argument to the startup command for the JVM.

Ensure to add the `-javaagent` argument before the `-jar` argument.

The argument should indicate the location of the Java Agent JAR file:

```
-javaagent:<agent_home>/javaagent.jar
```

On Windows, include the drive letter in the path to the agent:

```
-javaagent:C:<agent_home>\javaagent.jar
```

Adding `javaagent` to the startup script requires a restart of the JVM. If it's not possible to restart the JVM when you are installing the agent and modifying the JVM start up script, you can attach the agent dynamically to the running Java process.

See Agent Installation by Java Framework for more information on how to install Java Agent by Java framework or technology.

Attach the Java Agent to a Running JVM Process

Attaching the agent to a running JVM allows you to install the Java Agent without requiring a JVM restart. This approach would normally be used alongside adding the `javaagent` argument to the JVM startup script, or some other persistent approach to ensure that the agent is loaded again at the next JVM restart. However, dynamic attachment allows you to install the agent when restarting the JVM is not possible or convenient.

Dynamic agent attachment works if:

- JVM is => 1.6.
- JVM is an Oracle (HotSpot) JVMs (unavailable for IBM or JRockit JVMs).

Other considerations include:

- Do not attach the agent dynamically to an environment that is already instrumented (either by the AppDynamics Java Agent or another type of agent). Doing so can cause unforeseeable issues and errors.
- Attaching the AppDynamics Java Agent to a running environment will impact the performance of the application while the agent performs the class retransformation needed to instrument the application. The agent overhead will return to its normal operating level when it finishes the process, but it is important to consider the potential performance impact to production services.

To attach the agent to the JVM:

1. Determine the PID of the JVM to which you want to attach.
   For Linux, use:

   ```
   ps -A | grep java
   ```

   On Windows, use:

   ```
   jps -l
   ```

2. Run the following command, replacing the placeholders for the path to the `tools.jar` file in your JDK, path to the AppDynamics Java Agent home directory, and the JVM process ID with values appropriate for your environment:

   ```
   java -Xbootclasspath/a:<path_to_jdk>/lib/tools.jar -jar /<agent_home>/javaagent.jar <jvm_process_id>
   appdynamics.controller.hostName=<controller_hostname>,appdynamics.controller.port=<controller_port_no>,
   appdynamics.controller.ssl.enabled=false,appdynamics.agent.applicationName=<app_name>,appdynamics.agent.
   tierName=<agent_tier_name>,appdynamics.agent.nodeName=<agent_node_name>
   ```

   Use the equivalent paths for Windows, including drive letter. The following shows an example with system output included:

   ```
   [appduser@my_centos6 ~]$ ps -A | grep java
    6780 pts/1    00:00:04 java
   [appduser@my_centos6 ~]$ java -Xbootclasspath/a:/usr/java/jdk1.7.0_79/lib/tools.jar -jar /home/appduser
   /appagent/javaagent.jar 6780
   Attaching to VM [6780]
   agent path >>>/home/appduser/appagent/javaagent.jar=
   ```

Verify Java Agent Installation

After an installation, the agent log in `<agent_home>/logs` will contain this message:

```
Started AppDynamics Java Agent Successfully
```

If the agent log file is not present, the Java Agent may not be accessing the `javaagent` command properties. To troubleshoot, check the application server log file where `STDOUT` is logged. It will have the fallback log messages, useful for troubleshooting the agent.

Also, verify that the agent is able to connect to the Controller in the Controller UI. To verify, log in to the Controller UI and click the **Settings** icon at the top right of the page, and then **AppDynamics Agents**. In the list, search for the agent by machine hostname.

Install the Java Agent in Containers

This page explains how to install the Java Agent in a containerized environment. See Container Installation Options to for your options. There are three options to install the Java Agent with containers:

- Use Auto-Instrumentation
- Use Init Containers
- Use a Dockerfile

Use Auto-Instrumentation

> This scenario applies to containers running in Kubernetes where the Cluster Agent is installed.

This option uses the Auto-Instrumentation feature of the Cluster Agent. It is the recommended option because it offers the highest level of automation and the simplest operational experience for instrumenting Java applications in a Kubernetes cluster.

To get started, see Auto-Instrument Applications with the Cluster Agent.

Use Init Containers

> This scenario applies to containers running in Kubernetes.

This option uses Kubernetes init containers to copy the agent binaries into the application container. It is the recommended option when Auto-Instrumentation is not possible. In this use case, it assumes there are two containers:

- Application image is built without any Java Agent binaries.
- Second init container image is built and only contains the Java Agent binaries.

The deployment spec for the application is configured to copy the agent binaries to the running application container. To use init containers to copy the agent binaries:

1. Build the Java Application Image
2. Build the Java Agent Init Container Image
3. Redirect Agent Output to Stdout
4. Add the Init Container to the Deployment Spec
5. Set the Java Agent Environment Variables
6. Add the -javaagent Argument to the Deployment Spec
7. (OpenShift only) Set the APPDYNAMICS_AGENT_UNIQUE_HOST_ID Environment Variable
8. (On-Premises Controller only) Copy the Controller Certificates to the Container

Build the Java Application Image

When building the Java application image, do not include the Java Agent binaries.

Build the Java Agent Init Container Image

The Java Agent image is built separately from the application image and can be reused across multiple Java application deployments. This Dockerfile is an example of building the Java Agent init container image using a multi-stage build. Alternatively, the init container can reference a pre-built image from AppDynamics on Docker Hub.

Redirect Agent Output to Stdout

It is a best practice to redirect agent output to `stdout` in containerized environments. If you build your own Java Agent init container image, you can redirect the Java agent output to `stdout` by updating the `ver<version>/conf/logging/log4j2.xml` file that is provided in the download package. Add two `<AppenderRef ref="Console"/>` elements as shown in the example log4j2.xml file. If you use a pre-built image from AppDynamics on Docker Hub, you can replace the default `ver<version>/conf/logging/log4j2.xml` file based on the example log4j2.xml file using a ConfigMap. Follow the same procedure shown for Copy the Controller Certificates to the Container.

Add the Init Container to the Deployment Spec

Edit the deployment spec to add the required sections that allow you to copy the agent binaries from the init container to the application image.

The following snippet from a deployment spec shows the required `volumes`, `volumeMounts`, and `initContainer` definitions. The code example assumes the Java application image is published to `myrepo/java-app:v1` and the init container image uses the pre-built image `docker.io/appdynamics/java-agent:20.6.0`:

```
kind: Deployment
spec:
  containers:
  - name: java-app
    image: myrepo/java-app:v1
    volumeMounts:
    - mountPath: /opt/appdynamics
      name: appd-agent-repo
  initContainers:
  - command:
    - cp
    - -r
    - /opt/appdynamics/.
    - /opt/temp
    name: appd-agent
        image: docker.io/appdynamics/java-agent:20.8.0
    volumeMounts:
    - mountPath: /opt/temp
      name: appd-agent-repo
  volumes:
    - name: appd-agent-repo
      emptyDir: {}
```

Set the Java Agent Environment Variables

To set all of the required Java Agent environment variables, you must follow these steps as explained in Best Practices to Configure Agents in Kubernetes.

- Use ConfigMaps to Configure the App Server Agent
- Use Secrets for the Controller Access Key
- Set Application-Specific Configuration in the Deployment Spec

Use ConfigMaps to Configure the App Server Agent

1. Use a ConfigMap to set the Java Agent environment variables that are shared across applications in a namespace:

   ```
   apiVersion: v1
   data:
     APPDYNAMICS_AGENT_APPLICATION_NAME: "eCommerce"
     APPDYNAMICS_AGENT_ACCOUNT_NAME: "<value>"
     APPDYNAMICS_CONTROLLER_HOST_NAME: "<value>"
     APPDYNAMICS_CONTROLLER_PORT: "<value>"
     APPDYNAMICS_CONTROLLER_SSL_ENABLED: "<value>"
     APPDYNAMICS_JAVA_AGENT_REUSE_NODE_NAME: "true"
     APPDYNAMICS_JAVA_AGENT_REUSE_NODE_NAME_PREFIX: "<value>"
   kind: ConfigMap
   metadata:
     name: ecommerce-java-config
   ```

2. Apply the ConfigMap to the namespace:

   ```
   kubectl -n ecommerce apply -f ecommerce-java-config.yaml
   ```

3. Update the deployment spec to reference the ConfigMap:

   ```
   spec:
     containers:
     - name: java-app
       envFrom:
       - configMapRef:
         name: ecommerce-java-config
    ...
   ```

Use Secrets for the Controller Access Key

1. Create a Secret using `kubectl`:

```
kubectl -n ecommerce create secret generic appd-agent-secret --from-literal=access-key=<access-key>
```

2. Update the deployment spec to reference the Secret:

```
spec:
  containers:
  - name: java-app
    env:
    - name: APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY
      valueFrom:
      secretKeyRef:
        name: appd-agent-secret
        key: access-key
 ...
```

Set Application-Specific Configuration in the Deployment Spec

Set the application-specific tier name environment variable APPDYNAMICS_AGENT_TIER_NAME in the deployment spec:

```
spec:
  containers:
  - name: java-app
    env:
    - name: APPDYNAMICS_AGENT_TIER_NAME
      value: ecommerce-service
```

Add the `-javaagent` Argument to the Deployment Spec

Edit the deployment spec to add the `-javaagent` argument to the application container startup command as shown in the example:

```
spec:
  containers:
    command: ["/bin/sh"]
    args: ["-c", "java -javaagent:/opt/appdynamics/javaagent.jar -jar /myapp.jar"]
```

Note that for some Java frameworks such as Spring Boot, you can leverage the standard `JAVA_TOOL_OPTIONS` environment variable to include the `-javaagent` argument.

(OpenShift Only) Set the `APPDYNAMICS_AGENT_UNIQUE_HOST_ID` Environment Variable

For Java applications running in OpenShift, set the `APPDYNAMICS_AGENT_UNIQUE_HOST_ID` environment variable to enable APM correlation with the Cluster Agent. Since the `APPDYNAMICS_AGENT_UNIQUE_HOST_ID` value depends on a runtime value, set this environment variable in the container startup command.

For example, for an OpenShift 3.10 or 3.11 environment, set the environment variable as shown:

```
spec:
  containers:
    command: ["/bin/sh"]
    args: ["-c", "APPDYNAMICS_AGENT_UNIQUE_HOST_ID=$(sed -rn '1s#.*/##; 1s/docker-(.{12}).*/\\1/p' /proc/self
/cgroup) && java -javaagent:/opt/appdynamics/javaagent.jar -jar /myapp.jar"]
```

For your use case, see values documented in Manually Configure App Agents to Correlate with the Cluster Agent.

(On-Premises Controller Only) Copy the Controller Certificates to the Container

If on-premises Controller certificates are required, define a ConfigMap to reference the `cert` file and use a `volume mount` in the deployment spec to mount the ConfigMap contents to the container.

```
$ kubectl create configmap appd-cert --from-file=cacerts.jks
```

Add the cert file, in this example, `appd-cert`, to the container file system using `volumes` and `volumeMounts` as shown in the deployment spec snippet:

```
kind: Deployment
spec:
  containers:
    image: myrepo/java-app:v1
    volumeMounts:
    - name: appd-cert
      subPath: cacerts.jks
      mountPath: /opt/appdynamics/ver4.5.11.26665/conf/cacerts.jks
  volumes:
    - name: appd-cert
      configMap:
        name: appd-cert
```

Example Configuration for Using an Init Container

A complete example of a deployment spec that uses an init container to copy the agent binaries can be found on Github: `java-app.yaml`.

Use a Dockerfile

This scenario applies to containers running in Docker and Kubernetes.

This option uses a Dockerfile to copy the Java Agent to the Docker image at build time. To use the option, create a single image that contains both the application and Java Agent binaries.

To copy the agent into the application image during the Docker image build:

1. Download and Unzip the Java Agent
2. Redirect Agent Output to Stdout
3. Copy the Agent Binaries to the Image
4. Set the Java Agent Environment Variables
5. Add the -javaagent Argument to the Startup Command
6. (OpenShift only) Set the APPDYNAMICS_AGENT_UNIQUE_HOST_ID Environment Variable
7. (On-Premises Controller only) Copy the Controller Certs to the Image

Download and Unzip the Java Agent

Download the Java Agent from or programmatically download the agent. The agent is packaged as a zip file. In a shell, unzip the AppServer Agent to the `AppServerAgent` directory:

```
$ unzip AppServerAgent-<version>.zip  -d AppServerAgent
```

Redirect Agent Output to Stdout

It is a best practice to redirect agent output to `stdout` in containerized environments. To redirect the output of the Java Agent to `stdout`, edit the ver<version>/conf/logging/`log4j2.xml` file and add two `<AppenderRef ref="Console"/>` elements as shown in the example log4j2.xml file.

Copy the Agent Binaries to the Image

Edit the Dockerfile to copy the unpackaged agent binaries to the target folder:

```
COPY AppServerAgent/ /opt/appdynamics/
```

Set the Java Agent Environment Variables

If you are running the application in a non-Kubernetes environment (using `docker run` for example), set the agent environment variables in the Dockerfile. For example:

```
ENV APPDYNAMICS_AGENT_APPLICATION_NAME=<value>
ENV APPDYNAMICS_AGENT_TIER_NAME=<value>
ENV APPDYNAMICS_AGENT_ACCOUNT_NAME=<value>
ENV APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY=<value>
ENV APPDYNAMICS_CONTROLLER_HOST_NAME=<value>
ENV APPDYNAMICS_CONTROLLER_PORT=<value>
ENV APPDYNAMICS_CONTROLLER_SSL_ENABLED=<value>
ENV APPDYNAMICS_JAVA_AGENT_REUSE_NODE_NAME=true
ENV APPDYNAMICS_JAVA_AGENT_REUSE_NODE_NAME_PREFIX=<value>
```

For Kubernetes applications, omit these environment variables from the Dockerfile and set them using ConfigMaps and Secrets as described in Best Practices to Configure Agents in Kubernetes.

The reuse node name and prefix environment variables are required to support unique naming for multiple container instances for the same application image. See Reuse Node Name.

Add the `-javaagent` Argument to the Startup Command

Edit the Dockerfile to include the `-javaagent` argument in the image start command based on the location where the agent binaries were copied. For example:

```
ENV JAVA_OPTS -javaagent:/opt/appdynamics/javaagent.jar
CMD ["java", "$JAVA_OPTS", "-jar", "/myapp.jar"]
```

Note that for some Java frameworks such as Spring Boot, the existing `JAVA_TOOL_OPTIONS` environment variable can be leveraged to include the `-javaagent` argument.

(OpenShift Only) Set the `APPDYNAMICS_AGENT_UNIQUE_HOST_ID` Environment Variable

For Java applications running in OpenShift, set the `APPDYNAMICS_AGENT_UNIQUE_HOST_ID` environment variable to enable APM correlation with the Cluster Agent. Since the value depends on a runtime value, set this environment variable in the image startup script using the values documented in Manually Configure App Agents to Correlate with the Cluster Agent. For example, for an OpenShift 3.10 or 3.11 environment, add this startup script `startup.sh` to the Docker image:

```
#!/bin/bash
# OpenShift 3.10 or 3.11:
APPDYNAMICS_AGENT_UNIQUE_HOST_ID=$(sed -rn '1s#.*/##; 1s/docker-(.{12}).*/\\1/p' /proc/self/cgroup)
exec java $JAVA_OPTS -jar /myapp.jar
```

and update the startup command in the Dockerfile to reference it:

```
CMD ["/startup.sh"]
```

(On-Premises Controller Only) Copy the Controller Certs to the Image

For Java Agents communicating with an on-premises Controller, edit the Dockerfile to copy the cert file containing the on-premises certs to the agent `conf` folder. For example:

```
COPY ./onprem-cacerts.jks /opt/appdynamics/ver4.5.11.26665/conf/cacerts.jks
```

See Enable SSL for the Java Agent.

Example Configuration for Using a Dockerfile

This Dockerfile builds a single image with the application and agent binaries included and this Kubernetes deployment spec references that Docker image. This Dockerfile uses a multi-stage build and downloads and unzips the agent.

Example Log4j Configuration File

This updated `ver<version>/conf/logging/log4j2.xml` configuration file redirects the Java Agent logs to stdout. It adds `<AppenderRef ref="Console"/>` to the Root element.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="INFO" monitorInterval="2">
   <Appenders>
      <Console name="Console" target="SYSTEM_OUT">
         <PatternLayout pattern="[%t] %d{ABSOLUTE} %5p %c{1} - %m%n" />
      </Console>
      <ADRRAFAppender name="DefaultAppender" fileName="agent.log">
         <PatternLayout pattern="[%t] %d{DATE} %5p %c{1} - %m%n" />
         <SizeBasedTriggeringPolicy size="20 MB" />
         <ADRolloverStrategy max="5" />
         <RegexFilter regex=".*REST.*" onMatch="DENY" onMismatch="ACCEPT" />
      </ADRRAFAppender>
      <ADRRAFAppender name="BCTAppender" fileName="ByteCodeTransformer.log">
         <PatternLayout pattern="[%t] %d{DATE} %5p - %m%n" />
         <SizeBasedTriggeringPolicy size="20 MB" />
         <ADRolloverStrategy max="5" />
      </ADRRAFAppender>
      <ADRRAFAppender name="RESTAppender" fileName="REST.log">
         <PatternLayout pattern="[%t] %d{DATE} %5p %c{1} - %m%n" />
         <RegexFilter regex=".*REST.*" onMatch="ACCEPT" onMismatch="DENY" />
         <SizeBasedTriggeringPolicy size="20 MB" />
         <ADRolloverStrategy max="5" />
      </ADRRAFAppender>
      <ADRRAFAppender name="DynamicServiceAppender" fileName="dynamic-service.log">
         <PatternLayout pattern="[%t] %d{DATE} %5p %c - %m%n" />
         <SizeBasedTriggeringPolicy size="20 MB" />
         <ADRolloverStrategy max="5" />
      </ADRRAFAppender>
      <ADRRAFAppender name="BusinessTransactionsLogger" fileName="BusinessTransactions.log">
         <PatternLayout pattern="[%t] %d{DATE} %5p - %m%n" />
         <SizeBasedTriggeringPolicy size="20 MB" />
         <ADRolloverStrategy max="5" />
      </ADRRAFAppender>
      <ADRRAFAppender name="DataPipelineAppender" fileName="data-pipeline.log">
         <PatternLayout pattern="[%t] %d{DATE} %5p %c - %m%n" />
         <SizeBasedTriggeringPolicy size="20 MB" />
         <ADRolloverStrategy max="5" />
      </ADRRAFAppender>
      <ADRRAFAppender name="APIAppender" fileName="api.log">
         <PatternLayout pattern="[%t] %d{DATE} %5p %c - %m%n" />
         <SizeBasedTriggeringPolicy size="20 MB" />
         <ADRolloverStrategy max="5" />
      </ADRRAFAppender>
   </Appenders>
   <Loggers>
      <!--  to control the logging level of the agent log files, change "level" attribute. level="
all|trace|debug|info|warn|error"-->
      <AsyncLogger name="com.singularity" level="info" additivity="false">
         <AppenderRef ref="DefaultAppender" />
         <AppenderRef ref="RESTAppender" />
      </AsyncLogger>
      <AsyncLogger name="com.singularity.BusinessTransactions" level="info" additivity="false">
         <AppenderRef ref="BusinessTransactionsLogger" />
      </AsyncLogger>
      <AsyncLogger name="com.singularity.dynamicservice" level="info" additivity="false">
         <AppenderRef ref="DynamicServiceAppender" />
      </AsyncLogger>
      <AsyncLogger name="com.singularity.ee.service.datapipeline" level="info" additivity="false">
         <AppenderRef ref="DataPipelineAppender" />
      </AsyncLogger>
      <AsyncLogger name="com.singularity.datapipeline" level="info" additivity="false">
         <AppenderRef ref="DataPipelineAppender" />
      </AsyncLogger>
      <AsyncLogger name="com.singularity.BCTLogger" level="info" additivity="false">
         <AppenderRef ref="BCTAppender" />
      </AsyncLogger>
      <AsyncLogger name="com.singularity.api" level="info" additivity="false">
         <AppenderRef ref="APIAppender" />
      </AsyncLogger>
      <AsyncLogger name="com.singularity.segment.TxnTracer" level="info" additivity="false">
         <AppenderRef ref="DefaultAppender" />
```

```
      </AsyncLogger>
      <Root level="error">
         <AppenderRef ref="DefaultAppender" />
         <AppenderRef ref="Console" />
      </Root>
   </Loggers>
</Configuration>
```

Agent Installation by Java Framework

These pages describe individual considerations and instructions for installing the Java Agent for some of the application servers supported for the Java Agent.

Apache Cassandra Startup Settings

The Java Agent bootstraps using the javaagent command line option. Add this option to the `cassandra` (Linux) or `cassandra.bat` (Windows) file.

Instrument Cassandra in a Windows Environment

1. Open the `apache-cassandra-x.x.x\bin\cassandra.bat` file.
2. Add the Java Agent `javaagent` path to the `JAVA_OPTS` variable. Make sure to include the drive in the full path to the Java Agent directory.

   ```
   -javaagent:<agent_home>\javaagent.jar
   ```

   For example:

   ```
   set JAVA_OPTS=-ea
       -javaagent:C:\appdynamics\agent\javaagent.jar
       -javaagent:"%CASSANDRA_HOME%\lib\jamm-0.2.5.jar
       . . .
   ```

3. Restart the Cassandra server for the changes to take effect.

Instrument Cassandra in a Linux Environment

1. Open the `apache-cassandra-x.x.x/bin/cassandra.in.sh` file.
2. Add the `javaagent` argument at the top of the file:

   ```
   JVM_OPTS=-javaagent:<agent_home>/javaagent.jar
   ```

   For example:

   ```
   JVM_OPTS=-javaagent:/home/software/appdynamics/agent/javaagent.jar
   ```

3. Restart the Cassandra server for the changes to take effect.

Apache Tomcat Startup Settings

To instrument applications on Apache Tomcat, add the Java Agent JAR location as a Catalina environment, or `CATALINA_OPTS` variable.

Instrument Apache Tomcat

1. Open (or create if it doesn't already exist) the `setenv.sh` (Linux) or `setenv.bat` (Windows) file. For Tomcat 6 and later, you can put the file in the `CATALINA_BASE/bin` directory. For previous versions of Tomcat, put the file in the `CATALINA_HOME/bin` directory
2. Add the `-javaagent` argument to the file as a Catalina environment variable, as follows:

   On Linux:

   ```
   export CATALINA_OPTS="$CATALINA_OPTS -javaagent:<agent_home>/javaagent.jar"
   ```

   Replace <agent_home> with the full path to the Java Agent JAR file.
   For example:

   ```
   export CATALINA_OPTS="$CATALINA_OPTS -javaagent:/home/appserver/appagent/javaagent.jar"
   ```

   On Windows:

   ```
   set CATALINA_OPTS=%CATALINA_OPTS% -javaagent:"Drive:<agent_home>\javaagent.jar"
   ```

   For example

   ```
   set CATALINA_OPTS=%CATALINA_OPTS% -javaagent:C:\appagent\javaagent.jar
   ```

3. Restart the application server for the changes to take effect. For more information on running Tomcat as a service, see Setting Properties and Options on Startup.

Instrument Tomcat When Running as a Windows Service

When running Tomcat as a Windows service, add the `javaagent` argument to your Tomcat startup properties. These instructions apply to Apache Tomcat >= 6.x.

To install the Java agent in Tomcat running as a Windows service:

1. Ensure that you are using administrator privileges.
2. Run the Apache `tomcat<version>w.exe` utility to configure your tomcat service to load the appdynamics agent (where `<version>` represents the major version number of the tomcat being instrumented).

   For example:

   ```
   tomcat8w //ES// <servicename>
   ```

3. Click the **Java** tab and in the **Java Options** add:

   ```
   -javaagent:"<agent_home>\javaagent.jar"
   ```

4. Restart the Tomcat service to have the changes take effect.

Apple WebObjects Startup Settings

This page describes how to instrument applications written with WebObjects 5.4.3 on OSX 10.9 systems.

We will use one of the developer examples to illustrate how to instrument an application created with Apple WebObjects. After installing WebObjects, you can find most of the artifacts in the following directories:

```
/Developer/Examples/JavaWebObjects
/Developer/Applications/WebObjects
```

When you run the `HelloWorld` application at `/Developer/Examples/JavaWebObjects/HelloWorld`, a script file is generated:

```
/Developer/Examples/JavaWebObjects/HelloWorld/dist/legacy/HelloWorld.woa/HelloWorld
```

Open the generated script file to edit. Towards the end of the file, line 310 in the following example, appears the Java execute line:

```
304 #
305 # Launch the application.
306 #
307 echo Launching ${SCRIPT_NAME}.woa ...
308
309 echo ${JAVA_EXECUTABLE} ${JAVA_EXECUTABLE_ARGS} -classpath WOBootstrap.jar com.webobjects._bootstrap.WOBootstrap ${COMMAND_LINE_ARGS}
310 eval exec ${JAVA_EXECUTABLE} ${JAVA_EXECUTABLE_ARGS} -classpath WOBootstrap.jar com.webobjects._bootstrap.WOBootstrap ${COMMAND_LINE_ARGS}
~
~
~
~
```

Add the standard Java Agent arguments to the Java execution script for the `HelloWorld` application:

```
307 echo Launching ${SCRIPT_NAME}.woa ...
308
309 echo ${JAVA_EXECUTABLE} ${JAVA_EXECUTABLE_ARGS} -classpath WOBootstrap.jar com.webobjects._bootstrap.WOBootstrap ${COMMAND_LINE_ARGS}
310 eval exec ${JAVA_EXECUTABLE} ${JAVA_EXECUTABLE_ARGS} -Dappdynamics.controller.hostName=localhost \
311 -Dappdynamics.controller.port=8080 \
312 -Dappdynamics.agent.accountName=customer1 \
313 '-Dappdynamics.agent.accountAccessKey=SJ5b2m7d1\$354' \
314 -Dappdynamics.controller.ssl.enabled=false \
315 -Dappdynamics.agent.applicationName=play-correlation \
316 -Dappdynamics.agent.uniqueHostId=play-correlation \
317 -Dappdynamics.agent.tierName=standalone-tier \
318 -Dappdynamics.agent.nodeName=standalone-node \
319 -javaagent:/Users/akilman/Work/codebase/agent/java/core/build/temp/javaagent.jar \
320 -agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=5006 \
321 -classpath WOBootstrap.jar com.webobjects._bootstrap.WOBootstrap ${COMMAND_LINE_ARGS}
NORMAL  /Developer/Examples/JavaWebObjects/HelloWorld/dist/legacy/HelloWorld.woa/HelloWorld
```

Note: Single quotes and extra escape to account for 'eval'

You can configure business transaction name using getter-chains. For more information, see

- Using Getter Chains
- See "Split by POJO Method Call" on Split Servlet Transaction by Payload Examples

Coherence Startup Settings

To add the `javaagent` command in Oracle Coherence:

1. In the `<coherence_home>/bin/cache-server.sh` file, update the following:

```
$JAVAEXEC -server -showversion $JAVA_OPTS -javaagent:<agent_home>/javaagent.jar -cp "$COHERENCE_HOME/lib
/coherence.jar" com.tangosol.net.DefaultCacheServer $1
```

2. Restart the application server to have the changes take effect.

GlassFish Startup Settings

Oracle GlassFish is an OSGi-based application container. This page describes how to set configure class bootloader settings and load the Java Agent in the runtime JVM.

Instrument Oracle GlassFish

1. Add the location of the Java Agent JAR file as a `-javaagent` JVM option in the `GlassFish` domain. You can add the agent using the `asadmin` tool as follows:

   For Windows:

   ```
   glassfish4\bin\asadmin.bat create-jvm-options '-javaagent:<Drive_letter>:<agent_home>\javaagent.jar'
   ```

   For Linux:

   ```
   glassfish4\bin\asadmin create-jvm-options "-javaagent\:/home/appduser/javaagent/javaagent.jar"
   ```

   Ensure you escape the colon character on Linux.
2. Configure the boot delegation entry for the AppDynamics package:
   - In GlassFish 3.x to 3.1.2, open the `config.properties` file located at `<glassfish_home>/glassfish/osgi/felix/conf` and add the following package prefix to the `config.properties` file:
     `org.osgi.framework.bootdelegation=com.singularity.*`
   - In GlassFish => 3.1.2, in the boot delegation list in the `<glassfish_home>/glassfish/config/osgi.properties` file, add `com.singularity.*`
     For example:

     ```
     eclipselink.bootdelegation=oracle.sql, oracle.sql.*

     # There is no need to use bootdelegation except for the following issues:
     # 1. EclipseLink
     # 4. NetBeans profiler packages exist in parent class loader (see issue #8612)
     # 5. BTrace exists in bootclasspath.
     org.osgi.framework.bootdelegation=${eclipselink.bootdelegation}, \
                                       com.sun.btrace, com.sun.btrace.*, \
                                       org.netbeans.lib.profiler, org.netbeans.lib.profiler.*, \
                                       com.singularity.*

     # The OSGi R4.2 spec says boot delegation uses the boot class loader by default. We need
     # to configure it to use the framework class loader because that class loader is
     ```
3. Restart the application server.

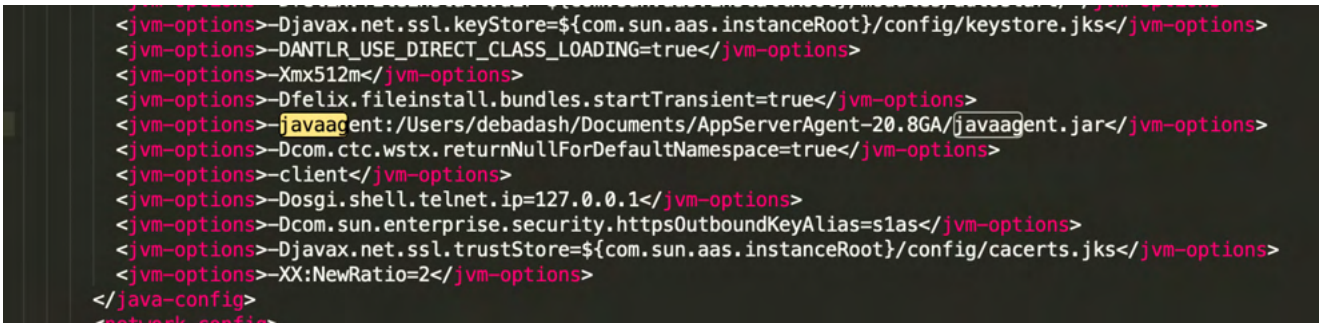
To verify the configuration, review the domain.xml file located at `<glassfish_home>\domains\<domain_name>\config`. The `domain.xml` file should have an entry for the `-javaagent` option.

Instrument Oracle GlassFish with the Admin Console

1. Start the GlassFish application server.
2. Navigate to the GlassFish admin console (`http://localhost:4848/`).
3. Navigate to **Configurations > server-config > JVM Settings**.
4. Click the **JVM Option**s tab and click **Add JVM Option** to add a blank field in the list.
5. Add the following path of the `javaagent.jar` file in the blank field:

   ```
   -javaagent:/home/appduser/javaagent/javaagent.jar
   ```

6. Click **Save** on the right top corner.
7. Restart the GlassFishapplication server and `<glassfish_home>\domains\<domain_name>\config`.
   The `domain.xml` file must contain an entry for the `-javaagent` option.



About Glassfish AMX Support

AppDynamics supports Glassfish AMX MBeans.

Set the `boot-amx` node property to enable AMX MBeans. See boot-amx.

You will see the AMX domain in the MBean Browser in the **JMX** tab of the node dashboard.

IBM WebSphere and InfoSphere Startup Settings

The Java Agent passes the `-javaagent` argument command line option to WebSphere to ensure the App Server Agent has the correct permissions to monitor your app.

Security Requirements and Configuration

Full permissions are required for the agent to function correctly with WebSphere. Grant all permissions on both the server level and the profile level.

Run WebSphere with Security Enabled

To run WebSphere while J2EE security or Global security is enabled, you must make changes to WebSphere's `server.policy` file to prevent problems with the interaction between WebSphere and the Java Agent.

A `codeBase` value indicates the location of the source code. You must grant permissions to the code from that location. The `codeBase` is a URL value, and depends on the characters at the end. A `codeBase` with a trailing `"/-"` matches all files, both class and JAR files, in the directory and recursively all files in subdirectories contained in the directory.

1. Navigate to the `server.policy` file, located in *<webshere_home>*`/properties` or in *<webshere_profile_home>*`/properties`.
2. Add this code block to the WebSphere `server.policy` file:

Syntax

```
grant codeBase "file:</full/path/to/agent_install_directory>/-" { permission java.security.
AllPermission; };
```

Example

```
grant codeBase "file:/opt/appdynamics/javagent/agent4.5.1.23676/-" { permission java.security.
AllPermission; };
```

3. Save the file.

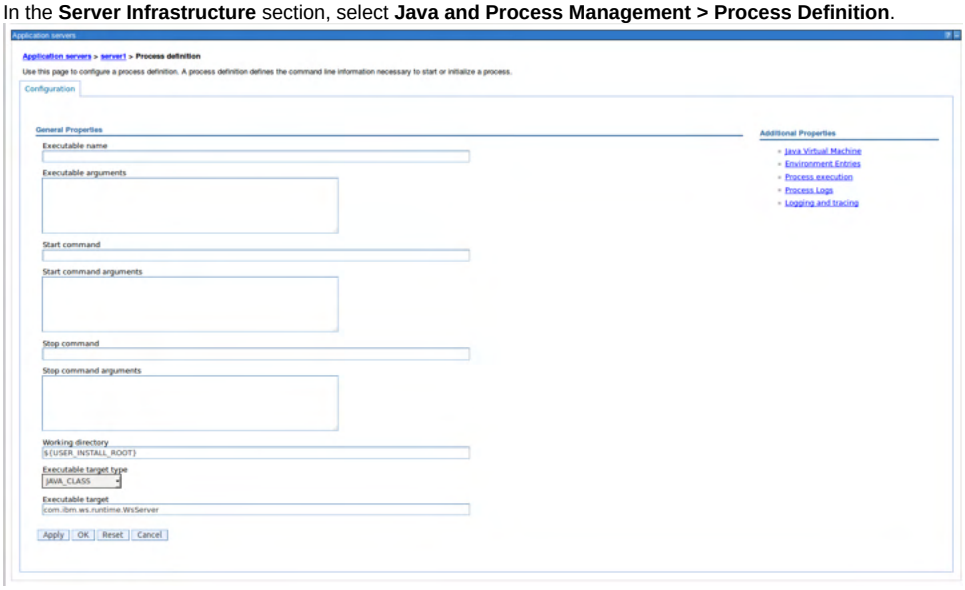
Performance Monitoring Statistic

Under **Application servers >** `<server_name>` **> Performance Monitoring Infrastructure (PMI)**, set a **Currently monitored statistic set** to an option other than **None** for the JMX functionality to work.

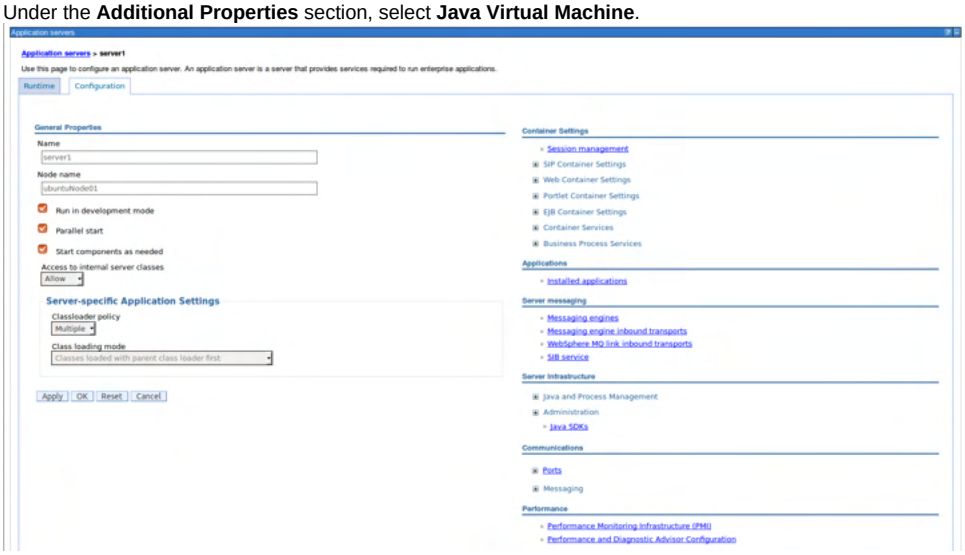
Instrument WebSphere 7.x, 8.x, and 9.x, or InfoSphere 8.x

1. Log in to the administrative console for the WebSphere node where you want to install the App Server Agent.
2. In the administrative console, select **Servers > Server Types > WebSphere application servers**.
3. Select the name of your server.

4. In the **Server Infrastructure** section, select **Java and Process Management > Process Definition**.



5. Under the **Additional Properties** section, select **Java Virtual Machine**.



6. Enter the `javaagent` option with the full path to the AppDynamics `javaagent.jar` file in the **Generic JVM** arguments field.

For Windows:

```
-javaagent:<Drive Letter>:<agent install location>\javaagent.jar
```

For Linux:

```
-javaagent:<agent install location>/javaagent.jar
```

7. Click **OK**.

WebSphere uses Equinox as its OSGi container. In some cases, you may also need to add the Java agent packages to the OSGi `bootdelegation` system property:

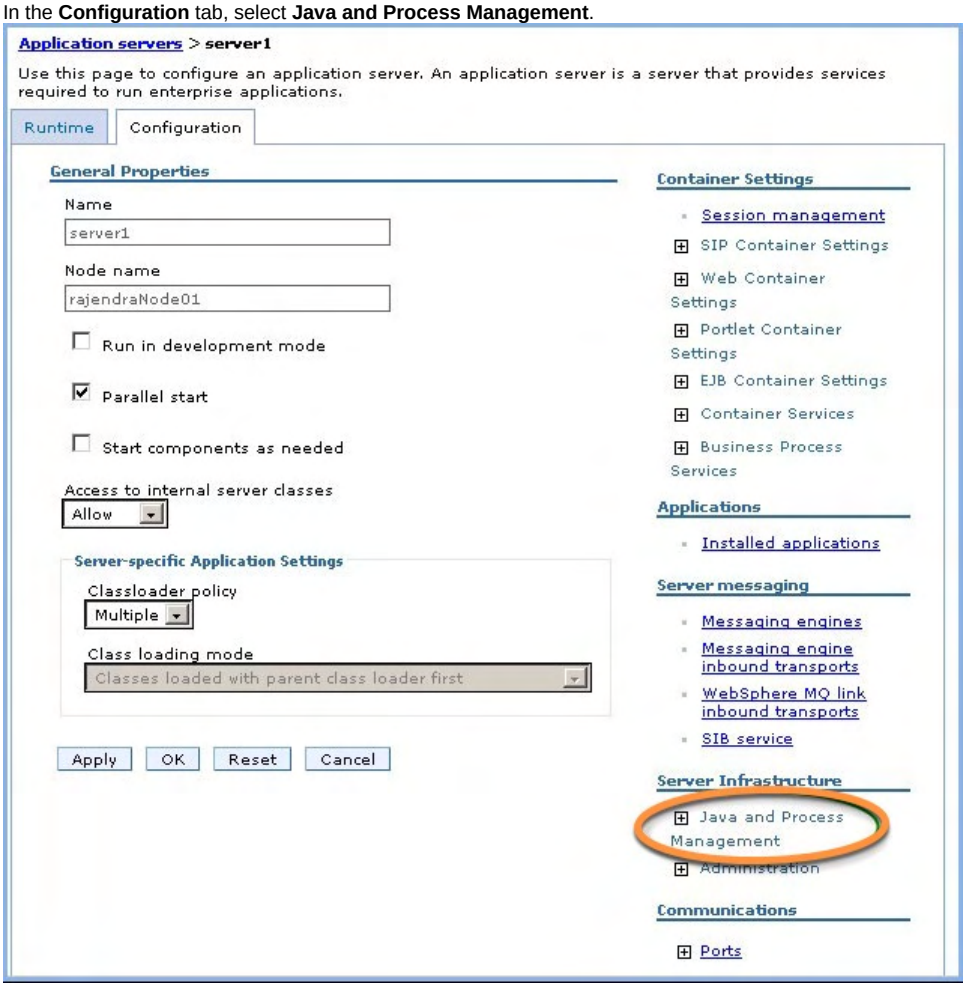
```
-Dorg.osgi.framework.bootdelegation=META-INF.services,com.singularity.*,com.ibm.*
```

Instrument WebSphere 6.x

1. Log in to the administrative console for the WebSphere node where you want to install the Java Agent.
2. In the left navigation tree, select **Servers > Application servers**.

3. Click the name of your server in the list of servers.

4. In the **Configuration** tab, select **Java and Process Management**.



5. Enter the `javaagent` option with the full path to the Java Agent `javaagent.jar` file in the **Generic JVM** arguments field.

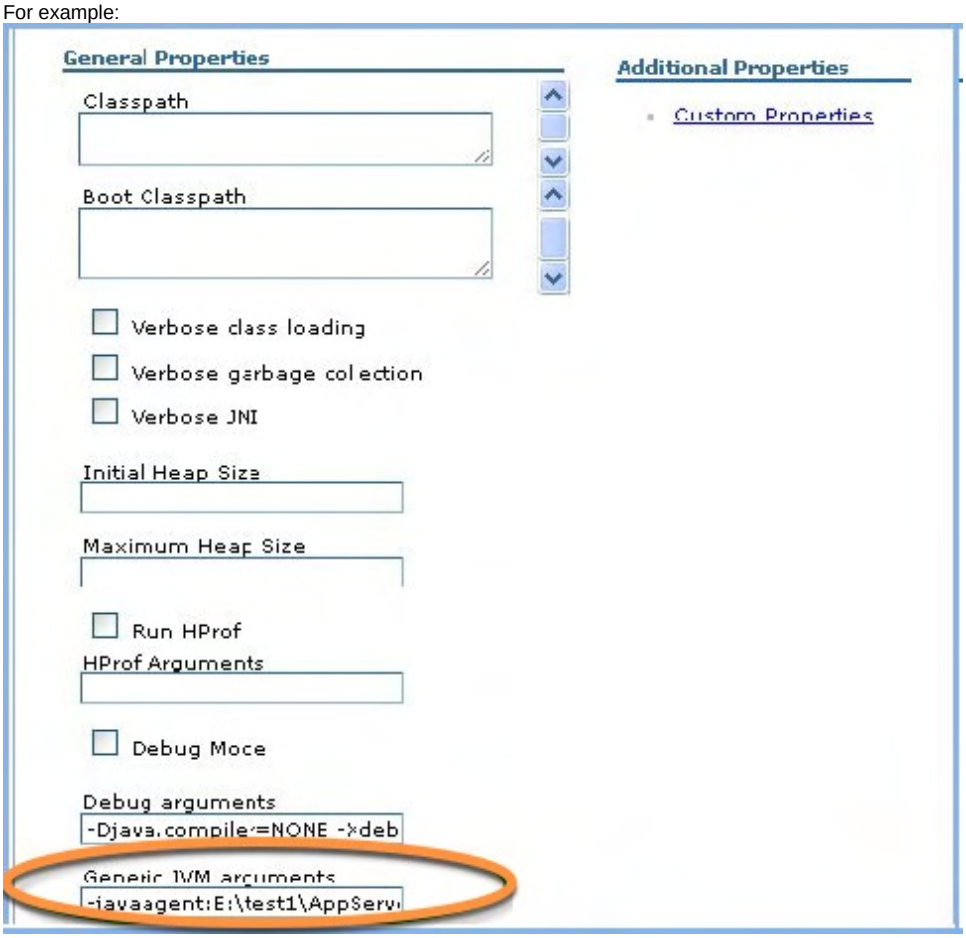
   For Windows:

   ```
   -javaagent:<Drive Letter>:<agent install location>\javaagent.jar
   ```

   For Linux:

   ```
   -javaagent:<agent install location>/javaagent.jar
   ```

For example:



6. Click **OK**.

Instrument WebSphere 5.x

1. Log in to the administrative console of the WebSphere node where you want to install the App Server Agent.
2. In the administrative console, click **Servers**.
3. Click **Application Servers**.
4. Click the name of your server.
5. Under **Additional Properties**, click **Process Definition**.
6. On the next page, under **Additional Properties**, click **Java Virtual Machine**.
7. Enter the `javaagent` option with the full path to the Java Agent `javaagent.jar` file in the **Generic JVM** arguments field.

   For Windows:

   ```
   -javaagent:<Drive Letter>:<agent install location>\javaagent.jar
   ```

   For Linux:

   ```
   -javaagent:<agent install location>/javaagent.jar
   ```

8. Click **OK**.

Verify the Java Agent Configuration

Verify the configuration settings by checking the `server.xml` file of the WebSphere node where you installed the Java Agent. The `server.xml` file should have this entry:

```
<jvmEntries ... genericJvmArguments='-javaagent:E:\test1\AppServerAgent\javaagent.jar' disableJIT="false"/>
```

JBoss and Wildfly Startup Settings

This page describes how to install the AppDynamics Java Agent on Red Hat JBoss Enterprise Application Server and JBoss Wildfly.

Before You Install

To install the agent on JBoss or Wildfly, you add the Java Agent and log manager packages to the server startup routine.

The configured location varies based on your framework version:

- In Linux, add the settings to `standalone.conf` or `standalone.sh`.
- In Windows, add the settings to `standalone.conf.bat`.
- If using JBoss 4.x or 5.x, add the configuration to `run.sh` for Linux or `run.bat` for Windows.

Initialize the JVM

To install the Java Agent on JBoss EAP or JBoss Wildfly, you need to initialize the JVM. Run this parameter:

```
JAVA_OPTS="$JAVA_OPTS -Djboss.modules.system.pkgs=org.jboss.byteman,com.singularity"
```

If you do not initialize the JVM, the installation generates a "class not found" exception.

Standalone Mode Agent Installation

These instructions assume the use of Red Hat JBoss (Application Server 7.x or Enterprise Application Server >= 6.x).

To install the Java Agent on JBoss Standalone:

Dynamic LogManager Location

On standalone JBoss instances, instead of hard coding the path and name of the log manager JAR, you can use glob pattern matching techniques to make the path to the log manager file dynamic, so that it is resilient to change or variances among systems.

The exact steps to accomplish this varies by environment. These sections provide an example of this configuration on Windows and Linux systems, and are meant as a starting point for your own implementation.

Windows

In Windows, the `standalone.conf.bat` gets this additional snippet:

```
...
rem jboss.modules.system.pkgs
set JAVA_OPTS=%JAVA_OPTS% -Djboss.modules.system.pkgs=org.jboss.byteman,com.singularity,org.jboss.logmanager

rem java.util.logging
set JAVA_OPTS=%JAVA_OPTS% -Djava.util.logging.manager=org.jboss.logmanager.LogManager

rem bootclasspath
set LOGMANAGER=
for /f %%i in ('dir /b "%JBOSS_HOME%\modules\system\layers\base\org\jboss\logmanager\main\jboss-logmanager-*.
jar"') do (
   set LOGMANAGER_JAR=%JBOSS_HOME%\modules\system\layers\base\org\jboss\logmanager\main\%%i
)
set JAVA_OPTS=%JAVA_OPTS% -Xbootclasspath/p:%LOGMANAGER_JAR%
```

The path to the LogManager JAR file under the JBoss home can vary by JBoss version. Be sure to check your system and adjust the path as shown in the example accordingly.

Linux

In Linux, you can populate the path dynamically with the following code:

```
JBOSS_MODULES_SYSTEM_PKGS ="org.jboss.byteman,com.singularity,org.jboss.logmanager"

JAVA_OPTS="$JAVA_OPTS -Djava.util.logging.manager=org.jboss.logmanager.LogManager"
JAVA_OPTS="$JAVA_OPTS -Xbootclasspath/p:$(ls ${JBOSS_HOME}/modules/system/layers/base/org/jboss/logmanager/main
/jboss-logmanager-*.jar)"
```

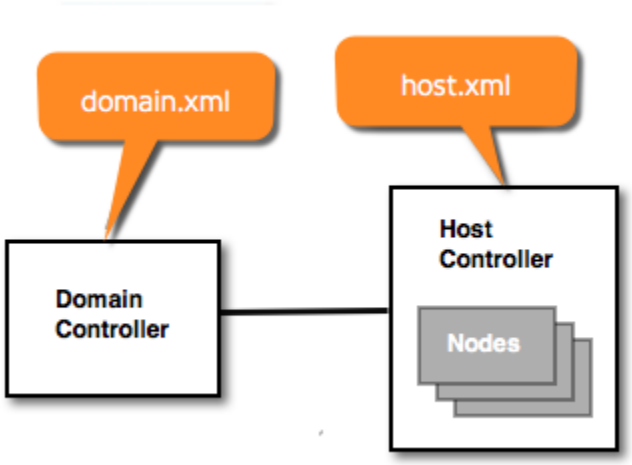
If using the `${JBOSS_HOME}` variable, as in the example, be sure to set the variable to the directory to the JBoss installation directory on your system.

The path to the `LogManager` JAR file under the JBoss home can vary by JBoss version. Be sure to check your system and adjust the path as shown in the example accordingly.

Domain Mode Agent Installation

For domain mode, the location in which you need to configure the settings depends upon specifics for your environment. Keep in mind that a domain is made up of these components:

- A domain controller, the administration and configuration server. The domain.xml configuration file is the global configuration for the managed hosts.
- Host controllers, which manage a particular host containing one or more application server nodes. There can be any number of host controllers and nodes. The `hosts.xml` file contains settings for the nodes on that host machine.



Where you put the configuration, therefore, varies as follows:

- `domain.xml`: Settings that can be identical for all hosts in the managed domain (i.e., the log manager and agent files are at the same location on all machines) can go into the domain.xml file for the Domain Controller.
- `host.xml`: Settings that need to be specialized for individual hosts (for example, if the paths to relevant files on hosts vary) need to go into the `host.xml` file.

You can add configuration settings to both `domain.xml` and `host.xml`, depending on which are global and which are host-specific. The following sections show an example of adding general settings to the domain configuration and node name setting to the host configuration.

Domain.xml Configuration

1. Locate and edit `domain.xml` for the domain. This is usually located under $JBOSS_HOME/domain/configuration/.
2. Find the the `system-properties` element and add a property named `jboss.modules.system.pkgs` with a value of `com.singularity` to the existing system properties. For example:

   ```
   <system-properties>
          <!-- IPv4 is not required, but setting this helps avoid unintended use of IPv6 -->
          <property name="java.net.preferIPv4Stack" value="true"/>
          <property name="jboss.modules.system.pkgs" value="com.singularity"/>
   </system-properties>
   ```

   This property tells the JBoss class loader to load the AppDynamics packages. This is required for the Java Agent to run.
3. Under the server group name where you want to enable your agents, add the JVM options using the appropriate values for your agent location, JBoss application name, and tier name.

```
<server-group name="main-server-group" profile="full">
    <jvm name="default">
         <heap size="1303m" max-size="1303m"/>
         <permgen max-size="256m"/>
         <jvm-options>
            <option value="-javaagent:<agent_install_dir>/javaagent.jar"/>
            <option value="-Dappdynamics.agent.applicationName=JBOSS-EAP-APP"/>
            <option value="-Dappdynamics.agent.tierName=JBOSS-EAP-TIER"/>
         </jvm-options>
    </jvm>
    <socket-binding-group ref="full-sockets"/>
 </server-group>
```

Changes to the `domain.xml` file require a restart of the management host to take effect. The changes are not propagated to server hosts until they are restarted as well.

Host.xml Configuration

For each host, specify the AppDynamics node name in the `host.xml` file, usually located under `$JBOSS_HOME/domain/configuration/`.

Add the `-Dappdynamics.agent.nodeName` JVM option to specify the node name for this instance:

```
<servers>
    <server name="server-one" group="main-server-group">
            <jvm name="default">
              <jvm-options>
                <option value="-agentlib:jdwp=transport=dt_socket,address=8787,server=y,suspend=n"/>
                <option value="-Dappdynamics.agent.nodeName=JBOSS-EAP-NODE-1"/>
              </jvm-options>
            </jvm>
     </server>
     <server name="server-two" group="main-server-group" auto-start="true">
            <socket-bindings port-offset="150"/>
            <jvm name="default">
              <jvm-options>
                <option value="-Dappdynamics.agent.nodeName=JBOSS-EAP-NODE-2"/>
              </jvm-options>
            </jvm>
      </server>
      <server name="server-three" group="other-server-group" auto-start="false">
            <socket-bindings port-offset="250"/>
      </server>
</servers>
```

For brevity, comments have been removed from the sample.

Troubleshooting JBoss Startup Issues

Most issues installing the Java Agent on JBoss result from conflicts between startup arguments. That is, settings you add for the Java Agent may be overridden or conflict in other ways with existing arguments. Such issues are not always easy to detect.

The best way to begin troubleshoot such startup issues is to print and inspect the startup arguments that JBoss actually gets when attempting to start. To do this, view JBoss process information during startup using the following command. You need to issue this command while the start up attempt is occurring but before it fails.

```
ps -ef | grep [o]rg.jbossas | tr ' ' '\n' | sed -e '/^$/d'
```

Troubleshooting JBoss Shutdown Issues

When enabling the JMXremote features of the JVM on a JBoss 5.1.2 server configuration, the following error is generated when calling the `/bin/shutdown.sh`:

```
javax.management.JMRuntimeException: Failed to load MBeanServerBuilder class org.jboss.system.server.jmx.
MBeanServerBuilderImpl:
java.lang.ClassNotFoundException: org.jboss.system.server.jmx.MBeanServerBuilderImpl
```

Resolution

Sometimes, there is JAVA_OPTS parameter in the environment, and it affects to a java process when the JVM start.
From the reason that EAP uses JAVA_OPTS parameter internally, it should not be in the environment.

To clear the parameter in the environment, executing the following from a console:

1. `export JAVA_OPTS=""`
2. `env|grep JAVA_OPTS`
   JAVA_OPTS= <You will see this message>
3. `shutdown.sh -s jnp://localhost:1099 -u USER -p PASSWORD`

Root Cause

JMX related options should not be set in JAVA_OPTS when calling shutdown.sh. Ensure JAVA_OPTS is only set when starting JBoss EAP, not when running.

Jetty Startup Settings

This page describes how details for instrumenting the Jetty web server with the AppDynamics Java Agent vary depending on the version of Jetty you are using.

Instrument Jetty Version 8.x or 9.x

You can instrument Jetty with the AppDynamics agent either from the server startup command, or by editing the Jetty startup configuration file.

- To add the agent to Jetty at the command line, pass the `javaagent` argument with the fully qualified location of the Java agent JAR file when starting the Jetty server. For example:

  ```
  java -javaagent:/<agent_home>/javaagent.jar -jar start.jar
  ```

- To use the startup configuration file, edit the `start.ini` file in the Jetty base directory by adding these lines:

  ```
  --exec
  -javaagent:/<agent_home>/javaagent.jar
  ```

Be sure to specify the location of the AppDynamics `javaagent.jar` file in the `javaagent` argument as appropriate for your system. Restart the Jetty server after modifying the configuration file to have your changes take effect.

Instrument Jetty Version 6.x or 7.x

For Jetty version 6.x or 7.x, you can add the `javaagent` command line option to your `jetty.sh` file:

1. Open the `jetty.sh` start script file.
2. Add the following `javaagent` argument to the beginning of the script.

   ```
   java -javaagent:/<agent_home>/javaagent.jar
   ```

3. Save the script file.
4. Restart the application server for the changes to take effect.

Custom Exclude Rule for WebAppContext

For the Eclipse version of Jetty, to enable AppDynamics to detect business transactions based on web services, you need to create a Servlet exclude rule for the default Jetty servlet `org.eclipse.jetty.webapp.WebAppContext`.

This ensures that AppDynamics can detect business transactions based on the web services provided by your web applications while ignoring unmapped URL or URLs for the underlying framework.

Mule ESB Startup Settings

To load the Java Agent in Mule ESB, pass the Java Agent JAR location as a JVM argument to Mule.

Mule ESB 3.X or later uses the Tanuki configuration environment. To specify JVM arguments in your Mule ESB environment, you need to configure them as additional parameters to the Tanuki Java Service Wrapper configuration file, `wrapper.conf`, as described below.

Configuring the Tanuki Service Wrapper

1. Open the Java Service Wrapper configuration file:
   `<MULE_HOME>/conf/wrapper.conf`
2. Find the location indicated for Java Additional Parameters:

   ```
   # Java Additional Parameters
   wrapper.java.additional.1=
   ```

3. Add the path to the Java Agent JAR file as a JVM argument using a `wrapper.java.additional.`*n* parameter, as follows.

   ```
   wrapper.java.additional.n="-javaagent:/path_to_appagent/javaagent.jar"
   wrapper.java.additional.n.stripquotes=TRUE
   ```

   Where "n" is the next available integer among the `wrapper.java.additional` parameters already in the `wrapper.conf` file, if any. The numbers serve to identify each Java Additional Parameter in the file. Do not skip numbers when adding the property. Replace `path_to_appagen` to the path to the `javaagent.jar` file in your system. The `stripquotes` parameter is necessary only if there are spaces in the path or filename, but is safe to include if not.

   For example, on a Linux system and with seven Java parameters already in the file, add the following properties:

   ```
   wrapper.java.additional.8="-javaagent:/opt/AppDynamics/Agent/app_agent/javaagent.jar"
   wrapper.java.additional.8.stripquotes=TRUE
   ```

   There may be additional `wrapper.java.additional` properties defined in other Mule files. These are generated by the system. As indicated by comments preceding these properties, you should not make changes directly to the properties. Mule automatically auto-increments the index number for these properties based on the highest integer number used in `wrapper.conf`, so you do not need to modify or otherwise account for the index numbers of auto-generated configuration properties.

Oracle WebLogic Startup Settings

To enable the AppDynamics Java Agent add the `javaagent` command-line option to the WebLogic startup file. If you start and stop clustered WebLogic servers using Node Manager, configure server startup in the WebLogic Server Administration Console.

Instrument Oracle WebLogic for Windows

1. Open the `startWebLogic.cmd` file, located at `<weblogic_version_install_dir>\user_projects\domains\<domain_name>\bin`.
2. Add following `javaagent` argument to the application server start script.

```
set JAVA_OPTIONS=%JAVA_OPTIONS% -javaagent:"<drive>:\<agent_home>\javaagent.jar"
```

   - The `javaagent` argument must reference the full path of the agent installation directory, including the drive letter.
   - The command must precedes the WebLogic start commands, for example:

```
@REM Enable the AppDynamics Java Agent
set JAVA_OPTIONS=%JAVA_OPTIONS% -javaagent:"E:\AppDynamics\AppServerAgent\javaagent.jar"
@REM AppDynamics Java Agent END

@REM START WEBLOGIC
echo starting weblogic with Java version:
%JAVA_HOME%\bin\java %JAVA_VM% -version
```

3. Restart the application server for the changes to take effect.

Instrument Oracle WebLogic for an Application Running as a Windows Service

Some applications have a pre-compiled startup method that installs WebLogic as a Windows service.

To add the agent to the service:

1. Open the script file that starts the application service, such as `install_XXXX_Server_Start_Win_Service.cmd`.
2. Add the `javaagent` command before the line starting with "`set CMDLINE=%JAVA_VM%...`" such as:

```
set CLASSPATH=%MYSERVER_CLASSPATH%;%PRE_CLASSPATH%;%WEBLOGIC_CLASSPATH%;%POST_CLASSPATH%;%
WLP_POST_CLASSPATH%

set JAVA_VM=%JAVA_VM% %JAVA_DEBUG% %JAVA_PROFILE%

set WLS_DISPLAY_MODE=Production

@REM Enable the AppDynamics Java Agent
set JAVA_OPTIONS=%JAVA_OPTIONS% -javaagent:"<drive>:\<agent_home>\javaagent.jar"
@REM AppDynamics Agent END

set CMDLINE=%JAVA_VM% %MEM_ARGS% -classpath %CLASSPATH% %JAVA_OPTIONS% weblogic.Server"
```

3. Open a command prompt and run the following script to remove the existing Windows Service for your application:

```
install_XXXXX_Server_Start_Win_Service.cmd  XXXXX_xxxxx_Production_Server  R
```

4. Install the updated Windows Service for your application:

```
install_XXXXX_Server_Start_Win_Service.cmd  XXXXX_xxxx_Production_Server  I
```

5. From the **WebLogic** web console, stop your application.
6. Start your application (which also starts WebLogic) from the Windows Services application, where the Windows service name = XXXXX_xxxx_Production_Server.
7. Ensure that your application is working properly.

See Creating a Server-Specific Script in the Oracle documentation.

Instrument Oracle WebLogic for Linux

1. Edit `startWebLogic.sh` located at `<weblogic_<version#>_install_dir>/user_projects/domains/<domain_name>/bin/startWebLogic.sh` .
2. Add the following to the beginning of your application server start script:

```
export JAVA_OPTIONS="$JAVA_OPTIONS -javaagent:/agent_home/javaagent.jar"
```

   - The `javaagent` argument must reference the full path of the agent installation directory.
   - The `export` command precedes the WebLogic start commands, for example:

```
#Enable the AppDynamics Java Agent
export JAVA_OPTIONS="$JAVA_OPTIONS -javaagent:/opt/AppDynamics/AppServerAgent/javaagent.jar"

# START WEBLOGIC
echo "starting weblogic with Java version:"
${JAVA_HOME}/bin/java ${JAVA_VM} -version
```

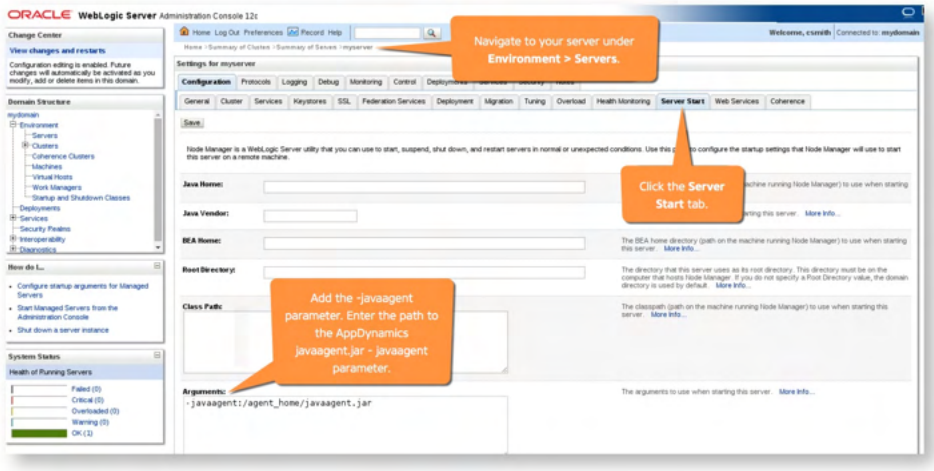
3. Restart the application server for the changes to take effect.

Instrument Clustered WebLogic Servers

For clustered WebLogic servers, you start and stop using Node Manager, and configure server startup in the WebLogic Server Administration Console.

1. Open the **WebLogic Server Administration Console**. See Overview of the Administration Console.
2. Navigate to **Environment > Servers** and click your server in the **Server List**.
3. Click the Server Start tab.
4. Add the `javaagent` argument and set the value to the path to the Java Agent:

```
-javaagent:/<agent_home>/javaagent.jar
```



5. Restart the application server.

Java 2 Security Configuration

If you have enabled Java 2 security on your WebLogic server, you must add this code block to the `weblogic.policy` file:

```
grant codeBase "file:/<agent_home>/-"
{ permission java.security.AllPermission; };
```

OSGi Infrastructure Configuration

The GlassFish application server versions 3.x and later uses OSGi architecture. By default, OSGi containers follow a specific model for bootstrap class delegation. Classes that are not specified in the container's `CLASSPATH` are not delegated to the bootstrap `classloader`; therefore you must configure the OSGi containers for the Java Agentclasses.

See GlassFish Startup Settings and GlassFish OSGi Configuration per Domain.

To ensure that the OSGi container identifies the Java Agent, specify this package prefix:

```
org.osgi.framework.bootdelegation=com.singularity.*
```

This prefix follows the regular boot delegation model so that the Java Agent classes are visible.

If you already have existing boot delegations, add "`com.singularity.*`" to the existing path separated by a comma. For example:

```
org.osgi.framework.bootdelegation=com.sun.btrace., com.singularity.
```

Configure Eclipse Equinox

If running Eclipse Equinox under Glassfish:

1. Open the `config.ini` file located at `<glassfish install directory>/glassfish/osgi/equinox/configuration`.
2. Add this package prefix to the `config.ini` file:

```
org.osgi.framework.bootdelegation=com.singularity.*
```

If running Eclipse Equinox under the WebSphere Application Server Liberty profile:

1. Open the bootstrap.properties file in the JVM directory <*WLP_home*>/usr/servers/<*server_name*> for editing.
2. Add this code line:

```
org.osgi.framework.bootdelegation=com.singularity.*
```

See Getting Started with Equinox.

Configure Apache Sling

1. Open the `sling.properties` file. The location of the `sling.properties` varies depending on the Java platform.
   In the Sun/Oracle implementation, the `sling.properties` file is located at `<java.home>/lib`.
2. Add this package prefix to the `sling.properties` file:

```
org.osgi.framework.bootdelegation=com.singularity.*
```

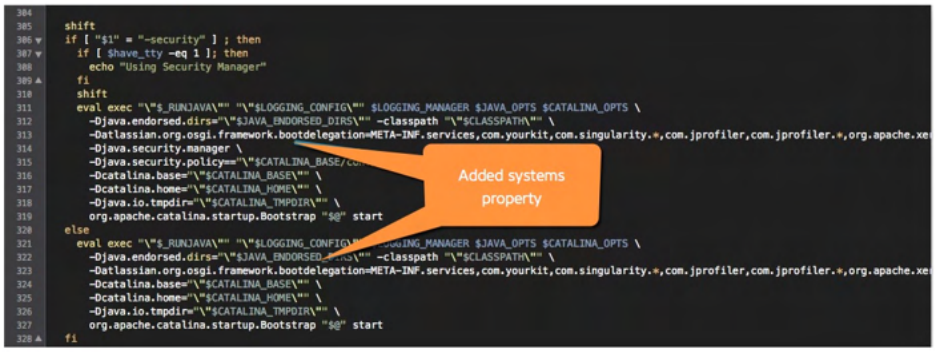
Configure JIRA or Confluence

For JIRA >= 5.1.8, and Confluence >= 5.3:

1. Open the startup script (that is, `catalina.sh`) for editing.
2. Look for the `start` command block (look for "`elif [ "$1" = "start" ] ; then`")
3. Add the following Java system property alongside the existing properties in both `else` blocks:

```
-Datlassian.org.osgi.framework.bootdelegation=META-INF.services,com.yourkit,com.singularity.*,com.
jprofiler,com.jprofiler.*,org.apache.xerces,org.apache.xerces.*,org.apache.xalan,org.apache.xalan.*,sun.
*,com.sun.jndi,com.icl.saxon,com.icl.saxon.*,javax.servlet,javax.servlet.*,com.sun.xml.bind.*\
```

4. Add the property to the run command block as well to instrument the application started in run mode. For example:



5. Before the comment line 'Execute The Requested Command', add the `-javaagent` argument to the file as a new Java option:
6. Restart the application.

Configure Other OSGi-based Containers

For other OSGI-based runtime containers, add this package prefix to the appropriate OSGi configuration.

```
file.org.osgi.framework.bootdelegation=com.singularity.*
```

Resin Startup Settings

The Java Agent bootstraps using the `javaagent` command-line option. Add this option to the `resin.sh` or `resin.bat` file.

Instrument Resin 1.x - 3.x for Windows

1. Open the `resin.bat` file, located at `<resin_home>/bin`.
2. At the beginning of your application server start script, add the `javaagent` argument with the full path (including drive) to the `javaagent.jar` file on your system:

```
exec JAVA_EXE -javaagent:"<drive>:\<agent_home>\javaagent.jar"
```

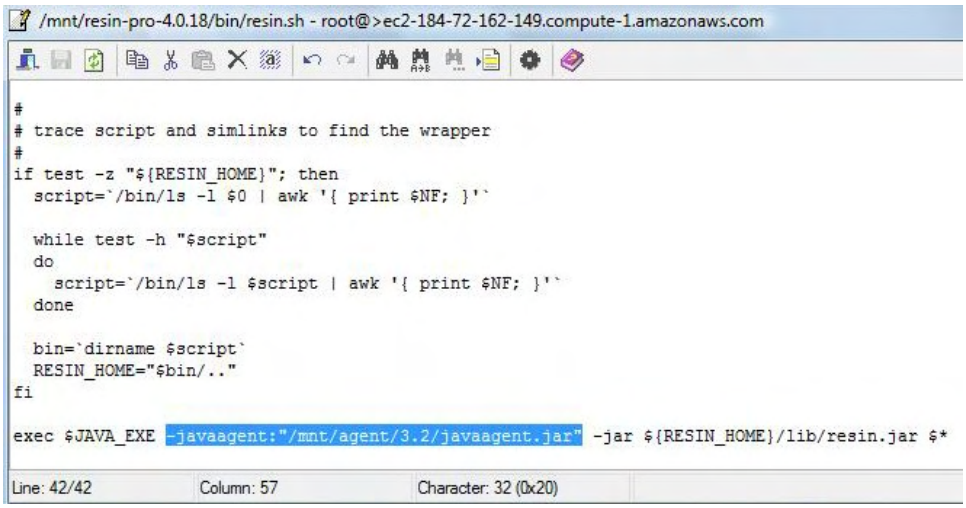
3. Restart the application server to have the changes take effect.

Instrument Resin 1.x - 3.x for Linux

1. Open the `resin.sh` file, located at `<resin_home>/bin`.
2. At the beginning of your application server start script, add the `javaagent` argument with the full path to the `javaagent.jar` file on your system:

```
exec $JAVA_EXE -javaagent:"<agent_home>/javaagent.jar"
```

For example:



3. Restart the application server for the changes to take effect.

Instrument Resin 4.x

1. To install the Java Agent into Resin => 4.X, edit the `./conf/resin.xml` file and add:

```
<jvm-arg>-Xmx512m</jvm-arg>
<jvm-arg>-javaagent:<agent_home>/javaagent.jar</jvm-arg>
```

2. Restart the application server for the changes to take effect.

Apache Solr Startup Settings

This page describes how to instrument Apache Solr with the Java Agent.

Instrument Solr in a Linux Environment

1. Open a terminal.
2. Execute the following commands to add the `javaagent` argument to the Solr server:

   ```
   >cd $Solr_Installation_Directory
   >java -javaagent:"<agent install directory>/javaagent.jar" -jar start.jar
   ```

   For the argument value, reference the full path to the Java Agent installation directory.

Instrument Solr in a Windows Environment

1. Add the Java Agent to the Solr configuration file, `solr.in`.
   You can find the file at:
   `<solr-install-dir>\bin\solr.in`
2. Add the following line just below "`REM set SOLR_OPTS=%SOLR_OPTS%`":

   ```
   set SOLR_OPTS=%SOLR_OPTS% -javaagent:"<java_agent_location>\javaagent.jar"
   ```

3. Restart the Solr service.

Standalone JVM Startup Settings

AppDynamics works with applications running outside of an application server or container.

To install the agent on a standalone JVM, pass the path to the Java Agent when running the Java application in the following form:

```
java -javaagent:"/<agent_home>/javaagent.jar" <classname>
```

Ensure to add the `-javaagent` argument before the `-jar` argument. The `javaagent` argument value must reference the full path to the Java Agent home directory, including the drive on Windows. The classname should be the fully qualified class name with `main` method.

For example, the is an example of the command on a Windows system:

```
java -javaagent:"C:\AppDynamics\agentDir\javaagent.jar"  com.main.HelloWorld
```

On Linux, it would be:

```
java -javaagent:"/mnt/AppDynamics/agentDir/javaagent.jar" com.main.HelloWorld
```

Tanuki Service Wrapper Settings

Related pages:

- Tanuki Service Wrapper Properties
- Example Configuration
- More Help On Tanuki Service Wrapper

The Java Agent bootstraps use the `javaagent` command-line option. Add this option to the Tanuki Service `wrapper.conf` file.

Configure the Tanuki Service Wrapper

1. Open the `wrapper.conf` file to edit.
2. Add a `wrapper.java.additional` property that contains the `javaagent` option with the location of the Java Agent JAR file. For example:

```
wrapper.java.additional.6=-javaagent:C://agent//javaagent.jar
```

Tibco ActiveMatrix BusinessWorks Service Engine Settings

Related pages:

- TIBCO Product Documentation
- TIBCO Community: tibbr

With Tibco ActiveMatrix BusinessWorks Service Engine, each application is started in its own JVM. As a result, for each application that you want to monitor, you must configure the Java Agent in the startup settings for the configuration file of that application.

The file to configure is usually the TRA file named for the application. For example:

`<application_name>.tra`

Instrument the Tibco Application

1. Open the TRA file for the application you want to monitor and add this code to the file:

   ```
   java.extended.properties=-javaagent:/opt/appagent/javaagent.jar
   ```

   AppDynamics recommends that you include all `-D` system property arguments in one `java.extended.properties` line. Otherwise, the `-javaagent` property may be overwritten.
2. Restart the application to have the changes take effect using the command-line tool `bwengine.sh` (or `.exe`) or the Tibco Admin UI.

Since Tibco traffic commonly takes the form of incoming SOAP actions sent through HTTP requests to a single URL (for example, `/BusinessServices/WebGateway`), it's likely that you will need to create a POJO split rule in the AppDynamics configuration to differentiate business transactions for Tibco. For information, see See "Split by POJO Method Call" on Split Servlet Transaction by Payload Examples.

webMethods Startup Settings

You can instrument the webMethods Integration Server or My webMethods Server by adding the Java Agent to the startup script of the server.

The Java agent settings can go in the `runtime.bat/runtime.sh` file, or the `server.bat/server.sh` file under the `bin` subdirectory of the server home.

Instrument webMethods with the Tanuki Wrapper

For webMethods servers that use the Tanuki Java service wrapper for start-up, you need to configure the agent in the `wrapper.conf` file. See Tanuki Service Wrapper Settings.

Note that the order of arguments in the file is important. Specifically, when instrumenting an OSGI-based platform, the agent configuration must precede the `bootclasspath` argument.

For example, the following listing shows the Java options used in a sample `wrapper.conf` file:

```
# Java Additional Parameters
...
wrapper.java.additional.4=-XX:MaxPermSize=256M
wrapper.java.additional.5=-javaagent:/opt/appd/appagent/javaagent.jar
wrapper.java.additional.6=-Xbootclasspath/a:"%OSGI_INSTALL_AREA%/lib/runtime/platform-jaasproxy.jar"
```

Java Security Manager Configuration

Run in a JVM with Security Enabled

To enable a Java Security Manager and instrument the JVM, you need to edit the active security policy file to prevent problems within the interaction between JVM and the Java Agent. The Java Agent requires the change listed in the code block. You can locate the policy file by inspecting the value of the `java.security.policy` system property.

Add this code block to the JVM `server.policy` file:

```
grant codeBase “file:\* AGENT_DEPLOYMENT_DIRECTORY \*/-”
{
   permission java.java security manager.AllPermission;
};
```

WSO2 API Microgateway Startup Settings

You can instrument WSO2 API Microgateway by adding the Java Agent to JVM arguments in the executable file of the gateway. Perform the following steps based on your environment to attach the Java Agent:

Java Agent supports only the HTTP1 protocol for entry and exit.

Automate Java Agent Deployment

Related pages:

- AppDynamics Cookbook on GitHub
- Deploy Multiple Machine Agents From a Common Directory

This page lists sample tools created by AppDynamics users to deploy Java Agents automatically. You can use these samples for ideas on how to automate AppDynamics agent deployment for your own environment.

In the samples, the agents are deployed independently of the application deployment:

- ChefExample1 and ChefExample2 use Opscode Chef recipes and provide examples of automating deployment on Java platforms. See Chef website.
- JavaExample1 uses a script, configuration file, and a package repository.

Click the sample to download:

- ChefExample1.tar
- ChefExample2.tar
- JavaExample1.tar

Install the AppDynamics Site Extension for Java

You can use the Windows Azure Portal to add the AppDynamics Azure Site Extension to your Azure App Service web app. Azure Site Extension is used by Ops teams that may not have access to source files, or would not like to modify or recompile them, yet still want to monitor their Azure projects and solutions.

Prepare to Install

To install the AppDynamics for Windows Azure Site Extension, you need:

- Connection information for your AppDynamics Controller. See Agent and Controller Compatibility.
- A Windows Azure account.
- An Azure web app to monitor.

If you are upgrading from a previous version, see Upgrade the AppDynamics Azure Site Extension.

Add the AppDynamics Azure Site Extension

Add the AppDynamics Azure Site extension as you would any site extension for any Azure web app.

1. Log in to the Windows Azure Portal.
2. Browse to your web app. To configure the Java Agent using environment variables, add the environment variables before you install the AppDynamics Azure Site Extension. See Configure the agent using environment variables.
3. From the **DEVELOPMENT TOOLS** list, click **Extensions**.
4. Click **+Add** to install the version of the **AppDynamics Java Agent** you want to add to your web app.
   After you install the AppDynamics Azure Site Extension, it displays in the installed extensions list.

Configure the Controller Connection

There are two options to configure the Java Agent to connect to the AppDynamics Controller:

- Configure the Controller connection using the Kudu Console.
- Configure the Controller connection settings using environment variables.

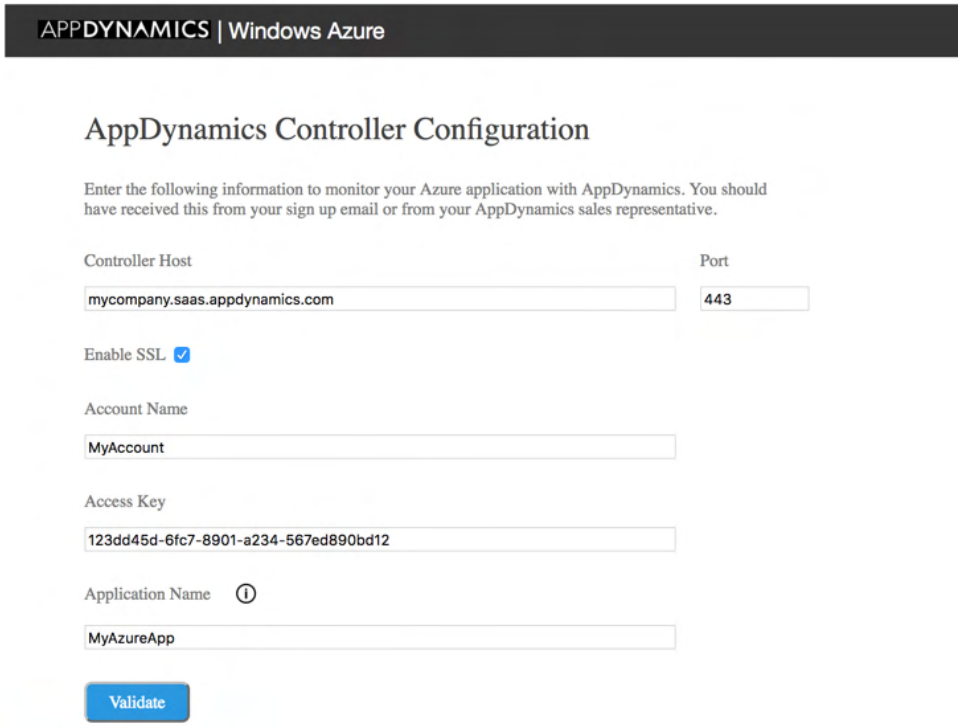
See Agent-to-Controller Connections.

Configure the Agent with the Kudu Console

When you add the AppDynamics Azure Site Extension to your web app, you can interactively configure the Java Agent using the Kudu Console.

1. Navigate to the AppDynamics Controller Configuration page in the Kudu Console:
   `http://{web app}.scm.azurewebsitesJava/appdynamics/`
   For example: `https://myazureexample.scm.azurewebsitesJava/appdynamics/`

2. On the AppDynamics Controller Configuration page, enter your Controller connection information. For example:



3. Click **Validate** to test the connection to the AppDynamics Controller and save your settings.
4. Restart your web app. After you apply some load to your web app, you can view it on flow maps in the AppDynamics Controller UI.

Configure the Agent Using Environment Variables

Configuring the Java Agent using environment variables allows for unattended configuration. To configure agents, add the environment variables before you install the AppDynamics Azure Site Extension:

1. Navigate to **SETTINGS > Application Settings** for your web app.
2. Add the Java Agent environment variables under **App settings:**
    - APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY: A unique key associated with the Controller account.
    - APPDYNAMICS_CONTROLLER_HOST_NAME: The hostname of the AppDynamics Controller.
    - APPDYNAMICS_CONTROLLER_SSL_ENABLED: Set it to `True` to enable SSL connection to the Controller. Otherwise set it to `False`.
    - APPDYNAMICS_AGENT_ACCOUNT_NAME: The account name you use to log on to the Controller.
    - APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY: The account key you use to log on the Controller.
    - APPDYNAMICS_AGENT_APPLICATION_NAME: The business application name in the Controller.
    - APPDYNAMICS_AGENT_TIER_NAME: The tier name in the Controller.
    - APPDYNAMICS_AGENT_NODE_NAME: The node name in the Controller.
3. Restart your web app. After you apply some load to your web app, you can view it on flow maps in the AppDynamics Controller UI.

Upgrade the AppDynamics Azure Site Extension

When you click the **Extensions** tab for your web app, the Microsoft Azure Portal displays the currently installed version of the AppDynamics Azure Site Extension. The **Update Available** column of the installed extensions list indicates if there is a more recent minor release of the Java Agent available. If so, you can click to update the extension from the list.

Upgrade a Major Version of the Java Agent

AppDynamics maintains major release versions of the Java Agent as separate site extensions. Therefore, you must uninstall the installed version of the AppDynamics Azure Site Extension before you upgrade to a new major release:

1. Log in to the Windows Azure Portal.
2. Stop your web app.
3. Click the **AppDynamics Azure Site Extension** from the list of installed extensions and click **Delete** to uninstall it.
4. Install the new version of the AppDynamics Azure Site Extension as normal.

Instrument Java System Classes

The Java Agent excludes system classes like `java.lang.*` from instrumentation by default. To enable instrumentation for a system class, use an agent configuration property.

Instrumented system classes add to the resource overhead introduced by the agent. The amount of overhead depends on the number of calls to the classes. AppDynamics recommends that you instrument a few nodes first and monitor the performance for these nodes before configuring all nodes in your system.

To instrument a Java system class:

1. Open the `<agent_home>/conf/app-agent-config.xml` file for the node where you want to enable instrumentation.
2. Add an `<override-system-exclude>` element with the fully qualified system class name to be instrumented in the `<bci-processing-excludes>` element. The `<override-system-exclude>` element is used to allowlist classes (for example, '`com.sun.jersey`') for instrumentation despite a blocklist (`'com.sun.*'`).

   For example:

   ```
   <bci-processing-excludes>
   <override-system-exclude filter-type="equals" filter-value="com.sun.jersey"/>
   </bci-processing-excludes>
   ```

   You can use these filter types:

   - `equals`
   - `startswith`
   - `contains`

3. Restart the JVM for which you have modified the XML file. Once you restart the JVM, the package will be included in the instrumentation.

Instrument JVMs in a Dynamic Environment

Application environments are increasingly dynamic. For example, in AWS or Rackspace hosted environments, an administrator can online or offline nodes quickly because of workload resulting in an environment where there can be many, short-lived JVMs.

Dynamic Environment Considerations

The AppDynamics app agent operates substantially the same way in a dynamic environment as it does in a traditional data center. However, these considerations apply when deploying agents in dynamic environments:

- Ensure that the agents can reach your Controller. Particularly if your Controller operates on-premises while the agents run on hosted servers, you need to ensure that the agents can access the Controller through the firewalls in your environment.
- Configure custom node expiration handling. By default, the Controller manages the lifecycle for a node based on default timeout settings. You will likely want to reduce the default expiration times for nodes, as described in Historical and Disconnected Nodes. To manage this directly, rather than relying on activity timeouts, a strategy would be to run a script in the JVM before it shuts down that invokes the mark-nodes-historical resource in the AppDynamics REST API to declare itself historical to the Controller.
- Depending on the nature of your environment, it may not make sense to track each JVM instance as a distinct node in AppDynamics, since the set of nodes, in this case, would be boundless. Also, since the JVMs are identical, representing them as different nodes in AppDynamics does not best reflect the logical model of the environment.

Configure the Agent for Dynamic Environments

You can configure AppDynamics to reuse node names for dynamic environments. With node name reuse enabled, after a node is shut down and marked as historical in AppDynamics, the name of the node becomes available for reuse in subsequently registered nodes. You can also specify a node name prefix when reusing node names to indicate the association between all concurrently running JVMs for the dynamic application.

To configure the agent for this environment, use these properties:

- Reuse Node Name Property - To enable re-use of node names.
- Auto Node Name Prefix Property - To set the prefix used for automatically-named nodes.

When enabled, the Controller manages name assignments for the agents. It forms node names by appending a sequentially incremented number as the suffix to the prefix you specify. Be sure to avoid specifying the node name using any other mechanism (such as in `controller-info.xml` file or through a system property).

This listing shows sample settings for the properties as they may appear in a startup script:

```
-Dappdynamics.agent.reuse.nodeName=true
-Dappdynamics.agent.reuse.nodeName.prefix=CloudActivator_
```

When you have enabled reuse node name, the Java Agent logs to standard output after starting up and before registration with the Controller. To troubleshoot agent registration failure, you may want to write the log files to a file instead.

To configure the agent to write logs to a file, edit the `log4j-unknown.xml` at `<agent_home>/<version_number>/conf/logging` and uncomment the `AgentLog Appender`:

```
<!-- To log files to an "unknown" folder before the agent registers
                         with the Controller, uncomment the line below -->
<appender-ref ref="AgentLogAppender"/>
```

This change causes the log files of an agent prior to registration to be written to a directory named "unknown" rather than standard output.

Mark Dynamic Nodes as Historical at Shutdown

As indicated in the Configuring the Agent for Dynamics Environments section, setting the node name reuse flag causes the JVM to report graceful shutdowns, allowing the Controller to mark the corresponding node as historical. If you are not reusing node names, you can use one of these alternative methods to mark a node as historical:

- Set this system property at JVM startup:

  ```
  -Dappdynamics.jvm.shutdown.mark.node.as.historical=true
  ```

- Set this property in `app-agent-config.xml` to `true`:

```
<app-agent-configuration>
   ...
   <configuration-properties>
      ...
      <property name="appdynamics.jvm.shutdown.mark.node.as.historical" value="true"/>
```

Note

If your application environment abortively terminates JVMs rather than allowing them to shut down gracefully, arrange for your orchestration framework to use the controller API (Configuration API > Mark Nodes as Historical) to mark the node as historical at the same time as it terminates the node.

Controller Handling of Historical Nodes

If a node has been out of contact with the Controller for a certain amount of time, the Controller marks the node as a historical node. The Controller suspends certain types of processing activities for the node, such as rule evaluation. See Historical and Disconnected Nodes.

Instrument JVMs in Restricted Environments

Some restricted environments do not allow any changes to the JVM startup script. For these environments, you can use the environmental variable JAVA_TOOLS_OPTIONS in a wrapper script that allows you to specify the initialization of tools, specifically, `-javaagent`.

Create the Wrapper Script

1. Create a wrapper script using this format:
   ```
   {{JAVA_TOOLS_OPTIONS="-javaagent:/opt/appdynamics/appserveragent/javaagent.jar -Dappdynamics.controller.hostName=<controller_host> -Dappdynamics.controller.ssl.enabled=false -Dappdynamics.controller.port=<port> -Dappdynamics.agent.applicationName=<application_name> -Dappdynamics.agent.tierName=<tier_name> -Dappdynamics.agent.nodeName=<node_name>" }}
   export JAVA_TOOLS_OPTIONS
   ```

   > To specify spaces for the application, tier, or node names, use backslashes and quotes around the value, for example: `-Dappdynamics.agent.applicationName=\"$APPLICATION_NAME\"`

2. Add the original startup script to the wrapper script.
3. Make the new script executable, for example, `chmod 750 wrapperscript.sh`.
4. Run your new wrapper startup script.

Full example: `startApplicationWithAppDynamics.sh`

```
#!/bin/bash

JAVA_TOOLS_OPTIONS="-javaagent:/opt/appdynamics/appserveragent/javaagent.jar -Dappdynamics.controller.hostName=<controller_host> -Dappdynamics.controller.ssl.enabled=false -Dappdynamics.controller.port=<port> -Dappdynamics.agent.applicationName=<application_name> -Dappdynamics.agent.tierName=<tier_name> -Dappdynamics.agent.nodeName=<node_name>"

export JAVA_TOOLS_OPTIONS

/path/to/original/script/startup.sh $*
```

Instrument JVMs Started by Batch or Cron Jobs

You can configure the Java Agent for those JVMs that run as cron or batch jobs where the JVM runs only for the duration of the job.

Configure the Java Agent

1. Add the application and tier name to the `controller-info.xml` file.
2. Add the `appdynamics.cron.vm` property to the AppDynamics `javaagent` command in the startup script of your JVM process:

```
-javaagent:<agent_home>/javaagent.jar -Dappdynamics.agent.nodeName=${NODE_NAME} -Dappdynamics.cron.
vm=true
```

The `appdynamics.cron.vm` property creates a delay between the end of the main method and the JVM exit so that the Agent has time to upload metrics to the Controller.

Use the Script Name as the Node Name

You can use the name of the script that executes a cron or batch job in the node name. These commands set the value of variable `NODE_NAME` using the combination of the script and hostname. Add these commands to the startup script of the JVM.

```
# Use the name of the script (no path, no extension) as the name of the node.
NODE_NAME=sample
NODE_NAME="${NODE_NAME%%.}"
echo $NODE_NAME
# Localize the script to the host.
NODE_NAME="$NODE_NAME@$HOSTNAME"
```

Use Environment Variables for Java Agent Settings

As an alternative to using system properties or `controller-info.xml` to configure the agent, you can use environment variables. Environment variables allow agent identity settings to be populated with dynamic environments which may be unique for the JVM.

Environment Variables

To configure the agent with environment variables, set the value of the environment variables. Environment variables exist for most of the agent settings you can configure in the `controller-info.xml` file, such as the application name, node name, and connection settings.

For example, set the node name for the agent using an environment variable using `APPDYNAMICS_AGENT_NODE_NAME`. On Linux, you would enter this command:

```
export APPDYNAMICS_AGENT_NODE_NAME=node23
```

When the agent subsequently starts up, the agent takes node23 as its node name. Typically, the environment variable would be set as part of the start-up routine of the JVM and populated with values determined at startup.

Environment Variables Used by the Agent

The Java Agent takes configuration settings from the following environment variables, when set:

Environment Variable	Equivalent property
`APPDYNAMICS_CONTROLLER_HOST_NAME`	<controller-host>
`APPDYNAMICS_CONTROLLER_PORT`	<controller-port>
`APPDYNAMICS_CONTROLLER_SSL_ENABLED`	<controller-ssl-enabled>
`APPDYNAMICS_AGENT_APPLICATION_NAME`	<application-name>
`APPDYNAMICS_AGENT_TIER_NAME`	<tier-name>
`APPDYNAMICS_AGENT_NODE_NAME`	<node-name>
`APPDYNAMICS_AGENT_ACCOUNT_NAME`	<account-name>
`APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY`	<account-access-key>
`APPDYNAMICS_AGENT_UNIQUE_HOST_ID`	Unique host name
`APPDYNAMICS_AGENT_BASE_DIR`	<agent-runtime-dir>
`APPDYNAMICS_JAVA_AGENT_REUSE_NODE_NAME`	–Dappdynamics.agent.reuse.nodeName
`APPDYNAMICS_JAVA_AGENT_REUSE_NODE_NAME_PREFIX`	–Dappdynamics.agent.reuse.nodeName.prefix

Notice that not all configurable agent settings are configurable through environment variables. For those settings, you need to use system properties or `controller-info.xml`, as described on Java Agent Configuration Properties.

Use System Properties for Java Agent Settings

The Java Agent can accept configuration settings specified as system properties. This is useful when you have more than one JVM running on a machine and need to set node properties dynamically.

The JVMs may be in the same tier, or on different tiers in the AppDynamics model. In either case:

- All common information should be configured using `controller-info.xml.`
- All information unique to a JVM should be configured using the system properties (`-D` option) in the startup script.
- Information in the startup scripts always overrides the information in the `controller-info.xml` file.

Java Agent Configuration Properties describes the system properties available for the Java Agent. See Install the Java Agent.

Reference System Properties

You can reference system properties from the command line, or in the agent configuration file to specify the node name, tier name, or other agent configuration properties dynamically.

For example, consider a Java application started with a script named `startserver.sh`. The script defines a system property as:

```
-Dserver.name=$1
```

The property `server.name` will get the value of the first argument passed when starting the script. For example, the following command to start the JVM would bind the `server.name` property to the value `ecommerce01`:

```
startserver.sh ecommerce01
```

You can use this value in your agent configuration by setting the `-Dappdynamics.agent.nodeName` property at startup:

```
-Dappdynamics.agent.nodeName=$server.name
```

Or, you can use the value in the `controller-info.xml` file, using this format:

```
${system_property_name}
```

For example:

```
<controller-info>
    ...
    <node-name>${server.name}</node-name>
</controller-info>
```

Note that `server.name` is the name of a sample system property used here to identify the node. Use an appropriate system property that is available in your environment for the Java Agent node name.

To have tier names assigned dynamically, supply a dynamic value to the tier name property:

```
java -javaagent:<agent_home>/javaagent.jar -Dappdynamics.agent.tierName=$tierName -Dappdynamics.agent.
nodeName=$nodeName
```

Combine Properties

You can combine multiple system properties to name nodes or tiers. For example, in `controller-info.xml`, combine properties as:

```
${system_property_name_1}${system_property_name_2}
```

You can combine system properties with literals. In the following example "_" and "inventory" are literals.

```
${myhost.name}_${myserver.name}.inventory
```

You can use existing system properties for the Controller host and port settings as well; however, combining properties as shown here is not supported for those settings.

Administer the Java Agent

This page describes how to maintain the AppDynamics Java Agent and how to tune its configuration for your requirements. These include, for example, how to upgrade the agent to a new version and how configure SSL for the agent.

Configuration Options and Precedence

You can control many aspects of the operation of the Java Agent through its configuration settings. The agent provides several approaches for specifying these settings, as listed below.

The order of the list matches the priority of the settings. The agent applies the first non-empty value for a configuration property it encounters, given the following order.

1. Environment variables**.** See Use Environment Variables for Java Agent Settings.
2. System properties passed in the start command for the JVM.
3. Versioned agent properties: `<agent_home>/<version_number>/conf/agent.properties`.
4. Global agent properties: `<agent_home>/conf/agent.properties`.
5. Versioned configuration file: `<agent_home>/<version_number>/conf/controller-info.xml`.
6. Global configuration file: `<agent_home>/conf/controller-info.xml`.

Not all properties are available as environment variables, system properties, or XML elements for the `controller-config.xml`. Property references include only the available configuration methods.

Choose an Effective Configuration Strategy

- Use versioned agent properties and versioned configuration files before using global agent properties and global configuration files. This minimizes any impact of configuration format changes in future agent releases.
- The agent only reads versioned agent properties and versioned configuration files from one versioned directory. During upgrades, manually migrate configurations from directories for previous versions. See Upgrade the Java Agent.
- The Agent Download Wizard automatically configures the agent using a versioned configuration file. The configuration in `controller-info.xml` statically defines the agent identity.
- For dynamic or elastic environments that require flexible agent identity, use an approach that provides for the dynamic node identification. For example, pass node identify or other agent configuration settings dynamically using environment variables or system properties.
- For shared binaries among multiple JVM instances, AppDynamics recommends you use a combination of configuration files and startup properties to configure the app agent. In this case, you configure properties common to all JVMs in the `controller-info.xml` file. Then specify the properties unique to each JVM using environment variables or system properties. See Instrument Multiple JVMs on a Single Machine f or example configurations.
- For some properties, you can use system properties already defined in the startup script as the Java Agent property values. For more information, see Use System Properties for Java Agent Settings.

System Properties Sample Configuration

The following command demonstrates a startup script that includes system properties to start and configure the Java Agent. The application is "ACMEOnline", the tier is "Inventory", and the node is "Inventory1". `SampleApplication` is the application file.

```
java -javaagent:/home/appdynamics/agent/javaagent.jar -Dappdynamics.controller.hostName=mycontroller.example.
com -Dappdynamics.controller.port=8090 -Dappdynamics.agent.applicationName=ACMEOnline -Dappdynamics.agent.
tierName=Inventory -Dappdynamics.agent.nodeName=Inventory1 MyApplication.jar
```

System property values are case-sensitive.

Agent Properties Sample Configuration

The Java Agent does not include an `agent.properties` file by default. The file is a list of key-value pairs of system properties. For example:

```
appdynamics.controller.hostName=mycontroller.example.com
appdynamics.controller.port=8090
```

- The JVM treats properties from the `agent.properties` file the same as system properties. Values in `agent.properties` do not override any properties passed explicitly in the JVM startup script.
- You can also include system properties not specific to AppDynamics in the `agent.properties` file. Values from `agent.properties` do not override values passed in the JVM startup script.
- Do not include `-D` switch for keys in the `agent.properties` file.

controller-info.xml Sample File

The controller-info.xml file resides in `<agent_home>/<version_number>/conf`. Comments in the file from the agent distribution describe the settings. They have been removed in the following example for brevity.

```
<?xml version="1.0" encoding="UTF-8"?>
<controller-info>
        <controller-host>192.168.1.20</controller-host>
        <controller-port>8090</controller-port>
        <controller-ssl-enabled>false</controller-ssl-enabled>
        <application-name>ACMEOnline</application-name>
        <tier-name>InventoryTier</tier-name>
        <node-name>Inventory1</node-name>
        <agent-runtime-dir></agent-runtime-dir>
        <enable-orchestration>false</enable-orchestration>
        <account-name>customer1</account-name>
        <account-access-key>341bf72e-7d7a-1234-b33d-9n712nn574</account-access-key>
        <force-agent-registration>false</force-agent-registration>
</controller-info>
```

For a complete list of configuration settings, see Java Agent Configuration Properties.

Java Agent Directory Structure

Unzipping the Java Agent distribution archive extracts the following files and directories to the current directory.

Java Agent Directory Contents

- `conf`: Common configuration files. The configuration files in this directory are used by an agent if a more specific version of the configuration file does not exist in the version-specific conf directory, `ver<version_number>/conf`.
- `javaagent.jar`: The common JAR file used to bootstrap the Java Agent. To enable the agent, pass the fully qualified location of this file as the `--javaagent` argument value to the JVM at startup.
- `readme.txt`: Instructions and notes for installing the Java Agent.
- `utils`: Binary files for utilities shared across versions of the agent.
- `ver<version_number>`: The `javaagent.jar` file, configuration files, and other resources specific for this version of the Java Agent. This directory is named to reflect the Java Agent version number, such as `ver4.4.0.5`.
  Among other things, the directory contains:
  - `conf`: The configuration files used by this version of the Java agent, including `controller-info.xml`, `app-agent-config.xml`, and more. These settings in these files take precedence over the configuration file in the `<agent_home>/conf` directory.
  - `external-services`: Dynamically loaded modules that extend the agent, such as the Analytics Data Sources dynamic service.
  - `javaagent.jar`: A version-specific JAR file that serves as the Java agent binary. Do not use this Java agent JAR when configuring instrumenting directly. Use `javaagent.jar` in the root directory instead.
  - `lib`: Libraries that support the operation of this version of the Java agent.
  - `logs`: Log files written by the agent.
  - `sdk`: Samples, APIs, and Javadoc for extending the capabilities of the AppDynamics Java Agent.
  - `utils`: Binary files for utilities that are specific to this version of the agent.

You can point the JVM argument at the top-level `javaagent.jar`, as it allows you to upgrade the agent without changing how you deploy your application (unzip the new agent version and place it at the old location on the file system). The top-level jar has a specific version that behaves the same as the `javaagent.jar` under the directory with the same version number in its name.

The version-specific `javaagent.jar` allows you to use different versions of the agent and at the same time re-use the same top-level config files. You can point different applications JVM arg at different versions of the agent.

Contents of the conf Directory

The files located in this directory are commonly used for agent configuration and deployment.

- `transactions.xml`: Configuration settings for business transactions identified by the agent.
- `controller-info.xml`: Configuration settings that identify the node, tier, and application associated with the data that this agent reports to the AppDynamics Controller, along with connection settings for the Controller. For more information, see Java Agent Configuration Properties.
- `app-agent-config.xml`: Local configuration settings for this agent. The settings in this file override any equivalent settings specified globally for the AppDynamics deployment. This file is typically used for short-term property settings or for debugging agent issues.
- `jmx`: Files for configuring the JMX and Websphere PMI metrics.
- `logging/log4j.xml`: Flags to control logging levels for the agent. It is highly recommended not to change the default logging levels.

Java Agent Logging

By default, the AppDynamics Java Agent writes log files to the `<agent_home>/ver<version_number>/logs/<node_name>` directory.

See Agent Log Files for information about how the logs are organized into sets that rollover.

Configure Appender Attributes

Each logger has one or more appenders. Each appender specifies where, and in what format, the data is logged. For each appender, you can configure these attributes:

Attribute	Description	Type	Default Value
`fileName`	The name of the log file.	String	-
`name`	The name of the appender.	String	-
`directory`	The directory that the log file is saved to.	String	""
`immediateFlush`	Whether or not to immediately flush log data. Options: `true`, `false`	String	true
`bufferSize`	The size, in MB, of the event buffer. The buffer temporarily stores messages before they are written to disk.	String	256 KB

Configure Log File Sizes and Rollovers

You may want to limit the length of your agent log files to make troubleshooting easier. To do this, you can specify a maximum size for your log files. When a log reaches that size, it is then rolled over: the log file gets zipped, and a new blank log file is generated to store the next batch of log data. By default, a log file has a maximum size of 20 MB and can be rolled over four times. See Troubleshooting Java Agent Issues.

You can configure log rollover attributes in the `log4j2.xml` file.

1. In `log4j2.xml`, find the `ADRRAFAppender` element that you want to configure rollover attributes for.
2. Modify the `ADRolloverStrategy` element and set values for these attributes:

Attribute	Description	Type
`max`	The maximum number of log files. When the maximum number of files is reached, the oldest log file after the initial log file is deleted.	String
`compressionLevel`	The degree of file compression, with 0 being uncompressed and 9 being the most compressed.	String
`format`	The file format that the log is saved in. Options: `zip`, `gz`	String

   For example:

```
<ADRRAFAppender name="BusinessTransactionsLogger" fileName="BusinessTransactions.log">
 <PatternLayout pattern="[%t] %d{DATE} %5p - %m%n" />
 <SizeBasedTriggeringPolicy size="20 MB" />
 <ADRolloverStrategy max="5", compressionLevel="8", format="zip" />
</ADRRAFAppender>
```

   In the above example, "5" is the total number of log files with backups per appender type per set that we keep. In essence, each agent restart will create {number of appenders} files as a set. Then each file can grow up to "5" files total consisting of the first file and 4 backup log files before it rolls over.
3. Modify the `SizeBasedTriggeringPolicy` element and set values for these attributes:

Attribute	Description	Type
`size`	The maximum log file size, in MB, before a log is rolled over. By default, the size is 20MB before the rollover, because t he logs are compressed.	String

Modify the Log Directory Location

To specify a different log directory, use this system property:

```
-Dappdynamics.agent.logs.dir
```

The default logging directory is `<agent_home>/ver<version_number>/logs/<node_name>`.

Set the Agent Log Level

The default logging level for most log files is `INFO`. Higher logging levels consume more disk space; you can change the logging level to `warn` or `error` to reduce the amount of logging. You can control the logging level for the Java Agent by changing the value of the "level value" parameter in the `log4j2.xml` file in the versioned logging configuration file directory: `<agent_home>/<version_number>/conf/logging`. A restart of the application or agent is not required when changing the agent log level. For example, to set the log level to `DEBUG`:

```
<!-- to control the logging level of the agent log files, use the level attribute below. value="
all|trace|debug|info|warn|error"-->
<AsyncLogger name="com.singularity" level="debug" additivity="false">
    <AppenderRef ref="Default"/>
    <AppenderRef ref="RESTAppender"/>
</AsyncLogger>
```

Direct Logging to Syslog

Instead of having the Java Agent write to the default log directory in the agent home directory, you can configure the agent to direct logging output to `syslog`. The agent supports syslog-based logging through `log4j SyslogAppender`.

Configure the Agent to Send Logs to syslog

1. Open this configuration file in the agent home to edit:
   `<agent_home>/ver<version_number>/conf/logging/log4j2.xml`
2. Add this section to the configuration file:

   ```
   <Syslog name="Syslog" facility="LOCAL1" host="localhost" port="514" protocol="TCP">
       <PatternLayout pattern="[%t] %d{DATE} %5p %c - %m%n"/>
   </Syslog>
   ```

3. Configure the agent to redirect its logs to this appender. Find and replace this section of the file:

   ```
   <!-- to control the logging level of the agent log files, use the level attribute below. value="
   all|trace|debug|info|warn|error"-->
   <AsyncLogger name="com.singularity" level="info" additivity="false">
       <AppenderRef ref="Default"/>
       <AppenderRef ref="RESTAppender"/>
   </AsyncLogger>
   ```

   With the following:

   ```
   <!-- to control the logging level of the agent log files, use the level attribute below. value="
   all|trace|debug|info|warn|error"-->
   <AsyncLogger name="com.singularity" level="info" additivity="false">
       <AppenderRef ref="SyslogAppender"/>
       <AppenderRef ref="RESTAppender"/>
   </AsyncLogger>
   ```

Enable SSL for the Java Agent

Related pages:

- Secure the Platform
- Controller SSL and Certificates
- Encrypt Agent Credentials
- Install the Java Agent

This page describes how to secure communication between the Java Agent and these AppDynamics components using SSL:

- AppDynamics Controller: Before you configure the agent to communicate with the Controller through SSL, you must either use a SaaS Controller or configure the on-premises Controller to use SSL.
- AppDynamics Analytics Agent: Before you configure the agent to communicate with the Analytics Agent through SSL, you must Enable SSL for the Analytics Agent.

The Java Agent supports extending and enforcing the SSL trust chain when in SSL mode.

Gather SSL Configuration Details

In preparation to secure Java Agent communications through SSL, you need information about the SSL configuration of the Controller or the Analytics Agent:

- The SSL port:
  - For SaaS Controllers, the SSL port is 443.
  - For on-premises Controllers, the default SSL port is 8181, but you may configure the Controller to listen for SSL on another port.
  - You configure the port for the Analytics Agent using the `ad.dw.http.port` property. See Enable SSL for the Analytics Agent.
- The signature method for the certificates:
  - A publicly known certificate authority (CA) signed the certificate. This applies for DigiCert, Verisign, Thawte, and other commercial CAs.
  - A CA internal to your organization signed the certificate. Some companies maintain internal certificate authorities to manage trust and encryption within their domain.
  - The Controller or Analytics agent uses a self-signed certificate.

Establish Trust for the Controller SSL Certificate

To establish trust between the Java Agent and the AppDynamics Controller, you must import the root certificate for the authority that signed the Controller's certificate into the agent truststore.

> If you secured your on-premises Controller with a self-signed certificate, see Keystore Certificate Extractor Utility for instructions to create the agent truststore.

1. Obtain the root certificate for the authority that signed the certificate for the Controller.
   a. For SaaS Controller deployments only: You can download the "DigiCert Global Root CA" certificate at the following location: https://www.digicert.com/digicert-root-certificates.htm.
   b. You must import only the Root CA Certificate into the Java SSL Trust Store because Host Certificates have a very short lifespan and change often. Trusting the Root CA Certificate ensures uninterrupted connectivity to the controller when the Host Certificate changes before expiry.
2. Run the Java `keytool` command to import the root certificate. The command creates the keystore in the versioned agent configuration directory if it does not exist:

   ```
   keytool -import -alias rootCA -file <root_certificate_file_name> -keystore <agent_home>/<version_number>
   /conf/cacerts.jks -storepass <truststore_password>
   ```

   For example:

   ```
   keytool -import -alias ControllerRootCA -file /usr/home/appdynamics/DigicertGlobalRootCA.pem -keystore
   /usr/local/appagent/4.3.0.0/conf/cacerts.jks -storepass MySecurePassword
   ```

   > Make note of the truststore password, you need it to configure the Java Agent.

Enable SSL between the Java Agent and the Controller

Configure these system properties in the versioned `controller-info.xml`: `<agent_home>/<version_number>/conf/controller-info.xml`. See SSL Configuration Properties for full details on each property.

- Controller Port: The SSL port for the controller. 443 for AppDynamics SaaS.

  ```
  <controller-port>443</controller-port>
  ```
- Controller SSL Enabled: true.

  ```
  <controller-ssl-enabled>true</controller-ssl-enabled>
  ```
- Controller Keystore Password: The plaintext password for the agent truststore.

  ```
  <controller-keystore-password>MySecurePassword</controller-keystore-password>
  ```

  If you enabled the Secure Credential Store, encrypt the password and enter it here. See Encrypt Agent Credentials.
- Controller Keystore Filename: **P**ath of the agent truststore relative to `<agent home>/<version>/conf`. Required if you use a truststore other than the default `<agent_home>/<version_number>/conf/cacerts.jks`.

  ```
  <controller-keystore-filename>../../conf/cacerts.jks</controller-keystore-filename>
  ```

  > You can specify the Controller port and enable SSL for the Controller in the JVM startup script, but you must specify the truststore password and filename in the versioned `controller-info.xml` file.

Restart the JVM after you complete the configuration changes.

Sample Controller-info.xml with SSL and Secure Credential Store Encryption Enabled

```
<?xml version="1.0" encoding="UTF-8"?>
<controller-info>
    <controller-host>mycompany.saas.appdynamics.com</controller-host>
    <controller-port>443</controller-port>
    <controller-ssl-enabled>true</controller-ssl-enabled>
    <!-- Encrypted Controller keystore / agent trust store password -->
    <controller-keystore-password>Tw49bd0hdCMBoQ5pfMMuYA/cA5B4pouVPkv48ovRm6c=</controller-keystore-password>
    <controller-keystore-filename>../../conf/cacerts.jks</controller-keystore-filename>
    ...
    <!-- Secure Credential Store configuration -->
    <!-- Enable the Secure Credential Store -->
    <use-encrypted-credentials>true</use-encrypted-credentials>
    <!-- Path to they secure credential keystore -->
    <credential-store-filename>/opt/appdynamics/secretKeyStore</credential-store-filename>
    <!-- Obfuscated secure credential keystore password -->
    <credential-store-password>n/8GvAZsKk4gM3Z6g+XQ1w==</credential-store-password>
</controller-info>
```

Establish Trust for the Analytics Agent SSL Certificate

To establish trust between the Java Agent and the Analytics Agent, you must import the root certificate for the authority that signed the Analytics Agent's certificate into the agent truststore.

1. Obtain the root certificate for the authority that signed the certificate for the Analytics Agent.
2. Run the Java `keytool` command to import the root certificate into the JRE truststore.

   ```
   keytool -import -trustcacerts -alias analytics-agent -file <root_certificate_file_name> -keystore
   $JAVA_HOME/jre/lib/security/cacerts
   ```

   For example:

   ```
   keytool -import -trustcacerts -alias analytics-agent -file /usr/home/appdynamics/MyAnalyticsCert.crt -
   keystore $JAVA_HOME/jre/lib/security/cacerts
   ```

Establish Trust for Analytics without an Analytics Agent

You can retrieve Transaction Analytics data through Java Agent >= 4.5.16 without the need for a dedicated Analytics Agent. In this deployment model, the Java Agent communicates with the Events Service directly.

If you use a custom trust store for the Java Agent, you need to establish trust with the Events Service to enable Transaction Analytics without an Analytics Agent. These commands demonstrate how to import the Events Service certificate to the Java Agent trust store:

- Using a signed certificate:

```
keytool -import -trustcacerts -v -alias events_service -file /path/to/CA-cert.txt -keystore cacerts.jks
```

- Using a self-signed certificate:

```
keytool -import -v -alias events_service -file events_service.crt -keystore cacerts.jks
```

If you do not establish trust for the private certificate, you cannot retrieve Transaction Analytics data through your Java Agent. Any applications and business transactions you have configured for Transaction Analytics cannot be enabled if the Java Agent cannot communicate with the Events Service.

You can override this behavior using the `-Dappdynamics.force.default.ssl.certificate.validation` property. See Java Agent Configuration Properties for details.

Enable SSL Between the Java Agent and the Analytics Agent

To enable the Java Agent to access the Analytics Agent over SSL, configure the `appdynamics.analytics.agent.url` system property for the JVM.

- Use the HTTPS protocol.
- Use the Analytics Agent SSL port. The port should be the same value as the Analytics Agent `ad.dw.http.port` property. See Enable SSL for the Analytics Agent.

```
-Dappdynamics.analytics.agent.url=https://<analytics_agent_host>:<analytics_agent_port>/v2/sinks/bt
```

For example:

```
java -javaagent:/home/appdynamics/agent/javaagent.jar -Dappdynamics.controller.hostName=mycontroller.
example.com -Dappdynamics.controller.port=8090 -Dappdynamics.analytics.agent.url=https://my.analytics.
host.example.com:9090/v2/sinks/bt -Dappdynamics.agent.applicationName=ACMEOnline -Dappdynamics.agent.
tierName=Inventory -Dappdynamics.agent.nodeName=Inventory1 ACMEOnline.jar
```

Restart the JVM after you complete the configuration changes.

Secure the Java Agent Truststore

To prevent tampering with the Java Agent truststore, you should:

- Secure the truststore file through filesystem permissions:
  - Make the agent truststore readable by any user.
  - Make the truststore owned by a privileged user.
  - Make the truststore writable only by the specified privileged user.
- Secure the agent configuration files so that they are only readable by the agent runtime user and only writable by a privileged user:
  - Versioned configuration file: `<agent_home>/<version_number>/conf/controller-info.xml`.
  - Global configuration file: `<agent_home>/conf/controller-info.xml`.

Keystore Certificate Extractor Utility

The Keystore Certificate Extractor Utility exports certificates from the Controller's Java keystore and writes them to an agent truststore. It installs to this location:

```
<agent_home>/<version_number>/utils/keystorereader/kr.jar
```

To avoid copying the Controller keystore to an agent machine, you can run this utility from the Controller server. Access the agent distribution on the Controller at this location:

```
<controller_home>/appserver/glassfish/domains/domain1/appagent
```

1. Execute `kr.jar` and pass these parameters:
   - The full path to the Controller's keystore:

```
<controller_home>/appserver/glassfish/domains/domain1/config/keystore.jks
```

- The truststore output file name. By default, the agent looks for a Java truststore file named `cacerts.jks` in the `<agent_home>/<version>/conf` directory in the agent home.
- The password for the Controller's certificate, which defaults to "changeit". If you don't include a password, the extractor applies the password "changeit" to the output truststore.

```
/<full path to application JRE>/bin/java -jar kr.jar <controller_home>/appserver/glassfish/domains
/domain1/config/keystore.jks cacerts.jks <controller_certificate_password>
```

2. Install the agent trust store to the versioned agent configuration directory:

```
<agent_home>/<version_number>/conf/
```

Resolve SSL Issues

If you run into problems with the version of TLS/SSL, see SSL Compatibility between Java Agent and Controller and AppDynamics Agent SSL Protocol.

Upgrade the Java Agent

To upgrade the AppDynamics Java Agent, you copy the existing agent directory to a backup location and replace it with the new agent directory. You then copy configuration file changes made in the old directory to the new agent directory and restart the application server.

This page provides additional background information and step-by-step instructions.

About the Upgrade

Before you begin, review the Release Notes for changes that affect your environment. If you are upgrading both the Controller and agents, first upgrade the Controller and then upgrade the Java Agents.

Also, if upgrading multiple agents in your monitored environment, upgrade the agents for the tiers on which business transactions are originated last. For more information about this requirement, along with Controller and agent compatibility information, see Agent and Controller Compatibility.

Upgrading the agent requires a restart of the application server.

An agent upgrade requires no action on the Controller.

Before starting, download the latest version of the Java Agent for your JVM from the AppDynamics Download Center.

Upgrade the Java Agent

1. Shut down the application server where the Java Agent is installed.
2. Create a backup copy of the current agent installation directory.
3. Extract the Java Agent archive to a new directory
4. Rename the existing agent directory.
5. Rename the new directory to the original name of the old agent directory. The new agent directory and its containing files should have the same directory path as the original one. Using the same directory path avoids the task of manually changing the agent-related configurations in your JVM startup script.
6. Copy `controller-info.xml` and, if necessary, agent.properties from the old agent config directory to the following location in the new directory :

   ```
   <agent_home>/<version>/conf
   ```

   For example:

   ```
   appd_javaagent/ver4.0.0.0/conf
   ```

7. If you previously made changes to the `app-agent-config.xml` or other configuration files, copy those changes to the new file as well.
8. Restart the application server.

Once completed, you can archive the original directory and remove it from the application server home.

Uninstall the Java Agent

Related pages:

- Manage App Agents
- Install the Java Agent
- Java Agent Configuration Properties

If you delete an app agent from the Controller UI, as described in Manage App Agents, but do not shut down the JVM that the Java Agent runs on, the Java Agent will reappear in the UI the next time it connects to the Controller.

To prevent a Java Agent from connecting to the Controller, you need to remove the Java Agent settings from the JVM configuration. This frees the license associated with the agent in the Controller and makes it available for use by another app agent. This page describes how to uninstall a Java Agent from the JVM configuration.

Uninstall the Java Agent

1. Stop the application server on which the Java Agent is configured.
2. Remove the `-javaagent` argument in the startup script of the JVM.
3. Remove any system properties configured for the Java Agent from the startup script of your JVM.
4. Restart the application server.

Tune Java Agent Performance

This page describes how to get the best performance from Java Agents. You can view performance statistics for deployed app agents in the diagnostic data panel.

View Agent Diagnostic Data

The agent diagnostic data shows you information about the performance of App Server agents. You can view the data from the Agents page in the Controller UI.

From the Node Dashboard of the node that contains the agent you want to view, click the Agent Diagnostic Stats subtab. Click **View Diagnostics for all Agents** to view statistics for all agents.

Business Transaction Thresholds

AppDynamics determines whether transactions are slow, very slow, or stalled based on the thresholds for Business Transactions. AppDynamics recommends using standard deviation based dynamic thresholds. See Transaction Thresholds.

Snapshot Collection Thresholds

Snapshot collection thresholds determine when snapshots are collected for a Business Transaction. Too many snapshots can affect performance. Therefore, you should consider snapshot collection thresholds carefully in production or load testing scenarios. See Transaction Thresholds.

Suggested Scheduling Settings

- 10 minutes for deployments with less than 10 BTs
- 20 minutes for deployments from 10 to 50 BTs
- 30 minutes for deployments from 50 to 100 BTs
- 60 minutes for deployments larger than 100 BTs

Suggested Diagnostic Session Settings

- Settings for Slow requests (%value): 20 – 30
- Settings for Error requests (%value): 10 – 20
- Click **Apply to all Business Transactions**.

Tune Call Graph Settings

You can tune call graph settings from the **Call Graph Settings** page, as described in Call Graph Settings.

Suggested SQL Query Time and Parameters

- Increase the **Minimum SQL query time** to 100 ms from the default of 10 ms.
- Enable **Filter SQL Parameter Values**.

Memory Monitoring

Memory monitoring features such as leak detection, object instance tracking, and custom memory should be enabled only for a specific node or nodes when debugging a memory problem. Automatic leak detection is on-demand, and therefore, the leak detection will execute only for the specified duration.

When you observe periods of growth in the heap utilization (%), you should enable on-demand memory leak capture. See Java Memory Leaks.

Agent Heap Storage Monitoring and Shutdown

The Java Agent monitors the heap storage used by the instrumented application and shuts itself down when conditions indicate excessive utilization of Java heap storage by the application. The agent takes this measure not necessarily because it is likely to be the source of the memory pressure—in fact, it can help you diagnose the root cause of the issue—but as a final protective measure to help relieve the pressure on available memory for the application.

Typical reasons for excessive heap utilization include:

- Java heap space is too small compared to your application traffic and footprint
- Java heap memory leak
- A rogue thread is consuming a large amount of memory in a short amount of time

Java Agent monitoring of heap storage works as follows. The agent check heap utilization every 30 seconds. When it detects that heap utilization has exceeded 95%, the Java Agent checks for a garbage collection event. When garbage collection occurs, it checks heap utilization again. If usage still exceeds the threshold, the agent shuts itself down and logs the following agent event:

```
ERROR HeapShortageMonitor - Agent shutdown triggered because max heap usage percentage reached
```

If heap memory utilization exceeds the threshold, but garbage collection never occurs, the agent resumes normal monitoring after 30 seconds. If the heap usage remains above the threshold, the heap shortage monitor resumes checking for garbage collection, as it did previously.

You can use a Java Agent node property to control whether heap storage monitoring is enabled and the action that results from excessive heap utilization. The heap-storage-monitor-enabled controls whether it is enabled (true, by default), and the heap-storage-monitor-shutdown-action property determines the triggered action, between:

- true: If the heap threshold is exceeded, the agent is disabled.
- false: If the threshold is exceeded, a warning-level message is written to the agent log and warning event is sent to the Controller (of type AGENT _EVENT with the text: "AppDynamics agent has detected shortage of JVM Heap Storage").

Troubleshooting Java Agent Issues

This page describes techniques for troubleshooting agent installation and operation. In particular, it describes how to find and interpret information in agent log files.

Resolve Java Agent Startup Issues

The first thing the Java agent does upon application startup is to register with the Controller. Once registered, the agent should appear in the **Settings** > **AppDynamics Agents** list.

If you do not see the agent in the list within a few minutes, check the following:

1. Make sure you have restarted the application server.
2. Verify that the `javaagent` argument has been added to the startup script of your JVM.
3. Verify that you configured the agent-controller communication properties and agent identification properties in the `controller-info.xml` file or as system properties in the startup script of your JVM. See Java Agent Configuration Properties.
4. Check the Agent logs directory located at `<agent_home>/ver<version_number>/logs/<node_name>` for the agent.log file.
5. Verify that the Agent is compatible with the Controller. See Agent and Controller Compatibility, and Troubleshoot Controller Issues.

Locate the Java Agent Log Files

Agent log files are located in the `<agent_home>/ver<version_number>/logs/<node_name>` folder.

The agent.log file is the recommended file to help you with troubleshooting. This log can indicate the following:

- Incomplete information in your Agent configuration
- The Controller port is blocked
- Incorrect file permissions

Error messages related to starting the Java Agent use this format:

```
ERROR com.singularity.JavaAgent - Could Not Start Java Agent
```

Resolve Incomplete Agent Configuration Issues

This table lists typical error messages for incomplete Agent configuration:

Error Message	Solution
Cannot connect to the Agent - ERROR com.singularity.XMLConfigManager - Incomplete Agent Identity data, Invalid Controller Port Value	This indicates that the value for the Controller port in controller-info.xml is missing. Add the Controller port and host value to resolve: • For on-premises Controller installations: 8090 for HTTP and 8181 for HTTPS. • For Controller SaaS service, use the default HTTPS port 443.
Caused by: com.singularity.ee.agent.configuration.a: Could not resolve agent-controller basic configuration	This is usually caused because of incorrect configuration in the Controller-info.xml file. Ensure that the information for agent communication (Controller host and port) and agent identification (application, tier and node names) is correctly configured. Alternatively, you can also use the system properties (-D options) or environment variables to configure these settings.

Unblock the Controller Port

This table lists the typical error message when the Controller port is blocked in your network:

Error Message	Solution

ERROR com.singularity.CONFIG.ConfigurationChannel - Fatal transport error: Connection refused WARN com.singularity.CONFIG.ConfigurationChannel - Could not connect to the controller/invalid response from controller, cannot get initialization information, controller host \x.x.x.x\, port 8090, exception Fatal transport error: Connection refused	Try pinging the Controller from the machine where you have configured the application agent. To check if a port is blocked in the network, use the commands: • `netstat -an` for Windows • `nmap` for Linux. The default ports are: • For on-premises Controller installations: 8090 for HTTP and 8181 for HTTPS. • For Controller SaaS service, use the default HTTPS port 443.

Correct File Permission Issues

This table lists typical error message when the file permissions are not correct:

Error Message	Solution
ERROR com.singularity.JavaAgent - Could Not Start Java Agent com.singularity.ee.agent.appagent.kernel.spi.c: Could not start services"	This is usually caused because of incorrect permissions for log files. To troubleshoot: Confirm whether the user who is running the server has read and write permission on the agent directories. If the user has **chmod a-r** equivalent permission, change the permission to **chmod a+r** "<agent_home>"

Maximum Async Handoff Call Graph Samples Error

This error indicates that the number of handoffs in an asynchronous has exceeded the limit:

```
"WARN AsyncHandOffIdentificationInterceptor - Reached maximum limit 500 of async handoff call graph samples. No more samples will be taken" Error
```

This can result from transactions being misidentified as async transactions. In AppDynamics >= 3.6, all Runnables, Callables, and Threads are instrumented by default except those that are excluded by the agent configuration in `app-agent-config.xml`.

In some environments, this may result in too many classes being instrumented, or cause common classes in a framework that implements the Runnable interface to be mistaken for asynchronous activity when it is not, for example, Groovy applications using Clojure.

To debug, check the call graph for asynchronous activities that are misidentified as asynchronous activities. If found, exclude the packages that are not really asynchronous activities. See Configure the Thread Correlation in Java Agent.

Java Agent Configuration Properties

Related pages:

- Administer the Java Agent
- Use System Properties for Java Agent Settings
- Instrument JVMs in a Dynamic Environment

This page is a reference for the configuration properties for the AppDynamics Java Agent. If you are also installing a Machine Agent on the same machine with the Java Agent, see Machine Agent Installation Scenarios.

Agent-Controller Communication Properties

The following are the Agent-Controller communication properties:

AWS Instance

The `appdynamics-aws-instance-enabled` property helps to enable agent retrieval of AWS instance-id by default during registration.

If Java Agent is running on an AWS instance, then the agent log includes *Agent AWS instance-id:<instance-id-of-host>* else the agent log includes *`Agent AWS instance-id:null`.*

The property is enabled by default and the agent log includes `Agent AWS instance-id retrieval enabled: true`. To disable the property, set `appdynamics.aws.instance.enabled=false`. The agent log includes `Agent AWS instance-id retrieval enabled: false`.

System Property: `-Dappdynamics.aws.instance.enabled`

Environment Variable: APPDYNAMICS_AWS_INSTANCE_ENABLED

Type: Boolean

Default: True

Required: Yes

Controller Host

The hostname or the IP address of the AppDynamics Controller. Example values are `192.168.1.22` or `myhost` or `myhost.example.com`. This is the same host that you use to access the AppDynamics browser-based user interface. For an on-premises Controller, use the value for Application Server Host Name that was configured when the Controller was installed. If you are using the AppDynamics SaaS Controller service, see the Welcome email from AppDynamics.

Element in controller-info.xml: `<controller-host>`

System Property: `-Dappdynamics.controller.hostName`

Environment Variable: APPDYNAMICS_CONTROLLER_HOST_NAME

Type: String

Default: None

Required: Yes, if the Enable Orchestration property is `false`.

If Enable Orchestration is `true`, and if the app agent is deployed in a compute cloud instance created by an AppDynamics workflow, do not set the Controller host unless you want to override the auto-detected value. See Enable Orchestration Property.

Controller Port

The HTTP(S) port of the AppDynamics Controller. This is the port used to access the AppDynamics browser-based user interface.

If the Controller SSL Enabled property is set to `true`, specify the HTTPS port of the Controller; otherwise specify the HTTP port. See Controller SSL Enabled Property.

Element in controller-info.xml: `<controller-port>`

System Property: `-Dappdynamics.controller.port`

Environment Variable: APPDYNAMICS_CONTROLLER_PORT

Type: Positive Integer

Default: For On-premises installations, port 8090 for HTTP and port 8181 for HTTPS are the defaults.
For the SaaS Controller Service, use port 443 for HTTPS connections.

Required: Yes, if the Enable Orchestration property is false.

If Enable Orchestration is true, and if the app agent is deployed in a compute cloud instance created by an AppDynamics workflow, do not set the Controller port unless you want to override the auto-detected value. See Enable Orchestration Property.

SSL Configuration Properties

Controller SSL Enabled

If true, specifies that the agent should use SSL (HTTPS) to connect to the Controller. If SSL Enabled is true, set the Controller Port property to the HTTPS port of the Controller. See Controller Port Property.

Element in controller-info.xml: `<controller-ssl-enabled>`

System Property: `-Dappdynamics.controller.ssl.enabled`

Environment Variable: APPDYNAMICS_CONTROLLER_SSL_ENABLED

Type: Boolean

Default: False

Required: No

Controller Keystore Password

The plain text value of the Controller certificate password. If Use Encrypted Credentials is true, encrypt the password. See Encrypt Agent Credentials.

Element in controller-info.xml: `<controller-keystore-password>`

Type: String

Default: None

Required: No

Controller Keystore Filename

By default, the agent looks for a Java truststore file named `cacerts.jks` in the `<agent_home>/<version>/conf` directory in the agent home. Use this property to enable full validation of Controller SSL certificates with a different Java truststore file. See Enable SSL for the Java Agent.

Element in controller-info.xml: `<controller-keystore-filename>`

Type: String

Default: None

Required: No

Force Default SSL Certificate Validation

Used to override the default behavior for SSL validation. The property can have three states:

- **true**: Forces the agent to perform full validation of the certificate sent by the controller, enabling the agent to enforce the SSL trust chain. Use this setting when a public certificate authority(CA) signs your Controller SSL certificate. See Enable SSL On-Premises with a Trusted CA Signed Certificate.
- **false**: Forces the agent to perform minimal validation of the certificate. This property disables full validation of the Controller's SSL certificate. Use this setting when full validation of a SaaS certificate fails.
- **unspecified**: The validation performed by the agent depends on the context:
    - If the agent is connecting to a SaaS controller, full validation is performed.
    - If the agent is connecting to an on-premises Controller and the `cacerts.jks` file is present, then full validation is performed using the c `acerts.jks` file.
    - If the agent is connecting to an on-premises Controller, and there is no `cacerts.jks` file, then minimal validation is performed

System Property: `-Dappdynamics.force.default.ssl.certificate.validation`

Type: Boolean

Default: None

Required: No

> The force default SSL validation property also applies when connecting the Java Agent to the Events Service for Transaction Analytics. See Enable SSL for the Java Agent.

AppDynamics Agent SSL Protocol

The SSL compatibility table in Agent and Controller Compatibility lists the default security protocol for the different versions of the Java Agent. If the default security protocol for your version of an agent is incompatible with the Controller or it is incompatible with an intervening proxy, pass the `-Dappdynamics.agent.ssl.protocol` system property to configure one of these security protocols:

- SSL
- TLS
- TLSv1.2
- TLSv1.1

System Property: `-Dappdynamics.agent.ssl.protocol`

Type: String

Default: See Agent and Controller Compatibility

Required: No

Configure Allowed TLS/SSL Protocols

Agent communication over TLS/SSL causes the agent to initialize the JVM security subsystem, which sets a permitted list of protocols. By default, AppDynamics excludes SSLv3 and TLSv1protocols due to the known vulnerabilities.

Once the JVM initializes, the list of permitted protocols cannot be changed. If you want to prevent the agent from disabling SSLv3 or TLSv1 protocols, you can configure them to be allowed by naming one (or both, separated by a comma) protocols using this property:

System Property: `-Dappdynamics.agent.tls.allowedAlgorithms`

Type: String

Default: None

Required: No

This example allows both SSLv3 and TSLv1 to be used by the JVM:

```
-Dappdynamics.agent.tls.allowedAlgorithms=SSLv3, TLSv1
```

> This property is applicable from the 4.5.13 version of Java Agent.

Secure Credential Store Properties

Use Encrypted Credentials

Before you enable Use Encrypted Credentials, see Encrypt Agent Credentials for instructions on how to initialize the Secure Credential Store.

Set Use Encrypted Credentials to True to configure the agent to use credentials encrypted with the Secure Credential Store. When you enable Use Encrypted Credentials, you must supply the Credential Store Filename and the obfuscated Credential Store Password.

When Use Encrypted Credentials is true, encrypt the following:

- Account Access Key
- Controller Keystore Password
- Proxy Password

Element in controller-info.xml: `<use-encrypted-credentials>`

Type: Boolean

Default: False

Required: No

Credential Store Filename

The absolute path to the Secure Credential Store keystore. See Encrypt Agent Credentials.

Element in controller-info.xml: `<credential-store-filename>`

Type: String

Default: None

Required: If Use Encrypted Credentials is set to True

Credential Store Password

The obfuscated keystore password for the Secure Credential Store. See Encrypt Agent Credentials.

Element in controller-info.xml: `<credential-store-password>`

Type: String

Default: None

Required: If Use Encrypted Credentials is set to True

Agent Identification Properties

Automatic Naming

If enabled and other agent identification properties are not specified in other settings, the tier and application for the agent are automatically named. The default names are in the format MyApp and MyTier.

Element in controller-info.xml: `<auto-naming>`

Type: Boolean

Default: None

Required: No

Application Name

The name of the logical business application that this JVM node belongs to. Note that this is not the deployment name(ear/war/jar) on the application server.

If a business application of the configured name does not exist, it is created automatically.

Element in controller-info.xml: `<application-name>`

System Property: `-Dappdynamics.agent.applicationName`

Environment Variable: `APPDYNAMICS_AGENT_APPLICATION_NAME`

Type: String

Default: None

Required: Yes

Tier Name

The name of the tier that this JVM node belongs to. Note that this is not the deployment name (ear/war/jar) on the application server.

If the JVM or application server startup script already has a system property that references a tier, such as `-Dserver.tier`, you can use `${server.tier}` as the tier name. See Use System Properties for Java Agent Settings.

The agent registers the named tier with the Controller, if the tier does not already exist, the first time it connects with the Controller. If a tier with the name already exists in the Controller model, the agent is associated with the existing tier.

Element in controller-info.xml: `<tier-name>`

System Property: `-Dappdynamics.agent.tierName`

Environment Variable: `APPDYNAMICS_AGENT_TIER_NAME`

Type: String

Default: None

Required: Yes

Node Name

The name of the node. Where JVMs are dynamically created, use the system property to set the node name.

If your JVM or application server startup script already has a system property that can be used as a node name, such as `-Dserver.name`, you could use `${server.name}` as the node name. You could also use expressions such as `${server.name}_${host.name}.MyNode` to define the node name. See Use System Properties for Java Agent Settings.

In general, the node name must be unique within the business application and physical host. If you want to use the same node name for multiple nodes on the same physical machine, create multiple virtual hosts using the Unique Host ID property. See Unique Host ID.

Element in controller-info.xml: `<node-name>`

System Property: `-Dappdynamics.agent.nodeName`

Environment Variable: `APPDYNAMICS_AGENT_NODE_NAME`

Type: String

Default: None

Required: Yes

Reuse Node Name

Set this property to true to reuse node names in AppDynamics. When you set the property to true, you do not need to supply a node name, but you do need to provide a node name prefix using `-Dappdynamics.agent.reuse.nodeName.prefix` .

> When `ReuseNodeName/prefix` and a node name is used, the `ReuseNodeName` property is given precedence.

This property is useful for monitoring environments where there are many JVMs with short life spans. When true, AppDynamics reuses the node names of historical JVMs for new JVMs. This avoids a proliferation of differently named nodes in AppDynamics over time, particularly when the nodes are essentially identical processes that run over different times. An example of this environment is a z/OS Dynamic Workload Manager based-environment where new JVMs are launched and shut down based on actual workload.

AppDynamics generates a node name with App, Tier and Sequence number. The node names are pooled. For example, the sequence numbers are reused when the nodes are purged (based on the node lifetime).

When the Java Agent starts up, it logs output to the console until it registers with the Controller and the Controller generates the node name. To configure the agent to write logs to a file, edit the log4j-unknown.xml at `<agent_home>/<version_number>/conf/logging`. See Instrument JVMs in a Dynamic Environment.

The Controller reuses node names based on the node retention period property.

System Property: `-Dappdynamics.agent.reuse.nodeName`

Environment Variable: APPDYNAMICS_JAVA_AGENT_REUSE_NODE_NAME (New in 4.5.8)

Type: Boolean

Default: False

Required: No

Example: With the following configuration, the Controller generates a node name with the prefix "reportGen". Node names will have suffixes -1, -2, and so on, depending on the number of nodes are running in parallel. The name of a node that is shut down and qualifies as a historical node may be reused by a new node.

```
-Dappdynamics.agent.reuse.nodeName=true -Dappdynamics.agent.reuse.nodeName.prefix=reportGen
```

Reuse Node Name Prefix

When you configure the agent to reuse node names, use this property to specify the prefix the Controller uses to generate node names dynamically.

System Property: `-Dappdynamics.agent.reuse.nodeName.prefix`

Environment Variable: APPDYNAMICS_JAVA_AGENT_REUSE_NODE_NAME_PREFIX (New in 4.5.8)

Type: String

Default: None

Required: When `-Dappdynamics.agent.reuse.nodeName=true`

Example: Using the following property specifications, the agent directs the Controller to generate a node name with the prefix "reportGen". Node names will have suffixes --1, --2, and so on, depending on how many nodes are running in parallel.

```
-Dappdynamics.agent.reuse.nodeName=true -Dappdynamics.agent.reuse.nodeName.prefix=reportGen
```

Self Service

Configure the Java Agent to automatically name nodes based upon the platform. For automatic node naming to work, you must specify an application name and a tier name.

System Property: `-Dappdynamics.agent.selfService`

Type: String

Values:

tibco: The Java Agent names nodes for the TIBCO process name. See Configure the Java Agent for TIBCO BusinessWorks for more information.

Default: None

Required: No

Account Properties

If the AppDynamics Controller is running in multi-tenant mode or if you are using the AppDynamics SaaS Controller, specify the account name and key for the agent to authenticate with the Controller.

If you are using the AppDynamics SaaS Controller, the account name is provided in the Welcome email sent by AppDynamics. You can also find this information in the `<controller_home>/initial_account_access_info.txt` file.

If the Controller is running in single-tenant mode, you only need to configure the account access key. You can find the unique access key for your Controller instance from the License Management page in the UI.

Account Name

The account name used to authenticate with the Controller.

Element in controller-info.xml: `<account-name>`

System Properties: `-Dappdynamics.agent.accountName`

Environment Variable: `APPDYNAMICS_AGENT_ACCOUNT_NAME`

Type: String

Default: None

Required: Yes for AppDynamics SaaS Controller and other multi-tenant users; no for single-tenant users.

Account Access Key

The account access key used to authenticate with the Controller. If Use Encrypted Credentials is true, encrypt the account access key. See Encrypt Agent Credentials.

Element in controller-info.xml: `<account-access-key>`

System Properties: `-Dappdynamics.agent.accountAccessKey`

Environment Variable: `APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY`

Type: String

Default: None

Required: Yes

If you provide application keys through JVM system properties or environment variables, ensure that you use quotes to wrap any shell special characters that may be contained within application keys to prevent the Shell from interpreting them. See the shell documentation for more detail.

Proxy Properties for the Controller

Use the proxy properties to configure the agent to connect to the Controller through a proxy.

Proxy authentication cannot be used in conjunction with agent SSL. To connect the agent through a proxy via SSL, the proxy must be open (not require the agent to authenticate).

Proxy Host

The proxy host name or IP address.

System Property: `-Dappdynamics.http.proxyHost`

Type: String

Default: None

Required: No

Proxy Port

The proxy HTTP(S) port.

System Property: `-Dappdynamics.http.proxyPort`

Type: Positive Integer

Default: None

Required: No

Proxy User Name

The name of the user that is authenticated by the proxy host.

System Property: `-Dappdynamics.http.proxyUser`

Type: String

Default: None

Required: No

Proxy Password

The absolute path to the file containing the password of the user that is authenticated by the proxy host. The password must be the first line of the file.

If Use Encrypted Credentials is false, enter the password in plain text. If Use Encrypted Credentials is true, encrypt the password. See Encrypt Agent Credentials.

System Property: `-Dappdynamics.http.proxyPasswordFile`

Type: String

Default: None

Required: No

Example: -Dappdynamics.http.proxyPasswordFile=/path/to/file-with-password

Other Properties

Enable Orchestration

When set to true, enables auto-detection of the controller host and port when the app server is a compute cloud instance created by an AppDynamics orchestration workflow. See Controller Host Property and Controller Port Property.

In a cloud compute environment, auto-detection is necessary for the Create Machine tasks in the workflow to run correctly.

If the host machine on which this agent resides is not created through AppDynamics workflow orchestration, this property should be set to false.

Element in controller-info.xml: `<enable-orchestration>`

Type: Boolean

Default: False

Required: No

Agent Runtime Directory

Sets the directory under which all files the agent writes at runtime. If this property is specified, all agent logs are written to <Agent-Runtime-Directory>/logs/node-name and transaction configuration is written to the <Agent-Runtime-Directory>/conf/node-name directory. The log folder location can be overridden with the `appdynamics.agent.logs.dir` property.

Element in controller-info.xml: `<agent-runtime-dir>`

System Property: `-Dappdynamics.agent.runtime.dir`

Environment Variable: `APPDYNAMICS_AGENT_BASE_DIR`

Type: String

Default: <agent_home>/nodes

Required: No

Redirect Logfiles

Sets the destination directory to which the logs will be written to.

System Property: `-Dappdynamics.agent.logs.dir`

Type: String

Default: <agent_home>/logs/<Node_Name>

Required: No

Custom Path for agent conf Directory

Sets a custom path for the agent conf directory. This is where the agent reads its static config files from. If you need to change `custom-activity-correlation.xml` or `app-agent-config.xml` and the agent installation is read-only, this instructs the agent to read the static config files from elsewhere.

System Property: `-Dappdynamics.agent.conf.dir`

Type: String

Default: <agent_home>/ver4.5.x.x.x/conf

Required: No

Force Agent Registration

Set to true only under these conditions:

- The agent has been moved to a new application or tier from the UI, and
- You want to override that move by specifying a new application name or tier name in the agent configuration

Element in controller-info.xml: `<force-agent-registration>`

Type: Boolean

Default: False

Required: No

Auto Node Name Prefix

Set this property if you want the Controller to generate node names automatically using a prefix that you provide.

The Controller generates node names by concatenating the specified prefix with a UUID suffix. For example, if you set the prefix as follows:

```
-Dappdynamics.agent.auto.node.prefix=JoannaAutoNode
```

The generated node name is

```
JoannaAutoNode_d39dbfc1-6f4b-4eb7-a788-c1c0135b6bcb
```

This property provides a similar function to the Reuse Node Name Prefix Property property. However, this property is not meant to be used in combination with reusing node names; use Reuse Node Name Prefix Property for those cases instead.

Element in controller-info.xml: Not applicable

System Property: -Dappdynamics.agent.auto.node.prefix=<your_prefix>

Type: String

Default: Serial number maintained by the Controller appended to the tier name

Required: No

Cron/Batch JVM

Set this property to true if the JVM is a batch/cron process. You can use this property to stall the shutdown to allow the agent to send metrics before shutdown.

Element in controller-info.xml: Not applicable

System Property: -Dappdynamics.cron.vm

Type: Boolean

Default: False

Required: No

Unique Host ID

Logically partitions a single physical host or virtual machine such that it appears to the Controller that the application is running on different machines. Set the value to a string that is unique across the entire managed infrastructure. The string may not contain any spaces. If you have a machine agent associated with the application monitored by the app agent, then this property must be set on the machine agent to the same value. See Machine Agent Installation Scenarios.

System Property: `-Dappdynamics.agent.uniqueHostId`

Environment Variable: APPDYNAMICS_AGENT_UNIQUE_HOST_ID

Type: String

Default: None

Required: No

Agent Meta Info

Allows you to associate arbitrary information with a node, which can then be used as a basis for applying health rules or policies by node. For example, you can exclude a health rule from applying to agents tagged as test agents based on a `meta-info` property. Pass the property in `key;value` format (for example, "`key1;value1;key2;value2`"). Do not use semicolons as value(s) as it is used as a delimiter.

For information on using the properties in health rules or policies (along with built-in `meta-info` properties), see Configure Health Rules or Configure Policies.

System Property: `-Dappdynamics.agent.node.metaInfo`

Type: String

Default: None

Required: No

Low Entropy

Addresses agent startup issues in systems with low to zero entropy available for seeding the PRNG algorithm. This is set to true by default. The NativePRNGNonBlocking algorithm is used through SecureRandom if the system property `appdynamics.low.entropy` is set. Windows is not affected by this change as it does not support NativePRNGNonBlocking, and continues to use the existing Secure Random implementation.

System Property: `appdynamics.low.entropy=true`

Element in controller-info.xml: No

Environment Variable: No

Type: Boolean

Default: True

Required: No

If `appdynamics.low.entropy=true` then the agent takes measures to ensure it does not block when generating random values, even in the absence of entropy.

For example: From Java Agent 20.11, the agent defaults to using NativePRNGNonBlocking as its SecureRandom implementation. This will try to use a non-blocking low entropy algorithm for UUID generation. Note that NativePRNGNonBlocking is not supported on Windows and defaults to the existing implementation of SHA1PRNG.

Analytics Agent

For use with the transaction analytics feature with a remote (or non-default) Analytics agent. See Enable the App Server Agent for a Remote Analytics Agent.

System Property: `-Dappdynamics.analytics.agent.url`

Element in controller-info.xml: No

Environment Variable: No

Type: String

Default: `http://localhost:9090/v2/sinks/bt`

Required: No

Use Simple Hostname

By default (unless overridden with the `uniqueHostId` system property), the agent determines the host name of the OS it is running in by reverse DNS lookup. In some circumstances, this host name may be set as the fully qualified domain name of the host name. If this property is set to true, the agent removes any domain name and uses the simple hostname to identify the host. In cases where the host name is an IP address (which happens if the DNS lookup fails) the full IP address in string form is used. The host name is used in mapping metrics gathered by the machine agent to application nodes (see Unique Host ID Property).

Element in controller-info.xml: `<use-simple-hostname>`

Type: Boolean

Default: False

Required: No

For example: If this property is set to true '`server.mydomain.com`' becomes 'server'.

Filter Sensitive Data

By default, the AppDynamics Java Agent sends transaction data to the Controller that your organization may classify as privileged information. Although such data is useful for diagnosis and troubleshooting, security considerations may require you to filter certain information from view in the Controller. You can use:

- Sensitive data filters to exclude environment variables, system property, and JMX data.
- Sensitive URL filters to exclude sensitive information from a URL in snapshot details.
- Sensitive message filters to exclude sensitive data that the application may place in log messages or exception detail messages.

Default Sensitive Data Filters

When you enable a sensitive data filter, the Controller displays asterisks for the values of matching environment variables or system properties. By default, the Java Agent enables two sensitive data filters in the `app-agent-config.xml`:

- Environment variables or system properties that contain the case insensitive substring "password".
- Environment variables or system properties that contain the case insensitive substring "key".

```
<sensitive-data-filters>
        <sensitive-data-filter applies-to="environment-variables,system-properties"
                               match-type="CONTAINS"
                               match-pattern="password"/>

        <sensitive-data-filter applies-to="environment-variables,system-properties"
                               match-type="CONTAINS"
                               match-pattern="key"/>
</sensitive-data-filters>
```

Add a Sensitive Data Filter

1. Edit a versioned app-agent-config.xml file: `<agent_home>/<version_number>/conf/app-agent-config.xml`.
2. Add a sensitive data filter element as a child of the Sensitive Data Filters element using one of these attributes.
    - Specify a comma-separated list in the `applies-to` attribute to filter:

      `environment-variables`

      `system-properties`

      `jmx-mbeans`
    - Set the `match-type` attribute as:

      `EQUALS`

      `CONTAINS`

      `STARTSWITH`

      `ENDSWITH`
    - Specify a string to match for the `match-pattern` attribute. String matches are case insensitive. The pattern matches against the environment variable and system property names, not values.
3. Restart the JVM.

In this example, the Java Agent checks for system properties and environment variables beginning with the string "DB_". The Controller displays the values of matching environment variables and system properties as asterisks. For instance, an environment variable "DB_USER" is replaced with an asterisk.

```
<sensitive-data-filter applies-to="environment-variables,system-properties"
                                    match-type="STARTSWITH"
                                    match-pattern="DB_"/>
```

Add a Sensitive URL Filter

You can use sensitive URL filters to configure the agent to obfuscate sensitive information from the URLs in transaction snapshot details.

1. Edit a versioned app-agent-config.xml file: `<agent_home>/<version_number>/conf/app-agent-config.xml`.
2. Add a sensitive URL filter element as a child of the sensitive URL filters element:

```
<sensitive-url-filter delimiter=""
   segment="" match-filter="EQUALS|INLIST|STARTSWITH|ENDSWITH|CONTAINS|REGEX|NOT_EMPTY"
   match-pattern="pattern"
   param-pattern=""/>
```

- `delimiter`: Specify the character that you want to use as URL segment endpoints. The agent splices the URL at each delimiter instance to create the segments. For HTTP, use the forward slash character "/". In the case of a forward slash, the agent does not split on the slashes immediately following the protocol. For example, "https://myapp.example.com/" constitutes a single segment. By default, the delimiter is "/".
- `segment`: Specify a comma-separated list to indicate the segments that you want the agent to filter. Segment numbering starts from 1.
- `match-pattern`: Specify the string that you want to be filtered by the `match-filter`.
- `param-pattern`: Specify the regular expression matching the query parameters to filter.

For example, the following configuration splits the URL on the "/" character and masks the second segment and the param-pattern in the third segment of the URL. In this case, the segmentation and obfuscation apply only to URLs containing "myapp".

```
<sensitive-url-filters>
<sensitive-url-filter delimiter="/"
 segment="2"
 match-filter="CONTAINS"
 match-pattern="myapp"
 param-pattern="[a-z]+_name"/>
</sensitive-url-filters>
```

The exit call to `https://myapp.example.com/sensitive/data?first_name=abc&last_name=xyz` breaks down to three segments: "`https://myapp.example.com/`", "sensitive", and "data?first_name=abc&last_name=xyz". The Controller shows the masked values of the URL and the param-pattern display `https://myapp.example.com/*****/data?first_name=***&last_name=***` in the snapshot details.

> In case you do not use any values for the query parameters, the Controller does not mask any query parameters in the URL.

Add a Sensitive Message Filter

You can use sensitive message filters to configure the agent to obfuscate sensitive information contained within text messages collected by the agent from log messages, or detail messages from exceptions.

1. Edit a versioned app-agent-config.xml file: `<agent_home>/<version_number>/conf/app-agent-config.xml`.
2. Add a sensitive message filter element as a child of the sensitive message filters element:

```
<sensitive-message-filter message-type="throwable,logger-message,all"
                                match-type="EQUALS|CONTAINS|STARTSWITH|ENDSWITH|REGEX"
                                match-pattern="CASESENSITIVE_PATTERN"
                                redaction-regex="SENSITIVE_INFO_REGEX_GROUP"/>
```

- `message-type` specify throwable, `logger-message` or all
- `match-type` specify the type of match that should be used to `opt-in` messages for redaction
- `match-pattern` specify the pattern that, when matched, opts the message in for redaction
- `redaction-regex` specify a regular expression identifying data that should be redacted from the `opted-in` messages

For example, if an application logs SQL queries including secret numeric values, the following configuration would remove all the numeric data from the logged messages:

```
<sensitive-message-filter message-type="logger-message"
                                match-type="CONTAINS"
                                match-pattern="SELECT"
                                redaction-regex="[0-9]+"/>
```

Resulting in –

```
"SELECT name from customer WHERE customer.id = 773"
```

Being collected as –

```
"SELECT name from customer WHERE customer.id = *****"
```

Optimize Heap Mode

The `enable-optimised-heap-mode` node property helps to fine-tune the agent heap consumption to deliver optimized performance. Additionally, exceptions are handled on the asynchronous entry maps to prevent stalled Business Transactions.

Element in controller-info.xml: `<enable-optimised-heap-mode>`

System Property: `-Dappdynamics.enable.optimised.heap-mode {{}}`

Type: `Boolean`

Default: `Disabled`

Required: `Yes`

IBM Java Agent

Under most circumstances, the IBM Java Agent works the same as the Java Agent. This page describes information specific to the IBM Java Agent.

Supported JVMs

IBM JVM 1.5.x, 1.6.x, 1.7.x, and 1.8.x

Instrument the IBM Java Agent

To change instrumentation for the IBM Java Agent, you must restart the IBM JVM. By default, the IBM Java Agent does not apply BCI changes without restarting the JVM. This is because in the IBM VM (J9 1.6.0) the implementation of re-transformation affects performance (changes the JIT behavior such that less optimization occurs).

These changes require that you restart the IBM JVM:

- Automatic leak detection
- Custom memory structures
- Information points
- Custom POJO rules for transaction detection
- Custom exit point rules
- End User Monitoring (EUM), when you enable it and/or disable it after first enabling it

Instrument WebSphere/InfoSphere with WebSphere Security Enabled

If your WebSphere/InfoSphere environment includes a security-enabled WebSphere Application Server, several InfoSphere Master Data Management (MDM) Server clients require security configuration.

See IBM WebSphere and InfoSphere Startup Settings.

Threading and the Java Agent

A pivotal part of the functionality of the Java Agent is tracing activity within the JVM such that all the activity (both internal and external to the JVM) that results from servicing any given inbound request can be associated with the request itself. In simple cases, each request is processed by one Java thread, making it only necessary to track the processing of that single thread, in a pattern widely known as 'thread per request'.

This was the case for JEE applications prior to the introduction of version 3.0 of the servlet specification (introduced in Java EE 6). In systems where a significant amount of time is spent waiting for external resources, the thread per request model is inefficient because many application threads spend most of their lifetime blocked awaiting external responses.

This observation motivated the asynchronous servlet capabilities provided as part of servlet 3 and also motivates many of the increasingly prevalent reactive frameworks that are in common usage (for example, Play, Akka, Vert.x and Spring Reactor). To correctly correlate transactions end to end in applications using this architectural approach, the agent must follow transactions from thread to thread to enable it to measure and link all the elements of work performed on behalf of each request, irrespective of which thread that work is executed on.

The agent provides two mechanisms for thread tracking, known as 'Constructor mode' and 'Executor mode' (for Java Agent >= 20.11.0).

These are two alternative approaches to the correlation of cross-thread transaction activity:

- Constructor Mode Thread Tracing - This is the mode that instruments *Threads*, *Runnables,* and *Callables* based on their creation. The agent used this mode formerly as the default mode.
- Executor Mode Thread Tracing - This tracking mode is the default mode and is based on instrumenting executor frameworks that are commonly used by the majority of applications and frameworks. This mode makes agent operation more efficient, especially in cases where large use of threading is in place, for example, in the presence of the Play, Akka, Vert.x, and Spring Reactor reactive frameworks.

Select the Thread Correlation Mode

You can select the thread correlation mode used by the agent using the `async-instrumentation-strategy` property within the `app-agent-config.xml` configuration file.

To run the applications in the Constructor mode, modify the value of `async-instrumentation-strategy` property, or use the following java command-line option:

```
-Dappdynamics.async.instrumentation.strategy=constructor
```

The system property takes precedence over the configuration file. You cannot change the configured thread instrumentation mode during runtime.

Constructor Mode Thread Tracing

When the agent is in Constructor Mode, transaction activity is tracked from thread to thread by instrumenting the constructors of Runnable objects, and their run methods, such that the agent can link the creation point of the object with their actual execution. Since in many cases the work executed by Runnable objects does not relate to the Business Transaction context within which they were created, the agent provides for exclude rules to limit which threads are tracked. The agent configuration maintains a default set which you can fine-tune when necessary.

Constructor Mode Configuration

You configure classes for multithreaded correlation in the `<excludes>` child elements of the `<fork-config>` element in the `<agent_home>/conf/app-agent-config.xml` file.

The default configuration excludes the Java, Org, WebLogic and WebSphere classes:

```
<fork-config>
    <!-- exclude java and org -->
    <excludes filter-type="STARTSWITH" filter-value="com.singularity/"/>
    <excludes filter-type="STARTSWITH" filter-value="java/,javax/,com.sun/,sun/,org/"/>
    <!-- exclude weblogic and websphere -->
    <excludes filter-type="STARTSWITH" filter-value="com.bea/,com.weblogic/,weblogic/,com.ibm/,net/sf/,com/mchange/""/>
    . . .
```

The agent supports package names where the levels in the hierarchy are either separated by dots (.) or slashes (/). The agent converts the dots to slashes internally.

Custom Configuration

You can edit the `app-agent-config.xml` file to exclude additional classes from thread correlation. All classes not excluded are by default included.

Use the `<excludes>` element to specify a comma-separated list of classes or packages to be excluded. Use the singular form, `<excludes>` , to specify a single class or a package.

For includes, you must specify each package or class individually using the `<include>` element. You can include a sub-package or subclass of an excluded element by specifying the sub-package or subclass with an `<include>` element. If a classname is matched by both an exclude and include pattern, the more specific pattern matched prevails. That is, an include match with `org.example.myclass` prevails over an exclude match for `org.example`.

Configuration Using Node Properties

You can also configure which classes or packages to include or exclude using a node property. See thread-correlation-classes and thread-correlation-classes-exclude.

Disable Thread Correlation at the Agent

To disable thread correlation (which disables end-to-end asynchronous transaction monitoring as well) at the agent, use one of these:

- Set the Java agent node property thread-correlation-classes-exclude to disable asynchronous monitoring for all the relevant classes.
  `thread-correlation-classes-exclude=a,b,c,d,e,f,...z`
- Add the following line under the fork-config section of the `app-agent-config.xml` file.

  `<exclude filter-type="REGEX" filter-value=".*"/>`

Executor Mode Thread Tracing

Executor mode is the default mode for instrumenting the asynchronous tasks beginning with the Java Agent 20.11 release.

When the agent is in Executor mode, transaction activity is tracked from thread to thread by instrumenting `Executor.exeute()`and similar method(s) such that the agent can identify application work to track as that work is being scheduled.

Benefits of Executor Mode

- Reduces the agent's resource consumption for most use cases
- Enables tracking thread hand-offs using the new agent API
- Improves reliability of last thread on tier asynchronous transaction demarcation
- Allows Service Endpoints to measure the response time of services implemented using asynchronous frameworks

Behavior of Service Endpoints in Executor Mode

The behavior of Service Endpoints differs between the Constructor and Executor modes. In Constructor mode, each asynchronous transaction segment is represented by its own Service Endpoint, and the Service Endpoint corresponding to the transaction entry point shows the execution time of the initiating thread as shown:

Name	Response Time (ms)	Calls	Calls / min	Errors	Errors / min	Type	Tier
/AsyncServletOne/one	112	15	3	0	0	Servlet	WebServer
RequestProcessor	1,126	15	3	0	0	Async	WebServer
AsyncServlet$RequestProcessor$$Lambda$7	119	15	3	0	0	Async	WebServer
AsyncServlet$RequestProcessor$$Lambda$8	95	15	3	0	0	Async	WebServer

With the Executor instrumentation strategy, metrics for a single Service Endpoint are reported, with a response time corresponding to the execution time of the entire asynchronous transaction as shown below.

Name	Response Time (ms)	Calls	Calls / min	Errors	Errors / min	Type	Tier
/AsyncServletOne/one	684	15	3	0	0	Servlet	WebServer

Currently**,** each service entry point must match a transaction entry point. Configuration of Service Endpoints at other points within the execution of asynchronous transactions is *not* supported in the Executor mode. Additionally, Service Endpoints only report if placed within the context of executing a Business Transaction when the agent is in Executor mode.

Raw Threads Support

Support for raw threads differs between the Executor and Constructor modes. While Constructor mode supports tracking of all threads created in the context of a transaction, the Executor mode supports only non-daemon threads that are started in the context of a transaction. The daemon status is inherited from the parent thread which can cause unexpected results. For example, the worker threads in an application server are often daemon threads, so any thread started directly within servlet code is not tracked in the Executor mode unless the application explicitly unsets its daemon status.

Node Properties

This table presents the node properties which are specific for the executor-based instrumentation strategy, or which work differently than for the constructor-based instrumentation strategy (default).

Property	Default value	Description
Property	**Default value**	**Description**
thread-correlation-classes	none	These properties have the same effect in the Executor mode as in Constructor mode, however, their use is discouraged in Executor mode, especially when used to limit the costs of thread tracking. Due to the differences in implementation of the async hand-offs, any performance gains achieved using these settings in constructor mode are likely to be available using the out of the box settings in Executor mode.

thread-correlation-classes-exclude		
min-transaction-stall-threshold-in-seconds	60	For Executor mode only, the transactions will not get checked for stall unless they run at least the specified number of seconds

Exclude Selected Activities from Transactions

Asynchronous frameworks use lower-level asynchronous mechanisms that are not related to the processing of user transactions (for example, a thread pool might grow and shrink automatically, depending on the load or a framework might initialize its thread pool upon the first use). You must ignore threads used in these contexts since their lifecycle is not bound to any individual application transaction.

To facilitate the exclusion of such asynchronous components (usually Threads and other Runnables), the Executor mode offers a mechanism called *capture suppression*. The suppression is tied to a specific method by a suppression rule in the async-config section of `app-agent-config.xml`. Consider the following example of such a rule provided by default:

```
<job>
  <match-class type="matches-class"><name filter-type="EQUALS" filter-value="java.util.concurrent.
ThreadPoolExecutor"/></match-class>
  <match-method><name filter-type="EQUALS" filter-value="addWorker"/></match-method>
  <action type="suppression"/>
</job>
```

The `addWorker` method on the `java.util.concurrent.ThreadPoolExecutor` class is a private method invoked by `ThreadPoolExecutor` whenever a new worker thread is to be created, started and added to the pool. Since such threads are not directly associated with any individual transaction at the point of creation, a suppression rule is used in order that any asynchronous task hands offs occurring within this method will not get associated with the transaction. Any asynchronous tasks subsequently executed by these threads will be associated with the transaction that created the task.

Trace Multithreaded Transactions for Java

Related pages:

- End-to-End Latency Performance
- Configure the Thread Correlation in Java Agent

The Java Agent can automatically track multithreaded applications. It correlates threads from classes that implement `Runnable` or `Callable` interfaces or extend the `Thread` class. Processes with such threads are identified in AppDynamics as asynchronous.

Monitor Asynchronous and Thread Processing Activity

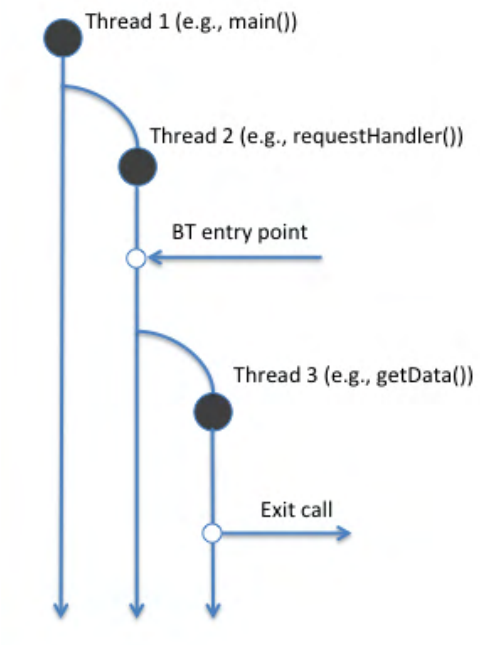
You can monitor performance around asynchronous calls and multithreaded activity in a variety of ways. It's helpful to understand what AppDynamics considers to be an asynchronous activity, what information it provides on that activity, and how that information relates to thread execution at the code level.

These sections describe the type of information available given various thread and exit call execution scenarios.

Asynchronous Exit Call

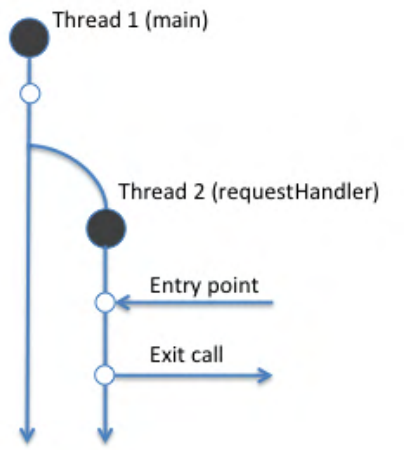
In the most common case, an asynchronous exit call is an exit call made by a thread that is spawned by a thread other than the primary business transaction thread.

The primary thread, in this case, is the one on which an entry point call is received. A common pattern for this case is when the main() function in the application spawns multiple threads as request handlers to receive and process requests. For example:



Flow maps in the Controller UI show this connection as a dotted line (asynchronous connection).

AppDynamics considers the call to be an asynchronous call only due to the context in which it was invoked. If the exit call is made from thread 2, as shown in the following illustration, the call may be considered asynchronous from the perspective of the overall design of the application, but is not considered asynchronous in the context of the business transaction:



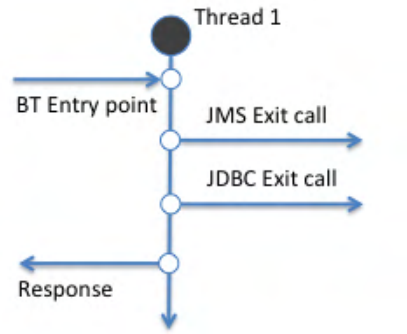
Without a business transaction, AppDynamics does not track asynchronous activity.

AppDynamics does not differentiate between the two common patterns in asynchronous processing, Wait-for-Completion (in which the threads report back into the primary thread) and Fire-and-Forget (in which the primary thread may finish processing before spawned threads do). Both are represented the same way in the Controller UI, although they would usually be distinguished by their execution time, in which the primary thread for a Wait-for-Completion transaction would exceed any of the child processes, whereas Fire-and-Forget the primary thread may be shorter than child processes.

Logical Asynchronous Exit Calls

An exception to the scenario described in the previous section exists for exit calls to backend systems which—from a conceptual point of view—are considered asynchronous in the AppDynamics model.

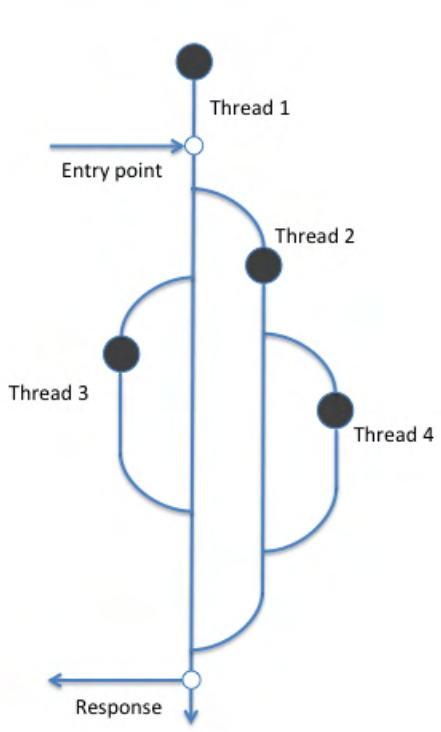
Message queue tiers such as JMS are considered logically asynchronous, so flow maps show connections to message queues as dotted lines (asynchronous connections) as well. Therefore, the following JMS exit call would be shown as an asynchronous call in a flow map, while the JDBC call would not be.



Time on Tier

Exit calls are one consideration in understanding multithreaded application performance; another consideration is the actual time spent processing a request.

For example, consider the following processing scenario. This occurs on a single tier. In fulfilling a request, the entry point thread spawns multiple child threads that report back to the primary thread to assemble the response.



For the transaction, the response time metric represents the time from when the request arrived at the entry point to the point the response was sent. However, while this may accurately represent the user experience of the transaction, it does not reflect the processing burden on the system, since many threads participate in the transaction. A slow performing thread may or may not be evident based on the user experience of the transaction.

To better understand the complete cost of the transaction in terms of time spent processing, you can use the time on tier metric. This metric shows the aggregate processing time for each of the threads involved in processing a request. In other words, the total execution time of threads 2, 3 and 4 from the figure, along with the processing time between the entry point and response for thread 1.

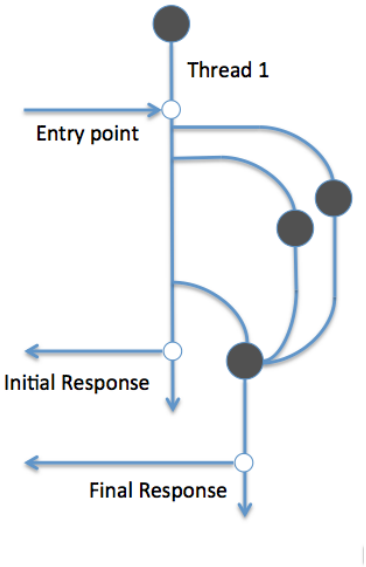
The value of the time on tier metric is indicated by the async tag in business transaction dashboards.

End-to-End Processing

A response from an entry point thread may not best represent the logical end of a business transaction.

For example, consider an application in which an entry point method in a request handler spawns multiple threads, including one to serve as the final response handler. The request handler then returns a preliminary response to the client. By default, this stops the clock for purposes of measuring the response time for that business transaction. Meanwhile, the spawned threads continue to execute until completion, at which point the response handler generates the final response to the client.

In this case, the initial response time is much shorter than the full logical transaction execution time.



End-to-end metrics let you monitor logical transactions that use this programming pattern. End-to-end metrics include the end-to-end transaction time, the number of end-to-end transactions per minute, and the number of slow end-to-end message processing events.

To enable end-to-end metrics, you configure an Asynchronous Transaction as described in Asynchronous Transaction Demarcators.

View Threads and Thread Tasks in the Metric Browser

In a multithreaded transaction, AppDynamics reports key business transaction performance metrics for individual threads in a Thread Tasks branch of the tier in the Metric Browser. The Thread Tasks branch is present in the Metric Browser only for multithreaded transactions. The Metric Browser path is **Business Transaction Performance > Business Transactions > *tier-name* > *business-transaction-name* > Thread Tasks**. Thread Tasks also appear for tiers under Overall Application Performance, where you can see metrics on specific calls made by each thread in a node or in a tier.

Thread Metrics in Health Rules

For each asynchronous thread spawned in the course of executing a business transaction, AppDynamics collects and reports these metrics:

- Average response time
- Calls per minute
- Errors per minute

You can create custom health rules based on the performance metrics for a thread task. When you click the metric icon in the Health Rule Wizard, the embedded metric browser includes the Thread Tasks if the entity for which you are configuring the health rule spawns multiple threads. See Configure Health Rules.

Use the Java Agent API and Instrumentation SDK

Related Pages

- AppDynamics APIs
- Data Collectors
- Add Custom Fields to Business Transactions
  Using the Java SDK

This page describes the Java Agent API, instrumentation SDK (iSDK), and the common use cases for the API.

Java Agent API and iSDK

AppDynamics automatically detects an application activity out-of-the-box. For supported application frameworks, the agent ships with an out-of-the-box configuration that applies instrumentation to the application that identifies the following:

- Where the Business Transaction activity starts
- Where thread handoffs occur
- Where calls to downstream systems are made, and how to inject correlation headers into outbound messages and retrieve them from inbound messages. This ensures that AppDynamics can establish the end-to-end transaction flow through the application architecture

For frameworks that do not benefit from out-of-the-box support, the agent provides a set of Agent APIs that allow the application developer to make code changes to add calls to the agent. This enables the agent to identify and follow Business Transactions as they execute, providing end-to-end visibility for any application.

For scenarios where it is not possible to change the application code, for example, to add instrumentation to a third-party product or framework, an 'instrumentation SDK' is provided that supports the use of these APIs. The iSDK enables developers to build 'interceptors', which the agent injects into application bytecode at specified points on startup. These interceptors can contain any Java code (including calls to the Agent API), which is executed when the code provided in the interceptor is run before, or after the intercepted method itself, executes.

You can seamlessly combine any Agent APIs with the out-of-the-box instrumentation to support situations where a mixture of supported and unsupported frameworks are in use (for example, a standard servlet making an external call using a proprietary API).

The Java Agent API enables you to programmatically:

- Define Business Transactions
- Define Exit Calls
- Inform the agent when the application hands-off transaction processing between threads
- Add application data to snapshots or transaction analytics
- Report custom metrics

When instrumenting any application, the Agent API design prioritizes the success of the application transactions over instrumentation. Therefore, if any of the Agent API calls fail, they do not throw exceptions disrupting the transaction flow, but log messages to aid diagnosis. This indicates that there is no hard dependency between the application and the presence of the AppDynamics Java Agent within the JVM.

See https://sdkdocs.appdynamics.com/javadocs/java-agent-api/v4.5/ for the Javadoc reference for the agent API. User guides for the Agent API and the instrumentation SDK are also provided to illustrate the usage and common use cases.

Install the Dependency

You can access the Agent API jar directly, or download it from Maven Central or from the AppDynamics portal. The library version changes with each new API release, and is not tightly coupled to the version of the underlying agent, which must be a minimum of version 4.5.11.

To use Maven Central, add the dependency to your build files in one of the following:

- Gradle:

```
dependencies {
    compile group: 'com.appdynamics.agent', name: 'agent-api', version: '4.5.18.29239'
}
```

or

- Maven:

```
<dependency>
     <groupId>com.appdynamics.agent</groupId>
     <artifactId>agent-api</artifactId>
     <version>4.5.18.29239</version>
</dependency>
```

iSDK Overview

The instrumentation SDK (iSDK) part of the Java Agent API enables you to create custom instrumentation to apply to the code in your application at runtime, and avoid making development time changes to instrument the code.

You can use the iSDK to implement interceptors making use of the imperative Java Agent APIs and apply custom instrumentation to proprietary applications or frameworks. This custom instrumentation works with the out-of-the-box instrumentation provided by the agent.

iSDK users need to create separate custom interceptor jar files that instrument their code at application startup when deployed with the Java Agent in the `sdk-plugins` directory. To remove custom instrumentation, remove the custom interceptor jar files from the agent deployment.

Use the iSDK

This procedure outlines how to use the iSDK:

1. Set up your development environment.
2. Set up your project.
3. Implement your interceptor(s).
4. Build your files.
5. Deploy.

Set Up Your Development Environment

To use the iSDK, you must write code with the external dependency of the iSDK library. To do this, these tools must be available:

- A Java IDE – Eclipse or IntelliJ. The following examples are for IntelliJ usage.
- Gradle.

Set Up Your Project

To set up your project:

1. In IntelliJ, create a new Gradle project for your work. The `build.gradle` file is created in the project directory that you specify.
2. Add the required information to your project's `build.gradle` file by following the procedure in the next section.

Install the Dependency

To use the iSDKs, do one of the following:

- Access the Agent API jar directly
- Download it from Maven Central
- Download it from the AppDynamics portal

The library version changes with each new API release, and is not tightly coupled to the version of the underlying agent, which must be a minimum of version 4.5.11.

For use with Maven Central, add the dependency to your build files in one of the following:

- Gradle:

```
dependencies {
    compile group: 'com.appdynamics.agent', name: 'agent-api', version: '4.5.18.29239'
}
```

or

- Maven:

```
<dependency>
     <groupId>com.appdynamics.agent</groupId>
     <artifactId>agent-api</artifactId>
     <version>4.5.18.29239</version>
</dependency>
```

See https://sdkdocs.appdynamics.com/javadocs/java-agent-api/v4.5/ for Javadoc reference for the Agent API.

Refresh your gradle project in intellij and run "gradlew idea" if you want intellij to reference the source properly when auto-populating methods and so on.

After this setup, you must be able to use Intellij to create a class, extend an iSDK template class, and auto-populate missing methods.

Deploy the iSDK with Agent API

To leverage Agent API functionality with the iSDK, you must package it with your custom interceptor jar. To accomplish this in Grade:

1. Create a configuration for iSDK:

```
configurations {
  isdk
}
```

2. Add the Agent-API dependency to this configuration:

```
dependencies {
    compile group: 'com.appdynamics.agent', name: 'agent-api', version: '4.5.18.29239'
        isdk group: 'com.appdynamics.agent', name: 'agent-api', version: '4.5.18.29239'
        …
}
```

3. Inside your task for generating the interceptors jar, include the `AppdynamicsAgent` class inside the jar:

```
jar {
  configurations.isdk.collect {
    it.isDirectory() ? from(it) : from(zipTree(it)) {
      include 'com/appdynamics/agent/api/AppdynamicsAgent.class'
    }
  }
  includeEmptyDirs = false
}
```

To accomplish this in Maven, you must write an Ant Task to include the AppdynamicsAgent class with your jar.

Implement Your Interceptor

Class or Method Identification

For all use cases, you must identify the class and method that you wish to instrument. This is also true when configuring custom POJO instrumentation using the User Interface but the technique is much different in the SDK.

Create Custom Interceptors and Rules

Implement Your Instrumentation Logic

The iSDK provides a class called `AGenericInterceptor`, which must be subclassed with the desired custom interceptor logic.

The agent examines .jar files deployed to the agent's `sdk-plugins` directory and looks for classes that extend `AGenericInterceptor`. These classes must implement the required abstract methods.

One key method all interceptors must implement is the `initializeRules` method, which returns a list of objects that extend the IRule interface. These IRules represent where in the code you must apply a given interceptor (class name, method name, method params, and so on).

If interceptors need to perform reflection on any of the java objects used in the instrumented code, an interface, `IReflector`, is provided to facilitate such reflection at runtime.

Important Methods to Implement in AGenericInterceptor

- `initializeRules` - Returns a list of objects that describe where in the code the interceptor must be placed
- `onMethodBegin` - Any code in the body of this method is run before the execution of the instrumented method
- `onMethodEnd` - Any code in the body of this method is run after the execution of the instrumented method

> `AGenericInterceptors` can be combined with the Agent API to start or end transactions, add data to snapshots, and so on.

These interceptors can be applied anywhere in the code, rather than just the beginning or end of a method. Also, they collect local variable information. Specific methods in the Rule builder, such as `atLineNumber`, `atField`, `atLocalVariable`, and so on only apply to this variety of iSDK interceptor.

- jars placed directly in the top-level "sdk-plugins" directory are also loaded.
- You can use the manifest attribute "Plugin-Classes: <comma separated list of fully qualified class names>" in your plug-in jar to specify specific classes to load as plug-ins. This improves jar loading performance, if the plug-in jar contains many classes that are not plug-ins.

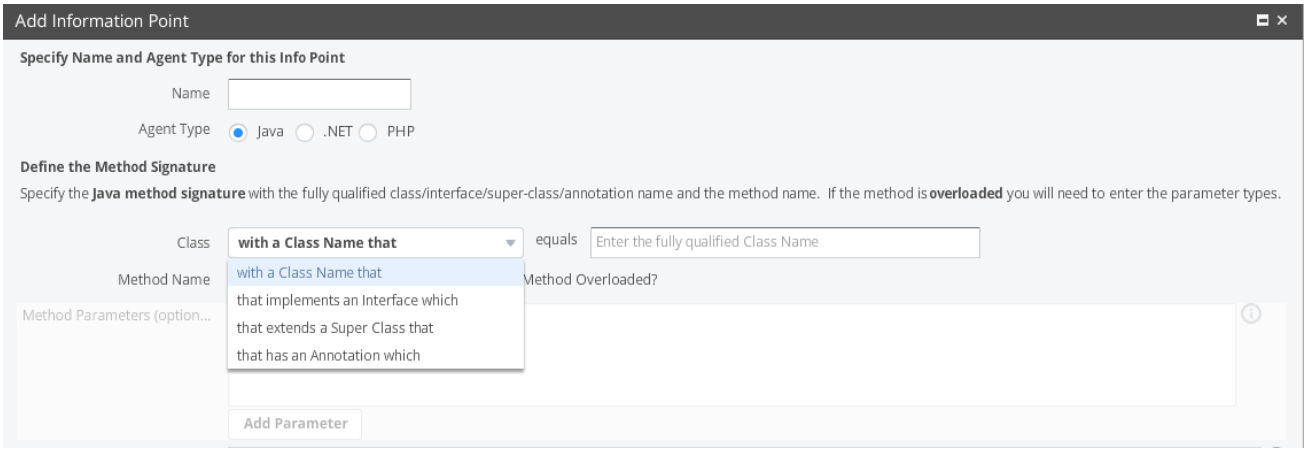
Use the Rule and Rule.Builder Classes

A `Rule` defines a set of classes to instrument. Your iSDK code returns a `java.util.List` of `Rule` objects from a method named *initializeRules*. `Rules` are created using the builder pattern – a `Rule.Builder` object is created using a match string:

```
Rule.Builder builder = new Rule.Builder("com.mynamespace.MyClass");
```

See the javadocs for all classes referenced in this section.

This is similar to what you have to do in the UI. For example, in the **Add Information Point** configuration screen, you must specify a name to match on – either a concrete class name, abstract class name, interface name or annotation name. The argument to the `Rule.Builder` constructor is the same as what you specify into the class match text box in the image below.



You then modify the `Rule.Builder` by calling APIs on the `Rule.Builder` class. `Rule.Builder` supports fluent style so that your code can be compact.

You need the following:

- Class match string: Given in the constructor to `Rule.Builder`
- Class match type: Set the `SDKClassMatchType` using `builder.setClassMatchType`
- Class match string match type: Set the `SDKStringMatchType` for the method using `builder.classStringMatchType`
- Method match string using `builder.setMethodMatchString`
- Method match type: Set the `SDKStringMatchType` for the method using `builder.methodStringMatchType`

The following is a screenshot of an initializeRules() method for creating an arbitrary interceptor. Note the following:

- The style is a mixture of fluent and iterative (lines have chained invocations, but multiple lines are still used)
- The class match is for an exact classname match to a concrete class (`SDKClassMatchType.MATCHES_CLASS` and `SDKStringMatchType.EQUALS` for `classStringMatchType`)
- The method match is for an exact method name string match (`SDKStringMatchType.EQUALS` for `methodStringMatchType`)

```
    private static final String CLASS_TO_INSTRUMENT = "com.appdynamics.util.async.TrackedCallable";
    private static final String METHOD_TO_INSTRUMENT = "linkContext";

    private static int pc = 0;

    public List<Rule> initializeRules() {

        Rule.Builder bldr = new Rule.Builder(CLASS_TO_INSTRUMENT);
        bldr = bldr.classMatchType(SDKClassMatchType.MATCHES_CLASS).classStringMatchType(SDKStringMatchType.EQUALS);
        bldr = bldr.methodMatchString(METHOD_TO_INSTRUMENT).methodStringMatchType(SDKStringMatchType.EQUALS);
        List<Rule> result =  new ArrayList<>();
        result.add(bldr.build());

        return result;
    }
```

Implement the Interceptor Logic

The Java code to be run when the custom interceptor is hit is provided in the `onMethodBegin` and/or `onMethodEnd` methods of the interceptor object.

A code example of a simple interceptor that leverages the Agent API is shown below.

```
public class SampleGenericInterceptor extends AGenericInterceptor {

    public SampleGenericInterceptor() {
        super();
    }

    @Override
    public List<Rule> initializeRules() {
        List<Rule> rules = new ArrayList<Rule>();
        rules.add(new Rule.Builder("Main").methodMatchString("genericInterceptorTarget").build());
        return rules;
    }

    public Object onMethodBegin(Object invokedObject, String className, String methodName, Object[]
paramValues) {
        System.out.print("In begin");
                AppdynamicsAgent.startTransaction("SampleGenericInterceptorBT", null, EntryTypes.POJO, false);
        Map<String, String> keyVsValue = new HashMap<String, String>();
        keyVsValue.put("foo", "bar");
        AppdynamicsAgent.getEventPublisher().publishEvent("an event happened", "INFO", "AGENT_STATUS",
keyVsValue);
        return null;

    }

    public void onMethodEnd(Object state, Object invokedObject, String className, String methodName,
            Object[] paramValues, Throwable thrownException, Object returnValue) {
        System.out.print("In end");
                Transaction currentTransaction = AppdynamicsAgent.getTransaction();
                Set<DataScope> dataScopes = new HashSet<DataScope>();
                dataScopes.add(DataScope.SNAPSHOTS); // collect data for BT snapshot.
                currentTransaction.collectData("key", "value", dataScopes);
                currentTransaction.end();
    }
}
```

Use the IReflector Interface

All iSDK classes have access to a toolbox interface called `IReflector`. This interface provides functionality that enables the use of Java reflection. Within any class that extends one of the template classes, the user can call `getNewReflectionBuilder()` to get an instance of `IReflectionBuilder`.

> `getNewReflectionBuilder()` must be called in the constructor of your plug-in class. You may hit a `NullPointerException` if it is called elsewhere.

This interface supports the following methods:

```
/**
 * load a class. classLoader specified in the IReflector.execute method
 **/
IReflectionBuilder loadClass(String className);

/**
 * Create a new object
 * @param className class to create
 * @param argTypes argument types
 * @return this
 */
IReflectionBuilder createObject(String className, String... argTypes);

/**
 * Invoke a static method
 * @param methodName method to invoke
 * @param searchSuperClass search super classes for the method. Else only declared methods are called
 * @param argTypes for overloaded methods
 * @return this
 */
IReflectionBuilder invokeStaticMethod(String methodName, boolean searchSuperClass, String... argTypes);

/**
 * Similar to invoke static method, invoke a instance method
 * @param methodName method to invoke
 * @param searchSuperClass search super classes for the method. Else only declared methods are called
 * @param argTypes for overloaded methods
 * @return this
 */
IReflectionBuilder invokeInstanceMethod(String methodName, boolean searchSuperClass, String... argTypes);

/**
 * Access a field
 * @param fieldName field name to access
 * @param searchSuperClass should search super class for the field name
 * @return this
 */
IReflectionBuilder accessFieldValue(String fieldName, boolean searchSuperClass);
These can be called in any order, essentially forming a getter chain. After crafting your chain of calls, the
build() method must be called, returning an IReflector.

The IReflector is used by invoking its execute() method:

/**
 * Execute the reflector chain
 * @param classLoader required - classloader referenced to load classes
 * @param target The steps will start executing on this object. null if first step is loadClass or staticMethod
 * @param params params to be passed to the steps
 * @return returns with correct cast
 * @throws ReflectorException
 */
<E> E execute(ClassLoader classLoader, Object target, Object[]... params) throws ReflectorException;

/**
 * Execute the reflector chain
 * @param classLoader required - classloader referenced to load classes
 * @param target The steps will start executing on this object. null if first step is loadClass or staticMethod
 * @param params params to be passed to the steps
 * @return returns with correct cast
 * @throws ReflectorException
 */
<E> E execute(ClassLoader classLoader, Object target, OperationParams params)
 throws ReflectorException;
```

The use of these reflectors is often critical for getting the desired functionality (extracting values from payloads, decorating objects with correlation headers, and so on).

Java Agent API Documentation

See the AppDynamics APM Agent API for the Java Agent.

Java Agent API User Guide

Related Pages

- AppDynamics APIs
- Data Collectors
- Add Custom Fields to Business Transactions Using the Java SDK

This page provides examples that describe how you can implement common use cases for the Agent API. See the javadoc at https://sdkdocs.appdynamics.com/javadocs/java-agent-api/v4.5/.

Common Use Cases

The following are the common use cases for the Agent API.

Start and End a Synchronous Business Transaction

The following example shows code that starts a Business Transaction called 'Checkout' whenever the method `checkout()` is called. The Business Transaction ends when the method returns. Encapsulating the method body in a try or finally block ensures that you end the Business Transaction even if the method itself throws an exception or otherwise terminates without reaching the end.

```
public String checkout(List<ItemOrders> orders) {
   Transaction transaction = null;
   try {
        transaction = AppdynamicsAgent.startTransaction("Checkout", null, EntryTypes.POJO, false);

        /******************
         * Method Body Here
         ******************/
    } finally {
        if (transaction != null) {
           transaction.end();
        }
    }
}
```

Alternatively, you can use try-with-resources pattern:

```
public String checkout(List<ItemOrders> orders) {
   try (Transaction transaction = AppdynamicsAgent.startTransaction("Checkout", null, EntryTypes.POJO, false)) {
        /******************
         * Method Body Here
         ******************/
    }
}
```

In this case, the Business Transaction ends when the try block closes.

Start and End an Asynchronous Business Transaction

This example shows code that starts a Business Transaction called 'CheckoutAsync' whenever the method `checkoutAsync()` is called. The originating segment created for the Business Transaction ends when the method `endSegment()` is called on the Business Transaction, or when it is closed if used in a try-with-resources construct. Encapsulating the method body in a try/finally block ensures that we end the segment even if the method itself throws an exception or otherwise terminates without reaching the end.

```
//The thread where the Business Transaction starts
public String checkoutAsync(List<ItemOrders> orders) {
   Transaction transaction = null;
   try {
        transaction = AppdynamicsAgent.startTransaction("CheckoutAsync", null, EntryTypes.POJO, true);
                //mark handoff to link this segment with the end segment
                transaction.markHandoff(commonPayload);

        /******************
         * Method Body Here
         ******************/
    } finally {
        if (transaction != null) {
           transaction.endSegment();
        }
    }
}
```

Alternatively, the try-with-resources pattern is supported:

```
//The thread where the Business Transaction starts
public String checkoutAsync(List<ItemOrders> orders) {
   try (Transaction transaction = AppdynamicsAgent.startTransaction("CheckoutAsync", null, EntryTypes.POJO,
true)) {
                //mark handoff to link this segment with the end segment
                transaction.markHandoff(commonPayload);

        /******************
         * Method Body Here
         ******************/
    }
}
```

This ends the segment in the thread where the Business Transaction was started when the try block closes. The Business Transaction itself needs to be ended in the method where the asynchronous Business Transaction ends:

```
//The thread where the Business Transaction ends
public String checkoutAsyncEnd(List<ItemOrders> orders, Transaction transaction, Object commonPayload) {
   //link to the originating segment
   Transaction transactionSegment = AppdynamicsAgent.startSegment(commonPayload);

   /******************
   * Method Body Here
   ******************/
   if (transactionSegment != null) {
      transactionSegment.endSegment();
   }
   if (transaction != null) {
      transaction.end();
   }
}
```

Sett the Transaction Name After a Start

The following example shows how to set the transaction name after it has been started. This is useful if you want to split transactions based on events that happen in your application:

> Transaction name can only be set after a start on the originating segment, and if no async hand-offs or exits have been made.

```
public String checkout(List<ItemOrders> orders) {
   AppdynamicsAgent.startTransaction("Checkout", null, EntryTypes.POJO, false);
   /******************
    * Method Body Here
    ******************/
   if (orders.isEmpty()) {
              AppdynamicsAgent.setCurrentTransactionName("Empty Cart");
   }
   AppdynamicsAgent.getTransaction().end();
}
```

Start a Transaction Using OOTB Servlet Naming Rules

The following example shows how to leverage automatic servlet naming rules to name a transaction:

```
public String checkout(List<ItemOrders> orders) {
        Map<String, String> headers = new HashMap<String, String>();
        headers.put("header1", "headerValue1");
        Map<String, String[]> parameters = new HashMap<String, String[]>();
        Map<String, Object> cookies = new HashMap<String, Object>();
        cookies.put("chip", "ahoy");
        ServletContext.ServletContextBuilder contextBuilder = new ServletContext.ServletContextBuilder()
        .withURL(new URL("https://www.yourstore.com/endpoint/to/checkout"))
        .withParameters(parameters)
        .withHeaders(headers)
        .withCookies(cookies)
        .withSessionAttributes(new HashMap<String, Object>())
        .withRequestMethod("GET")
        .withHostValue("hostValue")
        .withHostOriginatingAddress("hostOriginatingAddress");
        Transaction t = AppdynamicsAgent.startServletTransaction(contextBuilder.build(), EntryTypes.HTTP, null,
false);
   /******************
    * Method Body Here
    ******************/
   t.end();
}
```

Define an Exit Call

For an `inventoryServer.verifyQuantities(orders)`, which makes a request to another process, you can monitor that request as an Exit Call. This helps to continue monitoring the Business Transaction through the call to the downstream server and identify the time spent in the remote service. You can do this by modifying the method as follows:

```
public void verifyQuantities(List<ItemOrders> orders) {
    ExitCall exitCall = null;
    try {
        exitCall = AppdynamicsAgent.getTransaction().startExitCall("Quantity Check", "Inventory Server",
ExitTypes.HTTP, false);

       /******************
        * Method Body
        ******************/
    } finally {
        if (exitCall != null) {
            exitCall.end();
        }
    }
}
```

Alternatively, if you want to use HTTP Backend detection rules, you can use the `startHttpExitCall` method:

```
public void verifyQuantities(List<ItemOrders> orders) {
    ExitCall exitCall = null;
    try {
         Map<String, String> identifyingProps = new HashMap<String, String>();
         URL url = new URL("https://www.appdynamics.com/solutions/microservices/");
           exitCall = AppdynamicsAgent.getTransaction().startExitCall(identifyingProps, url, false);
           /******************
           * Method Body
           ******************/
     } finally {
        if (exitCall != null) {
            exitCall.end();
        }
     }
}
```

The above code modifications define the Exit Call that manifests it as a remote service in the controller. To tag and follow the request into an instrumented downstream tier, add a correlation header:

```
public void verifyQuantities(List<ItemOrders> orders) {
    ExitCall exitCall = null;
    try {
        exitCall = AppdynamicsAgent.getTransaction().startExitCall("Quantity Check", "Inventory Server",
ExitTypes.HTTP, false);

        // Generate the appdynamics correlation header
        String correlationHeader = exitCall.getCorrelationHeader();

        // ... Method code including request creation

        // Modify the request/payload to include the correlation header
        inventoryRequest.addHeader(AppdynamicsAgent.TRANSACTION_CORRELATION_HEADER, correlationHeader);

    } finally {
        if (exitCall != null) {
            exitCall.end();
        }
    }
}
```

The following example shows how to instrument an asynchronous exit:

```
public class StartEndAsyncExitCall {

    private void makeAsyncExitTransaction() throws Exception {
        System.out.println("Starting transaction");
        try (Transaction transaction = AppdynamicsAgent.startTransaction("StartEndAsyncExitCall", null,
EntryTypes.POJO, true)) {
            Thread.sleep(1000);
            new TransactionEndingThread(transaction).start();
            System.out.println("Starting exitcall");
            ExitCall exitCall = null;
            exitCall = transaction.startExitCall("Custom", "Custom", ExitTypes.CUSTOM, true);
            new ExitCallEndingThread(exitCall).start();
        }
    }

    public class TransactionEndingThread extends Thread {
        Transaction transaction = null;

        TransactionEndingThread(Transaction transaction) {
            this.transaction = transaction;
        }

        public void run() {
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            transaction.end();
            System.out.println("Ending transaction");
        }
    }

    public class ExitCallEndingThread extends Thread {
        ExitCall exitCall = null;

        ExitCallEndingThread(ExitCall exitCall) {
            this.exitCall = exitCall;
        }

        public void run() {
            try {
                Thread.sleep(50);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            exitCall.end();
            System.out.println("Ending exitcall");
        }
    }
}
```

Define an Asynchronous Thread Handoff

If your checkout method also does a thread handoff and executes some business logic that you want to monitor in a separate thread, register the worker thread with the Business Transaction:

```
public String checkout(List<ItemOrders> orders) {
    Transaction transaction = null;
    try {
        transaction = AppdynamicsAgent.startTransaction("Checkout", null, EntryTypes.POJO, false);

        // ... Method code

        // Custom thread handoff using custom queue
        asyncTaskQueue.add(task);

    } finally {
        if (transaction != null) {
           transaction.end();
        }
    }
}
```

To instrument this, modify the add method to mark a thread handoff and then start a new segment where the thread begins running:

```
public class AsyncTaskQueue {
    public void add(Task task) {
        AppdynamicsAgent.getTransaction().markHandoff(task);

        /******************
         * Method Body
         ******************/
}
```

```
public class Task {
    public void run() {
        Transaction transaction = null;
        try {
            transaction = AppdynamicsAgent.startSegment(this);

            /******************
             * Method Body
             ******************/
        } finally {
            if (transaction != null) {
                transaction.endSegment();
            }
        }
    }

    public void cancel() {
        AppdynamicsAgent.cancelHandoff(this);

        /******************
         * Method Body
         ******************/
    }
}
```

The task object is used by the agent to link the segments. Correlating thread segments using the agent API requires that the agent is running in the Executor mode.

Add Data to Snapshot or Analytics

In many instances, there are values of interest in the code that are helpful to add to the snapshot. This aids in the root cause diagnosis of issues or to send to the AppDynamics Business Transaction analytics to help answer real-time business-oriented questions about your application. Data reported using this API appears in the same way as it had been collected with a Method Invocation Data collector.

To report a total checkout amount to Business Transaction analytics and have it presented in the APM snapshots, use the following code:

```
private static final Set<DataScope> dataScopeSet = new HashSet(Arrays.asList(DataScope.ANALYTICS, DataScope.
SNAPSHOTS));

public String checkout(List<ItemOrders> orders) {
    Transaction transaction = null;
    try {
        transaction = AppdynamicsAgent.startTransaction("Checkout", null, EntryTypes.POJO, false);

        // ... Method code

        double shoppingCartTotal = total(orders);
        transaction.collectData("cart total", Double.toString(shoppingCartTotal), dataScopeSet);

    } finally {
        if (transaction != null) {
           transaction.end();
        }
    }
}
```

Define a Custom Metric or Event

It can also be useful to report a value as a custom metric:

```
public String checkout(List<ItemOrders> orders) {

    // ... Method code

    double shoppingCartTotal = total(orders);
    AppdynamicsAgent.getMetricPublisher().reportSumMetric("Cart Total", (long) shoppingCartTotal);

}
```

It is recommended to provide a metric path along with the metric name in the code with format: `Server|Component:<TierName>|Custom Metrics|<MetricName>,` while publishing the metric.

Reporting custom metrics and events is possible irrespective of the Business Transaction context.

.NET Agent

To monitor .NET applications with AppDynamics, you install the .NET Agent on the servers where the applications run. You only need to install the agent once per server even to monitor more than one application on the server. This page describes how to install the .NET Agent for IIS applications using the Getting Started wizard in the Controller.

For alternative approaches, see:

- Install the .NET Agent for Windows, if you downloaded the agent directly from the AppDynamics Download Center.
- Configure the .NET Agent for Windows Services and Standalone Applications, if you monitor Windows services or standalone applications.

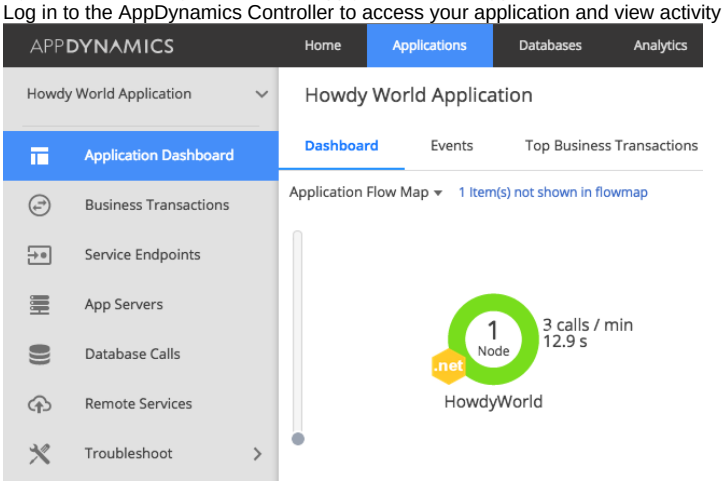
Before You Begin

1. Verify support for your environment at .NET Supported Environments.
2. Confirm that you have access to a compatible Controller. See Agent and Controller Compatibility.
3. Confirm the connection settings to the Controller where your agent will report data:
    - If you use a SaaS Controller, AppDynamics sent you the Controller hostname in your Welcome Email. Use port 443 for HTTPS, or port 80 for HTTP.
    - If you use an on-premises Controller, you supplied the hostname and port during installation.
4. Verify that you have access to the machine (where the application runs) as a user account with privileges to install the agent software, and restart the application.
5. Verify that the machine where the application runs can connect to the Controller. Proxies or firewalls on the network between the agent and Controller may require additional configuration.

Install the .NET Agent

To install the agent for IIS applications:

1. Log in to the Controller UI and access the Getting Started Wizard for .NET.
2. Follow the steps in the wizard to configure and download the agent. The wizard guides you through some preliminary configuration steps. When finished, the wizard enables you to download the agent as a ZIP archive named (in format of): `dotNetAgent-Portal-<architecture>-<version>.zip`.
3. Extract the agent archive on the destination computer.
4. Launch an elevated command prompt with full administrator privileges.
5. Execute the Installer.bat file you extracted. The batch file installs the agent and starts the AppDynamics Agent Coordinator service.
6. Restart IIS.
7. If you are in a testing environment, apply load to your application to view the activity in the Controller UI.
8. Log in to the AppDynamics Controller to access your application and view activity similar to:



Next, you can install more agents, or begin monitoring your application. See AppDynamics Essentials.

.NET Supported Environments

Supported Runtime Environments

This section lists the environments where the .NET Agent does some automatic discovery after little or no configuration. See Browser RUM Supported Environments for additional supported environments.

OS Versions

- Microsoft Windows Server 2012
- Microsoft Windows Server 2012 R2
- Microsoft Windows Server 2016
- Microsoft Windows Server 2019
- Microsoft Windows 8, 8.1, 10

Microsoft .NET Frameworks

Microsoft .NET Framework 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.2, 4.6, 4.7, and 4.8 on these runtime environments:

- Microsoft IIS 6.0, 7.0, 7.5, 8.0, 8.5, 10
- Managed Windows Services
- Managed Standalone Applications
- Microsoft SharePoint 2010, 2013 as services running inside IIS
- Microsoft ASP.NET Core 2.1 for Windows

Microsoft .NET Core and Microsoft .NET

Microsoft .NET Core 2.1, 3.1, and Microsoft .NET 5.0 with:

- Microsoft ASP.NET Core 2.1, 3.1, and Microsoft .NET 5.0 for Windows for .NET Agent >= 20.3

Microsoft Windows Azure

- Azure App Services for .NET 4.6 environments in the Azure Portal
  - Web Apps
  - Web Jobs
  - API Apps
  - Container Services

  For Azure App Services, the .NET Machine Agent disables certain .NET Machine Agent infrastructure monitoring features: CLR crash reporting, machine snapshots, and Windows performance counter monitoring.

- Azure Cloud Services
  - Web Roles
  - Worker Roles

Unsupported Frameworks

- Microsoft .NET 1.0 and 1.1
- Unmanaged native code

Automatically Discovered Business Transactions

The .NET Agent discovers business transactions for the following frameworks by default. The agent enables detection without additional configuration.

Type	Custom Configuration Options?	Downstream Correlation?
ASP.NET*	Yes	Yes
ASP.NET MVC 2 ASP.NET MVC 3 ASP.NET MVC 4 ASP.NET MVC 5	Yes	Yes
ASP.NET Core on the full framework	Yes	Yes
Open Web Interface for .NET (OWIN) web API	Yes	Yes
.NET Remoting	No	See Enable Correlation for .NET Remoting.
Windows Communication Foundation (WCF)	No	Yes

Web Services including SOAP	No	Yes
Message Queues		
Apache ActiveMQ NMS framework and related MQs	No	Yes
IBM WebSphere MQ	No	Yes
Microsoft Message Queuing (MSMQ)	No	Yes
Microsoft Service Bus / Windows Azure Service Bus	No	Yes
NServiceBus over MSMQ or RabbitMQ transport	No	Yes
RabbitMQ	Yes	Yes
TIBCO Enterprise Message Service	No	Yes
TIBCO Rendezvous	No	Yes
Windows Azure Queue	No	No

***** The .NET Agent automatically discovers entry points for ASP.NET web forms with the Async property set to "true" in the Page directive.

Supported Loggers for the .NET Agent

- Log4Net
- NLog
- System Trace
- Windows Event Log
- Loggers on .NET Core that implement the Microsoft.Extensions.Logging.ILogger API (Linux Agent >= 4.5.19 and Windows Agent >= 21.2.0)

If you are using a different logger, see Error Detection.

Remote Service Detection

The .NET Agent automatically detects these remote service types. The agent enables detection by default. You do not need to perform extra configuration.

Type	Custom Configuration Options?	Async Detection?*	Downstream Correlation?
CosmosDB: • v2.x (Microsoft.Azure.DocumentDB.Core) • v3.x (Microsoft.Azure.Cosmos) (Linux Agent >= 20.9.0, and Windows Agent >= 21.2.0)	No	See Asynchronous Exit Points for .NET.	N/A
Directory Services, including LDAP	No	No	N/A
HTTP	Yes	See Asynchronous Exit Points for .NET.	Yes
MongoDB: C# and .NET MongoDB Driver version 1.10, 2.0	No	See Asynchronous Exit Points for .NET.	N/A
.NET Remoting	Yes	No	See Enable Correlation for .NET Remoting.
WCF	Yes	See Asynchronous Exit Points for .NET.	Yes
WCF Data Services	Yes	No	No
Web Services, including SOAP	Yes	See Asynchronous Exit Points for .NET.	Yes
Azure Service Fabric Remoting v1 and v2—for the .NET Microservices Agent	-	-	-
Data Integration			
Microsoft BizTalk Server 2010, 2013	No	Yes	See Correlation Over Microsoft BizTalk.
Message Queues			
Apache ActiveMQ NMS framework and related MQs	Yes	No	Yes
IBM WebSphere MQ (IBM XMS)	Yes	No	Yes

Microsoft Message Queuing (MSMQ)	Yes	See MSMQ Backends for .NET	See MSMQ Backends for .NET
Microsoft Service Bus / Windows Azure Service Bus	No	Async exit points only	Yes
NServiceBus over MSMQ or RabbitMQ transport	No	See NServiceBus Backends for .NET	Yes
RabbitMQ	See RabbitMQ Backends for .NET	No	Yes
TIBCO Enterprise Message Service	Yes	No	Yes
TIBCO Rendezvous	Yes	No	Yes
Windows Azure Queue	No	No	No

* The agent discovers asynchronous transactions for the Microsoft .NET 4.5 framework. See Asynchronous Exit Points for .NET

Supported Windows Azure Remote Services

Type	Customizable Configuration?	Downstream Correlation?
Azure Blob	No	No
Azure Queue	No	No
Microsoft Service Bus	No	Yes

Cache Clients

Type	Customizable Configuration?	Async Detection?*	AppD for Databases?
StackExchange.Redis	No	Yes	No

Data Storage Detection

The .NET Agent automatically detects the following data storage types. The agent enables detection by default. You do not need to perform extra configuration.

Type	Customizable Configuration?	Async Detection?*	AppD for Databases?
ADO.NET (see supported clients below)	Yes	Yes	No
Windows Azure Blob Storage	No	Yes	No
Windows Azure File Storage	No	Yes	No
Windows Azure Table Storage	No	Yes	No

* The agent discovers asynchronous transactions for the Microsoft .NET 4.5 framework. See Asynchronous Exit Points for .NET

Supported ADO.NET Clients

AppDynamics can monitor any ADO.NET client version and type. Clients we've tested include the following:

Database Name	Database Version	Client Type
Oracle	10, 11, 12	ODP.NET
Oracle	10, 11, 12	Microsoft Provider for Oracle
MySQL	5.x	Connector/Net and ADO.NET
Microsoft SQL Server*	2005, 2008, 2012	ADO.NET

Install the .NET Agent for Windows

Related pages:

- Configure the .NET Agent
- Unattended Installation for .NET
- Upgrade the .NET Agent for Windows
- Download AppDynamics Software

To monitor IIS applications, Windows services, or standalone applications, install the AppDynamics .NET Agent once on each machine that hosts managed .NET applications. At startup, the agent initializes an individual instance of itself for each application running in the CLR.

This page describes a new installation for the .NET Agent using the MSI package.

- To install from the command line, see Unattended Installation for .NET
- To install the AppDynamics Site Extension for Windows Azure, see Install the AppDynamics Azure Site Extension for .NET
- To install the AppDynamics .NET Microservices Agent using one of the new AppDynamics NuGet Packages, see .NET Microservices Agent
- To upgrade, see Upgrade the .NET Agent

Installation Overview

1. Prepare to Install.
2. Install the agent.
3. Configure the agent.
4. Restart instrumented applications.
    - For Microsoft IIS, the configuration utility provides an option to restart IIS or not.
    - If you do not restart IIS, monitoring does not begin until the next time IIS restarts.
    - You must restart Windows services and standalone applications manually.

Prepare to Install

Review these requirements before you install the .NET Agent:

- You must enable Windows Management Instrumentation service on the machine where you are installing the .NET Agent, and the service must be running.
- To run the .NET Agent and the Standalone Machine Agent on the same machine, you must enable .NET Compatibility Mode on both the machine and .NET Agent. See .NET Compatibility Mode.

Install the .NET Agent

1. Download the MSI installer package from the AppDynamics Download Center.
2. Run the MSI installer package.
3. Read the End User Agreement and click to accept. Then, click **Next**.
4. (Optional) On the Confirm location panel, change the .NET Agent installation directories, including:
    - Destination directory for the .NET Agent executables and supporting files.
    - Parent directory for local data including agent configuration files and log files.

   See .NET Agent Directory Structure for the default directories.

   > You can also change the destination of the logs directory later using the AppDynamics Agent Configuration utility.

5. On the User Account Control panel, click **Yes** to allow the installer to make changes to the computer. If the current account does not have administrator privileges, the installer prompts you to supply the password for an administrator account.
6. Wait for the installation to complete.
7. For new installs, AppDynamics recommends you launch the AppDynamics Agent Configuration utility.

If you encounter problems installing, see Troubleshoot .NET Agent Issues

Configure the .NET Agent

Launch the AppDynamics Agent Configuration utility to configure the .NET Agent.

- For Microsoft IIS applications, see Configure the .NET Agent
- For Windows services or standalone applications, see Configure the .NET Agent for Windows Services and Standalone Applications

Configure the .NET Agent

Related pages:

- Install the .NET Agent for Windows
- Name .NET Tiers
- Configure the .NET Agent for Windows Services and Standalone Applications
- Unattended Installation for .NET

After you install the AppDynamics .NET Agent, configure the agent based on the types of applications to monitor:

- For Microsoft IIS applications, use the configuration utility with either automatic or manual tier naming.
- For Windows services or standalone applications, use the configuration utility, and then manually update the config.xml. See Configure the .NET Agent for Windows Services and Standalone Applications.

You can instrument any combination of IIS Applications, Windows services, and standalone applications on a single server. Run the configuration utility to configure IIS, then follow the instructions for Windows services and Standalone applications.

Use the AppDynamics Agent Configuration Utility to configure the agent immediately after installation, or make changes to existing agent configurations. Th e utility configures the agent for one machine at a time.

Prepare to Configure the .NET Agent

- Install the .NET Agent for Windows.
- You need these account credentials based on your installation type:
    - Single-tenant: Membership in a role with the View License account level permission. See Create and Manage Custom Roles.
    - Multi-tenant or SaaS: Your AppDynamics Welcome Email that contains your account credentials.
- Uninstall any pre-existing profiler, such as Ant, VS 2010 Performance Tools, or others. The configuration utility alerts you if it finds a pre-existing profiler.
- Determine your tier naming scheme:
    - Automatic
    - Manual
    - Pre-configured using Config Management in the Controller

> The configuration utility must restart IIS to apply configurations. The utility offers you the option to restart IIS or not. If you choose not to restart, configurations apply the next time IIS restarts.

File System Security Settings

These Windows accounts require specific file system permissions:

- The account you use to run your web application as defined by its application pool, or the Windows service account.
- The account you use to run the AppDynamics Agent Coordinator, by default the Local System account.

The required permissions are:

- Write permission to the .NET App Agent logs directory.
  Default: Windows Server 2008 and later: `%ProgramData%\AppDynamics\DotNetAgent\Logs`
- Read and Execute permissions to the .NET App Agent install directory.
  Default: `C:\Program Files\AppDynamics\AppDynamics .NET Agent`
- Read and Execute permissions in the web application installation directory.
  For example: `C:\inetpub\wwwroot\myapp`

Configure the .NET Agent

1. Start the Agent Configuration Utility. In the Windows pane, select **AppDynamics > .NET Agent > AppDynamics Agent Configuration**.
   If a warning message displays, click **Yes** to exit and uninstall any pre-existing profiler.

   > Use the Registry editor to check the Windows registry and ensure that the uninstall process cleaned up the registry entries. Use the warning message to identify any undeleted profiler environment variables. See Troubleshoot .NET Agent Issues.

2. If the configuration utility detects legacy agent configurations from the .NET Agent <= 3.7.7, it displays the **Upgrade Configuration** pane. Choose either:
    - **Yes** to remove legacy configurations. See Upgrade the .NET Agent for Windows.

      > Removing legacy configurations modifies `web.config` files and causes IIS to restart affected applications.

- **No** to retain legacy configurations.

When the utility discovers no further profiler conflicts, or after any configuration clean up, it displays the welcome pane.

3. From the Log directory permissions pane, you can optionally change the location of the log directory. Click **Change** and select a new location.
4. If needed, enter an account to grant log directory permissions. Click **Add**. If a warning message displays, verify that the account is valid on the system. The wizard confirms the list of accounts.
5. From the Controller Configuration pane, enter the Controller access information and credentials.
    - The AppDynamics Agent Configuration utility only supports the configuration of one Controller and business application per server.
        - Use tiers to organize different applications you instrument on a single server, or
        - Manually configure support for multiple business applications. See Configure Multiple Business Application Support for .NET.
    - Enter the Controller server name, or IP and port number.
    - For single-tenant accounts, enter your Account Access Key. Locate your credentials under **Settings > License** in the Controller.
    - For multi-tenant accounts, click **Multi-Tenant Controller**. Enter the Account Name and Account Access Key as provided to you by AppDynamics.
    - For a secure connection, click **Enable SSL**.
        - Make sure you have secured the Controller with a trusted certificate. See Enable SSL for the .NET Agent.
        - When you enable SSL, the agent secures communication to the Controller using the protocols set for `ServicePointManager.SecurityProtocol` in your application.
        - By default, the configuration utility enables TLS 1.2, making it the first option in the list of secure protocols. This affects all secure communications from your application, not just requests to the AppDynamics Controller. To disable TLS 1.2, click to deselect this option.
    - If needed, enter the HTTP proxy information. Proxies that use authentication require additional configuration.
6. Click **Test Controller Connection** to verify the connection.
   The Application Configuration pane displays existing business application information from the Controller and Controller connection status.
7. Configure the business application for the Controller:
    - If you already have a business application on the Controller, click **Existing Applications from the Controller** and click the appropriate business application. If you have not defined business applications in the Controller, the utility displays an empty list.
    - Click **New Application** to define a new business application. Be careful about spellings and capitalization and note down the exact name. Ampersands are not supported in application names.
    - To use Config Management in the Controller, click **Register Machine with No Application**.
8. Click **Next** to advance to Assign IIS applications to tiers where you can use one of the tier naming options.

Automatically Name IIS Tiers

1. In the Assign IIS applications to the tiers pane, click **Automatic**.
2. If prompted, click **OK** to confirm the Automatic configuration. The configuration utility summarizes the configuration settings.
3. By default when you click **Next**, the configuration utility restarts IIS.

> If you do not want to apply the configuration right away, uncheck the box. The Agent Configuration Utility saves the information and applies it the next time you restart IIS.

4. If you proceed and click **Next**, the configuration utility logs its activities, including stopping and restarting IIS, and reports any problems. Review the summary for any issues in red font. The green font indicates the logged events. The summary shows any Warnings (W) or Errors (E).
5. Click **Done**.

Manually Name IIS Tiers

1. From the Assign IIS applications to tiers pane, click **Manual**, then click **Next**.
2. Assign IIS Applications to AppDynamics tiers by selecting a tier on the right and a business application on the left. The utility highlights the assigned tier in boldface.

> For large IIS installations, use the Max IIS tree depth pulldown to display all the projects. A large tree depth may take some time to view.

   To create new tiers, enter a name and click **Add Tier**.
3. When you are done click **Next**.
4. On the **Configuration Summary** pane, uncheck **Restart IIS** if you do not want to immediately restart IIS. You may restart later to apply your changes, or they will take effect after a reboot.
5. If you proceed and click **Next**, the Configuration Utility logs its activities, including stopping and restarting IIS, and reports any problems.
6. Review the configuration log summary. As it applies the configuration, AppDynamics generates a log of the configuration activities and displays a summary. Review the summary for any issues in red font. The green font indicates the logged events. The summary shows any Warnings (W) or Errors (E).
7. When finished, click **Next**. The wizard completes.

Use Config Management

1. From the Assign IIS applications to the tiers pane, click **Register Machine with No Application**, and then click **Next**.

2. From the Configuration Summary pane, uncheck **Restart IIS** if you do not want to immediately restart IIS. You may restart later to apply your changes, or they will take effect after a reboot.
3. Follow the instructions on Manage Configuration for .NET to assign a configuration from the Controller. The .NET Agent and .NET Machine Agent do not report metrics to the Controller until after you assign a configuration using the Config Management tool.

See Troubleshoot .NET Agent Issues.

.NET Agent SSL Support

When a .NET Agent establishes a secure (SSL) connection with a Controller, the .NET Agent uses a default mechanism embedded in the .NET framework to verify a Controller (server) certificate which relies on the local Trust Store. As a result, the Controller certificate must be signed by one of the publicly trusted authorities.

Therefore, if the Controller uses:

- a self-signed certificate, or
- a certificate signed by a custom authority

Then the SSL connection cannot be established until:

- a self-signed certificate, or
- a custom authority are manually added to the local Trust Store.

> If a Controller uses a certificate signed by a publicly trusted authority, then no extra set up is required.

If you do not want to manually add a certificate to the local Trust Store, you can configure the .NET Agent to consume the Controller custom trusted certificate and establish a secured connection automatically.

Limitations

Custom trusted certificate validation is supported with these noted limitations for the .NET runtimes:

- .NET Core >= 2.0
- .NET Framework >= 4.7.2
- .NET Standard >= 2.1

> You may use applications that were complied using earlier .NET versions as long as the .NET runtime is later than or equal to the listed limitations.

If you reach a runtime limitation, then you can either:

- Upgrade the .NET runtime to a supported version, or
- Add a Controller custom certificate to the local Trust Store, disable the feature, and then continue using the SSL connection to a Controller based on the default validation mechanism.

Supported Configurations

You can configure the Controller Custom Trusted Certificates by providing either a:

- Path to one certificate file: A file may contain one, or multiple valid certificates, in one of these formats:
  - Base-64 Encoded X.509 (.cer)
  - DER Encoded Binary X.509 (.cer)
  - PKCS 7 (.p7b)
  - PKCS 12 (.pfx)

Or

- Folder containing multiple certificate files: A folder may contain several files with one or multiple certificates inside; and the folder may contain subfolders. There is no limitation on the structure except that all files in the folder must contain valid certificates; if one file is not a valid certificate, then all of them are ignored.

Select only one of these configurations. If you configure both a certificate file and a certificate folder, then a warning displays alerting you that only the certificate file will be used in the configuration.

> You are not required to provide a private key in your certificate files; only a public key is required. AppDynamics does not expose any sensitive information from the provided certificates. Use caution when providing files to an outside source.

Standalone Windows Agent Configuration

Configure the `config.json` file:

```
{
        "controller": {
                "certfile": "C:\certs\certificate.crt",
                "certdir": "C:\certs\",
        }
}
```

Environment Variables Used for Standalone Windows Agent

Using the command line, set these environment variables to override the `config.json` file:

```
APPDYNAMICS_CONTROLLER_SSL_CERTFILE=C:\certs\certificate.crt
APPDYNAMICS_CONTROLLER_SSL_CERTDIR=C:\certs\
```

Agent MSI Agent Configuration

Configure the `config.json` file without overriding the environment variables:

```
<appdynamics-agent xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <controller ssl-certificate-file = "C:\certs\certificate.crt" ssl-certificate-directory = "C:\certs\" >
    ...
  </controller>
  ...
</appdynamics-agent>
```

Parity Difference with Linux .NET Agent <= 20.11.x

These are the existing parity differences with Linux .NET Agent <= 20.11.x:

- Certificate files formats are extended.
- Multiple certificates are supported in files.
- There is no name constraint for the files.
- There is no constraint on the folder structure.
- Full framework is supported.
- You are not required to provide a full chain of the Controller certificates. You can configure the .NET Agent trust using an end leaf certificate, or just a custom authority certificate, or both; whichever is convenient.

Enable SSL for the .NET Agent

Related pages:

- Secure the Platform
- Administer the .NET Agent
- .NET Agent Configuration Properties

This page describes how to configure the AppDynamics .NET Agent to connect to the Controller with SSL.

Requirements

Before you configure the agent to enable SSL, gather this information:

- Identify the Controller SSL port:
  - For SaaS Controllers, the SSL port is 443.
  - For on-premises Controllers, the SSL port is 8181 by default, but it is possible to configure on-premises Controllers to use other ports at installation.
- Identify the signature method for the Controller's SSL certificate:
  - A publicly known certificate authority (CA) signed the certificate. This applies for Verisign, Thawte, and other commercial CAs.
  - A CA internal to your organization signed the certificate. Some companies maintain internal certificate authorities to manage trust and encryption within their domain.
  - .NET Agent supports self-signed certificates when these conditions exist:
    - The Common Name (CN) on the certificate matches the URL of the Controller that the agent is calling.
    - The public key for the self-signed certificate is installed on the Windows Trusted Root Certification Authorities store where the agent is installed.

Establish Trust for the Controller's SSL Certificate

The .NET Agent requires that the Common Name (CN) on the Controller certificate matches the DNS name of the Controller. Additionally, certificates for the root CA that signed the Controller's SSL certificate must reside in the Windows Trusted Root Certification Authorities store for the Local Computer.

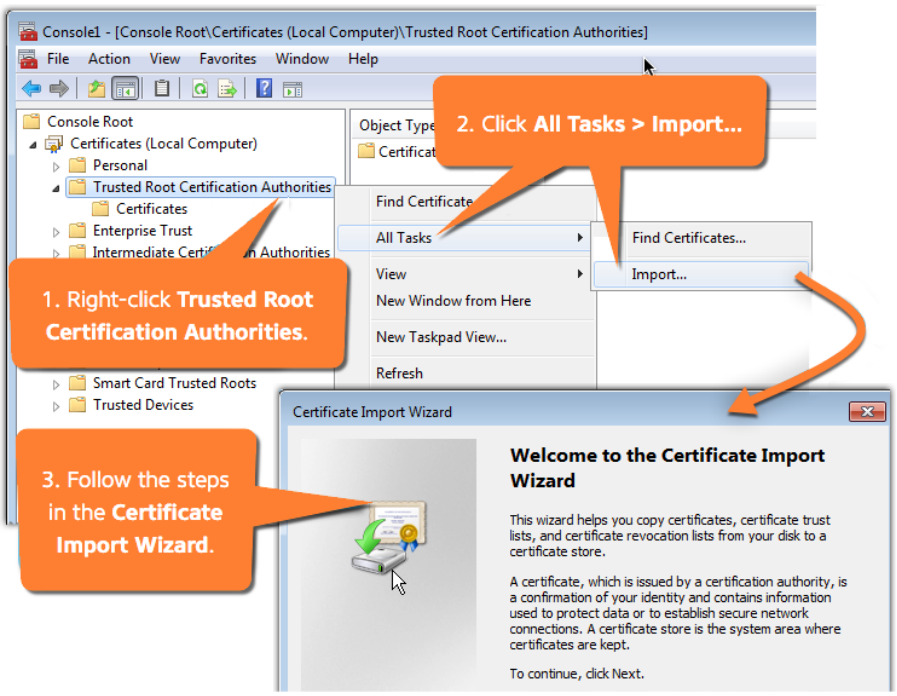
Certificates Signed by a Publicly Known Certificate Authority

The root certificates for most publicly trusted CA signing authorities, such as DigiCert, Verisign, Thawte, and other commercial CAs, are in the Trusted Root Certification Authorities store by default.

Certificates Signed by an Internal Certificate Authority

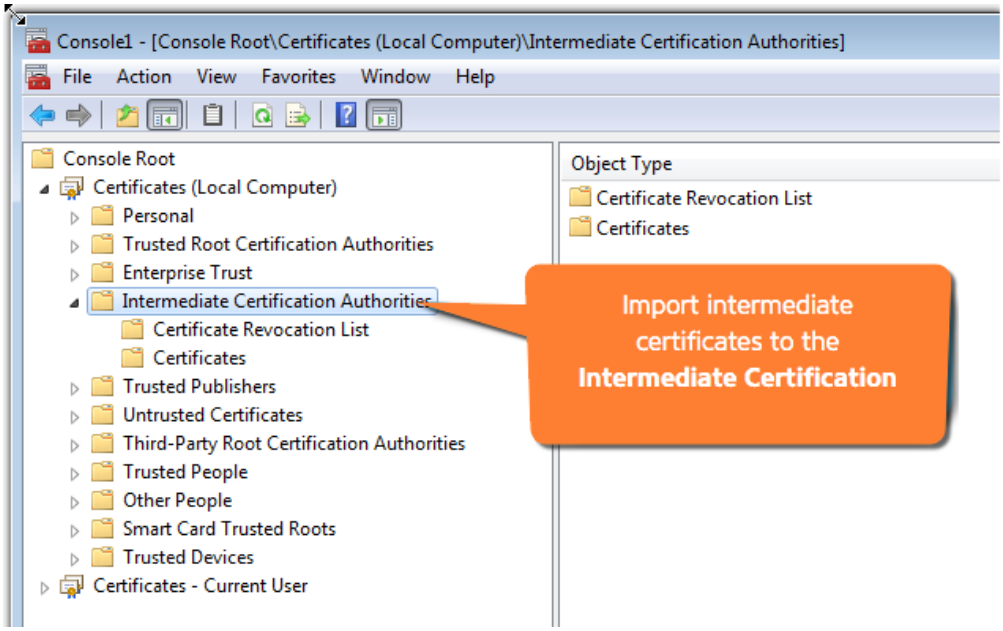
If your organization uses internal CA to sign certificates, you may need to obtain the root CA certificate from your internal security management resource. To import the root certificate, see Adding Certificates to the Trusted Root Certification Authorities store for a Local Computer.

This example shows how to use the Certificate snap-in for the Microsoft Management Console to import a certificate for a Trusted Root Certification Authority:



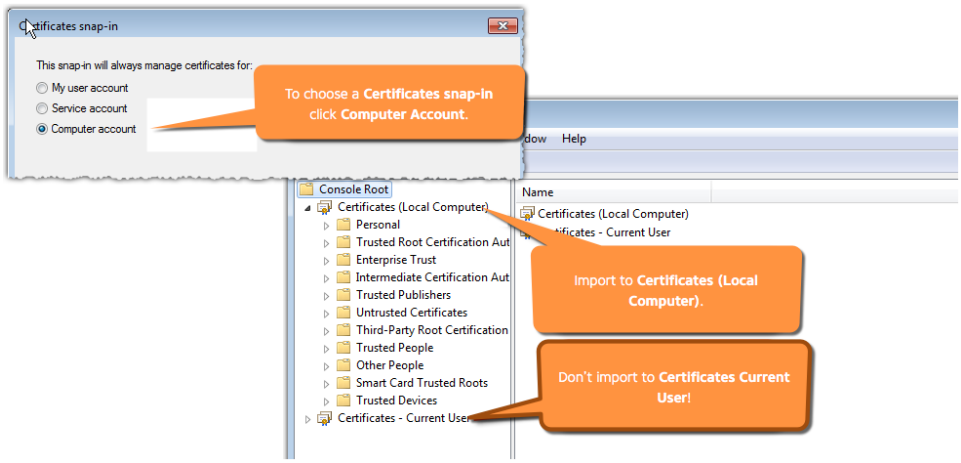
If an intermediate CA signed the Controller certificate, you must import the certificate for the intermediate CA in addition to the one for the root CA that signed the intermediate CA certificate. If your Controller is publicly accessible, you can use a certificate checker to identify the certificates required to complete the trust chain.

This example shows the **Intermediate Certification Authorities** store:



Certificate Management Tips

- If you imported certificates for a root or intermediate CA, verify the certificate store where you imported them. Import them to **Certificates (Local Computer)**.



- The AppDynamics SaaS Controller uses certificates signed by DigiCert. In some cases, SaaS customers must import the DigiCert root certificates into the **Windows Trusted Root Certification Authorities** store.
- In some cases system administrators set up group policies that require external certificates be imported to the **Third-Party Root Certification Authorities** store. If importing the certificate for the root CA to the **Windows Trusted Certification Authorities** store is not successful, then try the **Third-Party Root Certification Authorities** store.

Enable SSL for the .NET Agent

You can update the SSL settings for the agent by:

- Using the AppDynamics Agent Configuration Utility, or
- Editing the settings directly in the config.xml

When you enable SSL for the .NET Agent, you automatically enable SSL for the .NET Machine Agent.

Configure SSL Using the AppDynamics Agent Configuration Utility

1. Launch the AppDynamics Agent Configuration utility.
2. In **Controller Configuration**, set the **Port Number** to the SSL port for the Controller.
    - For a SaaS Controller, set the **Port Number** to `443`.
    - For an on-premises Controller, set the **Port Number** to the on-premises SSL port. The default is `8181`.
3. Click **Enable SSL**.
    - When you enable SSL, the agent secures communication to the Controller using the protocols set for `ServicePointManager.SecurityProtocol` in your application.
    - By default, the configuration utility enables TLS 1.2, making it the first option in the list of secure protocols. This affects all secure communications from your application, not just requests to the AppDynamics Controller. To disable TLS 1.2, click to deselect this option.
4. Click **Next** and proceed with the rest of the panes to complete the configuration.
5. Restart instrumented applications: IIS applications or application pools, Windows services, and standalone applications.

If you use automatic tier configuration, restart IIS. For example, open a command prompt and enter:

```
iisreset
```

Upon restart, the agent connects with the Controller via SSL.

Configure SSL Using config.xml

1. Open the config.xml file as administrator. See Administer the .NET Agent.
2. Update the following SSL settings:
    - Controller port attribute: set to the on-premises SSL port. The default is `8181`. See Controller Port Attribute.
    - Controller SSL attribute: set to `true`. See Controller SSL Attribute. When you enable SSL, the agent secures communication to the Controller using the protocols set for `ServicePointManager.SecurityProtocol` in your application.
    - Controller enable TLS 1.2 attribute: Optionally set to `true` to add TLS 1.2 as the first option in the list of protocols. This affects all secure communications from your application, not just requests to the Controller.
3. Save your changes.
4. Restart the `AppDynamics.Agent.Coordinator` service.
5. Restart instrumented applications: IIS applications or application pools, Windows services, and standalone applications.

If you use Automatic configuration, restart IIS. For example, open a command prompt and run:

```
iisreset
```

Upon restart, the agent connects with the Controller via SSL.

Sample SaaS SSL config.xml Configuration

```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001
/XMLSchema">
  <controller host="mycompany.saas.appdynamics.com" port="443" ssl="true" enable_tls12="true">
    <application name="MyDotNetApplication" />
  </controller>
...
</appdynamics-agent>
```

Sample On-Premises SSL config.xml Configuration

```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001
/XMLSchema">
  <controller host="mycontroller.mycompany.com" port="8181" ssl="true" enable_tls12="true">
    <application name="MyDotNetApplication" />
  </controller>
...
</appdynamics-agent>
```

Troubleshooting Issues

If you verified all prerequisites and still have communication issues, verify that the default ciphers are enabled in Windows Server.

Check this Registry key:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers\
```

If subkeys exist, your operations team may have disabled certain ciphers.

Encrypt Credentials in .NET Agent Configuration

Related pages:

- Unattended Installation for .NET
- Upgrade the .NET Agent for Windows

By default, the AppDynamics .NET Agent stores these credentials in the `config.xml` file:

- Controller account authentication
- Proxy server authentication

For environments where security policies require you to secure credentials stored on disk, you can run an unattended installation that encrypts the credentials for the .NET Agent and writes them to the Windows Credential Manager.

> ⚠ Storing credentials for the .NET Agent using the Windows Credential Manager updates the `config.xml` file to use a schema which the AppDynamics Agent Configuration utility does not support. If you follow these instructions, then you cannot use the configuration utility to make configuration changes afterward. If you launch the configuration utility on a server where you have stored credentials in the Windows Credential Manager, the utility prompts you to delete the configurations.

Requirements

- You must run the `AppDynamics.Agent.Coordinator` service as the `LocalSystem` account.
- To modify credentials after installation, you need Windows Sysinternals.

Setup Configuration File

You must generate a setup configuration file to run an unattended installation. See 'Setup Configuration File Properties' on Unattended Installation for .NET.

New Installation

For new installations, use one of these methods to create the setup configuration file:

- Run the AppDynamics Agent Configuration utility from the command line and pass the `-s` parameter to specify the setup configuration file destination. For this option, you must execute the .NET Agent MSI installer package on one machine before running the configuration utility.

  ```
  %ProgramFiles%\AppDynamics\AppDynamics .NET Agent\AppDynamics.Agent.Winston.exe -s <path to setup configuration file>
  ```

- Manually create a setup configuration file from a sample template.

Remove any plain-text authentication elements from the setup configuration file. You pass the credentials as part of the unattended installation command:

- Controller Account element:
  `<account name="myaccount" password="myaccesskey"/>`
- Proxy Authentication element: If you are using a proxy authentication, use this format in the setup configuration file.

  ```
  <proxy host="myproxy.example.com" port="3128" enabled="true">
     <authentication enabled="true" domain="mydomain.com"/>
  </proxy>
  ```

Upgrade

If your upgrade meets the criteria for an in-place upgrade on Upgrade the .NET Agent for Windows, you can encrypt the credentials for the .NET Agent and upgrade the agent at the same time.

Copy the AppDynamics Agent element from your existing `config.xml` file to the setup configuration file. Remove any plain-text authentication elements from the setup configuration file. You pass the credentials as part of the unattended installation command:

- Controller Account element:
  `<account name="mycontroller.saas.appdynamics.com" password="myaccesskey"/>`
- Proxy Authentication element:
  `<authentication enabled="true" user_name="my_proxy_user" password="password" domain="my_windows_domain"/>`

Sample Setup Configuration File

This example shows a setup configuration file that instruments: two IIS Applications, `MainBC` and `SampleHTTPService`; a Windows service, `BasicWindowsService`; and a standalone application, MyStandaloneApp.exe.

```
<winston>
  <logFileDirectory directory="C:\ProgramData\AppDynamics\DotNetAgent\Logs" />
  <logFileFolderAccessPermissions defaultAccountsEnabled="false">
    <account name="NT AUTHORITY\LOCAL SERVICE" displayName="LOCAL SERVICE" />
    <account name="NT AUTHORITY\SYSTEM" displayName="SYSTEM" />
    <account name="NT AUTHORITY\NETWORK SERVICE" displayName="NETWORK SERVICE" />
    <account name="IIS_IUSRS" displayName="ApplicationPool Identity" />
  </logFileFolderAccessPermissions>
  <appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001
/XMLSchema">
    <controller host="mycontroller.appdyanmics.com" port="443" ssl="true">
      <application name="My Business Application" />
    </controller>
    <machine-agent />

    <app-agents>
      <IIS>
        <applications>
          <application path="/" site="MainBC">
            <tier name="Main Site" />
          </application>
          <application path="/" site="SampleHTTPService">
            <tier name="HTTP Services" />
          </application>
        </applications>
      </IIS>
      <standalone-applications>
         <standalone-application name="BasicWindowsService" args="-x">
            <tier name="Windows Service Tier"/>
         </standalone-application>
         <standalone-application executable="MyStandaloneApp.exe">
            <tier name="Standalone App" />
         </standalone-application>
      </standalone-applications>
    </app-agents>

  </appdynamics-agent>
</winston>
```

Install from the Command Line

To install the .NET Agent from the command line:

1. Download the .NET Agent MSI Installer Package from the AppDynamics Download Center.
2. Launch an elevated command prompt with full administrator privileges. See Start a Command Prompt as an Administrator.

   > Logging on to Windows as a member of the Administrators group does not grant sufficient permissions to run the installer.

3. Stop IIS and, if you are upgrading, stop instrumented Windows services and Standalone applications.
4. Run this command to install the agent with encrypted credentials. See command line options for descriptions.

   ```
   msiexec /i <path_to_MSI_installer_package> /l log.txt /q AD_SETUPFILE=<path_to_setup_configuration_file>
   AD_SECURED_CREDENTIALS=true AD_CONTROLLER_ACCOUNT_NAME=<SaaS or multi-tenant account>
   AD_CONTROLLER_ACCOUNT_ACCESS_KEY=<access key> AD_PROXY_USERNAME=<proxy user name>
   AD_PROXY_PASSWORD=<proxy password>
   ```

   For example:

   ```
   msiexec /i "%USERPROFILE%\Downloads\dotNetAgentSetup.msi" /l log.txt /q AD_SETUPFILE="%USERPROFILE%
   \Documents\SetupConfig.xml" AD_SECURED_CREDENTIALS=true AD_CONTROLLER_ACCOUNT_NAME=MyAppDynamicsAccount
   AD_CONTROLLER_ACCOUNT_ACCESS_KEY=changeme AD_PROXY_USERNAME=MyProxyUser AD_PROXY_PASSWORD=ProxyPass
   ```

   The MSI installer package installs the .NET Agent and encrypts the credentials and writes them to the Windows Credential Store. It adds the Controller secure attribute to the Controller element in the `config.xml` file and sets the value to `true`:

```
<controller host="mycontroller.appdyanmics.com" port="443" ssl="true" secure="true">
```

5. Start IIS. Restart or start instrumented Windows services and standalone applications.

Update Credentials in the Windows Credential Manager

The .NET Agent includes a Credentials Tool for you to modify credentials stored in the Windows Credential Manager. To change credentials under the Local System account, you need to use PsExec to launch the command prompt.

1. If you have not already, download and install Windows Sysinternals.
2. Use `PsExec` to launch a command prompt as the Local System account.

```
psexec -i -s cmd.exe
```

3. Run the Credentials Tool and pass the updated credentials.

```
"%programfiles%\AppDynamics\AppDynamics .NET Agent\AppDynamics.CredentialsTool.exe"
AD_CONTROLLER_ACCOUNT_NAME=<SaaS or multi-tenant account> AD_CONTROLLER_ACCOUNT_ACCESS_KEY=<access key>
AD_PROXY_USERNAME=<proxy user name> AD_PROXY_PASSWORD=<proxy password>
```

Command Line Options

- `AD_SECURED_CREDENTIALS`: Set to `true` to encrypt credentials to the Windows Credential Store and configure the agent to use the encrypted credentials.
- `AD_CONTROLLER_ACCOUNT_NAME`: The account name for the SaaS or multi-tenant Controller.
- `AD_CONTROLLER_ACCOUNT_ACCESS_KEY`: The account access key for the SaaS or multi-tenant Controller.
- `AD_PROXY_USERNAME`: The proxy server user account.
- `AD_PROXY_PASSWORD`: The password for the proxy server user account.

Private Key and Client Certificate for .NET Agents

To enable mutual SSL authentication, the .NET Agent loads the client certificate and private key from your local computer's personal Certificate Store. The instrumented application identities need the ability to access the private key of the client certificate.

Import the Private Key into the Certificate Store

To import the private key into the Certificate Store:

1. From your Microsoft Management Console (MMC), navigate to and expand **Certificates (Local computer) > Personal > Certificates**.
2. Right-click **Certificates**.
3. Expand **All Tasks > Import**.
4. From the Import panel, change the Filter to **All** or **.pfx** files.
5. Select the *.pfx file.
6. Enter your password and then complete the import procedure.

By default, for IIS applications (Full Framework and .NET Core hosted in- and out-of-process), the `IIS_IUSRS` group must have read access to the private key. For standalone, self-hosted .NET Core, and IIS applications running on application pools using custom service accounts, the appropriate accounts and groups have read access to the private key.

To add read access to the private key:

1. From your Microsoft Management Console (MMC), navigate to and expand **Certificates (Local computer) > Personal > Certificates**.
2. Right-click **Certificates**.
3. Expand **All Tasks > Manage Private Keys** to display a popup.
4. Add the application identities or user groups (as needed) and then add read access to the private key.

Configure the Certificate Attribute

To use the certificate from the Certificate Store, you must add the certificate thumbprint as the `certificate` attribute.

To obtain the certificate thumbprint:

a. From your Microsoft Management Console (MMC), navigate to and expand **Certificates (Local computer) > Personal > Certificates**.
b. Double-click the certificate.
c. From the **Details** tab, locate the thumbprint.
d. Add the thumbprint as the `certificate` attribute:

Configure the .NET Agent for Windows Services and Standalone Applications

Related pages:

- Configure the .NET Agent
- .NET Agent Configuration Properties

The .NET Agent automatically instruments IIS applications only. However, you can follow these instructions to manually configure the .NET Agent to instrument Windows services or standalone .NET applications.

If your web application is not configured to run as an in-process handler, the application may run as `dotnet.exe` or `mywebappapplication.exe`. In this scenario, the application's site name and application path are not discovered automatically. You must configure these applications manually, and provide tier and node names that best describe the application.

Before You Begin

To instrument Windows services and standalone .NET applications, edit the .NET Agent configuration file, `config.xml`.

> If your application runs using the default domain, you must manually instrument it. AppDynamics does not automatically instrument the default domain out-of-the-box. To determine whether you should use the default domain, and how to configure it, see Instrument the DefaultDomain for Standalone Applications.

Before starting, verify that the services or applications you want to instrument are .NET applications rather than native applications or another type of application. From a command line, enter:

```
tasklist /m "mscor*"
```

> This command is for 64-bit processes only. For 32-bit processes, use the process explorer to view the dependencies of the process, and determine if there are any .NET .dll files loaded into the file.

The output lists the processes with DLLs starting with mscor*, indicative of .NET processes. Processes not displaying on the list are not .NET processes, and you cannot instrument them with the .NET Agent.

If you have previously instrumented IIS applications on the server that hosts the Windows services and standalone applications, the server should already have a `config.xml` file that you can edit. If not, perform these steps to generate one:

1. Install the .NET Agent for Windows.
2. Run the AppDynamics Agent Configuration utility.

To avoid instrumenting IIS applications, select the manual tier naming approach and omit the step of assigning tiers for the IIS application. This disables instrumentation for the IIS applications and enables you to instrument only the intended Windows services or standalone applications.

The utility performs these configuration tasks:

1. Changes the location of the logs directory, and assigns permissions.
2. Configures and tests connectivity to the Controller.
3. Sets the business application for the agent.

Manually Configure the .NET Agent

Once you have configured the Controller properties for the .NET Agent, instrument your Windows service or standalone application by adding the Standalone Applications element to the `config.xml`.

1. Edit the `config.xml` file as an administrator. See Administer the .NET Agent.

   If you have not instrumented any IIS applications, the file contains the minimal configuration for the Controller connectivity and the Machine Agent. Verify the Controller properties and the Business Application name (shown in this example):

```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org
/2001/XMLSchema">
  <controller host="mycontroller.example.com" port="8090" ssl="false">
    <application name="My Business Application" />
  </controller>
...
</appdynamics-agent>
```

If you have already instrumented IIS applications, those configurations display under the IIS element.

2. Add a single standalone applications element, `standalone-applications` under the app-agents element, and under the `standalone-applications` element, add a standalone application element, `standalone-application` for each Windows service or standalone application to instrument. For example:

```
...
<app-agents>
   ...
   <standalone-applications>
      <standalone-application executable="MyStandaloneApp.exe">
        <tier name="Standalone Tier 1" />
      </standalone-application>
      <!-- Instrument a standalone application using a partial path. -->
      <standalone-application executable="MyApplication\MyOtherStandaloneApp.exe">
        <tier name="Standalone Tier 2" />
      </standalone-application>
      <!-- Instrument a Windows service using arguments. -->
      <!-- The following example matches the command "MyWindowsService.exe -d -x -r". -->
      <standalone-application executable="MyWindowsService.exe" command-line="-x">
         <tier name="Windows Service Tier" />
      </standalone-application>
   </standalone-applications>
</app-agents>
...
```

In the standalone application element configuration:

- Use the `tier` element to assign the instrumented application to a tier in the Controller. See .NET Agent Configuration Properties.
- Identify the executable file of the application in the Standalone Application element `executable` attribute using one of these formats:
  - Executable name: For example, `MyStandaloneApp.exe` or `MyWindowsService.exe`. The file extension is optional, so `MyStandaloneApp` is valid. If you use `dotnet.exe` to start the .NET Core application, then specify `"dotnet.exe"` as the application executable (see Sample Configuration File).
  - Full or partial path to the executable: For example, `C:\Program Files\MyApplication\MyStandaloneApp.exe` or `MyApplication\MyStandaloneApp.exe`. Use the full or partial path to assign different AppDynamics tiers to separate instances of the same executable file running from different paths.
  - To differentiate between two instances of the same executable, specify any unique portion of the command line invocation format of the application, such as an argument, in the Standalone Application command-line attribute.

You can discover the path to a Windows service executable in the Services panel of the administrative tools. In Services, click the service and select **Properties**. The path display in the General tab.

3. Restart the `AppDynamics.Agent.Coordinator` service.
4. Restart the Windows service or standalone application.
5. If your Windows service or standalone application does not implement an auto-detected framework, you must configure a POCO entry point for a class or method in your service for the agent to begin instrumentation.

Sample Configuration File

This sample `config.xml` shows instrumentation for a Windows service and standalone application:

```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001
/XMLSchema">
  <controller host="mycontroller.mycompany.com" port="8090" ssl="false">
    <application name="My Business Application" />
  </controller>
  <machine-agent />
  <app-agents>
    <IIS>
        <applications />
    </IIS>
    <standalone-applications>
        <standalone-application executable="MyWindowsService.exe" command-line="-x">
           <tier name="Windows Service Tier" />
        </standalone-application>
        <standalone-application executable="MyStandaloneApp.exe">
           <tier name="Standalone Tier" />
        </standalone-application>
    </standalone-applications>
  </app-agents>
</appdynamics-agent>
```

If the .Net Core application is published as an executable, then you can configure it similar to .NET Framework applications by specifying the executable name, and optionally, the command line. However, if the application is published as a library, then you start it by by invoking the `dotnet.exe` tool.

This sample configuration shows how to instrument a .NET Core standalone application hosted by the `dotnet.exe`:

```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001
/XMLSchema">
  <controller host="mycontroller.mycompany.com" port="8090" ssl="false">
    <application name="My Business Application" />
  </controller>
  <machine-agent />
  <app-agents>
    <IIS>
        <applications />
    </IIS>
    <standalone-applications>
        <standalone-application executable="dotnet.exe" command-line="MyApp.dll"> //command-line option
instructs the agent to monitor only "MyApp"
           <tier name="DotNet Core Tier" />
        </standalone-application>
      </standalone-applications>
  </app-agents>
</appdynamics-agent>
```

Troubleshooting

Intermittent Loss of Windows Services Instrumentation

When instrumenting Windows Services, there may be cases where the instrumented service initializes before the `AppDynamics.Agent.Coordinator` service. When this happens, the Profiler shuts down and no instrumentation occurs. This problem is uncommon.

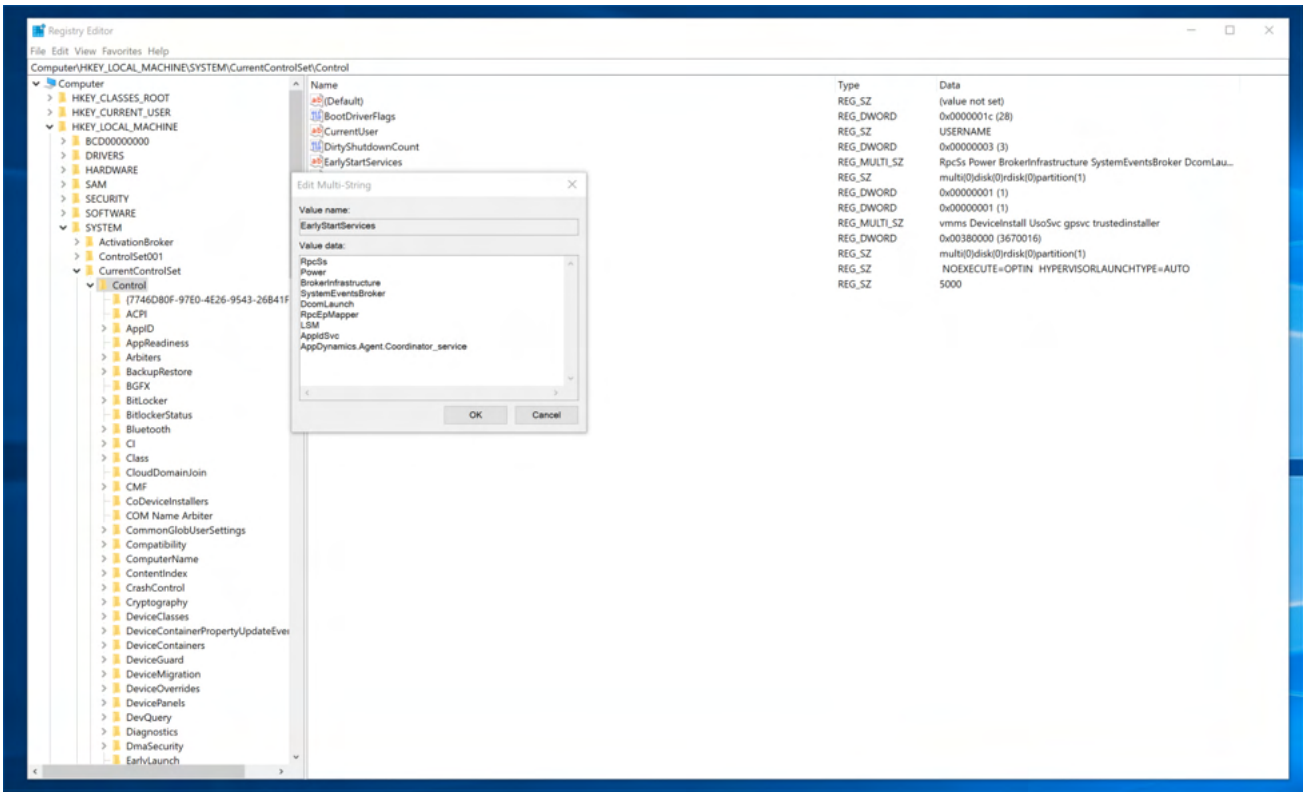
These procedures describe how to overcome this problem for Windows 2008, and Windows >= 2012.

Windows 2008

If you are using Windows 2008, please contact AppDynamics Support for assistance.

Windows >= 2012

To ensure the AppDynamics.Agent.Coordinator service initializes before the instrumented Windows Service, you must modify the registry.
Edit the Reg key `EarlyStartServices @HKLM:\SYSTEM\CurrentControlSet\Control`, and add AppDynamics.Agent.Coordinator_service to the list of early start services.


Registry Editor
File Edit View Favorites Help
Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control
Computer
HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
BCD00000000
DRIVERS
HARDWARE
SAM
SECURITY
SOFTWARE
SYSTEM
ActivationBroker
ControlSet001
CurrentControlSet
Control
{7746D80F-97E0-4E26-9543-26B41F
ACPI
AppID
AppReadiness
Arbiters
BackupRestore
BGFX
BitLocker
BitlockerStatus
Bluetooth
CI
Class
CloudDomainJoin
CMF
CoDeviceInstallers
COM Name Arbiter
CommonGlobUserSettings
Compatibility
ComputerName
ContentIndex
CrashControl
Cryptography
DeviceClasses
DeviceContainerPropertyUpdateEve
DeviceContainers
DeviceGuard
DeviceMigration
DeviceOverrides
DevicePanels
DevQuery
Diagnostics
DmaSecurity
EarlyLaunch
Name
(Default)
BootDriverFlags
CurrentUser
DirtyShutdownCount
EarlyStartServices
Type
Data
REG_SZ
(value not set)
REG_DWORD
0x0000001c (28)
REG_SZ
USERNAME
REG_DWORD
0x00000003 (3)
REG_MULTI_SZ
RpcSs Power BrokerInfrastructure SystemEventsBroker DcomLau...
REG_SZ
multi(0)disk(0)rdisk(0)partition(1)
REG_DWORD
0x00000001 (1)
REG_DWORD
0x00000001 (1)
REG_MULTI_SZ
vmms DeviceInstall UsoSvc gpsvc trustedinstaller
REG_DWORD
0x00380000 (3670016)
REG_SZ
multi(0)disk(0)rdisk(0)partition(1)
REG_SZ
NOEXECUTE=OPTIN HYPERVISORLAUNCHTYPE=AUTO
REG_SZ
5000
Edit Multi-String
Value name:
EarlyStartServices
Value data:
RpcSs
Power
BrokerInfrastructure
SystemEventsBroker
DcomLaunch
RpcEptMapper
LSM
AppIdSvc
AppDynamics.Agent.Coordinator_service
OK
Cancel

Name .NET Tiers

Related pages:

- Configure the .NET Agent
- Name .NET Nodes

In AppDynamics, a tier represents a service in your application environment, such as an ASP.NET front end, WCF service, .NET web service, or standalone application. Tiers display on flow maps, so you should name your tiers that is logical and easy to understand for your users.

Use the AppDynamics .NET Agent Configuration Utility to map IIS sites to tiers (see Configure the .NET Agent). Some manual configuration options require you to edit the `config.xml` file (see Administer the .NET Agent).

Name IIS Tiers Automatically

Name IIS tiers automatically using the configuration utility. Select Automatic on the Assign IIS applications to the tiers pane. The .NET Agent instruments all IIS sites except the Default Web Site and names the tiers using this scheme:

`IIS site/app`

> The agent omits `app` when the application is the root application for the IIS site.

Use this option when:

- You are new to AppDynamics and the .NET Agent.
- You want to instrument all IIS applications, and your team understands the form of IIS application names.

> Automatic Tier Naming for the .NET Agent does not automatically assign a virtual application to a tier. To map virtual applications to tiers, see Name IIS Tiers Manually.

Name Azure Tiers Automatically

The .NET Agent automatically names Azure tiers using these schemes:

- Cloud Services: `Azure role name`
- App Services: `Azure site name`

To customize tier naming for Azure, see Name IIS Tiers Manually.

Name IIS Tiers Manually

The .NET Agent provides two options to name IIS tiers manually, using the configuration utility, or editing the config.xml file directly.

Use the Configuration Utility

On the Assign IIS applications to tiers pane, select **Manual**. The configuration utility enables you to create new tiers and assign IIS applications to tiers.

Use this option when:

- You do not want to instrument all IIS applications on the server
- You want custom tier names
- You want to assign multiple applications to a single tier

Edit the config.xml Directly

Use this option to customize Azure tier names, or when it is not possible to use the configuration utility.

For each IIS tier to instrument, add an application element as a child element of the IIS applications element in the `config.xml` file. You can specify a static IIS tier name ,or a regular expression (which is helpful for variable Azure site names). For the full syntax and an example, see .NET Agent Configuration Properties.

For example, to add all Azure sites that begin with `MvcWebRole` to a tier named `My Azure Tier` using a regular expression:

```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <controller host="nativedemocon1.cloudapp.net" port="8090" ssl="false">
    <application name="AzureEmailService" />
    <account name="customer1" password="APJC234bcd$123" />
  </controller>
  <machine-agent />
  <app-agents azure="true" azure-role-name="" azure-role-instance-id="">

    <IIS>
        <applications>
              <!-- Configure IIS tier names with a regular expression. -->
            <application path="/" site="MvcWebRole.*" site-regex="true">
                 <tier name="My Azure Tier"/>
            </application>
        </applications>
    </IIS>
    <standalone-applications>
         <standalone-application executable="WaWorkerHost.exe">
            <tier name="" />
        </standalone-application>
    </standalone-applications>
  </app-agents>
</appdynamics-agent>
```

Name Windows Service or Standalone Application Tiers

See Configure the .NET Agent for Windows Services and Standalone Applications on how to name tiers for each instrumented Windows service or standalone application manually in the config.xml.

Name .NET Nodes

Related pages:

- Configure the .NET Agent
- Name .NET Tiers

IIS Node Naming

By default, AppDynamics .NET Agent names nodes use a combination of the Windows machine name, the tier name, and the name of the .NET application, as shown in this format:

```
<machine NetBIOS name>-<tier>-<IIS site>/<app>
```

However, you may omit elements in the name under these conditions:

- The app name is omitted when the application is the root application for the IIS site.
- The tier name is omitted when the tier name is the same as the IIS site name.

These examples show IIS node naming.

Example 1

```
WIN-86M7CEJO6P5-Order Server-OrderSvc
```

- `WIN-86M7CEJO6P5` is the machine NetBIOS name.
- `Order Server` is the tier name.
- `OrderSvc` is the IIS site name. The application is the site root, so the agent omits the application name.

Example 2

```
WIN-86M7CEJO6P5-Order Server-Store/ProcessOrder
```

- `Store` is the IIS site name.
- `ProcessOrder` is the application name within the site.

Different .NET versions of the same application have their own versions of the CLR and run on independent processes. Therefore, the agent identifies the two processes as different nodes.

IIS Web Gardens

The syntax for web gardens is the same as IIS Nodes, except that the agent appends a zero-based process index to differentiate the worker processes:

```
<machine NetBIOS name>-<tier>-<IIS site>/<app>-<process index>
```

When IIS first launches web garden processes, the agent assigns a sequential index to each process. However, when IIS recycles a process, the agent re-uses the available index freed by the terminated process. Therefore, there is no correlation between the index sequence and the chronological start of the process.

Sometimes more nodes display than the maximum number of worker processes. This occurs when a long-running request prevents a process from shutting down before its replacement launches.

Windows Service or Standalone Application Nodes Naming

By default, the agent names Windows service and standalone application nodes as:

```
<machine NetBIOS name>-<tier>-<Windows service name or executable name>
```

The agent omits tier when the tier name is the same as the service name or executable name.

These examples show Windows service and standalone application naming.

Example 1

```
WIN-86M7CEJO6P5-MyWindowsService
```

- `WIN-86M7CEJO6P5` is the machine name.
- `MyWindowsService` is the Windows service name.

Example 2

```
WIN-86M7CEJO6P5-MyStandaloneApp.exe
```

- `WIN-86M7CEJO6P5` is the machine name.
- `MyStandaloneApp.exe` is the executable file name.

Unattended Installation for .NET

Related pages:

- Configure the .NET Agent
- .NET Agent Configuration Properties
- Configure the .NET Agent for Windows Services and Standalone Applications

The .NET Agent provides a command-line unattended installation when you have multiple servers that require the same AppDynamics configuration. Using unattended installation, you only configure once, and then use the command-line scripts to automate installation and instrumentation on multiple servers

To install and configure the agent manually, see Install the .NET Agent for Windows.

Create a Setup Configuration File

You can use the .NET Agent MSI installer package to specify the path to a setup configuration file to perform an unattended installation. The setup configuration file contains all the properties you need to enable instrumentation for your .NET applications.

You must run the .NET Agent MSI installer package on one machine before you can use the AppDynamics Agent Configuration utility to create a setup configuration file. See Install the .NET Agent for Windows.

Setup configuration files created in previous versions of the AppDynamics Agent Configuration utility work with the 4.0 installer.

1. Launch the AppDynamics Agent Configuration utility from the command line. Use the `-s` parameter to specify the setup configuration file destination.

   ```
   %ProgramFiles%\AppDynamics\AppDynamics .NET Agent\AppDynamics.Agent.Winston.exe -s <path to setup configuration file>
   ```

   For example:

   ```
   %ProgramFiles%\AppDynamics\AppDynamics .NET Agent\AppDynamics.Agent.Winston.exe -s "c:\temp\configurationSavedSetupConfiguration.xml"
   ```

2. Complete the steps of the configuration wizard step.

   The configuration utility saves the setup configuration file to the path you specified.

   > The configuration utility only configures instrumentation for IIS applications.

3. (Optional) To perform the unattended installation for Windows services or for standalone applications, you must edit the setup configuration file manually. See Configure the .NET Agent for Windows Services and Standalone Applications.

Sample Setup Configuration File

This example shows a setup configuration file that instruments: two IIS Applications, MainBC and SampleHTTPService; a Windows service, MyWindowsService.exe; and a standalone application, MyStandaloneApp.exe.

The configuration file sets the log directory to `C:\ProgramData\AppDynamics\DotNetAgent\Logs`, and grants write permission to four accounts.

```
<winston>
  <logFileDirectory directory="C:\ProgramData\AppDynamics\DotNetAgent\Logs" />
  <logFileFolderAccessPermissions defaultAccountsEnabled="false">
    <account name="NT AUTHORITY\LOCAL SERVICE" displayName="LOCAL SERVICE" />
    <account name="NT AUTHORITY\SYSTEM" displayName="SYSTEM" />
    <account name="NT AUTHORITY\NETWORK SERVICE" displayName="NETWORK SERVICE" />
    <account name="IIS_IUSRS" displayName="ApplicationPool Identity" />
  </logFileFolderAccessPermissions>
  <appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001
/XMLSchema">
    <controller host="mycontroller.mycompany.com" port="8090" ssl="false">
      <application name="My Business Application" />
      <account name="customer1" password="changeme" />
    </controller>
    <machine-agent />
    <app-agents>
      <IIS>
        <applications>
          <application path="/" site="MainBC">
            <tier name="Main Site" />
          </application>
          <application path="/" site="SampleHTTPService">
            <tier name="HTTP Services" />
          </application>
        </applications>
      </IIS>
      <standalone-applications>
         <standalone-application executable="MyStandaloneApp.exe">
            <tier name="Standalone App" />
         </standalone-application>
         <standalone-application executable="MyWindowsService.exe">
            <tier name="Windows Service" />
         </standalone-application>
      </standalone-applications>
    </app-agents>
  </appdynamics-agent>
</winston>
```

Unattended Installation

Before you perform an unattended installation, verify these requirements:

- Microsoft Distributed Transaction Coordinator (MSDTC): MSDTC must run under the NT Authority\NetworkServices account. See Verify MSDTC on the Troubleshoot .NET Agent Issues page.
- Windows Management Instrumentation (WMI)

To perform an unattended installation:

1. Launch an elevated command prompt with full administrator privileges. See Start a Command Prompt as an Administrator. Logging on to Windows as a member of the Administrators group does not grant sufficient permissions to run the installer.
2. Run the agent MSI installer package from the elevated command prompt. Use the `AD_SetupFile` parameter to pass the absolute file path to the setup configuration file.

   ```
   msiexec /i dotNetAgentSetup64.msi /q /norestart /lv %TEMP%\AgentInstaller.log AD_SetupFile=<absolute
   path to setup config.xml>
   ```

   You can set these optional parameters:

   `INSTALLDIR`: Specify the directory to install the .NET Agent executables and supporting files.

   `DOTNETAGENTFOLDER`: Specify the parent directory for local data including agent configuration files and log files.

   For example:

   ```
   msiexec /i dotNetAgentSetup64.msi /q /norestart /lv %TEMP%\AgentInstaller.log AD_SetupFile=C:
   \temp\SetupConfig.xml INSTALLDIR=D:\AppDynamics DOTNETAGENTFOLDER=D:\AppDynamicsData
   ```

If you specify the DOTNETAGENTFOLDER on the command line and a logFileDirectory in the setup configuration file, the agent creates config and data folders under DOTNETAGENTFOLDER but writes the log files to the logFileDirectory specified in the setup configuration file.

3. Start the AppDynamics.Agent.Coordinator process.

```
net start AppDynamics.Agent.Coordinator
```

4. Restart applications you have instrumented: IIS services, Windows services, and standalone applications.

   For example, to restart IIS:

```
iisreset
```

Setup Configuration File Properties

Winston Element

The Winston element is the root element of the configuration file.

Required element: <winston>

Log File Directory Element

The Log File Directory element is a child element of the Winston element. Use the directory attribute to specify the log directory. If you omit the logFileDirectory element, AppDynamics uses the default directory:

Windows Server 2008 and later: %ProgramData%\AppDynamics\DotNetAgent\Logs

Optional element: <logFileDirectory directory="C:\ProgramData\AppDynamics\DotNetAgent\Logs" />

Log File Folder Access Permissions Element

The Log File Folder Access Permissions is a child element of the Winston element. Unless you set the defaultAccountsEnabled attribute to false, AppDynamics grants write access to the Logs folder for the default accounts:

- LOCAL SERVICE
- SYSTEM
- NETWORK SERVICE
- ApplicationPool Identity

Optional element: <logFileFolderAccessPermissions defaultAccountsEnabled="false">

Account Element

The Account element is a child element of the Log File Folder Access Permissions element. Create an Account element for the Windows account you use to run your application.

Set the name attribute to the name of the account you use to run your application, that is the account for the application pool for IIS or the Windows service account.

The display name attribute is a user-friendly name you choose for the account. The display name shows up in log entries about assigning permissions for the account.

Optional element: <account name="MyAppPoolIdentity" displayName="Custom ApplicationPool Identity" />

For example, if you run a Windows service under a domain account:

<account name="MYDOMAIN\service_acct" displayName="Domain Service Account" />

AppDynamics Agent Element

The AppDynamics Agent element is a child of the Winston element. It follows the same format as the config.xml file to define the agent configuration for all your .NET applications. See .NET Agent Configuration Properties.

Required element: <appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">

Upgrade the .NET Agent for Windows

Related pages:

- Agent and Controller Compatibility
- Install the .NET Agent for Windows
- Configure the .NET Agent
- .NET Agent Directory Structure
- Troubleshoot .NET Agent Issues

The MSI installer package for AppDynamics .NET Agent installs updated agent files and maintains legacy configurations. When you upgrade from .NET Agent > 3.9, you must first uninstall the existing .NET Agent.

If you use the AppDynamics Azure Site Extension, see Install the AppDynamics Azure Site Extension.

Requirements

- Before you begin, review the Release Notes.
- AppDynamics requires an account access key for agent connections to single-tenant Controller accounts. AppDynamics <= 4.1 only required an account access key for multi-tenant Controller accounts.
  If you have a single-tenant Controller, you can locate the account access key in the Controller under **Settings** > **License** > **Account**. You must be a member of a role with the View License account level permission, see Create and Manage Custom Roles.

The .NET Agent only supports in-place upgrade for these versions:

- >= 3.9

Major and minor releases after the currently installed version. For patch releases, you must uninstall the existing agent before upgrading.

For example, to install the patch release 4.1.0.1 over 4.1.0.0, uninstall the existing agent before upgrading.

Upgrade the .NET Agent from >= 3.9

Except for patches to the current version, you do not need to uninstall the old agent first when you upgrade from the .NET Agent >= 3.9.

1. Stop `w3wp` processes for instrumented IIS applications. Stop instrumented Windows services or standalone applications.
2. Download the MSI installer package from AppDynamics Downloads.
3. Launch an elevated command prompt with full administrator privileges. Logging on to Windows as a member of the Administrators group does not grant sufficient permissions to run the installer. See Start a Command Prompt as an Administrator.
4. If you use a single-tenant Controller account, in the Controller click **Settings > License > Account** to view your access key.
5. Run a command-line agent install. For single-tenant Controller accounts, specify your account access key using the `AD_CONTROLLER_ACCOUNT_ACCESS_KEY` parameter. For example:

   ```
   msiexec /i "%USERPROFILE%\Downloads\dotNetAgentSetup.msi" /l log.txt /q
   AD_CONTROLLER_ACCOUNT_ACCESS_KEY=changeme
   ```

   The installation runs silently in the background.

   > If you forget to add the account access key for a single-tenant Controller account, you can run the Agent Configuration Utility or manually add it to the config.xml later, see Account Element.

6. Restart IIS:
   - Launch the AppDynamics Agent Configuration utility and click **Restart IIS** on the Configuration Summary window.
     or
   - Execute `iisreset` from the command line.

   Restart Windows services and standalone applications.

Upgrade the .NET Agent from 3.7.8 through 3.8.6

Uninstall the Old Agent Version

1. Stop IIS, instrumented Windows services, and instrumented standalone applications.

   > If you shut down IIS but continue to see active IIS Worker Processes, check the Application Pools pane in the IIS Manager and stop any started application pools.
   >
   > Failing to stop instrumented applications before uninstalling the .NET Agent may require you to reboot the machine.

2. Stop the `AppDynamics.Agent.Coordinator` service.
3. In the Control Panel, select Add/Remove Programs. Remove the `AppDynamics .NET Agent`.

> In some cases, another process interferes with the .NET Agent uninstallation process by locking the profiler.dll. If uninstallation fails, use a utility such as Process Explorer to see if a process is using profiler.dll. If so, terminate the process. Otherwise, try rebooting the machine. Then retry the uninstallation.

Install the New Agent Version

1. Stop `w3wp` processes for instrumented IIS applications. Stop instrumented Windows services or standalone applications.
2. Download the MSI installer package from the AppDynamics Downloads.
3. Launch an elevated command prompt with full administrator privileges. Logging on to Windows as a member of the Administrators group does not grant sufficient permissions to run the installer. See Start a Command Prompt as an Administrator.
4. If you use a single-tenant Controller account, in the Controller click **Settings > License > Account** to view your access key.
5. Run a command-line agent install. For single-tenant Controller accounts, specify your account access key using the `AD_CONTROLLER_ACCOUNT_ACCESS_KEY` parameter. For example:

```
msiexec /i "%USERPROFILE%\Downloads\dotNetAgentSetup.msi" /l log.txt /q
AD_CONTROLLER_ACCOUNT_ACCESS_KEY=changeme
```

   The installation runs silently in the background.

> If you forget to add the account access key for a single-tenant Controller account, you can run the Agent Configuration Utility or manually add it to the config.xml later, see Account Element.

6. Restart IIS:
   - Launch the AppDynamics Agent Configuration utility and click **Restart IIS** on the Configuration Summary window.
     or
   - Execute `iisreset` from the command line.

   Restart Windows services and standalone applications.

Upgrade the .NET Agent from <= 3.7.7

Identify the right upgrade path based upon the method of tier naming and assignment (manual or automatic) and the type of application you instrument:

- If you use manual tier naming and assignment, the installer package upgrades configurations for IIS applications and Windows services.
- If you used automatic tier naming and assignment, run the configuration utility to update configurations.
- If you used standalone applications with 3.7.7 or earlier, follow the steps for standalone applications on Configure the .NET Agent for Windows Services and Standalone Applications.

After installation, you may need to run the configuration utility to update your configuration and optionally remove legacy configurations.

Uninstall the Old.NET Agent Version

1. Stop IIS and instrumented Windows services.

> If you shut down IIS but continue to see active IIS Worker Processes, check the Application Pools pane in the IIS Manager and stop any started application pools.
>
> Failing to stop instrumented applications before uninstalling the .NET Agent may require you to reboot the machine.

2. Stop the `AppDynamics.Agent.Coordinator` service.
3. In the Control Panel, select Add/Remove Programs. Remove the `AppDynamics .NET Agent`.

> In some cases, another process interferes with the .NET Agent uninstallation process by locking the profiler.dll. If uninstallation fails, use a utility such as Process Explorer to see if a process is using profiler.dll. If so, terminate the process. Otherwise, try rebooting the machine. Then retry the uninstallation.

Install the New .NET AgentVersion

1. Stop `w3wp` processes for instrumented IIS applications. Stop instrumented Windows services or standalone applications.
2. Download the MSI installer package from AppDynamics Downloads.
3. Launch an elevated command prompt with full administrator privileges. Logging on to Windows as a member of the Administrators group does not grant sufficient permissions to run the installer. See Start a Command Prompt as an Administrator.
4. Optional, if you use a single-tenant Controller account, in the Controller click **Settings > License > Account** to view your access key.

5. Run a command-line agent install. For single-tenant Controller accounts, specify your account access key using the `AD_CONTROLLER_ACCOUNT_ACCESS_KEY` parameter. For example:

```
msiexec /i "%USERPROFILE%\Downloads\dotNetAgentSetup.msi" /l log.txt /q
AD_CONTROLLER_ACCOUNT_ACCESS_KEY=changeme
```

   The installation runs silently in the background.

   > If you forget to add the account access key for a single-tenant Controller account, you can run the Agent Configuration Utility or manually add it to the config.xml later, see Account Element.

6. Restart IIS:
   - Launch the AppDynamics Agent Configuration utility and click **Restart IIS** on the Configuration Summary window.
   - Execute `iisreset` from the command line.

   Restart Windows services.

If you used the following environment variables with the earlier version, the MSI installer migrates the configurations to the new configuration file:

- `AppDynamicsAgent_CallGraphOptions`
- `AppDynamicsAgent_DisableAppPools`
- `AppDynamicsAgent_EnableInProcesses`
- `AppDynamicsAgent_IgnoreCLREnv`
- `AppDynamicsAgent_Profiler_Classes`

Configure the .NET Agent

Configure the agent based on your method of tier generation and assignment: automatic or manual.

> The .NET Agent configuration utility only supports the configuration of one Controller per server. To configure multiple business applications, see Configure Multiple Business Application Support for .NET

Configure the Agent Using Automatic Tier Generation and Assignment

If you used automatic configuration with the earlier version of the .NET Agent, run the configuration utility to configure the agent:

1. Use the .NET Agent Configuration utility to reconfigure instrumentation for IIS applications. Choose **Automatic** for the method of tier generation and assignment. See Configure the .NET Agent.
2. Configure instrumentation for Windows services manually. See Configure the .NET Agent for Windows Services and Standalone Applications.

Configure the Agent Using Manual Tier Generation and Assignment

For agents using manual tier generation and assignment, the installer package migrates the configurations for IIS applications and for Windows services to the `config.xml`. At this stage, the configuration for IIS applications and Windows services is complete.

Clean Up Legacy Configurations

You can clean up legacy configurations by launching the AppDynamics Agent Configuration utility. When the utility detects agent settings from a previous version, it offers you the option to clean up.

The cleanup procedure modifies the `web.config` files causing an IIS restart.

1. Launch the AppDynamics Agent Configuration utility.
2. Select **Yes** to clean up old AppDynamics configurations.
3. Proceed through the wizard.
   - Verify or update the log directory, and grant write permissions to it.
   - Verify the controller connection information.
   - Verify or update manual tier assignment.

The utility removes these configurations:

- AppDynamics configurations from `web.config` files for IIS applications and `application.config` files for Windows services.
  Environment variables:
  - `AppDynamicsAgent_IgnoreCLREnv`
  - `AppDynamicsAgent_CallGraphOptions`
  - `AppDynamicsAgent_EnableInProcesses`
  - `AppDynamicsAgent_DisableAppPools`
  - `AppDynamicsAgent_Profiler_Classes`

Resume Monitoring

Start IIS and instrumented Windows services.

Troubleshoot .NET Agent Issues

Related pages:

- Install the .NET Agent for Windows
- Configure the .NET Agent
- Instrument SharePoint
- .NET Instrumentation Topics on the Community Knowledge Base

This page describes issues you may encounter when installing the .NET Agent.

Verify Agent Controller Communication

Use the AppDynamics UI to verify that the agent can connect to the Controller.

1. In a browser open:

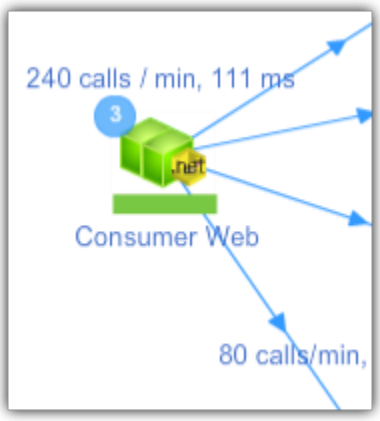
   ```
   http://<controller-host>:<controller-port>/controller
   ```

   If you cannot connect to the Controller in Internet Explorer, see Check Internet Explorer proxy settings.
2. Log in to the AppDynamics UI.
3. Select the application to open the Application Dashboard.
4. In the left navigation panel, select **Servers > Tiers & Nodes** and open the Health tab.

The Health tab lists the tiers, their nodes, and App Agent Status. When an agent successfully reports to the Controller, an *up* arrow symbol displays.

See Agent-to-Controller Connections.

- When deploying multiple agents for the same tier, determine if the correct number of nodes reports into the same tier.
- After sending a request to your web application, data should display on the AppDynamics UI. The agents should display in the Application Flow Map of the Application Dashboard.



If no data displays after a few minutes:

- Verify that the Agent is writing its log files.
  Windows Server 2008 and later: `%ProgramData%\AppDynamics\DotNetAgent\Logs\AgentLog.txt`
- If the log file exists, open it and review for errors.
- If the log file does not exist, run the Windows Event Viewer and review the application messages.
- If there are no AppDynamics event messages, look for messages from the .NET Runtime.

Check Internet Explorer Proxy Settings

> Use this procedure to resolve issues connecting to the AppDynamics Controller when step 1 of 'Verify Agent Controller Connection' fails. To configure the .NET Agent to work through a proxy, see 'Controller Element' on .NET Agent Configuration Properties.

Misconfigured proxy settings in Internet Explorer may cause the App Agent for .NET to fail to connect to the Controller.
If the test Controller connection fails on the Controller Configuration pane in the AppDynamics Agent Configuration utility:

1. Verify the Controller host and port settings are correct.
2. In Internet Explorer, open:

```
http://<controller-host>:<controller-port>/controller
```

3. If the connection also fails in Internet Explorer, check the proxy settings. See Change IE Proxy Settings.
4. Correct or remove any incorrect proxy settings.

Checklist for Resolving .NET Agent Installation Issues

	Item	Notes
✓	Run the installer as Administrator.	Verify Administrative privileges
✓	Verify that MSDTC is enabled and that it is running under the correct account.	Verify MSDTC is enabled and running under the correct account
✓	Verify permissions for Agent directory.	Verify that the .NET Agent directory has the correct permissions based on the site's application pool identity.
✓	Verify that the Agent is compatible with the Controller.	Agent and Controller Compatibility
✓	Verify the correct settings in the `config.xml`: Windows Server 2008 and later: `%ProgramData%\AppDynamics\DotNetAgent\Config\config.xml`	Update the config.xml file to include the .NET Agent Configuration Properties.

Resolve .NET Agent Installation Issues

If the Agent installation is failing, check these configurations in your environment:

Verify Administrative Privileges

Ensure that you have the administrative privileges when you launch the installer. If the current user doesn't have sufficient privileges, the installer prompts you for an administrator password.

Verify MSDTC is Enabled and Running for the Correct Account

If you encounter an error that MSDTC is not enabled or it is running for the wrong account, launch an elevated command prompt with full administrative privileges and execute the this command:

```
msdtc -install
```

Even if MSTDC is already installed, this command resets the service to run using the "NT Authority\NetworkServices" account.

Generate a Log for Agent Installation Failures

If the installer fails, use this command line utility to launch the installer:

```
msiexec /i $Path_to_the_MSI_File /l*v verbose.log
```

A verbose log for the .NET Agent is created at the same location where you saved the installer file.

Correct Failed Installation Caused by other APM Products

The .NET Agent installation may fail if there are other Application Performance Management (APM) products installed in the same managed environment. Remove the associated *Environment* subkey for certain services for the installed APM products.

To remove the associated environment subkey for W3SVC and WAS services in the registry:

1. Run Regedit or regedt32.

2. In regedit.exe, locate these registry keys:
   `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\W3SVC`
   `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\WAS`
3. Expand the keys.
4. Modify the `Environment` subkey to delete these values:

```
COMPLUS_ProfAPI_ProfilerCompatibilitySetting=EnableV2Profiler
    COR_ENABLE_PROFILING=1
    COR_PROFILER= {a GUID}
```

5. Restart the services. See How to restart the W3SVC and WAS services?

Resolve Configuration Errors

- Set your IIS applications to a Full trust level. .NET Agent reporting will not work if an IIS application is set to anything other than a Full trust level.
- Ensure that you have correctly configured the `config.xml` file for the App Agent for .NET. See .NET Agent Configuration Properties.
- If you manually edited the `config.xml` file, check the `AgentLog.txt` file and `WarnLog.txt` file for errors. Invalid XML displays in the log as:

```
2014-03-13 10:49:18.7199 1232 dllhost 1 1 Error ConfigurationManager Error reading the configuration file
```

- Resolve Log Issues

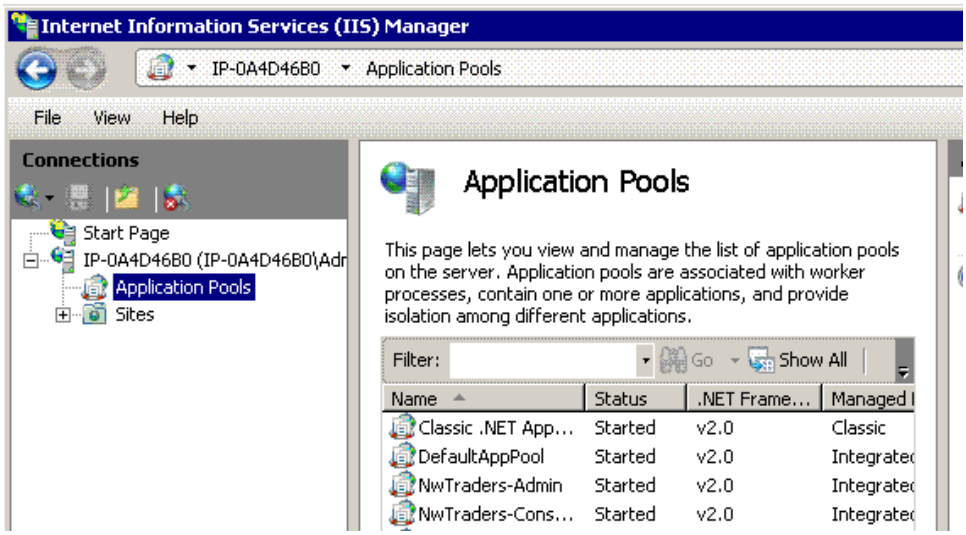
The .NET Agent writes logs to these directories:

`%ProgramData%\AppDynamics\DotNetAgent\Logs`

The agent will not generate logs if the agent directory does not have sufficient permissions.

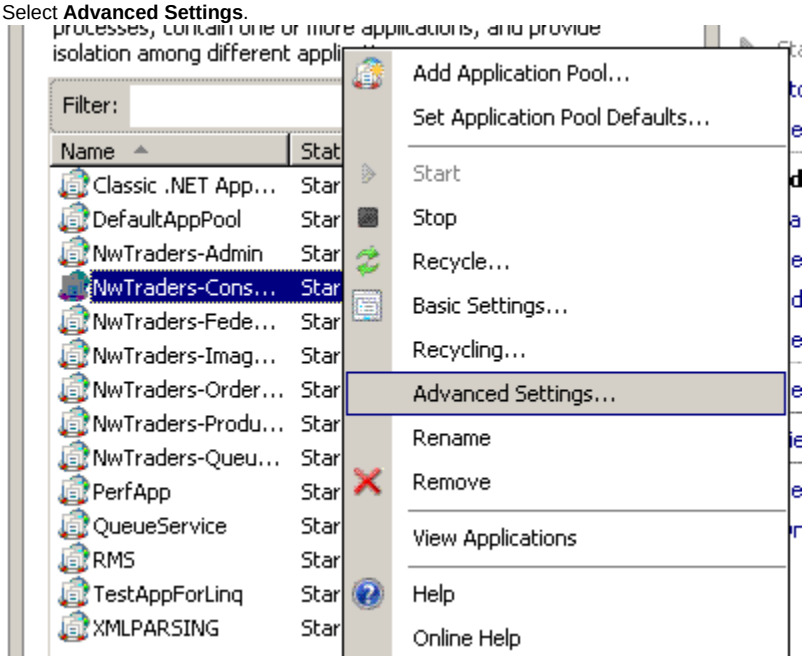
Verify that the .NET Agent Directory has the Correct Permissions

1. Select **IIS > Application pools**.
   IIS displays the list of application pools for your machine.

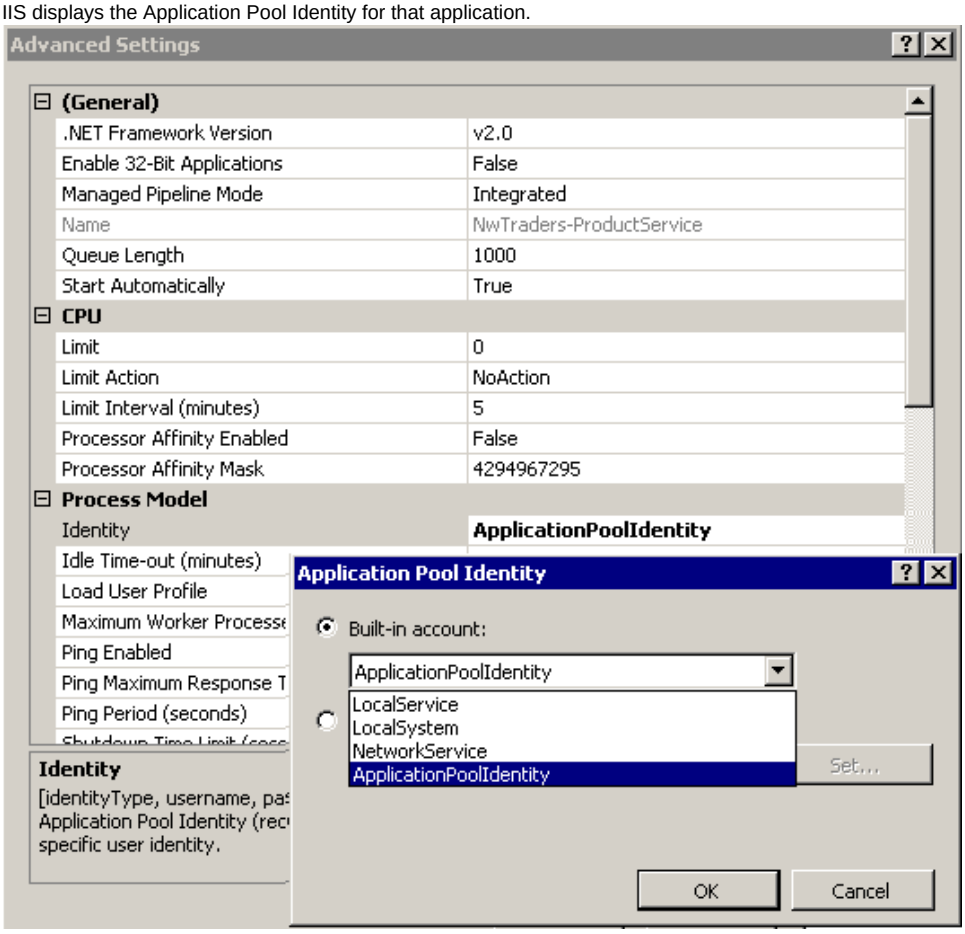


2. Right-click a particular application pool.

3. Select **Advanced Settings**.

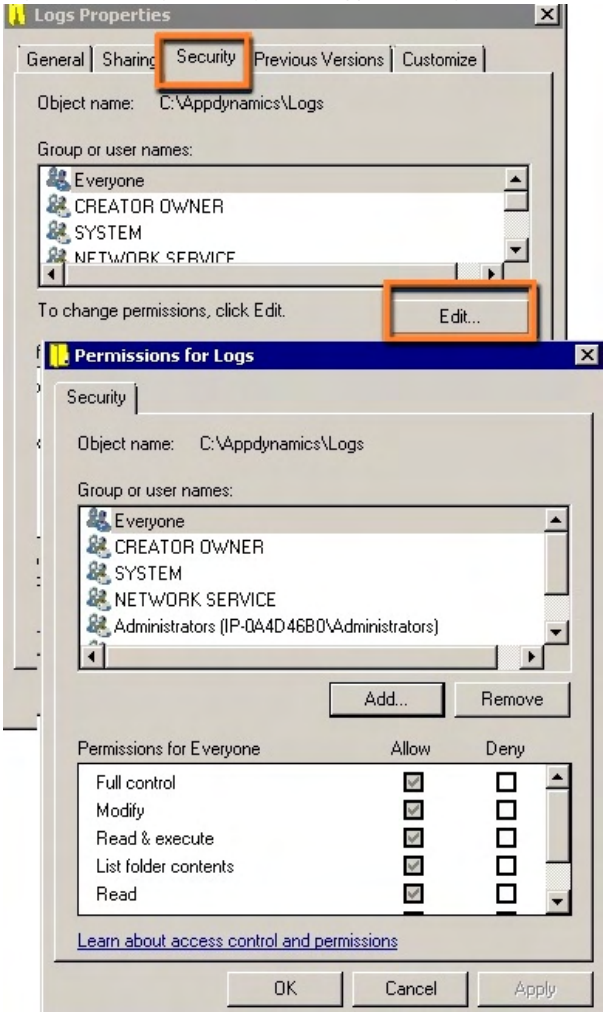


IIS displays the Application Pool Identity for that application.

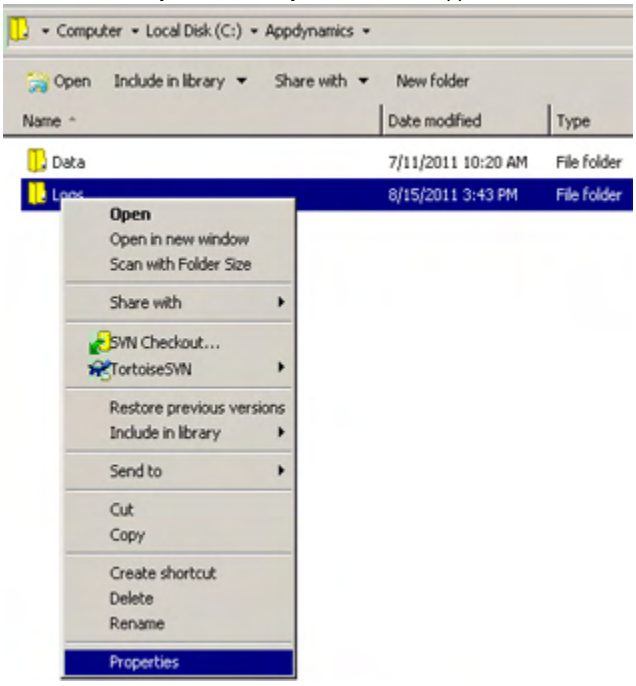


4. Ensure that your Agent Directory also has the same permissions as your site application pools.
    - Navigate to AppDynamics .NET App Server Agent directory location.

- Right-click the `logs` directory for the App Server Agent and select **Properties**.

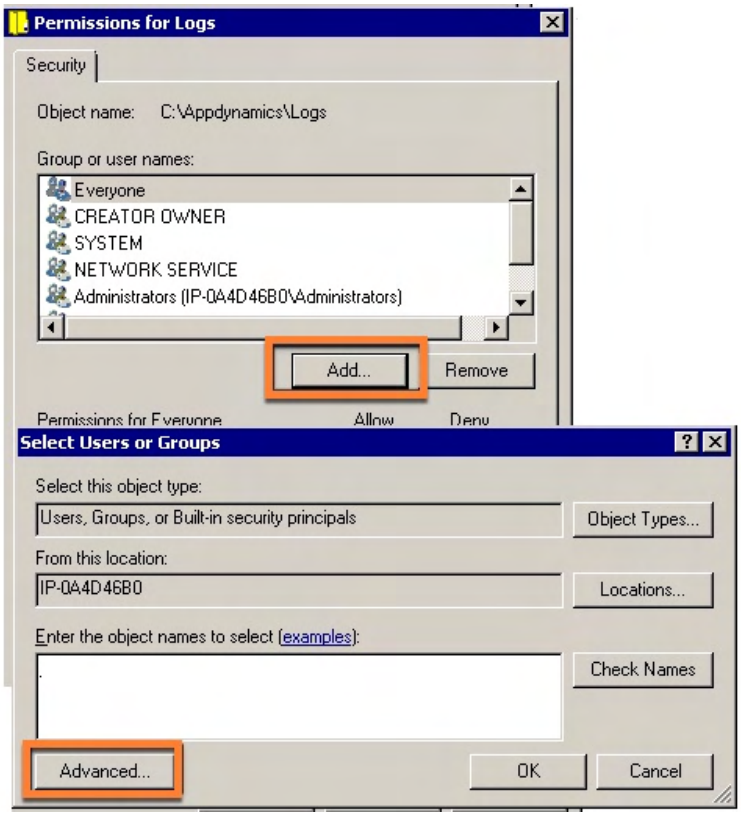


- Select the Security tab, and verify that the same Application Pool Identity is specified for the .NET Agent directory.



If the Agent logs directory does not have the required permissions:

1. In the Security tab, select **Edit**.
2. Select **Add** to add new permissions to the Agent directory.
3. Select **Advanced**.



4. Click **Find Now** to find all the users, groups, or built-in security principals on your machine.
5. Select the required group from the list and click **OK**. Review the information that follows for Allowed groups.
6. Provide the read and write permissions for the selected user/group/security principal to the Agent directory and click **OK**.
7. Click **Apply**.

Allowed Groups for Different IIS Versions

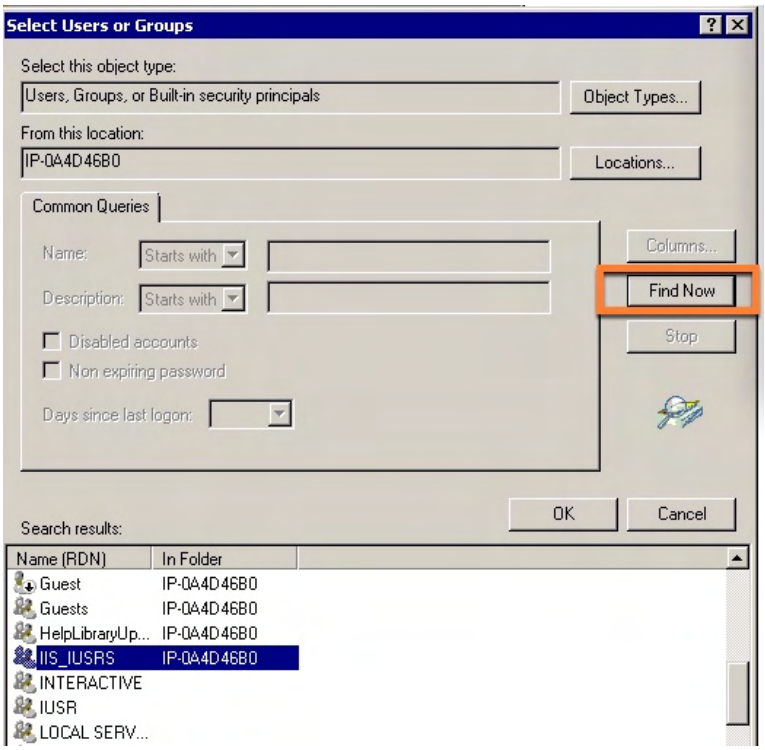
For IIS 6.x, these settings are applicable for Application Pool Identities:

Application Pool Identity	Permission Level
`LocalService`	`LOCAL SERVICE`
`LocalSystem`	`SYSTEM`
`NetworkService`	`NETWORK SERVICE`
`Custom Account`	Provide the exact name of the account.

For IIS >= IIS 7.0, these settings are applicable for Application Pool Identities:

Application Pool Identity	Permission Level
`LocalService`	`LOCAL SERVICE`
`LocalSystem`	`SYSTEM`
`NetworkService`	`NETWORK SERVICE`
`ApplicationPoolIdentity`	Provide the group level permissions for `IIS_IUSRS` Group See the screenshot that follows
`Custom Account`	Provide the exact name of the account.

For example, if your application has the identity `ApplicationPoolIdentity`, you must provide the permissions for `IIS_IUSRS` group to your agent directory.



Another Version of AppDynamics Detected

AppDynamics recommends that you contact support to resolve this issue.

.NET Agent >= 20.3 supports .NET Core. If you used the Microservice Agent to monitor .NET Core applications you can monitor .NET Core with the full .NET Agent. You may receive an error message in the MSI Installer stating that another version of AppDynamics is detected.

The Microservice Agent uses a JSON file to store configuration information while the .NET Agent uses an XML file. The data does not migrate between these files.

To fix this issue, you need to map the data in the JSON file to the XML file, `config.xml` and remove environment variables. If you have completed the instructions above and still see the error, you may need to reboot your machine. You may want to plan reboots according to your planned outage schedule.

Known Issues

You may ignore the following error from the `AppDynamics.Agent.Coordinator` service in the agent.log or warn.log:

```
2015-12-01 10:12:03.0435 3872 AppDynamics.Coordinator 1 7 Warn MaxQueueItemAgePolicy MaxQueueItemAge is beyond
the integer limit: 13093467123
```

Instrument SharePoint

Business operations must ensure their internal and external teams have full access to the content and collaborative services that SharePoint offers. To do that, administrators need an APM solution with total visibility across the SharePoint environment. AppDynamics works with SharePoint to provide details for complex tasks such as end-user monitoring, log analytics, and server support.

By using AppDynamics to monitor SharePoint, you can:

- Monitor most SharePoint components (web applications, search components, workflow and Office Web Apps), using a mixture of .NET APM agents, EUM agents, analytics agents, and machine agents.
- Understand what is occurring on the end-user page, and document navigation and browsing using End User Monitoring.
- View APM partitions of user activity by content web applications.
- Gain insights into Office Web Apps document rendering behavior.
- View more details on search and query usage.

> AppDynamics is not a replacement for the Systems Center Operations Management (SCOM) pack or a well-planned IBM Tivoli implementation, particularly when just interpreting PerfMon counters. SharePoint Server 2013 integrates with SCOM, which is very useful but still optional. AppDynamics is not meant to be a replacement for native monitoring tools but rather work together with tools from Microsoft and its robust partner ecosystem.

Before implementing this solution, download and read this eBook, and work with an AppDynamics expert in your organization, or AppDynamics Professional Services.

Uninstall the .NET Agent

This page describes how to uninstall the .NET Agent. This procedure frees the license associated with the agent and makes it available for another app agent to use.

> Do not follow these instructions if you are doing an upgrade. To upgrade to a new version, see Upgrade the .NET Agent for Windows.

To uninstall the .NET Agent completely:

1. Stop IIS, instrumented Windows services, and instrumented standalone applications.

   > Failing to stop instrumented applications before uninstalling the agent requires you to reboot the machine to complete the uninstall.

2. Stop the `AppDynamics.Agent.Coordinator` service.
3. From the Control Panel, select **Add/Remove Programs**, and remove the **AppDynamics .NET Agent**.
4. The uninstall procedure does not remove configuration files. You must delete the configuration directory: `%ProgramData%\AppDynamics`
5. Restart IIS, Windows services, and standalone applications.

> In some cases, another process interferes with the .NET Agent uninstallation process by locking the `profiler.dll`. If uninstallation fails, use a utility such as Process Explorer to see determine if a process is using `profiler.dll`. If so, then terminate the process; otherwise, reboot the machine. Then, retry the uninstallation.

Administer the .NET Agent

Related pages:

- .NET Agent Configuration Properties
- Manage Configuration for .NET

The .NET Agent uses a single configuration file to control agent behavior and .NET Machine Agent behavior. The configuration file specifies Controller connectivity, .NET Machine Agent operations, and app agent functionality for IIS applications, Windows services, and standalone applications. The single configuration file enables you to:

- Maintain agent configurations separately from `web.config` files
- Instrument Windows services and standalone applications without environment variables
- Control agent behavior for specific applications with hierarchical configuration

You use the AppDynamics Controller to upload and manage .NET Agent configuration files. From the Controller, you can deploy a configuration to a machine or multiple machines where the agent is installed. See Manage Configuration for .NET.

Configure Agent Properties

You configure the agent properties in the `config.xml` file in the agent `Config` directory. To edit the `config.xml`, you must launch the editor as an administrator. When you run the .NET Agent Configuration Utility, it writes the `config.xml` file to one of these locations:

- Windows Server 2008 and later
  `%ProgramData%\AppDynamics\DotNetAgent\Config\config.xml`
- Windows Azure
  For Windows Azure deployments, the .NET Agent NuGet package contains the `config.xml` file.
  Sample config.xml files install to this location:
  `%ProgramFiles%\AppDynamics\AppDynamics .NET Agent\SampleConfigurations`

> After you edit the `config.xml` file, you must restart the `AppDynamics.Agent.Coordinator` service. Then restart your IIS services, Windows services, or standalone applications for your instrumentation changes to take effect.

Customize .NET Agent Behavior in `config.xml`

Some .NET Agent configurations require that you edit the `config.xml`:

1. Shut down the `AppDynamics.Agent.Coordinator` service.
2. Edit the `config.xml` file as an administrator.
3. Modify the XML file according to the configuration instructions.
4. Save the `config.xml` file.
5. Start the `AppDynamics.Agent.Coordinator` service.
6. In some cases, you may need to restart IIS, instrumented Windows services, or instrumented Standalone applications. See individual .NET Agent administration topics.

Example Minimal `config.xml`

The most basic configuration shows the required sections for agent configuration. This example instruments all IIS applications using the automatic element, `<automatic />`. For this example, AppDynamics does not instrument Windows services or standalone applications.

```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001
/XMLSchema">
  <controller host="mycontroller.mycompany.com" port="8090" ssl=false">
    <application name="MyDotNetApplication" />
        <account name="customer1" password="changeme" />
  </controller>
  <machine-agent />
  <app-agents>

    <IIS>
      <automatic />
    </IIS>

  </app-agents>
</appdynamics-agent>
```

Agent Log Files

The configuration file that controls log files for the .NET Agent is located here:

```
%ProgramFiles%\AppDynamics\AppDynamics .NET Agent\AppDynamicsAgentLog.config
```

The configuration file uses NLog configuration format.

.NET Agent Directory Structure

Related pages:

- Install the .NET Agent for Windows
- Administer the .NET Agent

This page describes the .NET agent directory structure for Windows Server 2008 and later.

Executables

The .NET Agent executables and supporting files install to the `AppDynamics .NET Agent` directory:

```
%ProgramFiles%\AppDynamics\AppDynamics .NET Agent
```

Log files

The `Logs` directory defaults to this location:

```
%ProgramData%\AppDynamics\DotNetAgent\Logs
```

Agent Configuration Files

The .NET Agent writes the `config.xml` configuration file to the `Config` directory:

```
%ProgramData%\AppDynamics\DotNetAgent\Config
```

If you use Config Management, the .NET Agent stores backup configuration files in the `Config\Backup` directory.

Coordinator Service Configuration Files

The AppDynamics Agent Coordinator service writes configuration files from the Controller to the `Data` directory:

```
%ProgramData%\AppDynamics\DotNetAgent\Data
```

About Windows Server System Directory Variables

`%ProgramData%` is located at `<system drive>\Program Data`.

.NET Agent Configuration Properties

Related pages:

- Administer the .NET Agent
- Configure the .NET Agent
- Name .NET Tiers
- Configure the .NET Agent for Windows Services and Standalone Applications
- .NET Agent for Linux Environment Variables

This page is describes the .NET Agent configuration properties that you set in the agent `config.xml` file. You can set some of these properties using Windows System Environment Variables. See .NET Agent Environment Variables. To edit the file and apply your changes, see Administer the .NET Agent.

> **Environment Variables Reference**
>
> You can use environment variables in the agent config.xml file for node, tier, and application names. To include an environment variable, reference it by specifying `%<variable>%`, which you can combine with other characters; for example, `Web-%COMPUTERNAME%` translates to `Web-HOST23`.

AppDynamics Agent Element

The `appdynamics-agent` element is the root container element for configurations in the config.xml.

Required Element:

```
<appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001
/XMLSchema">
```

Controller Element

The `controller` element is a child of the `appdynamics-agent` element. It specifies the connection information for the AppDynamics Controller.

The .NET Agent configuration utility only supports the configuration of one Controller per server.

Required Element:

```
<controller host="mycontroller.mycompany.com" port="8090" ssl="false" enable_tls12="false" high_availability="
false" secure="false" enable_config_deployment="true"
```

Controller host attribute

The `controller host` attribute indicates the hostname or the IP address of the AppDynamics Controller. For an on-premises Controller, use the value for Application Server Host Name you provided when you installed the Controller. If you use the AppDynamics SaaS Controller, see the Welcome email from AppDynamics.

Type: String
Default: None
Required: Yes

Environment Variable: APPDYNAMICS.CONTROLLER.HOSTNAME

Controller port attribute

The `controller port` attribute specifies the HTTP or HTTPS port of the AppDynamics Controller. If the `controller` ssl attribute is set to `true`, specify the HTTPS port of the Controller; otherwise, specify the HTTP port.

Type: Positive Integer
Default: 8090

- For On-premises installations, the defaults are port 8090 for HTTP and port 8181 for HTTPS.
- For the SaaS Controller, use port 80 for HTTP or port 443 for HTTPS.

Required: Yes

Environment Variable: APPDYNAMICS.CONTROLLER.PORT

Controller ssl attribute

To enable encryption over SSL between the agent and the Controller, set the controller `ssl` attribute to true.

Type: Boolean
Default: false
Required: No

Environment Variable: APPDYNAMICS.CONTROLLER.SSL.ENABLED

Controller enable TLS 1.2 attribute

When you enable SSL, the agent secures communication to the Controller using the protocols set for `ServicePointManager.SecurityProtocol` in your application. Set the controller `enable TLS 1.2` attribute to `true` to add TLS 1.2 as the first option in the list of protocols. This affects all secure communications from your application, not just requests to the AppDynamics Controller.

Type: Boolean
Default: false
Required: No

Controller high availability attribute

If you have your Controller setup for high availability, set the controller `high availability` attribute to `true`.

Type: Boolean
Default: false
Required: No

Controller secure attribute

If you set up the .NET Agent to encrypt credentials, the MSI installer package automatically sets the `controller secure` attribute to `true`. When `controller secure` is `true`, the .NET Agent ignores any credentials in the Account element or the Proxy element and uses the credentials from the Windows Credential Store. See Encrypt Credentials in .NET Agent Configuration

Type**:** Boolean
Default**:** false
Required: No

Controller enable config deployment attribute

By default, the .NET Agent checks for configuration updates to the `config.xml` from the Controller. If you want to disable the agent from `config.xml` configuration files from the Controller set the `controller enable config deployment` attribute to `false`. See Manage Configuration for .NET

Type: Boolean
Default: true
Required: No

Controller Application Element

The `controller application` element is a child of the `controller` element. It indicates the name of the logical business application you see in the Controller interface.

The .NET Agent configuration utility only supports one business application per server. Use one of the following methods to support multiple applications:

- Use tiers to organize different applications you instrument on a single server.
- Or manually configure support for multiple business applications, see Configure Multiple Business Application Support for .NET
  Multiple application support requires different configuration elements, including the `<applications>` container element.

Required Element: `<application name="MyDotNetApplication"/>`

Application name attribute

Set the `application name` attribute to the business application you use in the Controller. If the application name does not exist, the Controller will create it when the agent registers. All instrumented applications in the `config.xml` register with the same business application in the Controller. See Overview of Application Monitoring

Type: String, may also reference an environment variable. See Reference of Environment Variables
Default: None
Required: Yes

Environment Variable: APPDYNAMICS.AGENT.APPLICATIONNAME

You specify a Tier for individual applications in the App Agents Element

Account Element

The `account` element is a child of the `controller` element. If the AppDynamics Controller runs in multi-tenant mode or if you use the AppDynamics SaaS Controller, specify the account name. For single-tenant accounts the default account name is `customer1`. If you are using the AppDynamics SaaS Controller, the account name is provided in the Welcome email from AppDynamics.

The agent requires to use your account access key as a password to authenticate with the Controller. For single-tenant accounts, you can view the access key in the Controller under Settings > License. If you are using the AppDynamics SaaS Controller, the account access key is provided in the Welcome email from AppDynamics.

Required Element: `<account name="mycompany" password="myaccesskey"/>`

Account name attribute

The `account name` attribute indicates the account name for the SaaS or multi-tenant Controller.

Type: String
Default: For single-tenant Controllers, the agent assumes the default of `customer1` if you do not specify an account name.
Required: Only for SaaS or multi-tenant Controllers.

Environment Variable: APPDYNAMICS.AGENT.ACCOUNTNAME

Account password attribute

The `account password` attribute indicates the account access key for the Controller.

Type: String
Default: None
Required: Yes

Environment Variable: APPDYNAMICS.AGENT.ACCOUNTACCESSKEY

Proxy Element

The `proxy` element is a child of the `controller` element. Use it to configure the connection to the Controller through a proxy server with no authentication.

Optional Element: `<proxy host="proxy-name" port="3128" enabled="true"/>`

Proxy host attribute

The `proxy host` attribute indicates the proxy server hostname or IP address.

Type: String
Default: None
Required: host is required for the proxy element

Proxy port attribute

The `proxy port` attribute indicates the proxy server port.

Type: Positive Integer
Default: None
Required: port is required for the proxy element

Proxy enabled attribute

To enable Controller access through a proxy server, set the `proxy enabled` attribute to `true`.

Type: Boolean
Default: true
Required**:** No

Proxy - Authentication Element

The `authentication` element is a child of the `proxy` element. Use the `authentication` element to enable proxy authentication and to supply the credentials for your proxy server. For environments where security policies require you to secure credentials stored on disk, you can encrypt the credentials and store them in the Windows Credential Manager.

Optional Element: `<authentication enabled="true" user_name="my_proxy_user" password="password" domain="my_windows_domain"/>`

Proxy authentication enabled attribute

Set the `proxy authentication enabled` attribute to `true` to configure the agent to send credentials to a proxy server.

Type: Boolean
Default: false
Required: No

Proxy authentication username attribute

The `proxy authentication username` attribute indicates the name of the proxy user.

Type: String
Default: None
Required: The username attribute is required for the Authentication element

Proxy authentication password attribute

The `proxy authentication password` attribute indicates the password for the proxy user.

Type: String
Default: None
Required: No

Proxy authentication domain attribute

The `proxy authentication domain` attribute specifies the domain or realm where the username is located. This could be the host computer name, an Active Directory domain or DNS domain.

Type: String
Default: None
Required: No

Sample Controller Configuration with Proxy Authentication enabled

In this example, the agent accesses the controller through the proxy, `myproxy.example.com` using proxy authentication. The .NET Agent uses the credentials `MyProxyUser@mydomain.com` with password `password` to access the proxy server.

```
<controller host="mycontroller.example.com">
  <application name="MyDotNetApplicaition" />
  <proxy host="myproxy.example.com" port="3128" enabled="true">
     <authentication enabled="true" user_name="MyProxyUser" password="password" domain="mydomain.com"/>
  </proxy>
</controller>
```

Machine Agent Element

The `machine-agent` element is a child of the AppDynamics Agent element. An empty `machine-agent` element enables the default instrumentation for the .NET machine agent. See Monitor CLRs and Monitor IIS

Enable optional additional Microsoft Performance Counters or .NET Agent instrumentors as children of the Machine Agent element.

Required Element: `<machine-agent/>`

Machine Snapshot Element

The `machine-snapshot` element is a child of the `machine-agent` element. Use it to tune the settings for machine snapshots in your environment. If you do not specify an attribute, the agent uses the default values for that attribute. See Machine Snapshots for .NET and Configure Machine Snapshots for .NET

Optional Element: `<machine-snapshot enabled="true" window-size="600" samples-per-window="60" violations-per-window="6" max-percent-cpu="80" max-percent-memory="80" max-queue-item-age="100" periodic-collection="600" />`

Machine Snapshot enabled attribute

Set the `machine-snapshot enabled` attribute to `false` to disable machine snapshots.

Type: Boolean
Default: true
Required: No

Machine Snapshot window size attribute

Specify the window size time range in seconds for the .NET Machine Agent to take samples. During a window, the agent takes samples and checks them for breached thresholds: `max percent CPU`, `max percent memory`, and `max queue item age`.

Type: Integer
Default: 600
Required: No

Machine Snapshot samples per window attribute

Specify the number of samples the .NET Machine Agent takes during the specified window. For example, if the `window size` is 600 and the `samples per window` is 60, the agent takes a sample once every 10 seconds.

Type: Integer
Default: 60
Required: No

Machine Snapshot violations per window attribute

When the .NET Machine Agent detects the number of violations per window for one threshold, it takes a snapshot. The agent only takes one snapshot per window for thresholds exceeded. For example, if `violations per window` is set to 6, and six samples show memory usage at 80% or greater, the agent takes a snapshot. The counters for each threshold are separate.

Type: Integer
Default: 6
Required: No

Machine Snapshot max percent memory attribute

When the .NET Machine agent detects memory usage on the machine equals or exceeds the `max percent memory` value, it flags the sample as a violation. The minimum value is 20. The maximum value is 100.

Type: Integer
Default: 80
Required: No

Machine Snapshot max percent CPU attribute

When the .NET Machine agent detects CPU usage on the machine equals or exceeds the `max percent cpu` value, it flags the sample as a violation. The minimum value is 20. The maximum value is 100.

Type: Integer
Default: 80
Required: No

Machine Snapshot max queue item age attribute

When the .NET Machine agent detects the oldest item in the IIS queue equals or exceeds the `max queue item age` value in milliseconds, it flags the sample as a violation.

Type: Integer
Default: 100
Required: No

Machine Snapshot periodic collection attribute

The .NET Machine agent takes one snapshot per periodic collection time range. Specify the value in seconds. The minimum is 60 seconds.

Type: Integer
Default: 600
Required: No

CLR Crash Reporting Element

The `clrcrash-reporting` element is a child element of the `machine-agent` element. Use the `clrcrash-reporting` element to control whether or not the .NET Machine Agent reports CLR crash events to the Controller. See Monitor CLR Crashes

Optional Element: `<clrcrash-reporting enabled="true"/>`

CLR Crash Reporting enabled attribute

Set the `clrcrash-reporting enabled` attribute to `false` to stop reporting CLR Crash events to the controller.

Type: Boolean
Default: true
Required: No

Process Monitor Element

The `process-monitor` element is a child element of the `machine-agent` element. By default, the agent enables process monitoring for all IIS processes.

Optional Element: <process-monitor report-all-iis-processes="true"/>

Process Monitor report all IIS processes attribute

Set the `report-all-iis-processes` attribute to `false` to enable process monitoring only for instrumented IIS processes.

Type: Boolean
Default: true
Required: No

Metrics Element

The `metrics` element is a child element of the `machine-agent` element. By default, the machine agent registers a maximum of 200 metrics. Use the `metrics` element to increase the number of metrics the .NET Machine Agent can register. See Metrics Limits

Use caution when increasing the metric registration limits. Increasing the limit can increase the resource overhead for agents and Controller.

Optional Element: <metrics max-metrics="200"/>

Metrics max-metrics attribute

Specify the maximum number of metrics the .NET Machine Agent can register.

Type: Integer
Default: 200
Required: No

Perf-metrics Element

The `perf-metrics` element is a child element of the `metrics` element. By default, the machine agent collects and reports a full set of performance metrics. You can modify the priority level for the performance metrics to limit the metrics the agent collects and thereby decrease agent overhead. For a full list of metrics and their priorities, see Manage Windows Performance Metrics Optional Element: `<perf-metrics priority-level="3">`

Perf-metrics priority-level attribute

Specify the set of performance metrics for the machine agent to collect as follows:

- 0 - disable metric collection
- 1 - collect only high priority metrics
- 2 - collect high and medium priority metrics
- 3 - collect all metrics: high, medium, and low priority

Type: Integer in the range 0 - 3
Default: 3
Required: If you use the `perf-metrics` element.

Perf-metric Element

The `perf-metric` element is a child element of the `perf-metrics` element. If you set the `perf-metric` priority to `0` to disable general collection of performance metrics, you can use the `perf-metric` element to enable individual performance metrics. You cannot enable individual metrics with `perf-metric` priority greater than 0.

Optional Element: `<perf-metric name="<metric_path>"/>`

Perf-metric name attribute

Specify the full path of the performance metric. See the example below for the `% Busy` metric and the `Errors Unhandled During Execution` metric.

Type: String: metric path
Default: N/A
Required: If you use the `perf-metric` element.

Sample Machine Agent Configuration with Individual Performance Metrics

```
<machine-agent>
   <metrics>
        <!-- Disable collection of performance metrics in general. -->
     <perf-metrics priority-level="0">
       <!-- Enable collection of individual performance metrics. -->
       <perf-metric name="Hardware Resources|CPU|%Busy"/>
       <perf-metric name="ASP.NET Applications|Errors Unhandled During Execution"/>
     </perf-metrics>
   </metrics>
 </machine-agent>
```

Performance Counters Element

The `perf-counters` element is a child of the `machine-agent` element. It is a container for all performance counters.

Optional Element: `<perf-counters>`

Performance Counter element

The `perf-counter` element is a child of the `perf-counters` element. For a list of performance counters to enable, see Performance Counters in the .NET Framework

Optional Element: `<perf-counter cat="category" name="name" instance="instance"/>`

Performance Counter cat attribute

The `perf-counter cat` attribute indicates the performance counter category.

Type: String
Default: None
Required: Category is required for the `perf-counter` element.

Performance Counter name attribute

The `perf-counter name` attribute indicates the performance counter name.

Type: String
Default: None
Required: Name is required for the `perf-counter` element.

Performance Counter instance attribute

The `perf-counter instance` attribute is the performance counter instance value.

Type: String
Default: None
Required: Instance is required for the `perf-counter` element.

Sample Machine Agent Configuration with Performance Counters

```
<machine-agent>
  <!-- Additional machine level Performance Counters -->
  <perf-counters>
    <perf-counter cat="Network Interface" name="Bytes Sent" instance="Local Area Connection"/>
  </perf-counters>
</machine-agent>
```

Instrumentation Element

The `instrumentation` element is a child of the `machine-agent` element. It allows you to enable additional .NET Agent instrumentors such as thread correlation or correlation for .NET remoting.

Optional Element: `<instrumentation>`

Instrumentor Element

The `instrumentor` element is a child of the `instrumentation` element. The `instrumentor` element specifies the .NET Agent instrumentor to implement.

Optional Element: `<instrumentor name="instrumentor name" enabled="true"/>/>`

Instrumentor name attribute

The `instrumentor name` attribute indicates the instrumentor name.

Type: String
Default: None
Required: Name is required for the Instrumentor element.

Instrumentor enabled attribute

Set the `instrumentor enabled` attribute to `true` to enable instrumentation.

Type: Boolean
Default: false
Required: No.

The current configuration syntax is `enabled="true"`. Versions prior to 3.7.8 used disabled="false".

Sample Machine Agent Configuration with Thread Correlation Instrumentors

```
<machine-agent>
   <!--Enable thread correlation-->
   <instrumentation>
     <instrumentor name="ThreadCorrelationThreadPoolCLR2Instrumentor" enabled="true"/>
     <instrumentor name="ThreadStartCLR2Instrumentor" enabled="true"/>
     <instrumentor name="ThreadStartCLR4Instrumentor" enabled="true"/>
   </instrumentation>
 </machine-agent>
```

Additional .NET Agent Instrumentors

- See Thread Correlation for .NET
- See Enable Correlation for .NET Remoting
- See Enable Instrumentation for WCF Data Services

App Agents Element

The `app-agents` element is a child of the `appdynamics agent` element. It is a container for app agent configurations for IIS applications, Windows services, and standalone applications.

Required Element: `<app-agents enabled="true">`

App agents enabled attribute

To disable application monitoring on the server, set the `app-agents enabled` attribute to `false`.

Type: Boolean
Default: true
Required: No

Default Profiler Element

The default `profiler` element is a child of the `app-agents` element. It defines customizations to the default profiler behavior for all instrumented .NET applications on the machine: IIS applications, application pools, Windows services, and standalone applications.

Optional Element: `<profiler>`

Profiler - Disabled Features Element

The `disabled-features` element is a child of the `profiler` element. Use this property to disable data collection mechanisms at the agent level for security or privacy reasons. This configuration overrides any configuration set by the Controller.

Optional Element: <disabled-features value="NONE"/>

Disabled features value attribute

The `disabled-features value` attribute is a comma-separated list of features to disable. The available values are as follows:

- LOG_PAYLOAD: Override the log-request-payload node property to suppress logging HTTP request payloads
- RAW_SQL: Override the `capture-raw-sql` node property to suppress logging raw SQL output
- CUSTOM_EXIT_SNAP_DATA: Suppress snapshot data from custom exits points
- METHOD_INV_DATA_COLLECTOR: Suppress method invocation data collector user data
- HTTP_DATA_COLLECTOR: Suppress HTTP request data collector user data
- INFO_POINT: Suppress information point metrics
- ALL: Disable all the available features
- NONE: Don't disable features

Type: String
Default: NONE
Required: No

Profiler - Successful Exit Code Element

The `successful-exit-code` element is a child of the `profiler` element. The default successful exit code determines whether or not the agent flags a CLR restart event as graceful or not for Windows services or standalone applications. This configuration does not apply to IIS.

Optional Element: `<successful-exit-code value="0"/>`

Successful exit code value attribute

Type: Integer
Default: 0
Required: No

Profiler - Runtime Reinstrumentation Element

The `runtime-reinstrumentation` element is a child of the `profiler` element. Use this property to configure runtime reinstrumentation. See Configure Runtime Reinstrumentation for .NET

Optional Element: `<runtime-reinstrumentation enabled="true" interval="60000" optimize="true" />`

Runtime reinstrumentation enabled attribute

Set to `true` to enable runtime reinstrumentation.

Type: Boolean
Default: false
Required: No

Runtime reinstrumentation interval attribute

The frequency in milliseconds that the agent checks for configuration updates that initiate runtime reinstrumentation.

Type: Integer
Default: 60000 milliseconds
Minimum: Because runtime reinstrumentation adds a small amount of system overhead, AppDynamics recommends a minimum interval of 1 minute or 60000 milliseconds.
Required: No

Sample Default Profiler Configuration

```
<app-agents>
    <profiler>
      <disabled-features value="LOG_PAYLOAD,RAW_SQL,CUSTOM_EXIT_SNAP_DATA"/>
          <!-- Set the successful exit code for Windows services and standalone applications to "1." -->
      <successful-exit-code value="1"/>
      <!-- Enable Runtime reinstrumentation -->
      <runtime-reinstrumentation enabled="true" interval="60000" />
    </profiler>
...
</app-agents>
```

App Agents - IIS Element

The `IIS` element is a child of the `app-agents` element. There are three options to configure IIS applications:

- Automatic configuration
- Application pool configuration
- Application configuration

The settings for any application pool apply to all applications within the app pool unless the individual application has a specific configuration.

Explicit child-level configurations override parent-level configurations. Otherwise, children inherit parent configurations.

Optional Element: `<IIS>`

Exclude child applications attribute

By default, when you instrument an IIS application, the .NET Agent instruments any child applications and assigns them to the same tier as the parent. To prevent the agent from automatically instrumenting child applications, set `exclude-child-applications` to `true`. For instance: `<IIS exclude-child-applications="true">`

If you disable instrumentation for child applications in general, you can instrument-specific child applications using the IIS Application element.

Type: Boolean
Default: false
Required: No

IIS Automatic Instrumentation Element

The `automatic` element is a child of the `IIS` element. Use the `automatic` element to enable or disable automatic instrumentation for all IIS apps. You can configure automatic instrumentation and manual instrumentation both. Manual configurations override automatic ones.

Optional Element: `<automatic enabled="false" />`

Automatic enabled attribute

Set the automatic `enabled` attribute to `true` to enable instrumentation for all IIS applications. This is the default setting if you use the .NET Agent Configuration Utility automatic configuration option. To disable automatic instrumentation for all IIS applications, set the value to `false`.

Type: Boolean
Default: true
Required: No

IIS Application name enabled attribute

By default, the agent does not report an IIS application name. To view the IIS application name in the CLR metadata, set the `iis-application-name-enabled` attribute to `true`. For example, `<automatic iis-application-name-enabled="true" />`. When enabled, the name appears as the `iis-application-name` value in the CLR metadata.

Type: Boolean
Default: false
Required: No

IIS Application Pools Element

The `IIS application pools` element is a child of the `IIS` element. It is a container element for all the IIS application pools you configure for instrumentation.
Optional Element: `<application-pools>`

IIS Application Pool Element

The `application-pool` element is a child of the `application-pools` element. You may have multiple application pool elements distinguished by the name attribute. Use the application pool element to configure the app agent for all applications within an application pool. For more information on IIS application pools, see Managing Application Pools in IIS

⚠️ Application-specific configurations in the IIS Applications element override application pool configurations.

Optional Element: `<application-pool name="DefaultAppPool" enabled="false">`

Application pool name attribute

The `application-pool name` attribute indicates the name of the IIS Application Pool.

Type: String
Default: None
Required: Name is required for the application-pool element.

Application pool enabled attribute

Set the `application-pool enabled` attribute to `false` to disable instrumentation for all applications in the application pool. Set the value to `true` to instrument all applications in the application pool.

Type: Boolean
Default: None. Defaults to `true` if not specified.
Required: No

Application Pool Tier Element

The `tier` element is a child of the `application-pool` element. If you enable instrumentation for an application pool, you must use a `tier` element to assign the applications of a pool to a tier in the Controller. See Overview of Application Monitoring

Required Element: `<tier name="Inventory" />`

Tier name attribute

Use the `tier name` attribute to specify the tier.

Type: String, may also reference an environment variable. See Reference of Environment Variables
Default: None
Required: Yes

IIS Applications Element

The IIS `applications` element is a child of the `IIS` element. It is a container element for all the IIS applications you configure for instrumentation.
Optional Element: `<applications>`

Application Element

The `application` element is a child of the `applications` element. Use multiple application elements to instrument different sites and applications. To learn more about IIS sites and applications, see Understanding Sites, Applications, and Virtual Directories on IIS 7 and Above

Optional Element: `<application path="/" site="FirstSite" port="8008"site-regex="false">`

Application site attribute

The `application site` attribute indicates the root site in IIS for the application. The site name accepts a regular expression for cases like Windows Azure where you may only know a partial site name. If you use a regular expression, set the Application site-regex attribute to true.

Type: String
Default: None
Required: The site is required for the Application element.

Application site-regex attribute

Set the `application site-regex` attribute to `true` to treat the value of the Application `site` attribute as a regular expression.

Type: Boolean
Default: false
Required: No

Application path attribute

The application `path` attribute indicates the path of the application, relative to the root site. Use the forward slash to indicate the root site and instrument all children applications. Use the path to an application to instrument the specific application and any children.

For example, Site1 hosts two applications `AppX` and `AppY`. To instrument Site 1, `AppY` and `AppZ`, set the path to "/". To instrument `AppY`, but not `AppZ`, set the path to `/AppY`.

Type: String
Default: /
Required: `path` is required for the `application` element.

Application port attribute

For cases where two or more sites in IIS 6 have the same site name, set the site `port` attribute to differentiate between the sites.

Type: Positive Integer
Default**:** None
Required: No

Application enabled attribute

In certain cases you may want to enable instrumentation for a parent application, but disable it for a child application. In this case create an `application` element for the child application to disable and set the `application enabled` attribute to `false`.

Type: Boolean
Default: true
Required: No

Application Tier Element

The `tier` element is a child of the `application` element. If you enable instrumentation for an application, you must use a `tier` element to assign the application to a tier in the Controller. See Overview of Application Monitoring

Required Element: `<tier name="Consumer" />`

Tier name attribute

The `tier name` attribute indicates the business application tier.

Type: String, may also reference an environment variable. See Reference of Environment Variables
Default: None
Required: Yes

Sample IIS Application Configuration

```
<IIS>
  <!-- Automatic instruments all IIS applications when enabled. -->
  <automatic enabled="false" />

  <!-- Application Pool agent configurations -->
  <application-pools>
    <!-- Do not instrument applications in DefaultAppPool when "enabled" attribute is set to false. -->
    <application-pool name="DefaultAppPool" enabled="false">
      <tier name="Tier Name"/>
    </application-pool>

    <!-- Instrument applications in the OtherAppPpool and assign them to the Inventory tier. -->
    <application-pool name="OtherAppPool">
      <tier name="Inventory"/>
    </application-pool>
  </application-pools>
  <applications>
    <!-- Instrument all applications in the First Site. -->
    <application path="/" site="FirstSite">
      <tier name="Order"/>
    </application>
    <!-- Instrument the /app application and child apps in the Second Site -->
    <!-- but not the root Second Site application. -->
    <application path="/app" site="SecondSite">
      <tier name="Consumer"/>
    </application>
    <!-- Regular expression for site name -->
    <!-- assigns all sites beginning with "MyRole" to the Credit Services tier. -->
    <application path="/" site="MyRole_\w+" site-regex="true">
        <tier name="Credit Services"/>
    </application>

  </applications>
</IIS>
```

App Agents - Standalone Applications Element

The `standalone-applications` element is a child of the `app-agents` element. It is a container element for all the Windows services and standalone applications you configure for instrumentation. See Configure the .NET Agent for Windows Services and Standalone Applications

Optional Element: `<standalone-applications>`

Standalone Application Element

The `standalone-application` element is a child of the `standalone-applications` element. It specifies a Windows service or standalone application to instrument.

Optional Element: `<standalone-application executable="MyWindowsApplication.exe" command-line="">`

Standalone Application executable attribute

The `standalone-application executable` attribute specifies the file name of the Windows application to instrument. Set the Standalone Application element executable attribute to one of the following:

- Executable name: for example, `MyStandaloneApp.exe` or `MyWindowsService.exe`. The file extension is optional, so `MyStandaloneApp` also works.
- Full path to the executable: for example, `C:\Program Files\MyApplication\MyStandaloneApp.exe`
- Partial path to the executable: for example, `MyApplication\MyStandaloneApp.exe`. Use the full or partial path to the executable when you want to assign different tiers to separate instances of the same executable running from different paths.

Type: String
Default: None
Required: Yes

Standalone Application command-line attribute

To differentiate between two instances of the same executable, specify any unique portion of the application command line, such as an argument, in the `standalone-application command-line` attribute.

Type: String
Default: None
Required: No

Standalone application app-domain-name attribute

For applications with multiple application domains, the `app-domain-name` attribute enables you to limit instrumentation to specific application domains. See Configure Application Domain Monitoring

Type: String
Default: None
Required: No

Standalone Application Tier Element

The `tier` element is a child of the `standalone-application` element. If you enable instrumentation for an application, you must use a `tier` element to assign the application to a tier in the Controller. See Overview of Application Monitoring

Required Element: `<tier name="Consumer" />`

Tier name attribute

The `tier name` attribute indicates the business application tier.

Type: String, may also reference an environment variable. See Reference of Environment Variables
Default: None
Required: Yes

Environment Variable: APPDYNAMICS.AGENT.TIERNAME

Sample Windows Service and Standalone Application Configuration

```
<standalone-applications>
  <standalone-application executable="ExampleApplication.exe">
    <tier name="Standalone Application Tier"/>
  </standalone-application>
  <!-- Instrument a Windows service using arguments. -->
  <!-- The following example matches the command "MyWindowsService.exe -d -x -r". -->
  <standalone-application executable="MyWindowsService.exe" command-line="-x">
    <tier name="Windows Service Tier"/>
  </standalone-application>
</standalone-applications>
```

.NET Agent Environment Variables

Related pages:

- .NET Agent Configuration Properties
- Administer the .NET Agent
- Configure the .NET Agent
- Name .NET Tiers
- Configure the .NET Agent for Windows Services and Standalone Applications

This page describes the .NET configuration properties you can set using Windows System Environment Variables.

Agent Account Name

Specifies the account name for the SaaS or multi-tenant Controller.

Environment Variable: APPDYNAMICS.AGENT.ACCOUNTNAME
Type: String
Default: For single-tenant Controllers, the agent assumes the default of "customer1" if you do not specify an account name.
Required: Only for SaaS or multi-tenant Controllers.

Agent Application Name

Specifies the business application you use in the Controller. If the application name does not exist, the Controller will create it when the agent registers. See Overview of Application Monitoring.

Environment Variable: APPDYNAMICS.AGENT.APPLICATIONNAME
Type: String
Default: None
Required: Yes

Agent Account Password

Specifies the account access key for the Controller.

Environment Variable: APPDYNAMICS.AGENT.ACCOUNTACCESSKEY
Type: String
Default: None
Required: Yes

Agent Node Name

The name of the node.

In general, the node name must be unique within the business application and physical host. If you want to use the same node name for multiple nodes on the same physical machine, create multiple virtual hosts using the Unique Host ID property. See Unique Host ID.

Environment Variable: APPDYNAMICS_AGENT_NODE_NAME
Type: String
Default: None
Required: Yes

Agent Reuse Node Name

Set this environment variable to true to reuse node names in AppDynamics. When you set the property to true, you don't need to supply a node name, but you do need to provide a node name prefix using appdynamics.agent.reuse.nodeName.prefix.

This property is useful for monitoring environments where there are many CLRs with short life spans. When true, AppDynamics reuses the node names of historical CLRs for new CLRs. This avoids a proliferation of differently named nodes in AppDynamics over time, particularly when the nodes are essentially identical processes that run over different times.

AppDynamics generates a node name with App, Tier, and Sequence number. The node names are pooled. For example, the sequence numbers are reused when the nodes are purged (based on the node lifetime).

When the .NET Agent starts up, it logs output to the console until it registers with the Controller and the Controller generates the node name.

The Controller reuses node names based on the node.retention.period property.

Environment Variable: appdynamics.agent.reuse.nodeName
Type: Boolean
Default: False
Required: No

Example: Using the following environmental variable specifications, the Controller generates a node name with the prefix "reportGen". Node names will have suffixes --1, --2, and so on, depending on the number of nodes are running in parallel. The name of a node that is shut down and qualifies as a historical node may be reused by a new node.

appdynamics.agent.reuse.nodeName=true

appdynamics.agent.reuse.nodeName.prefix=reportGen

Agent Reuse Node Name Prefix

When you configure the agent to reuse node names, use this property to specify the prefix the Controller uses to generate node names dynamically.

Environment Variable: appdynamics.agent.reuse.nodeName.prefix
Type: String
Default: None
Required: When appdynamics.agent.reuse.nodeName=`true`
Example: Using the following environmental variable specifications, the agent directs the Controller to generate a node name with the prefix "reportGen". Node names will have suffixes --1, --2, and so on, depending on how many nodes are running in parallel.

appdynamics.agent.reuse.nodeName=true

appdynamics.agent.reuse.nodeName.prefix=reportGen

Agent Tier Name

Specifies the name of the tier that this .NET node belongs to. Note that this is not the deployment name on the application server.

Environment Variable: APPDYNAMICS_AGENT_TIER_NAME
Type: String
Default: None

Required: Yes

Agent Unique Host ID

Logically partitions a single physical host or virtual machine such that it appears to the Controller that the application is running on different machines. Set the value to a string that is unique across the entire managed infrastructure. The string may not contain any spaces. If you have a machine agent associated with the application monitored by the app agent, then this property must be set on the machine agent to the same value. See Standalone Machine Agent Installation Scenarios.

Environment Variable: APPDYNAMICS_AGENT_UNIQUE_HOST_ID
Type: String
Default: None
Required: No

AWS Instance ID

If the app that you are instrumenting runs on AWS, then .NET Agent supports collecting the AWS Instance ID. To collect the AWS Instance ID, set `APPDYNAMICS.AWS.INSTANCE.ENABLED=true`

Environment Variable: APPDYNAMICS.AWS.INSTANCE.ENABLED
Type: Boolean
Default: false
Required: No

Controller Hostname

Specifies the hostname or the IP address of the AppDynamics Controller. For an on-premises Controller, use the value for Application Server Host Name you provided when you installed the Controller. If you use the AppDynamics SaaS Controller, see the Welcome email from AppDynamics for the name of your Controller.

Environment Variable: APPDYNAMICS.CONTROLLER.HOSTNAME
Type: String
Default: None
Required: Yes, may also be set in the agent config.xml file or using the agent configuration utility.

Controller Port Number

Specifies the HTTP(S) port of the AppDynamics Controller. If the APPDYNAMICS.CONTROLLER.SSL.ENABLED environment variable is set to true, specify the HTTPS port of the Controller; otherwise, specify the HTTP port.

Environment Variable: APPDYNAMICS.CONTROLLER.PORT
Type: Positive Integer
Default: 8090

- For On-premises installations, the defaults are port 8090 for HTTP and port 8181 for HTTPS.
- For the SaaS Controller, use port 80 for HTTP or port 443 for HTTPS.

Required: Yes, may also be set in the agent config.xml file or using the agent configuration utility.

Controller SSL Enabled

When set to "true" enables encryption over SSL between the agent and the Controller.

Environment Variable: APPDYNAMICS.CONTROLLER.SSL.ENABLED
Type: Boolean
Default: false

Required: No, may also be set in the agent config.xml file or using the agent configuration utility.

Reference of Environment Variables

The .NET Agent (through the config.xml file) can reference any environment variables when configuring Agent Node Name, Agent Tier Name, or Controller Application Name.

These environment variables can be referenced in the configuration file or in the agent environment variables for the node, tier, and application names listed above. To include an environment variable, reference it by specifying `%<variable>%` which can be combined with the other characters; for example, `Web-%COMPUTERNAME%` will be translated to `Web-HOST23`.

Configure Multiple Business Application Support for .NET

Related pages:

- .NET Agent Configuration Properties
- Overview of Application Monitoring

By default, applications on a single Windows host are mapped to a single business application in the Controller. If needed, you can manually configure the .NET Agent to map different applications on the same Windows host to different business applications in the Controller.

To map applications on a single host to a different business application, you must edit the `config.xml` file manually. The AppDynamics Agent Configuration utility does not offer this configuration option.

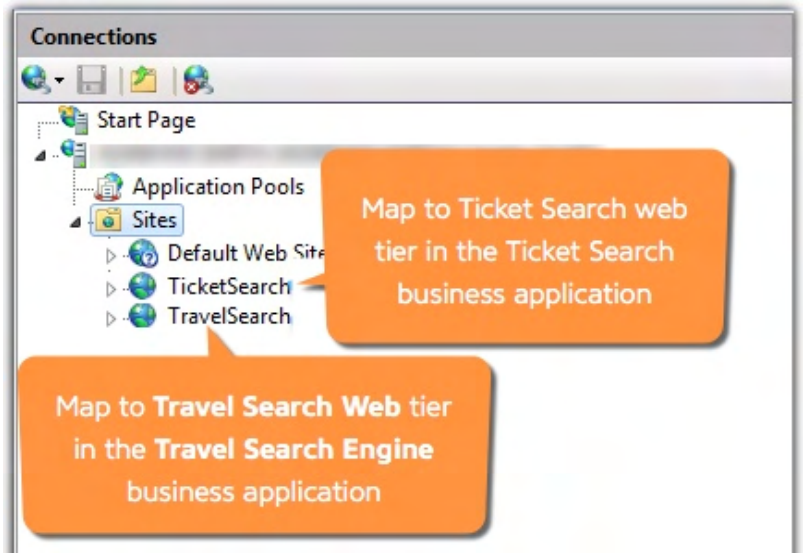
After configuring multiple application support, you cannot use the configuration utility to make configuration changes afterward. If you launch the configuration utility on a server where you configured multiple application support, the utility prompts you to delete the configurations.

Prepare to Configure Multiple Business Applications

Before you configure the .NET Agent, you must install the agent. Use the AppDynamics Agent Configuration utility to perform basic configuration tasks.

1. Identify how to organize your business applications and identify the business application names.

   For example, consider a Windows host running IIS. The IIS instance serves two applications for two separate customers: Ticket Search and Travel Search. The applications perform similar functions, but it makes sense to monitor them separately because they function independently. In this case, create two business applications based upon the application name: Ticket Search Engine and Travel Search Engine.
2. Map your IIS applications or application pools, Windows services, and standalone applications to tiers in the different business applications. For example, map the TicketSearch site to the Ticket Search Web tier in the Ticket Search Engine business application. Map the TravelSearch site to the Travel Search Web tier in the Travel Search Engine business application.



3. If you have not already done so, install the .NET Agent. See Install the .NET Agent for Windows.
4. Run the AppDynamics Agent Configuration utility to generate a config.xml and configure the Controller connection. See Configure the .NET Agent.
5. When prompted, select **Manual** for the method of tier generation and assignment.

> The configuration utility only supports mapping one business application per server.

Manually Configure the .NET Agent

Once you have configured the Controller properties for the .NET Agent, instrument your .NET Applications in the config.xml.

1. Open the `config.xml` file as administrator and edit the file. See 'Where to Configure Agent Properties' on Administer the .NET Agent.
2. Copy the `controller applications` block and paste it as a child element of the `controller` element. Replace any existing `<applications>` or `<application>` elements:

```
 <!--Configure multiple business applications-->
    <applications>
      <application name="ApplicationName1" default ="true"/>
      <application name="ApplicationName2"/>
    </applications>
```

3. Add an `application` element for each of the business applications in the Controller.

   - Edit the `name` attribute for the `application` elements to match the business application names in the Controller. If the application does not exist, the Controller creates it.
   - Set the `default` attribute to `true` for one `application` element. If the agent cannot find a match for the business application name in the IIS application, Windows service, or standalone application configuration, the tier reports to the default business application.

   In this example, `Ticket Search` is the default business application:

```
 <?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org
/2001/XMLSchema">
  <controller host="mycontroller.mycompany.com" port="8090" ssl="false">
        <account name="customer1" password="changeme" />
    <!--Configure multiple business applications-->
    <applications>
      <application name="Ticket Search" default ="true"/>
      <application name="Travel Search"/>
    </applications>
  </controller>
...
```

4. Add configuration elements for the IIS applications or application pools, Windows services, or standalone applications to instrument.
   For IIS applications, add the `applications` block as a child of the `IIS` element. Replace the existing `applications` element. See IIS Applications Element.

```
<applications>
     <application path="/" site="FirstSite" controller-application="Application1">
         <tier name="FirstSite Tier"/>
     </application>
     <application path="/" site="SecondSite" controller-application="Application2">
         <tier name="SecondSite Tier"/>
     </application>
</applications>
```

   For IIS application pools, add the `application pools` block as a child of the `IIS` element. Replace the existing `<application-pools>` element. See IIS Application Pools Element.

```
<application-pools>
    <application-pool name="MyAppPool1" controller-application="Application1">
        <tier name="App1 AppPool Tier"/>
    </application-pool>
    <application-pool name="MyAppPool2" controller-application="Application2">
        <tier name="App2 AppPool Tier"/>
    </application-pool>
</application-pools>
```

   For Windows services or standalone applications, add the `standalone-applications` block as a child of the `app-agents` element. See Standalone Applications Element.

```
<standalone-applications>
    <standalone-application executable="MyStandaloneApp.exe" controller-application="ApplicationName1"
>
        <tier name="Standalone App Tier"/>
    </standalone-application>
    <standalone-application executable="MyWindowsService.exe" command-line="-x" controller-
application="ApplicationName2">
        <tier name="Windows Service Tier"/>
    </standalone-application>
</standalone-applications>
```

5. Configure your application elements:
    - Add the corresponding element for each IIS application or application pool, Windows service, or standalone application to instrument. See .NET Agent Configuration Properties.
    - For each application element set the `controller-application` attribute to the name of the corresponding business application. If you omit the controller-application attribute, the agent adds the application to a tier in the configured default business application.
    - Set the `tier` element `name` attribute to the business application tier name.
6. After you complete configuration, save the changes to the `config.xml` file.
7. Restart the `AppDynamics.Agent.Coordinator` service.
8. Restart IIS applications or application pools, Windows services, and standalone applications.
9. As your applications begin processing traffic, the agent registers them with the Controller. Log on to the Controller to see that your applications have registered with the corresponding business application.

Sample Configuration

This sample `config.xml` file demonstrates configuration for multiple business applications in the Controller. Because the Windows service TicketService does not specify a `controller-application` attribute, it reports to the default business application, `Ticket Search Engine`. All applications in the `TravelAPIPool` pool report to the `Travel Search Engine`.

```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001
/XMLSchema">
<controller host="mycontroller.mycompany.com" port="8090" ssl="false">
<!--Configure multiple business applications-->
  <applications>
    <application name="Ticket Search Engine" default ="true"/>
    <application name="Travel Search Engine"/>
  </applications>
</controller>
<machine-agent />
<app-agents>
  <IIS>
    <automatic enabled="false" />
    <application-pools>
      <application-pool name="TravelAPIPool" controller-application="Travel Search Engine">
        <tier name="Travel APIs"/>
      </application-pool>
    </application-pools>
    <applications>
      <application path="/" site="TicketSearch" controller-application="Ticket Search Engine">
        <tier name="Ticket Search Web"/>
      </application>
      <application path="/" site="TravelSearch" controller-application="Travel Search Engine">
        <tier name="Travel Search Web"/>
      </application>
    </applications>
  </IIS>
<standalone-applications>
  <standalone-application executable="StandaloneApp.exe" controller-application="Ticket Search Engine">
    <tier name="Ticket Standalone Tier"/>
  </standalone-application>
  <standalone-application executable="WindowsService.exe" command-line="-x" controller-application="Travel
Search Engine">
    <tier name="Travel Windows Service Tier"/> </standalone-application>
  </standalone-applications>
</app-agents>
</appdynamics-agent>
```

Agent Configuration Properties for Multiple Application Support

Multiple business application support includes configuration properties for the .NET Agent. These configuration properties supersede the ones documented in .NET Agent Configuration Properties.

Controller Applications Element

The `applications` element is a child of the `controller` element. It is a container element for all controller applications elements that map to business applications in the Controller.

Required Element: `<applications>`

Controller Application Element

The `controller application` element is a child of the `controller applications` element. It indicates the name of the logical business application you see in the Controller. When you have more than one Controller Application element, you must set the default attribute to `true` for one of them.

Required Element: `<application name="MyDotNetApplication" default="true"/>`

Application name attribute

Set the application `name` attribute to the business application name in the Controller. If the application name does not exist, the Controller creates it when the agent registers.

Type: String
Default: None
Required: Yes

Application default attribute

Set the application `default` attribute to `true` for one `controller application` element. Instrumented applications without the `controller application` attribute register with the default business application in the Controller.

Type: Boolean
Default: false
Required: For one application in multiple application configurations

Controller-Application Attribute

The `IIS application`, `IIS application-pool`, `windows-service`, and `standalone-application` elements accept the `controller-application` attribute. Set the value to the `controller application` element name. If you do not include a `controller-application` attribute, the application registers with the default business application.

For example, an IIS application:

```
<application path="/" site="MySite" controller-application="My Business Application">
```

Type: String
Default: None
Required: No

Disable Instrumentation for an IIS Application Pool

Related pages:

- Administer the .NET Agent
- .NET Agent Configuration Properties

When you install the .NET Agent on a machine and use automatic tier naming, the agent instruments every IIS application by default. If you do not need to monitor all application pools, disable monitoring for selected pools.

1. Open the `config.xml` file for editing as administrator. See 'Where to Configure Agent Properties' on Administer the .NET Agent
2. Add the `application-pools` block as a child of the `IIS` element. See .NET Agent Configuration Properties

```
<application-pools>
   <!-- Do not instrument applications in DefaultAppPool when "enabled" attribute is set to false -->
   <application-pool name="DefaultAppPool" enabled="false" >
      <tier name="Disabled App Pool" />
   </application-pool>
</application-pools>
```

> `tier` is a required element even if you are disabling an app pool.

3. Set the `application-pool` element name attribute to the application pool name. This example disables instrumentation for the `DefaultAppPool`. You may add multiple `application-pool` elements.
4. Restart the `AppDynamics.Agent.Coordinator` service.
5. Restart IIS.

Configure Application Domain Monitoring

Related pages:

- Instrument DefaultDomain for Standalone Applications
- Application Domains
- .NET Agent Configuration Properties
- Configure the .NET Agent for Windows Services and Standalone Applications

*Y*ou can configure the .NET Agent to monitor ASP.NET applications with multiple Application Domains (AppDomains). This page assumes that you have a working knowledge of AppDomains and are familiar with the AppDomain implementation in your application.

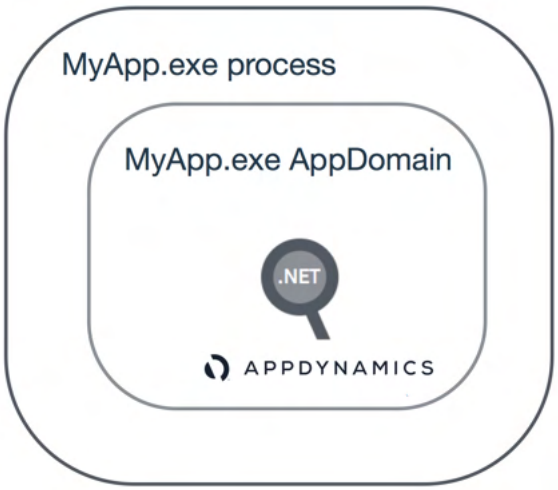
This page does not describe the System Domain, Shared Domain, or DefaultDomain AppDomains that the CLR instantiates before it executes the managed code. If your standalone application runs in the DefaultDomain, see Instrument the DefaultDomain for Standalone Applications.

AppDomains in .NET

Windows uses processes to manage security and performance isolation between running applications. Process isolation ensures that the application's running code does not interfere with another application. However for applications that share data, making calls between Windows processes can introduce complications and performance issues. AppDomains enable developers to create several applications that run inside a single process but maintain application isolation.

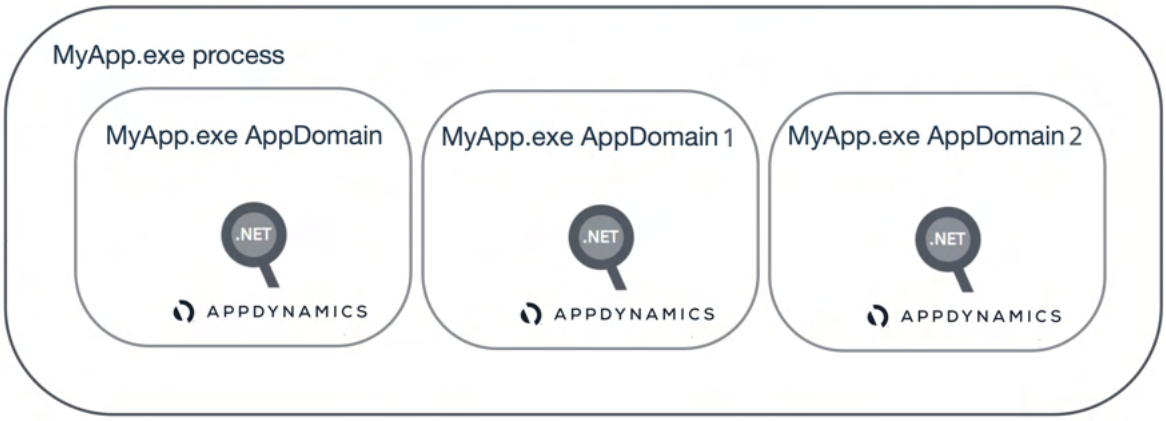
Single Application Domain

When a single application runs inside its own process, the runtime host manages the AppDomain. The application executable and the AppDomain have the same name. The .NET Agent installs itself inside the single AppDomain and creates a node for the application.



Multiple Application Domains

When developers include multiple AppDomains in an application, all the AppDomains run inside a single process. The application executable may have the same name as one AppDomain, but there are other, uniquely-named AppDomains. By default, the agent installs itself inside all the AppDomains and creates nodes for them.



Configure Monitoring for Multiple Application Domains

If the application you monitor contains multiple AppDomains, the App Agent for .NET automatically instruments each AppDomain and creates a node. You can configure the .NET Agent to instrument only the AppDomains you specify. Use this to exclude AppDomains you do not want to monitor, and to limit the number of nodes in a tier.

You can configure application domain monitoring for:

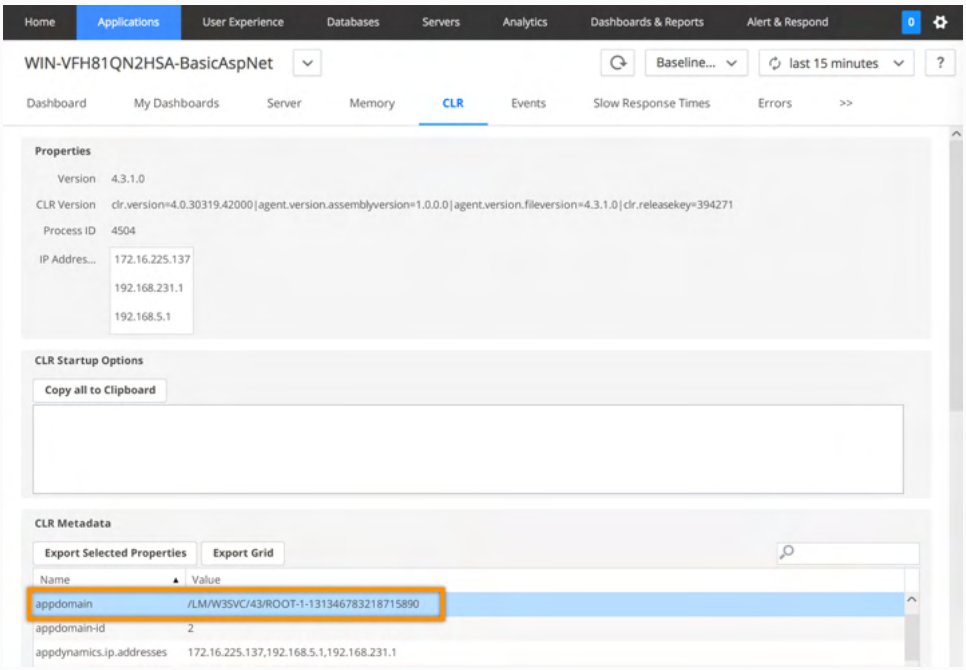
- Windows Services
- Standalone Applications

Configure all instrumentation settings for the .NET Agent in the config.xml file. See Administer the .NET Agent.

1. Identify the name of the AppDomains to instrument.

   If you have already instrumented your application, the AppDomain names display in the Node Dashboard.

   Click the node in the left navigation pane, then click **CLR**.

2. Launch a text editor as administrator.
3. Edit the `config.xml` file as an administrator. See Administer the .NET Agent.
4. Find the element that corresponds to your application with multiple AppDomains:
   Standalone Application element: `<standalone-application executable="MyWindowsApplication.exe">`
5. Add the `app-domain-name` attribute to the element.
   For example, to instrument the `MyApp.exe AppDomain` for the `MyApp.exe` standalone application:

   ```
   <standalone-application executable="MyApp.exe" app-domain-name="MyApp.exe">
       <tier name="StandaloneApplication Tier"/>
   </standalone-application>
   ```

   As soon as you instrument one AppDomain in the `config.xml` file, the agent instruments only the AppDomains you specify. Other AppDomains are not instrumented.

6. To instrument additional AppDomains, add an element for each AppDomain as if they were separate applications.
   For example, to instrument `MyAppDomain1` in `MyApp.exe`:

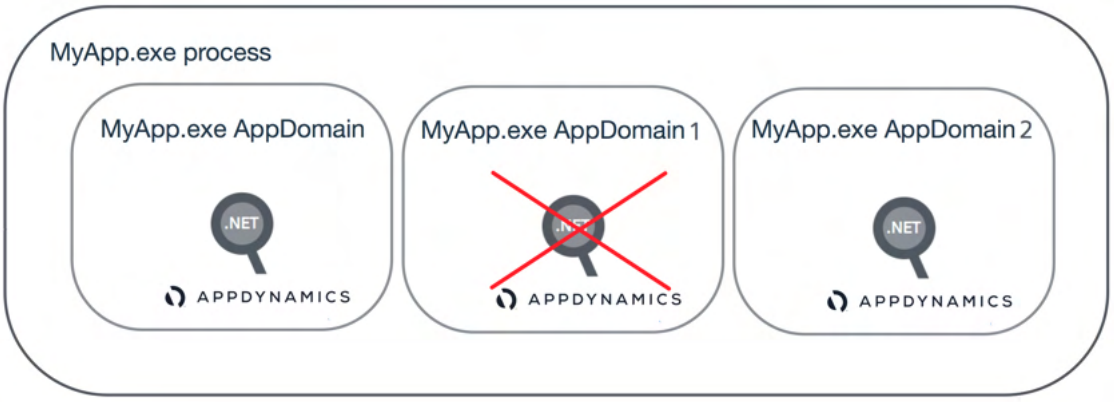
   ```
   <standalone-application executable="MyApp.exe" app-domain-name="MyAppDomain1">
     <tier name="StandaloneApplication Tier"/>
   </standalone-application>
   ```

7. Save the `config.xml` file.

8. Restart the `AppDynamics.Agent.Coordinator` service.
9. Restart instrumented applications: Windows services or standalone applications.

Sample Standalone Application Configuration with Multiple AppDomains

This sample `config.xml` file shows the configuration for the application, `MyApp.exe`. Instrumentation only applies to the AppDomains specified in the `config.xml` file: `MyApp.exe` and `MyAppDomain2`.




```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001
/XMLSchema">
  <controller host="mycontroller.mycompany.com" port="8090" ssl=false">
        <account name="customer1" password="changeme" />
    <application name="MyDotNetApplication" />
  </controller>
  <machine-agent />
  <app-agents>
    <standalone-applications>
      <standalone-application executable="MyApp.exe" app-domain-name="MyApp.exe">
        <tier name="StandaloneApplication Tier"/>
      </standalone-application>
      <standalone-application executable="MyApp.exe" app-domain-name="MyAppDomain2">
        <tier name="StandaloneApplication Tier"/>
      </standalone-application>
    </standalone-applications>
  </app-agents>
</appdynamics-agent>
```

Instrument the DefaultDomain for Standalone Applications

Related pages:

- Configure the .NET Agent for Windows Services and Standalone Applications
- POCO Entry Points
- .NET Agent Configuration Properties

By default, the .NET Agent does not instrument the .NET `DefaultDomain AppDomain`. Before you instrument the `DefaultDomain`:

- Follow the instructions to instrument a standalone application
- Create a POCO entry point for a class/method in the application

If you complete those steps and still do not see business transactions in the Controller, check if your managed code runs in the `DefaultDomain`. If so, you must configure the agent to instrument the `DefaultDomain`.

Check if Your Application Runs in the DefaultDomain

If you are unfamiliar with your application's managed code, you can use the agent logs to identify the AppDomain.

1. Open the agent log:
   Windows Server 2008 and later: `%ProgramData%\AppDynamics\DotNetAgent\Logs\AgentLog.txt`
2. Search the agent log for `AppDomain`.

   Few log entries contain `AppDomain` when the agent starts up. Look for an entry by `dllhost` or your instrumented application similar to the following:

   ```
   2013-12-16 08:23:02.3120 3068 MYPROGRAM 1 1 Info Configuration appDomainName=DefaultDomain appDomainId=1
   iis-app=null site=null port=null appPoolId=
   2013-12-16 08:23:02.6240 3192 dllhost 1 17 Info ConfigurationManager Not instrumenting DefaultDomain for
   pid 3068
   ```

   In this example, `MYPROGRAM` is the name of the instrumented standalone application. You can see the name of the AppDomain in the log entry, `appDomainName=DefaultDomain.`

Instrument the DefaultDomain

1. Open the `config.xml` file for editing as administrator. See 'Where to Configure Agent Properties' on Administer the .NET Agent
2. Copy the code block below to a child element of the `standalone-application` element. See .NET Agent Configuration Properties

   ```
   <profiler>
       <instrument-defaultdomain enabled="true"/>
   </profiler>
   ```

   ⚠️ The Profiler element must follow the Standalone Application Tier element.

   For example:

```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org
/2001/XMLSchema">
  <controller host="mycontroller.mycompany.com" port="8090" ssl="false">
    <application name="My Business Application" />
        <account name="customer1" password="changeme" />
  </controller>
  <machine-agent />
  <app-agents>
    <IIS>
        <applications />
    </IIS>
    <standalone-applications>
      <standalone-application executable="MyStandaloneApp.exe">
        <tier name="Standalone Tier" />
        <profiler>
            <instrument-defaultdomain enabled="true"/>
        </profiler>
      </standalone-application>
    </standalone-applications>
  </app-agents>
</appdynamics-agent>
```

3. Save the `config.xml` file.
4. Restart the `AppDynamics.Agent.Coordinator` service.
5. Restart the standalone application for your changes to take effect.

Configure the .NET Machine Agent

Related pages:

- Administer the .NET Agent
- .NET Agent Configuration Properties
- Monitor Windows Hardware Resources

The AppDynamics .NET Agent includes an embedded .NET Machine Agent that runs as part of the `AppDynamics.Agent.Coordinator` service. Among other things, the Machine Agent regularly gathers system performance data and reports it back to the Controller as metrics.

> Do not confuse the .NET Machine Agent with the Standalone Machine Agent, a Java application. The Standalone Machine Agent provides the capability to use extensions such as plugins, metric listeners, and orchestrations. See Standalone Machine Agent.

The app agent MSI file and .NET Agent Configuration Utility automatically install and configure the .NET Machine Agent to connect to the Controller. The connection information is the same for the app agent and the .NET Machine agent. See Configure the .NET Agent.

To manage the .NET Machine Agent from the Controller, see 'Manage the .NET Machine Agent' on Monitor Windows Hardware Resources.

Customize .NET Machine Agent Behavior

Customize instrumentation settings for the Machine Agent element in the config.xml file. See 'Where to Configure Agent Properties' on Administer the .NET Agent.

1. Shut down the AppDynamics.Agent.Coordinator service.
2. Edit the `config.xml` file as an administrator. See 'Where to Configure Agent Properties' on Administer the .NET Agent.
3. Modify the `machine-agent` element and add any children elements according to the .NET Machine Agent configuration topics.
4. Save the `config.xml` file.
5. Start the `AppDynamics.Agent.Coordinator` service.
6. In some cases, you may need to restart IIS, instrumented Windows services or instrumented Standalone applications. See individual .NET Machine Agent configuration topics.

.NET Machine Agent Configuration Options

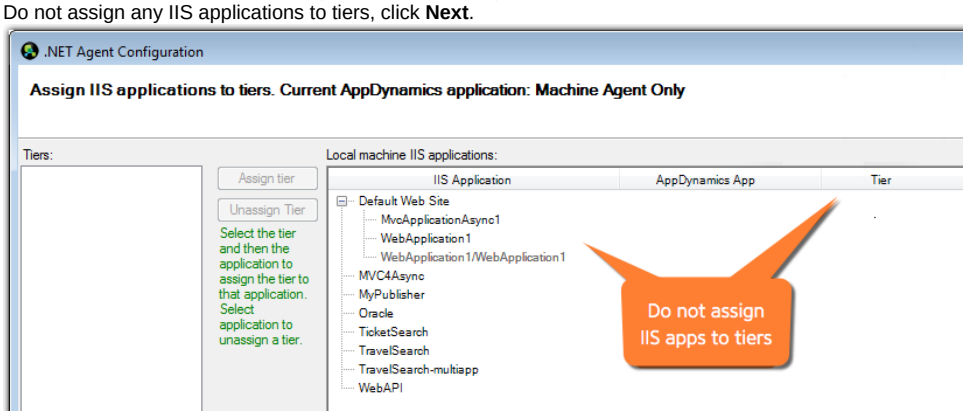
These topics describe specific customizations for the .NET Machine Agent:

- Manage Windows Performance Metrics
- Thread Correlation for .NET
- Enable Correlation for .NET Remoting
- Enable Instrumentation for WCF Data Services
- Configure Machine Snapshots for .NET

Configure the .NET Machine Agent Without App Agents

If you want to monitor the Windows hardware performance data, but do not want to monitor any .NET applications, you can configure the .NET Machine agent to run without the .NET Agent.

1. Install the .NET Agent for Windows.
2. Launch the AppDynamics Agent Configuration utility and follow the steps until you reach the Assign IIS applications to tiers pane.
3. Click **Manual** for the method of tier generation and assignment and click **Next**.
4. Do not assign any IIS applications to tiers, click **Next**.



> If you configured any Windows services or standalone applications, manually disable those agents in the `config.xml` file.

5. Continue with the remaining steps and click **Done**.

   Monitoring resumes for the .NET Machine Agent only. Metrics appear in the Controller under the Machine Agent tier, see Monitor Windows Hardware Resources.

Manage Windows Performance Metrics

The AppDynamics .NET Machine Agent uses Microsoft Performance Counters to gather and report .NET metrics. To manage the metrics that the agent collects:

- Tune the set of metrics that the agent automatically collects
- Configure the agent to collect metrics from additional counters that are not part of the default settings

Tune the Default Performance Metrics for the .NET Agent

The .NET Agent prioritizes the default set of Performance Counters into three categories: low, medium, and high. By default, the agent is set to collect metrics for all three categories.

Metric Browser Location	Metric	Priority
ASP.NET	Application Restarts	3 - Low
ASP.NET	Applications Running	3 - Low
ASP.NET	Request Wait Time	2 - Medium
ASP.NET	Requests Disconnected	3 - Low
ASP.NET	Requests Queued	1 - High
ASP.NET	Requests Rejected	3 - Low
ASP.NET	Worker Process Restarts	3 - Low
ASP.NET Applications	Anonymous Requests	3 - Low
ASP.NET Applications	Anonymous Requests/Sec	3 - Low
ASP.NET Applications	Cache API Entries	3 - Low
ASP.NET Applications	Cache API Hit Ratio	3 - Low
ASP.NET Applications	Cache API Turnover Rate	3 - Low
ASP.NET Applications	Cache Total Entries	3 - Low
ASP.NET Applications	Cache Total Hit Ratio	3 - Low
ASP.NET Applications	Cache Total Turnover Rate	3 - Low
ASP.NET Applications	Errors During Compilation	3 - Low
ASP.NET Applications	Errors During Execution	3 - Low
ASP.NET Applications	Errors During Preprocessing	3 - Low
ASP.NET Applications	Errors Total	2 - Medium
ASP.NET Applications	Errors Total/sec	2 - Medium
ASP.NET Applications	Errors Unhandled During Execution	3 - Low
ASP.NET Applications	Errors Unhandled During Execution/sec	3 - Low
ASP.NET Applications	Output Cache Entries	3 - Low
ASP.NET Applications	Output Cache Hit Ratio	3 - Low
ASP.NET Applications	Output Cache Turnover Rate	3 - Low
ASP.NET Applications	Pipeline Instance Count	3 - Low
ASP.NET Applications	Requests Executing	3 - Low
ASP.NET Applications	Requests Failed	3 - Low
ASP.NET Applications	Requests in Applicaton Queue	1 - High
ASP.NET Applications	Requests Not Authorized	3 - Low
ASP.NET Applications	Requests Not Found	3 - Low

ASP.NET Applications	Requests Succeeded	3 - Low
ASP.NET Applications	Requests Timed Out	2 - Medium
ASP.NET Applications	Requests Total	2 - Medium
ASP.NET Applications	Requests/sec	2 - Medium
ASP.NET Applications	Session SQL Server Connections Total	3 - Low
ASP.NET Applications	Session State Server Connections Total	3 - Low
ASP.NET Applications	Sessions Abandoned	3 - Low
ASP.NET Applications	Sessions Active	3 - Low
ASP.NET Applications	Sessions Timed Out	3 - Low
ASP.NET Applications	Sessions Total	3 - Low
ASP.NET Applications	Transactions Aborted	3 - Low
ASP.NET Applications	Transactions Committed	3 - Low
ASP.NET Applications	Transactions Pending	3 - Low
ASP.NET Applications	Transactions Total	3 - Low
ASP.NET Applications	Transactions/sec	3 - Low
CLR	Process CPU Burnt %	2 - Medium
CLR -> Classes	Current Loaded Class Count	3 - Low
CLR -> Classes	Total Classes Loaded	3 - Low
CLR -> Locks and Threads	Contention Rate Per Sec	2 - Medium
CLR -> Locks and Threads	Current Logical Threads	2 - Medium
CLR -> Locks and Threads	Current Physical Threads	2 - Medium
CLR -> Locks and Threads	Sink Blocks in Use	2 - Medium
CLR -> Memory -> Heap	Committed (bytes)	1 - High
CLR -> Memory -> Heap	Current Usage (bytes)	1 - High
CLR -> Memory -> Heap	Gen 0 Usage (bytes)	1 - High
CLR -> Memory -> Heap	Gen 1 Usage (bytes)	1 - High
CLR -> Memory -> Heap	Gen 2 Usage (bytes)	1 - High
CLR -> Memory -> LOH	Current Usage (bytes)	1 - High
Hardware Resources -> CPU	%Busy	1 - High
Hardware Resources -> CPU	%Idle	3 - Low
Hardware Resources -> Disks	%Free	1 - High
Hardware Resources -> Disks	KB read/sec	3 - Low
Hardware Resources -> Disks	KB written/sec	3 - Low
Hardware Resources -> Disks	MB Free	1 - High
Hardware Resources -> Disks	Reads/sec	1 - High
Hardware Resources -> Disks	Writes/sec	1 - High
Hardware Resources -> Memory	Free %	3 - Low
Hardware Resources -> Memory	Free (MB)	3 - Low
Hardware Resources -> Memory	Total (MB)	1 - High
Hardware Resources -> Memory	Used %	1 - High
Hardware Resources -> Memory	Used (MB)	2 - Medium

Hardware Resources -> Network	Incoming KB/sec	2 - Medium
Hardware Resources -> Network	Incoming packets/sec	3 - Low
Hardware Resources -> Network	Outgoing KB/sec	2 - Medium
Hardware Resources -> Network	Outgoing packets/sec	3 - Low
IIS	CPU %	1 - High
IIS	Number of Working Processes	3 - Low
IIS	Working Set	1 - High
IIS -> Application Pools -> <name>	CPU %	1 - High
IIS -> Application Pools -> <name>	Number of Working Processes	3 - Low
IIS -> Application Pools -> <name>	Working Set	1 - High
IIS -> INFO	VersionMajor	3 - Low
IIS -> INFO	VersionMinor	3 - Low

To reduce the number of metrics that the agent sends to the Controller, or to view less than a full set of metrics, you can configure that agent to collect fewer performance metrics. You can also set the agent to only collect specific metrics from the list. See .NET Agent Configuration Properties.

Configure Additional Performance Counters for .NET

If the agent does not monitor performance counters you are interested in, you can add additional performance counters.

1. Shut down the `AppDynamics.Agent.Coordinator` service.
2. Open the `config.xml` file for editing as an administrator. See 'Where to Configure Agent Properties' on Administer the .NET Agent.
3. Add the Performance Counters block as a child of the Machine Agent element.

```
<perf-counters>
  <perf-counter cat="" name="" instance=""/>
</perf-counters>
```

4. Create a Performance Counter element for each performance counter you want to add. Use any of the performance counters as specified in Performance Counters in .NET Framework.

    - Set the `cat` attribute to the category of the performance counter.
    - Set the `name` attribute to the performance counter name.
    - Set the `instance` attribute to the instance of the performance counter.
      If the counter does not have an instance name, leave an empty string `""`.
      If a particular performance counter has many instances you can specify the following options:
        - instance =`"*"` OR
        - instance =`"all"` (This reports the sum of all instances)

    For example, to add the performance counter for measuring CPU Idle time(%), add this element in the `<perf-counters>` block:

```
<perf-counter cat="Processor" name="% Idle Time" instance="_Total"/>
```

5. Save the `config.xml` file.
6. Start the `AppDynamics.Agent.Coordinator` service.

After you enable Performance Counter metrics, they display under the Custom Metrics tree in the Metric Browser.

Sample .NET Machine Agent Configuration with Performance Counters

```
<machine-agent>
  <!-- Additional machine level Performance Counters -->
  <perf-counters>
    <perf-counter cat="Processor" name="% Idle Time" instance="_Total"/>
    <!-- Perf counter with no instance name -->
    <perf-counter cat="Memory" name="Available MBytes" instance="" />
  </perf-counters>
</machine-agent>
```

Enable Correlation for .NET Remoting

Related pages:

- .NET Backend Detection
- Remote Services

Developers use .NET remoting to build distributed applications that share objects across processes or across application domains running in the same process. AppDynamics disables correlation for .NET remoting functions by default.

Instrument Applications that Use .NET Remoting

You can configure the .NET Agent to discover .NET remoting entry and exit points.

1. Open the `config.xml` file for editing as administrator. See 'Where to Configure Agent Properties' on Administer the .NET Agent
2. Copy the code block below to a child element of the Machine Agent element. See .NET Agent Configuration Properties

   ```
       <instrumentation>
           <instrumentor name="RemotingMscorlibEntryInstrumentor" enabled="true"/>
           <instrumentor name="RemotingExitInstrumentor" enabled="true"/>
       </instrumentation>
   ```

   For example:

   ```
   <?xml version="1.0" encoding="utf-8"?>
   <appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org
   /2001/XMLSchema">
   ...
    <machine-agent>
       <!--Enable correlation for .NET remoting-->
       <instrumentation>
           <instrumentor name="RemotingMscorlibEntryInstrumentor" enabled="true"/>
           <instrumentor name="RemotingExitInstrumentor" enabled="true"/>
       </instrumentation>
     </machine-agent>
   ...
   </appdynamics-agent>
   ```

3. Save the `config.xml` file.
4. Restart the `AppDynamics.Agent.Coordinator` service.
5. Restart instrumented applications for your changes to take effect.

If the agent does not discover the entry points after configuration, specify an agent trigger.

Specify an Agent Trigger

.NET remoting entry point functions execute in low-level .NET libraries that may not trigger automatic agent instrumentation. If the agent does not discover the .NET remoting entry points after configuration you can specify a function that triggers the agent to begin instrumentation.

1. Identify a function to trigger the agent to begin instrumentation. The function can be any function that executes as part of the application process.

   For example, consider the following code for a `MovieTicket` remoting object. In this case, use the function `GetTicketStatus` to trigger the agent.

```
using System;
namespace MovieGoer
{
    public class MovieTicket : MarshalByRefObject
    {
        public MovieTicket()
        {
        }
        public string GetTicketStatus(string stringToPrint)
        {
            return String.Format("Enquiry for {0} -- Sending back status: {1}", stringToPrint, "Ticket
Confirmed");
        }
    }
}
```

2. Edit the `config.xml` file as an administrator. See Administer the .NET Agent.
3. Update the Instrumentation element to include the `AgentTriggerInstrumentor` attribute. See .NET Agent Configuration Properties.

```
    <instrumentation>
                <instrumentor name="AgentTriggerInstrumentor" enabled="true" args="" />
        <instrumentor name="RemotingMscorlibEntryInstrumentor" enabled="true"/>
        <instrumentor name="RemotingExitInstrumentor" enabled="true"/>
    </instrumentation>
```

4. Set the `AgentTriggerInstrumentor args` value to the name of the trigger function, using information in the code block in step 1. The `args` value should be formatted as follows: `namespace.public_class.public_string`.

   For example:

```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org
/2001/XMLSchema">
...
 <machine-agent>
    <!--Enable correlation for .NET remoting-->
    <instrumentation>
        <instrumentor name="AgentTriggerInstrumentor" enabled="true" args="MovieGoer.MovieTicket.
GetTicketStatus" />
        <instrumentor name="RemotingMscorlibEntryInstrumentor" enabled="true"/>
        <instrumentor name="RemotingExitInstrumentor" enabled="true"/>
    </instrumentation>
  </machine-agent>
...
</appdynamics-agent>
```

5. Save the `config.xml` file.
6. Restart instrumented applications for your changes to take effect.

Thread Correlation for .NET

Related pages:

- Configure Backend Detection for .NET
- Remote Services

By default, the AppDynamics .NET Agent enables multi-threaded correlation for these patterns on the Common Language Runtime (CLR) >= 4.x:

- `Task.Start`
- `Task.Run`
- `TaskFactory.StartNew`

The .NET Agent also supports thread correlation for the following patterns which are disabled by default:

- `Thread.Start` on the CLR 2.x and CLR 4.x
- `ThreadPool.QueueUserWorkItem` on the CLR 2.x and CLR 4.x

For AppDynamics < 4.3, for the .NET Agent, you did not need to configure correlation for `ThreadPool.QueueUserWorkItem` for the CLR 4 in the `config.xml` file. If you explicitly call `ThreadPool.QueueUserWorkItem` for the CLR 4, when you upgrade to the .NET Agent 4.5 from an earlier version, you must enable the instrumentor for `ThreadPool.QueueUserWorkItem` on the CLR 4 in the `config.xml` file.

Configure Thread Correlation for .NET

Configure all instrumentation settings for the .NET Agent in the `config.xml` file. See Administer the .NET Agent

1. Open the `config.xml` file for editing as an administrator. See Administer the .NET Agent.
2. Copy the code block below to a child element of the Machine Agent element. See Machine Agent Element.

   ```
       <instrumentation>
           <instrumentor name="ThreadCorrelationThreadPoolCLR2Instrumentor" enabled="true"/>
           <instrumentor name="ThreadCorrelationThreadPoolCLR4Instrumentor" enabled="true"/>
           <instrumentor name="ThreadStartCLR2Instrumentor" enabled="true"/>
           <instrumentor name="ThreadStartCLR4Instrumentor" enabled="true"/>
       </instrumentation>
   ```

   For example:

   ```
   <?xml version="1.0" encoding="utf-8"?>
   <appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org
   /2001/XMLSchema">
   ...
    <machine-agent>
       <!--Enable thread correlation-->
       <instrumentation>
           <instrumentor name="ThreadCorrelationThreadPoolCLR2Instrumentor" enabled="true"/>
           <instrumentor name="ThreadCorrelationThreadPoolCLR4Instrumentor" enabled="true"/>
           <instrumentor name="ThreadStartCLR2Instrumentor" enabled="true"/>
           <instrumentor name="ThreadStartCLR4Instrumentor" enabled="true"/>
       </instrumentation>
     </machine-agent>
   ...
   </appdynamics-agent>
   ```

   The configuration syntax is `enabled="true"`.
3. Save the `config.xml` file.
4. Restart the `AppDynamics.Agent.Coordinator` service.
5. Restart instrumented applications for your changes to take effect.

Enable Instrumentation for WCF Data Services

Related pages:

- .NET Business Transaction Detection
- .NET Agent Configuration Properties

This page describes how to configure the AppDynamics .NET Machine agent to enable instrumentation for Windows Communication Foundation (WCF) Data Services including WCF RIA Services for Microsoft LightSwitch.

1. Open the `config.xml` file for editing as administrator. See Administer the .NET Agent.
2. Copy the code block below to a child element of the Machine Agent element. See .NET Agent Configuration Properties.

   ```
       <instrumentation>
           <instrumentor name="WCFDSEntryInstrumentor" enabled="true" />
       </instrumentation>
   ```

   For example:

   ```
   <?xml version="1.0" encoding="utf-8"?>
   <appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org
   /2001/XMLSchema">
   ...
    <machine-agent>
       <!--Enable instrumentation for WCF Data Services correlation-->
       <instrumentation>
           <instrumentor name="WCFDSEntryInstrumentor" enabled="true" />
       </instrumentation>
     </machine-agent>
   ...
   </appdynamics-agent>
   ```

   The configuration syntax is `enabled="true"`.
3. Save the `config.xml` file.
4. Restart the `AppDyanmics.Agent.Coordinator` service.
5. Restart instrumented applications for your changes to take effect.

Configure Machine Snapshots for .NET

The .NET Machine Agent takes machine snapshots to capture vital system data for Windows and IIS. Use these guidelines to tune the frequency the .NET Machine Agent takes machine snapshots in your environment. See Machine Snapshots for .NET.

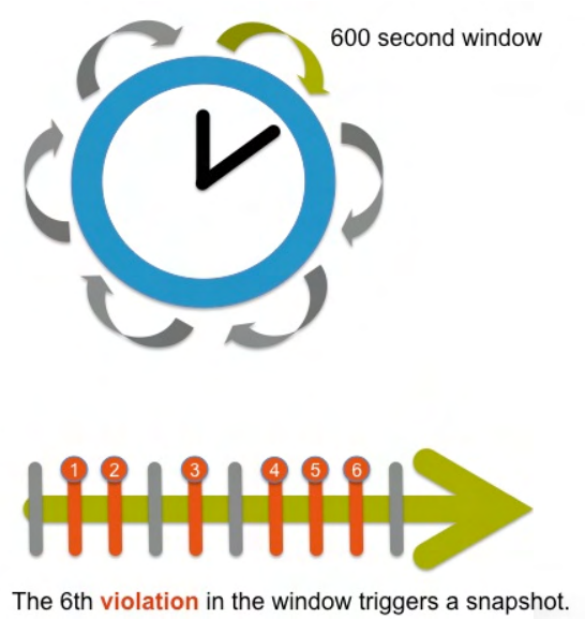
Default Settings and Configuration Considerations

By default, the .NET Machine Agent takes machine snapshots under these conditions:

- Periodic collection: The agent takes one snapshot every 10 minutes.
- Breached thresholds: The .NET Machine Agent takes samples of machine statistics every 10 seconds within a 10-minute window. For each sample, the agent checks the CPU percent usage, the memory percent usage, and the oldest item in the IIS application pool queue. The agent flags a sample as a violation when the current usage meets or exceeds one of these thresholds:
  - CPU at 80% or higher
  - Memory at 80% or higher
  - IIS application pool queue item older than 100 milliseconds

  The agent takes a snapshot when it identifies 6 violations of a single type, such as CPU usage, within the window. The agent only takes one snapshot per window for breached thresholds.

With the default window size of ten minutes and the violations per window of six, the sixth violation of a single type triggers a machine snapshot:



Before you change the machine snapshot settings, determine which configuration options work best for your environment. Use these questions and considerations to help determine your threshold settings:

- What percentage CPU or memory usage might flag the beginnings of an issue in your environment?
- How long is too long for items to wait in the IIS queue?
- Do you expect occasional CPU or memory spikes?
- Are periodic collections every ten minutes frequent enough?
- Make sure the value for violations is larger than the number of samples per window.

For example, if you decrease the window size to 60 seconds and take six samples per window, the agent takes samples at the same frequency as the default settings, once every 10 seconds. However, you are likely to get more snapshots because the agent only takes one snapshot per window. If you set the violations per window to five, the agent takes a snapshot any time half the samples in the window violate a specific threshold. See .NET Agent Configuration Properties.

Configure Machine Snapshots for the .NET Machine Agent

Configure all instrumentation settings for the .NET Machine Agent in the config.xml file:

1. Open the `config.xml` file for editing as an administrator. See Administer the .NET Agent.
2. Copy the code block below to a child element of the AppDynamics Agent element. See AppDynamics Agent Element.

```
<machine-agent>
        <machine-snapshot enabled="true" window-size="600" samples-per-window="60" violations-per-
window="6" max-percent-cpu="80" max-percent-memory="80" max-queue-item-age="100" periodic-collection="
600" />
</machine-agent>
```

If you have already customized the .NET Machine Agent, only copy the `machine-snapshot` element and paste it as a child of the `machine-agent` element.

3. Edit the values for the `machine-snapshot` element attributes:

```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org
/2001/XMLSchema">
...
 <machine-agent>
    <!--Configure machine snapshots-->
        <machine-snapshot enabled="true" window-size="60" samples-per-window="10" violations-per-window="
5" max-percent-cpu="80" max-percent-memory="80" max-queue-item-age="100" periodic-collection="600"/>
  </machine-agent>
...
</appdynamics-agent>
```

4. Save the `config.xml` file.
5. Restart the `AppDynamics.Agent.Coordinator` service.

Configure Runtime Reinstrumentation for .NET

By default, you must restart your monitored application after you create or update:

- POCO transaction detection entry points
- POCO service endpoints
- Custom exit points
- Method invocation and HTTP data collectors
- Information points

When you enable runtime reinstrumentation, the .NET Agent detects the types of instrumentation changes and automatically requests the CLR to recompile the affected code. With runtime reinstrumentation enabled, you do not need to restart your application for the instrumentation changes to take effect. See Administer the .NET Agent.

Requirements

Runtime reinstrumentation works for .NET Framework >= 4.5.2. To verify the framework version in the `web.config` file of your application, enter:

```
<system.web>
    <compilation targetFramework="4.5.2" />
    <httpRuntime targetFramework="4.5.2" />
</system.web>
```

Configure Runtime Reinstrumentation for .NET

*C*onfigure all instrumentation settings for the .NET Agent in the `config.xml` file:

1. Open the `config.xml` file for editing as an administrator. See Administer the .NET Agent.
2. Copy the code block below to a child element of the `app-agents` element. See Default Profiler Element.

   ```
   <profiler>
      <runtime-reinstrumentation enabled="true" interval="60000" />
   </profiler>
   ```

   For example:

   ```
   <?xml version="1.0" encoding="utf-8"?>
   <appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org
   /2001/XMLSchema">
   ...
   <app-agents>
       <profiler>
         <!-- Enable runtime reinstrumentation -->
         <runtime-reinstrumentation enabled="true" interval="60000"/>
       </profiler>
   ...
   </app-agents>
   </appdynamics-agent>
   ```

3. If appropriate configure the `runtime-reinstrumentation` options:
   - `enabled`: Set to `true` to enable runtime reinstrumentation. Default: `false`
   - `interval`: The frequency the agent checks for configuration updates in milliseconds. Default: `60000`.

   > Because runtime reinstrumentation adds a small amount of system overhead, AppDynamics recommends a minimum interval of 1 minute or 60000 milliseconds.
   >
   > The .NET Agent checks for configuration updates every minute, so it may take up to 2 minutes for reinstrumentation to take effect with the default setting of 60000 milliseconds.

4. Save the `config.xml` file.
5. Restart the `AppDynamics.Agent.Coordinator` service.
6. Restart instrumented applications for your changes to take effect.

Filter Sensitive Data with the .NET Agent

By default, the AppDynamics .NET Agent sends transaction data to the Controller that your organization may classify as privileged information. Although this data is useful for diagnosis and troubleshooting, security considerations may require you to filter certain information from view in the Controller. You can use sensitive data filters to exclude environment variables or URLs in the Controller or information in snapshot details.

Sensitive Data Filters are available for the .NET Windows and Linux Agents (as of 21.4).

Add a Sensitive Data Filter

To instrument sensitive data filters, you edit the .NET Agent configuration file. See Administer the .NET Agent.

1. Edit a versioned agent configuration file:
   - Full .NET Agent: `config.xml`
   - Standalone Agent: `config.json`
2. Add a `sensitive-data-filter` element as a child of the Sensitive Data Filters element using one of these attributes:

```
<!-- Filter environment variable values sent to controller -->
<sensitive-data-filters>
        <sensitive-data-filter applies-to="environment-variables, system-properties, http-headers, http-
cookies",
                                    match-type="EQUALS|CONTAINS|STARTSWITH|ENDSWITH"
                                    match-pattern=""/>
</sensitive-data-filters>
```

   Sensitive Data Filter Attributes:

   - Specify a comma-separated list in the `applies-to` attribute to filter `environment-variables, system-properties`, or data collectors (`http-headers`, `http-cookies`)
   - Set the match-type attribute as:

     `EQUALS`

     `CONTAINS`

     `STARTSWITH`

     `ENDSWITH`
   - Specify a string to match for the `match-pattern` attribute. String matches are case insensitive. The pattern matches the environment variable and system property names, not values.
3. Restart the .NET Agent and application.

In this example, the .NET Agent checks for environment variables beginning with the string "DB_". The Controller displays the values of matching environment variables and system properties as asterisks. For example, an environment variable `DB_USER` is obfuscated and replaced with an asterisk in the Controller.

Example config.xml file:

```
<sensitive-data-filter applies-to="environment-variables, system-properties",
                                    match-type="STARTSWITH"
                                    match-pattern="DB_"/>
</sensitive-data-filters>
```

Example config.json file:

```
sensitive-data-filters: [
{
"applies-to": "environment-variables, system-properties",
"match-type": "STARTSWITH",
"match-pattern": "DB_"
}
]
```

Add a Sensitive URL Filter

You can use sensitive URL filters to configure the agent to obfuscate sensitive information in the URLs in the Controller.

To instrument sensitive URL filters, you edit the .NET Agent configuration file. See Administer the .NET Agent.

1. Edit a versioned agent configuration file:
    - Full .NET Agent: `config.xml`
    - Standalone Agent: `config.json`
2. Add attributes to the `sensitive-url-filters` element:

```
<sensitive-url-filters>
        <sensitive-url-filter delimiter=""
                                          segment=""
                                          match-filter=
“EQUALS|INLIST|STARTSWITH|ENDSWITH|CONTAINS|REGEX|NOT_EMPTY"
                                          match-pattern="pattern"
                                          param-pattern=""/>
</sensitive-url-filters>
```

Sensitive URL Filter Attributes:

- `delimiter`: Specify the character that you want to use as URL segment endpoints. The agent splices the URL at each delimiter instance to create the segments. For HTTP, use the forward-slash character "/". In the case of a forward slash, the agent does not split on the slashes immediately following the protocol. For example, "`https://myapp.example.com/`" constitutes a single segment. By default, the delimiter is "/".
- `segment`: Specify a comma-separated list to indicate the segments that you want the agent to filter. Segment numbering starts from 1.
- `match-pattern`: Specify the string that you want to be filtered by the `match-filter`.
- `param-pattern`: Specify the regular expression matching the query parameters to filter.

For example, this configuration splits the URL on the "/" character and masks the second segment, and the param-pattern in the third segment of the URL. In this example, the segmentation and obfuscation apply only to URLs containing "`myapp`".

```
<!-- Filter URL/URI segments and query parameters -->
<sensitive-url-filters>
        <sensitive-url-filter delimiter="/"
                                           segment="2"
                                           match-filter="CONTAINS"
                                           match-pattern="myapp"
                                           param-pattern="[a-z]+"/>
</sensitive-url-filters>
```

If you do not use any values for the query parameters, the Controller does not mask any query parameters in the URL.

The exit call to `https://myapp.example.com/sensitive/data?first_name=abc&last_name=xyz` breaks down to three segments:

- `"https://myapp.example.com"`
- `"sensitive"`
- `"data?first_name=abc&last_name=xyz"`

The Controller shows the masked values of the URL and the param-pattern displays `https://myapp.example.com/*****/data?first_name=***&last_name=***` in the snapshot details.

Add a Sensitive Message Filter

You can use the `sensitive-message-filters` element to obfuscate sensitive information contained within text messages collected from exception messages.

1. Edit a versioned agent configuration file.
2. Add attributes to the `sensitive-message-filters` element:

```
<sensitive-message-filters>
        <sensitive-message-filter message-type="all"
                                  match-type="EQUALS|CONTAINS|STARTSWITH|ENDSWITH|REGEX"
                                  match-pattern="CASESENSITIVE_PATTERN"
                                  redaction-regex="SENSITIVE_INFO_REGEX_GROUP"/>
</sensitive-message-filters>
```

Sensitive Message Filter Attributes:

- `message-type`: Specify `all`
- `match-type`: Specify the type of match that should be used to opt-in messages for redaction
- `match-pattern`: Specify the pattern that, when matched, opts the message in for redaction
- `redaction-regex`: Specify a regular expression identifying data that should be redacted from the opted-in messages

This example filters out a test where `match-pattern` contains `"Sensitive"`.

```
<sensitive-message-filters>
        <sensitive-message-filter message-type="all"
                                  match-type="CONTAINS"
                                  match-pattern="Sensitive"
                                  redaction-regex="[0-9]+"/>
</sensitive-message-filters>
```

Manage Configuration for .NET

Related pages:

- Administer the .NET Agent
- .NET Agent Configuration Properties

Highly-scaled environments frequently contain multiple machines that run the same .NET applications and therefore use the same AppDynamics .NET Agent configuration. Manual configuration updates to several machines can be tedious and error-prone.

Use the Config Management tool in the AppDynamics Controller to upload and manage .NET Agent `config.xml` files. By default the .NET Agent checks for configuration updates from the Controller every minute. Use the Config Management tool to assign configurations to machines where the .NET Agent is running.

Click **Settings > AppDynamics Agents > Config Management** to manage all the .NET Agent configurations files for your machines.

Requirements

- Membership in a role with 'Administer user, groups, roles, authentication, and so on' account level permission.
- A valid `config.xml` configuration file. The Config Management tool does not validate the `config.xml` file.
- .NET Agent >= 4.1 installed on machines where you plan to deploy configuration.

.NET Agents deployed on Azure Cloud Services using the AppDynamics NuGet package do not register for Config Management because web role and worker role machine names are not always unique. Duplicate machine names may lead to unexpected consequences.

Identify the Configuration File

To start managing your configurations in the Controller, identify a configuration file or set of configuration files to upload. If you are not familiar with .NET Agent configuration, AppDynamics recommends you use the Agent Configuration Utility on each machine to configure the agent. See Administer the .NET Agent and Configure the .NET Agent.

For a full list of configuration properties, see .NET Agent Configuration Properties.

- Use `config.xml` files configured with the AppDynamics Agent Configuration Utility to help ensure the XML is valid.
- Use the `config.xml` file from a previously-configured machine as a template for similar machines that need the exact same configuration.

Import Configuration

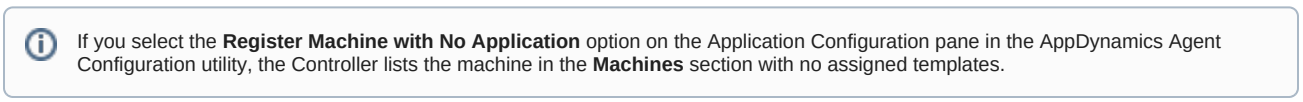
The Import Config dialog enables you to upload a `config.xml` file to the Controller, and specify a unique name for the configuration.

- Use a naming convention that helps you identify the type of machine for the configuration.
- Consider using a date or version scheme for the configuration name.
- Click a configuration, then click **Edit Config** to modify the configuration name.
- Double-click a config to display the XML. You can copy the XML, but you cannot modify it.

Assign Configuration to Machines

The Config Management tool enables you to assign machines to a configuration, or change the configuration from one to another. After you assign the configuration in the Controller, the .NET Agent automatically downloads the configuration file.

- If a machine does not have a managed configuration, the Config Management tool displays it following `Machine with no assigned templates`.

  If you select the **Register Machine with No Application** option on the Application Configuration pane in the AppDynamics Agent Configuration utility, the Controller lists the machine in the **Machines** section with no assigned templates.

- Drag and drop to assign a machine to a configuration.
- The .NET Agent checks for updated configurations every minute.
- The agent backs up the old `config.xml` file as `config-<date>.xml` in the backup subdirectory of the `econfig` directory. See .NET Agent Directory Structure.
- If the agent detects an invalid `config.xml file`, it does not apply the managed configuration and it raises an Agent Configuration Error event in the Controller. The event summary indicates the affected tiers and nodes.
- Regardless of the Controller connection information in the managed configuration, the agent always retains existing Controller connection information.
- After the agent receives and validates the assigned configuration, any changes take effect after the app pools recycle or IIS restarts. You must restart Windows services or standalone applications. It is unnecessary to restart the AppDynamics Agent Coordinator service.

Opt Out of Configuration Management

To configure an agent so that it does not receive configuration files from the Controller, set the `enable_config_deployment` Controller attribute to `false` in the `config.xml` file. See .NET Agent Configuration Properties.

For example:

```
<controller host="mycontroller.saas.appdynamics.com" port="443" ssl="true" enable_config_deployment="false">
```

Restore an Old Configuration

If you mistakenly assign a configuration, follow this procedure to restore a previous version of the config.xml:

1. On the Config Management pane, drag the machine from its currently assigned configuration to **Machine with no assigned templates**.
2. On the machine itself, rename the existing configuration file, `config.xml`, to a name such as `config.old.xml`.
3. Copy the backup config file from the Backup subdirectory to the `Config` directory. Backup files are named using this format: `config-<date>.xml`.
4. Rename the backup config file to `config.xml`.
5. Optionally edit the `config.xml` to opt out of Config Management.
6. Restart the AppDynamics Agent Coordinator service.
7. Restart Windows services, instrumented Windows services, and instrumented standalone applications.

.NET Agent on Windows Logging

Related pages:

- Agent Log Files
- Troubleshoot Agentless Analytics Issues
- Troubleshoot .NET Agent Issues

This page provides an overview of logging for the .NET Agent on Windows.

The .NET Agent hosts log files in the `%ProgramData%\AppDynamics\DotNetAgent\Logs` directory.

To learn how logs are organized into sets that rollover, see Agent Log Files.

Log File Size

Within a set, the first file for the agent log can reach 5 MB. This file never gets deleted because it contains valuable information about the context in which the agent was started.

Each of the remaining logs in the set can reach a maximum of 5 MB.

To allow for several restarts and all possible logs that could be generated (.NET Agent, Business Transaction, and REST), the maximum number of rollover logs is five. The maximum size per set is 5 MB + (5 X 5M B) = 30 MB.

Some log files, such as SamplingInfo and Winston are configured differently and can be 9 MB in size with only two archive files.

Profiler logging is different; a logging framework called Boost logs to the Profiler folder and creates one file per instrumented process. While Boost uses the archive files and max size, it has two other limits: one to limit the number of files in the Profiler folder (defaults to 50) and the other to limit the max size (defaults to 30 MB).

Control the .NET Agent Logging Level

You control the logging level for the .NET Agent by changing the value of the `minlevel` parameter in the `AppDynamicsAgentLog.config` file found in the .NET Agent logs directory.

The log configuration is per server. The .NET Machine Agent and the .NET Agent read the same configuration file and write to the same set of output files.

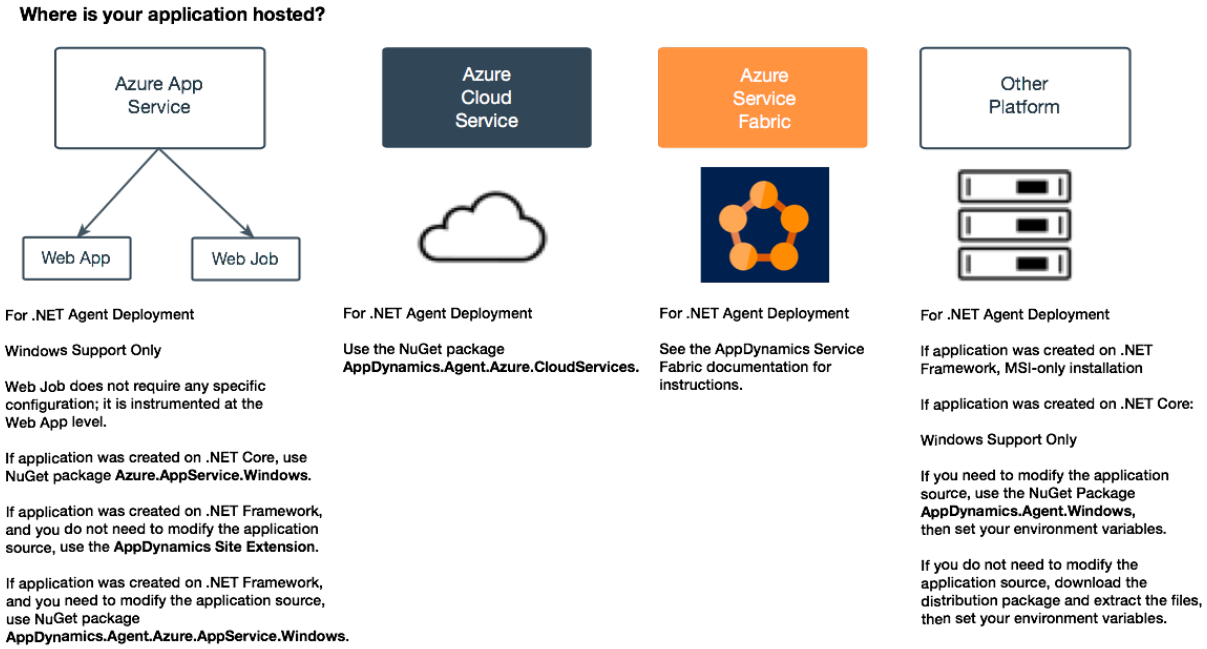
```
<logger name="com.appdynamics.REST.HeartBeatLog" minlevel="Info" writeTo="RESTHearbeat" final="true" />
    <logger name="*" minlevel="Warn" writeTo="warnfile" />
    <logger name="com.appdynamics.BusinessTransactions" minlevel="Info" writeTo="btlog" final="true" />
    <logger name="com.appdynamics.bci.*" minlevel="Info" writeTo="bcifile" final="true" />
    <logger name="com.appdynamics.REST.*" minlevel="Info" writeTo="RESTfile" final="true" />
    <logger name="AppDynamics.Agent.Coordinator.Recovery.*" minlevel="Info" writeTo="CoordinatorRecovery"
final="true" />
    <logger name="com.appdynamics.METRICS.MetricSender" minlevel="Info" writeTo="RESTfile" final="true" />
    <logger name="AppDynamics.Winston.*" minlevel="Info" writeTo="Winston" final="true" />
    <logger name="com.appdynamics.tm.AFastBackendResolver" minlevel="Warn" writeTo="logfile" final="true" />
    <logger name="com.appdynamics.ManagedAgentAPI.DumpStats" minlevel="Trace" writeTo="SamplingInfo" final="
true" />
    <logger name="com.appdynamics.ee.service.analytics.Analytics" minlevel="Info" writeTo="Analytics" final="
true" />
    <logger name="com.appdynamics.profiler.*" minlevel="Info" writeTo="Profiler" />
    <logger name="*" minlevel="Info" writeTo="logfile" />
```

.NET Microservices Agent

To monitor standalone applications on different platforms, install the .NET *microservices agent* on the solution of your choice. Where and how you install the .NET microservices agent depends on your platform—for example, Azure App Service or Service Fabric—and what applications you want to monitor. For example, you can now monitor applications developed using .NET Core for Windows using the .NET microservices agent.

Before You Begin

To accurately determine the steps you need to take to deploy a .NET microservices agent to your solution, review this decision matrix to determine the appropriate path for your platform and applications.



Install the .NET Microservices Agent

Review this table to determine which NuGet package you should use to install the .NET microservices agent. For details about managing NuGet packages, see the documentation for your version of Visual Studio.

`AppDynamics.Agent.Distrib.Micro.Windows`	AppDynamics NuGet package for .NET. This package *should not* be installed directly and is intended for download and file distribution. This package is used for Azure Service Fabric deployments. See Install AppDynamics for Azure Service Fabric.
`AppDynamics.Agent.Windows`	AppDynamics .NET Core microservices agent for Windows. Recommended for standalone installations. See Install the .NET Core Microservices Agent for Windows. This package does not support .NET Framework, only .NET Core for Windows.
`AppDynamics.Agent.Azure.CloudServices`	AppDynamics .NET agent for Azure Cloud Services. See Install AppDynamics for Azure Cloud Services.
`AppDynamics.Agent.Azure.AppService.Windows`	AppDynamics .NET microservices agent for Azure App Service. This package is intended for applications deployed to Azure App Service (Azure Web Apps and Azure API Apps). See Install the AppDynamics .NET Microservices Agent and Install AppDynamics for Azure App Service.

.NET Core Microservices Agent Support

Supported Runtime Environments

This section lists the environments where the .NET Microservices Agent does some automatic discovery after little or no configuration.

The .NET microservices agent works with .NET Core 2.0, 2.1, 3.0, and 3.1 on these operating systems:

- Microsoft Windows Server 2012
- Microsoft Windows Server 2012 R2
- Microsoft Windows Server 2016
- Microsoft Windows Server 2019
- Microsoft Windows 8, 8.1, 10

Microsoft Windows Azure

- Azure App Services for .NET Core 2.0, 2.1, 3.0, and 3.1 environments in the Azure Portal:
  - Web Apps
  - API Apps
  - Container Services

Automatically Discovered Business Transactions

The .NET Microservices Agent discovers business transactions for the following frameworks by default. The agent enables detection without additional configuration.

Type	Custom Configuration Options?	Downstream Correlation?
ASP.NET Core 3.0	Yes	Yes
Web Services including SOAP	No	Yes
Message Queues		
Microsoft Service Bus	No	Yes
RabbitMQ	Yes	Yes

Supported Loggers for the .NET Agent

- Log4Net
- NLog
- System Trace
- Windows Event Log
- Loggers on .NET Core that implement the Microsoft.Extensions.Logging.ILogger API (Linux Agent >= 4.5.19 and Windows Agent >= 21.2.0)

If you are using a different logger, see Error Detection.

Remote Service Detection

The .NET Agent automatically detects the following remote service types. The agent enables detection by default. You don't need to perform extra configuration.

Type	Custom Configuration Options?	Async Detection? *	Downstream Correlation?
HTTP	Yes	See Asynchronous Exit Points for .NET	Yes
MongoDB: C# and .NET MongoDB Driver version 1.10, 2.0	No	See Asynchronous Exit Points for .NET	N/A
Web Services, including SOAP	Yes	See Asynchronous Exit Points for .NET	Yes
Data Integration			
Microsoft BizTalk Server 2010, 2013	No	Yes	See Correlation Over Microsoft BizTalk
Message Queues			
Apache ActiveMQ NMS framework and related MQs	Yes	No	Yes

RabbitMQ	See RabbitMQ Backends for .NET.	No	Yes

* The agent discovers asynchronous transactions for the Microsoft .NET 4.5 framework. See Asynchronous Exit Points for .NET

Data Storage Detection

The .NET Agent automatically detects the following data storage types. The agent enables detection by default. You don't need to perform extra configuration.

Type	Customizable Configuration?	Async Detection?*	AppD for Databases?
ADO.NET (see supported clients below)	Yes	Yes	No

* The agent discovers asynchronous transactions for the Microsoft .NET 4.5 framework. See Asynchronous Exit Points for .NET

Supported ADO.NET Clients

AppDynamics can monitor any ADO.NET client version and type. Clients we have tested include the following:

Database Name	Database Version	Client Type
MySQL	5.x	Connector/Net and ADO.NET

Install the .NET Core Microservices Agent for Windows

To monitor an application executed on Windows, you install the AppDynamics .NET Core microservices agent for Windows to your project. Additionally, you can instrument .NET standalone services or IIS services using the MSI installer. Then, set the environment variables for your Controller.

Install the .NET Core Microservices Agent

Before you install the .NET Core microservices agent, ensure that you have access to the AppDynamics Controller where your application metrics will display.

You can deploy the .NET Core microservices agent using either of these methods:

- Deploy the .NET Core microservices agent during development using IDE and Visual Studio
- Deploy the .NET Core microservices agent separately to different development teams to divide controls, pipelines, and CI/CD processes

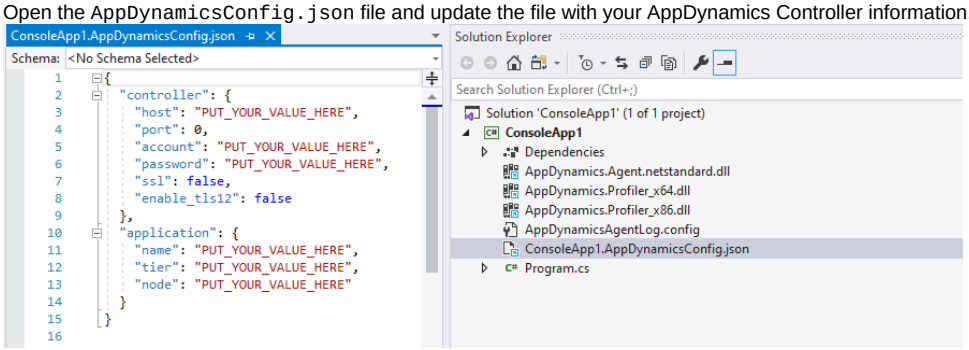
Deploy the .NET Core Microservices Agent During Development

1. Install the `AppDynamics.Agent.Windows` NuGet package in your project. See your IDE documentation.

   When you install the NuGet package, the `AppDynamicsConfig.json` file is copied to the project's root directory. A link file prefixed with the project name is included as content to the project. All the files are copied to the app's output directory after the build.
2. Set up these environment variables for your system application:

   ```
   CORECLR_ENABLE_PROFILING=1
   CORECLR_PROFILER={39AEABC1-56A5-405F-B8E7-C3668490DB4A}
   CORECLR_PROFILER_PATH_32=<actual_path>\AppDynamics.Profiler_x86.dll
   CORECLR_PROFILER_PATH_64=<actual_path>\AppDynamics.Profiler_x64.dll
   ```

   Where `<actual_path>` is the complete path to the `AppDynamics.Profiler` dll.
3. Open the `AppDynamicsConfig.json` file and update the file with your AppDynamics Controller information.



4. Use these options to configure the .NET Agent to connect to the AppDynamics Controller:

   - During development: As shown in the previous screenshot, you can enter your environment variables in the `AppDynamicsConfig.json` file, and save it in source control.
   - During build: Define your msbuild parameters or environment variables that are passed to the `AppDynamicsConfig.json` file at build time. The `AppDynamicsConfig.json` file does not exist at build time, so if you are defining your msbuild parameters or environment variables at build time, you will need to ignore it in source control so that the new `AppDynamicsConfig.json` file can be created.
   - During runtime: Enter your environment variables in Azure.
5. Deploy the application.

Deploy the .NET Core Microservices Agent Separately to Different Development Teams

By using this deployment method, you divide controls, pipelines, and CI/CD processes among different development teams. Additionally, you may encounter a situation where it may not be possible to return to the project solution and add the agent (particularly if the software was developed by a third party).

1. From nuget.org, download the NuGet package AppDynamics.Agent.Distrib.Micro.Windows from https://www.nuget.org/packages/AppDynamics.Agent.Distrib.Micro.Windows/.
2. Extract (unzip) the `dlls` and `AppDynamicsConfig.json` configuration file from the Nuget package to a folder.
3. Update the `AppDynamicsConfig.json` configuration file as required.
   a. Make a copy of the of the `AppDynamicsConfig.json` file and rename it to *<<executable_name>>*`.AppDynamicsConfig.json.` For example, `<<servicefabricapplicationname>>.AppDynamicsConfig.json.`
   b. Copy `AppDynamics.Agent.dll`, `AppDynamics.Profiler_x64.dll`, `AppDynamicsAgentLog.config`, `AppDynamicsConfig.json` from `<<nuget_package>>\content\AppDynamics` and add these files in the solution of each of the service project at the top level, not under any subfolders.
   c. Make a copy of `AppDynamicsConfig.json` and rename it to:`<<executable_name>>.AppDynamicsConfig.json`, for example, `<<servicefabricapplicationname>>.AppDynamicsConfig.json`. Put it into the root of each service project.

   > When you rename the `.json` file to the application name, do not include `.exe` at the end of your application name.

d. Right-click the files in the AppDynamics sub-folder, and then select **Copy Always**.
e. Update `<<executable_name>>.AppDynamicsConfig.json` with your configuration information:

```
{
 "controller":
 {
 "host":"",
 "port":0,
 "account":"",
 "password":""
 },
 "application":
 {
 "name":""
 }
}
```

Do not specify node; it will be assigned automatically. Specifying tier is optional because it could be assigned automatically.

4. Set up these environment variables for your system application:

```
CORECLR_ENABLE_PROFILING=1
CORECLR_PROFILER={39AEABC1-56A5-405F-B8E7-C3668490DB4A}
CORECLR_PROFILER_PATH_32=<actual_path>\AppDynamics.Profiler_x86.dll
CORECLR_PROFILER_PATH_64=<actual_path>\AppDynamics.Profiler_x64.dll
```

Where `<actual_path>` is the complete path to the `AppDynamics.Profiler dll`.

.NET Core for Linux SDK

Related pages:

- .NET Core for Linux SDK Use Cases
- .NET Core for Linux SDK Reference
- .NET Core for Linux SDK Supported Environments
- Overview of Application Monitoring
- Business Transactions

You can use the AppDynamics .NET Core for Linux SDK to monitor the performance of .NET Core applications running on Linux.

The SDK supports these features:

- Business transaction registration and error reporting
- Exit call with correlation and error reporting
- Metrics
- Snapshots with user properties but without callgraphs

Together, these tools provide visibility on application load and response times, and any custom metrics you define.

You monitor .NET Core applications by deploying the .NET Core for Linux SDK to your application directory and project.

Once running, the agent registers business transactions with the AppDynamics Controller. Then, you can view your application flow map and monitor performance in the Controller.

Before You Begin

1. Confirm that you have access to a compatible Controller. See Agent and Controller Compatibility.
2. Confirm the connection settings to the Controller where your agent will report data:
    - If you use a SaaS Controller, AppDynamics sent you the Controller host in your Welcome Email. Use port 443 for HTTPS or port 80 for HTTP.
    - If you use an on-premises Controller, you supplied the host and port during installation.
3. Verify that you have access the machine where the application runs as a user account with privileges to install the agent software, and restart the application.
4. Verify that the machine where the application runs can connect to the Controller. Proxies or firewalls on the network between the agent and Controller may require additional configuration.

Prepare to Install

Ensure you have:

- .NET Core SDK
- .NET Core application running on Linux
- AppDynamics.AgentSDK NuGet package
- Visual Studio or other IDE
- The .NET Core command-line interface (CLI)

Install the SDK

Complete these steps manually, using Visual Studio, or another integrated development environment (IDE).

1. To add the `AppDynamics.AgentSDK` NuGet package in the project, from the command line, enter:

    ```
    dotnet add package AppDynamics.AgentSDK_x64
    ```

    Once you enter the command, there's a compatibility check runs to ensure the package is compatible with the frameworks in the project. If the check passes, then a `<PackageReference>` element is added to the project file, and `dotnet restore` is run implicitly.
2. Modify your project to use the AppDynamics SDK. See .NET Core for Linux SDK Reference.
   Add the SDK namespace and SDK code to the application. For example:

```
using AppDynamics;
...
namespace coremvc.Controllers
{
   public class HomeController : Controller
   {
      public IActionResult SomePage()
      {
          var currCtxId = AgentSDK.StartBusinessTransaction("DemoBT", "ASP_DOTNET", "");
          // Code here calls to other ASP_DOTNET component
                        ...
          AgentSDK.StopBusinessTransaction(currCtxId);
          return View();
       }
       ...
```

3. Run `dotnet build` or `dotnet publish`.
   A warning message displays stating the AppDynamics configuration has not been updated. This is normal.
   - Running `dotnet build` adds this AppDynamics configuration file to your application:
     `<application_directory>: AppDynamicsConfig.json`
   - Running `dotnet publish` renames the AppDynamicsConfig.json file and copies it to the publish directory for the .NET Core for Linux SDK to reference and use:
     `<publish_directory): <project_name>.AppDynamicsConfig.json`
4. Proceed to Using .NET Core for Linux SDK for instructions to update the `AppDynamics.Config.json` file and connect the agent to the Controller.

Known Issues

The requirement to disable loading of native images (NGEN) for self-contained apps compiled with .NET Core 2.0.

Due to a bug in the .NET Core runtime (see **https://github.com/dotnet/coreclr/issues/13021**), the command the .NET Core for Linux SDK profiler sends to the runtime to disable loading of native images is ignored. This prevents AppDynamics from instrumenting any methods in these assemblies which include most of the .NET Standard libraries.

To work around this issue, disable using native images by setting this environment variable:

```
COMPLUS_ReadyToRun=0
```

.NET Core for Linux SDK Supported Environments

You must use the `StartBusinessTransaction` and `CreateExitCall` methods with the entry and exit point types listed below in order for the .NET Core for Linux SDK to recognize them. See .NET Core for Linux SDK Reference.

.NET Core 1.x, 2.0.x, 2.1.x, and 3.1 for Linux SDK Support

Operating Systems

- CentOS 7
- Ubuntu 14.04

Entry Point Types

- ASP.NET
- POCO

Exit Point Types

- HTTP
  - HOST
  - PORT
  - URL
  - QUERY STRING
- WCF
  - URL
  - OPERATION
- ADO.NET
  - HOST
  - PORT
  - DATABASE
  - VENDOR
  - CONNECTION STRING

Using .NET Core for Linux SDK

Before you instrument your applications with the .NET Core for Linux SDK, you must have the current version of the .NET Core for Linux SDK. The page describes how to connect the .NET Core for Linux Agent in your application to the Controller.

Once you have the deployed the SDK, you can configure your application instrumentation. The SDK provides routines to create and manage business transactions, transaction snapshots, backends, exit points and custom metrics collection.

Configuration

1. Edit the `AppDynamicsConfig.json` file with this information:

```
{
  "controller": {
    "host": "<controller_host_name>",
    "port": <controller_port_name>,
    "account": "<controller_account_name>",
    "password": "<controller_account_key>",
    "ssl": <true if using https controller, or false>
    "proxy" : {
        "host": "proxy-host",
        "port": 9090,
           "authentication": {
              "username": "proxy-user",
              "password": "proxy-password"
            }
      }
  },
  "application": {
    "name": "<application_name>",
    "tier": "<tier_name>",
    "node": "<node_name>"
  },
  "log": {
    "directory": "<log_folder_path>",
    "level": "<log_level>"
  }
}
```

2. Add these environment variables to your application, and then rebuild or republish your application.

Environment Variable Name	Value
`CORECLR_PROFILER`	`{57e1aa68-2229-41aa-9931-a6e93bbc64d8}`
`CORECLR_ENABLE_PROFILING`	`1`
`CORECLR_PROFILER_PATH`	Path to the `libappdprofilerdynamic` library. For example, `<application_folder_path> /libappdprofiler.so`

The .NET command that you used to install the SDK determines the location of the `libappdprofiler.so` library. AppDynamics recommends using `dotnet build`, which is the first deployment command option in this table:

Type of Deployment	Command	Work Directory	Profiler Location	Executable Path
Development	`dotnet build, dotnet run`	`<project_path>`	`~/.nuget/packages/appdynamics.agentsdk/4.5.0 /runtimes/linux-x64/native/libappdprofiler.so`	N/A
Framework Dependent	`dotnet publish`	`<project_path>`	`<project_path>/bin/<Flavor>/publish/runtimes /linux-x64/native/libappdprofiler.so`	`bin/<flavor>/publish /linux-x64/<appname.dll>`
Self-contained	`dotnet publish -r linux_x64`	`<project_path>`	`/<project path>/bin/<Flavor>/linux_x64/publish /libappdprofiler.so` `<project path>` is the root directory of the application	`bin/<flavor>/linux_x64 /publish/<appname>` `<flavor>` can be something like `release/netcoreapp2.0`

After you rebuild or republish your application, the agent registers to the Controller, and AppDynamics starts instrumenting your application. See .NET Core for Linux SDK Use Cases.

.NET Core for Linux SDK Use Cases

Related pages:

- .NET Core for Linux SDK
- Overview of Application Monitoring
- Business Transactions

Business Transaction Definition

You define a custom business transaction using the `StartBusinessTransaction` and `StopBusinessTransaction` SDK calls.

A downstream transaction in a distributed transaction cannot start until its upstream transaction has made its exit call.

The order of transactions respects the nonlinear nature of .NET applications. An upstream transaction might end immediately after its exit call completes or sometime later. It is not necessary for an upstream transaction to end before or after its downstream transaction starts or ends. A downstream transaction can end before its upstream transactions have ended.

Sample Business Transaction Creation

This example shows how you use the `StartBusinessTransaction` and `StopBusinessTransaction` SDK calls to define the entry point and exit point of a business transaction.

```
using AppDynamics;
...
namespace coremvc.Controllers
{
   public class HomeController : Controller
   {
      public IActionResult SomePage()
      {
//define start of an ASP_DOTNET type business transaction named DemoBT
          var currCtxId = AgentSDK.StartBusinessTransaction("DemoBT", "ASP_DOTNET", "");
          // Code here calls to other ASP_DOTNET component
                        ...
          AgentSDK.StopBusinessTransaction(currCtxId);
          return View();
       }
       ...
```

Exit Call Management

An exit call is either a custom exit call or an automatically detected exit call. A transaction needs its exit call object to create correlation information to provide to a downstream transaction that needs to correlate with it.

Custom Exit Calls

If you want an exit call to be reported to the Controller, and it is not automatically detected by the agent, create a custom exit call with `CreateExitCall( )` and then define the start and end of the exit call —`startExitCall()` and `endExitCall()`. Use `AddIdentifyingPropertyToExitCall` to help you to identify this exit call in the transaction snapshot.

See .NET Core for Linux SDK Supported Environments for the list of backends that are automatically detected by the .NET Core for Linux SDK. If the exit call invokes a backend not listed on this page, you probably need to create a custom exit call.

Sample Exit Call Creation

```
using AppDynamics;
...
namespace coremvc.Controllers
{
        public class HomeController : Controller
        {
      public IActionResult SomePage()
      {

//define start of an ASP_DOTNET type business transaction called DemoBT. StartBusinessTransaction statement
equals the value of the currCtxId variable
          var currCtxId = AgentSDK.StartBusinessTransaction("DemoBT", "ASP_DOTNET", "");

//define HTTP type exit call named CorrelationExit for the DemoBT business transaction. CreateExitCall
statement equals the value of the exitCall variable
                  var exitCall = AgentSDK.CreateExitCall(currCtxId, "HTTP", "CorrelationExit");

//add identifying properties Host and Port, along with their respective values to the exit call

                AgentSDK.AddIdentifyingPropertyToExitCall(exitCall, "Host", request.RequestUri.Host);
                AgentSDK.AddIdentifyingPropertyToExitCall(exitCall, "Port", request.RequestUri.Port.ToString());

//define the beginning of the exit call
        AgentSDK.StartExitCall(exitCall);

                //code here to make exit call
        ...

//define end of the exit call

         AgentSDK.StopExitCall(exitCall);

                //code here calls to other ASP_DOTNET component
         ...
//define end of the DemoBT business transaction
          AgentSDK.StopBusinessTransaction(currCtxId);
          return View();
       }
       ...
```

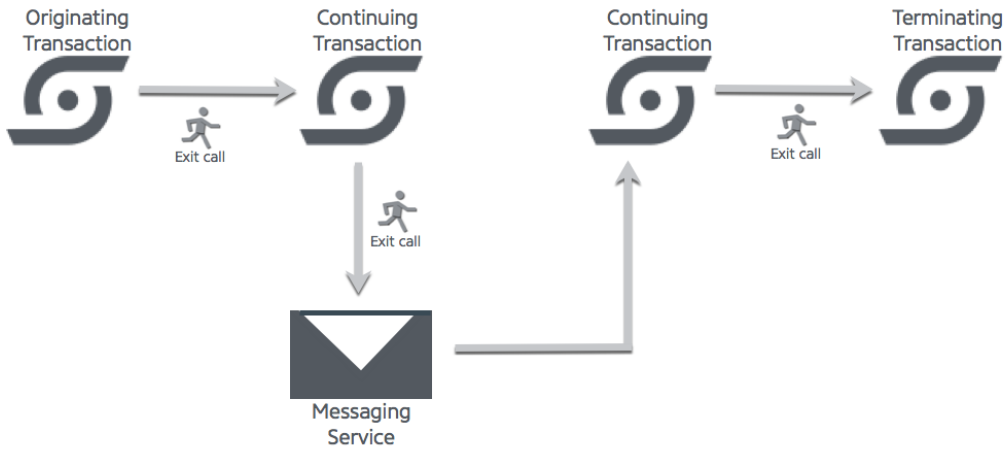
Transaction Correlation Management

Transaction correlation is the functionality that maintains the business transaction context across all tiers (servers) traversed by a distributed transaction. The tiers may be built on platforms other than .NET. For example, Java, Node.js, and PHP tiers may participate in a distributed business transaction in which a .NET tier participates as the originating, continuing or terminating tier in the transaction.

The .NET Core for Linux SDK provides facilities for managing transaction correlation among various transactions, which can be custom or automatically-detected.

While the AppDynamics default auto-detection mechanism obtains transaction correlation information from an incoming HTTP request, the .NET Core for Linux SDK allows you to obtain this information from other sources, such as a custom field in a Redis cache entry, and to supply it to your custom transactions. You can use the Agent SDK classes to support forward correlation by supplying transaction correlation information on exit calls that you make to downstream transactions.

A downstream transaction starts after its immediate upstream transaction makes its exit call. That exit call could be a direct exit call to the next tier or a call to a backend service that does some processing, such as publishing a message onto a queue.



Sample Transaction Correlations Between Upstream and Downstream Tiers

Use Case 1: .NET Core on Linux application calling an upstream .NET Core on Linux application.

Both applications are monitored by AppDynamics.

```
using AppDynamics;
...
namespace coremvc.Controllers
{
public class HomeController : Controller
   {
      public IActionResult SomePage()
      {
        var currCtxId AgentSDK.StartBusinessTransaction("DemoBT", "ASP_DOTNET", "");
            var client = new HttpClient();
            var request = new HttpRequestMessage()
            {
                        RequestUri = new Uri("http://localhost:8080 "),
                        Method = HttpMethod.Get,
                };

            var exitCall = AgentSDK.CreateExitCall(currCtxId, "HTTP", "CorrelationExit");
            AgentSDK.AddIdentifyingPropertyToExitCall(exitCall, "Host",   request.RequestUri.Host);
            AgentSDK.StartExitCall(exitCall);
            var correlationHeader = AgentSDK.GetCorrelationHeader(exitCall);
            request.Headers.Add("singularityheader",correlationHeader);
            await client.SendAsync(request);
            AgentSDK.StopExitCall(exitCall);
            AgentSDK.StopBusinessTransaction(currCtxId);       }
       ...
```

Use Case 2: .NET Core on Linux application calling a downstream .NET Core on Linux application.

Both applications are monitored by AppDynamics.

```
using AppDynamics;
...
namespace coremvc.Controllers
{
public class HomeController : Controller
   {
      public IActionResult SomePage()
      {

bool hasCorrelationHeader = Request.Headers.TryGetValue("singularityheader", out var correlationHeader);

var currCtxId = AgentSDK.StartBusinessTransaction("DemoBT", "ASP_DOTNET",

hasCorrelationHeader ? (string)correlationHeader:"");

          // Code here calls to other ASP_DOTNET component

          AgentSDK.StopBusinessTransaction(currCtxId);

          return View();
       }
     ...
```

Use Case 3: Non .NET Core on Linux application monitored by AppDynamics calling a .NET Core on Linux application.

To correlate, the .NET Core on Linux application must include this code:

```
using AppDynamics;
...
namespace coremvc.Controllers
{
public class HomeController : Controller
   {
      public IActionResult SomePage()
      {

//Obtain the HTTP header value of the page via the TryGetValue method and use that value to define
correleationHeader
        bool hasCorrelationHeader = Request.Headers.TryGetValue("singularityheader", out var correlationHeader);

//Define the start of the ASP_DOTNET type business transaction DemoBT to include the correleation header if
available.
        var currCtxId = AgentSDK.StartBusinessTransaction("DemoBT", "ASP_DOTNET",
        hasCorrelationHeader ? (string)correlationHeader:"");


          // Code here calls to other ASP_DOTNET component
                       ...

//Define the end of the DemoBT business transaction
          AgentSDK.StopBusinessTransaction(currCtxId);
          return View();
       }
       ...
```

Use Case 4: .NET Core on Linux application calling a non .NET Core on Linux application monitored by AppDynamics.

To correlate, the .NET Core on Linux application must include this code:

```
using AppDynamics;
...
namespace coremvc.Controllers
{
public class HomeController : Controller
   {
      public IActionResult SomePage()
      {
           var currCtxId AgentSDK.StartBusinessTransaction("DemoBT", "ASP_DOTNET", "");

            var client = new HttpClient();

            var request = new HttpRequestMessage()
            {
                        RequestUri = new Uri("http://localhost:8080 "),
                        Method = HttpMethod.Get,
                };

            var exitCall = AgentSDK.CreateExitCall(currCtxId, "HTTP", "CorrelationExit");
            AgentSDK.AddIdentifyingPropertyToExitCall(exitCall, "Host",   request.RequestUri.Host);
            AgentSDK.StartExitCall(exitCall);
            var correlationHeader = AgentSDK.GetCorrelationHeader(exitCall);
            request.Headers.Add("singularityheader",correlationHeader);
            await client.SendAsync(request);
            AgentSDK.StopExitCall(exitCall);
            AgentSDK.StopBusinessTransaction(currCtxId);       }
       ...
```

.NET Core for Linux SDK Reference

Related pages:

- .NET Core for Linux SDK
- Overview of Application Monitoring
- Business Transactions

To instrument .NET Core for Linux applications, you use the .NET Core for Linux SDK. Once you have deployed the .NET Core for Linux SDK into your application, AppDynamics automatically instruments the application. You can customize the application instrumentation using these business transaction and exit call methods.

You must start all transactions using the `StartBusinessTransaction` method in the .NET Core for Linux SDK.

This page describes the methods defined in the SDK.

Using the SDK

Namespace: `AppDynamics.AgentSDK`
Assembly: `AppDynamics.Agent.SDK.dll`

Business Transaction Methods

StartBusinessTransaction

Starts the Business Transaction.

Format: `StartBusinessTransaction(<name>, <entryPointType>, <correlationHeader>)`

Parameters:

- `name` – Business Transaction name.
- `entryPointType` – Type of entry point, such as ASP_DOTNET and POCO
- `correlationHeader` – AppDynamics correlation header. Optional.

Return Type: `AppDynamics.BusinessTransaction`

StopBusinessTransaction

Stop the Business Transaction

Format: `AgentSDK.StopBusinessTransaction(<bt>)`

Parameters: `bt` – Business transaction object returned by the StartBusinessTransaction method.

Return Type: void

AddDataToTransactionSnapshot

Add identifying information to the transaction snapshot.

Format: `AgentSDK.AddDataToTransactionSnapshot(<bt>, <key>, <value>)`

Parameters:

- `bt` – The business transaction for which AppDynamics creates a snapshot. Returned by the StartBusinessTransaction method.
- `key` – Key in key/value pair
- `value` – Value in key/value pair

Return Type: void

Exit Call Methods

CreateExitCall

Create an Exit Call

Format: `AgentSDK.CreateExitCall(<bt>, <exitPointType>, <name>, <isAsync>)`

Parameters:

- `bt` – BusinessTransaction on which to create an exit call.
- `exitPointType` – Type of exit call, such as HTTP, WCF, and ADO.NET
- `name` – Name of exit call.
- `isAsync` – Boolean value to identify exit call as asynchronous. Optional. Default value: false

Return Type: `AppDynamics.ExitCall`

AddIdentifyingPropertyToExitCall

Add properties to the exit call.

Format: `AgentSDK. AddIdentifyingPropertyToExitCall(ExitCall <exitCall>, <string_property>, <string_value>)`

Parameters:

- `exitCall` – Business transaction for which AppDynamics creates a the snapshot. Returned by StartBusinessTransaction.
- `string_property` – Name of the property. For example, the URL.
- `string_value` – Value of the property. For example, http://www.google.com.

Return Type: void

StartExitCall

Start an exit call to the specified backend as part of a business transaction

`Format: AgentSDK.StartExitCall(ExitCall <exitCall>)`

Parameters: `exitCall` – ExitCall type returned by CreateExitCall.

Return Type: void

StopExitCall

Complete the exit call

Format: `AgentSDK.StopExitCall(ExitCall <exitCall>)`

Parameters: `exitCall` – ExitCall type returned by CreateExitCall.

Return Type: void

AddErrorToTransaction

Register an error for the business transaction

Format: `AgentSDK. AddErrorToTransaction (<bt>, <name>, <message>, <errorCode>, <markBtAsError>)`

Parameters

- `bt` – BusinessTransaction for which AppDynamics creates a snapshot. Returned by StartBusinessTransaction.
- `name` – Name of Error.
- `message` – Error message.
- `errorCode` – Integer value of the error.
- `markBtAsError` – Boolean value to qualify the BusinessTransaction as an error. (optional. Default Value: false)

Return Type: void

AddErrorToExitCall

Register an error for the exit call

Format: `AgentSDK.AddErrorToExitCall(ExitCall <exitCall>, string <errorName>, string <errorMessage>, bool <markBtAsError>)`

Parameters

- `exitCall` – ExitCall on which the error should be registered.
- `errorName` – Name of Error
- `errorMessage` – Error message
- `markBtAsError` – Boolean value to qualify the ExitCall as an error.

Return Type: void

GetCorrelationHeader

Sets the AppDynamics correlation header at the upstream tier. The downstream tiers read this value to continue correlation. Example available below.

Format: `AgentSDK.GetCorrelationHeader(<exitCall>)`

Parameters: `exitCall` – Exit Call created for business transaction to downstream tier monitored by AppDynamics.

Return Type: string

.NET Agent for Linux

Related pages:

- .NET Core for Linux SDK
- Create an ASP.NET Core web app in Azure
- Overview of Application Monitoring
- Business Transactions

You use the AppDynamics .NET Agent for Linux to monitor the performance of .NET Core applications running on Linux OS.

The agent supports these features:

- ASP.NET MVC transactions. See Name MVC Transactions by Area, Controller, and Action.
- ASP.NET WebAPI transactions
- ASP.NET Razor Pages transactions
- HttpClient backend calls
- ADO.NET backend calls such as SqlClient, MySql, SQLite, and PostgreSQL
- EntityFramework
- Upstream and downstream correlation using HTTP
- Snapshot waterfall view with async support
- Redis exit calls
- Detects and monitors calls to an Oracle database

You can run the AppDynamics .NET Agent for Linux on microservice platforms, such as those based on Docker containers and Pivotal Cloud Foundry (PCF). See AppDynamics Pivotal Cloud Foundry and .NET Agent for Linux Advanced Configuration Options.

The .Net Agent for Linux is supported on AppDynamics Controller >= 4.4.1. See .NET Agent for Linux Supported Environments.

Install the .NET Agent for Linux

Related pages:

- .NET Agent for Linux
- Create an ASP.NET Core web app in Azure
- Overview of Application Monitoring
- Business Transactions

Getting Started

Before you begin, review these prerequisites:

1. Confirm you have access to a compatible Controller. See Agent and Controller Compatibility.
2. Confirm the connection settings to the Controller where your agent will report data:
    - If you use a SaaS Controller, AppDynamics sent you the Controller host in your Welcome Email. Use port 443 for HTTPS or port 80 for HTTP.
    - If you use an on-premises Controller, you supplied the host and port during installation.
3. Verify you have access to the machine where the application runs as a user account with privileges to install the agent software and restart the application.
4. Verify that the machine where the application runs can connect to the Controller. Proxies or firewalls on the network between the agent and Controller may require additional configuration.
5. Verify that the application environment meets the requirements on .NET Agent for Linux Supported Environments prior to running the agent.

Applications and Licenses Overview

Ensure you have these applications and licenses:

- .NET Core application running on Linux
- .NET license for each .NET Core application running on Linux. See License Management.

Binaries Overview

Download the agent binaries from the AppDynamics Download Site and then extract them from the zip file into the desired folder.

The archive contains these files:

- `AppDynamics.Agent.netstandard.dll`
- `libappdprofiler.so`
- `libappdprofiler_glibc.so`
- `libappdprofiler_musl.so`
- `README.md`

> You must extract all four of the agent binaries from the zip file.

Deployment

There are two options for starting your deployment using Docker:

- Create Your Own Image, start your agent from scratch
- Run an Existing Image, if you already have the .NET Agent enabled

Create Your Own Image

You can use the ASP.NET sample application from Microsoft and a Dockerfile to build your own image and get started:

1. Download the binaries.
2. Update the Dockerfile variables to configure the connection to the Controller and your application identity in AppDynamics.
3. Create the Docker image.

This is an example Dockerfile with commented instructions:

```
FROM mcr.microsoft.com/dotnet/core/samples:aspnetapp

####### Requirements
# Have the following files alongside the Dockerfile:
# * libappdprofiler.so
# * AppDynamics.Agent.netstandard.dll

####### Instructions
# Building image: docker build --rm -t appdynamicstest:latest .
# Running container: docker run --rm -p 8000:80 appdynamicstest:latest
# Open the application using http://localhost:8000/

# Copy agent binaries to the image from current folder
RUN mkdir -p /opt/appdynamics/dotnet
ADD libappdprofiler.so /opt/appdynamics/dotnet/
ADD libappdprofiler_glibc.so /opt/appdynamics/dotnet/
ADD libappdprofiler_musl.so /opt/appdynamics/dotnet/
ADD AppDynamics.Agent.netstandard.dll /opt/appdynamics/dotnet/

# Mandatory settings required to attach the agent to the .NET application
ENV CORECLR_PROFILER={57e1aa68-2229-41aa-9931-a6e93bbc64d8} \
    CORECLR_ENABLE_PROFILING=1 \
    CORECLR_PROFILER_PATH=/opt/appdynamics/dotnet/libappdprofiler.so

# Configure connection to the controller
ENV APPDYNAMICS_CONTROLLER_HOST_NAME=controller.saas.appdynamics.com
ENV APPDYNAMICS_CONTROLLER_PORT=443
ENV APPDYNAMICS_CONTROLLER_SSL_ENABLED=true
ENV APPDYNAMICS_AGENT_ACCOUNT_NAME=account-name
ENV APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY=access-key

# Configure application identity in AppDynamics
ENV APPDYNAMICS_AGENT_APPLICATION_NAME="My Application"
ENV APPDYNAMICS_AGENT_TIER_NAME="Sample Tier"
ENV APPDYNAMICS_AGENT_REUSE_NODE_NAME=true
ENV APPDYNAMICS_AGENT_REUSE_NODE_NAME_PREFIX="Instance"

# It is possible to configure .NET agent using AppDynamicsConfig.json configuration file instead of environment
variables
# ADD AppDynamicsConfig.json /opt/appdynamics/dotnet/
```

Configuration with AppDynamicsConfig.json: If you are configuring your agent using environment variables with the json configuration file or only with the json configuration file, include the FULL_AGENT feature in the json file. Otherwise, the agent will not work. See the AppDynamicsConfig.json file example on .NET Agent for Linux Advanced Configuration Options.

```
,
  "feature": [
    "FULL_AGENT"
  ]
```

FULL_AGENT feature mode is automatically enabled when configuring the agent purely by environment variables.

On Alpine OS

Applications running on Alpine require an additional mandatory environment variable, `LD_LIBRARY_PATH`, such as `ENV LD_LIBRARY_PATH=/opt/appdynamics/dotnet`.

Run an Existing Image

You can run an existing image with an agent enabled. This is the simplest way to start because it does not require building a new image:

1. Download the aforementioned binaries.
2. Update the Docker command variables to configure the connection to the Controller and your application identity in AppDynamics.
3. Run the command in the desired folder where you extracted the agent binaries from the zip file.
   You can replace `$(pwd)` in the command with this folder.

This example runs the sample application. It assumes you have AppDynamics binaries in the current directory:

Example command to run existing image with .NET agent enabled

```
docker run \
-p 8000:80 \
-e CORECLR_PROFILER={57e1aa68-2229-41aa-9931-a6e93bbc64d8} \
-e CORECLR_ENABLE_PROFILING=1 \
-e CORECLR_PROFILER_PATH=/opt/appdynamics/dotnet/libappdprofiler.so \
-e APPDYNAMICS_CONTROLLER_HOST_NAME=controller.saas.appdynamics.com \
-e APPDYNAMICS_CONTROLLER_PORT=443 \
-e APPDYNAMICS_CONTROLLER_SSL_ENABLED=true \
-e APPDYNAMICS_AGENT_ACCOUNT_NAME=account-name \
-e APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY=access-key \
-e APPDYNAMICS_AGENT_APPLICATION_NAME="My Application" \
-e APPDYNAMICS_AGENT_TIER_NAME="Sample Tier" \
-e APPDYNAMICS_AGENT_REUSE_NODE_NAME=true \
-e APPDYNAMICS_AGENT_REUSE_NODE_NAME_PREFIX="Instance" \
-v "$(pwd)":/opt/appdynamics/dotnet/ \
mcr.microsoft.com/dotnet/core/samples:aspnetapp
```


On Alpine OS

Applications running on Alpine require an additional mandatory environment variable, `LD_LIBRARY_PATH`, such as `-e LD_LIBRARY_PATH=/opt/appdynamics/dotnet \`

The command mounts agent files as a volume and sets environment variables required for the agent to attach.

Startup Flow

During the application startup, the AppDynamics agent writes messages to the console as well as the application and .NET framework.

This .NET Agent startup console output indicates proper agent initialization:

```
appd.agent.profiler(Info): ...
```

See .NET Agent for Linux Troubleshooting.

.NET Agent for Linux Container Installation

You can install the .NET Agent for Linux in container environments, such as Docker and Kubernetes.

Overview of Container Installation

To learn more about installation AppDynamics agents in containers, see:

- Container Installation Options
- Configuring Agents in Kubernetes

Install the .NET Agent in Containers

To install the .NET Agent in containers, see:

- Prepare to Install the .NET Agent in Containers
- Install the .NET Agent in Containers

Prepare to Install the .NET Agent for Linux in Containers

Related pages:

- .NET Agent for Linux
- Install the .NET Agent for Linux in Containers
- Create an ASP.NET Core web app in Azure
- Overview of Application Monitoring
- Business Transactions

Before you install the .NET Agent for Linux in a container environment, make sure you meet the requirements and complete the steps listed on this page.

General Requirements

Verify that:

- The machine where the application runs can connect to the Controller. Proxies or firewalls on the network between the agent and Controller may require additional configuration.
- The application environment meets the requirements on .NET Agent for Linux Supported Environments prior to running the agent.

Ensure you have these applications and licenses:

- .NET Core application running on Linux
- .NET license for each .NET Core application running on Linux. See License Management in the Controller.

Controller Requirements

Confirm you meet the following Controller requirements:

- Access to a compatible Controller. See Agent and Controller Compatibility for details.
- Confirm the connection settings to the Controller where your agent reports data:
    - SaaS Controllers: AppDynamics sent you the Controller host in your Welcome Email. Use port 443 for HTTPS or port 80 for HTTP.
    - On-premises Controller: you supplied the host and port at install time.

.NET Agent Environment Variables

The `FULL_AGENT` environment variable is required to instrument the agent in a container environment. This variable is automatically enabled when configuring the .NET Agent by environment variables only.

However, if you configure your agent using environment variables along with the JSON configuration file, `AppDynamicsConfig.json`, or just with the JSON configuration file, you need to include the `FULL_AGENT` feature in the JSON file. For example:

```
{
  "feature": [
    "FULL_AGENT"
  ]
}
```

See the `AppDynamicsConfig.json` file example on .NET Agent for Linux Advanced Configuration Options for details.

Applications running in an Alpine Linux container require an additional mandatory environment variable, `LD_LIBRARY_PATH`, which can be defined in the Dockerfile as follows:

```
ENV LD_LIBRARY_PATH=/opt/appdynamics/dotnet
```

Install the .NET Agent for Linux in Containers

This page explains how to install the .NET Agent for Linux in a Docker or Kubernetes containerized environment. There are three ways to install the .NET Agent for Linux with containers:

- Use Auto-Instrumentation
- Use Init Containers
- Use a Dockerfile

Use Auto-Instrumentation

> This scenario applies to containers running in Kubernetes where the Cluster Agent is installed.

This option uses the Auto-Instrumentation feature of the Cluster Agent. It is the recommended option because it offers the highest level of automation and the simplest operational experience for instrumenting .NET applications in a Kubernetes cluster.

To get started, see Auto-Instrument Applications with the Cluster Agent.

Use Init Containers

> Init container installation requires a Kubernetes environment.

You can use Kubernetes init containers to instrument the .NET Agent. In this method, the init container copies the agent binaries into the application container at runtime. The deployment spec used by the application references two containers:

- Application container based on an image that does not contain any .NET Agent binaries
- Second init container based on an image that only contains the .NET Agent binaries.

The deployment spec is updated to reference these two containers and copy the agent binaries from the init container to the application container at deployment time. Once the copy is performed, the init container terminates.

To use init containers to copy the .NET Agent for Linux binaries, perform these steps:

1. Build the .NET Core Application Image
2. Build the .NET Agent for Linux Init Container Image
3. Add the Init Container to the Deployment Spec
4. Set the .NET Agent for Linux Environment Variables
5. Set the APPDYNAMICS_AGENT_UNIQUE_HOST_ID Environment Variable
6. Copy the AppDynamicsConfig.json File to the Container
7. (On-Premises Controller Only) Copy the Controller Certs to the Container
8. Example Configuration for Using an Init Container

Build the .NET Core Application Image

Build a .NET Core application image. Do not include the .NET Agent for Linux binaries.

Build the .NET Agent for Linux Init Container Image

Build the .NET Agent for Linux image separately from the application image. You can reuse this image across multiple .NET Core application deployments.

Alternatively, the init container can reference a pre-built image from AppDynamics on Docker Hub.

Add the Init Container to the Deployment Spec

Edit the deployment spec to add the required sections that allow you to copy the agent binaries from the init container to the application image.

The following snippet from a deployment spec shows the required `volumes`, `volumeMounts`, and `initContainer` definitions. The code example assumes the .NET Core application image is published to `dotnet-samples:aspnetapp` and the init container image uses the pre-built image `docker.io/appdynamics/dotnet-core-agent:<version>` (where `<version>` is the .NET Agent version; for example, 21.5.0):

```
kind: Deployment
spec:
  containers:
  - name: dotnet-app
    image: microsoft/dotnet-samples:aspnetapp
    volumeMounts:
    - mountPath: /opt/appdynamics
      name: appd-agent-repo
  initContainers:
  - name: appd-agent
        image: docker.io/appdynamics/dotnet-core-agent:<version>
    volumeMounts:
    - mountPath: /appdynamics
      name: appd-agent-repo
  volumes:
    - name: appd-agent-repo
      emptyDir: {}
```

Set the .NET Agent for Linux Environment Variables

To set all of the required .NET Agent environment variables, you must follow the steps in Best Practices to Configure Agents in Kubernetes listed below:

1. Use ConfigMaps to Configure the App Server Agent
2. Use Secrets for the Controller Access Key
3. Set Application-Specific Configuration in the Deployment Spec

Use ConfigMaps to Configure the App Server Agent

1. Use a ConfigMap to set the .NET Agent for Linux environment variables that are shared across applications in a namespace:

   ```
   apiVersion: v1
   data:
     CORECLR_PROFILER: "{57e1aa68-2229-41aa-9931-a6e93bbc64d8}"
     CORECLR_ENABLE_PROFILING: "1"
     CORECLR_PROFILER_PATH: "/opt/appdynamics/libappdprofiler.so"
     APPDYNAMICS_AGENT_APPLICATION_NAME: "<value>"
     APPDYNAMICS_AGENT_ACCOUNT_NAME: "<value>"
     APPDYNAMICS_CONTROLLER_HOST_NAME: "<value>"
     APPDYNAMICS_CONTROLLER_PORT: "<value>"
     APPDYNAMICS_CONTROLLER_SSL_ENABLED: "<value>"
     APPDYNAMICS_AGENT_REUSE_NODE_NAME: "true"
     APPDYNAMICS_AGENT_REUSE_NODE_NAME_PREFIX: "<value>"
     # variables required to send transaction analytics data
     APPDYNAMICS_ANALYTICS_HOST_NAME: "<value>"
     APPDYNAMICS_ANALYTICS_PORT: "<value>"
     APPDYNAMICS_ANALYTICS_SSL_ENABLED: "<value>"
   kind: ConfigMap
   metadata:
     name: appd-dotnet-config
   ```

   Note that the analytics host, port and ssl settings depend on how the Analytics Agent is deployed. See Deploy Analytics in Kubernetes for options.

2. (Optional) If you are running your .NET Core application in an Alpine Linux container, set `LD_LIBRARY_PATH` in the ConfigMap:

   ```
   apiVersion: v1
   data:
     CORECLR_PROFILER: "{57e1aa68-2229-41aa-9931-a6e93bbc64d8}"
     CORECLR_ENABLE_PROFILING: "1"
     CORECLR_PROFILER_PATH: "/opt/appdynamics/libappdprofiler.so"
     LD_LIBRARY_PATH: "/opt/appdynamics"
     ...
   kind: ConfigMap
   metadata:
     name: appd-dotnet-config
   ```

3. Apply the ConfigMap to the namespace:

```
kubectl -n dotnetapp apply -f appd-dotnet-config.yaml
```

4. Update the deployment spec to reference the `ConfigMap`:

```
spec:
  containers:
  - name: dotnet-app
    envFrom:
    - configMapRef:
      name: appd-dotnet-config
 ...
```

Use Secrets for the Controller Access Key

1. Create a Secret using `kubectl`:

```
kubectl -n dotnetapp create secret generic appd-agent-secret --from-literal=access-key=<access-key>
```

2. Update the deployment spec to reference the Secret:

```
spec:
  containers:
  - name: dotnet-app
    env:
    - name: APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY
      valueFrom:
      secretKeyRef:
        name: appd-agent-secret
        key: access-key
 ...
```

Set Application-Specific Configuration in the Deployment Spec

1. Set the application-specific tier name environment variable `APPDYNAMICS_AGENT_TIER_NAME` in the deployment spec:

```
spec:
  containers:
  - name: dotnet-app
    env:
    - name: APPDYNAMICS_AGENT_TIER_NAME
      value: dotnet-service
```

Set the `APPDYNAMICS_AGENT_UNIQUE_HOST_ID` Environment Variable

The `APPDYNAMICS_AGENT_UNIQUE_HOST_ID` environment variable is supported in version 20.7.0+ of the .NET Agent for Linux. For earlier versions, a property must be set in `AppDynamicsConfig.json` based on a runtime value. See this example deployment spec.

Set the `APPDYNAMICS_AGENT_UNIQUE_HOST_ID` environment variable to enable APM correlation with the Cluster Agent. Since the value depends on a runtime value, set this environment variable in the container startup command using the values documented in Configure App Agents to Correlate with Cluster Agent. For example, for a Kubernetes environment with the Docker runtime, set the environment variable as shown (`export` is required):

```
kind: Deployment
spec:
  containers:
    image: microsoft/dotnet-samples:aspnetapp
    command: ["/bin/sh"]
    args: ["-c", "export APPDYNAMICS_AGENT_UNIQUE_HOST_ID=$(sed -rn '1s#.*/##; 1s/(.{12}).*/\\1/p' /proc/self
/cgroup) && dotnet aspnetapp.dll"]
...
```

Copy the AppDynamicsConfig.json File to the Container

Some .NET Agent for Linux options must be set in the `AppDynamicsConfig.json` file. This includes setting the `outputtype` to `console`, which facilitates sending the agent logs to log aggregation tools and viewing the logs using `kubectl logs`. When `AppDynamicsConfig.json` is required, the `FULL_AGENT` feature must be set to support the use of environment variables and the configuration file. See .NET Agent for Linux Advanced Configuration Options.

1. Create the `AppDynamicsConfig.json` file with the required configuration. For example:

   ```
   {
       "feature": [
           "FULL_AGENT"
       ],
       "log": [
           {
           "outputtype": "console"
           }
       ]
   }
   ```

2. Create a ConfigMap based on the contents of `AppDynamicsConfig.json`:

   ```
   $ kubectl -n dotnetapp create configmap appd-config --from-file=AppDynamicsConfig.json
   ```

3. Update the deployment spec to add the `AppDynamicsConfig.json` file to the container file system. Use `volumes` and `volumeMounts` that reference the ConfigMap contents:

   ```
   kind: Deployment
   spec:
     containers:
       image: microsoft/dotnet-samples:aspnetapp
       command: ["/bin/sh"]
       args: ["-c", "export APPDYNAMICS_AGENT_UNIQUE_HOST_ID=$(sed -rn '1s#.*/##; 1s/(.{12}).*/\\1/p' /proc
   /self/cgroup) && dotnet aspnetapp.dll"]
       volumeMounts:
       - name: appd-config
         subPath: AppDynamicsConfig.json
         mountPath: /opt/appdynamics/AppDynamicsConfig.json
     volumes:
       - name: appd-config
         configMap:
           name: appd-config
   ```

(On-Premises Controller Only) Copy the Controller Certs to the Container

If the .NET Agent for Linux communicates with an on-premises Controller, the Controller certs must be copied to the container. See Controller SSL and Certificates.

1. Define a ConfigMap to reference the cert file. Use a volume mount in the deployment spec to mount the ConfigMap contents to the container:

   ```
   $ kubectl create configmap appd-cert --from-file=cacerts
   ```

2. Add the cert file to the container file system using `volumes` and `volumeMounts` as shown in the deployment spec snippet:

```
kind: Deployment
spec:
  containers:
    image: microsoft/dotnet-samples:aspnetapp
    volumeMounts:
    - name: appd-cert
      subPath: cacerts
      mountPath: /opt/appdynamics/cacerts
  volumes:
    - name: appd-cert
      configMap:
        name: appd-cert
```

3. Set the `APPDYNAMICS_CONTROLLER_SSL_ENABLED` and `APPDYNAMICS_CONTROLLER_SSL_CERTFILE` environment variables in the ConfigMap. See .NET Agent for Linux Environment Variables.

```
apiVersion: v1
data:
  CORECLR_PROFILER: "{57e1aa68-2229-41aa-9931-a6e93bbc64d8}"
  CORECLR_ENABLE_PROFILING: "1"
  CORECLR_PROFILER_PATH: "/opt/appdynamics/libappdprofiler.so"
  APPDYNAMICS_CONTROLLER_SSL_ENABLED: true
  APPDYNAMICS_CONTROLLER_SSL_CERTFILE: /opt/appdynamics/cacerts
  ...
kind: ConfigMap
metadata:
  name: appd-dotnet-config
```

Example Configuration for Using an Init Container

This Dockerfile is an example of building the .NET Agent for Linux init container image using a multi-stage build. A complete example of a deployment spec that uses an init container to copy the agent binaries can be found on Github: dotnet-app.yaml.

Use a Dockerfile

This scenario applies to containers running in Docker and Kubernetes.

You can use a Dockerfile to copy the .NET Agent for Linux to the Docker image at build time. To use the option, create a single image that contains both the application and .NET Agent for Linux binaries.

To copy the agent into the application image during the Docker image build:

1. Download and Unzip the .NET Agent for Linux
2. Copy the Agent Binaries to the Image
3. Set the .NET Agent for Linux Environment Variables
4. Set the APPDYNAMICS_AGENT_UNIQUE_HOST_ID Environment Variable
5. Copy the AppDynamicsConfig.json File to the Image
6. (On-Premises Controller only) Copy the Controller Certs to the Image
7. Example Configuration for Using a Dockerfile

Download and Unzip the .NET Agent for Linux

Download the .NET Agent for Linux programmatically or from the downloads portal. The agent is packaged as a zip file. In a shell, unzip it to the `AppDynamics-DotNetCore-linux-x64` folder:

```
$ unzip AppDynamics-DotNetCore-linux-x64-<version>.zip -d AppDynamics-DotNetCore-linux-x64
```

Copy the Agent Binaries to the Image

Edit the Dockerfile to copy the unpackaged agent binaries to the target folder:

```
COPY AppDynamics-DotNetCore-linux-x64/ /opt/appdynamics/
```

Set the .NET Agent for Linux Environment Variables

If your application runs in a non-Kubernetes environment (for example, using `dockerrun`), set the agent environment variables in the Dockerfile. For example:

```
ENV CORECLR_PROFILER="{57e1aa68-2229-41aa-9931-a6e93bbc64d8}"
ENV CORECLR_ENABLE_PROFILING=1
ENV CORECLR_PROFILER_PATH="/opt/appdynamics/libappdprofiler.so"
ENV APPDYNAMICS_AGENT_APPLICATION_NAME=<value>
ENV APPDYNAMICS_AGENT_TIER_NAME=<value>
ENV APPDYNAMICS_AGENT_ACCOUNT_NAME=<value>
ENV APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY=<value>
ENV APPDYNAMICS_CONTROLLER_HOST_NAME=<value>
ENV APPDYNAMICS_CONTROLLER_PORT=<value>
ENV APPDYNAMICS_CONTROLLER_SSL_ENABLED=<value>
ENV APPDYNAMICS_AGENT_REUSE_NODE_NAME=true
ENV APPDYNAMICS_AGENT_REUSE_NODE_NAME_PREFIX=<value>
# variables required to send transaction analytics data
ENV APPDYNAMICS_ANALYTICS_HOST_NAME=<value>
ENV APPDYNAMICS_ANALYTICS_PORT=<value>
ENV APPDYNAMICS_ANALYTICS_SSL_ENABLED=<value>
```

For Kubernetes applications, omit these environment variables from the Dockerfile and set them using ConfigMaps and Secrets.

The reuse node name and prefix environment variables are required to support unique node naming for multiple container instances for the same application image. See .NET Agent for Linux Environment Variables. The analytics host, port and ssl settings depend on how the Analytics Agent is deployed. See Deploy Analytics in Kubernetes for options.

If you are running your .NET Core application in an Alpine Linux container, set `LD_LIBRARY_PATH` to the location of the agent binaries. For a Docker application, set `LD_LIBRARY_PATH` in the Dockerfile:

```
ENV CORECLR_PROFILER="{57e1aa68-2229-41aa-9931-a6e93bbc64d8}"
ENV CORECLR_ENABLE_PROFILING=1
ENV CORECLR_PROFILER_PATH="/opt/appdynamics/libappdprofiler.so"
ENV LD_LIBRARY_PATH="/opt/appdynamics"
...
```

For a Kubernetes application, set LD_LIBRARY_PATH in the ConfigMap.

Set the `APPDYNAMICS_AGENT_UNIQUE_HOST_ID` Environment Variable

> The `APPDYNAMICS_AGENT_UNIQUE_HOST_ID` environment variable is supported in version 20.7.0+ of the .NET Agent for Linux. For earlier version, a property must be set in `AppDynamicsConfig.json` based on a runtime value. This must be performed in the container startup script.

For Kubernetes applications, set the `APPDYNAMICS_AGENT_UNIQUE_HOST_ID` environment variable to enable APM correlation with the Cluster Agent. Since the value depends on a runtime value, set this environment variable in the container startup command using the values documented in Manually Configure App Agents to Correlate with the Cluster Agent. For example, for a Kubernetes environment with a Docker runtime, set the environment variable as shown (`export` is required):

```
kind: Deployment
spec:
  containers:
    image: microsoft/dotnet-samples:aspnetapp
    command: ["/bin/sh"]
    args: ["-c", "export APPDYNAMICS_AGENT_UNIQUE_HOST_ID=$(sed -rn '1s#.*/##; 1s/(.{12}).*/\\1/p' /proc/self
/cgroup) && dotnet aspnetapp.dll"]
...
```

Copy the AppDynamicsConfig.json File to the Image

Some .NET Agent for Linux options must be set in the `AppDynamicsConfig.json` file. This includes setting the `outputtype` to `console`, which facilitates sending the agent logs to log aggregation tools. When `AppDynamicsConfig.json` is required, the `FULL_AGENT` feature must be set to support the use of environment variables and the configuration file. See .NET Agent for Linux Advanced Configuration Options.

1. Create the `AppDynamicsConfig.json` file with the required configuration. For example:

```
{
    "feature": [
        "FULL_AGENT"
    ],
    "log": [
        {
        "outputtype": "console"
        }
    ]
}
```

2. Edit the Dockerfile to copy `AppDynamicsConfig.json` to the image:

```
COPY ./AppDynamicsConfig.json /opt/appdynamics/AppDynamicsConfig.json
```

(On-Premises Controller only) Copy the Controller Certs to the Image

For .NET Agent for Linux agents communicating with an on-premises Controller, the Controller certs must be copied to the image. See Controller SSL and Certificates.

1. Edit the Dockerfile to copy the file containing the on-premises certs to the image. For example:

```
COPY ./onprem-cacerts /opt/appdynamics/conf/cacerts
```

2. Set the `APPDYNAMICS_CONTROLLER_SSL_ENABLED` and `APPDYNAMICS_CONTROLLER_SSL_CERTFILE` environment variables. See .NET Agent for Linux Environment Variables. For a Docker application, set these environment variables in the Dockerfile:

```
ENV APPDYNAMICS_CONTROLLER_SSL_ENABLED="true"
ENV APPDYNAMICS_CONTROLLER_SSL_CERTFILE="/opt/appdynamics/cacerts"
```

   For a Kubernetes application, set these environment variables in the ConfigMap.

Example Configuration for Using a Dockerfile

This Dockerfile builds a single image with the application and agent binaries included and this Kubernetes deployment spec references that Docker image.

.NET Agent for Linux Supported Environments

Supported Environments

The AppDynamics .NET Agent for Linux is supported on these environments:

Operating Systems

- CentOS 7+
- Red Hat Enterprise Linux 7
- Debian 9+
- Ubuntu 14+
- Alpine 3.11+

.NET Versions

- .NET Core 2.0, 2.1, 3.0, and 3.1
- .NET 5.0

ASP.NET Core

- MVC
- Razor
- WebAPI

Backend Calls

- HttpClient backend calls
- ADO.NET backend calls (SqlClient, MySql, SQLite, PostgreSQL)
- EntityFramework
- Redis
- Oracle
- MongoDB
- RabbitMQ

Application Environment

- Applications running in a container (needs to match OS)
- Applications running on a host/VM (needs to match OS)

AppDynamics Controller >= 4.4.1

The ability to customize Business Transaction detection is supported by .NET Agent for Linux 4.5.9 and requires Controller >= 4.5.2. The .NET Agent 4.5.9 works with Controller >= 4.4.1, however the customizable transaction detection and configuration capabilities require >= 4.5.2.

Current Version Limitations

EUM for .NET Core on Linux is currently not supported.

These locations in the Controller do not support the .NET Agent for Linux. Their panes appear empty in such cases:

- Node memory
- Tier IIS AppPools
- Tier network dashboard

.NET Core 3.0 Limitations

.NET Agent >= 4.5.16 supports .NET Core 3.0 with limitations. These items are not supported:

- Partial snapshots do not include callgraph support
- Callgraph granularity
- Snapshot configuration using node properties
- .NET Core 3.0 assembly linking
- .NET Core 3.0 assembly unloadability

.NET Agent for Linux Environment Variables

Related pages:

- .NET Agent Configuration Properties
- Administer the .NET Agent
- Configure the .NET Agent
- Name .NET Tiers

This is a reference page for the configuration properties for the .NET Agent for Linux that you can set using system environment variables.

Agent Account Name

Specifies the account name for the SaaS or multi-tenant Controller.

Environment Variable: `APPDYNAMICS_AGENT_ACCOUNT_NAME`
Type: String
Default: For single-tenant Controllers, the agent assumes the default of `customer1` if you do not specify an account name.
Required: Only for SaaS or multi-tenant Controllers.

Agent Application Name

Specifies the business application you use in the Controller. If the application name does not exist, the Controller will create it when the agent registers. See Overview of Application Monitoring.

Environment Variable: `APPDYNAMICS_AGENT_APPLICATION_NAME`
Type: String
Default: None
Required: Yes

Agent Account Password

Specifies the account access key for the Controller.

Environment Variable: `APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY`
Type: String
Default: None
Required: Yes

Agent Node Name

The name of the node.

In general, the node name must be unique within the business application and physical host. To use the same node name for multiple nodes on the same physical machine, create multiple virtual hosts using the Unique Host ID property. See Unique Host ID.

Environment Variable: `APPDYNAMICS_AGENT_NODE_NAME`
Type: String
Default: None
Required: Yes

Agent Reuse Node Name

Set this environment variable to true to reuse node names in AppDynamics. When you set the property to true, you don't need to supply a node name, but you do need to provide a node name prefix using `APPDYNAMICS_AGENT_REUSE_NODE_NAME_PREFIX`.

This property is useful for monitoring environments where there are many CLRs with short life spans. When true, AppDynamics reuses the node names of historical CLRs for new CLRs. Reusing node names avoids a proliferation of differently named nodes in AppDynamics over time, particularly when the nodes are essentially identical processes that run over different times.

AppDynamics generates a node name with App, Tier, and Sequence number. The node names are pooled. For example, the sequence numbers are reused when the nodes are purged (based on the node lifetime).

When the .NET Agent for Linux starts up, it logs output to the console until it registers with the Controller and the Controller generates the node name.

The Controller reuses node names based on the `node.retention.period` property. See Historical and Disconnected Nodes.

Environment Variable: `APPDYNAMICS_AGENT_REUSE_NODE_NAME`
Type: Boolean
Default: False
Required: No

Example: Using the following environmental variable specifications, the Controller generates a node name with the prefix `reportGen`. Node names will have suffixes `--1`, `--2`, and so on, depending on the number of nodes are running in parallel. The name of a node that is shut down and qualifies as a historical node may be reused by a new node.

```
APPDYNAMICS_AGENT_REUSE_NODE_NAME=true

APPDYNAMICS_AGENT_REUSE_NODE_NAME_PREFIX=reportGen
```

Agent Reuse Node Name Prefix

When you configure the agent to reuse node names, use this property to specify the prefix the Controller uses to generate node names dynamically.

Environment Variable: `APPDYNAMICS_AGENT_REUSE_NODE_NAME_PREFIX`
Type: String
Default: None
Required: When `APPDYNAMICS_AGENT_REUSE_NODE_NAME=true`
Example: Using the following environmental variable specifications, the agent directs the Controller to generate a node name with the prefix `reportGen`. Node names will have suffixes `--1`, `--2`, and so on, depending on how many nodes are running in parallel.

```
APPDYNAMICS_AGENT_REUSE_NODE_NAME=true

APPDYNAMICS_AGENT_REUSE_NODE_NAME_PREFIX=reportGen
```

Agent Tier Name

Specifies the name of the tier that this .NET node belongs to. Note that this is not the deployment name on the application server.

Environment Variable: `APPDYNAMICS_AGENT_TIER_NAME`
Type: String
Default: None

Required: Yes

Analytics SSL Enabled

Specifies whether or not the .NET Agent for Linux sends the default transaction data to the Analytics Agent via SSL.

Environment Variable: `APPDYNAMICS_ANALYTICS_SSL_ENABLED`
Type: String
Default: False
Required: No

Analytics Hostname

Specifies the hostname or IP address of the analytics host.

Environment Variable: `APPDYNAMICS_ANALYTICS_HOST_NAME`
Type: String
Default: None
Required: No

Analytics Port

Specifies the port number of the analytics port.

Environment Variable: `APPDYNAMICS_ANALYTICS_PORT`
Type: Positive integer
Default: None
Required: No

Controller Hostname

Specifies the hostname or the IP address of the AppDynamics Controller. For an on-premises Controller, use the value for Application Server Host Name you provided when you installed the Controller. If you use the AppDynamics SaaS Controller, see the Welcome email from AppDynamics for the name of your Controller.

Environment Variable: `APPDYNAMICS_CONTROLLER_HOST_NAME`
Type: String
Default: None
Required: Yes, may also be set in the agent `AppDynamicsConfig.json` file as `"host"`.

Controller Port Number

Specifies the HTTP(S) port of the AppDynamics Controller. If the `APPDYNAMICS.CONTROLLER.SSL.ENABLED` environment variable is set to `true`, specify the HTTPS port of the Controller; otherwise, specify the HTTP port.

Environment Variable: `APPDYNAMICS_CONTROLLER_PORT`
Type: Positive Integer
Default: `8090`

- For On-premises installations, the defaults are port `8090` for HTTP and port `8181` for HTTPS.
- For the SaaS Controller, use port `80` for HTTP or port `443` for HTTPS.

Required: Yes, may also be set in the agent AppDynamicsConfig.json file as `"port"`.

Controller SSL Enabled

When set to `true` enables encryption over SSL between the agent and the Controller.

Environment Variable: `APPDYNAMICS_CONTROLLER_SSL_ENABLED`
Type: Boolean
Default: false

Required: No, may also be set in the agent `AppDynamicsConfig.json` file as `"ssl"`.

Controller SSL Certfile

The path to the folder containing certificate files used to connect to the SSL-enabled Controller.

Environment Variable: `APPDYNAMICS_CONTROLLER_SSL_CERTFILE`
Type: String
Default: None

Required: No. Optionally, you can set in the agent `AppDynamicsConfig.json` file as `"certfile"`.

Controller SSL Certdirectory

The path to the folder containing the certificate directory used to connect to the SSL-enabled Controller.

> Certificate directory values are valid only if you do not input certificate files (`APPDYNAMICS_CONTROLLER_SSL_CERTFILE`).

Environment Variable: `APPDYNAMICS_CONTROLLER_SSL_CERTDIR`
Type: String
Default: None

Required: No. Optionally, you can set in the agent `AppDynamicsConfig.json` file as `"certdir"`.

HTTP Proxy Host

The name of the proxy host.

Environment Variable: APPDYNAMICS_PROXY_HOST_NAME
Type: String
Default: None

Required: No, may also be set in the agent `AppDynamicsConfig.json` file in the `"proxy"` section.

HTTP Proxy Port

The port number of the proxy.

Environment Variable: APPDYNAMICS_PROXY_PORT
Type: Positive integer
Default: None

Required: No, may also be set in the agent `AppDynamicsConfig.json` file in the `"proxy"` section.

HTTP Proxy User

The name of the user that connects to the proxy.

Environment Variable: APPDYNAMICS_PROXY_AUTH_USER
Type: String
Default: None

Required: No, may also be set in the agent `AppDynamicsConfig.json` file in the `"proxy"` section.

HTTP Proxy Password

The password of the user that connects to the proxy.

Environment Variable: APPDYNAMICS_PROXY_AUTH_PASSWORD
Type: String
Default: None

Required: No, may also be set in the agent `AppDynamicsConfig.json` file in the `"proxy"` section.

Sample Agent Configuration File

The following environment variables can be set in the `AppDynamicsConfig.json` file:

- `APPDYNAMICS_CONTROLLER_HOST_NAME`
- `APPDYNAMICS_CONTROLLER_PORT`
- `APPDYNAMICS_CONTROLLER_SSL_ENABLED`
- `APPDYNAMICS_CONTROLLER_SSL_CERTFILE`
- `APPDYNAMICS_CONTROLLER_SSL_CERTDIR`

The example file below demonstrates how to set these variables:

```
"controller":
{
   "host": "ControllerHost",
    "port": 8181,
     "account": "customer1",
      "password": "test",
       "ssl": true,
       "certdir": "/opt/appdynamics/dotnet/conf/",
       "certfile": "/opt/appdynamics/dotnet/conf/cacerts.pem",
        "enable_tls12": true,
        "proxy":
                {
                        "host" : "proxy",
                        "port" :900,
                        "authentication" :
                        {
                                "username" : "user",
                                "domain"  : "domainname",
                                "password" :"pwd"
                        }
                }
}
```

Configure Analytics for .NET Agent for Linux

Related pages:

- Analytics
- .NET Agent for Linux Environment Variables

AppDynamics Application Analytics enables you to do real-time analysis of business performance that is correlated with the performance of your application software.

Configure Analytics Settings

If you are configuring the .NET Agent for Linux to send the default transaction data to the Analytics Agent, add a setting for the analytics host and port using either environment variables or in the agent json configuration file, config.json.

Configure Analytics Settings via JSON

The following json syntax shows the analytics configuration settings:

```
"analytics: { "host": <analyticsHostName>, "port: <analyticsPort>, "ssl": <true || false> }"
```

Configure Analytics Settings via Environment Variables

The following are the environment variables that you can use to configure Analytics:

- APPDYNAMICS_ANALYTICS_SSL_ENABLED
- APPDYNAMICS_ANALYTICS_HOST_NAME
- APPDYNAMICS_ANALYTICS_PORT

See .NET Agent for Linux Environment Variables

.NET Agent for Linux Advanced Configuration Options

This page describes the advanced configuration properties for the AppDynamics .NET Agent for Linux you use to enable the .NET Agent to work without Docker. Before you start instrumenting your applications with the .NET Agent, you must use the current version of the .NET Core 2.0+. See .NET Agent Configuration Properties.

Agent Configuration File

The agent configuration file is similar to the one used to configure .NET Core Agent on Windows.

Once you have deployed the agent, you can customize the instrumentation of your application.

You can edit the `AppDynamicsConfig.json` file using this information:

```
{
  "controller": {
    "host": "controller.saas.appdynamics.com",
    "port": 443,
    "account": "account-name",
    "password": "access-key",
    "ssl": true
    "proxy" : {
        "host": "proxy-host",
        "port": 9090,
           "authentication": {
              "username": "proxy-user",
              "password": "proxy-password"
           }
      }
  },
  "application": {
    "name": "My Application",
    "tier": "Sample tier",
    "node": "Instance1",
        "reusenodename": "true",
        "nodenameprefix": "prefix"
  },
  "analytics": {
        "host": "localhost",
        "port": 9090
  },
  "feature": [
    "FULL_AGENT"
  ]
}
```

The .NET Agent for Linux supports reuse node name configuration to help manage monitoring environments where there are many CLRs with short life spans. The CLR node names are reused to avoid a proliferation of differently named nodes in AppDynamics. Include the `"reusenodename": "true"` configuration, with `"nodenameprefix": "prefix"` to enable reuse node name without having to supply a specific node name.

You can also use environment variables to configure reuse node name, such as `ENV APPDYNAMICS_AGENT_REUSE_NODE_NAME=true` and `ENV APPDYNAMICS_AGENT_REUSE_NODE_NAME_PREFIX=<prefix>`.

Configure Using AppDynamicsConfig.json: If you are configuring your agent using environment variables with the json configuration file, or just with the json configuration file, you must include the FULL_AGENT feature in the json file; otherwise, the agent will not work.

```
,
  "feature": [
    "FULL_AGENT"
  ]
```

FULL_AGENT feature mode is automatically enabled when configuring the agent just using environment variables.

.NET Agent for Linux does not support extended functionality, such as specifying instrumentors and allowlisting. Do not specify these if you copy the configuration from .NET Core Agent on Windows.

Configuration Options

You can enable the agent to work without Docker. However, you must ensure that the OS is supported by the current agent.

1. Make these files available to the application. They can be located next to the application files or in a separate folder:
    - `AppDynamics.Agent.netstandard.dll`
    - `libappdprofiler.so`
    - `libappdprofiler_musl.so`
    - `libappdprofiler_glibc.so`
2. Create these environment variables by adding them to your application:

Environment Variable Name	Value
CORECLR_PROFILER	{57e1aa68-2229-41aa-9931-a6e93bbc64d8}
CORECLR_ENABLE_PROFILING	1
CORECLR_PROFILER_PATH	Path to the `libappdprofilerdynamic` library. For example, `<application_folder_path>/libappdprofiler.so` See Using .NET Core for Linux SDK to determine the location of the `libappdprofiler.so` library.

   You can create environment variables on multiple levels so it is important that you set them in the context of the monitored application:
    - Environment variables are inherited from global host level.
    - Environment variables are parent process or service.
    - Environment variables are set prior to the start of the application using the shell script or similar.
3. Place the agent configuration file next to the appropriate binaries. There are two supported options:
    - Global agent configuration file - You can place the AppDynamicsConfig.json file next to the agent binaries. This method ensures that the default option to monitor each application configured is used for the agent.

      The .NET Agent for Linux does not support overriding node name in the agent configuration. Therefore, monitoring multiple applications on the same host using global configuration is not supported. Only the first application reports.

    - Local agent configuration - You can place the `[appname].AppDynamicsConfig.json` file next to the application binaries, where [app name] should match your application DLL/EXE name. Using this option, you can attach AppDynamics configuration to the application package and deploy it as a bundle. It is activated when you set the proper environment variables and the agent binaries are present.

      Local agent configuration has higher priority than global configuration.

Log Configuration Options

By default, you can use the boost log for both the profiler and the .NET Agent. Logging occurs even if no log file exists. By default, these three log files are created:

- Agent Log - Filename is `Agent_%N.log` and the default log level is `INFO`.
- Profiler Log - Filename is `Profiler_%N.log` and the default log level is `INFO`.
- Warn Log - Filename is `Warn_%N.log` and the default log level is `WARN`.

The log configuration is read during the profiler initialization. Before the log configuration is read, some logs are printed to the console.

This table describes the properties that you can configure in the log configuration file.

Log Property	Description	
`max_size`	Maximum total size of stored files, in bytes. Default value is 10485760 bytes (10 MB).	• Maximum capacity per logger is: `max_size * max_files`. • Rotation occurs when the file size reaches `max_size`. For example, if you configure the Agent log using the default values for `max_size` and `max_files`:

max_files	Maximum total number of stored files. Default value is 10.	• When a log file (Agent_0.log) reaches its max_size (10 MB), a new file named (Agent_1.log) is created. • Once it reaches the maximum number of files, the older file is deleted and a new file is created. The log directory is updated with a new set of files [Agent_1.log, Agent_2.log, ... Agent_9.log, Agent_10.log], Agent_0.log is deleted, and a new file named Agent_10.log is created. If the log directory contains the log files from a previous run, then the newly initiated agent creates a log file using an incremental value from the previous run. For example, If the previous run has two agent logs, Agent_0.log and Agent_1.log, then the newly initiated Agent creates Agent_3.log.
directory	The root directory. It accepts both relative and absolute path; however, no environment variable is accepted. Default value is %TEMP%/appd/dotnet.	
filename	The file name. By default, no process ID is included in the file name; however, you can include and configure the process ID in the filename (%P). For example, if you configure the filename as Log_%P_%N.log, then the filename appears as Log_1123_0.Log (where 1123 is the process ID). Default value is Agent_%N.log (where %N represents the incremental value starting from 0).	
exclude_tid	Determines whether to include the native thread ID in the log file. Default value is false.	
level	The minimum log level. Accepts these values: Trace, Debug, Info, Warn, Error, and Fatal. Default value is Info. These values are case-insensitive.	
channel	This field accepts: • Fully qualified class name. • Class name with wildcard (*) at the start, end, or at both the ends of the name. • Empty channel name, or .* or * These values are used as the default channel names.	

To incorporate any changes you made to the logging configuration, you must restart the application. The auto reload feature for logging configuration is not supported.

Log Content Example

This log example shows content for the Agent_0.log file:

Agent_0.log

```
2021-05-24 15:26:03.741862 2056 11672 3 INFO [com.appdynamics.LifetimeManager] Initializing via agent bootstrap
```

Whereas:

- 2021-05-24 15:26:03.741862 is the timestamp.
- 2056 is the process ID.
- 11672 is the native thread ID.
- 3 is the managed thread ID.
- INFO is the log level.
- com.appdynamics.LifetimeManager is the channel name.
- Initializing via agent bootstrap is the log content.

Log Structure

This log example contains two processes: 11120 and 76890. Based on the log configuration, each process has its own set of log files.

- If process 11120 is initiated first, then it may create Agent_0.log (for Agent logs) and Profiler_0.log (for Profiler logs).
- When process 76890 is initiated, then it may create Agent_1.log (for Agent logs) and Profiler_1.log (for Profiler logs).
- Whichever log file reaches its maximum size first, its next log file is created by adding an incremental number to the file name. Log ordering is not maintained and differs for each process.

Logs
Profiler_0.log Profiler_1.log Warn_0.log BusinessTransactionsLog_0.log .... Agent_0.log Agent_1.log Profiler_2.log .... Agent_2.log

Supported Features

This table describes the supported Boost log features:

Feature	Description
Log order rules	No rules are processed after a final rule matches. Rules are applied: • Based on the order of the logging configuration in the `AppDynamicsConfig.json` file. • When the log message channel name matches the channel name of the log configuration. • When the severity of the log message =< the level of the log configuration. ⓘ However, if the log is written using the first logger, and the channel name is matched using the second logger with a wildcard, the second logger will not write messages to log files.
`minlevel`	Minimal level to log.

Wildcard character (*)	• If the wildcard is in the middle of a word (for example, com.*.test), then it is used as the default channel name (*)
	• You can include a wildcard at:
	  • The beginning (for example, *.appdynamics.test
	  • The end (for example, com.appdynamics.*)
	  • Both the beginning and end (for example, *.appdynamics.*)

Log Order Rules Examples

For this log order rules example, there are N loggers: logger1, logger2 ... loggerN.

where loggerN depends on logger(N-1) ... logger1.
and logger(N-1) depends on logger(N-2) ...logger1, and so on.

1. The "`ByteCode`" log is part of the "`AgentLog`". As a result, no logging occurs through `ByteCode` logger.

```
{"filename" : "RESTHearbeat", "level" : "Info", "channel" : "com.appdynamics.REST.HeartBeatLog" },
{"filename" : "AgentLog", "level" : "Info", "channel" : "*" },
{"filename" : "ByteCode","level" : "Info","channel" : "com.appdynamics.bci.*" }
```

2. The "`RESTHeartbeat`" log is part of "`REST`". As a result, the "`RESTHeartbeat`" log is ignored.

```
{"filename" : "REST", "level" : "Info", "channel" : "com.appdynamics.REST.*" },
{"filename" : "RESTHeartbeat", "level" : "Info", "channel" : "com.appdynamics.REST.HeartBeatLog" }
```

3. Although "RESTHeartbeat" and "REST" log have the exact same channel name, only the `minLogLevel` of the first logger is considered. In this case, it is "Info".

```
{"filename" : "REST", "level" : "Info", "channel" : "com.appdynamics.REST.HeartBeatLog" },
{"filename" : "RESTHeartbeat", "level" : "Info", "channel" : "com.appdynamics.REST.HeartBeatLog" }
```

4. Channels not written through loggers "RESTHearbeat" and "ByteCode" are written through the "AgentLog" logger though the channel name with "com.appdynamics.bci.test" which matches both the ByteCode and AgentLog loggers.

```
{"filename" : "RESTHearbeat", "level" : "Info", "channel" : "com.appdynamics.REST.HeartBeatLog" },
{"filename" : "ByteCode", "level" : "Info", "channel" : "com.appdynamics.bci.*" },
{"filename" : "AgentLog","level" : "Info","channel" : ".*" }
```

Log Configuration Examples

This example shows how to set logging configuration. Each log file can have its own set of configurations.

AppDynamicsConfig.json

```
{
  "controller": {
    "host": "......"
  },
  "application": {
    "name": "..."
  },
  "log": [
    {
      "filename": "WarnLog",
      "level": "WARN",
      "channel": "*"
    },
    {
      "filename": "Profiler",
      "level": "INFO",
      "directory": "/temp/appd/logs",
      "channel": "appd.agent.Profiler"
    },
    {
      "filename": "BusinessTransactionsLog",
      "level": "INFO",
      "channel": "com.appdynamics.BusinessTransactions"
    },
    {
      "filename": "RESTHearbeat",
      "level": "INFO",
      "channel": "com.appdynamics.REST.HeartBeatLog"
    },
    {
      "filename": "ByteCode",
      "level": "INFO",
      "channel": "com.appdynamics.bci.*"
    },
    {
      "filename": "RESTCommunications",
      "level": "INFO",
      "channel": "com.appdynamics.REST.*"
    },
    {
      "filename": "RESTCommunications",
      "level": "INFO",
      "channel": "com.appdynamics.METRICS.MetricSender"
    },
    {
      "filename": "SamplingTrace",
      "level": "TRACE",
      "channel": "com.appdynamics.ManagedAgentAPI.DumpStats"
    },
    {
      "filename": "Analytics",
      "level": "INFO",
      "channel": "com.appdynamics.ee.service.analytics.Analytics",
      "max_size": 31457280,
      "max_files": 2,
      "directory": "./logs",
      "exclude_tid": false
    },
    {
      "filename": "Agent",
      "level": "INFO",
      "channel": "*"
    }
  ]
}
```

This example shows multiple log entries for the same file (both the directory and filename have the same value). The file log configuration uses the first entry, and all subsequent configuration for `max_size`, `max_files` and `exclude_tid` is ignored.

AppDynamicsConfig.json

```
{
     "filename": "agent",
     "level": "TRACE",
     "max_size":1000,
     "max_files":2,
     "channel": "com.appdynamics.METRICS.MetricSender"
    },
    {
     "filename": "agent",
     "level": "DEBUG",
     "max_size":2000,
     "max_files":2,
     "channel": "com.appdynamics.METRICS.*"
    },
    {
     "filename": "agent",
     "level": "INFO",
     "max_size":3000,
     "max_files":1,
     "channel": "com.appdynamics.*"
    }
}
```

Enable Preview Features

Preview features have been thoroughly tested inhouse at AppDynamics but because of the wide variety of possible use cases, AppDynamics releases these features for you to test and use at your discretion. We do not recommend that you use preview features in a production environment.

Enable Preview Features

To use features released under the preview feature flag, you must first enable them.

There are two ways to turn on the preview feature flag:

- Add the flag `PREVIEW_FEATURE` in the feature section of the `AppDynamicsConfig.json` file. For example:

  ```
  "feature": [ "FULL_AGENT", "PREVIEW_FEATURE" ],
  ```

- Define the environment variable `APPDYNAMICS_PREVIEW_FEATURE_ENABLED` as `true`.

Enabling Preview Features enables HTTP Custom Discovery Rules for the .NET Agent for Linux.

.NET Agent for Linux Troubleshooting

This page describes general techniques when troubleshooting the AppDynamics .NET Agent for Linux deployment problems, and provides a few specific scenarios you may encounter with workarounds.

Agent Startup Situations

The agent startup is dependent on three basic steps:

1. .NET Core loads the agent using profiler settings in the environment variables.
2. AppDynamics agent loads the configuration and additional library.
3. AppDynamics agent registers on the Controller using connection details found in the configuration file.

The sample startup console output on .NET Agent for Linux indicates a successful startup when going through these steps.

The following situations may occur:

Symptom	Probable root causes
No console output from AppDynamics was written to the console	• Environment variables were not set correctly • Agent binaries do not match the OS version
Agent complains about failing to load the configuration file	`AppDynamicsConfig.json` or `[appname].AppDynamicsConfig.json` were not found in the corresponding folders.
Agent complains about HTTP errors	Agent failed to communicate to the Controller using connection settings provided in the configuration file.

Accessing Logs While Running Docker Container in the Background

The default agent configuration redirects all agent output to the console. This is enabled for troubleshooting purposes.

If you are running the container in the background—-d for detach—it is possible to access all output using the following Docker command:

```
docker logs <container id>
```

Default Agent Log Files Location

The agent log files location is at `/tmp/appd/dotnet` by default.

You can change this setting to redirect all logs to the console output for a single view troubleshooting by adding the following configuration to the `AppDynamicsConfig.json` file as follows:

```
"log": [
   {
     "outputtype": "console"
   }
 ]
```

.NET Agent for Linux Business Transaction Configuration

Related pages:

- .NET Agent for Linux
- Overview of Application Monitoring
- Business Transactions
- ASP.NET Entry Points
- POCO Entry Points
- Custom Match Rules

As of 4.5.9, you can configure simple POCO or ASP.NET business transactions for the .NET Agent for Linux through the Controller UI.

The .NET Agent for Linux 4.5.9 works with Controller >= 4.4.1, but the customizable transaction detection and configuration capabilities require Controller >= 4.5.2. For Controller 4.4.2 - 4.5.1, and the .NET Agent for Linux, AppDynamics defaults to using the first two segments of the URI for ASP.NET business transaction naming.

Define Custom ASP.NET Business Transactions Using the Controller UI

You can create ASP.NET business transactions detection include and exclude rules with criteria that match HTTP Requests against:

Transaction splitting is supported for the HTTP request criteria.

- Method
- URI
- HTTP parameter
- Header
- Cookie

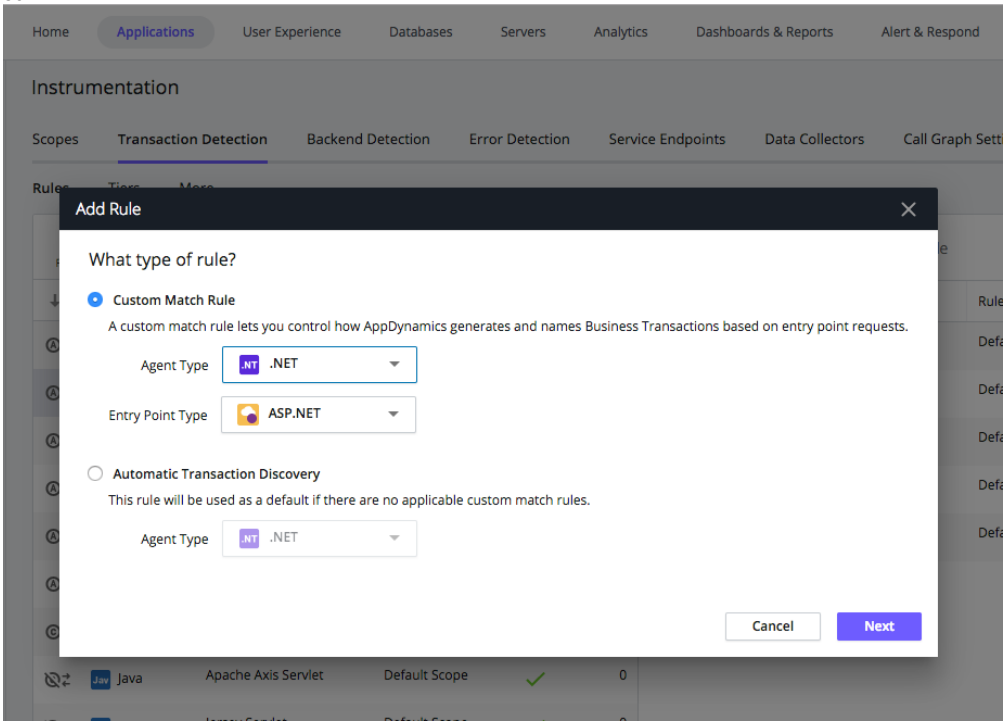
Although these HTTP request options display in the dropdown list, they are not supported by the .NET Agent for Linux. There is no effect if you configure them in the UI.

- Hostname
- Port
- Class name

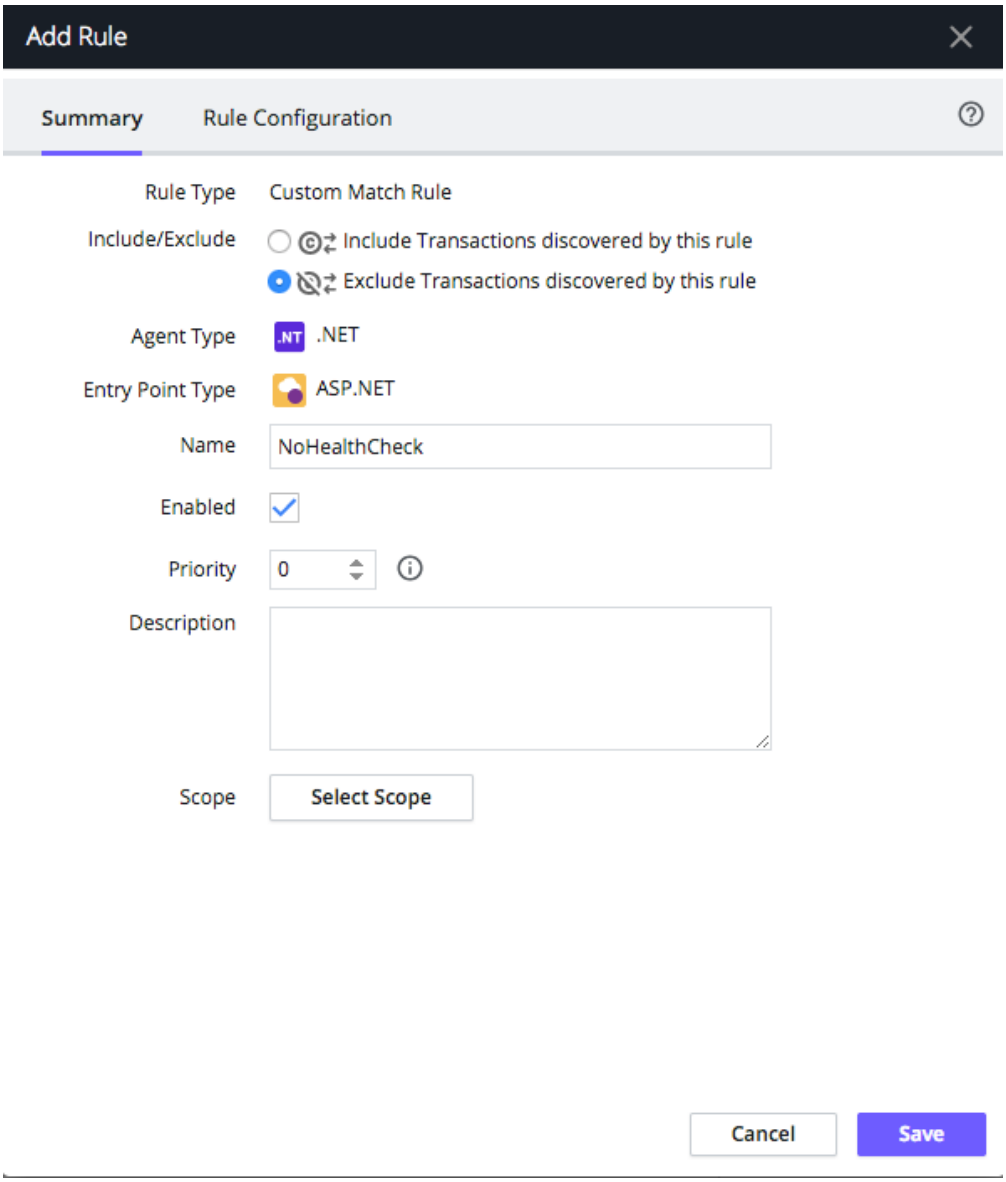
You customize business transactions for the .NET Agent for Linux similarly as other agents. Click **Configure > Instrumentation > Add Rule** and then select to create either an include or exclude rule.

ASP.NET Business Transaction Detection Rule Example

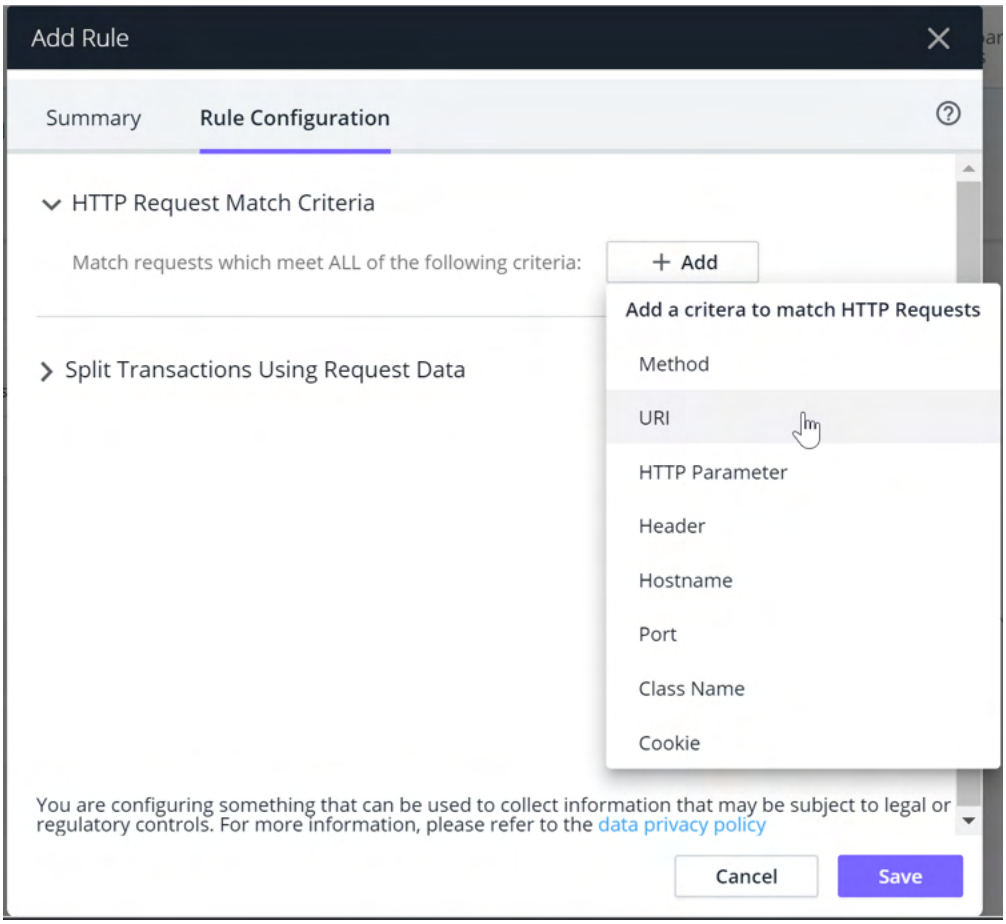
This example shows how to create a .NET Agent custom match rule that uses an ASP.NET entry point to exclude all requests whose URI equals `/healthcheck`.



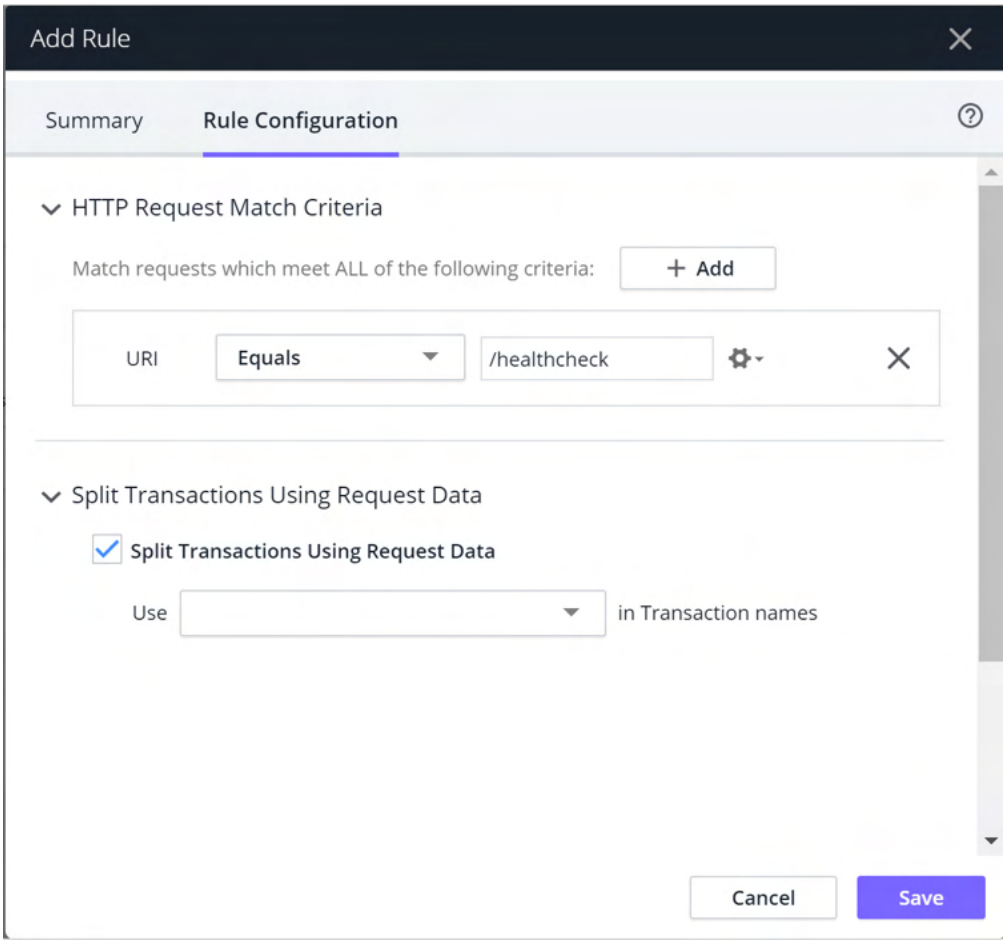
This Rule Summary specifies an *exclude* match rule. You must create an *include* rule before you can create an *exclude* rule to omit some of the transactions monitored by the include rule.



You specify the HTTP request criteria by using the URI attribute of the HTTP request to compare against. Valid criteria are **Method**, **URI**, **HTTP Parameter**, **Header**, and **Cookie**. You can split the transactions based on these request match criteria.



This HTTP request match criteria specifies where the URI equals ***/healthcheck***.



Define Custom POCO Business Transactions Using the Controller UI

Using the .NET Agent for Linux, you can create POCO business transaction rules that match synchronous or asynchronous transactions by:

- Method name
- Class name

Although these Match Classes options display in the dropdown list, they are not supported. However, there is no effect if you configure them in the UI:

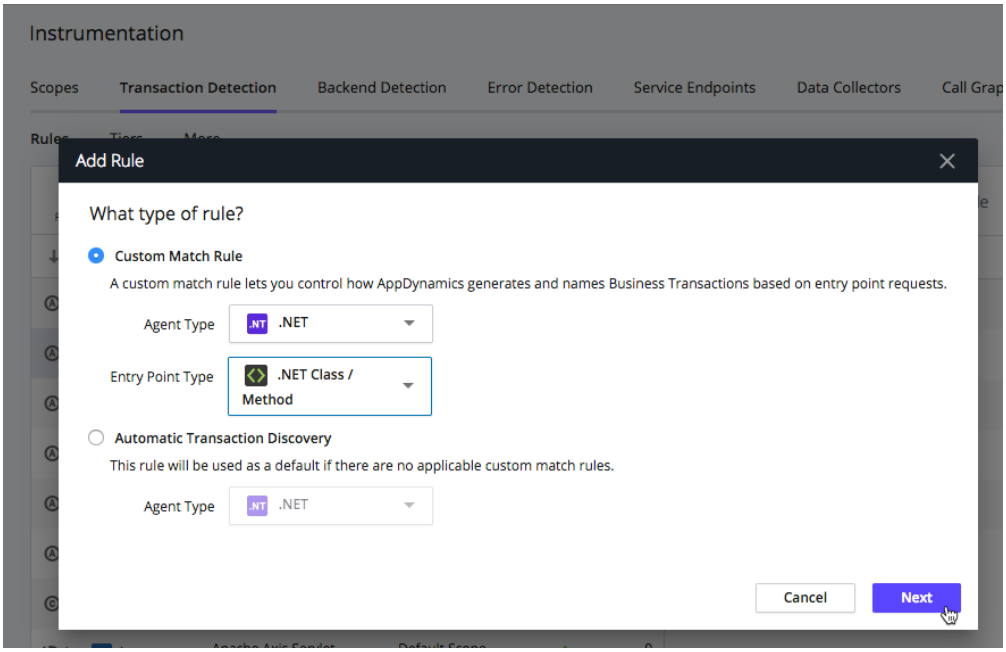
- that implements an interface which
- that extends a super class that
- that has an Annotation which

Additionally, these criteria are not supported:

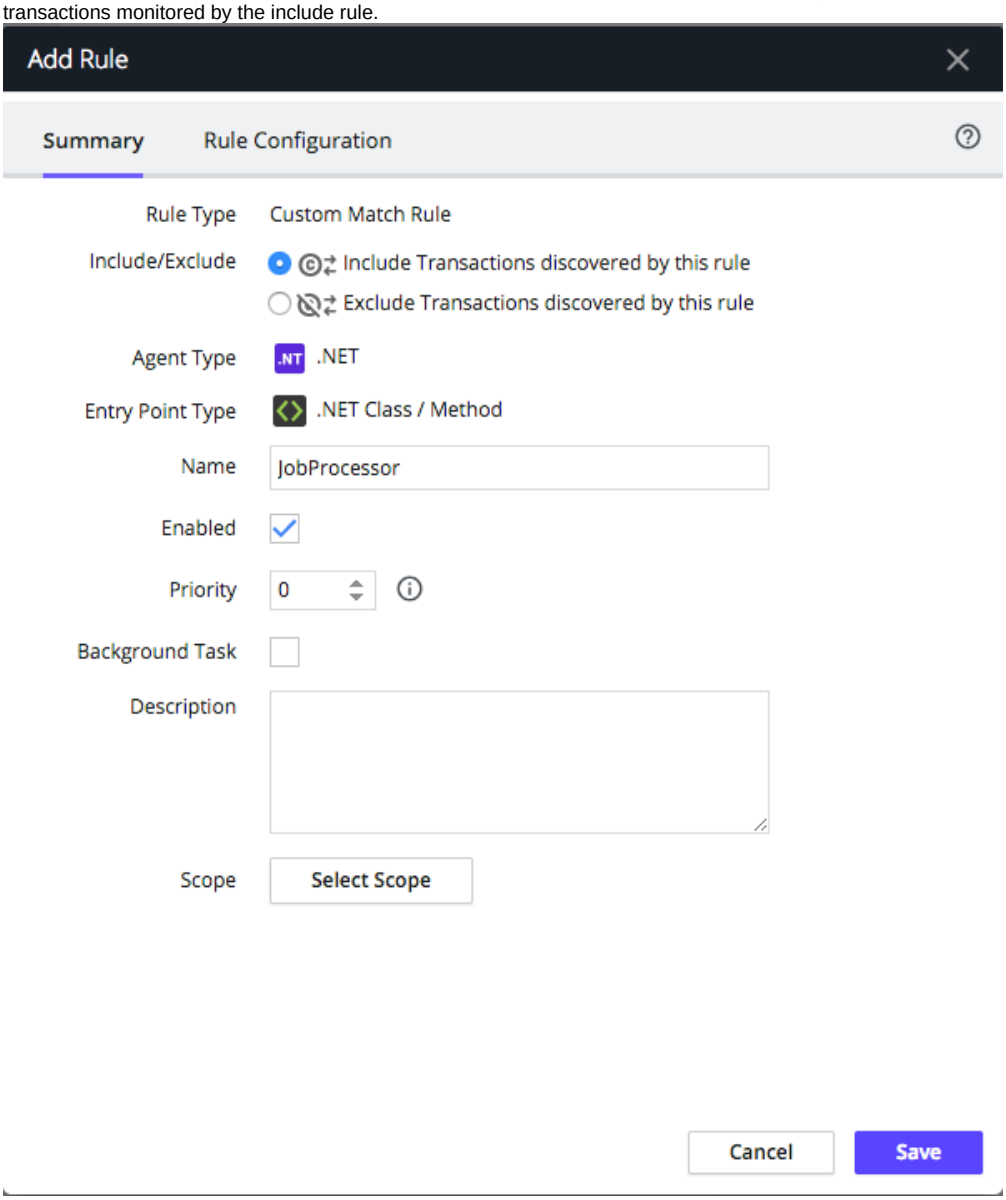
- Transaction splitting using method/class data
- Background jobs
- Interface, base class or annotation POCO conditions

Example POCO Business Transaction Detection Rule Example

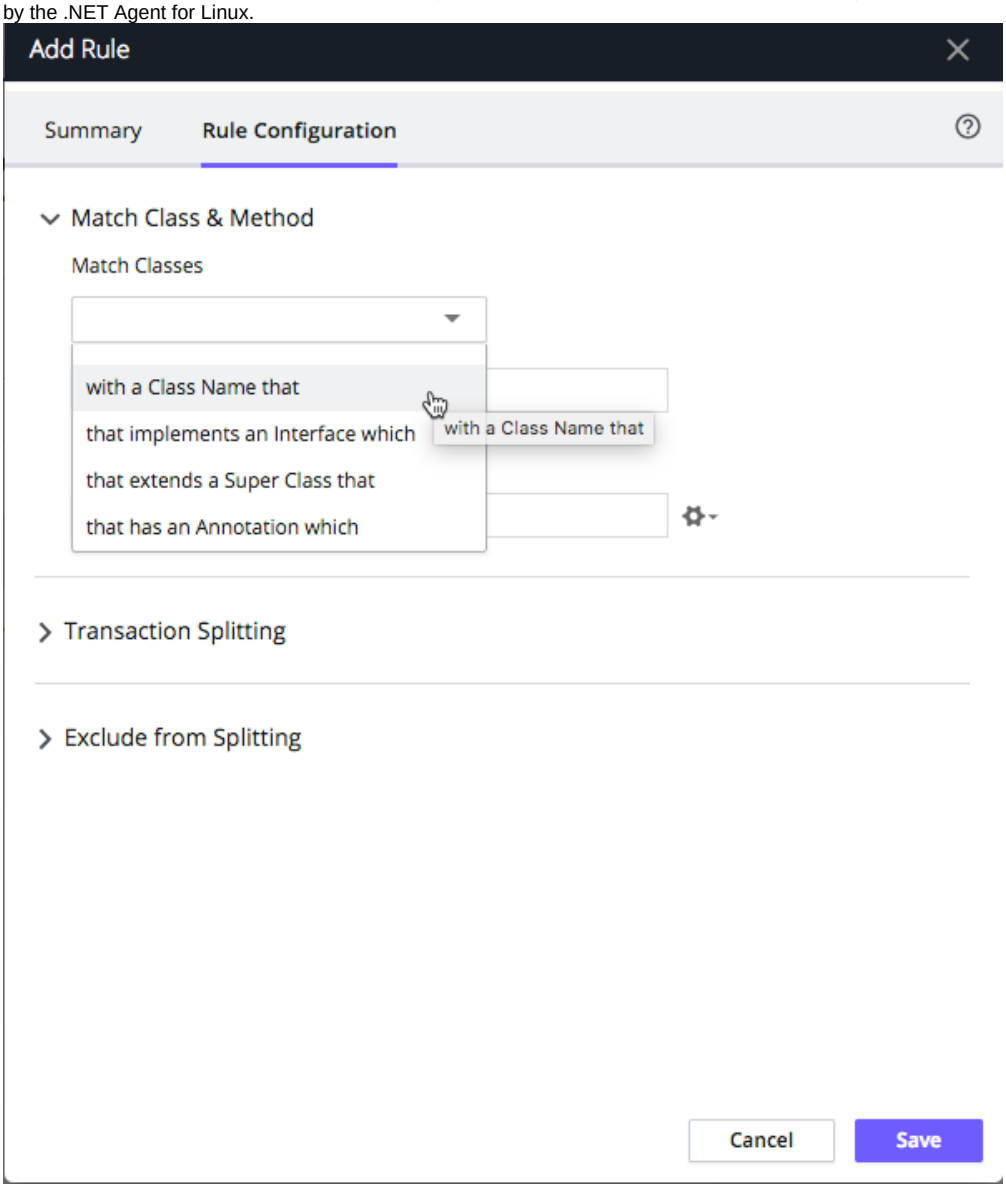
This example shows how to create a .NET Agent custom match rule that uses a POCO entry point to include all requests whose class equals `MyCompanyJobs.JobProcessor` and the method name equals `OnEvent`.


Instrumentation
Scopes
Transaction Detection
Backend Detection
Error Detection
Service Endpoints
Data Collectors
Call Grap
Rules
Add Rule
What type of rule?
Custom Match Rule
A custom match rule lets you control how AppDynamics generates and names Business Transactions based on entry point requests.
Agent Type
.NET
Entry Point Type
.NET Class / Method
Automatic Transaction Discovery
This rule will be used as a default if there are no applicable custom match rules.
Agent Type
.NET
Cancel
Next

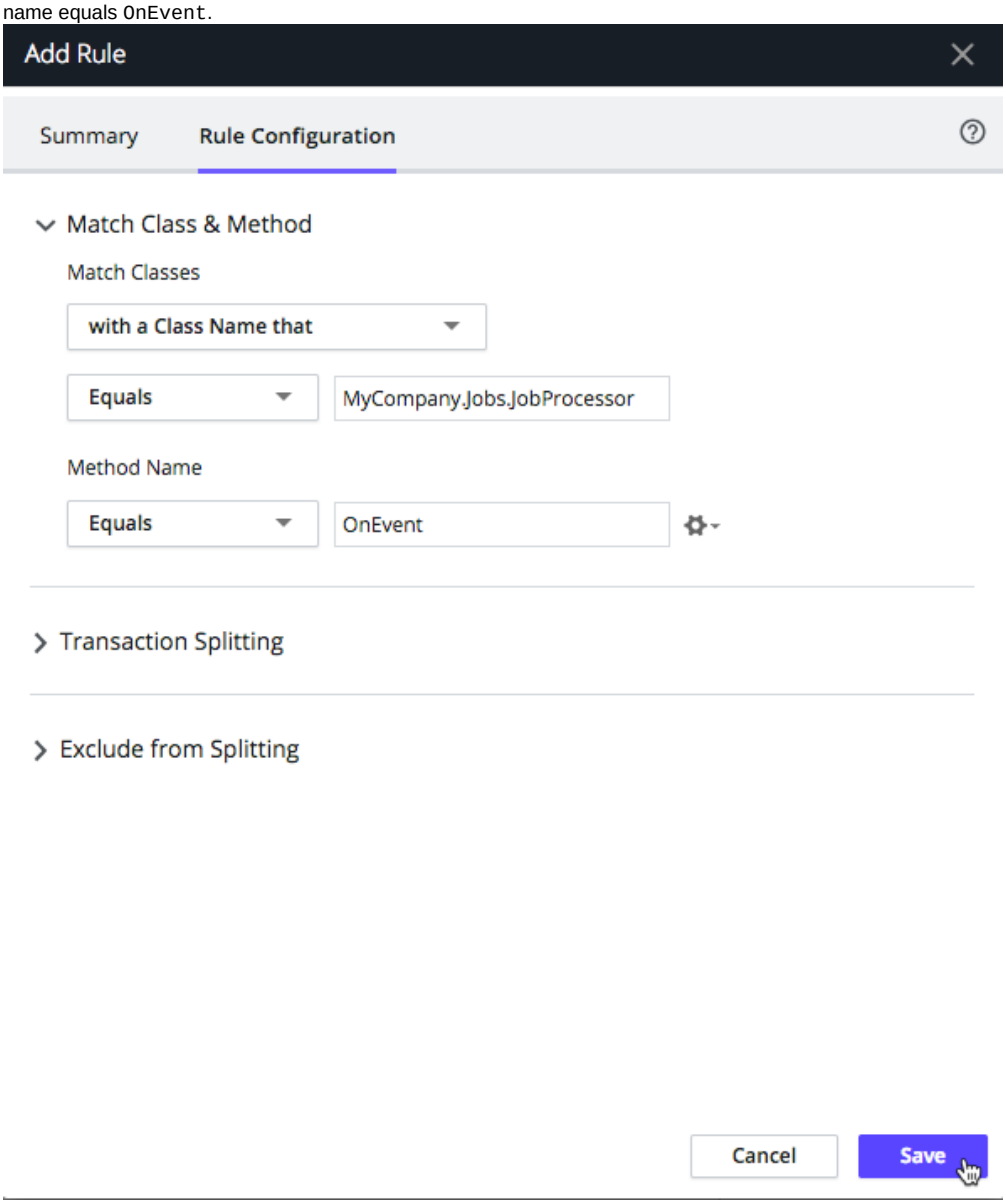
This Rule Summary specifies an *include* match rule. You must create an include rule before you can create an exclude rule to omit some of the transactions monitored by the include rule.



You can configure the rule to match classes using the ***with a Class Name that*** Match Classes option, which is the only Match Classes option supported by the .NET Agent for Linux.



The Match Class & Method specifies match criteria to match classes with a class name that equals `MyCompanyJobs.JobProcessor` and the method name equals `OnEvent`.



Transaction splitting is not supported for the .NET Agent for Linux.

Upgrade the .NET Agent for Linux

Related pages:

- Agent and Controller Compatibility
- Install the .NET Agent for Windows
- Configure the .NET Agent
- .NET Agent Directory Structure
- Troubleshoot .NET Agent Issues

This page describes how to upgrade the .NET Agent for Linux on VMs such as Docker and on hardware running Linux.

Upgrading the .NET Agent for Linux updates the agent files and maintains legacy configurations.

.NET Agent for Linux 21.5

.NET Agent for Linux 21.5 incorporates significant architectural changes to accelerate the development of missing features in the Linux Agent relative to .NET Agent for Windows.

> These architectural changes do not affect .NET Agent for Windows (MSI and Microservices).

Generally, .NET Agent for Linux 21.5 maintains parity with existing features in .NET Agent for Linux < 21.5. However due to architectural updates, there are differences between .NET Agent for Linux >= 21.5 and .NET Agent for Linux < 21.5. Before you upgrade to .NET Agent for Linux 21.5, please review the differences in this table:

Difference	.NET Agent for Linux >= 21.5	.NET Agent for Linux < 21.5
HTTP URL backend for: Default naming	.NET Agent uses the full URL including http://host:port in the naming. Full URL address is used. For example, two calls to these addresses: • http://service.cisco.com/api/add/12 • http://service.cisco.com/api/add/32 Results with these two backend names: • http://service.cisco.com/api/add/12 • http://service.cisco.com/api/add/32	.NET Agent uses the URI path of the URL for the naming. It excludes http://host:port, but assumes the "/" prefix as the name. Uses the URI path prefixed with a "/". For example, two calls to these addresses: • http://service.cisco.com/api/add/12 • http://service.cisco.com/api/add/32 Results with these two backend names: • /api/add/12 • /api/add/32

HTTP URL backend for: Custom naming	.NET Agent uses the full URL including http://host:port in the naming. The URL address split by "/" counts http://host:port as the first three segments: • "http:" • "" • "host:port" For example, two calls to these addresses with first five URL segments using "/" as the split and merge delimiter: • http://service.cisco.com/api/add/12 • http://service.cisco.com/api/add/32 Results with one backend name: • http://service.cisco.com/api/add	.NET Agent uses the URI path of the URL for the naming. It excludes http://host:port, but assumes the "/" prefix as the name. The URL address split by "/" counts segments starting from the prefixed "/" in the URI path. For example, two calls to these addresses with first three URL segments using "/" as the split and merge delimiter: • http://service.cisco.com/api/add/12 • http://service.cisco.com/api/add/32 Results with one backend name: • /api/add

Sensitive data filtering for environment variables	To filter sensitive data for environment variables:
	
	1. Add this sensitive data filter to the `AppDynamicsConfig.json` configuration file (located next to the Agent DLL file) :
	
	`{`
	`  "applies-to": "environment-variables",`
	`  "match-type": "CONTAINS",`
	`  "match-pattern": ""`
	`}`
	
	2. To integrate this filter with the Agent configuration file, see Filter Sensitive Data with the .NET Agent.
		Available using the `APPDYNAMICS_AGENT_MASK_ENV_VARS` environment variable with the Linux Agent.

Process identity	Not available	Available
In-process async call chain segments limit	5 per node (default)	10 per node (default)
POCO definitions for the generic format changes any existing generic method POCO definitions and must be re-defined using the new format. Similar to POCO definitions, the new format applies to any similar definitions for Custom Exits, Custom Data collectors, and so on because their support is enabled in the .NET Agent for Linux.	This shows the common format used to define generic POCO rules for all .NET Agents: Format `class: GenericClass`2` `method: GenericMethod<T1,T2>(T1,T2,T1,T2)`	This shows the common format used to define generic POCO rules for all .NET Agents: Format `class: GenericClass`2` `method: GenericMethod(!T1,!T2,!!M1,!!M2)`
Alpine 3.9 and 3.10	Not supported	Supported
Analytics Agent connection configuration through the `AppDynamicsConfig.json` configuration file or through the `APPDYNAMICS_ANALYTICS_*` environment variables.	Not supported Use the single environment variable `appdynamics.analytics.agent.url` to specify the full Analytics Agent URL, such as `http://localhost:9090/v2/sinks/bt`.	Supported
Analytics Agent SSL trust configuration in JSON or agent specific environment variables.	Not supported The SSL connection works if the certificate used by the Analytics Server is trusted by the operating system. As a workaround, you can configure the certificate using the `SSL_CERT_FILE/SSL_CERT_DIR` environment variable. This is a global setting that overrides your SSL configuration for all connections.	Supported with these environment variables: • `APPDYNAMICS_CONTROLLER_SSL_CERTFILE` • `APPDYNAMICS_CONTROLLER_SSL_CERTDIR`

Usage of multiple Base-64 encoded certificate (PEM) in a single file with `APPDYNAMICS_CONTROLLER_SSL_CERTFILE`.	Not supported
	
	Possible workarounds are:
	
	• Use the SSL_CERT_FILE and SSL_CERT_DIR system environment variables instead.
	
	ⓘ This is a global setting that overrides your SSL configuration for all connections.
	
	• Split certificates into multiple files. For example, enter this command:
	
	`csplit -sz INPUT_FILE '/.*BEGIN/' '{*}' --suffix-format=%i.cer --prefix=cert_`
	
	ⓘ The INPUT_FILE is the PEM certificate.
	
	• Convert PEM to PKCS#7. For example, enter this command:
	`openssl crl2pkcs7 -nocrl -certfile INPUT > OUTPUT`
	
	ⓘ The INPUT_FILE is the PEM certificate, and the OUTPUT is the PKCS#7 certificate. INPUT and OUTPUT should not be the same file.
		Supported

Requirements

- Before you begin, review the Release Notes for changes that affect your environment.
- AppDynamics requires an account access key for agent connections to single-tenant Controller accounts. AppDynamics < 4.1 only requires an account access key for multi-tenant Controller accounts.

  Single-tenant Controller customers can find the account access key in the Controller under **Settings** > **License** > **Account**.

  You must be a member of a role with the View License account level permission, see Create and Manage Custom Roles.

Binaries Overview

After downloading the agent binaries, you must extract them from the zip file into the folder you want.

The folder should contain these files:

- `AppDynamics.Agent.netstandard.dll`
- `libappdprofiler.so`
- `libappdprofiler_glibc.so`
- `libappdprofiler_musl.so`

Upgrade the .NET Agent for Linux

1. Unzip the .NET Agent for Linux installation package to a temporary directory.
2. Replace the existing agent binaries with their latest versions. The files to replace are:
   - `libappdprofiler.so`
   - `AppDynamics.Agent.netstandard.dll`
3. Copy these files to the same directory:
   - `libappdprofiler_glibc.so`
   - `libappdprofiler_musl.so`
4. Restart your application, or rebuild and then run your container image with the new versions of the agent binaries.

Node.js Agent

Related pages:

- Dynamic Language Agent Proxy
- Monitor Node.js Processes

This page provides an overview of the Node.js Agent, which enables automated transaction detection and correlation across your Node.js-enabled environment.

Node Naming for Node.js Agents

Each instrumented Node.js process corresponds to a node in the AppDynamics model.

The nodes are named by combining the prefix that you specify for the nodeName in the requires statement that you add to the application source code when you install the Node.js agent with a hyphen and a digit.

For example, if you designate a prefix of MyNode for the nodes in the MyTier tier, the nodes in that tier are named MyNode-0, MyNode-1, MyNode-2, and so on.

Instrument a Node.js Application Overview

The Getting Started Wizard provides instructions on installing the agent. In the Controller **Home** page, click **Getting Started > Node.js**.

The wizard provides the minimum information that the agent needs to communicate with the AppDynamics Controller:

- Controller Host
- Controller Port
- SSL (optional)
- Application Name
- Tier Name

Node.js Supported Environments

This page provides an overview of supported environments for the Node.js Agent.

Node.js Agent Versions

Agent Versions	Node.js Versions
>= 21.1.0	8.6+, 9, 10, 11, 12, 13, 14, 15
>= 20.5.0	8.6+, 9, 10, 11, 12, 13, 14
20.4.0	8.6+, 9, 10, 11, 12, 13
>= 4.5.21	8.6+, 9, 10, 11, 12
>= 4.5.12	6, 7, 8, 9, 10, 11
4.5.11	0.8, 0.10, 0.12, 4, 5, 6, 7, 8, 9, 10, 11

For agent >= 4.5.12, the `npm install` command will stop and print a message for the following scenarios.

- Node.js version less than 6.x:
  - This version of AppDynamics agent supports Node.js versions 6.0 and above.
  - For older versions of Node.js, use the AppDynamics agent 4.5.11 by installing with `npm install appdynamics@4.5.11`.
- Node.js version less than 8.x on Mac:
  - This version of AppDynamics agent on Mac OS supports Node.js >= 8.0.
  - For older versions of Node.js, use the AppDynamics agent 4.5.11 by installing with `npm install appdynamics@4.5.11`.

Operating Systems

- Alpine Linux Docker >= 3.7 on 64-bit platform
- Linux distribution based on glibc 2.5 and later and the x86/x86-64 architecture
- Mac OS X >= 10.8
- Windows Server 2012 R2 and later on 64-bit platform with Node.js >= 0.12.0

Transaction Naming

Entry Type	Default Transaction Naming
Node.js Web	URI

HTTP Exit Points

Supported HTTP Exit Points
Node.js HTTP client library

See Node.js Documentation for the Node.js HTTP client library.

Database Exit Points

Supported Database Exit Points

Cassandra
Couchbase
DynamoDB (using AWS SDK Driver)
Redis
Memcached
MongoDB
MSSQL
MySQL
PostgreSQL

Node.js Settings Reference

Related pages:

- Configure Transaction Analytics for Node.js and PHP Applications
- Install the Node.js Agent

This page describes the available settings that configure the Node.js Agent.

You can use environment variables for some of the settings. To apply the settings, add them to the `require` statement for the Node.js Agent in the monitored application.

General Settings

This is the complete list of settings in the require statement that you insert into your application code. Not all of these settings are required.

Property Name	Description
`controllerHostName`	The IP address or hostname of your Controller. For SaaS, you will receive this URL in your welcome email from AppDynamics. F will set these when you install the Controller.
`controllerPort`	The port to which the agent connects to the Controller. It is 8090 for an on-premises Controller and 443 for a SaaS Controller.
`controllerSslEnabled`	Set this property to true if connecting to the Controller via SSL.
`accountName`	The account name on the Controller to which this agent will report. See Account Name and Access Key Requirements for details
`accountAccessKey`	Account access key on the Controller.
`applicationName`	Name that represents the entire application in the AppDynamics console.
`tierName`	Name that represents your Node.js app or service in the flow maps.
`nodeName`	Name of the Node.js process to be monitored by this agent. See Node Names.
`noNodeNameSuffix`	Optional, set to true if you do not want the agent to add a suffix, such as -0, -1, -2, and so on, to the node name. See Node Nam
`proxyHost`, `proxyPort`	Set these options to route data to the Controller through a proxy server. The `proxyHost` is the hostname or IP address of the p `Port` is the proxy server's HTTP or HTTPS port, whichever you are using. If you set the host, you must set the port as well.
`proxyUser`, `proxyPasswordFile`	Configure the proxy username and password file if the proxy server requires credentials.
`proxy`	Set this property to true if you want to use the Java proxy version of the Node.js Agent.
`alwaysAddEumMetadataInHttpHeaders`	Set this property to true to have the Node.js Agent write business transaction metadata to the XHR header and in a cookie even considered cross-origin. See the description for the equivalent setting `always-add-eum-metadata-in-http-headers` in A Properties Reference.
`btEntryPointDelayDisabled`	true | false - Optional, defaults to false. Setting this property to true can accelerate the startup of business transactions, but it car drill-down in distributed transactions.
`debug`	Set this property to true to enable debug level logging for the agent. The default is false.
`logging`	The location, level, and other settings related to agent logging. See Node.js Agent Logging.
`maxProcessSnapshotsPerPeriod`	Optional. The default is 2. The number of automatic process snapshots allowed in `processSnapshotCountResetPeriodSec`
`processSnapshotCountResetPeriodSeconds`	Optional. The default is 60. Frequency, in seconds, at which the automatic process snapshot count is reset to 0.
`autoSnapshotDurationSeconds`	Optional. The default is 10. Length, in seconds, of automatically-triggered process snapshots.
`proxyAutolaunchDisabled`	true | false - Optional, defaults to false. Set this property to true if you need to manually launch the proxy for this agent. See Star for Node.js.
`proxyCtrlDir`	Directory path for the directory containing the domain control socket, which the agent uses to start an AppDynamics node. Optio Set manually if you are setting up a multi-tenant proxy. See Share a Proxy Among Node.js Agents.
`rootTmpDir`	Directory path for the root of the directory that stores the agent files. Optional, defaults to `/tmp/appds`.

`tmpDir`	Directory path for the subdirectory of the root directory for the monitored node. Optional, defaults to a hash of the Controller info node.
`reuseNode`	Set this property to true to enable reusing node names. This property is useful for monitoring environments where there are mar spans. The default is false.
`reuseNodePrefix`	Set this property to the string that prefixes the node name when reuseNode is set to true. The Controller uses this prefix to gene dynamically.
`certificateFile`	Directory path for the directory containing the .pem SSL certificate.
`uniqueHostId`	The unique host ID can be set to any value that will identify the host. If a value is not specified, the agent will generate the host I available for the proxyless version of the Node.js Agent.

Analytics Settings

You can configure the Node.js Agent to send default transaction data to the Analytics Agent.

To add a setting for the Analytics host and port with the following format, specify the host and port numbers. The Analytics Agent may report on the same or different host and port numbers from the Node.js Agent.

```
analytics: {
    host: <analyticsHostName>,
    port: <analyticsPort>,
    ssl: <true || false>
}
```

To configure Analytics data limits, see Configure Transaction Analytics for Node.js and PHP Applications.

Windows Settings

On Windows, the Node.js Agent requires three additional ports:

- `proxyCommPort`: default is `10101`
- `proxyRequestPort`: default is `10102`
- `proxyReportingPort`: default is `10103`

See Run the Node.js Agent on Windows for more information.

Filter Sensitive Data

If your application contains sensitive data that should not be displayed in the Controller, you can apply the following data filters to the require statement of the Node.js Agent.

Add a Sensitive Data Filter

When you enable a sensitive data filter, the Controller displays asterisks for the values of matching environment variables or system properties.

```
dataFilters: [{
   "appliesTo": "http-headers",
   "matchPattern": "host"
}],
```

Environment Variable Filter

By default, the Node.js Agent enables a sensitive data filter with an environment variables/system properties that contain the case-insensitive substring `password|key`.

```
dataFilters: [{
   "appliesTo": "env-vars",
   "matchPattern": "password|key"
}],
```

Sensitive URL Filter

You can use sensitive URL filters to configure the agent to obfuscate sensitive information from the URLs in the **Transaction Snapshot** details.

```
urlFilters: [{
   "delimiter": "/",
   "segment": "1",
   "matchPattern": "a",
   "paramPattern": "bar"
}],
```

The sensitive URL filter checks for the `matchPattern` in one segment. To check for additional segments, you can add an additional JSON array.

The filter processes requests in the following format: `/aa/bb/cc ***/bb/cc`.

The `matchpattern/parampattern` is a standard regex.

Property Name	Description
`urlFilters`	An array that you can add more objects to if you want more rules on the segment.
`delimiter`	Specifies the desired character as the URL segment endpoints. The agent splices the URL at each delimiter instance to create the segments.
`segment`	Needs to be at least one number and in ascending order (such as `3,5` or `2,6`). Wildcards (`*`) are not possible.

URL Segment Filter

With the `matchpattern` regex, the Node.js Agent decides whether or not to filter a specified segment.

If you have multiple rules, list the more specific rule before the more general rule. The rules listed in the array are evaluated based on order.

Rule A:

```
{
 'delimiter': '/',
 'matchPattern': '[0-9a-z]{8}-[0-9a-z]{4}-[0-9a-z]{4}-[0-9a-z]{4}-[0-9a-z]{12}\/.{50,}',
 'segment': '3,5'
}
```

Rule B:

```
{
 'delimiter': '/',
 'matchPattern': '[0-9a-z]{8}-[0-9a-z]{4}-[0-9a-z]{4}-[0-9a-z]{4}-[0-9a-z]{12}',
 'segment': '3'
}
```

In this example, the first rule, Rule A, should be listed before Rule B because rule A masks the third and fifth segment.

If Rule B were to be listed and evaluated before Rule A, the third segment will be masked and the fifth segment will not masked, potentially revealing sensitive information.

URL Parameter Filter

With the paramPattern regex, the Node.js Agent decides whether or not to filter a specified query parameter.

If you need to mask all URL parameters, you can add further instruction to the array.

```
{ "delimiter": "/", "matchPattern": "", "segment": "99", "paramPattern": ".+"}
```

The Node.js Agent will filter out all URL parameters.

Sensitive Message Filter

You can use sensitive message filters to configure the agent to obfuscate sensitive information contained within text messages collected by the agent from log messages or detail messages from exceptions.

```
messageFilters: [{
   "messageType":
"throwable",
   "matchPattern":
"Error.*MySQL.",
   "redactionRegex": "SQL"
}]
```

Environment Variables

If you have not provided the required settings in the require statement, the agent uses the values of the following environment variables if those variables are set. If both are set, the `require` statement value takes precedence over the environment variable.

Environment Variable	Maps to
APPDYNAMICS_CONTROLLER_HOST_NAME	controllerHostName
APPDYNAMICS_CONTROLLER_PORT	controllerPort
APPDYNAMICS_CONTROLLER_SSL_ENABLED	controllerSslEnabled
APPDYNAMICS_AGENT_ACCOUNT_NAME	accountName
APPDYNAMICS_LOGGER_OUTPUT_TYPE	You can stream logs to `stdout` by setting its value to `console`.
APPDYNAMICS_LOGGER_LEVEL	You can set the logging level of the agent. Possible values include: `FATAL`, `ERROR`, `WARN`, `INFO`, `DEBUG`, and `TRACE`.
APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY	accountAccessKey
APPDYNAMICS_AGENT_APPLICATION_NAME	applicationName
APPDYNAMICS_AGENT_TIER_NAME	tierName
APPDYNAMICS_AGENT_NODE_NAME	nodeName
APPDYNAMICS_AGENT_UNIQUE_HOST_ID	
APPDYNAMICS_ANALYTICS_HOST_NAME	host—in Analytics settings
APPDYNAMICS_ANALYTICS_PORT	port—in Analytics settings
APPDYNAMICS_ANALYTICS_SSL_ENABLED	ssl—in Analytics settings
UNIQUE_HOST_ID	uniqueHostId The unique host ID can be set to any value that will identify the host. If a value is not specified, the agent will generate the host ID. This environment variable is available by default (not for Java proxy mode).

Install the Node.js Agent

Related pages:

- Download AppDynamics Software
- Install the Node.js Agent (Java Proxy)

This page provides instructions on how to install the Node.js Agent.

The Getting Started Wizard in the Controller provides the easiest way to get started with the AppDynamics Node.js Agent. The wizard constructs a pre-configured require statement for the agent based on your input.

Before You Begin

You must add a require statement to the source code of your Node.js application to install the Node.js Agent. In addition to running the AppDynamics Controller, you need to have write access to the application source code and the ability to restart the Node.js application, which installs the agent.

The Getting Started Wizard in the Controller generates the require statement for you. It populates the statement with the connection settings for the Controller and the values you provide for the wizard to model the Node.js application in AppDynamics.

For installation on Windows, see Supported Environment for your version of Windows and install the Visual C++ Redistributable for Visual Studio 2015.

About Node.js Agent Node Identity

You configure the identify of a Node.js application instance using the node name setting. By default, the agent uses the value configured for `nodeName` as a prefix and adds a dash and number as a suffix. For example, given the following setting:

```
nodeName=MyNode
```

The first node to start with this configuration would be named `MyNode-0`. Auto-numbered suffixes on node names are most useful for machines with more than one worker process.

If you are not running multiple worker processes, you can prevent auto-numbering in node names by setting `noNodeNameSuffix` to `true`. In the following example, the node would be named `MyNode`.

```
nodeName=MyNode
noNodeNameSuffix=true
```

If you are instrumenting worker processes on different machines, keep in mind that each application and node name combination must be unique. Therefore, be sure to specify different node name prefixes for each server, for example, by configuring `nodeName=Server1` for the first server and `nodeName=Server2` for the second server.

If you are instrumenting a Node.js application that uses the PM2 process manager, set the node suffix name to `process.env.pm_id`.

Use the Node.js Agent with the Machine Agent

If you install the Machine Agent on the machine hosting the instrumented Node.js node and you specify the tier and node name in the `controller-info.xml` file of the Machine Agent, the Node.js Agent will fail to register.

To avoid this problem:

- Install the Node.js Agent before you install the Machine Agent.
- If you install the Machine Agent on the machine hosting the instrumented Node.js node, do not specify the application, tier, or node name in the machine agent's `controller-info.xml` file. If you do, the Node.js Agent may fail to register.

Install the Node.js Agent

To install the agent, run the install command in the directory of your application and add a `require` command to add the agent module to your application.

If you are using Node.js 0.8.1 through 0.8.18, see Set User Agent for Node.js 0.8.1 through 0.8.18 before running the install command.

Install via `npm`

Refer to Agent and Controller Compatibility to determine which versions of the Node.js Agent are compatible with your Controller.

To install the latest agent, run the following command:

```
npm install appdynamics@next
```

If you know which specific version of the Node.js agent you want to install, you can specify it:

```
npm install appdynamics@<x.y.z>
```

If you are using npm 5, you must disable the lock file by setting `package-lock=false` in the npm configuration settings.

The version of npm that ships with Node.js v6 does not automatically run post-install scripts used by 4.5.12 and later versions of the agent. AppDynamics recommends using the latest Node.js version.

If Node.js v6 must be used, npm needs to be updated prior to installing the agent:

```
npm install -g npm
npm install appdynamics
```

Add the Require Statement

Paste the following require statement as the very first line of your application source code before any other require statement. Then, replace the variables with the values for your setup. To find your account name and access key, select **Settings** and then **License**.

```
require("appdynamics").profile({
  controllerHostName: '<controller host name>',
  controllerPort: <controller port number>,
  controllerSslEnabled: false,  // Set to true if controllerPort is SSL
  accountName: '<AppDynamics_account_name>',
  accountAccessKey: '<AppDynamics_account_key>', //required
  applicationName: 'your_app_name',
  tierName: 'choose_a_tier_name',
  nodeName: 'choose_a_node_name'
 });
```

For reference information on the settings, along with other settings you can use, see Node.js Settings Reference.

You can place the `require` statement as the first line in the require statement of another module that appears as the first line of code.

In this case, you would need to modify your point-of-entry source file; this it can be just a single line to the `require()` the file that you place the call to the agent into; for example, `require("<script-that-initializes-the-agent>")`. You could also parameterize the `profile()` call to name different instances without having to have multiple versions of the agent initialization script.

If it is not possible to place the require statement as the first line of code, you can insert the statement elsewhere, but it must occur before the `require()` of any core or third party module that needs to be instrumented. In general, the `require("appdynamics")` statement should occur as early as possible in the code.

Test the Configuration

To verify the installation, restart the application and put load on it. The new node should appear in the flow map for the business application you specified in the configuration.

Stop the Node.js Application

By default, closing the Node.js application stops the application.

Run with Other Profiling Tools

The agent is incompatible with other profiling tools, such as running the node process with the `--inspect` flag.

Instrument a Node.js Cluster

If your application uses the cluster module, place the `appdynamics.profile require` statement in both the primary and worker processes.

Set User Agent for Node.js 0.8.1 through 0.8.18

If running a Node.js version between 0.8.1 and 0.8.18, inclusive, you need to set the `user-agent npm` property before you run the `npm install` command.

The syntax is:

```
npm config set user-agent "node/<version> {linux|darwin} {x64|i86}"
```

For example:

```
npm config set user-agent “node/v0.8.14 linux x64"
```

Resolve Installation Issues for Node.js

If you are trying to install the Node.js Agent on a version of Node.js that the agent does not support, the installation will fail and you will see a message similar to the following:

```
npm ERR! notsup Unsupported
npm ERR! notsup Not compatible with your version of node/npm: appdynamics@3.9.3
npm ERR! notsup Required: {"node":">=0.8.0 <=0.10.31"}
npm ERR! notsup Actual:   {"npm":"1.4.28","node":"0.10.32"}
...
```

See Node.js Supported Environments.

Install the Node.js Agent in Containers

Related pages:

- Container Installation Options

This page explains your options to install the Node.js Agent in a containerized environment:

- Use Auto-Instrumentation
- Use Init Containers
- Use a Dockerfile

Use Auto-Instrumentation

> This scenario applies to containers running in Kubernetes where the Cluster Agent is installed.

This option uses the Auto-Instrumentation feature of the Cluster Agent. It is the recommended option because it offers the highest level of automation and the simplest operational experience for instrumenting Node.js applications in a Kubernetes cluster.

To get started, see Auto-Instrument Applications with the Cluster Agent.

Use Init Containers

> This option applies to containers running in Kubernetes.

This option uses Kubernetes init containers to copy the Node.js Agent binaries into the application container when the application starts up. It assumes that the application image is built without any Node.js Agent binaries and that a second init container image is built which contains the Node.js Agent binaries. The deployment spec for the application is configured to copy and send the agent binaries to the running application container.

To copy the agent binaries with init containers:

1. Build the Node.js Application Image
2. Build the Node.js Agent Init Container Image
3. Add the Init Container to the Deployment Spec
4. Set the Node.js Agent Environment Variables
5. Set the `UNIQUE_HOST_ID` Environment Variable
6. (On-Premises Controller Only) Copy the Controller Certs to the Container

> These steps assume the application image is built using Node.js version 8 and later, which supports the `NODE_OPTIONS` environment variable. This environment variable is set in a ConfigMap. For Node.js versions 7 and earlier, refer to this example that uses a start script to include the a `ppdynamics` package.

Build the Node.js Application Image

Build the Node.js application image based on the application binaries and dependencies. Do not include the Node.js Agent binaries or settings in the application image.

Build the Node.js Agent Init Container Image

The agent init container image is built separately from the application image and can be reused across multiple Node.js application deployments.

> For Linux and Alpine environments, the Node.js Agent init container images are available on Dockerhub. For all other deployment environments, continue with the following instructions to build the init container image.

The Node.js and operating system versions of the init container image should match the Node.js and operating system versions of the application image to ensure that compatible Node.js Agent binaries are selected when running `npm install`. For more information, see Node.js Supported Environments.

The Node.js and operating system versions are determined by the `FROM node:14.4-alpine` statement in the Dockerfile:

```
FROM node:14.4-alpine
…
```

Follow these steps to build the init container image:

1. Create a `shim.js` file that contains a `require` statement for `appdynamics`. Set the `reuseNode` property to `true`. Set `reuseNodePrefix` to the value of an environment variable that will be provided in a ConfigMap in a later step:

```
require("appdynamics").profile({
  reuseNode: true,
  reuseNodePrefix: process.env.APPDYNAMICS_AGENT_REUSE_NODE_NAME_PREFIX
});
```

2. If the Node.js Agent should report Transaction Analytics data, add the analytics properties to the require statement and set the value to environment variables that will be provided in a ConfigMap in a later step. See Node.js Settings Reference.

```
require("appdynamics").profile({
  reuseNode: true,
  reuseNodePrefix: process.env.APPDYNAMICS_AGENT_REUSE_NODE_NAME_PREFIX,
  analytics: {
    host: process.env.APPDYNAMICS_ANALYTICS_HOST_NAME,
    port: process.env.APPDYNAMICS_ANALYTICS_PORT,
    ssl: process.env.APPDYNAMICS_ANALYTICS_SSL_ENABLED
  }
});
```

3. If the Node.js Agent communicates with an on-premises Controller, add the `certificateFile` property and set it to the location where the `cert` file will be copied in a later step:

```
require("appdynamics").profile({
  reuseNode: true,
  reuseNodePrefix: process.env.APPDYNAMICS_AGENT_REUSE_NODE_NAME_PREFIX,
  certificateFile: /opt/appdynamics/cacerts
});
```

4. In the Dockerfile, copy `shim.js` to the image:

```
COPY ./shim.js /opt/appdynamics/shim.js
```

5. Run the `npm install` command to resolve the `appdynamics` package:

```
$ cd appdynamics
$ npm install appdynamics@next
```

6. In the Dockerfile, copy the contents of the folder containing the `appdynamics` package to the image:

```
COPY appdynamics/ /opt/appdynamics/
```

For a complete example of building the Node.js Agent init container image using a multi-stage build, see this Dockerfile.

Add the Init Container to the Deployment Spec

Edit the deployment spec to add the sections required to copy the agent binaries from the init container to the application image.

The following snippet from a deployment spec shows the required `volumes`, `volumeMounts`, and `initContainer` definitions. It assumes the Node.js application image is published to `myrepo/nodejs-app:v1` and the init container image is published to `myrepo/appd-nodejs:latest`. The `shim.js` file from the init container image is copied to `/opt/appdynamics/shim.js` in the application container.

```
kind: Deployment
spec:
  containers:
  - name: nodejs-app
        image: myrepo/nodejs-app:v1
    volumeMounts:
    - mountPath: /opt/appdynamics
      name: appd-agent-repo
  initContainers:
  - command:
    - cp
    - -r
    - /opt/appdynamics/.
    - /opt/temp
       name: appd-agent
    image: myrepo/appd-nodejs:latest
    volumeMounts:
    - mountPath: /opt/temp
      name: appd-agent-repo
  volumes:
    - name: appd-agent-repo
      emptyDir: {}
```

Set the Node.js Agent Environment Variables

To set all of the required Node.js Agent environment variables, try the following options depending on your use case:

- Use ConfigMaps to Configure the App Server Agent
- Use Secrets for the Controller Access Key
- Set Application-Specific Configuration in the Deployment Spec

See Best Practices to Configure Agents in Kubernetes for more information.

Use ConfigMaps to Configure the App Server Agent

1. Use a ConfigMap to set the agent environment variables that are shared across Node.js applications in a namespace. For example, see this `appd-nodejs-config.yaml` snippet:

   ```
   apiVersion: v1
   data:
     APPDYNAMICS_AGENT_APPLICATION_NAME: eCommerce
     APPDYNAMICS_AGENT_ACCOUNT_NAME: <value>
     APPDYNAMICS_CONTROLLER_HOST_NAME: <value>
     APPDYNAMICS_CONTROLLER_PORT: <value>
     APPDYNAMICS_CONTROLLER_SSL_ENABLED: <value>
     APPDYNAMICS_AGENT_REUSE_NODE_NAME_PREFIX: <value>
     APPDYNAMICS_AGENT_NODE_NAME: <value> # not used in node name but required by Node.js agent
     APPDYNAMICS_LOGGER_OUTPUT_TYPE: console
     NODE_OPTIONS: '--require /opt/appdynamics/shim.js'
     # variables required to send transaction analytics data
     APPDYNAMICS_ANALYTICS_HOST_NAME: <value>
     APPDYNAMICS_ANALYTICS_PORT: <value>
     APPDYNAMICS_ANALYTICS_SSL_ENABLED: <value>
   kind: ConfigMap
   metadata:
     name: appd-nodejs-config
   ```

   The `NODE_OPTIONS` environment variable, supported in Node.js 8 and later, sets the `--require` option to the location of the `shim.js` file copied from the init container image. The analytics host, port and ssl settings depend on how the Analytics Agent is deployed. See Deploy Analytics in Kubernetes for options.

2. Apply the ConfigMap to the namespace:

   ```
   $ kubectl -n ecommerce apply -f appd-nodejs-config.yaml
   ```

3. Update the deployment spec to reference the ConfigMap:

```
spec:
  containers:
  - name: nodejs-app
    envFrom:
    - configMapRef:
      name: appd-nodejs-config
 ...
```

Use Secrets for the Controller Access Key

1. Create a Secret using `kubectl`:

```
$ kubectl -n ecommerce create secret generic appd-agent-secret --from-literal=access-key=<access-key>
```

2. Update the deployment spec to reference the Secret:

```
spec:
  containers:
  - name: nodejs-app
    env:
    - name: APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY
      valueFrom:
      secretKeyRef:
        name: appd-agent-secret
        key: access-key
 ...
```

Set Application-Specific Configuration in the Deployment Spec

Set the application-specific tier name environment variable `APPDYNAMICS_AGENT_TIER_NAME` in the deployment spec:

```
spec:
  containers:
  - name: nodejs-app
    env:
    - name: APPDYNAMICS_AGENT_TIER_NAME
      value: nodejs-service
```

Set the `UNIQUE_HOST_ID` Environment Variable

Set the `UNIQUE_HOST_ID` environment variable to enable APM correlation with the Cluster Agent. Since the value depends on a runtime value, set this environment variable in the container startup command with the values in Manually Configure App Agents to Correlate with the Cluster Agent.

For a Kubernetes environment with the Docker runtime, set the environment variable as shown in the following example:

```
spec:
  containers:
    command: ["/bin/sh"]
    args: ["-c", "export UNIQUE_HOST_ID=$(sed -rn '1s#.*/##; 1s/(.{12}).*/\\1/p' /proc/self/cgroup) && node
/nodejsapp/myapp.js"]
```

(On-Premises Controller Only) Copy the Controller Certs to the Container

If on-premises Controller certs are required, define a ConfigMap to reference the `cert` file and use a volume mount in the deployment spec to mount the ConfigMap contents to the container. See Controller SSL and Certificates.

```
$ kubectl create configmap appd-cert --from-file=cacerts
```

Add the cert file to the container file system using `volumes` and `volumeMounts` as shown in the deployment spec snippet:

```
kind: Deployment
spec:
  containers:
    image: myrepo/nodejs-app:v1
    volumeMounts:
    - name: appd-cert
      subPath: cacerts
      mountPath: /opt/appdynamics/cacerts
  volumes:
    - name: appd-cert
      configMap:
        name: appd-cert
```

The `mountPath` value should match the `certificateFile` property in the `shim.js` file used when building the init container image.

Example Configuration for Using an Init Container

See this deployment spec for a complete example using an init container to copy the agent binaries.

Use a Dockerfile

> This option applies to containers running in Docker and Kubernetes.

This option uses a Dockerfile to include the Node.js Agent binaries in the Docker image at build time. It assumes a single image is built that contains the application and Node.js Agent binaries.

To include the agent in the application image during the image build:

1. Perform the Node.js Agent Installation Steps
2. Copy the Application Folder to the Image
3. Set the Node.js Agent Environment Variables
4. Set the UNIQUE_HOST_ID Environment Variable
5. (On-Premises Controller only) Copy the Controller Certs to the Image

Perform the Node.js Agent Installation Steps

Perform the installation steps described in Install the Node.js Agent.

1. Run the NPM command to include the `appdynamics` package from your application folder:

   ```
   $ cd nodejsapp
   $ npm install appdynamics@next
   ```

2. Add the `require` statement to the application source code and include `reuseNode` and `reuseNodePrefix` properties. Set the `reuseNodePrefix` property to refer to an environment variable that will be set in a later step:

   ```
   require("appdynamics").profile({
     reuseNode: true,
     reuseNodePrefix: process.env.APPDYNAMICS_AGENT_REUSE_NODE_NAME_PREFIX
   });
   ```

   The reuse node name and prefix properties are required to support unique naming for multiple container instances for the same application image. See Node.js Settings Reference.

3. If the Node.js Agent should report transaction analytics data, add the analytics properties to the require statement and set the value to an environment variable that will be set in a later step. See Node.js Settings Reference.

```
require("appdynamics").profile({
  reuseNode: true,
  reuseNodePrefix: process.env.APPDYNAMICS_AGENT_REUSE_NODE_NAME_PREFIX,
  analytics: {
    host: process.env.APPDYNAMICS_ANALYTICS_HOST_NAME,
    port: process.env.APPDYNAMICS_ANALYTICS_PORT,
    ssl: process.env.APPDYNAMICS_ANALYTICS_SSL_ENABLED
  }
});
```

Copy the Application Folder to the Image

Edit the Docker file to copy the application folder to the image:

```
COPY nodejsapp/ /nodejsapp/
```

Set the Node.js Agent Environment Variables

If you are running the application in a non-Kubernetes environment (using `docker run` for example), set the agent environment variables in the Dockerfile. For example:

```
ENV APPDYNAMICS_AGENT_APPLICATION_NAME=<value>
ENV APPDYNAMICS_AGENT_TIER_NAME=<value>
ENV APPDYNAMICS_AGENT_ACCOUNT_NAME=<value>
ENV APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY=<value>
ENV APPDYNAMICS_CONTROLLER_HOST_NAME=<value>
ENV APPDYNAMICS_CONTROLLER_PORT=<value>
ENV APPDYNAMICS_CONTROLLER_SSL_ENABLED=<value>
ENV APPDYNAMICS_AGENT_REUSE_NODE_NAME_PREFIX=<value>
ENV APPDYNAMICS_AGENT_NODE_NAME=<value> # not used in node name but required by Node.js agent
ENV APPDYNAMICS_LOGGER_OUTPUT_TYPE=console
# variables required to send transaction analytics data
ENV APPDYNAMICS_ANALYTICS_HOST_NAME=<value>
ENV APPDYNAMICS_ANALYTICS_PORT=<value>
ENV APPDYNAMICS_ANALYTICS_SSL_ENABLED=<value>
```

For Kubernetes applications, omit these environment variables from the Dockerfile and set them using ConfigMaps, Secrets, and the deployment spec as described in Use Init Containers. The analytics host, port and ssl settings depend on how the Analytics Agent is deployed. See Deploy Analytics in Kubernetes for options.

Set the `UNIQUE_HOST_ID` Environment Variable

Set the `UNIQUE_HOST_ID` environment variable to enable APM correlation with the Cluster Agent. Since the value depends on a runtime value, set this environment variable in the image startup script using the values documented in Manually Configure App Agents to Correlate with the Cluster Agent.
For a Kubernetes environment with a Docker runtime, add the following startup script `startup.sh` to the Docker image in the following example:

```
#!/bin/bash
# OpenShift 3.10 or 3.11:
UNIQUE_HOST_ID=$(sed -rn '1s#.*/##; 1s/(.{12}).*/\\1/p' /proc/self/cgroup)
exec node /nodejsapp/myapp.js
```

(On-Premises Controller only) Copy the Controller Certs to the Image

For Node.js Agents communicating with an on-premises Controller, edit the Dockerfile to copy the cert file containing the on-premises certs to the image.

For example:

```
COPY ./onprem-cacerts /opt/appdynamics/cacerts
```

Update the application source code to add the `certificateFile` property in the `require` statement. Set the `certificateFile` path to the location of the cert file.

```
require("appdynamics").profile({
  reuseNode: true,
  reuseNodePrefix: process.env.APPDYNAMICS_AGENT_REUSE_NODE_NAME_PREFIX,
  certificateFile: /opt/appdynamics/cacerts
});
```

Node.js Agent API User Guide

Related pages:

- Node.js Agent API Reference
- Node.js Supported Environments
- Backend Detection Rules

The AppDynamics Node.js Agent APIs enable the Node.js Agent to monitor business transactions that a web application performs outside the context of an HTTP request or an application that is not web-based.

Business Transaction Definition

You define a custom business transaction using `appd.startTransaction()` and `end()`.

If the custom transaction runs for a long time or if it crosses an asynchronous boundary, you can use `resume()` to rejoin the running transaction.

A downstream transaction in a distributed transaction cannot start until its upstream transaction has made its exit call.

The order of transactions respects the nonlinear nature of Node.js applications. An upstream transaction might end immediately after its exit call completes or sometime later. It is not necessary for an upstream transaction to end before or after its downstream transaction starts or ends. It is possible for a downstream transaction to end before its upstream transactions have ended.

Marking a Transaction as an Error Transaction

An error transaction is one in which an exception occurred. It is reported as an error transaction and not as a slow, very slow or stalled transaction, even if the transaction in which the error occurred was also slow or stalled.

To attach a JavaScript error object to a transaction using the Node.js APIs do one of the following:

- Use `trx.markError()` passing in the JavaScript error object.
- Attach the error object to the HTTP response associated with the transaction as response.error.

When an error is marked, the status code defaults to 500, unless you set it using one of the following techniques:

- Use `trx.markError()` passing in the JavaScript error object and the optional status code.
- Set the `statusCode` property in the error object passed to `trx.markError()` as `error.StatusCode`.
- Set the `statusCode` property on the `response.error` object as `response.error.StatusCode`.

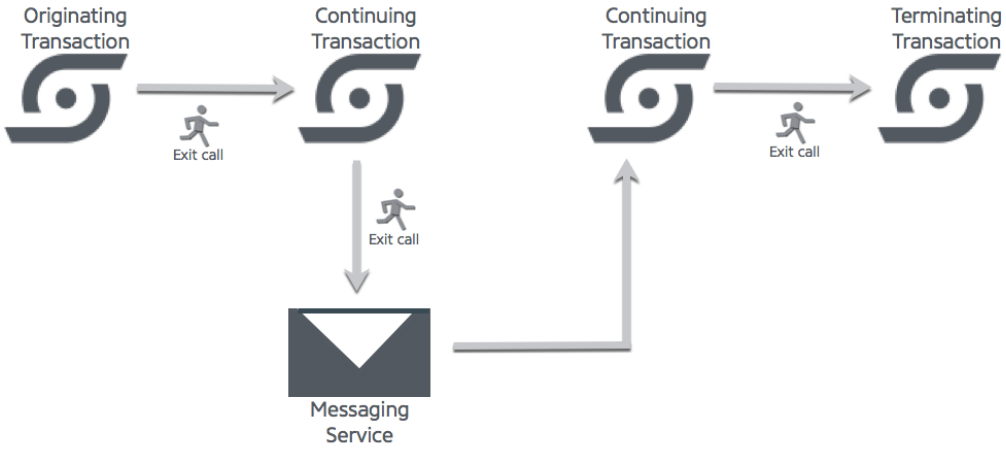
Transaction Correlation Management

Transaction correlation is the functionality that maintains the business transaction context across all tiers—servers—traversed by a distributed transaction. The tiers may be built on platforms other than Node.js. For example, Java, .NET, and PHP tiers may participate in a distributed business transaction in which a Node.js tier participates, as originating, continuing or terminating tiers in the transaction.

The Node.js Agent APIs provide facilities for managing transaction correlation among various transactions, which may be custom or automatically-detected.

While the AppDynamics default auto-detection mechanism obtains transaction correlation information from an incoming HTTP request, the Node.js Agent APIs allow you to obtain this information from other sources, such as a custom field in a Redis cache entry, and to supply it to your custom transactions. You can use the APIs to support forward correlation by supplying transaction correlation information on exit calls that you make to downstream transactions.

A downstream transaction starts after its immediate upstream transaction makes its exit call. That call could be a direct exit call to the next tier or a call to a backend service that does some processing, such as publishing a message onto a queue.



.

Exit Call Management

An exit call is either a custom exit call or an automatically-detected exit call. A transaction needs its exit call object to create correlation information to provide to a downstream transaction that needs to correlate with it.

Custom Exit Calls

If you want an exit call to be reported to the controller, and it is not automatically detected by the agent, create a custom exit call with `startExitCall()` and end it with `endExitCall()`.

See Node.js Supported Environments for the list of backends that are automatically detected by the Node.js Agent. If the exit call invokes a backend not listed on this page, you probably need to create a custom exit call.

You can also use a `beforeExitCall()` callback function to check for the existence of an automatically-detected exit call.

Processing a custom exit call is straightforward. First, create the exit call and then pass its returned `ExitCall` object to `createCorrelationInfo()` to create a string-encoded correlation information object that a downstream transaction can fetch later.

Workflow for a custom exit call

```
// start a custom transaction
var trx = appd.startTransaction(. . . )
. . .

// start a custom exit call
exit = trx.startExitCall(. . .);

// create correlation information
cinfo = trx.createCorrelationInfo(exit)
//store cinfo where it can be made available to a downstream transaction

// make the exit call
. . .
trx.endExitCall(. . .)
```

Automatically Detected Exit Calls

The code executing in your custom transaction might make an exit call that is automatically detected by the Node.js Agent.

In this case, you can provide a `beforeExitCall()` callback function to process that exit call. When your code makes an automatically-detected exit call, the agent intercepts the exit call request and invokes the callback, if you have provided one. Then it allows the application to proceed with the exit call request.

You would install a `beforeExitCall()` callback function for the following purposes:

- To test for the existence of an automatically-detected exit call
  If you are uncertain that the exit call will be automatically detected, you can install a callback function to test for the exit call. If the callback is not invoked, the exit call is not automatically detected.

```
// start a custom transaction
var trx = appd.startTransaction(. . . )
. . .

var detected = false;
trx.beforeExitCall = function(call) { detected = true; return call; };

// make the exit call
. . .

// test whether the exit call was automatically detected
if (detected == false)
. . .
```

If `detected` is `false` after the exit call, you need to create a custom exit call to report the exit call to the Controller. If `detected` is `true`, you can capture the exit call and create the correlation information with the callback.

- To capture the ExitCall object to create correlation information

```
// start a custom transaction
var trx = appd.startTransaction(. . . )
. . .

trx.beforeExitCall = function getExitInfo(exitCall) {
    // create the correlation header from the captured exit call
    var c = trx.createCorrelationInfo(exitCall);
    // store c somewhere
    // … (c)
    return exitCall;
}
// make the exit call
```

- To modify the exit call that is reported to the Controller

```
// start a custom transaction
var trx = appd.startTransaction(. . . )
. . .

trx.beforeExitCall = function customExitCallHandler(exitCall) {
    // don't report database access for this transaction
    if (exitCall.isSql) return;
    // customize label for all other exit calls in this transaction
    exitCall.label += " (my custom transaction)";
    return exitCall;
}
// make the exit call
```

Workflow: Create Originating Transaction with Correlation Information

In this scenario, you create a custom originating transaction and the correlation header that a downstream transaction will need to correlate with it.

1. Start a custom transaction: `var trx=appd.startTransaction(...)`.
2. Either start a custom exit call using `e=trx.startExitCall(einfo)` or capture a detected exit call by installing a callback function to `e=trx.beforeExitCall(e)`.
3. Create correlation information from the exit call: `c=trx.createCorrelationInfo(e)`.
4. Store the correlation information.
5. Make the exit call.
6. Optionally provide a `trx.exitCallCompleted(e)` callback function, if you want to modify what is reported to the controller based on information obtained from the execution of the exit call.
7. If you created a custom exit call, end the exit call after you have processed the exit call response: `trx.endExitCall(e)`.
8. End the transaction: `trx.end()`.

Workflow: Create Terminating Transaction Correlated with Upstream Transaction

In this scenario, you create a custom transaction, which needs to correlate with an upstream transaction but not with another downstream transaction.

1. Retrieve the correlation information stored by the previous transaction.
2. Parse the retrieved correlation information into a correlation header: `ch=appd.parseCorrelationInfo(c)`.
3. Start a custom transaction using the correlation header: `trx=appd.startTransaction(ch)`.
4. Optionally make an exit call.
5. End the transaction: `trx.end()`.

Workflow: Create Continuing Transaction Correlated with Upstream and Downstream Transactions

In this scenario, you create a continuing transaction that needs to correlate both with an upstream originating transaction and a downstream transaction. It uses the correlation techniques described in both of the previous workflows.

1. Retrieve correlation information stored by the previous transaction.
2. Parse the retrieved correlation information into a correlation header: *c*`h=appd.parseCorrelationInfo(c1)`.
3. Start a custom transaction using the correlation header: `trx=appd.startTransaction(ch`*)*.
4. Either start a custom exit call using `e=trx.startExitCall(einfo)`, or capture a detected exit call by installing a callback function to `e=trx.beforeExitCall(e)`.
5. Create correlation information from the exit call in this transaction: `c2=trx.createCorrelationInfo(e)`.
6. Store the correlation information for the next downstream transaction.
7. Make the exit call.
8. Optionally provide a `trx.exitCallCompleted(e)` callback function, if you want to modify what is reported to the controller based on information obtained from the execution of the exit call.
9. If you created a custom exit call, end the exit call after you have processed the exit call response: `trx.endExitCall(e)`.
10. End the transaction: `trx.end()`.

Node.js Agent API Reference

Related pages:

- Node.js Agent API User Guide
- Backend Detection Rules

This page describes the Node.js Agent API. The Node.js Agent APIs include methods to manage business transactions and exit points in Node.js applications.

API Call List

Loading the AppDynamics Module

You can use the APIs in your application by adding the `appdynamics` Node.js module in a require statement, after `appd.profile()`.

```
var appd = require('appdynamics');
appd.profile({ . . . })

...

var transaction = appd.startTransaction(...);
...


transaction.end();
```

Basic Business Transaction Management

appd.startTransaction(transactionInfo)

- `transactionInfo` string, HttpRequest or `CorrelationHeader`

The `transactionInfo` parameter can be one of the following:

- A string specifying the name of the custom transaction. The following characters cannot be used in transaction names: `{`, `}`, `[`, `]`, `|`, `&`, and `;`.
- A `HttpRequest` object. Normal transaction matching and naming rules are applied, and any included RUM and/or transaction correlation information is picked up automatically.
- A `CorrelationHeader` object providing information to correlate with a transaction that is upstream to this one. See parseCorrelationInfo().

Creates and starts a custom business transaction and returns a transaction handle as a `TimePromise` instance, which is used for subsequent operations on this transaction.

TimePromise.prototype.resume()

Re-joins a long-running or asynchronous transaction.

Call `resume()`:

- When a long-running transaction needs to be kept alive beyond the default transaction timeout of five minutes. This prevents the agent from assuming the transaction has failed and then deleting the transaction.
- When the code being instrumented crosses an async boundary. This re-associates the current execution context with the correct transaction and ensures any exit calls performed are charged to the right transaction.

TimePromise.prototype.end(error)

- `error:` any; optional

Ends the transaction and reports transaction information to the Controller.

If an `error` is passed, the error is associated with the transaction and the transaction is flagged as an error transaction.

appd.getTransaction(request)

- `request:` HTTP request object

Returns a handle to the business transaction associated with the specified request, if one exists. If no associated business transaction exists, returns false.

Use `appd.getTransaction()` to access both custom and automatically detected business transactions.

appd.addAnalyticsData (key, value)

- `key:` key to custom data being added
- `value:` value being added for this key

Attaches custom data to Analytics generated for the current transaction. Your application does not require a transaction handle.

To enable this feature, the Analytics settings in the require statement must be set. See Node.js Settings Reference. For information about Analytics, see Analytics Data Sources.

appd.addSnapshotData (key, value)

- `key:` key to custom data being added
- `value:` value being added for this key

Attaches custom data to snapshots generated for the current transaction.

`addSnapshotData()` can be called on a transaction at any time, but the added data is used only when a transaction snapshot is generated. For information on when transaction snapshots are generated, see Transaction Snapshots.

The following sample adds session count to USER DATA in the snapshot:

```
txn.addSnapshotData(“active sessions”, my.api.getSessionCount());
```

If the data to be added to the snapshot is available after the transaction completes, it can be attached using a txn.onSnapshotCaptured(txn) callback. This technique avoids overhead for collecting the data when no snapshot has been generated. The custom data is exposed in the **USER DATA** tab of the transaction snapshot details in the Controller UI.

appd.markError(error, [statusCode])

- `error:` JavaScript error object
- `statusCode:` optional transaction status code

This value can also be passed as a statusCode property on the `error` object.

If neither a `statusCode` parameter nor an `error.statusCode` property in the `error` parameter is provided, the status code defaults to 500.

See also TimePromise.prototype.end(error) for another way to attach an error object to a business transaction.

Marks the transaction as an error transaction. The marked transaction is reported as an error transaction and not as a slow, very slow or stalled transaction, even if the transaction was also slow or stalled.

txn.markError(error, [statusCode])

- `error` JavaScript error object
- `statusCode` optional transaction status code. This value can also be passed as a `statusCode` property on the `error` object.

If neither a `statusCode` parameter nor an `error.statusCode` property in the `error` parameter is provided, the status code defaults to 500.

See also TimePromise.prototype.end(error) for another way to attach an error object to a business transaction.

Marks the transaction as an error transaction. The marked transaction is reported as an error transaction and not as a slow, very slow or stalled transaction, even if the transaction was also slow or stalled.

txn.addSnapshotData(key, value)

- `key:` key to custom data being added
- `value:` value being added for this key

Attaches custom data to snapshots generated for this transaction.

`addSnapshotData()` can be called on a transaction at any time, but the added data is used only when a transaction snapshot is generated. For information on when transaction snapshots are generated, see Transaction Snapshots.

The following sample adds session count to USER DATA in the snapshot.

```
txn.addSnapshotData(“active sessions”, my.api.getSessionCount());
```

If the data to be added to the snapshot is available after the transaction completes, it can be attached using a txn.onSnapshotCaptured(txn) callback. This technique avoids overhead for collecting the data when no snapshot has been generated. The custom data is exposed in the **USER DATA** tab of the transaction snapshot details in the Controller UI.

txn.onSnapshotCaptured(txn)

- `txn`: transaction

Callback method that can be set on a transaction to add custom data to the transaction snapshot. Used with txn.addSnapshotData(key, value).

When the `callback` function is supplied, the agent invokes this API immediately after capturing a transaction snapshot.

If specified, the `callback` is invoked on transaction completion, if the transaction triggers a snapshot.

You must define a function to call and assign it to the callback.

```
mytxn.onSnapshotCaptured = customTitle (txn) {
    // get book title for current transaction instance
    title = getBookTitle();
    txn.addSnapshotData ("book title", title);
}
```

Note that `txn === mytxn`. It is passed into the callback so the callback does not need to be defined in the same scope as `mytxn`.

txn.addAnalyticsData(key, value)

- `key:` key to analytics data being added
- `value:` value being added for this key

Attaches custom data to Analytics generated for this transaction. If your application has a transaction handle, you can call `txn.addAnalyticsData (key, value)`. If not, use `appd.addAnalyticsData(key, value)`.

To enable this feature, the Analytics settings in the require statement must be set. See Node.js Settings Reference. For more information about Analytics, see Analytics Data Sources.

txn.onResponseComplete(req, res)

- `req`: HTTP request for the transaction
- `res`: HTTP response for the transaction

Callback method that can be set on a transaction to be notified when the associated HTTP response has been completed. This can be used with `txn.addAnalyticsData(key, value)` to add data from response after it is complete, before the transaction completes. When the callback function is supplied, the agent invokes this API immediately after the HTTP response completes, before closing out the transaction. You must define a function to call and assign it to the callback.

```
mytxn.onResponseComplete = function customTitle (req, res) {
    // get status code of HTTP response
    code = res.statusCode;
    txn.addAnalyticsData ("http response code", code);
};
```

Exit Call Management

exitCall.addAnalyticsData (key, value)

- `key:` key to analytics data being added
- `value:` value being added for this key

Attaches existing Analytics data generated for this transaction to the exit call.

Attaches custom data to Analytics generated for this transaction. If your application has a transaction handle, you can call `txn.addAnalyticsData (key, value)`. If not, use `appd.addAnalyticsData(key, value)`.

To enable this feature, the Analytics settings in the require statement must be set. See Node.js Settings Reference.

TimePromise.prototype.startExitCall(exitCallInfo)

- `exitCallInfo` object

  The `exitCallInfo` object has the following required properties:

  - `exitType`: string—the type of exit call, one of "`EXIT_HTTP`", "`EXIT_DB`", "`EXIT_CACHE`", "`EXIT_RABBITMQ`", "`EXIT_WEBSERVICE`"
  - `label`: string—label for the exit call in the AppDynamics UI
  - `backendName`: string—name of the backend remote system or service
  - `identifyingProperties`: Object—a hash of name/value pairs uniquely identifying the remote system or service being called. See Backend Identifying Properties.

Creates and starts a custom exit call, described by the `exitCallInfo` object.

You can supply more identifying properties in addition to the required ones.

Returns an `ExitCall`. See Exit Call Properties.

Sample exitCallInfo Object for MySQL

```
{
  exitType: “EXIT_DB”,
  label: "New SQL",
  backendName: "NewDB"
  identifyingProperties: {
    “HOST”: “database-server”,
    “PORT”: “12345”,
    “DATABASE”: “my-database”,
    "VENDOR" = “MYSQL"
  },
  …etc...
}
```

TimePromise.prototype.endExitCall(exitCall, error)

- `exitCall`: `ExitCall` instance returned by `startExitCall()`
- error: any; optional

Ends a custom exit call and attaches it to this transaction.

If an `error` is passed, the error is associated with the exit call, and the exit call is flagged as being in an error state.

TimePromise.prototype.beforeExitCall(exitCall)

- `exitCall: ExitCall`

Returns an `ExitCall`.

Optional callback for modifying an automatically-detected (not custom) exit call.

The agent invokes this API immediately before your application makes the exit call when you supply a callback function. You can use the callback to:

- Modify the exit call that is reported to the controller.
- Suppress all reporting of the exit call by returning nothing.
- Capture the exit call to create correlation information for a downstream transaction.

The following is an example of a modify exit call reported to the Controller.

```
txn.beforeExitCall = function customExitCallHandler(exitCall) {
    // don't report database access for this transaction
    if (exitCall.isSql) return;
    // customize label for all other exit calls in this transaction
    exitCall.label += " (my custom transaction)";
    return exitCall;
}
```

TimePromise.prototype.exitCallCompleted(exitCall)

- `exitCall`: ExitCall

Optional. Returns an ExitCall. Callback for modifying either a custom exit call created by `startExitCall()` or an automatically-detected exit call on completion of the exit call.

The agent invokes this API immediately after an exit call completes when you supply a callback function.

Use `exitCallCompleted()` to modify how the exit call is reported to the Controller by returning a modified exit call. Do not modify the backend identifying properties using this callback.

The following code is an example that modifies the exit call reported to the Controller.

```
txn.exitCallCompleted = function customExitCallPostProcessor(exitCall) {
    // report MongoDB read and write operations as distinct exit calls
    if (exitCall.backendName === "MongoDB")
        exitCall.label += " " + exitCall.category;
}
```

Transaction Correlation Management

appd.parseCorrelationInfo(source)

- `source:` String or `HttpRequest`

Describes the upstream transaction with which to correlate. If a `HttpRequest` object is passed, it must have a transaction correlation header attached.

On success, returns a `CorrelationHeader` object that can be used by startTransaction() to create a transaction that is correlated with the upstream transaction described by `source`. Returns `false` if `source` could not be parsed. This can occur if the source is an HTTP request with no correlation header attached or the string parameter is not recognized as a correlation header.

HTTP requests made through the standard `http.request()` API have correlation information added automatically. You can correlate a custom transaction created in response to an HTTP request made from another Node.js process by passing the incoming request as `source`.

For other exit call types, you need to define how correlation information is attached to the originating transaction and retrieved in the downstream transaction.

TimePromise.prototype.createCorrelationInfo(exitCall, doNotResolve)

- `exitCall:` ExitCall
- `doNotResolve:` optional, true|false

The input `exitCall` is one of the following:

- For a custom exit call, value returned from startExitCall().
- For an automatically-detected exit call, input parameter to the beforeExitCall() callback.

Set the `doNotResolve` parameter to `true` if you do not want the backend to be resolved to a tier. It defaults to false, which means that the backend is resolved to the calling tier. You may want to set this for an exit call to a service, such as a messaging queue, that does not have a 1:1 relationship between the consumer and producer of the service. For more information see Resolve Remote Services to Tiers.

Returns a string-encoded correlation header, which a downstream transaction can use to correlate with this transaction.

Exit Call Properties

This table describes the properties of the exit call object for each detected exit call type.

A dash (–) in a table cell indicates that the property is present in a discovered exit call but its format is unspecified. This allows you to set your own values when you create custom exit calls.

N/A indicates that the property is not used with that exit call type. Discovered exit calls may have additional properties not documented here.

Property	HTTP	MySQL	Postgres	MongoDB	Memcached	Redis
`backendName`	`"HTTP"`	`"MySQL"`	`"PostgresSQL"`	`"MongoDB"`	`"Memcached"`	`"Redis"`
`label`	String	String	String	String	String	String
`method`	String	N/A	N/A	N/A	N/A	N/A
`requestHeaders`	Object	N/A	N/A	N/A	N/A	N/A
`responseHeaders`	Object	N/A	N/A	N/A	N/A	N/A
`statusCode`	Integer	N/A	N/A	N/A	N/A	N/A
`category`	`"read"|"write"`	N/A	N/A	`"read"|"write"`	N/A	N/A
`user`	N/A	String	String	N/A	N/A	N/A
`command`	String (URL)*	SQL query executed	SQL query executed	N/A	N/A	N/A
`commandArgs`	N/A	SQL query positional parameters	SQL query positional parameters	N/A	N/A	N/A
isSql	N/A	`true`	`true`	`false`	N/A	N/A
`stackTrace`	--	--	--	--	--	--
`exitType`	`"EXIT_HTTP"`	`"EXIT_DB"`	`"EXIT_DB"`	`"EXIT_DB"`	`"EXIT_CACHE"`	`"EXIT_CACHE"`
`identifyingProperties`	See Backend Identifying Properties and TimePromise.prototype.startExitCall(exitCallInfo).					

* host, port and path; does not include query string.

Backend Identifying Properties

This table lists the identifying properties for the backends that are automatically detected by the Node.js Agent.

In the Controller UI, these properties are visible in the upper right panel of the backend dashboards.

Backend Service	Exit Call Type	Identifying Properties
HTTP Server	`"EXIT_HTTP"`	`"HOST", "PORT"`
MySql Server	`"EXIT_DB"`	`"Host", "Port", "Database", "Vendor" = "MYSQL"`
Postgres Server	`"EXIT_DB"`	`"Host", "Port", "Database", "Vendor" = "POSTGRESQL"`
MongoDB Server	`"EXIT_DB"`	`"Server Pool"`[1]`, "Database", "Vendor" = "MONGODB"`
Memcached	`"EXIT_CACHE"`	`"EXIT_CACHE", "Server Pool"`[1]
Redis Server	`"EXIT_CACHE"`	`"Server Pool"`[2]

[1] "\n" separated list of server addresses in <host>:<port> format.
[2] single server address in <host>:<port> format.

Node.js Agent Logging

Related pages:

- Dynamic Language Agent Proxy
- Node.js Java Proxy Mode

This page explains logging for the default mode of the Node.js Agent.

By default, the Node.js Agent generates log information at the informational level. You can increase/decrease the logging verbosity, change the log file location, and configure other logging settings.

Node.js Agent Logging Location

The log file appears at the following location on the machine where the Node.js Agent runs:

```
/tmp/appd/<hash>/appd_node_agent_<datetime>.log
```

If you are running the Node.js Agent, <hash> is generated by the agent.

In Windows, you can find the tmp directory with `cd %tmp% d` and then look for the <hash> directory.

Configure Node.js Agent Logging

To configure logging, use the logging module configuration in the Node.js require statement. You can change the level, maximum size/number of files, and mode.

The following sample shows the logging settings in default mode:

```
{
...

  logging: {
    'logfiles': [
      {
        'root_directory': '/tmp/appd',
                'filename': 'echo_%N.log',
        'level': 'TRACE',
        'max_size': 5242880,
        'max_files': 10,
        'outputType': 'console'  // Set this parameter if you want to log to STDOUT/STDERR. Omit this parameter
if you want to log to a file.
      }
    ]
  }

...
}
```

The level determines the verbosity of logging output and can contain one of the following values listed from least to most verbose:

- FATAL
- ERROR
- WARN
- INFO
- DEBUG
- TRACE

By default, once the size of the logging files on disk reaches the maximum size, old logs are purged. You can reduce or increase the size, depending on your needs and the disk space available on the machine.

Debug Logging

Instead of the logging configuration above, you can use this shortcut to set DEBUG to true, resulting in debug level logging.

```
debug : true,
```

The location of the debug log file is written to standard output.

See Node.js Agent Settings.

Node.js Agent Node Properties

This reference page contains information about app agent node properties. The properties are listed in alphabetical order.

max-continuing-concurrent-snapshots

A limit for the number of concurrent snapshots continuing on this tier or node. For example, if too much overhead is being taken by snapshots, then tweak this value to reduce the snapshots.

Type:	Integer
Default Value:	200
Platform(s):	Node.js

max-continuing-snapshots-per-minute

A limit for the number of snapshots continuing on this tier or node per minute. For example, if too much overhead is being taken by snapshots, then tweak this value to reduce the snapshots.

Type:	Integer
Default Value:	100
Platform(s):	Node.js

max-error-snapshots-per-minute

A limit for the number of snapshots on this tier or node per minute to capture errors. For example, if too many error snapshots are being seen, then tweak this value to reduce *noise* in the snapshots.

Type:	Integer
Default Value:	5
Platform(s):	Node.js

max-originating-concurrent-snapshots

A limit for the number of concurrent snapshots originating on this tier or node. For example, if too much overhead is being taken by snapshots, then tweak this value to reduce the snapshots.

Type:	Integer
Default Value:	10
Platform(s):	Node.js

max-originating-snapshots-per-minute

A limit for the number of snapshots originating on this tier or node per minute. For example, if too much overhead is being taken by snapshots, then tweak this value to reduce the snapshots.

Type:	Integer
Default Value:	20
Platform(s):	Node.js

Upgrade the Node.js Agent

If you are upgrading both the Controller and agents, first upgrade the Controller and then upgrade the agents.

Also, if you are upgrading multiple agents in your monitored environment, upgrade the agents for the tiers on which business transactions originate last. For more information about this requirement, along with Controller and agent compatibility information, see Agent and Controller Compatibility.

To upgrade the Node.js agent:

1. Stop the Node.js application.
2. From the root directory of the application for which you want to upgrade the agent, uninstall the old version of the agent by entering:

   ```
   npm uninstall appdynamics
   ```

3. From the same directory install the new version by entering:

   ```
   npm install appdynamics@<version>
   ```

   The version is the three-digit version number of the new version that you are installing; for example, 4.1.1.
4. Restart the Node.js application.

Uninstall the Node.js Agent

To uninstall the Node.js Agent:

1. From the application root directory of the application from which you want to uninstall the agent enter:

   ```
   npm uninstall appdynamics
   ```

2. Remove the `"require("appdynamics")"` statement from your Node.js applications.
3. Restart the Node.js application.

gRPC Support for the Node.js Agent

This page describes how to use the HTTP naming rules required to name gRPC Business Transactions.

To form the gRPC URI, you link the `SERVICE_NAME` and the `OPERATION_NAME` resulting in: `URI = SERVICE_NAME/OPERATION_NAME`. Then , the naming rule uses the `URI = SERVICE_NAME/OPERATION_NAME` to generate the gRPC Business Transactions name.

Automatic Naming Rules

To add automatic naming rules:

1. From the **Add Rule** tab, click **Rule Configuration**.
2. Under the **Configure Naming** tab, select which part of the URI to use in the **Transaction Name**:
    a. Click **Use the full URI** to use the full URI in the **Transaction Name**.
    b. Click **Use a part of the URI** to use only a part of the URI in the **Transaction Name**. If you select **Use a part of the URI**, you can only use one or two segments of the URL.
3. Click **Save**.

Custom Naming Rules

To add custom naming rules:

1. From the **Add Rule** tab, click **Rule Configuration**.
2. Under **HTTP Request Match Criteria**, click **+Add**.
3. Click **URI**.
4. Select **Split Transactions Using Request Data**.
5. In the dropdown, you can select the **first few URI segment(s)** or the **last few URI segment(s)** in the **Transaction names**.
    a. You can only use one or two segments of the URL.
6. Click **Save**.

Node.js Java Proxy Mode

Related pages:

- Dynamic Language Agent Proxy

This page provides an overview of the existing Java proxy mode for the Node.js Agent.

Every Node.js Agent on a machine launches a proxy instance, creating a control directory for the proxy based on a unique combination of application, tier, and node names for the agent. The control directory contains the configuration for the agent and the domain control socket, which is what the agent uses to start an AppDynamics node.

If you are running multiple Node.js Agents on a single machine, you can configure them to use a shared Java proxy. Referred to as a "multi-tenant proxy," the shared proxy reduces the overhead of additional proxies and is a required configuration for external process managers, such as PM2.

You can use the Java proxy mode as the Node.js Agent supports the existing Java proxy.

Install the Node.js Agent (Java Proxy)

Related pages:

- Run the Node.js Agent on Windows
- SELinux Installation Issues

This page provides instructions on how to install the Node.js Agent in the Java proxy mode.

The Getting Started Wizard in the Controller provides the easiest way to get started with the AppDynamics Node.js Agent. The wizard constructs a pre-configured require statement for the agent based on your input.

Before You Begin

You must add a require statement to the source code of your Node.js application and install the Node.js Agent. In addition to running the AppDynamics Controller, you need to have write access to the application source code and the ability to restart the Node.js application, which installs the agent.

The Getting Started Wizard in the Controller generates the require statement for you. It populates the statement with the connection settings for the Controller and the values you provide for the wizard to model the Node.js application in AppDynamics.

For installation on Windows, see Supported Environment to determine the appropriate version of Windows and install the Visual C++ Redistributable for Visual Studio 2015.

Install the Node.js Agent

To install the agent, run the install command in the directory of your application and add a `require` command to add the agent module to your application.

If you are using Node.js 0.8.1 through 0.8.18, see Set User Agent for Node.js 0.8.1 through 0.8.18 before running the install command.

Install via `npm`

Refer to Agent and Controller Compatibility to determine which versions of the Node.js Agent are compatible with your Controller.

To install the latest agent in the Java proxy mode, run the following command:

```
npm install --appd_include_java_proxy=true appdynamics
```

To run the Node.js Agent on Windows with the Java proxy, see Run the Node.js Agent on Windows.

Add the Require Statement

Paste the following require statement as the very first line of your application source code before any other require statement. Replace the variables with the values for your setup. To find your account name and access key, select **Settings**  and then **License**.

```
require("appdynamics").profile({
  controllerHostName: '<controller host name>',
  controllerPort: <controller port number>,
  controllerSslEnabled: false,  // Set to true if controllerPort is SSL
  accountName: '<AppDynamics_account_name>',
  accountAccessKey: '<AppDynamics_account_key>', //required
  applicationName: 'your_app_name',
  tierName: 'choose_a_tier_name',
  nodeName: 'choose_a_node_name',
  proxy: true
 });
```

For reference information on the settings, along with other settings you can use, see Node.js Settings Reference.

You can place the `require` statement as the first line in the require statement of another module that appears as the first line of code.

In this case, you would need to modify your point-of-entry source file; this it can be just a single line to the `require()` the file that you place the call to the agent into; for example, `require("<script-that-initializes-the-agent>")`. You could also parameterize the `profile()` call to name different instances without having to have multiple versions of the agent initialization script.

If it is not possible to place the require statement as the first line of code, you can insert the statement elsewhere, but it must occur before the `require()` of any core or third party module that needs to be instrumented. In general, the `require("appdynamics")` statement should occur as early as possible in the code.

Test the Configuration

To verify the installation, restart the application and put load on it. The new node should appear in the flow map for the business application you specified in the configuration.

Stop the Node.js Application

When running the Node.js Agent with the Java proxy, configure the script/mechanism that stops an instrumented Node.js instance to use the SIGTERM signal if possible. SIGKILL (`kill -9`) prevents the resources for the agent's Java proxy from being released.

When running the Node.js Agent in the multi-tenant mode, you must manually shut down the Java proxy after stopping your Node.js application.

Run with Other Profiling Tools

The agent is incompatible with other profiling tools, such as running the node process with the `--inspect` flag.

Instrument a Node.js Cluster

If your application uses the cluster module, place the `appdynamics.profile require` statement in both the primary and worker processes.

Running the Node.js Agent with the Java proxy does not require a manual launch of the proxy. Running the Node.js Agent with the Java proxy and an external process manager (such as PM2) for your application requires a manual launch of the proxy component.

See Sharing a Proxy Among Node.js Agents.

Run the Node.js Agent on Windows

Related pages:

- Install the Node.js Agent
- Start the Proxy Manually for Node.js
- Share a Proxy Among Node.js Agents
- Dynamic Language Agent Proxy
- SELinux Installation Issues

This page provides instructions on how to run the Node.js Agent in the Java proxy mode on a Windows machine.

Proxy Ports

Depending on your Node.js environment on Windows, you may need to configure three ports in the Node.js require statement. These port settings are:

- `proxyCommPort`: Default is `10101`
- `proxyRequestPort`: Default is `10102`
- `proxyReportingPort`: Default is `10103`

All three ports must be inclusive of the 1024—65535 range.

Auto-Launch Windows Environments

The default behavior is for the agent to auto-launch the Java proxy, where `proxyAutolaunchDisabled` is `false`.

If you allow the agent to auto-launch and you have one Node.js process per machine, the agent will automatically use the default port values if they are available. In this case, you do not have to configure these settings.

If the default ports are not available, configure these settings to ports that are available in the require statement in the `require` statement.

If you have multiple Node.js processes on a single machine each reporting to its own proxy, each process needs its own set of ports. In this case, configure unique settings for all three ports in the `require` statement for each process.

Manual-Launch Windows Environments

Some environments have multiple Node.js processes on the same machine, all reporting to a single proxy. In these environments, it is necessary to launch the proxy manually, where `proxyAutolaunchDisabled` is `true` and to configure the three port settings.

The `proxyCommPort` setting in the `require` statement and the `commPort` option to the `runProxy.cmd` must be the same value. The agent and the proxy must use the same port.

The `proxyRequestPort` and the `proxyReportingPort` are two unused TCP ports that the user running the proxy has permission to bind to on Windows.

Each distinct app-tier-node combination on a single machine must use different request and reporting ports. Two processes reporting as the same app-tier-node may use the same request and reporting ports. Set these ports in the `require` statement for each app/tier/node combination.

Execute runProxy Manually on Windows

By default, the proxy component is automatically started when you start the agent. If you have configured the agent to manually launch the proxy, use the following command.

Before any traffic is run on the instrumented server, execute `runProxy.cmd` to start the proxy using the following syntax. If you use the optional `-d` option, it must precede the `-j` option.

```
Usage: runProxy.cmd [-d <proxyDir>] -j <jreDir> [logDir] [commPort]

Options:
  -d <proxyDir> the directory where the proxy is installed; optional, defaults to the directory containing the
runProxy.cmd script
  -j <jreDir> the directory where the JRE is installed; required
  logDir the directory the proxy should log to; optional, defaults to <proxyDir>\logs
  [commPort] port used for the proxy control channel on Windows; defaults to 10101; if specified it must match
the proxyCommPort setting in the require statement
```

The following example assumes that the command is run from the root of the Node.js application instrumented by the agent.

```
.\node_modules\appdynamics\node_modules\appdynamics-proxy\proxy\runProxy.cmd
                -j .\node_modules\appdynamics\node_modules\appdynamics-jre\jre /var/log/proxy
                .\node_modules\appdynamics\node_modules\appdynamics-proxy\proxy\logs
                10101
```

Execute the runProxy Command Manually on Windows Task Scheduler

You can schedule the runProxy command in the Windows Task Scheduler.

1. Open Task Scheduler.
2. Click **Create Basic Task**.
3. In the General tab, enter a name for your task.
4. Select **Run whether user is logged on or not**.
5. Check **Run with highest privileges**.
6. In the Triggers tab, add a new trigger. Set the trigger to when the device starts.
7. In the Actions tab, create a new action. Set the action to **Start a program**.
8. In the **Program/script** field, enter:
   `<app_root>/node_modules/appdynamics-proxy/proxy/runProxy.cmd`
9. In the **Add arguments** field, enter:
   `-d <app_root>/node_modules/appdynamics-proxy -j <app_root>/node_modules/appdynamics-jre/jre %TEMP% 10101`
10. In the **Start in** field, enter:
    `<app_root>/node_modules/appdynamics-proxy/proxy`
11. Click **OK** to finish the wizard.
12. Reboot the system, and check the Task Manager for a Java process.

Share a Proxy Among Node.js Agents

Related pages:

- Start the Proxy Manually for Node.js
- User Preferences

This page describes how to configure multiple Node.js Agents on a machine to share a single, multi-tenant Java proxy.

Configure a Shared Proxy

1. Stop the Node.js application.
2. Create directories for the agent and the agent logs and for the proxy control directory. The permissions on this directory must be readable and executable by the application user that the application and writable by the proxy user.
   For example:

   ```
   mkdir /tmp/appd /tmp/appd/logs /tmp/appd/proxy_ctrl_dir
   ```

   When you upgrade the Node.js Agent, you may want to create these directories automatically.
3. Disable automatic proxy launching for each agent. In the AppDynamics `require.profile()` block, set `proxyAutolaunchDisabled` to true and set the `proxyCtrlDir` to the directory that you created.
   For example:

   ```
   ...
   proxyAutolaunchDisabled: true,
   proxyCtrlDir: '/tmp/appd/proxy_ctrl_dir',
   ...
   ```

4. Ensure that each agent that reports to the multi-tenant proxy is configured with a unique node name.
5. Before you start the agents, start the shared Java proxy and pass the agent control directory from the `proxyCommunicationDir` argument to the `runproxy` script. AppDynamics recommends that you configure the proxy to start on system startup. See Start the Proxy Manually for Node.js.
6. Restart the Node.js application. The nodes should appear in the Controller UI flow maps in a couple minutes.

Proxy Sharing Limits

By default, a single proxy can handle up to ten agents. If you need to run more, you will have to adjust the `maxHeapSize` and `maxPermSize` settings in the `runproxy` script.

When adding agents, use debug mode and monitor the `proxy.out` file, which indicates whether the proxy is running out of heap. The file is visible when running in debug mode.

Start the Proxy Manually for Node.js

Related pages:

- Run the Node.js Agent on Windows

This page describes how to manually launch the Java proxy for Node.js.

Upon startup, the Node.js Agent automatically executes the `runproxy` shell script, which starts the Java proxy. This process is suitable for most situations, but there are some cases where you may want to suppress automatic proxy startup and manually launch the proxy.

For example, you are using an external process manager or you are running a Node.js cluster on a machine and want to reduce overhead of multiple proxies.

As the Node.js process owner, you will need to launch the proxy manually.

The following example configures the manual Java proxy launch:

```
nohup /<application_root_dir>/node_modules/appdynamics-proxy/proxy/runProxy -j /<application_root_dir>
/node_modules/appdynamics-jre/jre -- /tmp/appd/proxy_ctrl_dir /tmp/appd/logs &
```

For the full description of the runProxy script syntax for PHP applications, see Start the PHP Agent Proxy Manually.

Node.js Agent Logging (Java Proxy)

Related pages:

- Proxy Logging
- Dynamic Language Agent Proxy

This page explains Node.js Agent logging for the Java proxy mode.

By default, the Node.js Agent generates log information at the informational level. You can increase/decrease the logging verbosity, change the log file location, and configure other logging settings.

Node.js Agent Logging Location

The log file appears at the following location on the machine where the Node.js Agent runs:

```
/tmp/appd/<hash>/appd_node_agent_<datetime>.log
```

If you are running the Node.js Agent, <hash> is generated by the agent.

On Windows, you can find the tmp directory with `cd %tmp% d` and then look for the <hash> directory.

Configure Node.js Agent Logging

To configure logging, use the logging module configuration in the Node.js require statement. You can change the level, maximum size/number of files, and mode.

The following sample shows the logging settings when you are in the Java proxy mode:

```
{
...

  logging: {
     'root_directory': '/tmp/appd',
         'filename': 'echo_%N.log',
     'level': 'TRACE',
     'max_size': 5242880,
     'max_files': 10,
     'outputType': 'console'  // Set this parameter if you want to log to STDOUT/STDERR. Omit this parameter if
you want to log to a file.
  }

...
}
```

The level determines the verbosity of logging output and can contain one of the following values listed from least to most verbose:

- FATAL
- ERROR
- WARN
- INFO
- DEBUG
- TRACE

By default, once the size of the logging files on disk reaches the maximum size, old logs are purged. You can reduce or increase the size, depending on your needs and the disk space available on the machine.

Proxy Logging

The Java proxy logs the transactions that are accepted from the agent and sent to the Controller. The proxy generates logs independently of the Node.js Agent's logging level or activity.

When the Node.js Agent launches the proxy, it prints the directory path at which the proxy is logging to its own log. By default, the proxy log is in `/tmp/appd/<hash>/proxy/logs`.

See Dynamic Agent Proxy Logging.

Debug Logging

Instead of the logging configuration above, you can use this shortcut to set `DEBUG` to true, resulting in debug level logging.

```
debug : true,
```

The location of the debug log file is written in standard output.

See Node.js Agent Settings.

PHP Agent

Related pages:

- AppDynamics Essentials
- PHP Supported Environments
- Dynamic Language Agent Proxy

This page describes the PHP Agent, the application monitoring agent for web and CLI PHP applications in AppDynamics.

About the PHP Agent

The PHP agent component discovers and monitors business transactions, application services, and backends in your PHP application. It injects AppDynamics instrumentation into the application at runtime.

The AppDynamics agent proxy is a Java process that handles the communication between the PHP agent and the Controller. The proxy reports performance metrics to the Controller, where the data is stored, baselined, and analyzed. You can access this performance data interactively using the Controller console or programmatically using the AppDynamics REST API.

The proxy component starts automatically when you start the PHP Agent. For some applications, such as for PHP CLI applications, you need to start the proxy separately from the agent. You need to provide for proxy startup either manually or in scripts.

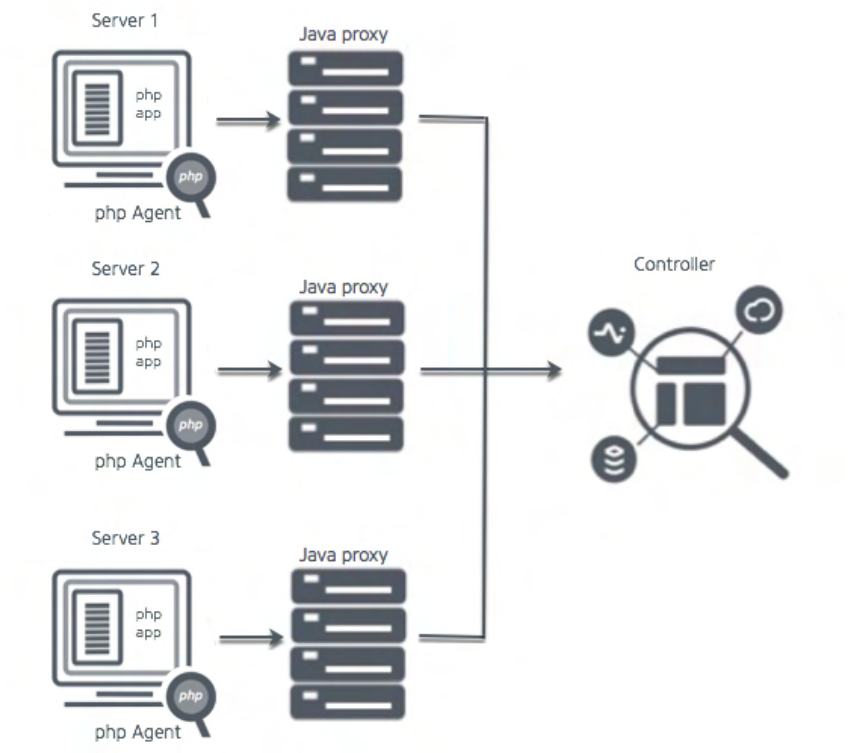
Deployment Overview

The proxy is the component that connects the agent and the controller. If you can ping the Controller but the agent is not connecting, it is possible that the proxy did not start or that the proxy is not configured correctly. See Resolve PHP Agent Installation Issues and Dynamic Language Agent Proxy.

When the connection is successfully established, log in to the Controller to monitor your application. See AppDynamics Essentials to get started.

Single Application Per Web Server Instance

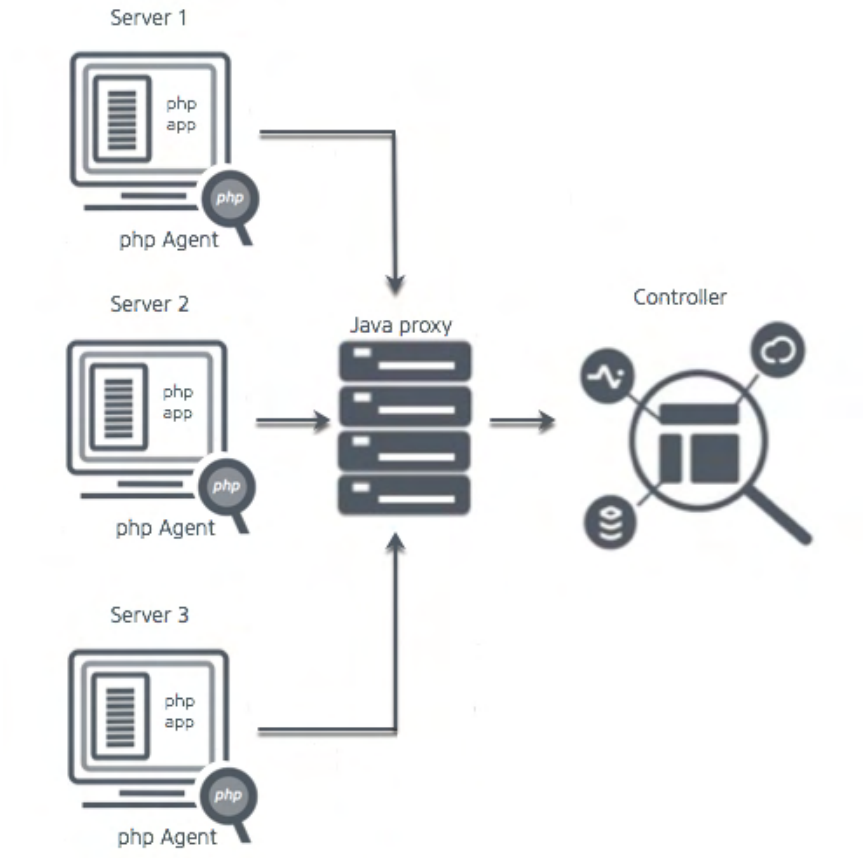
This is the default setup assumed by the installer—multiple servers, each with a single application.



Each application has its own agent and its own agent and proxy. There is a single tier and a single node for each application.

Single Application Per Web Server Instance with Multi-Tenant Proxy

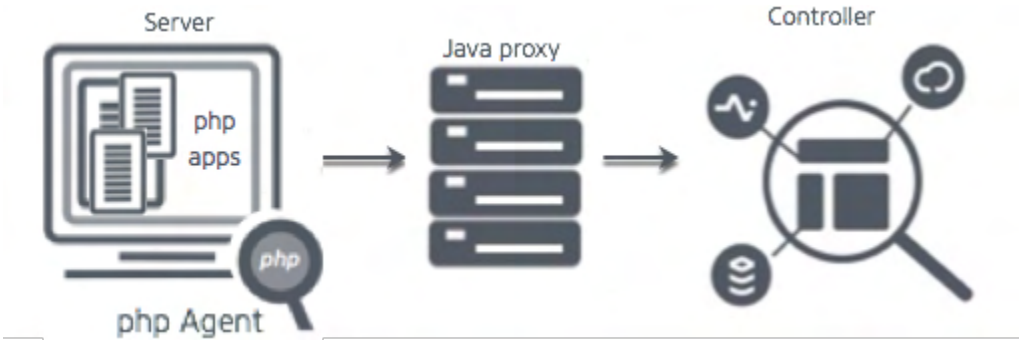
This scenario also assumes multiple servers, each with a single application.



In this case, each application has its own agent component but all the agents share a single proxy. There is a single tier and a single node for each application. Sharing a proxy can save some overhead in some situations. See Use a Shared Proxy for PHP Agents.

Multiple Applications on a Single Server

This scenario assumes multiple applications running in a single server**.**



There is one agent and one proxy supporting multiple applications. By default, the proxy automatically starts when the server starts. The applications can be configured with different tiers and nodes. See Multiple PHP Apps on a Single Server for information on how to configure the Apache vhosts or FPM pool config blocks.

Getting Started with the Download Wizard

If you have never installed the agent, the best way to start is with the Getting Started Wizard in the Controller. The Getting Started Wizard walks you through configuration steps and helps you download the agent. To access the wizard, in the **Home** page of the Controller, click **Getting Started** and then **P HP**.

The wizard provides the minimum information that the agent needs to communicate with the AppDynamics Controller: Controller host and port, optionally SSL, application name and tier name.

Node Naming in a PHP Environment

A PHP runtime instance maps to a node. Your naming convention may depend on your exact environment. Use a name that clearly identifies the Web service that corresponds to the node. Some options are:

- `hostName-appName-nodeName`
- `hostName-tierName-nodeName`
- `appName-nodeName`
- tierName-nodeName
- `IP address`
- `fully qualified domain name`

PHP Supported Environments

Related pages:

- PHP Business Transaction Detection

PHP Agent Support

PHP Versions

PHP Agent supports these versions of PHP:

- 5.6
- 7.0
- 7.1
- 7.2
- 7.3
- 7.4

PHP Web Servers

Apache 2.2 and 2.4 in these modes:

- `prefork` mode using `mod_php`
- worker MPM mode using `mod_fastcgi` with `php-fpm` or `mod_fcgid` with `php-cgi`

Any web server compatible with `php-fpm`.

Operating Systems

- Any Linux distribution based on `glibc 2.5`+ and the x86 32-bit or x86 64-bit architecture
- Mac OS X 10.9+

PHP Agent supports 32-bit operating system only on PHP 5.6.

PHP Frameworks and Protocols

Framework/Protocol	Version	Entry Point Type
Drupal	7	Drupal
Drupal	8	PHP MVC
WordPress	3.4+, 4.x, 5.x	Wordpress
Zend	1, 2, 3	PHP MVC
CodeIgniter	2.x, 3.x, 4.x	PHP MVC
FuelPHP	1.5x, 1.6x, 1.8x	PHP MVC
Magento	1.5, 1.6, 1.7, 2.3	PHP MVC
Symfony	1, 2, 3, 4	PHP MVC
CakePHP	2.x, 3.x, 4.x	PHP MVC
Laravel	5.7, 6, 8	PHP MVC
HTTP		PHP Web
CLI		PHP CLI

If your PHP framework is not listed here, the agent detects your entry points as PHP Web and names the business transactions based on the first two segments of the URI — the default naming convention for PHP Web transactions. So it is still possible to monitor applications on *unsupported* frameworks. Laravel BTs are detected as symfony, as laravel itself is built on top of symfony.

There are few limitations of the PHP Agent. The PHP Agent does not:

- Monitor PHP applications in Zend Thread Safety (ZTS) mode. If you are using ZTS, AppDynamics suggests that you review your dependencies on ZTS to confirm that you actually need it, and if you do not, to switch to the non-ZTS mode
- Support Zend Monitor
- Officially support plug-ins that encrypt and, or obfuscate PHP code such as Zend Guard or ionCUBE Loader
- Support compatibility with the Xdebug module

Transaction Naming

Framework/Environment	Default Transaction Naming
Drupal	page callback name
Wordpress	template name
PHP MVC Frameworks	controller:action
PHP Modular MVC Frameworks	module:controller:action
PHP Web	URI
PHP Web Service	service name.operation name
PHP CLI	last two segments of the script's directory path plus the name of the script

Virtual host prefixing is available for all supported entry point types except PHP CLI.

PaaS Providers

PaaS Provider	Buildpack
Pivotal Cloud Foundry	https://github.com/Appdynamics/php-buildpack See http://docs.pivotal.io/appdynamics/index.html for information about integration with PCF.

Exit Points

Supported HTTP Exit Points
`curl/curl-multi`*
`drupal_http_request()`
`fopen(), file_get_contents()`
`Zend_HTTP_Client::request()`

*The total time reported for a curl/multi_curl request in the Controller is the same as reported by the function `curl_getinfo`. Also, we report the the following execution metrics in the exit call details for the curl/multi_curl request which are included in the total time:

- `namelookup_time`
- `connect_time`
- `pretransfer_time`
- `redirect_time`

Supported Database Exit Points
MySQL old native driver (removed for PHP 7)
MySQLi Extension*
OCI8
PDO
PostgreSQL accessed via PDO and pgsql extensions

* `mysqli_multi_query` is not supported.

Supported Cache Exit Points
Memcache
Memcached
Predis 0.8.5 and 1.1.1, on PHP versions 5.6 and higher
Phpredis 4.1

Although Predis is a full PHP client library, the PHP Agent supports Predis as an exit point only, not as an entry point.

Supported Web Service Exit Points
PHP SOAPClient
NuSOAP 0.9.5

Supported Message Queue Exit Points
RabbitMQ

RabbitMQ support requires the amqp extension.

Opcode Cache Compatibility

Alternative PHP Cache (APC)	

Install the PHP Agent

Related Pages:

- Download AppDynamics Software
- AppDynamics Download Portal
- SELinux Installation Issues

This page provides an overview of AppDynamics PHP Agent installation.

Installation Overview

In the Controller UI, the Getting Started Wizard walks you through the configuration and installation steps for the PHP Agent.

The installer that you get from the wizard uses the PHP directory specified in your environment PATH variable to determine where to install the agent. If you are instrumenting a PHP installation not specified in your PATH, you can manually configure and invoke the RPM or script installer.

The PHP Agent runs on Linux and MacOS machines. The PHP Agent is designed for PHP operating environments in which:

- There is a single PHP installation
- PHP is running in a single Apache or FPM pool
- Standard packages have been used to install PHP, Apache and/or PHP-FPM
- No customizations have been made to the PHP configuration

The PHP Agent may work in environments that do not meet all assumptions. However, you should use extra care in installing and thoroughly testing the PHP agent and application in a staging environment.

Before Starting

Before attempting to install the AppDynamics PHP Agent, confirm that you use a supported PHP version. You can check your PHP version using this command:

```
php -i
```

The PHP Agent *does not work* with PHP applications that are built with the `enable-debug` configuration option or in any build otherwise compiled with debugging symbols. To determine whether the application was built with debugging symbols, you can use this command:

```
php -i | grep -e "Debug Build"
```

The response should be:

```
Debug Build => no
```

You can get the PHP Agent distribution from the AppDynamics Download Portal. Be sure to get the distribution file appropriate for your system:

- For RedHat or CentOS, use the RPM Package Manager distribution, such as PHP Agent—64-bit RPM.
- For all other Linux distributions, use the distribution identified PHP Agent—64-bit Linux.
- For MacOS, use PHP Agent—64-bit OSX

At installation time, you will need to provide Controller connection settings and identifying information for this PHP node. The Getting Started Wizard walks you through this configuration, so if it is your first time installing an AppDynamics Agent, we recommend using the Getting Started Wizard. For general information about the Controller connection settings, see Agent-to-Controller Connections.

To complete the instrumentation, you will need to stop and start the Apache Server. Be sure to perform the installation in a way that minimizes disruption to service users, if applicable.

> Do not install the PHP Agent along with other non-AppDynamics Application Performance Management (APM) tools, especially in a production environment. The PHP Agent installation may fail if there are other APM products installed in the same managed environment.

Installing the PHP Agent

These topics provide installation details for various scenarios:

- To use install.sh, see Install the PHP Agent by Shell Script. Use install.sh to install on OSX as well as Linux.
- To use RPM, see Install the PHP Agent by RPM.
- To instrument PHP CLI, see also Configure the Agent for PHP CLI Applications.

- To instrument an application on MacOS, see MacOS X Installation Considerations.
- To instrument multiple PHP applications running on the same server, see Multiple PHP Apps on a Single Server.

For cURL installation, see Download AppDynamics Software.

Files Added by Installation

PHP configuration files

For AppDynamics, the PHP configuration files of interest are the `php.ini` and `appdynamics_agent.ini` fragment. AppDynamics settings can be found in either `.ini` file, depending on the operating system under which your PHP is installed.

The PHP Agent installer adds the `appdynamics_agent.ini` file to the directory that contains your `php.ini` file. You can find this directory using the following command:

```
php -i | grep -e "Additional .ini files parsed"
```

If the installer is not able to determine the directory where the ini fragments for your PHP deployment live, it displays the required AppDynamics ini fragment and prompts you to copy and paste it into your main `php.ini` file.

Also, see http://php.net/manual/en/configuration.file.php for information about possible locations.

.so files

The installer also installs the appdynamics_`agent.so` file in your PHP extensions directory. You can find this directory using the following command:

```
php -i | grep  extension_dir
```

Logs

There is an agent log and a proxy log for each application.

By default, the agent log is written to `$<php_agent_install>/logs/agent.log`. The log contains the transactions that the agent processes and then sends to the proxy. The default pattern for agent log naming is:

- `agent.log`: the current log
- `agent.log.1`: most recent log
- `agent.log.2`: second most recent log
- `agent.log.3`: third most recent log
- `agent.log.4`: fourth most recent log
- `agent.log.5`: fifth recent log

By default, the proxy log is written to `$<php_agent_install>/logs/proxy_$date.log`. This log contains the transactions that the proxy accepts from the agent and then sends to the Controller.

If you configure the logs directory with the `--log-dir` option, the proxy logs are written to the same directory as the agent logs.

For information about the location of log files generated by an RPM installation, see RPM Log file in Install the PHP Agent by RPM.

Using a Machine Agent on a PHP Node

You can install a machine agent on a node that runs the PHP Agent. However, note that if you install the machine agent on a PHP Agent node and you specify the tier and node name in the `controller-info.xml` file of the machine agent, the PHP Agent will not successfully register with the Controller.

To avoid this problem:

- Install the PHP Agent before you install the machine agent
- If you install the machine agent on the machine hosting the instrumented PHP node, do not specify the application, tier or node name in the `controller-info.xml` file of the machine agent. If you do, the PHP Agent may fail to register.

Install the PHP Agent by Shell Script

Related pages:

- AppDynamics Downloads
- Install the PHP Agent

This page describes how to install the PHP Agent using the included `install.sh` script.

Installing the PHP Agent

These steps describe how to install the PHP Agent with the installation script, `install.sh`.

1. Verify support and system requirements for your environment as described in Install the PHP Agent.
2. Create the `php-agent` installation directory. The directory should be owned by the same user that runs Apache or PHP-FPM (FastCGI Process Manager). AppDynamics recommends naming the directory `/opt/appdynamics/php-agent`.

   > This document refers to the PHP Agent installation directory as: `<php_agent_install>`. Replace the `<php_agent_install>` syntax with the full path to your agent installation directory. For example, change `<php_agent_install>` to `/opt/appdynamics/php-agent`

3. If there is more than one Apache instance on a machine, run `install.sh` once for each Apache instance, each time with the appropriate node, p `hp_ini` dir and `php_ext` dir options.

   See Run the PHP Proxy Daemon Manually

   See Files Added to Your Installation

   From the command line, change directories to the PHP agent installation directory and untar the agent distribution tarball, as in the following example:

   ```
   cd <php_agent_install>
   tar -xvjf appdynamics-php-agent-x64-linux.tar.bz2
   ```

4. Set the following permissions on your `<php_agent_install>` directory.

   - `php`: Make every directory that leads to the PHP agent logs directory readable and executable by all and writable by the directory owner:

     ```
     chmod -R 755 <php_agent_install>
     ```

   - `logs`: If possible, make the logs subdirectory readable/writable/executable by all:

     ```
     chmod 777 <php_agent_install>/logs
     ```

     Directory access permissions of 777 may be too permissive for some organizations. In this case, simply make sure that the directory is owned by the `apache/php/proxy` user.

5. Run the installation script using this syntax:

   ```
   <php_agent_install>/install.sh  [-s] -a=<account_name>@<account_access_key>
   [--http-proxy-host=<proxy_host>] [--http-proxy-port=<proxy_port>] [-e <php_ext_dir>] [-i <php_ini_dir>]
   [-p <php_binary_dir>] [-v <php_version>]
    <controller-host> <controller-port> <app_name> <tier_name> <node_name>
   ```

   The command arguments are described as follows:

   - -s option: You can optionally specify the `-s` option if you want the agent to use SSL (HTTPS) to connect to the controller. In this case, set the Controller port to the HTTPS port of the controller.
   - `-a account_name@account_access_key`: The required controller account name and account key. To find your account name and access key, click  in the upper right corner of the AppDynamics UI, then click **License**.
   - `http-proxy-host` and `http-proxy-port`: Set the `<http-proxy-host>` and `<http-proxy-port>` to route data to the controller through a proxy server. The `<http-proxy-host>` is the hostname or IP address of the proxy server. The `<http-proxy-port>` is the proxy server HTTP or HTTPS port, whichever you are using. If you set the `http-proxy-host` you must set the `http-proxy-port` as well.
   - `http-proxy-user`, `http-proxy-password-file:` The username and password file if using a proxy and the proxy server requires credentials.

- `-e` option: Extensions directory for the `appdynamics_agent.so` file. Needed on Ubuntu as well as when the default PHP CLI binary cannot be determined.
- `-i` option: `ini` directory for the `appdynamics_agent.ini` file. Needed on Ubuntu as well as when the default PHP CLI binary cannot be determined.
- `-v` option: version of PHP that you are instrumenting. Valid formats are version numbers to one or two decimal positions, for example, `5.4` and `5.4.21` are both valid. Needed only when the default PHP CLI binary cannot be determined or there is no PHP CLI binary.
- `-p` option: path to the PHP binary
- `enable-cli-long-running`: Set to `true` to defend PHP in long-running CLI applications. Defaults to `false`. See Long-Running CLI Applications with the Suhosin Patch.
- `--log-dir=<dir>`: The directory to which the agent and proxy logs should be written. Defaults to the `<php_agent_install>/logs` directory.
- `--proxy-ctrl-dir=<dir>`: The proxy control directory. If not specified, the installer creates a temporary directory.

If all options are used, the `-e`, `-i` and `-v` options have precedence over the -p option.

> On Ubuntu, the installation needs to be performed as the root user. Also, you need to use the `-e` option to indicate the correct extensions directory for the `appdynamics_agent.so` file and the `-i` option for the correct `ini` directory for the `appdynamics_agent.ini` file.

6. Restart the webserver, unless you are installing an agent to monitor PHP-CLI only.

If your installation failed, see Resolve PHP Agent Installation Issues.

Installation Command Samples

The following illustrates a sample command to install the agent:

```
install.sh controller 8090 -a=PHPCust@XC6v2n8m2$543 myApp myTier myNode
```

To install the agent using SSL, use the `-s` switch, as follows:

```
install.sh -s -a=PHPCust@XC6v2n8m2$543 controller1.appdynamics.com 8818 myApp myTier myNode
```

The following sample command illustrates installation with a proxy server present:

```
install.sh --http-proxy-host=myproxyhost --http-proxy-port=8099 -a=PHPCust@XC6v2n8m2$543 controller 8090 myApp
myTier myNode
```

Install the PHP Agent by RPM

Related pages:

- Resolve PHP Agent Installation Issues
- Install the PHP Agent
- Download AppDynamics Software
- AppDynamics Download Portal

You can use the AppDynamics PHP Agent RPM to install the PHP Agent on RedHat or CentOS Linux systems.

About the RPM Installer

You must use the PHP Agent RPM installer as the root user.

You pass configuration settings for the agent to the installer using environment variables rather than command-line arguments. Before starting—or as part of your installation script—you need to set the environment variables. You can configure settings such as node name, tier name, business application name, and others for the agent. However, the installer will provide sensible defaults if not provided.

The RPM installer determines the location of your PHP installation based on the PATH environment variable. It uses the first PHP installation that it encounters in the PATH to configure the installer. If you have installed PHP in a non-standard location, you must provide the directory of your PHP binary in the `APPD_PHP_PATH` environment variable.

Before starting, get the RPM package from the AppDynamics Download Portal.

Installing the PHP Agent with RPM

To install PHP Agent with RPM:

1. Verify support and system requirements for your environment as described in Install the PHP Agent.
2. Set environment variables with the configuration settings for the agent if needed. The account key is a required setting. Other settings may be required depending on your environment. See RPM Environment Variables for more information. Alternatively, you can pass environment variables to the installation script directly.
3. Run the installer.

   ```
   sudo rpm -i <package_name>
   ```

   Replace `<package_name>` with the name of the package you downloaded.

   If you are using `sudo` to pass the environment variables to the installation script you can use:

   ```
   sudo APPD_PHP_PATH=/opt/php rpm -i <package_name>
   ```

   or

   ```
   APPD_PHP_PATH=/opt/php sudo -E rpm -i <package_name>
   ```

   If the install reports errors, check the log file as described in RPM Log File.
4. Restart Apache. Not required if you are installing an agent to monitor PHP CLI only.

The RPM installs a single agent. If you have more than one PHP installation on the machine, run the RPM once for each PHP installation, each time with the appropriate `APPD_PHP_PATH` and `APPD_CONF_NODE` settings.

RPM Environment Variables

Environment Variable	Description	Default	Required?
`APPD_PHP_PATH`	Directory containing the PHP binary	None	If your PHP binary is not in a standard location. By default, the installer uses the PHP CLI binary to determine where to install the app agent.
`APPD_PHP_CONFIGURATION_DIR`	INI directory in which to install the `appdynamics_agent.ini` file. Takes precedence over the `APPD_PHP_PATH` setting.	Directory containing your `php.ini` file. See information on files added to your installation in Install the PHP Agent.	If your PHP binary is not in the standard location or if no PHP CLI binary is available.

APPD_PHP_EXTENSION_DIR	Extensions directory in which to install the appdynamics_agent.so file. Takes precedence over the APPD_PHP_PATH setting.	Your PHP extensions directory. See information on files added to your installation in Install the PHP Agent.	If your PHP binary is not in the standard location or if no PHP CLI binary is available.
APPD_PHP_VERSION	Version of PHP that you are instrumenting. Valid formats are version numbers to one or two decimal positions, for example, 5.4 and 5.4.21 are both valid. Takes precedence over the APPD_PHP_PATH setting.	Version used by your PHP CLI binary	If no PHP CLI binary is available, along with the APPD_PHP_CONFIGURATION_DIR and APPD_PHP_EXTENSION_DIR variables.
APPD_CONF_CONTROLLER_HOST	Controller hostname	localhost	
APPD_CONF_CONTROLLER_PORT	Controller port	8080	
APPD_CONF_APP	Application name	MyApp	
APPD_CONF_TIER	Tier name	Hostname of the machine running the script—same as the node name	
APPD_CONF_NODE	Node name	Hostname of the machine running the script	
APPD_CONF_ACCOUNT_NAME	Account name	Defaults to customer1.	If you have an on-premises AppDynamics Controller running in multi-tenant mode or if you are using the AppDynamics SaaS Controller.
APPD_CONF_ACCESS_KEY	Account key	None	Yes. To find your access key, click  in the upper right corner of the AppDynamics UI, then click **License**.
APPD_CONF_SSL_ENABLED	True to enable SSL communication with the controller, false otherwise	false	
APPD_CONF_HTTP_PROXY_HOST	Hostname or IP address of the http proxy server	None	If you want to route data to the controller through a proxy server.
APPD_CONF_HTTP_PROXY_PORT	HTTP or HTTPS port of the http proxy server; must be set if APPD_CONF_HTTP_PROXY_HOST is set	None	If you want to route data to the controller through a proxy server.
APPD_CONF_HTTP_PROXY_USER	Username on the http proxy server	None	If you want to route data to the controller through a proxy server that requires a username.
APPD_CONF_HTTP_PROXY_PASSWORD_FILE	Password on the http proxy server	None	You want to route data to the controller through a proxy server that requires a password.
APPD_PROXY_CTRL_DIR	Initial control communication directory between the agent and the Java proxy	None	
APPD_CONF_LOG_DIR	Directory to which the agent writes the agent and proxy logs	<php_agent_install>/logs	
APPD_CONF_CLI_LONG_RUNNING_ENABLED	Defends PHP in long-running CLI applications.	false	Advisable if instrumenting long-running CLI applications. See Long-Running CLI Applications with the Suhosin Patch in Configure the Agent for PHP CLI Applications.

Updating the Installation

Any changes that you made to the configuration files are preserved when you re-run the installer. RPM saves your original settings and appdynamics_agent_log4cxx.xml files with the settings from the previous installation.

RPM Installation Log File

If errors occurred during installation, a message displays the location of the log file generated in the /tmp directory. Examine this log file to identify the cause of the problem. A successful installation does not produce a log file.

Uninstall the PHP Agent using RPM

If you installed the agent using RPM, use RPM to uninstall it, as follows:

```
sudo rpm -e appdynamics-php-agent-<version>
```

Notice that you need to identify the version of the package. To find the version of the package that you installed, you can enter:

```
rpm -qa | grep appdynamics-php-agent
```

The existing configurations are saved in a tar file in `/tmp`, as indicated upon completion of the uninstall process.

Configure the Agent for PHP CLI Applications

Related pages:

- Start the PHP Agent Proxy Manually
- PHP Agent Configuration Settings

If you are instrumenting a PHP CLI Application with the AppDynamics PHP Agent, after installing the agent you need to take a few additional configuration steps described here.

Instrumenting PHP CLI Applications

When instrumenting a PHP CLI application, you need to provide for the startup of the proxy daemon, as follows:

1. Create the proxy control directory, which is used for agent/proxy communication.
2. In your PHP configuration file—`php.ini` or `appdynamics_agent.ini` depending on your environment. Include the following settings:
    - `agent.cli_enabled = 1`
    - `agent.auto_launch_proxy = 0`
    - `agent.proxy_ctrl_dir  = <proxy control directory>`
      If you are using the RPM installer to install the agent you may have configured the proxy control directory using the `APPD_PROXY_CTRL_DIR` environment variable. See Install the PHP Agent by RPM. This environment variable takes precedence over the setting in the ini file.
3. Before running any traffic through the CLI, run the proxy from the directory into which you installed the PHP agent, passing the proxy control directory and proxy log directory as arguments.

    ```
    proxy/runProxy <proxy_control_dir> <log_dir>
    ```

    For example:

    ```
    proxy/runProxy /tmp/proxy.communication /tmp/agentLogs
    ```

For the full set of options to runProxy, see Start the PHP Agent Proxy Manually.

PHP CLI and the Apache Web Server

If you have PHP CLI applications and an Apache web server on the same machine, your setup depends on whether you want all the traffic reported against a single AppDynamics node or separate nodes. A separate proxy is required for each AppDynamics node that you want to monitor in the controller.

If you want all the CLI traffic to be reported against one node and all the web traffic to be reported against a different node, configure Apache to auto-launch the proxy (the default) and configure CLI to use a manually-launched proxy. This requires separate `.ini` files—one for the web PHP with `agent.auto_launch_proxy` set to `1` and another for PHP CLI with `agent.auto_launch_proxy` set to `0`.

If you want the web traffic and the CLI traffic to be reported against the same node, configure both Apache and CLI to use the same manually launched proxy.

Long-Running CLI Applications with the Suhosin Patch

A side effect of the Suhosin patch is that it prevents the PHP Agent from ensuring cleanup in long-running CLI applications.

If your PHP has the Suhosin patch, it is possible that resources will not be freed in long-running applications. Thus memory leaks could result if the application itself does not explicitly free these resources.

The `long-running-cli` feature defends PHP applications in an environment in which both of the following conditions exist:

- PHP with the Suhosin patch is running on Debian or Ubuntu. It is common for Debian and Ubuntu PHPs to have this patch.
  This feature is not needed for PHPs with only the Suhosin extension, which is different from the patch. Be aware that some PHPs use both the extension and the patch.
- Using the PHP Agent API, you are instrumenting a CLI application that has multiple unbounded business transactions running on the same process,

How the Long-Running CLI Feature Works

At installation time, if the installer determines that PHP has the Suhosin patch and CLI is enabled—`agent.cli_enabled=1`—the value of the installer option results as follows:

- If `true`, a fatal error is generated and the installer terminates. With the option set, the installer refuses to instrument a long-running CLI application on a PHP installation with the Suhosin patch.
- If `false`—the default—the installation continues and warns that memory leaks could occur in long-running CLI processes.

If the installer determines that PHP does not have the Suhosin patch, the installation continues. Long-running CLI processes are supported by the agent, since there is no Suhosin patch.

If the agent could not determine whether your PHP has the Suhosin patch at installation but it does detect the patch at runtime, having set the installer option to 'true' prevents the agent from instrumenting any CLI processes, not just long-running ones. This prevents the Suhosin-patched PHP from exiting.

If CLI is enabled and the installer did not terminate because of the detection of the Suhosin patch, AppDynamics recommends that you install the agent with the `enable-cli-long-running` option—shell script installs—or the `APPD_CONF_CLI_LONG_RUNNING_ENABLED` environment variable —for RPM installs—set to `true`. This will defend your PHP if the patch is detected at runtime.

If the CLI part of your application does not get instrumented, because the installer detected the Suhosin patch, you can unset the option by setting the `agent.cli_long_running` option in the PHP ini file to `off`. Or alternatively, you can re-install with the installer.

Multiple PHP Apps on a Single Server

Related pages:

- Install the PHP Agent
- PHP Business Transaction Detection
- Overview of Application Monitoring

If you follow only the default installation instructions, the PHP Agent treats your entire PHP deployment as a business application with a single tier with a single node. It models your AppDynamics environment this way even if in reality you have multiple PHP applications running in the same Apache or PHP-FPM pool.

You may have configured your PHP applications as different virtual hosts or FPM pools. With multi-application/single-server (MASS) support, you can monitor several PHP applications running on the same server as separate AppDynamics entities, represented as different business applications, different tiers or different nodes. Multiple applications running in the same PHP server can have separate metrics, dashboards, health rules, events, and so on. You can monitor them on different controllers, even from different AppDynamics accounts. You can mix and match the settings to configure the AppDynamics model that works best for monitoring your entire PHP application environment.

AppDynamics recommends that you not instrument more than ten applications on a single server.

Download and Install

To set up your PHP installation for multiple nodes, download a single agent from AppDynamics.

Then follow the instructions in Install the PHP Agent by RPM—RedHat and CentOSX—and Install the PHP Agent by Shell Script—all other Linux and Mac OS—for instrumenting the agent. Configure the required settings using an app/tier/node configuration that fits one of the applications you are instrumenting. Make sure you configure the required settings listed below.

Then, for each additional application that you want the agent to monitor, add an AppDynamics configuration with different settings that reflect how you want the performance metrics to be reported in your AppDynamics model.

Required Configuration Settings for Multi-Node Support

The required settings for multi-node support for each monitored app are:

- `agent.controller.hostName`
- `agent.applicationName`
- `agent.tierName`
- `agent.nodeName`
- `agent.accountName`

It may be necessary to configure additional settings depending on your environment, for example, if you need to route data to the AppDynamics controller through a proxy server.

Where to Configure Settings

There are three files from which the PHP Agent configuration settings are read:

1. appdynamics_agent.ini
2. Apache (vhost) or FMP (fpm-pool) configuration (overrides 1)
3. .htaccess for Apache or .user.ini for FPM (overrides 1 and 2)

When you instrument your first application, passing the required settings on the command-line—shell script install—or through environment variables (RPM install), those settings are written to `appdynamics_agent.ini`.

To apply different sets of settings for the different applications, as best practice specify the settings in each vhost or FPM pool config block.

Sample Apache Configuration

Here are sample virtual host configurations for two applications running on the same server.

Virtual Host Configuration for the `api` App

```
<VirtualHost *:80>
    ServerName api.myhost.com
    DocumentRoot "/var/www/api"
    ErrorLog "/private/var/log/apache2/api_error.log"
    CustomLog "/private/var/log/apache2/api_access.log" common

    php_value agent.applicationName API
    php_value agent.tierName api-tier1
    php_value agent.nodeName api-node1
    php_value agent.controller.hostName appd1.saas.appdynamics.com
    php_value agent.controller.port 8080
    php_value agent.accountName customer1
    php_value agent.accountAccessKey 123456789
</VirtualHost>
```

Virtual Host Configuration for the `frontend` App

```
<VirtualHost *:80>
    ServerName www.myhost.com
    DocumentRoot "/var/www/frontend"
    ErrorLog "/private/var/log/apache2/www_error.log"
    CustomLog "/private/var/log/apache2/www_access.log" common

    php_value agent.applicationName FrontEnd
    php_value agent.tierName frontend-tier1
    php_value agent.nodeName frontend-node1
    php_value agent.controller.hostName appd1.saas.appdynamics.com
    php_value agent.controller.port 8080
    php_value agent.accountName customer1
    php_value agent.accountAccessKey 123456789
</VirtualHost>
```

Sample FPM Configuration

Here are sample FPM configurations for two applications running on the same server.

.conf Configuration for the "api" App

```
api.conf:

php_value[agent.applicationName] = API
php_value[agent.tierName] = api-tier1
php_value[agent.nodeName] = api-node1
php_value[agent.controller.hostName] = appd1.saas.appdynamics.com
php_value[agent.controller.port] = 8080
php_value[agent.accountName] = customer1
php_value[agent.accountAccessKey] = 123456789
```

.conf Configuration for the `frontend` App

```
www.conf:

php_value[agent.applicationName] = FrontEnd
php_value[agent.tierName] = frontend-tier1
php_value[agent.nodeName] = frontend-node1
php_value[agent.controller.hostName] = appd1.saas.appdynamics.com
php_value[agent.controller.port] = 8080
php_value[agent.accountName] = customer1
php_value[agent.accountAccessKey] = 123456789
```

MacOS X Installation Considerations

If you are instrumenting an application running on MacOS X, you need to instrument it manually using the shell script.

If you are installing the PHP Agent on MacOS X, set `max_execution_time` to `0` in the `appdynamics_agent.ini` file

If you have another INI file that is loaded after `appdynamics_agent.ini`, and sets `max_execution` time to a non-zero value, set `max_execution_time` to `0` in the `php.ini` file instead.

Use a Shared Proxy for PHP Agents

Related pages:

- Install the PHP Agent

By default, if you are running multiple agents, each agent automatically launches its own Java proxy to communicate with the controller.

However, if you are running multiple PHP Agents on the same machine, you can reduce your overhead by setting up the agents to report to a single shared or multi-tenant proxy. In this case, you would need to start that proxy manually.

The number of nodes that can report to a single proxy is limited by the size of the heap given to the proxy. You may need to adjust the maxHeapSize and maxPermSize settings in the `runproxy` script if you have a large number of agents reporting to a single proxy.

Setting up a Multi-Tenant Proxy

1. Configure each agent for manual launch of the proxy. To do this, in the PHP configuration file—`php.ini` or `appdynamics_agent.ini` depending on your setup—for each agent, set the agent.auto_launch_proxy value to 0.
2. Configure a single proxy control directory for all the agents that will share the proxy. They must all be on the same machine. To do this, in the PHP configuration file for each agent, set `agent.proxy_ctrl_dir` to the same proxy control directory.
   The permissions on this directory should be readable and executable by the process that runs Apache and writable by the process that runs the proxy.
3. Before you start the agents, arrange to launch the proxy manually, passing the proxy control directory configured in step 2 as the `proxyCommunicationDir` argument to the `runProxy` script. See information on executing the runProxy in Start the PHP Agent Proxy Manually. AppDynamics recommends launching the proxy on system startup.
4. Verify that each agent reporting to the multi-tenant proxy is configured with a unique `app_name`/`node_name` combination. The `app_name` and `node_name` are arguments to the agent install script.

Start the PHP Agent Proxy Manually

Related pages:

- Use a Shared Proxy for PHP Agents
- Configure the Agent for PHP CLI Applications

When the PHP agent starts, it automatically starts the Java proxy that handles communication between the PHP agents and the controller.

Automatic startup of the proxy works for the great majority of situations. However, you can suppress the automatic startup of this script and run it manually. You would do this if:

- You plan to instrument a PHP CLI entry point. The PHP CLI entry point requires manual startup of the proxy and immediate creation of the node on startup.
- You have multiple Apache or FPM pools on the same machine reporting to the same proxy.

To do this, you first need to configure the agent for manual proxy launch. Then you need to launch the proxy manually.

Configure the Agent for Manual Proxy Launch

The PHP Agent configuration needs to be modified to reflect manual launching.

To configure the agent for manual launching:

1. Open the PHP configuration file for editing. The file is `php.ini` or `appdynamics_agent.ini`, depending on your setup.
2. Set the `agent.auto_launch_proxy` setting to 0.
3. Set the `agent.proxy_script` to the path of the `runproxy` that you want to use. The file in the script is relative to the root of the PHP agent. You can specify the absolute path if you prefer.
4. Set the `agent.proxy_ctrl_dir` to the directory to use for initial control communication between the agent and the proxy. This directory contains the domain control socket, which the agent uses to start an AppDynamics node. This directory is where the agent gets the configuration for the node. The application user must have read permission on the `proxy_ctrl_dir`.

Whenever you install the PHP agent, the installer overwrites the `runproxy` script and the `appdynamics_agent.ini` file, but not the `php.ini`. If you re-install, you need to reset the `agent.auto_launch_proxy` setting in the `appdynamics_agent.ini` file before you restart the server.

Run the Proxy

Before any traffic runs on the instrumented server, execute the `runProxy` script to start the proxy.

The following lists the full set of options for the `runProxy` script. The proxy control directory is required.

```
Usage: runProxy options -- proxyCommunicationDir logDirectory [jvmOption [ jvmOption [...] ]

Options:
  -r <dir>, --proxy-runtime-dir=<dir>   Specifies proxy runtime directory
  -d <dir>, --proxy-dir=<dir>           Specifies root proxy directory
  -j <dir>, --jre-dir=<dir>             Specifies root JRE directory
  -v, --verbose                         Enable verbose output
  -h,--help                             Show this
```

The following shows an example:

```
./proxy/runProxy -d ./proxy -r <agent.proxy_ctrl_dir> /tmp/proxy.communication /tmp/agentLogs
```

A node is created when the agent first detects traffic on it.

Every time you reboot the server, you need to execute the `runProxy` script if you have opted to start the proxy manually.

Resolve PHP Agent Installation Issues

If traffic does not display in the Controller after you have installed the AppDynamics PHP Agent and started the instrumented server, try these troubleshooting steps.

Ensure that the Agent is Installed in the Correct Directory

It is possible that the agent was installed in the wrong directory. Verify the location of your PHP installation.

- Verify the location of your PHP by running `phpinfo`. See http://us1.php.net/phpinfo.

Then check where the installer actually installed the agent files.

- The `appdynamics_agent.ini` file should be in the same directory that contains the php.ini file for your PHP installation.
- The `appdynamics_agent.so` file should be in the extensions directory for your PHP installation.

See Files Added to Your Installation for information about how to locate these directories.

In addition, on Linux you can use `pstree` to locate the agent. The `pstree` command displays the AppDynamics agent running under Apache if the agent is installed properly. See http://freecode.com/projects/pstree.

If the agent files are not in the correct directories, re-install the agent with the `-i` and `-e` options.

Re-install the agent

If the app agent is not installed in the right directory, re-install the agent using the `install.sh` installer with the `-i` and -e options. Use the `-i` to install the `appdynamics_agent.ini` file in the same directory as your `php.ini` file and the -e option to install the `appdynamics_agent.so` file in the same directory as your PHP extensions directory. See Install the PHP Agent by Shell Script.

If you initially installed the agent using the RPM installer, you can find the shell script installer at `/usr/lib/appdynamics-php5/install.sh`.

Check error messages in the installation output

When you reinstall, examine carefully any error messages in the output of the install script, especially those that direct you to copy some settings into your `php.ini` file.

If necessary copy those settings into the php.ini file. See PHP Agent Configuration Settings for information on common settings that may be missing.

Check the AppDynamics Settings Block in the Configuration

Run this command:

```
php -i | less
```

and examine the output. You should see an `appdynamics_agent.ini` file and a configuration block listing appdynamics `ini` values.

Confirm User Permissions

Check that the following permissions are set:

```
chown -R <apacheuser>:<apacheuser> <php_agent_install>
chmod -R 755 <path_to_php_agent_install_logs>
chmod 777 <php_agent_install>/logs
```

Verify that the Proxy is Running

The Java proxy is the part of the agent that communicates with the Controller. If the agent is installed in the right place, confirm that the Java proxy is running.

1. From the command line enter, `ps aux | grep java`.
2. Inspect the list. You should see output similar to the following if the proxy is running:

```
/usr/lib/appdynamics-php5/proxy/jre/bin/java -server -Xmx120m -classpath /usr/lib/appdynamics-php5/proxy/conf
/logging/*:/usr/lib/appdynamics-php5/proxy/lib/*:/usr/lib/appdynamics-php5/proxy/lib/tp/*:/usr/lib/appdynamics-
php5/proxy/* -Djava.library.path=/usr/lib/appdynamics-php5/proxy/lib/tp -Dappdynamics.agent.logs.dir=/usr/lib
/appdynamics-php5/logs -Dcomm=/tmp/ad-siJ4rp -DagentType=PHP_APP_AGENT -Dappdynamics.agent.runtime.dir=/usr/lib
/appdynamics-php5/proxy com.appdynamics.ee.agent.proxy.kernel.Proxy
```

If you are instrumenting a PHP CLI script, you need to start the proxy manually. You may also need to start the proxy manually if you have special requirements for running Java processes. See Start the PHP Agent Proxy Manually.

Check the Configuration Properties

It is possible that the properties that the proxy uses to communicate with the Controller were not set properly.

You can modify these properties in the `php.ini` or `appdynamics_agent.ini` file, wherever they are set in your environment. See PHP Agent Configuration Settings.

Upgrade the PHP Agent

Related pages:

- Install the PHP Agent
- Uninstall the PHP Agent
- Resolve PHP Agent Installation Issues

If you are upgrading both the Controller and agents, first upgrade the Controller and then upgrade the agents.

Also, if you are upgrading multiple agents in your monitored environment, upgrade the agents for the tiers on which business transactions originate last. For more information about this requirement, along with Controller and agent compatibility information, see Agent and Controller Compatibility.

Upgrade the PHP Agent

1. Shut down the web server or `php-fpm`.

2. Copy the controller host, controller port, application name, tier name and node name property values from your `ini` file.
If you are running in multi-tenant mode, also copy the account name and account access key property values.

3. Recursively remove or rename the old AppDynamics PHP installation directory.

4. Download and extract the most recent agent tarball.

5. Run the installation script.
   Beginning in 4.1, you must pass the controller access key as one of the options to the installer.

   To find your access key, click  **> License > Account**.

6. Restart Apache or php-fpm.

> If you are using the agent to monitor PHP CLI without running a web server, you can omit steps 1 and 6.

Uninstall the PHP Agent

Related pages:

- Install the PHP Agent by Shell Script
- Install the PHP Agent by RPM

The method you use to install the agent varies based on the installation mechanism you used. If you are not sure which method was used, see Checking the Installation Method.

Uninstall an RPM-Installed PHP Agent

1. If one is running, shut down the web server. This may not apply for a PHP CLI application.
2. Run the following command:

   ```
   rpm -e appdynamics-php-agent-<version>
   ```

   where `<version>` is the package you installed. You can find the version using:

   ```
   rpm -qa | grep appdynamics-php-agent
   ```

Uninstall a Script-Installed PHP Agent

If you installed the agent using `install.sh`, uninstall that agent as follows:

1. Shut down the web server, if one is running. This may not apply for PHP CLI application.
2. From the PHP agent install directory, run the PHP installer with the `-u` option:

   ```
   install.sh -u
   ```

3. Delete the `<php_agent_install>` directory.

Checking the Installation Method

If you used the Agent Download Wizard, it might not be obvious which installer was used since the installer is called by the `runme.sh` wizard script.

Normally, if **RHEL and CentOS** was selected in the wizard, the RPM installer was used. If **All Other Linux OS** was selected in the wizard, the shell script installer was used. So use the appropriate uninstall command based on the command that was used to install the agent.

If you do not know which selection was checked in the wizard, run:

```
rpm -qa | grep appdynamics-php-agent
```

If this does not list any results, assume that the shell script installer was used to install the agent.

PHP Agent Configuration Settings

The PHP installers write information that the agent uses to communicate with the Controller to the AppDynamics Agent section of the PHP configuration files.

By default, the PHP Agent marks an HTTP error response, such as status code 4xx response as an error. When the following setting is added to the php.ini file, it will not mark the HTTP error response as an error.

```
agent.http_error_detection = false
```

This is different from some of the other AppDynamics application agents, which write this information to an XML file called `controller-info.xml`.

If you re-install the agent, the installer overwrites your settings only if you are using a fragments directory. It never overwrites the `php.ini` file. You can edit the PHP configuration files after installation to add, delete or modify these settings.

If a setting documented as *required* is not provided, the agent will not start. In this case, the agent logs the error to the web server error log. For example, if the `controller.Hostname` is not set you would see the following message in the Apache error log:

```
[AD agent] agent.controller.hostName is not set. Agent is disabled.
```

If after successful startup you remove a required setting or set it to an empty value, the change is ignored as long as the application is running, using the original value of the setting as it was at startup. If you change a required setting to a different valid value, restart the web server to apply the change.

The Controller information settings are described below:

agent.controller.hostName

This is the host name or the IP address of the AppDynamics Controller. Example values are 192.168.1.22, `myhost` or `myhost.abc.com`. This is the same host that you use to access the AppDynamics browser-based user interface. For an on-premises Controller, use the value for application server hostname that was configured when the Controller was installed.

This setting is required.

agent.controller.port

This is the HTTP(S) port of the AppDynamics Controller. This is the same port that you use to access the AppDynamics browser-based user interface.

If `agent.controller.ssl.enabled` is `true`, specify the HTTPS port of the Controller; otherwise, specify the HTTP port. See agent.controller.ssl.enabled.

For on-premises installations, port 8090 for HTTP and port 8181 for HTTPS are the defaults.

This setting is required.

agent.applicationName

This is the name of the logical business application that the instrumented node belongs to.

If a business application of the configured name does not exist, it is created automatically.

This setting is required.

agent.tierName

This is the name of the logical tier that this node belongs to.

This setting is required.

agent.nodeName

This is the name of the instrumented node.

This setting is required.

agent.accountName

This is the account name used to authenticate with the Controller.

This setting is required if the AppDynamics Controller is running in multi-tenant mode or if you are using the AppDynamics SaaS Controller. It specifies the account name for the agent to use to authenticate with the Controller. If you are using the AppDynamics SaaS Controller, the Welcome email sent by AppDynamics provides the account name.

This setting is not required if the Controller is running in single-tenant mode.

agent.accountAccessKey

This is the account access key used to authenticate with the Controller.

This setting is required. To find your access key, click the gear () icon in the upper right corner of the AppDynamics UI, then click **License**.

agent.controller.ssl.enabled

When set to `true`, this setting specifies that the agent should use SSL (HTTPS) to connect to the Controller. If `agent.controller.ssl.enabled` is `true`, set the `agent.controller.port` to the HTTPS port of the Controller.

agent.proxy_ctrl_dir

This specifies the initial control communication directory between the agent and the proxy. Needed only for a manual proxy start. See Start the PHP Agent Proxy Manually

agent.uniqueHostId

When you add this setting to the `php.ini` file in the PHP Agent, you can set a unique host ID for the App agents.

```
agent.uniqueHostId = <Insert uniqueHostId>
```

For example, `agent.uniqueHostId =  AppDynamics`

agent.http_error_detection=true

When adding this setting to `php.ini`, the PHP agent, marks HTTP error response, such as status code 4xx response as an error.

Node Reuse for PHP Agent

This page describes how to use the node reuse feature for the PHP Agent.

Reuse Node Name

This property is useful for monitoring environments where there are many VMs (Virtual Machines) with short life spans. When set to `true`, AppDynamics reuses the node names of historical VMs for new VMs. Therefore, this prevents the rapid increase of differently named nodes, especially when the nodes are identical processes that run over different times.

AppDynamics generates a node name with App, Tier, and Sequence number, and the node names are pooled. For example, the sequence numbers are reused when the nodes are purged (based on the node lifetime).

1. Edit the PHP configuration file, `php.ini` or `appdynamics_agent.ini` depending on your environment.
2. In the PHP configuration file, configure the following settings:

```
agent.nodeReuse
```

When the value of `agent.nodeReuse` is set to `true`, the agent uses reuse node name. The default value is `false`.

```
agent.nodeReusePrefix
```

When you configure the agent to reuse node names, use the `agent.nodeReusePrefix` property to specify the prefix that the controller uses to generate node names dynamically.

This setting is required when `agent.nodeReuse` is set to `true`.

An example of the reuse node feature:

With the following configuration in the `php.ini` file, the Controller generates a node name with the prefix *reportGen*. Node names will have suffixes -1, -2, and so on, depending on the number of nodes are running in parallel. The name of a node that is shut down and qualifies as a historical node that may be reused by a new node.

```
agent.nodeReuse=true agent.nodeReusePrefix=reportGen
```

Note

When `agent.nodeReuse` and a node name is used, the `agent.nodeReusePrefix` property is given precedence.

PHP Agent API User Guide

Related pages:

- PHP Agent API Reference
- PHP Supported Environments

This page describes the PHP Agent API and provides example use cases for the API.

About the PHP Agent API

The PHP Agent API enables you to:

- Define custom business transactions programmatically
- Provide correlation headers for entry points not supported by default detection
- Create custom exit calls to discover backends that are not automatically detected by the PHP Agent

Include the AppDynamics API Header

You should include the `appdynamics_api_header.php` file in your application to ensure that it works correctly if the agent is uninstalled or temporarily disabled. This file contains empty API functions that will prevent the application from throwing errors if the agent is not present.

The `appdynamics_api_header.php` file is located in the PHP Agent package in the same directory as the `install.sh` script.

To include the header file:

1. Copy `appdynamics_api_header.php` to where you keep the header files for the monitored application.
2. Make sure that `appdynamics_api_header.php` is in your include path.
3. Then add the following to your script:

```
require 'appdynamics_api_header.php';
```

Backend Detection with the MySQLi Driver

As noted on PHP Supported Environments, the PHP Agent does not work with PHP 5.2 applications that use the `new` keyword to instantiate a database backend with the MySQLi database driver. For example, AppDynamics does not detect the MySQLi backend created as follows in a PHP 5.2 application:

```
$db = new mysqli("localhost", "user", "password", "database");
```

You can work around this by using `mysqli_connect()` instead:

```
$db = mysqli_connect("localhost", "user", "password", "database");
```

Parts of a Script are Business Transactions

If you have a long PHP script application that performs a number of discrete tasks, but you want the agent to detect only one or more of them as business transactions, enclose the code for each of those tasks within `appdynamics_start_transaction()` and `appdynamics_end_transaction()` calls. The agent detects those blocks as separate business transactions. Otherwise, the agent detects the entire script as a single business transaction.

Handling Business Transaction Code Executing in a Loop

For a script-based CLI application that executes in a loop, perhaps fetching items from a database or remote service, you may want the agent to detect every iteration of the loop as a separate business transaction. In this case, enclose the code inside the loop within `appdynamics_start_transaction()` and `appdynamics_end_transaction()` calls.

If you do not do this, the agent will aggregate each iteration through the loop into a single business transaction.

In the following example, the agent detects a business transaction named `getItem` for every iteration.

```
while (true){
    appdynamics_start_transaction("getItem", AD_CLI);
    //your code goes here
    . . .
    appdynamics_end_transaction();
}
```

Correlating with an Upstream Service

If you have a distributed business transaction in which a tier needs to correlate with an upstream service that is not an entry point supported by the PHP Agent, you can maintain transaction correlation using `appdynamics_continue_transaction()` in the downstream tier, passing it the correlation header from the service.

You need to extract the correlation header from the service as shown in the following sample. The sample function extracts the correlation header from each message in an AMQP message queue and passes it to `appdynamics_continue_transaction()`.

After processing the message, it calls `appdynamics_end_transaction()`, which ends the continuation of the transaction on the calling tier. The `appdynamics_end_transaction()` call does not end the entire distributed transaction in the case where that tier makes a distributed call to another downstream tier.

```
function amqp_receive($exchangeName, $routingKey, $queueName) {
    $amqpConnection = amqp_connection();
    $channel = new AMQPChannel($amqpConnection);
    $queue = new AMQPQueue($channel);
    $queue->setName($queueName);
    $queue->bind($exchangeName, $routingKey);
    while($message = $queue->get()) {
        // Extracting the correlation header.
        echo("Message #".$message->getDeliveryTag()." '".$message->getBody()."'");
        echo("Correlation header: " . $message->getHeader("singularityheader"));
        // Passing correlation header to API.
        appdynamics_continue_transaction($message->getHeader("singularityheader"));
        doStuff($message);
        // End transaction.
        appdynamics_end_transaction();
    }
    if(!$amqpConnection->disconnect()) {
        throw new Exception("Could not disconnect !");
    }
}
```

If the service is not a supported entry point and you do not do this, the tier will not be correlated with the upstream transaction.

Make Socket-based HTTP Calls Example

While the API does not include built-in calls for socket-based HTTP call, you can implement monitoring of socket-based HTTP exit calls yourself as shown in the following example:

```
<?php

function doSocketHTTPCall($url, $corrHeader = null)
{
    $parts = parse_url($url);
    $fs = @fsockopen($parts['host'], isset($parts['port']) ? $parts['port'] : 80, $errno, $error);
    if (!$fs)
        return null;
    $send = "GET {$parts['path']} HTTP/1.1\r\n" .
        "Host: {$parts['host']}\r\n" .
        "Connection: Close\r\n";
    if ($corrHeader)
        $send .= "singularityheader: $corrHeader\r\n";
    $send .= "\r\n";
    fwrite($fs, $send);
    $data = stream_get_contents($fs);
    fclose($fs);
    return $data;
}

$url = 'http://httpstat.us/200';
$parts = parse_url($url);

$exitCall = appdynamics_begin_exit_call(
                AD_EXIT_HTTP,
                'HTTP Status Service',
                array('HOST' => $parts['host'],
                      'PORT' => (string)$parts['port'])
            );

doSocketHTTPCall($url);

appdynamics_end_exit_call($exitCall);
?>
```

Inject a Correlation Header into an HTTP Payload Example

The next example injects a correlation header into the socket-based HTTP payload.

```
<?php

$url = 'http://myhost.mydomain/continue.php';
$parts = parse_url($url);

$exitCall = appdynamics_begin_exit_call(
                AD_EXIT_HTTP,
                'HTTP Status Service',
                array('HOST' => $parts['host'],
                      'PORT' => (string)$parts['port'])
            );

$corrHeader = $exitCall->getCorrelationHeader();

doSocketHTTPCall($url, $corrHeader);

appdynamics_end_exit_call($exitCall);
?>
```

The next example shows how to use a non-exclusive flag to start an exit call that may be wrapping other exit calls. The outer socket-HTTP call is started, then the `file_get_contents()` call is processed by the agent normally, and finally, the outer call is finished. We also pass the exception object to report any errors. The end result is that both backends are displayed on the flowmap.

```
<?php

class SocketHTTPException extends Exception
{
}

$url = 'http://httpstat.us/200';
$javaTierURL = 'http://myhost.mydomain/process.jsp';
$parts = parse_url($url);

$exitCall = appdynamics_begin_exit_call(
                AD_EXIT_HTTP,
                'HTTP Status Service',
                array('HOST' => $parts['host'],
                      'PORT' => (string)$parts['port']),
                false
            );

$contents = file_get_contents($javaTierURL);

if (doSocketHTTPCall($url) == null) {
    $error = new SocketHTTPException("something bad happened");
}

appdynamics_end_exit_call($exitCall, $error);

?>
```

PHP Agent API Reference

Related pages:

- PHP Agent API User Guide

The PHP Agent APIs support custom business transaction definition and correlation. They provide a way to generate multiple business transactions in a single PHP request.

They also enable you to monitor exit calls that are not automatically detected by the PHP Agent.

API Reference Overview

The operations in the API fall into two categories, business transaction management and exit call management.

Business transaction management methods are:

- bool appdynamics_start_transaction($transaction_name, $entry_point_type)
- bool appdynamics_continue_transaction($correlation_header, $entry_point_type)
- bool appdynamics_end_transaction()

Exit call management methods are:

- ADExitCall appdynamics_begin_exit_call($type, $label, $properties, $exclusive=true)
- void appdynamics_end_exit_call(ADExitCall $exitCall, $exception = null)
- string ADExitCall::getCorrelationHeader()
- void ADExitCall::setSQLQueryInfo($querystring, $boundparams)

The following sections detail the methods in the API.

Start Transaction

Starts a custom business transaction.

Format

```
bool appdynamics_start_transaction($transaction_name, $entry_point_type)
```

Description

If the business transaction initiated by this call is not matched by an appdynamics_end_transaction() call, the transaction terminates at the end of the request or script.

Custom business transactions cannot be nested. If you call appdynamics_start_transaction() multiple times before calling `appdynamics_end_transaction()`, the last `appdynamics_start_transaction()` is used and the previous calls are discarded.

Parameters

`$transaction_name`: The name used for the transaction in the controller. The following characters are not allowed in transaction names: { } [ ] | & ;

`$entry_point_type`: Indicates the framework or protocol of the entry point. Valid entry point types are provided as PHP extension constants, shown below:

- `AD_WEB`
- `AD_MVC`
- `AD_DRUPAL`
- `AD_WORDPRESS`
- `AD_CLI`
- `AD_WEBSERVICE`

Entry point types are case sensitive.

Returns

True on success, false on failure.

Failure conditions are reported in the Apache log. Reasons for failure include:

- Invalid transaction name, contains disallowed characters
- Invalid entry point type
- Agent not initialized
- EUM headers were sent prior to the `appdynamics_start_transaction()` call.
- Correlation headers were sent prior to the `appdynamics_start_transaction()` call.

Continue Transaction

Correlates a custom business transaction with an upstream service.

Format

```
bool appdynamics_continue_transaction($correlation_header, $entry_point_type)
```

Description

Used by a downstream tier to correlate with a service that is not an entry point supported by the PHP Agent.

Parameters

`$correlation_header`: Correlation header of the upstream service with which to correlate.

It is the developer's responsibility to extract the correlation information from the service to provide the $correlation_header. See getCorrelationHeader() .

`$entry_point_type`: Optional parameter that indicates the framework or protocol of the entry point for the continued transaction. By default, the value is automatically set to AD_CLI when the application is running in CLI mode, and AD_WEB otherwise.

Use this parameter to indicate a different originating PHP application type for continuing transactions that cross PHP types, such as for a transaction originating at a PHP web application that crosses to a PHP CLI leg of the transaction.

Valid entry point types are provided as PHP extension constants:

- `AD_WEB`
- `AD_MVC`
- `AD_DRUPAL`
- `AD_WORDPRESS`
- `AD_CLI`
- `AD_WEBSERVICE`

Entry point types are case sensitive.

Return

Returns true on success, false on failure.

End Transaction

Ends a transaction created by a previous call to `appdynamics_begin_transaction()` or continued by `appdynamics_continue_transaction()`.

Format

```
bool appdynamics_end_transaction()
```

Description

When paired with an `appdynamics_continue_transaction() call`, this call ends the transaction on the tier being continued but does not end any subsequent calls downstream from that tier that are part of the distributed transaction.

If the business transaction is invalid or if there were no previous `appdynamics_begin_transaction()` or `appdynamics_continue_transaction()` calls in the request/script, this function returns false and throws a warning message that is captured by the application and reported as an exception.

Your application is recommended to wrap `appdynamics_begin_transaction()` and handle possible exceptions so that they are not logged as application exceptions.

Begin Exit Call

Marks the start of an exit call.

Format

```
ADExitCall appdynamics_begin_exit_call($type, $label, $properties, $exclusive=true)
```

Parameters

`$type:` The type of exit call. Must be one of the following:

- `AD_EXIT_HTTP`
- `AD_EXIT_DB`
- `AD_EXIT_CACHE`

- AD_EXIT_RABBITMQ
- AD_EXIT_WEBSERVICE

$label

Label for the exit call in the AppDynamics UI. Use a label under 40 characters long so that it fits in flowmaps.

$properties

An associative array of identifying properties—name/value pairs—for the exit call. Property names and values must be strings.

Each exit call type has its own properties. There is no validation of property names, but each exit type has traditionally used the names listed below.

type					
AD_EXIT_HTTP	"HOST"	"PORT"	"URL"	"QUERY_STRING"	
AD_EXIT_DB	"HOST"	"PORT"	"DATABASE"	"VENDOR"	"URL"
AD_EXIT_CACHE	"SERVER POOL"	"VENDOR"			
AD_EXIT_RABBITMQ	"HOST"	"PORT"	"EXCHANGE"	"ROUTING KEY"	
AD_EXIT_WEBSERVICE	"SERVICE"	"OPERATION"	"SOAP ACTION"	"VENDOR"	"URL"

For AD_EXIT_DB exit call type it is advisable to specify at least HOST, PORT and VENDOR, as these properties are used by the AppDynamics DB integration. The VENDOR property for AD_EXIT_DB backends should be one of the following:

- MYSQL
- POSTGRESQL
- SQLSERVER
- ORACLE
- SYBASE
- DB2

$exclusive: Boolean that Indicates whether the exit call is exclusive.

Only one exclusive exit call can be in progress at a time. For example, if this API is used to start an HTTP exit call and there is a mysql_connect() call immediately following it, the MySQL call will not be detected while the HTTP call is in progress. An exclusive exit call has to be explicitly ended before subsequent exit calls can be detected or initiated.

Exit calls are exclusive by default, but you can make them non-exclusive by setting this parameter to false.

See the last code sample in the 'Scenario: Application makes socket-based HTTP calls' in PHP Agent API User Guide for an example of how setting this flag to `false` can be used to support nested exit calls.

Return

Returns an instance of the `ADExitCall` class on success, `NULL` on error.

End Exit Call

Marks the end of an exit call.

Format

```
void appdynamics_end_exit_call(ADExitCall $exitCall, $exception = null)
```

Parameters

`$exitCall`: An object representing the exit call to be ended, and returned from its `appdynamics_begin_exit_call()`.

$exception: An exception object—either Exception class or one derived from it—specifying if an error occurred during the exit call.

Get Correlation Header

Returns the correlation header for this exit call. The returned correlation header can be passed to appdynamics_continue_transaction to correlate with this exit call.

See also the 'Scenario: Application makes socket-based HTTP calls" 'ample in PHP Agent API User Guide for an example of injecting the correlation header into an HTTP payload.

Format

```
string ADExitCall::getCorrelationHeader()
```

Set SQL Query Info

Sets the SQL query for this exit custom call.

Format

`void ADExitCall::setSQLQueryInfo($querystring, $boundparams)`

Parameters

`$querystring`: String containing the SQL query. Use question marks to indicate the bound parameters.

`$boundparams`: Optional comma-separated array of bound parameters as quoted strings.

Example

```
$exitCall->setSQLQueryInfo("SELECT * FROM mytable where id1=? and id2=?;", ('Susie', '12345'));
```

PHP Agent Logging

For the PHP Agent there is an agent log and a proxy log for each instrumented application.

Agent Log

The agent log is located at `$<php_agent_install>/logs/agent.log`. The log contains the transactions that the agent processes and then sends to the proxy.

The default pattern for agent log naming is the following:

- `agent.log`: the current log
- `agent.log.1`: most recent log
- `agent.log.2`: second most recent log
- `agent.log.3`: third most recent log
- `agent.log.4`: fourth most recent log
- `agent.log.5`: fifth recent log.

The agent creates and rotates a maximum of six log files. Maximum log size is 20 MB, which gives you a maximum of the most recent 120 MB of log data at one time.

Proxy Log

The proxy log is located `$<php_agent_install>/logs/proxy_$date.log.` This log contains the transactions that the proxy accepts from the agent and then sends to the Controller. See Proxy Logging and Dynamic Language Agent Proxy.

Filter Data

By default, AppDynamics PHP Agent sends transaction data to the Controller that your organization may classify as privileged information. Though this data is useful for diagnosis and troubleshooting, security considerations may require you to filter certain sensitive information from being displayed on the Controller. You can use the following for the security considerations:

- URL filters to exclude sensitive information from a URL in snapshot details.
- Data filters to exclude sensitive HTTP cookies.

Add a URL Filter

1. Edit the PHP configuration file, `php.ini` or `appdynamics_agent.ini` depending on your environment.
2. In the PHP configuration file, configure the following settings. If a value in the ini file for any of the following config contains any non-alphanumeric characters, it must be enclosed within double-quotes (") as per the PHP guidelines.

    - Config Delimiter: Specify the character that you want to use as the configuration delimiter for the sensitive data filter for redaction. The config delimiter must be chosen such that it should not occur in the redaction config strings. This value is REQUIRED.

        For example:

```
agent.sensitive_data_filter.configDelimiter = "|"
```

    - URL Delimiter: Specify the character that you want to use as URL segment endpoints. The agent splits the URL at each delimiter instance to create the segments. For HTTP, use the forward-slash character "/". For a forward slash, the agent does not split on the slashes immediately following the protocol. For example, "https://myapp.example.com/" constitutes a single segment. By default, the delimiter is "/" but this is REQUIRED for successful redaction.

        For example:

```
agent.sensitive_data_filter.delimiter = "/"
```

> ‘#’ and ‘;’ cannot be used as a delimiter or configDelimiter because the ini file considers it as a comment.

    - Segment: Specify a comma-separated list to indicate the segments that you want the agent to filter. Segment numbering starts from 1 and providing 0 or negative value fails to redact the segments with an error message in the agent logs. The segment numbers must be in the ascending order. This value is REQUIRED.

        For example:

```
agent.sensitive_data_filter.segment = "2,3"
```

    - Matchfilter: Specify the type of filter to be used to match the URL among - `NOT_EMPTY|EQUALS|STARTSWITH|ENDSWITH|CONTAINS|REGEX`. PERL standard must be followed if REGEX is used. The value is REQUIRED.

        For using this correctly, query parameters must not be considered for match-filtering. With an example of the call "https://myapp.example.com/sensitive/data?first_name=abc&last_name=xyz", to specify match-filter as STARTSWITH, it matches a specified string starting with the hostname “https://myapp.example.com” in this example. If the URL contains the port in the hostname, it must be present in the config string. Similarly for ENDSWITH, it will correspond to the last segment leaving out the query parameters, “data” in this case, as query parameters are never reported in the snapshots.

        For example:

```
agent.sensitive_data_filter.matchFilter = "CONTAINS"
```

    - MatchPattern: Specify the string that you want to be filtered by the match-filter. This value is REQUIRED.

        For example:

```
agent.sensitive_data_filter.matchPattern = "myapp"
```

    - ParamPattern: Specify the regular expression matching the query parameters to filter. PERL Standard should be followed for the regular expression. This value is OPTIONAL.

        For example:

```
agent.sensitive_data_filter.paramMatcher = "[a-z]+_name"
```

For example, the following configuration splits the URL on the "/" character and masks the second segment and the `ParamPattern` in the third segment of the URL. Here, the segmentation and obfuscation apply only to URLs containing "myapp".

```
agent.sensitive_data_filter.configDelimiter = "|"
agent.sensitive_data_filter.delimiter = "/"
agent.sensitive_data_filter.segment = "2"
agent.sensitive_data_filter.matchFilter = "CONTAINS"
agent.sensitive_data_filter.matchPattern = "myapp"
agent.sensitive_data_filter.paramMatcher = "[a-z]+_name"
```

The exit call to "https://myapp.example.com/sensitive/data?first_name=abc&last_name=xyz" breaks down to three segments: "https://myapp.example.com", "sensitive", and "data?first_name=abc&last_name=xyz". The Controller shows the masked values of the URL and the param-pattern display "https://myapp.example.com/*****/data?first_name=***&last_name=***" in the snapshot details.

If you do not use any values for the query parameters, the Controller does not mask any query parameters in the URL.

Redaction of Multiple URLs

Due to the limitations of the PHP configurations, if you want to redact multiple URLs separately, arguments must be written separated by the config delimiter as described below:

```
agent.sensitive_data_filter.configDelimiter = "|"
agent.sensitive_data_filter.delimiter = "/|/"
agent.sensitive_data_filter.segment = "1,2,3|1,4"
agent.sensitive_data_filter.matchFilter = "CONTAINS|ENDSWITH"
agent.sensitive_data_filter.matchPattern = "One|.php"
agent.sensitive_data_filter.paramMatcher = "[a-z]+_name|[a-z]+_name"
```

Each ‘|’ separated values correspond to an additional URL filter added. For ‘n’ number of separate URL filters, you need to have ‘n’ different ‘I’ configurations correspondingly. These filters work independently on the URLs and will redact based on the configurations specified for each filter. The first matching configuration is used for the redaction.

For a successful redaction, ensure that each configuration contains the same number of configuration segments.

Data Filter for Cookies

You can use Cookie filters to configure the agent to obfuscate sensitive information from the URLs in transaction snapshot details.

1. Edit the PHP configuration file, `php.ini` or `appdynamics_agent.ini` depending on your environment.
2. Add sensitive cookie filter element as directives:

    - AppDynamicsCookieMatchPattern: Specify a regular expression identifying cookies that must be redacted.

    For example:

```
agent.sensitive_data_filter.cookieMatcher = "cookieKey"
```

    - For masking multiple cookies values simultaneously, provide names of all those cookies separated by the config delimiter as a single string as follows:

```
agent.sensitive_data_filter.cookieMatcher = "PHPSESSID|X-CSRF-TOKEN|cookiekey"
```

If the config delimiter is present in the cookie name itself, change it to something else such that it should not occur in the cookie matcher strings.

PHP Agent Node Properties

This reference page contains information about app agent node properties. The properties are listed in alphabetical order.

max-business-transactions

The number of business transactions discovered once an agent is started. This is done to prevent business transaction metric explosion because an unsuitable discovery scheme can potentially produce thousands of transactions,

Warning: Changing this setting can affect the resource consumption of your deployment. Before you change this setting, verify that your application environment and Controller can handle any increased resource requirements.

Type:	Integer
Default value:	50
Range:	Minimum=N/A; Maximum=300
Platform(s):	PHP

on-demand-snapshots

Collect snapshots for all Business Transactions executed in this node. Does not need a restart.

Type:	Boolean
Default value:	false
Platform(s):	PHP

Python Agent

Related pages:

- Dynamic Language Agent Proxy

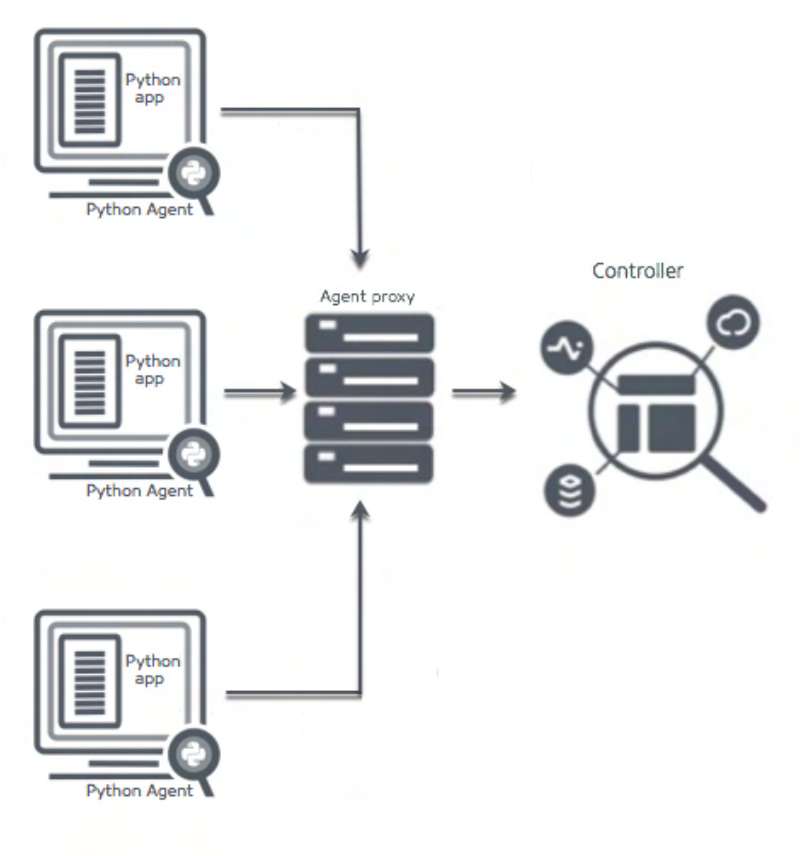
This page describes how to install and administer the Python Agent, the runtime agent used to monitor Python with AppDynamics Application Monitoring.

Python Agent Architecture

The Python Agent discovers, and maps and tracks metrics for business transactions, app services, and backends in your web application by injecting instrumentation into the Python application at runtime. The agent operates from inside the WSGI server that contains the instrumented application.

The agent proxy is a Java process that handles the communication between the Python Agent and Controller. The proxy reports performance metrics to the Controller, where the data is stored, analyzed, and presented.

As shown in the following figure, the proxy can serve multiple Python Agents simultaneously.



In pure Python environments, the proxy is automatically started when you start the Python Agent. In other types of environments, you need to start the proxy manually.

However the proxy is started, the commands that start the agent check whether the proxy is running before attempting to start it.

Install the Agent Overview

If you have never installed the agent, the best way to start is with the Getting Started Wizard in the Controller. This installer configures your agents with the values that you supply in the wizard.

If you downloaded the agent from PyPI, use the instructions in Install the Python Agent to instrument your application manually. In this case, see Python Agent Settings for a complete list of all the Python Agent settings that you can use to instrument the agent.

Getting Started with the Download Wizard

The Getting Started Wizard walks you through configuration steps and helps you download the agent. To access the wizard, from the home page of the Controller, click **Getting Started** and then **Python**.

The wizard provides for a minimally configured agent, with settings for the Controller connection, the business application name, and tier name. For advanced scenarios, see Install the Python Agent.

Python Supported Environments

Related pages:

- Python Agent

Python Agent Support

Python Versions

The Python agent supports Python 2.7, 3.4, 3.5, 3.6, 3.7, 3.8, and 3.9.

Operating Systems

The Python agent operates on any Linux distribution based on glibc 2.12+ and the x86 32-bit or x86 64-bit architecture.

- Support for Mac OS X will be deprecated from the 20.7.0 version of the Python Agent.
- Support for centos5 has been deprecated.

Python Frameworks and Protocols

Framework /Protocol	Version	Entry Point Type	Default Transaction Naming
WSGI	1.0	Python Web	First two segments of URI
Tornado	3.2 - 4.5 (all versions), Tornado 5.x without asyncio library, Tornado 6 (for Python 3.5.3+)	Python Web	First two segments of URI

AppDynamics has tested the Python Agent on Tornado, Django, Flask, CherryPy, Bottle, and Pyramid.

You can configure the agent to instrument any WSGI-based application or framework as Python Web, including but not limited to those listed below.

At present, the Python agent fully supports exception detection in Django, Flask, CherryPy, Bottle, Pyramid, and Tornado frameworks. Other WSGI frameworks and custom WSGI applications may install exception handlers that effectively hide some exceptions from the agent. In such cases, the agent will only detect exceptions during exit calls, missed exceptions that are propagated to the WSGI server, and exceptions reported via the custom business transaction API.

WSGI-Based Frameworks	Notes
Bottle	0.12.19
CherryPy	18.6.0
Django	3.1.5
Flask	1.1.2
PasteDeploy	2.1.0
Pyramid	1.10.5
mod_wsgi	4.7.1

Database Exit Points

Supported Database Exit Points	Version
cx_Oracle	5.1.x
MongoDB	3.1+
MySQL-Python	
mysqlclient	
MySQL Connector/Python	

Psycopg 2	
PyMySql	
TorMySql	

HTTP Exit Points

Supported HTTP Exit Points
httplib*
httplib2
requests
urllib
urllib2
urllib3
tornado.httpclient

* The agent detects calls to any external library built on top of `httplib`. Therefore, backend calls to such services, such as, boto, dropbox, python-twitter are detected and displayed as HTTP exit calls.

Cache Exit Points

Supported Cache Exit Points
Memcache
Redis-py

Install the Python Agent

Related Pages:

- SELinux Installation Issues

This page describes how to prepare the application environment and install the AppDynamics Python Agent.

Before You Begin

1. Verify support for your application environment at Python Supported Environments.
2. Verify that the machine where you will install the agent can access the Python Package Index.
3. Provide a WSGI-based application to monitor.
4. Verify that you can access the machine where the application runs as a user with privileges to install the agent software and restart the application. Verify that you have a user account with these privileges.
5. If you are using uWSGI, set `enable-threads=1` in the uWSGI configuration file. The agent requires multi-threading.
   There is a known incompatibility between the Python Agent and versions of uWSGI installed via OS packages managers such as 'apt-get'. For this reason, AppDynamics recommends installing uWSGI from pip to avoid this issue.
6. If the application to monitor runs in a virtual environment, activate the virtual environment. For example, the following source command activates a virtual environment:

   ```
   source /<path_to_virtual_environment>/bin/activate
   ```

   Activating a virtual environment is not necessary if the application runs in the global Python environment.

Install the Agent

Log in to the machine on which the Python application runs using appropriate user credentials, as follows:

- For a virtual environment, you need to be the user who owns the virtual environment.
- For the global Python environment, you need to run the install command as root.

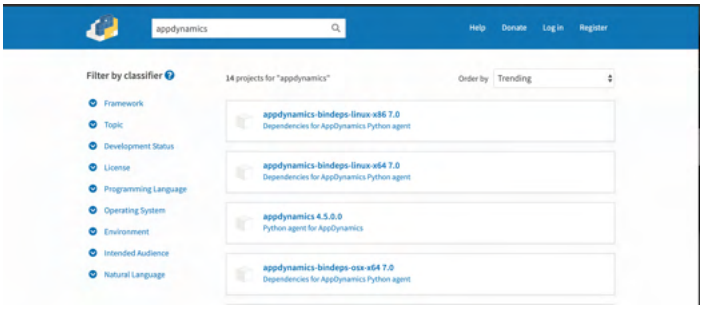
You can install the agent using one of the installation methods explained in the following sections.

pip Installation

To install or upgrade to the latest version of the agent, run the `pip install` command as follows:

```
pip install -U appdynamics
```

When there are multiple packages, you can locate the agent using the `pip list` command or using the List packages facility at https://pypi.python.org/pypi and then find `appdynamics` in the output. Here is a sample output:



For production deployments, AppDynamics recommends that you freeze the versions of dependencies so that they are not upgraded in production without first being deployed to your test/staging environments. The command to install or upgrade to a particular version of the AppDynamics Python Agent is:

```
pip install -U appdynamics==<released_agent_version>
```

For example:

```
pip install -U appdynamics==4.4.0.0
```

This command always installs the exact same version of the agent each time it is run.

Installation from AppDynamics Download Portal

1. Download `PythonAgent*.tar.bz2` file (based on target platform) from the portal and untar the contents to a folder.
   Run the following command to ensure that you have latest pip to support wheel installation:

   ```
   pip install --upgrade pip
   ```

2. Run the following command to install the wheel format:

   ```
   pip install -f <untar_folder> appdynamics
   ```

Configure the Agent

Provide a configuration file that specifies the required AppDynamics settings for agent-Controller communication. The file should be in Python ConfigParser format. The Python application user must have read access on the configuration file.

Note that lines in the configuration file must not start with spaces. Lines that begin with a # are comments and are ignored by the agent.

The following is a simple sample configuration file with minimum required settings.

```
[agent]
app = <app_name>
tier = <tier_name>
node = <node_name>

[controller]
host = <controller_host>
port = <controller_port>
ssl = true
account = <your AppDynamics controller account name>
accesskey = <your AppDynamics controller account access key>
```

Note the following points in the configuration:

1. The ssl settings determines whether the agent connects to the Controller by SSL. This is required for SaaS Controllers.
2. The `account` value is required if you are using a SaaS account or a multi-tenant on-premises controller. It defaults to `customer1` for a single tenant controller.
3. The accesskey is required for all controllers. To find your account name and access key, click  in the upper right corner of the AppDynamics UI, then click **License**.
4. Other settings, such as `ssl`, `http-proxy` or `wsgi_module`, may be required for your environment. See Python Agent Settings for a complete list of settings.

When you instrument an application using `pyagent run`, you pass the configuration file path as a parameter to the pyagent run command. In other deployments set the `APPD_CONFIG_FILE` environment variable as illustrated below in the samples for uWSGI with Emperor and Apache with mod_wsgi.

Instrument the Application

Which instrumentation instructions to use depends on how the application is deployed, from these deployment options:

- pyagent run
- uWSGI with Emperor
- Apache with mod_wsgi

Irrespective of your Python environment, if you built your application using PasteDeploy, you can install the Python Agent by modifying the PasteDeploy configuration. See PasteDeploy.

pyagent run

If you can control the way your WSGI server is invoked, you can instrument the application using `pyagent run`. This command runs your WSGI server with the Python agent enabled. This option is generally possible if you use a process launcher/manager that takes a command to execute. For example, frameworks managed by Supervisor, uWSGI without Emperor, init.d scripts, and so on.

To use the `pyagent run` command, prepend to your existing application run command the `pyagent run` command, passing the AppDynamics configuration file described in Configure the Agent as a parameter.

Do not overwrite `PYTHONPATH` for any reason. Doing so will prevent the `pyagent run` command from loading the agent. If you need to add to PYTHONPATH, use the pythonpath configuration variable. For example, these commands add `/foo` and `/bar` to the `PYTHONPATH` instead of overwriting it:

Correct way to add to PYTHONPATH:

```
pythonpath = /foo
pythonpath = /bar
```

Do not add the values to PYTHONPATH using the follwoing syntax. This is the wrong way to add to PYTHONPATH:

```
env = PYTHONPATH=/foo:/bar
```

Using supervisorctl

If you use `supervisorctl`, after updating your Supervisor configuration you must use the `supervisorctl reload` command to have the Python agent loaded. Supervisor does not re-read its configuration files when you use the `supervisorctl restart` command.

To verify that the agent was loaded, look for the Python agent log file. Its default location is `/tmp/appd/logs/<app_name>-<node_name>.log`. For example, if your application name is `myapp` and your node name is `mynode` as specified in the agent configuration file, and you have not changed the location of the log file, the log file will be `/tmp/appd/logs/myapp-mynode`.

If the log file exists, the agent was loaded. If the log file does not exist, the agent was not loaded, in which case you should try reloading the Supervisor configuration with `supervisorctl reload`.

Django and Flask

If your framework is Django or Flask, simply prepend `pyagent run` to your run command. For example, if your current run command looks like this:

```
gunicorn -w 8 -b '0.0.0.0:9000' example.app:application
```

Replace it with the following:

```
pyagent run -c <path_to_appdynamics_config_file> -- gunicorn -w 8 -b '0.0.0.0:9000' example.app:application
```

Other Pure Python WSGI-Based Frameworks

If you use a WSGI-based framework that is not Django or Flask:

1. In the AppDynamics configuration file, specify your WSGI application by setting the `APPD_WSGI_MODULE` directive to point to your app module. See Python Agent Settings.
2. Prepend `pyagent run` to your run command.
3. Run the AppDynamics-generated application.

For example, if your run command looks like this:

```
gunicorn -w 8 -b '0.0.0.0:9000' example.app:application
```

Replace it with these two commands:

```
pyagent run -c /path/to/appdynamics.cfg -- gunicorn -w 8 -b '0.0.0.0:9000'
appdynamics.scripts.wsgi:application
```

uWSGI with Emperor

If your environment is uWSGI with Emperor, you need to modify your WSGI configuration files and then manually launch the proxy.

uWSGI Emperor is a process manager specific to the uWSGI server. It does not allow you to control how the uWSGI processes that it manages are launched and therefore cannot be used with the `pyagent run` command.

The location of the WSGI configuration files is deployment-dependent. See http://uwsgi-docs.readthedocs.org/en/latest/Emperor.html for details of Emperor deployment.

To instrument an application for uWSGI with Emperor:

1. Create the configuration file described in Configure the Agent.
2. Modify the uWSGI configuration file. Do one of the following, depending on whether the configuration uses a module directive or a wsgi-file directive:
   Module Directive
   If the uWSGI configuration has a module directive such as the following:

   ```
   module = yourcompany.sample:app
   ```

   modify that configuration by changing the module setting and adding the `APPD_WSGI_MODULE` and `APPD_CONFIG_FILE` settings to look like this, assuming that you have stored the configuration file in `/etc/appdynamics.cfg`:

   ```
   env = APPD_CONFIG_FILE=/etc/appdynamics.cfg
   env = APPD_WSGI_MODULE=yourcompany.sample:app
   module = appdynamics.scripts.wsgi:application
   ```

   WSGI-File Directive
   If the uWSGI configuration has a `wsgi-file` directive:

   ```
   wsgi-file = /var/www/yourcompany/sample.py
   callable = app
   ```

   Modify the configuration to look like the following, assuming you have stored the configuration file in `/etc/appdynamics.cfg`:

   ```
   env = APPD_CONFIG_FILE=/etc/appdynamics.cfg
   env = APPD_WSGI_SCRIPT_ALIAS=/var/www/yourcompany/sample.py
   env = APPD_WSGI_CALLABLE_OBJECT=app
   module = appdynamics.scripts.wsgi
   ```

3. Before running any traffic through the instrumented application, manually launch the proxy by executing:

   ```
   pyagent proxy start
   ```

Apache with mod_wsgi

The Python Agent beta supports only `mod_wsgi` configurations that use `WSGIScriptAlias` that point to a single WSGI file. For example, the following type of configuration is supported:

```
WSGIScriptAlias /books /var/www/acme/bookstore/app.wsgi
```

If, instead, the script alias points to a directory, or if the script is using the `WSGIScriptAliasMatch` directive, contact python@appdynamics.com to discuss how the Python Agent can be deployed in your environment.

If the environment is Apache with `mod_wsgi` with a supported configuration as described above, you need to modify its `mod_wsgi` configuration files and manually launch the proxy.

To instrument an app for Apache with mod_wsgi:

1. Create the configuration file described in Configure the Agent.
2. Modify the `mod_wsgi` configuration file.
   If the `mod_wsgi` configuration file has an entry like this:

   ```
   WSGIScriptAlias /books /var/www/acme/bookstore/app.wsgi
   WSGICallableObject application
   ```

   modify it to look like this, assuming that you have stored the configuration file in `/etc/appdynamics.cfg`:

   ```
   SetEnv APPD_CONFIG_FILE /etc/appdynamics.cfg
   SetEnv APPD_WSGI_MODULE acme.bookstore:app
   WSGIScriptAlias /books /<path_to_virtualenv>/lib/python2.7/site-packages/appdynamics/scripts/wsgi.py
   ```

3. Before running any traffic through the instrumented app, manually launch the proxy by executing:

   ```
   pyagent proxy start
   ```

PasteDeploy

You can instrument a Python application built with PasteDeploy by modifying your PasteDeploy configuration to use a composite factory supplied by AppDynamics. This feature can be used to instrument applications described by the other deployment options if they were built with PasteDeploy.

The AppDynamics composite factory is named `egg:appdynamics#instrument`. It requires a parameter named `target` that points to the application to the original application and the full path to the APPD_settings.

To instrument an application built with PasteDeploy:

1. Manually launch the AppDynamics proxy:

   ```
   pyagent proxy start
   ```

2. In the PasteDeploy configuration file, rename the existing composite to a unique name.
   For example, if the existing composite configuration for an application named `metadata` is:

   ```
   [composite:metadata]
   use = egg:Paste#urlmap
   /: meta
   ```

   you could rename it:

   ```
   [composite:_orig_metadata]
   use = egg:Paste#urlmap
   /: meta
   ```

3. Create a new composite section for the `metadata` application above the original one that you just renamed, as follows:

   a. Give the name of the old renamed application to the new composite application.
   b. Configure it to use the AppDynamics composite factory: `egg:appdynamics#instrument`.
   c. Set its target to the renamed application.
   d. Set the AppDynamics configuration file environment variable, `APPD_CONFIG_FILE`, to the path of your configuration file. For example:

      ```
      [composite:metadata]
      use = egg:appdynamics#instrument
      target = orig_metadata
      APPD_CONFIG_FILE = /etc/appdynamics.cfg
      ```

      You can also set other APPD_ configuration variables here. For example, `APPD_LOGS_DIR=/var/log/appdynamics`.

4. Restart the application.

Start the Python Agent Proxy Manually

Related pages:

- Install the Python Agent
- Python Agent Settings

If you use the `pyagent run` command to instrument your application, the proxy is automatically started when you start the agent.

However, you may need to arrange to start the proxy separately from the agent. The most likely reason you would need to do this is because your Python environment uses uWSGI with Emperor or Apache with mod_wsgi. You can do so using the `pyagent run` command described here. Only use this command only if you need to launch the proxy manually.

Directory Configuration Setting

The command you use to start the proxy needs to read the directory configuration setting, APPD_DIR, the base directory for the AppDynamics Python Agent configuration. Make sure it is correctly set. The default is `/tmp/appd/`.

You can make this setting available to the proxy in any of the following ways:

- Setting the environment variable `APPD_DIR` manually.
- Setting the `APPD_CONFIG_FILE` environment variable to point to the configuration file and then setting the base directory in the configuration file. See Install the Python Agent.
- Passing in the path to the configuration file to the pyagent proxy command using -c (—config-file) option as illustrated below.

Start, Stop, and Restart Options for the pyagent proxy

To start, stop, or restart the proxy, use the `pyagent run` command, passing the desired operation and any options. The options include:

- —debug (-d): starts the proxy in debugging mode
- —no-watchdog: disables the proxy watchdog (not recommended)
- —config-file (-c): path to the Python Agent configuration file

Usage:

```
pyagent proxy start|stop|restart start_options  -- [jvmOption [jvmOption [...] ]
```

Examples:

```
pyagent proxy start -c appdynamics.config -d
pyagent proxy restart -c appdynamics.config -d
pyagent proxy stop -c appdynamics.config
```

When agent is stopped without stopping the proxy it continues to be shown as active for the next 10 minutes. To avoid the application from reappearing in the controller, it is recommended that you wait for at least 10 minutes before deleting the application after the agent shut down.

Upgrade the Python Agent

If you are upgrading both the Controller and agents, first upgrade the Controller and then upgrade the agents.

To upgrade the Python to the latest version of the 4.5 agent, run this command:

```
pip install -U appdynamics\<4.5
```

The upgrade will take effect the next time you restart the agent.

Uninstall the Python Agent

Related pages:

- Python Agent

You can remove Python Agent instrumentation from your application by reverting deployment script changes made when installing the agent to remove all `pyagent` commands and APPD environment variables.

Removing instrumentation is generally sufficient for most purposes when it comes to uninstalling the Python Agent, however, you can additionally remove the Python Agent packages using `pip`.

Because `pip install appdynamics` installed a separate dependencies package as well as the agent, two `pip uninstall` commands are required:

```
pip uninstall appdynamics
pip uninstall <python_agent_package>
```

If you do not know which package is installed, you can find out using `pip freeze` and `grep`:

```
pip freeze | grep appdynamics
```

Python Agent Settings

You can configure operating settings for the Python Agent using a configuration file or by setting environment variables in the application environment. This page lists the Python Agent settings.

See Configure the Agent for information about the configuration file.

> Older version of the environment variables are backward compatible and are deprecated which will eventually be removed in future releases. You will have to update the environment variables to the latest version.
> The new environment variables take precedence over the deprecated variables in case both are set in an environment (eg. `APPDYNAMICS_AGENT_APPLICATION_NAME` takes precedence over the deprecated `APPD_APP_NAME` if both are set).

[agent]

Directive	Description	Example	Default	Environment Variable
`app`	App Name	`MyApp`	*Required*	`APPDYNAMICS_AGENT_APPLICATION_NAME`
`tier`	Tier Name	`web-fe`	*Required*	`APPDYNAMICS_AGENT_TIER_NAME`
`node`	Node Name	`web-fe1`	*Required*	`APPDYNAMICS_AGENT_NODE_NAME`
`dir`	Base directory for files related to the AppDynamics agent	`/mysite/appd/agent/python/`	`/tmp/appd/`	`APPDYNAMICS_AGENT_BASE_DIR`
`nodereuse`	Reuse Node Name	`true` or `false`	*Optional*	`APPDYNAMICS_AGENT_REUSE_NODE_NAME`
`nodereuseprefix`	Reuse Node Name Prefix	`My Node Name`	*Optional*	`APPDYNAMICS_AGENT_REUSE_NODE_NAME_PREFIX`
`uniquehostid`	Unique host id for the app agents		Optional	`APPDYNAMICS_AGENT_UNIQUE_HOST_ID`

[wsgi]

Directive	Description	Example	Default	Environment Variable
`script`	Path to WSGI script file	`/var/www/acme/bookstore.py`	*n/a*	`APPDYNAMICS_WSGI_SCRIPT_ALIAS`
`callable`	Name of WSGI callable in script/module	`app`	`application`	`APPDYNAMICS_WSGI_CALLABLE_OBJECT`
`module`	Fully-qualified name of app module	`acme.bookstore:app`	*n/a*	`APPDYNAMICS_WSGI_MODULE`

If both the `script` and `module` directives are specified, the `module` directive takes precedence.

The `module` value may be the fully-qualified name of a module, or it may be the fully-qualified name of a module followed by a colon and the name of the WSGI callable in that module. In the latter form, the `module` directive overrides the `callable` directive.

Both the long-form of the `module` directive and the `callable` directive may take either the name of a symbol, or the name of a symbol followed by an empty pair of parentheses. In the latter form, the callable is taken to be the result of calling the callable specified by the directive. This latter form can be used with Django; for example:

```
module = django.core.handlers.wsgi:WSGIHandler()
```

[log]

Directive	Description	Example	Default	Environment Variable
`dir`	The directory to write proxy and agent logs to	`/var/log/appdynamics`	`/tmp/appd/logs`	`APPDYNAMICS_LOGS_DIR`
`level`	The level to log at one of: warning, info, or debug	`debug`	`warning`	`APPDYNAMICS_LOGGING_LEVEL`
`debugging`	On to write DEBUG level logs to stderr and log files	`on`	`off`	`APPDYNAMICS_DEBUG_LOG`

[controller]

This section specifies configuration for the AppDynamics Controller.

Directive	Description	Example	Default	Environment Variable
`host`	Controller host	`mycontroller.example.org`	*Required*	`APPDYNAMICS_CONTROLLER_HOST_NAME`
`port`	Controller port	`9000`	8090 for HTTP on-prem 443 for HTTPS, in which case SSL must also be set	`APPDYNAMICS_CONTROLLER_PORT`
ssl	Is SSL set be used to talk to the controller? `on` or `off`	`on`	`off`	`APPDYNAMICS_CONTROLLER_SSL_ENABLED`
`account`	AppDynamics Controller account	`user1`	For a single-tenant controller defaults to `customer1`. Otherwise required.	`APPDYNAMICS_AGENT_ACCOUNT_NAME`
`accesskey`	AppDynamics Controller account access key	`XC6v2n8m2$543`	*Required*. To find your account name and access key, click  in the upper right corner of the AppDynamics UI, then click **License**.	`APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY`
`certfile`	Controller Certificate This certfile must be the default agent keystore (found under `site-packages/appdynamics_proxysupport/lib/security/cacerts`) with necessary additional `.pem/.crt` certs embedded within it using keytool.	/tmp/ssl/cacerts	Optional	`APPDYNAMICS_CONTROLLER_SSL_CERTFILE`

[controller:http-proxy]

If you need to use an HTTP proxy to talk to your Controller, use this section to configure the HTTP proxy.

Directive	Description	Example	Default	Environment Variable
`host`	HTTP proxy host	`proxy.example.org`	*n/a*	`APPDYNAMICS_HTTP_PROXY_HOST`
`port`	HTTP proxy port	`8090`	`80`	`APPDYNAMICS_HTTP_PROXY_PORT`
`user`	HTTP proxy user	`proxyuser`	*n/a*	`APPDYNAMICS_HTTP_PROXY_USER`
`password-file`	HTTP proxy password file	`/etc/http-proxy.passwd`	*n/a*	`APPDYNAMICS_HTTP_PROXY_PASSWORD_FILE`

[proxy]

Directive	Description	Example	Default	Environment Variable
`max-heap-size`	Max heap size for proxy	450m	300m	`APPDYNAMICS_MAX_HEAP_SIZE`
`min-heap-size`	Min heap size for proxy	100m	50m	`APPDYNAMICS_MIN_HEAP_SIZE`
`max-perm-size`	Max permanent generation size	150m	120m	`APPDYNAMICS_MAX_PERM_SIZE`
`proxy-debug-port`	Port number to which to attach the JAVA debugger	8092	None	`APPDYNAMICS_PROXY_DEBUG_PORT`
`start-suspended`	Specifies whether to debug proxy startup with a JAVA debugger.	on	off	`APPDYNAMICS_START_SUSPENDED`
`debug-opt`	Specifies the debug opt for debugging	-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=8090	None	`APPDYNAMICS_DEBUG_OPT`
`agent`	Specifies the Agent type (Eg PYTHON_APP_AGENT, NODEJS_APP_AGENT etc)	NODEJS_APP_AGENT	PYTHON_AGENT	`APPDYNAMICS_AGENT_TYPE`

If `PROXY_DEBUG_PORT` is defined, then `-agentlib:jdwp=transport=dt_socket,server=y,suspend=${START_SUSPENDED},address=${PROXY_DEBUG_PORT}` is used as the debug opt. `PROXY_DEBUG_PORT` takes priority over `DEBUG_OPT` if both are specified. Also ensure that `PROXY_DEBUG_PORT` is not set if you want to use a self defined `DEBUG_OPT`.

[eum]

Directive	Description	Example	Default	Environment Variable
`disable-cookie`	If set, the agent does not add EUM correlation data to WSGI response headers.	`on`	`off`	`APPDYNAMICS_EUM_DISABLE_COOKIE`
`user-agent-whitelist`	If specified overwrites the default whitelist for user agent added as EUM correlation data headers Use this setting to specify alternate user agents as a comma separated list. Use '*' to allow all user agents.	`'iPad, Android'`	`'Mozilla, Opera, WebKit, Nokia'`	`APPDYNAMICS_EUM_USER_AGENT_WHITELIST`

[services:snapshot]

Directive	Description	Example	Default	Environment Variable
`exit-call-details-length`	Specifies the number of characters in the details string describing exit calls in transaction snapshots.	`200`	`100`	`APPDYNAMICS_EXIT_CALL_DETAILS_LENGTH`

[services:transaction-monitor]

Directive	Description	Example	Default	Environment Variable
`bt-max-duration-ms`	Maximum duration of a business transaction in milliseconds.	`60000`	`120000`	`APPDYNAMICS_BT_MAX_DURATION_MS`

Node Reuse for Python Agent

Reuse Node Name

To reuse node names in AppDynamics, you have to set `nodereuse` property to `true`. When you set this property to true, you do not have to supply a node name, but you do need to provide a node name prefix using `nodereuseprefix` .

> When `nodereuse` and a node name is used, the `nodereuseprefix` property is given precedence.

This property is useful for monitoring environments where there are many VMs (Virtual Machines) with short life spans. When set to true, AppDynamics reuses the node names of historical VMs for new VMs. This avoids a proliferation of differently named nodes in AppDynamics over time, particularly when the nodes are essentially identical processes that run over different times.

AppDynamics generates a node name with App, Tier, and Sequence number, and the node names are pooled. For example, the sequence numbers are reused when the nodes are purged (based on the node lifetime).

System Property: `nodereuse`

Type: Boolean

Default: False

Required: No

Reuse Node Name Prefix

When you configure the agent to reuse node names, use this property to specify the prefix that the Controller uses to generate node names dynamically.

System Property: `nodereuseprefix`

Type: String

Default: None

Required: When `nodereuse=true`

> Reuse node feature example:
>
> With the following configuration, the Controller generates a node name with the prefix "reportGen". Node names will have suffixes -1, -2, and so on, depending on the number of nodes are running in parallel. The name of a node that is shut down and qualifies as a historical node may be reused by a new node.
>
> ```
> nodereuse=true nodereuseprefix=reportGen
> ```

Python Agent Debugging and Logging

Logging Settings

Logging is configured in the logging section of the AppDynamics configuration file. See Python Agent Settings and Python Agent.

Logging Levels

Set the logging level for the amount of detail that you want to see: `INFO`, `WARNING` or `DEBUG`. The default is level is `WARNING`.

For maximum debugging information, set the logging level to `DEBUG`. Debug-level logs show the information that the Python Agent is sending to the proxy, which can be useful for debugging the agent.

For quick debugging, set the debugging configuration option to on. This causes debug-level logs to be written to both stderr and log files.

This is the behavior you can expect with various combinations of logging level and debug settings:

The various combinations of logging level and debug settings produce behavior as follows:

- `level=info, debugging=off`: Info-level logs are written to the log file
- `level=warning, debugging=off`: Warning-level logs are written to the log file
- `level=debug, debugging=off`: Debug-level logs are written to the log file
- `level=info, debugging=on`: Debug-level logs are written to the log file and to stderr
- `level=warning, debugging=on`: Debug-level logs are written to the log file and to stderr
- `level=debug, debugging=on`: Debug-level logs are written to the log file and to stderr

The Proxy logging level is separate and is not affected by the configuration settings of the Agent logging levels. Proxy logging level is "info" by default. Typically, you can redirect that output to a file if desired. In case if you wish to increase the logging level output, you must do so by starting the Proxy manually. First assign the new logging level within the Proxy's configuration file (`proxy/conf/logging/log4j.xml`) and then invoke the Proxy task.

There is no way to automatically adjust the Proxy logging level from the Agent configuration file settings.

Location and Names of Agent Log Files

Agent logs are written in the directory specified by the dir directive, `APPD_LOGS_DIR` in the logging section of the configuration file.

For each node, the log is output to a file named `APP_NAME-<NODE_NAME>.log`. These files live in the log dir directory, `APPD_LOGS_DIR`.

The agent creates and rotates a maximum of five log files. Maximum log size is 20 MB, which gives you a maximum of the most recent 100 MB of log data at one time.

For example:

1. myApp-myNode.log
2. myApp-myNode.log.1
3. myApp-myNode.log.2
4. myApp-myNode.log.3
5. myApp-myNode.log.4

Location of Proxy Logs

The proxy logs are always written to dir `APPD_LOGS_DIR`, which defaults to `/tmp/appd/logs`, `APPD_DIR/logs`.

Python Agent API

Related pages:

- Python Agent API Guide
- Python Agent API Reference

The Python Agent APIs provide facilities to support:

- Custom business transactions
- Custom exit points
- Custom data collectors

Python Agent API Guide

On this page:

Related pages:

- Python Agent API Reference
- Python Supported Environments
- Business Transactions
- Backend Detection Rules

The Python Agent APIs:

- Let you programmatically define custom business transactions that would not automatically be detected by the agent.
- Let you create custom exit calls to discover and monitor backends and that are not automatically detected by the agent.

If your instrumented application starts up and you do not see all the business transactions or backends that you expect to see, first check Python Supported Environments to see if your framework or exit point is supported. If not, you can use the APIs to instrument your code manually.

Custom Business Transactions

You can use the `start_bt()` and `end_bt()` methods to surround the code that you want to monitor as a custom business transaction.

Or you can use the "bt" context manager. Consider using the `bt` context manager where you start and end the business transaction in the same code. For example, where you can wrap the whole business transaction in a `with` statement.

For example, given the code:

```
setup()
do_work()
teardown()
```

you want to report `do_work()` as a business transaction.

Use start_bt() and end_bt()

This example uses `start_bt()` and `end_bt()` to create a business transaction named `do_work`.

```
from appdynamics.agent import api as appd
setup()

bt_handle = appd.start_bt('do work')
try:
    do_work()
except Exception as exc:
    raise
finally:
    appd.end_bt(bt_handle, exc)

teardown()
```

Use bt context manager

If the business transaction starts and ends in the same context, you can use the `bt` context manager instead. This is simpler:

```
setup()

with bt('do work'):
    do_work()

teardown()
```

Custom Exit Calls

You can use the `start_exit_call()` and `end_exit_call()` methods to create a custom exit call from a specific business transaction to a backend that the Python Agent does not automatically detect.

The business transaction must be a custom business transaction.

If you want to make a custom exit call from a business transaction that is normally automatically detected, you can exclude that business transaction to prevent it from being automatically detected and then create it as a custom business transaction. This enables you to get the BtHandle that you need to create the custom exit call. See Configure Python Web Custom Match and Exclude Rules for information on excluding a business transaction.

Given the code:

```
try:
    db = custom_db.connect(host='financials-lb', port=3456)
    all_employees = db.query_path('/financials/employees')
    individual_contributors = all_employees.filter(lambda r: r.level < 3)
    salaries_by_dept = individual_contributors.sum(value='salary', group='dept', as='total')

    for dept, total in salaries_by_dept.extract('dept', 'total'):
        report_salary_data(dept, total)
```

You want to send the query via an exit call to a proprietary database.

You want the database to be labeled `Financials Database` in the Controller UI.

You want the backend properties that appear in the backend dashboard to appear as:

Host:

`financials-lb`

Port:

`3456`

Vendor:

custom db

The following examples assume you are wrapping the exit call in a custom business transaction named `department rollup`, created in another part of your code.

Use start_exit_call() and end_exit_call()

This example uses `start_exit_call()` and `end_exit_call()`.

```
from appdynamics.agent import api as appd
appd.init()

# Set the identifying properties
FINANCIALS_ID_PROPS = {'Host': 'financials-lb', 'Port': 3456, 'Vendor': 'custom db'}

with appd.bt('department rollup') as bt_handle:
    # Start the exit call
    exit_call = appd.start_exit_call(bt_handle, appd.EXIT_DB, 'Financials Database', FINANCIALS_ID_PROPS)
    exc = None

    try:
        db = custom_db.connect(host='financials-lb', port=3456)
        all_employees = db.query_path('/financials/employees')
        individual_contributors = all_employees.filter(lambda r: r.level < 3)
        salaries_by_dept = individual_contributors.sum(value='salary', group='dept', as='total')

        for dept, total in salaries_by_dept.extract('dept', 'total'):
            report_salary_data(dept, total)
    except Exception as exc:
        raise  # Assuming something above handles exceptions for you
    finally:
        #End the exit call
        end_exit_call(exit_call, exc)
```

Use exit_call context manager

If the business transaction starts and ends in the same context, you can use the simpler `exit_call` context manager instead.

```
from appdynamics.agent import api as appd
appd.init()

with appd.bt('department rollup') as bt_handle:
    with appd.exit_call(bt_handle, appd.EXIT_DB, 'Financials Database', FINANCIALS_ID_PROPS):
        db = custom_db.connect(host='financials-lb', port=3456)
        all_employees = db.query_path('/financials/employees')
        individual_contributors = all_employees.filter(lambda r: r.level < 3)
        salaries_by_dept = individual_contributors.sum(value='salary', group='dept', as='total')

        for dept, total in salaries_by_dept.extract('dept', 'total'):
            report_salary_data(dept, total)
```

The next example starts a custom exit call to a Cassandra backend from a business transaction that was auto-detected by the Python Agent default Flask instrumentation. It uses the Flask import feature to get the request object which it passes to `appd_get_active_bt_handle()`.

Get bt handle using the flask request context

```
from flask import request
from appdynamics.agent import api as appd

@app.route('/metrics/recent')
def metrics_recent():
    bt = appd.get_active_bt_handle(request)  # Get the active BT from the Flask request object
    with appd.exit_call(bt, appd.EXIT_DB, 'cassandra time-series', {'VENDOR': 'Cassandra', 'SERVER POOL':
'10.0.0.1'}):
        load_recent_data_from_cassandra()
```

Other supported frameworks have different mechanisms for getting the request object.

Python Agent API Reference

On this page:

Related pages:

- Install the Python Agent
- Python Agent Settings
- Python Agent API Guide

The Python Agent APIs provide facilities to support:

- Custom business transactions
- Custom exit points
- Custom data collectors

List of the APIs:

Access to the APIs

Before your app can access the Python Agent APIs, install the Python Agent using:

```
pip install appdynamics
```

Then at the top of the instrumented application add:

```
from appdynamics.agent import api as appd
```

Start and Stop Agent

init(*environ*=None, *timeout_ms*=NO_TIMEOUT)

Initializes the Python Agent.

Call this function at the very beginning of your instrumented application, preferably before any other imports and before creating any other threads.

Agent initialization occurs `asynchronously. init()` returns immediately if no timeout is specified but it is possible that the agent will not immediately be ready to report business transactions. In this case, the start_bt() method will return `None`.

This function will never raise an exception.

Returns True if the agent is properly configured, False if the agent is not properly configured.

To be properly configured the agent configuration file must contain the minimal settings and possibly some additional settings if they are required for your environment. See Configure the Agent in Install the Python Agent for the minimal settings and Python Agent Settings for the complete list.

environ: dict, optional

If specified, a dictionary of environment variables to use to override the agent's configuration, derived from the actual OS environment variables.

timeout_ms: int or None, optional

Timeout interval in milliseconds.

By default, `init()` returns immediately, even if the Python agent has not received its configuration from the controller. If timeout_ms is None, init() blocks until the agent is properly configured. Otherwise, `init()` waits for up to timeout_ms for agent to become properly configured.

shutdown(timeout_ms=None)

Shuts down the Python Agent. The agent stops reporting metrics to the Controller.

This function ends all active business transactions and waits for them to finish reporting to the controller before returning.

timeout_ms: int, optional

By default, this function waits until all pending business transactions have been reported before returning. If `timeout_ms` is set, this function will return after `timeout_ms`, regardless of whether all business transactions have been reported.

Business Transaction Management

start_bt(*name*, *correlation_header*=None)

Starts a business transaction of the specified name.

There can only be one active business transaction per thread. Attempts to start subsequent business transactions in the same thread will return None.

Returns a BtHandle on starting the transaction or None if no transaction was started.

name: str

Name of the business transaction being started. The following characters cannot be used in business transaction names: `{ } [ ] | & ;`

correlation_header: str, optional

If specified, a correlation header that has been generated by another AppDynamics agent and passed to this agent as a string. A correlation header provides information to enable this transaction to correlate with a transaction that is upstream to this one.

end_bt(*bt_handle*, *exc*=None)

Ends the business transaction identified by bt_handle.

bt_handle: BtHandle

Identifies the business transaction being ended. The handle is returned by start_bt().

exc: Exception, optional

If an exception occurred during processing this business transaction that you have caught with a try-except block, and you wish to report the exception as part of the business transaction, pass the exception as the `exc`.

add_snapshot_data(*bt_handle*, *key*, *value*)

Attaches custom data to transaction snapshots generated for the business transaction identified by `bt_handle`.

The custom data is exposed in the `USER DATA` tab of the transaction snapshot details in the Controller UI.

The custom data is added to the business transaction but reported only when a transaction snapshot is generated. See Transaction Snapshots for information on when transaction snapshots are generated.

bt_handle: BtHandle

Identifies the business transaction to which custom data is added. The handle is returned by start_bt().

key: bytes or unicode

Name of the data item to be reported in the snapshot.

If passed as bytes, the buffer must represent a UTF-8 encoded string.

value: any

Value of the data.

If passed as bytes, the data is treated as a UTF-8 encoded string. If passed as unicode, it is directly reported.
All other objects are converted to `str(value)`.

get_active_bt_handle(request)

Returns a BtHandle to the business transaction associated with the request object or None if no such business transaction was found.

This is useful for passing a business transaction handle to another API function, such as `start_exit_call()` when the business transaction was created by the default Python Agent instrumentation rather than through the APIs.

request object

The request object associated with an active business transaction. Varies depending on the framework in which the business transaction was started.

See the documentation for the applicable framework to find out how to get the request object.

Framework	Documentation for the Request Object
Flask	https://flask-cn-doc.readthedocs.io/en/latest/reqcontext.html
Django	https://docs.djangoproject.com/en/1.8/ref/request-response/
CherryPy	https://cherrypy.readthedocs.io/en/latest/basics.html
WSGI environ	https://www.python.org/dev/peps/pep-0333/

Exit Call Management

start_exit_call(*bt_handle*, *exit_type*, *display_name*, *identifying_properties*, *optional_properties*=None)

Starts a custom exit call from within the specified business transaction.

There can be only one active exit call per business transaction. Attempts to start subsequent exit calls will return None.

Returns an ExitCallHandle that identifies this exit call.

bt_handle: BTHandle

Identifies the business transaction making the exit call. Handle is returned from start_bt().

exit_type: int

The type of exit call. Valid values are:

- `appd.EXIT_HTTP`
- `appd.EXIT_DB`
- `appd.EXIT_CACHE`
- `appd.EXIT_QUEUE`

display_name:str

Name used to identify this exit call in the controller. This is the label that will be used to display the exit call in the AppDynamics UI.

identifying_properties: dict

A dictionary of name/value pairs that uniquely identify the downstream component being called.

In the Controller UI, these properties are visible in the upper right panel of the backend dashboards.

The key of the dict is the name of the property; the value is the value of the property.

operation: str, optional

A string describing the operation that is being performed, such as the URL of an HTTP request or an SQL query.

optional_properties: dict, optional

A dictionary of name/value pairs that identify additional properties for this exit call. In the Controller UI, these properties are visible in the Exit Calls and Async Activities modal in the snapshot drill-down pane.

end_exit_call(exit_call_handle, exc=None)

Ends the exit call identified by `exit_call_handle`.

exit_call_handle: ExitCallHandle

Identifies the exit call being ended. The handle is returned from start_exit_call().

exc: Exception

If an exception occurred during the exit call that you have caught with a `try-except` block, and you wish to report the exception as part of the exit call, pass the exception as the `exc` argument.

make_correlation_header(bt_handle, exit_call_handle)

Make a correlation header for a custom exit call.

If you are performing custom exit calls to other instrumented tiers, adding a correlation header allows continuing business transactions on the downstream tier.

It is up to you to send the header, as well as parse it at the other end and pass it to start_bt().

Returns a tuple of header name and header value, or None if correlation is disabled.

bt_handle: BtHandle

A handle identifying the business transaction.

exit_call_handle: ExitCallHandle

A handle identifying the exit call.

Context Managers

bt(*name*, *correlation_header*=None)

Context manager for reporting some work as a business transaction. Yields a `BtHandle`.

If you need to start and end the transaction in different places in the code, use start_bt() and end_bt().

exit_call(*bt_handle*, *exit_type*, *display_name*, *identifying_properties*, *optional_properties*=None)

Context manager for adding exit calls to a business transaction. Yields an `ExitCallHandle`.

If you need to start and end exit calls in different places in your code, you can use start_exit_call() and end_exit_call().

HTTP Status Codes

HTTP status codes will not cause error transactions when you use the using the Python Agent API to instrument your application, regardless of the error detection configuration.

Serverless APM for AWS Lambda

Deployment Support

SaaS

AppDynamics Serverless Application Performance Monitoring (Serverless APM) for AWS Lambda gives you visibility into the performance of your application's components that run as functions on serverless compute environments.

Serverless APM gives you an end-to-end view of applications through business transaction correlation. Serverless APM correlates business transactions between AWS Lambda functions and:

- Components instrumented with AppDynamics app agents
- Devices instrumented with AppDynamics End User Monitoring (EUM) agents

Additionally, Serverless APM correlates business transactions through serverless functions, such an AWS Lambda function that invokes another function.

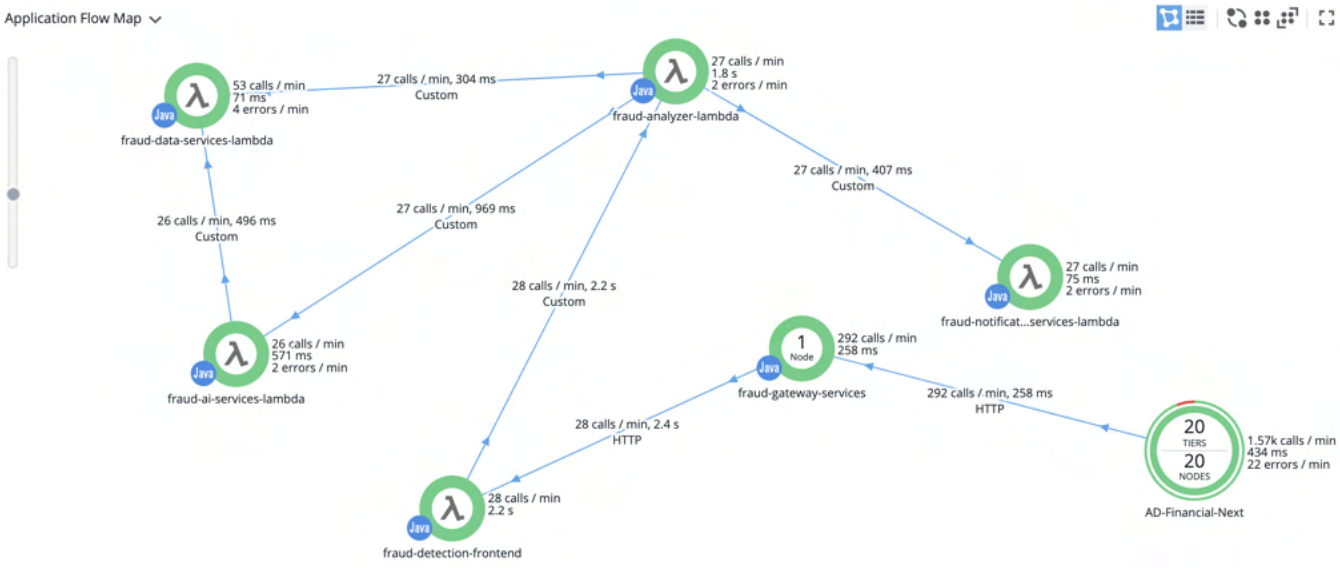
Serverless APM in the Controller

When business applications contain serverless functions, the Controller experience differs slightly in flow maps, dashboards and metric browser pages, and health rules.

Flow Maps

Flow maps are a dynamic visual representation of your monitored environment's components and activities. When business applications have serverless functions, you cannot view the nodes within flow map tiers because of the opaque nature of serverless platforms. A serverless icon replaces the node count in the application flow map.

The screenshot below depicts an application composed of AWS Lambda tiers:



Dashboards and Metric Browser Pages

You can view AWS Lambda functions on your application dashboards and metrics pages. All functionality is identical to that which you would get with any other tier type, with the exception of node-level granularity. AWS Lambda tiers do not offer node-level dashboards or metrics because serverless platform runtime instances spin up and down on demand.

Health Rules

When you configure a health rule for an application comprised of serverless functions, you can choose to monitor the serverless tiers or business transactions that originate in or flow through the serverless functions. See Configure Health Rules.

Get Started

Serverless APM requires a subscription through AWS Marketplace. See Subscribe to Serverless APM for AWS Lambda to get started.

If you have already subscribed to Serverless APM, you can begin instrumentation. See:

- Learn how Serverless APM correlates business transactions with the Serverless Tracer.
- Set environment variables to enable Tracer-to-Controller communication.
- Instrument the Java Serverless Tracer, Node.js Serverless Tracer, or Python Serverless Tracer. The tracer's language matches your AWS function code's implementation.

Subscribe to Serverless APM for AWS Lambda

> This page contains links to AWS documentation. AppDynamics makes no representation as to the accuracy of Amazon documentation because Amazon controls its own documentation.

This page explains how to subscribe to Serverless APM for AWS Lambda. AppDynamics provides Serverless APM functionality in the form of a tracer library. See Serverless APM.

Requirements

Serverless APM for AWS Lambda supports AWS Lambda functions implemented in Java, Node.js, or Python. A subscription to Serverless APM requires:

- AppDynamics SaaS Controller >= 4.5.11.
- An active AppDynamics Pro account.
- Your Controller access key. This requires you to be an AppDynamics Account Owner or have Administrator privileges.
- AWS Identity and Access Management (IAM) role with permission to one of these policies:
  - **AWSMarketplaceManageSubscriptions**
  - **AWSMarketplaceFullAccess**

Subscribe to Serverless APM

Subscribe to AppDynamics Serverless APM for AWS Lambda through AWS Marketplace.

1. In the AWS Marketplace, navigate to the **AppDynamics Serverless APM for AWS Lambda** listing.
2. Click **Continue to Subscribe**.
3. Sign in to your AWS account.
4. Click **Subscribe**. A message, Congratulations! You are now subscribed!, appears confirming your subscription.
   If the Subscribe button is unavailable:
   a. Navigate to **Having issues signing up for your product?**
   b. Select **click here**.
   c. Proceed to step six.
5. Click **Set Up Your Account**.
6. Enter your SaaS Controller URL and Controller Access Key (Default), then click **Continue**.

   - The SaaS Controller URL must be a full path URL with no whitespace or trailing slashes.
   - Registration may take up to 30 seconds.
   - You can only access the registration page through AWS Marketplace.
   - Do not close the registration page while it attempts to connect to the Controller. If you close the registration page, you will have to subscribe again through AWS Marketplace.
7. Click **Getting Started** to learn more about Serverless APM for AWS Lambda.

Subscribe Multiple Controllers

You can subscribe multiple Controllers to Serverless APM, as long as each Controller meets the requirements described on this page. Begin with step 4a in Subscribe to Serverless APM above and complete the steps for each Controller.

Unsubscribe from Serverless APM for AWS Lambda

Unsubscribe from Serverless APM for AWS Lambda to stop all tracing by AppDynamics.

1. Sign in to the AWS Management Console.
2. Go to **Your Account** > **Your Software Subscriptions** > **SaaS**. Locate the **AppDynamics Serverless APM for AWS Lambda** subscription.
3. Select **Cancel Subscription**. **Cancel Subscription** unsubscribes all Controllers from Serverless APM.

You can disable Serverless APM at any time. When you disable Serverless APM, the tracer continues to monitor any instances currently executing until those instances finish. Requests executed by instances started after you disable the tracer are not monitored.

> Technical support is available through AppDynamics Support. All billing related questions are handled by AWS Marketplace.

View the Serverless APM Subscription

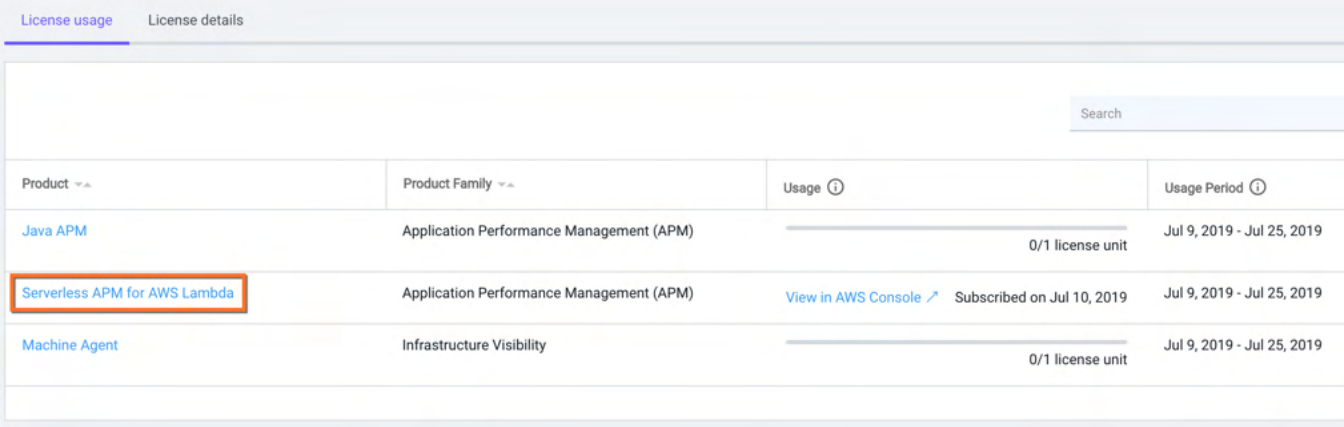
After you have subscribed to Serverless APM for AWS Lambda, you can view your subscription and its status through your AppDynamics account.

Verify Your Serverless APM Subscription

You can verify your subscription to Serverless APM for AWS Lambda in your AppDynamics account.

1. Go to https://accounts.appdynamics.com/subscriptions.
2. Sign in to your account.
3. Select the Controller you subscribed to Serverless APM.
4. Confirm that **Serverless APM for AWS Lambda** subscription is activated in the License usage tab.

Under the **Usage** column, you can view the date your subscription began, and access billing information on the AWS Console. Serverless APM for AWS Lambda matches the **Usage Period** of your APM license, regardless of the Serverless APM subscription status.

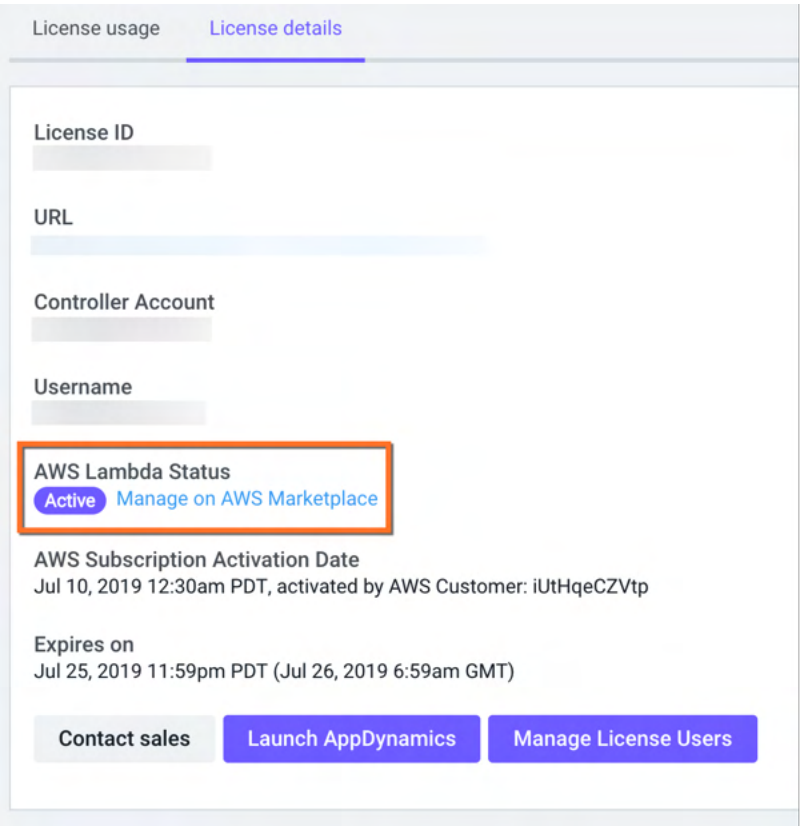


Check Your Serverless APM Subscription Status

You can check your Serverless APM for AWS Lambda subscription status in your AppDynamics account. To access your account see the instructions in Verify Your Serverless APM Subscription. Click the License details tab and navigate to **AWS Lambda Status**.

Active status indicates you are currently subscribed to Serverless APM for AWS Lambda. You can see the date your subscription began and the encrypted AWS Customer ID of the user who subscribed.

Inactive status indicates you have canceled your subscription to Serverless APM for AWS Lambda for all Controllers. You can see the cancellation date and the encrypted AWS Customer ID of the user who unsubscribed.



Troubleshoot the Serverless APM Subscription

> This document contains links to AWS documentation. AppDynamics makes no representation as to the accuracy of Amazon documentation because Amazon controls its own documentation.

This page provides troubleshooting information for issues that may arise when subscribing to Serverless APM for AWS Lambda.

Subscribe Button Unavailable

After an AWS user clicks **Subscribe** on the AWS product listing, the button is unavailable for future visits to the AWS Marketplace subscription page. To complete the subscription process, see step four in Subscribe to Serverless APM for AWS Lambda.

Error Messages

You may encounter error messages while subscribing through the AWS Marketplace.

Upgrade to Enable Serverless APM for AWS Lambda

This error message refers to your current Controller version and AppDynamics edition. To subscribe to serverless monitoring, you need SaaS Controller 4.5.11 and an active AppDynamics Pro edition.

- **AppDynamics SaaS Controller >= 4.5.11**

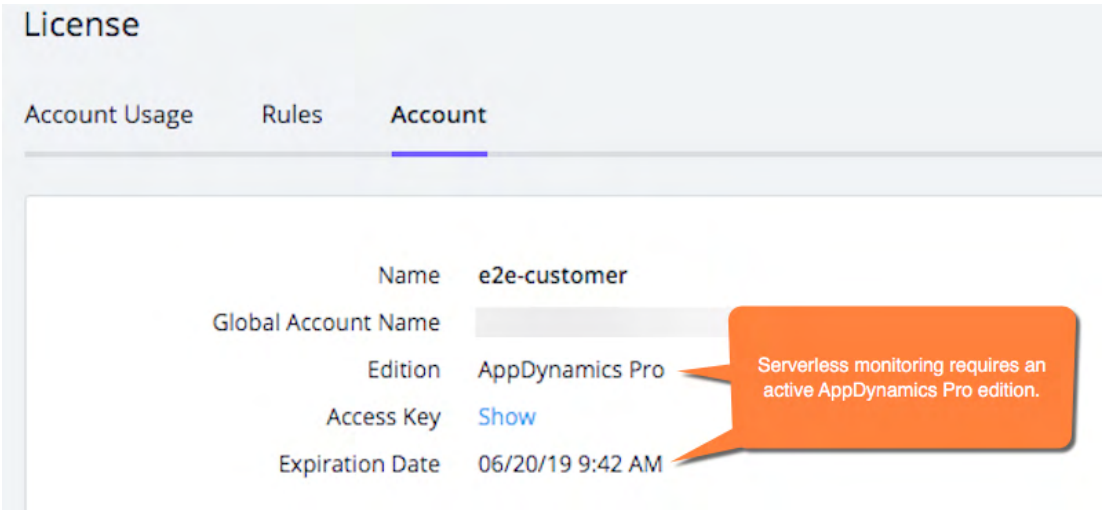
  Serverless APM for AWS Lambda is available for AppDynamics Pro SaaS Controllers. Your Controller must be >= 4.5.11. You can request an upgrade if you have a previous version of the Controller. To view your current AppDynamics build version in the Controller UI, select > **About AppDynamics**.

  Serverless monitoring is not available for on-premises Controllers at this time.

- **Active AppDynamics Pro Edition**

  You need an active AppDynamics Pro edition to enroll in serverless monitoring. If you are currently enrolled in a trial or unpaid edition, or if your account is inactive, purchase a license.

  To view your current AppDynamics edition and expiration date in the Controller UI, select > **License** > **Account**.



Incorrect Credentials

You need your SaaS tenant URL and the access key for the Controller you want to enable. You must be an AppDynamics Account Owner or Administrator to view the Controller access key. To find your Controller access key, see Agent-to-Controller Connections.

Controller Already Registered

You cannot register a single Controller for serverless monitoring multiple times. You can view your current Serverless APM for AWS Lambda subscriptions in your AppDynamics account.

1. Go to https://accounts.appdynamics.com/subscriptions.

2. Sign in to your account.
3. Select the desired Controller.
4. In the License Usage tab, find **Serverless APM for AWS Lambda**.

Technical support is available through AppDynamics Support. All billing related questions are handled by AWS Marketplace.

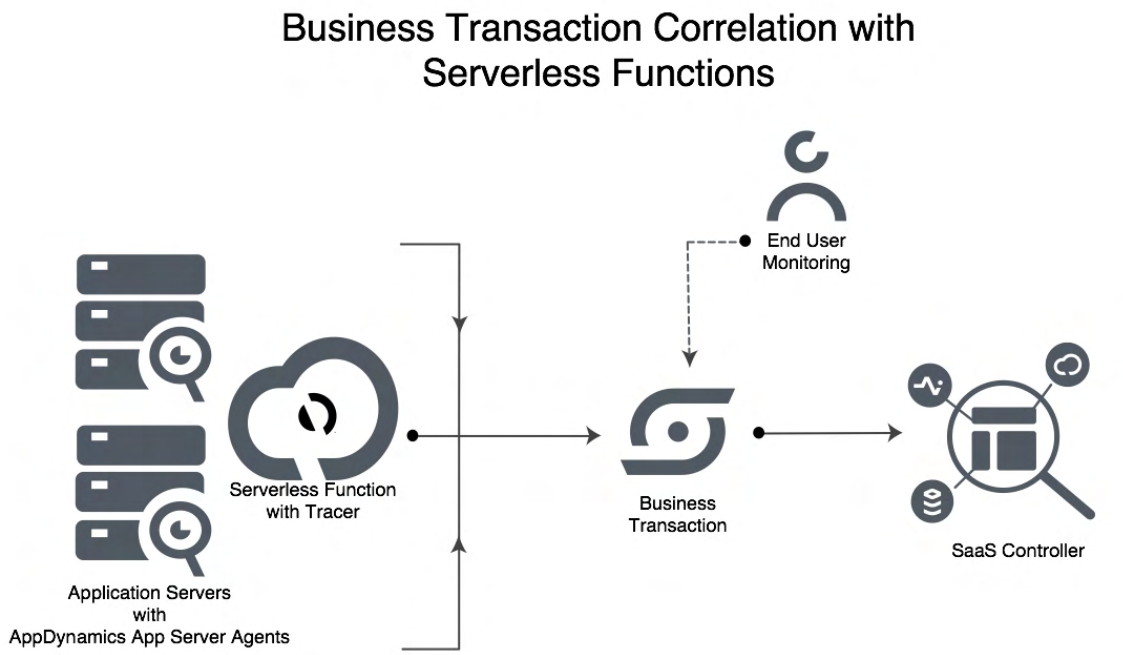
Serverless APM Business Transaction Correlation

AppDynamics provides Serverless Application Performance Monitoring (Serverless APM) functionality in the form of a serverless tracer, available in Java, Node.js, and Python. This page discusses how the tracer correlates business transactions.

Business Transaction Correlation Architecture

The Serverless Tracer correlates business transactions with upstream and downstream components. Business transactions can be started or continued within application components instrumented with APM agents or Lambda tracers providing end to end transaction visibility in AppDynamics.

This diagram illustrates how the tracer correlates business transactions through a serverless function to APM and EUM agents:



Business Transaction Correlation Process

Business transaction correlation occurs through an opaque correlation string that the application passes across the wire. First, the tracer generates the correlation string at an exit call, and then the application passes the correlation string to the downstream components. The downstream component retrieves the correlation string to continue the business transaction. This process enables the Serverless Tracer to correlate business transactions.

Correlation String Transportation

Your application needs to pass the correlation string, generated by the tracer, to correlate a business transaction between multiple services. The correlation string must be serialized and communicated alongside the application payload.

Inbound HTTP Calls

If you pass the correlation string through an HTTP call, the string is passed as an HTTP header. The tracer searches for the key that holds the correlation header string in your function's invocation object. Correlation occurs when the Serverless Tracer finds the correlation header key. If the key cannot be found, the tracer creates a new business transaction.

Other Protocols

For protocols other than HTTP, you need to define protocol-specific transportation of the correlation string to enable business transaction correlation.

The tracer creates a new business transaction if:

- You have not arranged for the correlation string to be passed, or
- The tracer cannot find the correlation string.

To get started, see Set Up the Serverless APM Environment.

Set Up the Serverless APM Environment

This page describes required and optional environment variables, including HTTP proxy support, and how to input the variables in the AWS Management Console.

Configure Environment Variables

To instrument Serverless APM for AWS Lambda, you must configure the environment variables listed in the table below. Enter all applicable information as key-value environment variables in the AWS Management Console.

> When you change an environment variable, any existing instances complete execution using the previous value. All new requests execute with the updated value.

Environment Variable	Description	Default	Required	Example Value
`APPDYNAMICS_ACCOUNT_NAME`	Account name associated with the Controller used by your AWS Lambda function.	`<AppDynamics_account_name>`	Yes	`customer1`
`APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY`	Access key for your Controller. See Agent-to-Controller Connections.	`<AppDynamics_account_key>`	Yes	`AB1a2b3c4$123`
`APPDYNAMICS_APPLICATION_NAME`	Name of the application where the tracer is instrumented.	`<Your_app_name>`	Yes	`testApp`
`APPDYNAMICS_CONTROLLER_HOST`	Host associated with the Controller used by your AWS Lambda function. Do not include `http://` or `https://`	`<Controller_host_name>`	Yes	`<accountname>.saas.appdynamics.com`
`APPDYNAMICS_CONTROLLER_PORT`	Port associated with your Controller.	`443`	No	8080
`APPDYNAMICS_DISABLE_AGENT`	Disables the Serverless Tracer and stops all serverless application monitoring. Accepted values are `true` and `false`.	`false`	No	`true`
`APPDYNAMICS_ENABLE_EUM`	Used by the Node.js Serverless Tracer to enable End User Monitoring for AWS Lamba functions. Set to `true` to enable EUM or `false` to disable EUM. See Integrate the Node.js Serverless Tracer with End User Monitoring for details.	`false`	No	`true`
`APPDYNAMICS_EUM_AJAX`	When set to `true` ensures EUM data is always returned in Ajax form. For example, `ADRUM_0, ADRUM_1`, and so on.	`false`	No	`true`
`APPDYNAMICS_HTTP_TIMEOUT_MS`	HTTP timeout in milliseconds for the tracer to send data downstream.	`2000`	No	3000
`APPDYNAMICS_LOG_LEVEL`	Log level for the tracer. Accepted values are `DEBUG`, `INFO`, `WARN`, `ERROR`, and `FATAL`.	`INFO`	No	`INFO`
`APPDYNAMICS_SERVERLESS_API_ENDPOINT`	AppDynamics endpoint that the tracer to reports to. Serverless API endpoints are available for the following AWS regions: Sydney, Frankfurt, and Oregon.	`<https://your-endpoint-name-api.saas.appdynamics.com/>`	Yes	• Asia Pacific (Sydney): `https://syd-sls-agent-api.saas.appdynamics.com` • EU (Frankfurt): `https://fra-sls-agent-api.saas.appdynamics.com` • US West (Oregon): `https://pdx-sls-agent-api.saas.appdynamics.com`
`APPDYNAMICS_TIER_NAME`	The name of your AWS Lambda function.	`<Your_tier_name>` Defaults to the name of your AWS Lambda function.	No	`serverlessTestTier`

HTTP Proxy Support

To enable HTTP Proxy Support, define the following variables in the AWS Management Console:

Environment Variable	Description	Example Value
APPDYNAMICS_HTTP_PROXY_HOST	Publicly-accessible hostname of the proxy.	myproxy.example.com
APPDYNAMICS_HTTP_PROXY_PORT	Port on which the proxy is running.	8080
APPDYNAMICS_HTTP_PROXY_SERVER_CERTIFICATE	Relative path to the proxy server's certificate. Required if your proxy server runs on a self-signed certificate for your AWS Lambda function to trust the proxy. If you have configured your runtime environment to trust your proxy server, you do not need this variable.	resources/proxy-cert.pem

Additionally, you can add credentials for your proxy server's basic authentication. Add the following variables in the AWS Management Console:

Environment Variable	Description	Example Value
APPDYNAMICS_HTTP_PROXY_USER	Username associated with your basic authentication.	user1
APPDYNAMICS_HTTP_PROXY_PASSWORD	Relative path to the proxy server's basic authentication password. Required for your proxy server's password to trust your AWS Lambda function.	password123
APPDYNAMICS_HTTP_PROXY_PASSWORD_FILE	Plaintext file to the proxy server's basic authentication password file. Update your deployment package to contain the proxy server's password file. Required for your proxy server's password to trust your AWS Lambda function.	resources/proxy-password.txt

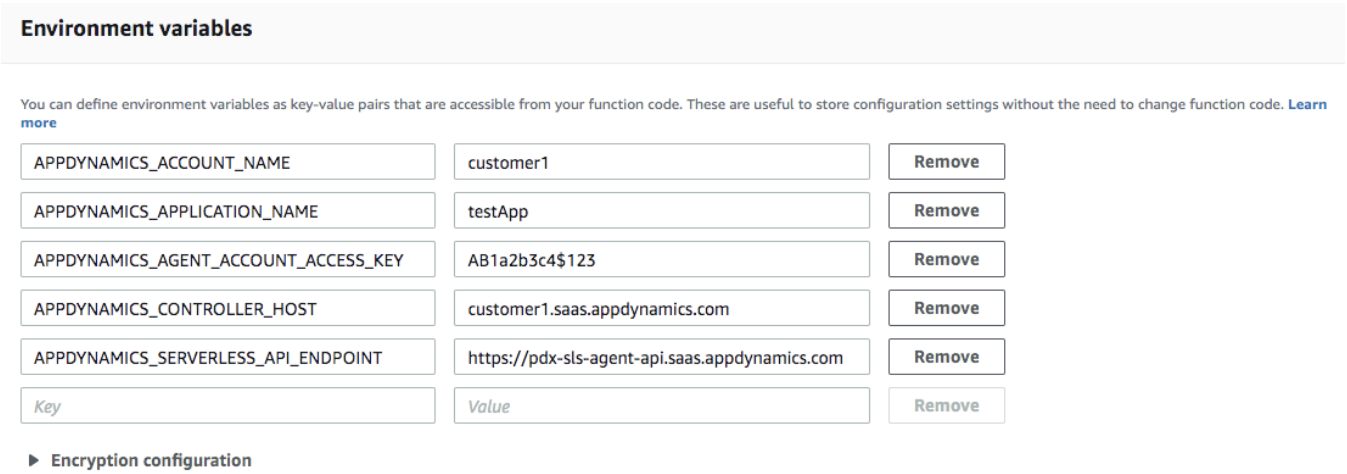
When you add basic credentials, you are required to include either APPDYNAMICS_HTTP_PROXY_PASSWORD or APPDYNAMICS_HTTP_PROXY_PASSWORD_FILE. If you include both, the tracer only passes APPDYNAMICS_HTTP_PROXY_PASSWORD_FILE.

Add Environment Variables in the AWS Console

To add environmental variables through the AWS Management Console:

1. Sign in to the AWS Management Console.
2. Go to **AWS Services** and open **Lambda**.
3. Select your AWS Lambda function.
4. Go to **Environment variables** and enter your function's key-value pair environment variables.
5. Click **Save**.

The following image shows an example of all required environment variables:



Java Serverless Tracer

This page describes requirements, download instructions, and an instrumentation overview for the Java Serverless Tracer.

Before You Begin

Ensure that your setup meets these requirements:

- Existing AWS Lambda functions implemented in Java
- Active Serverless APM for AWS Lambda subscription
- AppDynamics SaaS Controller >= 4.5.11

Instrumentation Overview

Instrumentation of Serverless APM for AWS Lambda consists of three steps:

1. Declare a Dependency on the Java Serverless Tracer
2. Choose an Instrumentation Method
    a. Automatic Tracer Instrumentation
    b. Manual Tracer Instrumentation
3. Verify the Serverless Tracer instrumentation

Declare a Dependency on the Java Serverless Tracer

AppDynamics provides Serverless APM functionality in the form of a tracer library.

Express your project's dependency on the Java Serverless Tracer using Apache Maven or Gradle as follows:

If you do not use Apache Maven or Gradle, contact AppDynamics to obtain the Serverless Tracer JAR file.

Instrumentation Options

You can instrument the tracer:

- Automatically, using AppDynamics class `MonitoredRequestStreamHandler`, or
- Manually, using AppDynamics method `AppDynamics.getTracer(context)`.

Automatic Tracer Instrumentation

You can automatically instrument the tracer if your function uses the `RequestStreamHandler` interface.

In automatic instrumentation, your function inherits the tracer's configurations. Automatic instrumentation preconfigures the tracer to:

- Create, start, and stop a transaction
- Locate any correlation header
- Report transaction errors

`RequestStreamHandler` and `RequestHandler` are predefined handlers provided by AWS. See AWS documentation.

Manual Tracer Instrumentation

In manual tracer instrumentation, the method `AppDynamics.getTracer(context)` instantiates the tracer at the beginning of your function's entry point method. Manually instrument the tracer if:

- Your AWS Lambda function does not implement the `RequestStreamHandler` interface.
- You do not want your function to inherit configurations from the `MonitoredRequestStreamHandler`.
- Your AWS Lambda functions are multi-threaded, to ensure that behavior is defined. See **Synchronize the Tracer** for more details.

Manual tracer instrumentation requires you to perform these actions:

- Instantiate the tracer
- Create, start, and stop a transaction
- Locate a correlation header
- Report transaction errors

The AWS Lambda context object in Java, `context,` is provided by AWS. See AWS documentation.

Synchronize the Tracer

For both instrumentation options, you should synchronize any access to the tracer.

In automatic instrumentation, you need to synchronize the transaction object returned by the `getTransaction()` method.

In manual instrumentation, you need to synchronize the tracer object returned by the `getTracer()` method. Manual tracer instrumentation is required for multi-threaded use of the tracer.

Automatic Tracer Instrumentation

Automatic tracer instrumentation consists of these steps:

1. Instantiate the Tracer
2. (Optional) Override Correlation Header Method
3. Obtain a Transaction Object
4. Create External Exit Calls

Instantiate the Java Serverless Tracer

> ⚠ Automatic instantiation is compatible with `RequestStreamHandler` interfaces. All other interfaces require manual instantiation.
>
> `RequestStreamHandler` is a predefined handler provided by AWS. See AWS documentation for more information.

To automatically instantiate the tracer:

1. Make your current handler class extend AppDynamics `MonitoredRequestStreamHandler`.
2. Rename your handler class's existing method `handleMonitoredRequest`.

The code snippet demonstrates how to instantiate the tracer automatically:

```
package <MyTestPackage>;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

//AppDynamics tracer imports
import com.appdynamics.serverless.tracers.aws.api.AppDynamics;
import com.appdynamics.serverless.tracers.aws.api.MonitoredRequestStreamHandler;
import com.appdynamics.serverless.tracers.aws.api.ExitCall;
import com.appdynamics.serverless.tracers.aws.api.Tracer;
import com.appdynamics.serverless.tracers.aws.api.Transaction;

//Keyword extends makes MonitoredRequestStreamHanlder your project's superclass
public class <MaryHadALittleLambda> extends MonitoredRequestStreamHandler {

   @Override
        //Change your method to handleMonitoredRequest
    public void handleMonitoredRequest(InputStream input, OutputStream output, Context context) throws
IOException {
           }

       // Your AWS Lambda function code starts here, for example
               int letter = 0;
               while((letter = input.read()) >= 0) {
                       output.write(Character.toUpperCase(letter));
       }
    }
```

(Optional) Override Correlation Header Method

Automatic instrumentation uses the `MonitoredRequestStreamHandler` class, which provides default logic to find a correlation header.

If you need to use a custom transport mechanism for your correlation header, you must override the tracer's default logic in the `getCorrelationHeader()` method. Insert your custom logic to find the inbound correlation header so the tracer can return the inbound header.

The code snippet demonstrates how to override the `getCorrelationHeader()` method:

```
//(Optional)Override getCorrelationHeader().
//Only override if the tracer will not be able to find a correlation header using its pre-configured logic.

@Override
public String getCorrelationHeader(InputStream input, Context context) {
//Logic to find the inbound header goes here, so the inbound header can be returned
return "correlation-header";
```

Obtain a Transaction Object

The `MonitoredRequestStreamHandler` class creates a transaction for you automatically. However, you need to obtain the `transaction` object for use with exit calls.

The code snippet below demonstrates how to obtain the `transaction` object:

```
//Obtain your transaction for use with exit calls.
Transaction transaction = getTransaction();
```

Create External Exit Calls

The `createExitCall` method is used to obtain an `exitCall` object. Exit call objects record the time spent in external exit calls and allow for correlation of downstream activity.

Exit Call Types and Identifying Properties

In the Controller, each exit call has a distinct type, determined by a set of identifying properties.

The `exitCall` types and identifying properties are listed below:

`exitCall` Type	**Identifying Properties**
HTTP	HOST PORT URL QUERY STRING
JDBC	URL HOST PORT DATABASE VERSION VENDOR
WEB SERVICE	SERVICE URL OPERATION SOAP ACTION VENDOR
CUSTOM	Any user-defined set of properties

Create an Exit Call

Call `createExitCall()` on your `transaction` object.

The following code sample demonstrates how you might perform an external exit call:

```
public void makeExitCall(URL url){

HttpURLConnection conn = (HttpURLConnection)url.openConnection();

String outgoingHeader = null;
String callType = null;
Map<String, String> identifyingProperties = new HashMap<>();

//Below properties are appropriate for an inter-AWS Lambda call
identifyingProperties.put("DESTINATION", functionName);
identifyingProperties.put("DESTINATION_TYPE", "LAMBDA");
callType="CUSTOM";

//Below properties are appropriate for an external HTTP call
identifyingProperties.put("HOST", url.getHost());
identifyingProperties.put("PORT", String.valueOf(url.getPort()));
callType="HTTP";

//Define the createExitCall method to obtain an exitCall object.
ExitCall exitCall = transaction.createExitCall(callType, identifyingProperties);
outgoingHeader =  exitCall.getCorrelationHeader();
exitCall.start();

try {
// Tracer.APPDYNAMICS_TRANSACTION_CORRELATION_HEADER_KEY is the name of the header that should be set

    if (outgoingHeader != null) {
        conn.setRequestProperty(Tracer.APPDYNAMICS_TRANSACTION_CORRELATION_HEADER_KEY, outgoingHeader); // set
the correlation header on an HttpURLConnection
    }

// Make the exit call here

} finally {
        exitCall.stop();
    }
}
```

Making Multiple External Exit Calls

If one AWS Lambda function makes multiple exit calls, each function should be identified by a unique `exitCall` object. To make multiple exit calls, create new `exitCall` objects for each distinct exit call. The `exitCall` objects are not reusable.

Manual Tracer Instrumentation

Manual tracer instrumentation consists of these steps:

1. Instantiate the Tracer
2. Create a Transaction
3. Start and Stop a Transaction
4. Create External Exit Calls

Instantiate the Java Serverless Tracer

To instantiate the tracer, call the `AppDynamics.getTracer(context)` method. Place the method at the beginning of your function's entry point method.

The code snippet demonstrates how to instantiate the tracer:

```
@Override
    public void handleRequest(InputStream input, OutputStream output, Context context) throws IOException {

                //Instantiate the tracer.
                  Tracer tracer = AppDynamics.getTracer(context);

                //Your AWS Lambda function code begins here.
                int letter = 0;
            while((letter = input.read()) >= 0) {
                output.write(Character.toUpperCase(letter));
                }
    }
```

Create a Transaction

Next, you need to create transactions. A transaction is a monitored segment in a larger business transaction. Creating a transaction involves locating the correlation header, which the tracer uses to pass contextual information downstream.

Transactions either:

- Start a new business transaction, or
- Continue an existing business transaction

Start a New Business Transaction

A transaction that originates in the current function does not have an inbound correlation header. To create a business transaction, use the `createTransaction()` method and provide an empty correlation header.

Continue an Existing Business Transaction

To continue an existing business transaction, you need to locate an inbound correlation header. You can locate a correlation header automatically or manually.

If the tracer cannot find the correlation header, the tracer creates a new business transaction.

Continue a Business Transaction by Locating the Correlation Header Automatically

Use the `createTransaction(inputStream, context)` method to create a transaction that locates the correlation header automatically. This method searches your code's object schema for a key called `singularityheader` in the `inputStream` object. The correlation header finds the `singularityheader` key, then continues the business transaction.

> The tracer reads the `inputStream` object, which is configured to be read once. If your application needs to read the `inputStream` object, you must use a converter method to allow both the tracer and your application to read the stream.

The code snippet shows an example of how to create a transaction and automatically locate the correlation header:

```
public class <LambdaChops> implements RequestStreamHandler {
    @Override
    public void handleRequest(InputStream input, OutputStream output, Context context) throws IOException {
          Tracer tracer = AppDynamics.getTracer(context);

        //Use a converter method if you need to read your inputStream more than once. The tracer reads your
inputStream once.
          inputStream = InputStreamConverter.convertToMarkSupportedInputStream(input);

        //Create a transaction.
        Transaction transaction = tracer.createTransaction(input, context);

        //Your AWS Lambda function code begins here.
        int letter = 0;
        while((letter = input.read()) >= 0) {
            output.write(Character.toUpperCase(letter));
        }
    }
}
```

Create a Transaction by Locating the Correlation Header Manually

To create a transaction that locates the correlation header using custom logic, you need to manually parse a `correlationHeader` string. Next, call the c`reateTransaction()`method and provide the `correlationHeader` object. This process gives you full control over the parsing of your function's input payload.

The code snippet shows an example of how to obtain the correlation header string and create a transaction:

```
public class <LambdaChops> implements RequestHandler {
    @Override
        public O handleRequest(InputStream input, OutputStream output, Context context) {
        Tracer tracer = AppDynamics.getTracer(context);

        //Manually parse the correlation string.
     String inputAsString = IOUtils.toString(inputStream, Charset.forName("UTF-8"));
     JsonParser parser = new JsonParser();
     JsonObject inputObject = parser.parse(inputAsString).getAsJsonObject();

        String correlationHeader = "";
        if (inputObject.has(Tracer.APPDYNAMICS_TRANSACTION_CORRELATION_HEADER_KEY)) {
            correlationHeader = inputObject.get(Tracer.APPDYNAMICS_TRANSACTION_CORRELATION_HEADER_KEY).
getAsString();
        } else {
            // Try reading from HTTP headers
            if (inputObject.has("headers")) {
                JsonObject httpHeaders = inputObject.getAsJsonObject("headers");
                if (httpHeaders.has(Tracer.APPDYNAMICS_TRANSACTION_CORRELATION_HEADER_KEY)) {
                    correlationHeader = httpHeaders.get(Tracer.APPDYNAMICS_TRANSACTION_CORRELATION_HEADER_KEY).
getAsString();
        }
    }
}

        // Create transaction object using the correlation header. If the correlationHeader string is empty,
the transaction you create uses the default transaction name.
        Transaction transaction = tracer.createTransaction(correlationHeader);

        //Your AWS Lambda function code starts here
        ...

}
```

Start and Stop a Transaction

After creating a transaction, you need to set transaction boundaries. Use the `transaction.start()` and `transaction.stop()` methods to place boundaries around any section of code that you want the tracer to monitor.

As a best practice, you can report all events, even if your function crashes, by running your entire function within a `try` block, and then stopping the transaction within a `finally` block.

This code sample demonstrates how to start and stop a transaction that monitors an entire function:

```
@Override
 public void handleRequest(InputStream input, OutputStream output, Context context) throws IOException {
  Tracer tracer = AppDynamics.getTracer(context);
  Transaction transaction = tracer.createTransaction(input, context);

        //Start the transaction monitoring.
          transaction.start();
          try {
                   int letter = 0;
        while((letter = input.read()) >= 0) {
            output.write(Character.toUpperCase(letter));
                }

        //Stop the transaction monitoring. Place in a finally block to monitor all events.
        } finally {
                   transaction.stop();
                 AppDynamics.cleanup();
          }
 }
```

Create External Exit Calls

The `createExitCall` method is used to obtain an `exitCall` object. Exit call objects record the time spent in external exit calls and allow for correlation of downstream activity.

Exit Call Types and Identifying Properties

In the Controller, each exit call has a distinct type, determined by a set of identifying properties.

This table lists the `exitCall` types and identifying properties:

`exitCall` Type	**Identifying Properties**
HTTP	HOST PORT URL QUERY STRING
JDBC	URL HOST PORT DATABASE VERSION VENDOR
WEB SERVICE	SERVICE URL OPERATION SOAP ACTION VENDOR
CUSTOM	Any user-defined set of properties

Create an External Exit Call

Call `createExitCall()` on your transaction object.

The following code sample demonstrates how you might perform an external exit call:

```
public void makeExitCall(URL url){

HttpURLConnection conn = (HttpURLConnection)url.openConnection();

String outgoingHeader = null;
String callType = null;
Map<String, String> identifyingProperties = new HashMap<>();

//Below properties are appropriate for an inter-AWS Lambda call
identifyingProperties.put("DESTINATION", functionName);
identifyingProperties.put("DESTINATION_TYPE", "LAMBDA");
callType="CUSTOM";

//Below properties are appropriate for an external HTTP call
identifyingProperties.put("HOST", url.getHost());
identifyingProperties.put("PORT", String.valueOf(url.getPort()));
callType="HTTP";

//Define the createExitCall method to obtain an exitCall object.
ExitCall exitCall = transaction.createExitCall(callType, identifyingProperties);
outgoingHeader =  exitCall.getCorrelationHeader();
exitCall.start();

try {
// Tracer.APPDYNAMICS_TRANSACTION_CORRELATION_HEADER_KEY is the name of the header that should be set

    if (outgoingHeader != null) {
        conn.setRequestProperty(Tracer.APPDYNAMICS_TRANSACTION_CORRELATION_HEADER_KEY, outgoingHeader); // set
the correlation header on an HttpURLConnection
    }

// Make the exit call here

} finally {
        exitCall.stop();
    }
}
```

Create Multiple External Exit Calls

If one AWS Lambda function makes multiple exit calls, each function should be identified by a unique `exitCall` object. To make multiple exit calls, create new `exitCall` objects for each distinct exit call. The `exitCall` objects are not reusable.

Java Serverless Tracer API

This page describes how to use the Java Serverless Tracer API to fine-tune your configuration.

Transaction Error Reports

If you automatically instrument the tracer, transaction errors are reported if an exception is thrown. If you manually instrument the tracer, transaction error reports are optional and require manual configuration.

If your business logic defines the transaction as in error for any reason, you can customize the error report method to mark the transaction in error.

To create transaction error reports, use `transaction.reportError()`.

The `transaction.reportError()` method reports one error per invocation. You can also use `transaction.reportError()` for any `errorName` or `errorMessage`:

```
transaction.reportError(errorName, errorMessage);
```

Exit Call Error Reports

You can report errors and exceptions that occur during exit call execution using the `exitCall.reportError()` method. You must call the method before `transaction.stop()`.

Use `exitCall.reportError()` for any exception or subclass:

```
exitCall.reportError(exception);
```

You can also use `exitCall.reportError()` for a specific `errorName` or `errorMessage`.

```
exitCall.reportError(errorName, errorMessage);
```

Override the Tracer's Default Behavior

If you prefer to customize the function and business transaction names, or keep the tracer settings in your code, you can override the environment variables using the `AppDynamics.Config.Builder` object. This option allows you to retrieve values at runtime from any desired source.

```
AppDynamics.Config.Builder configBuilder = new AppDynamics.Config.Builder();
configBuilder.accountName(accountName)
   .applicationName(appName)
   .tierName(tierName)
   .controllerHost(controllerHost)
   .controllerPort(controllerPort)
   .defaultBtName(context.getFunctionName() + "_bt")
   .controllerAccessKey(controllerAccessKey)
   .lambdaContext(context);
```

Implementation of the `configBuilder` object differs slightly depending on which instrumentation method you choose, automatic or manual.

Automatic Instrumentation

Automatic instrumentation uses the `MonitoredRequestStreamHandler` class, which implements a factory method, `getConfigBuilder()`, to instrument the tracer.

You can override this factory method and use any builder method to configure environment variables.

The code snippet shows how to enter variables in your code for automatic instrumentation:

```
@Override
public AppDynamics.Config.Builder getConfigBuilder(Context context) {
String controllerHost = System.getenv("APPDYNAMICS_CONTROLLER_HOST");
int controllerPort = Integer.parseInt(System.getenv("APPDYNAMICS_CONTROLLER_PORT"));
String accountName = System.getenv("APPDYNAMICS_ACCOUNT_NAME");
String appName = System.getenv("APPDYNAMICS_APPLICATION_NAME");
String tierName = System.getenv("APPDYNAMICS_TIER_NAME");
String controllerAccessKey = System.getenv("APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY");

AppDynamics.Config.Builder configBuilder = new AppDynamics.Config.Builder();
configBuilder.accountName(accountName)
   .applicationName(appName)
   .tierName(tierName)
   .controllerHost(controllerHost)
   .controllerPort(controllerPort)
   .defaultBtName(context.getFunctionName() + "_bt")
   .controllerAccessKey(controllerAccessKey)
   .lambdaContext(context);
return configBuilder;
  }
```

Manual Instrumentation

In manual instrumentation, the overloaded version of the `AppDynamics.getTracer()` method configures the tracer's settings. The overloaded method takes a `configBuilder` object as a parameter, instead of the `context` object, to instrument the tracer.

When you use this variant of `AppDynamics.getTracer()`, you must pass the `context` object via the `AppDynamics.Config.Builder.lambdaContext()` method.

The code snippet shows how to enter variables in your code for manual instrumentation.

```
@Override
public void handleRequest(InputStream input, OutputStream output, Context context) throws IOException) {
        String controllerHost = System.getenv("APPDYNAMICS_CONTROLLER_HOST");
        int controllerPort = Integer.parseInt(System.getenv("APPDYNAMICS_CONTROLLER_PORT"));
        String accountName = System.getenv("APPDYNAMICS_ACCOUNT_NAME");
        String appName = System.getenv("APPDYNAMICS_APPLICATION_NAME");
        String tierName = System.getenv("APPDYNAMICS_TIER_NAME");
        String controllerAccessKey = System.getenv("APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY");
        AppDynamics.Config.Builder configBuilder = new AppDynamics.Config.Builder();
        configBuilder.accountName(accountName)
                .applicationName(appName)
                .tierName(tierName)
                .controllerHost(controllerHost)
                .controllerPort(controllerPort)
                .defaultBtName(context.getFunctionName() + "_bt")
                .controllerAccessKey(controllerAccessKey)
                .lambdaContext(context);

        //Replace the context object with config.Builder.build()
        Tracer tracer = AppDynamics.getTracer(configBuilder.build());
```

Integrate the Java Serverless Tracer with End User Monitoring

> This document contains links to AWS documentation. AppDynamics makes no representation as to the accuracy of Amazon documentation since Amazon controls its own documentation.

Serverless APM for AWS Lambda is designed to integrate with your existing End User Monitoring (EUM) configurations. The EUM integration provides complete end-to-end visibility on the performance of your web and mobile applications, linking calls from an end-user device through your serverless functions to continue your business transactions

AWS Lambda functions can correlate EUM and AWS Lambda-originated business transactions, in conjunction with the following EUM agents:

- Browser Real User Monitoring (Browser RUM)
- Mobile Real User Monitoring (Mobile RUM)
- IoT Monitoring

Integration of the Java Serverless Tracer with End User Monitoring consists of these steps:

1. Add and Return EUM Metadata
2. Configure Response Header Mappings
    a. Use the Amazon API Gateway
    b. Use AWS Lambda Proxy Integration
3. Update the Front End Application
4. Enable CORS configuration (If applicable)
5. Troubleshoot the EUM Integration

> This document assumes you are familiar with AppDynamics End User Monitoring. See End User Monitoring for more information.

Before You Begin

Ensure your deployment has:

- An AWS Lambda function instrumented with the Java Serverless Tracer
- Active EUM licenses and JavaScript agents

Add and Return EUM Metadata

To integrate EUM with the tracer, you need to query the `transaction` object and return EUM metadata.

Call the method below after stopping a transaction. Assign each string separately, in the order specified, to the headers ADRUM_0, ADRUM_1, ADRUM_2, and ADRUM_3 for consumption by the downstream JavaScript Agent.

```
public void stopTransactionAndRecordEUM(OutputStream output) throws IOException {
        Transaction transaction = getTransaction();

// Your AWS Lambda function code

//You must call transaction.stop(); before calling transaction.getEumMetadata();
 transaction.stop();
// call transaction.getEumMetadata(); function to query transaction object for EUM metadata
List<String> eumMetadata = transaction.getEumMetadata();
```

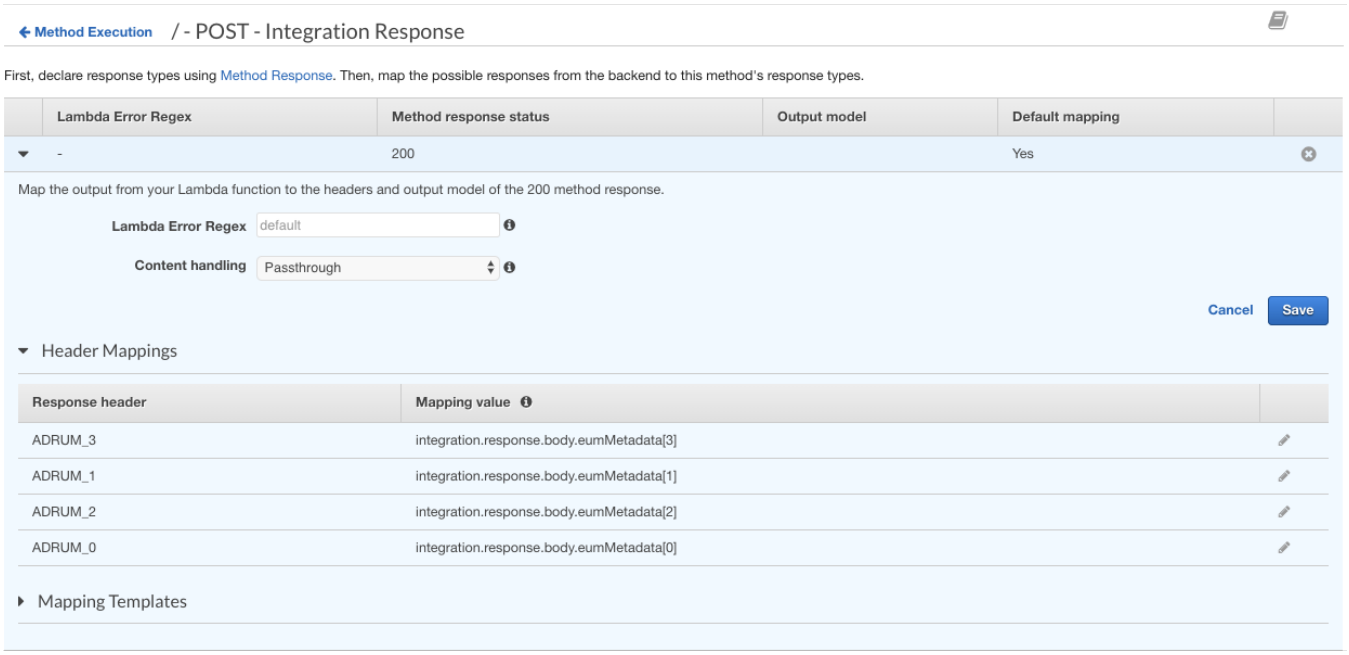
Configure Response Header Mappings

Configure response header mappings to pass ADRUM_*n* headers back to the browser. You can configure response header mappings using the AWS API Gateway or pass them directly using AWS Lambda Proxy Integration.

Response Header Mappings in Amazon API Gateway

Add header mappings for each ADRUM_*n* header in the Amazon API Gateway. To configure response header mappings in the Amazon API Gateway, refer to the Amazon documentation.

The screenshot illustrates how to map these headers in the AWS API Gateway:



Next, return the ADRUM_*n* headers as part of the AWS Lambda function output. You can use another field name, as long your function code is consistent with the response header mappings.

This code snippet shows how to add the headers field to an output object class:

```
// add the field eumMetadata to your output object class
// return the EUM ADRUM_n headers in the output object
String outputString = "success";
Output outputObj = new Output(outputString);
OutputObj.eumMetadata = eumMetadata;
output.write(new Gson().toJson(outputObj).getBytes(StandardCharsets.UTF_8));
```

Response Header Mappings in AWS Lambda Proxy Integration

As an alternative to custom integration, AWS Lambda proxy integration allows the client to call each function in the backend and returns output in a JSON format. Map these responses by adding each single-value ADRUM_*n* header directly to the headers field in your output object class. Refer to the Amazon documentation for details.

Update the Front End Application

To see the correlation between EUM and AWS Lambda-originated business transactions, you must inject the JavaScript Agent into your browser's HTML, as described in the EUM documentation.

Enable CORS Configuration

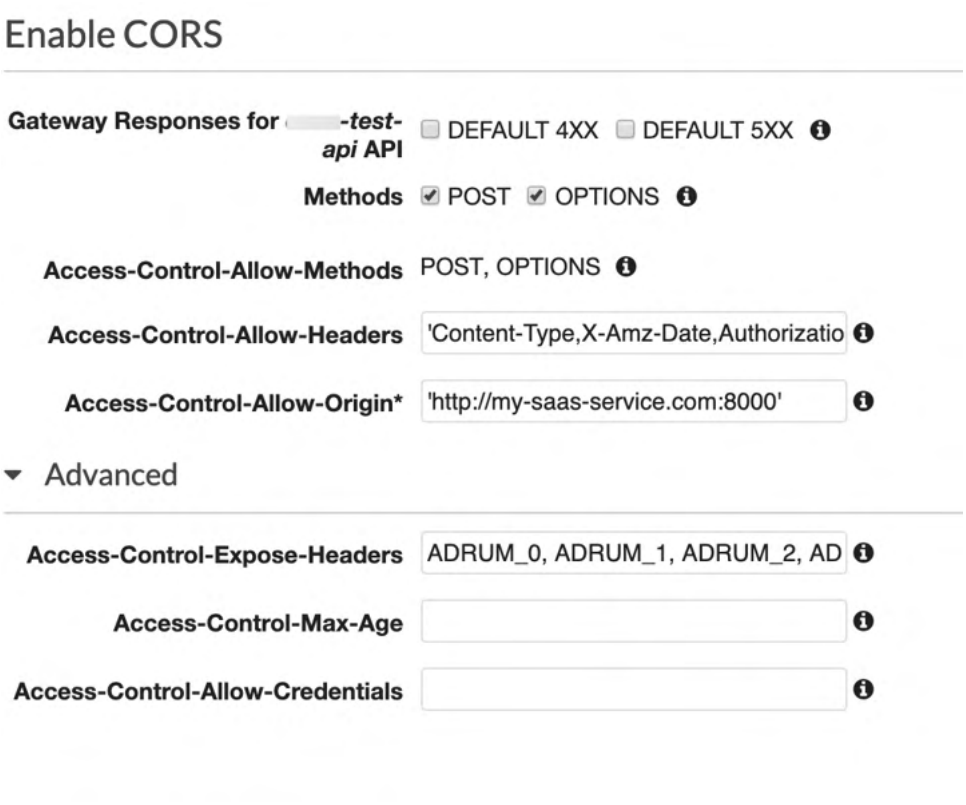
Browsers require cross-origin resource sharing (CORS) for functions that access responses from browser requests to a domain other than the base page's domain. You must explicitly grant cross-domain access to all applicable ADRUM_*n* headers.

To enable CORS configuration, define the required AWS response headers in the Amazon API Gateway:

- **Access-Control-Expose-Headers** - maps the custom ADRUM headers. These headers must have the names ADRUM_0 through ADRUM_3 to be compatible with the JavaScript Agent.
- **Access-Control-Allow-Methods** - check the header(s) with the appropriate method.
- **Access-Control-Allow-Origin** - set to the full URL of the web page that originated the request, including schema, hostname, and port.

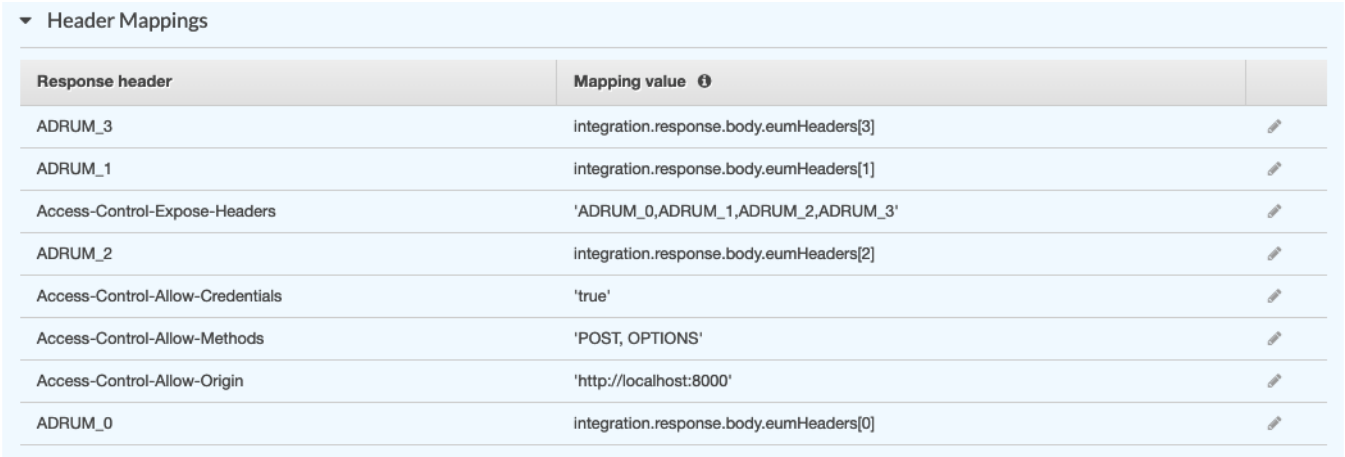
See AWS documentation for advanced headers and additional details.

The screenshot illustrates how to enable CORS in the Amazon API Gateway:



After you have enabled CORS, you must define the response header mappings. To configure response header mappings in the Amazon API Gateway, refer to the Amazon documentation.

The screenshot illustrates response header mappings after CORS configuration:



Troubleshoot the EUM Integration

If you do not see a correlation between EUM data and business transactions, collect an HTTP Archive (HAR) file that captures the activity in your browser. Use the HAR file to confirm all necessary CORS headers are set and expected data populates in the ARUM_*n* headers.

The EUM documentation provides additional troubleshooting advice.

Use the AppDynamics AWS Lambda Extension to Instrument Serverless APM at Runtime

This page explains how to use the AppDynamics AWS Lambda Extension for Serverless APM (the AppDynamics Lambda extension), the recommended way to inject the Serverless Tracer at runtime. Use this option to dynamically add a layer to a Node.js or Python application.

This document is intended to provide information specifically for using the AppDynamics AWS Lambda Extension for Serverless APM in conjunction with AppDynamics. AWS Lambda is managed by AWS and is a separate product from AppDynamics solutions.

- For questions related to AWS Lambda, see AWS Lambda Documentation.
- For questions related to AppDynamics AWS Lambda Extension for Serverless APM, see Serverless APM for AWS Lambda.

Before You Begin

After you have subscribed to Serverless APM for AWS Lambda and set up your environment variables, you can begin instrumentation. Ensure that your setup meets the following requirements:

- Existing AWS Lambda functions built with Node.js version 10.x or later or Python 3.6 or later
- Active Serverless APM for AWS Lambda subscription
- AppDynamics SaaS Controller version 4.5.11 or later
- Supported AWS Regions: eu-central-1, eu-north-1, eu-west-1, eu-west-2, eu-west-3, ap-northeast-1, ap-northeast-2, ap-south-1, ap-southeast-1, ap-southeast-2, ca-central-1, sa-east-1, us-east-1, us-east-2, us-west-1, us-west-2

Use AWS Lambda Layers to Inject the Serverless Tracer at Runtime

To install the Node.js or Python Serverless Tracer, configure your Lambda function to use the layer implementing the AppDynamics AWS Lambda Extension for Serverless APM. The method for doing this differs, depending on whether your Lambda function is packaged as a zip archive or as a container image.

Add the AWS Lambda Extension to Your Function when packaged as a zip archive

To pull the AppDynamics layer from Amazon, add the Amazon Resource Number (ARN) to your Lambda using either the AWS Management Console UI or the AWS CLI.

1. Add the AWS Lambda Extension to Your Function
2. Configure Environment Variables

Instrument a Lambda Function using the AWS UI

1. In the AWS Management Console console, select **Compute > Lambda**.
2. From the **AWS Lambda** panel, select a lambda that you want to instrument.
3. Select **Layers**.
4. Click **Add a layer.**
5. Under **Choose a layer**, select **Specify an ARN.**
6. Under **Specify an ARN,** enter the AWS Layer information:

```
arn:aws:lambda:us-west-2:338050622354:layer:appdynamics-lambda-extension:10
```

The ARN region changes based on the AWS region where the lambda is deployed. For example, if your lambda is **Europe (Frankfurt) eu-central-1**, use:

```
arn:aws:lambda:eu-central-1:338050622354:layer:appdynamics-lambda-extension:10
```

7. Click **Add.**
8. To verify the layer is added to the lambda, click **Layers**. Under **Layers**, the **appdynamics-lambda-extension** displays.
9. See Configure the Environment Variables.

Instrument a Lambda Function using AWS CLI

To pull the AppDynamics layer from Amazon, using the AWS CLI:

1. See Configuring a function to use layers.
2. See Configure the Environment Variables.

Configure Environment Variables

To finish injecting the tracer into AWS Lambda, configure environment variables depending on your programming language and version:

Node 10.x, Node 12.x, Node 14.x, and Python 3.8

Add the environment variable `AWS_LAMBDA_EXEC_WRAPPER` with the value: `/opt/appdynamics-extension-script:`

```
AWS_LAMBDA_EXEC_WRAPPER = /opt/appdynamics-extension-script
```

For Python 3.6 and Python 3.7

Add an environment variable `APPDYNAMICS_PYTHON_AUTOINSTRUMENT` with the value set to `true:`

```
APPDYNAMICS_PYTHON_AUTOINSTRUMENT = true
```

Add the AWS Lambda Extension to Your Function when packaged as a container image

When your Lambda functions are deployed as container images, Lambda layers cannot be added via the function configuration. Instead, you must package the extension as part of your container image during the build process. For more details refer to this AWS Blog article.

To add the AppDynamics Lambda extension to your Lambda container, you will need to use a Dockerfile like those provided below:

To build the container image containing your code as well as the AppDynamics Lambda extension, make sure you have the correct Dockerfile in your current directory and use the docker build command; for example:

```
docker build --build-arg version_number=$version_number --build-arg access_key=$(aws configure get
aws_access_key_id) --build-arg secret_key=$(aws configure get aws_secret_access_key) --build-arg region=$(aws
configure get region) -t <your tag> --build-arg cachebust=$(date +%s) .
```

Verify your Serverless Tracer Instrumentation

See Verify the Serverless Tracer Instrumentation.

Node.js Serverless Tracer

This page describes how to use the Node.js Serverless Tracer with AWS Lambda functions during development. The preferred method is to use the AppDynamics AWS Lambda Extension. See Use the AppDynamics AWS Lambda Extension to Instrument Serverless APM at Runtime.

Before You Begin

After you have subscribed to Serverless APM for AWS Lambda and set up your environment variables, you can begin instrumentation. Ensure that your setup meets the following requirements:

- Existing AWS Lambda functions built with Node.js >= 10.x
- Active Serverless APM for AWS Lambda subscription
- AppDynamics SaaS Controller >= 4.5.11

This document contains links to Amazon Web Services (AWS) documentation. AppDynamics makes no representation as to the accuracy of Amazon documentation because Amazon controls its own documentation.

Install the Node.js Serverless Tracer Using `npm` During Development

To instrument your lambda we recommend the AWS Lambda Extension. You can also install the Node.js Serverless Tracer using these steps:

1. Download the Node.js Serverless Tracer
2. Add the Require Statement
3. Instrument the Function with the Node.js Serverless Tracer

Download the Node.js Serverless Tracer

To install the tracer, navigate to the directory where the `package.json` file resides. Run the following `npm` command:

```
npm install appdynamics-lambda-tracer --save
```

The command downloads the tracer dependency into a `node_modules/` folder.

Add the Require Statement

To add the tracer to your application, add a `require` statement in your application code's primary file that includes the AWS Lambda `handler` function:

```
const tracer = require('appdynamics-lambda-tracer');
```

Remember to add the tracer's `require` statement before any other `require` statement.

Instrument the Function with the Node.js Serverless Tracer

Initialize the Node.js Serverless Tracer below the `require` statement, by calling the `tracer.init` method.

To complete instrumentation, call `tracer.mainModule(module)` as the last line of your code.

The code snippet below demonstrates how to instrument a serverless function with the tracer:

```
const tracer = require('appdynamics-lambda-tracer');
//Initialize the tracer
tracer.init();

// Your AWS Lambda handler function code here, for example:
exports.handler = async (event) => {
    // TODO implement
    const response = {
        statusCode: 200,
        body: JSON.stringify('Hello from Lambda!'),
    };
    return response;
};
//Complete the instrumentation
tracer.mainModule(module);
```

By default, the Node.js Serverless Tracer automatically discovers exit calls passed over HTTP or inter-AWS Lambda calls. You need to create an exit call if your function does not use these protocols. See Node.js Serverless Tracer API.

Next Steps

You are ready to Verify the Serverless Tracer Instrumentation.

You can also Integrate the Node.js Serverless Tracer with End User Monitoring or use the Node.js Serverless Tracer API.

Customize the Node.js Serverless Tracer

This page describes how to fine-tune your configuration with the Node.js Serverless Tracer API.

Create or Modify Exit Calls

If your function does not use HTTP or inter-AWS Lambda exit calls, you will need to create an exit call.

The following code snippet demonstrates how to create an exit call for an AWS Lambda function that calls a MySQL database:

```
// Use the exit call API in your AWS Lambda handler function

module.exports.myLambdaHandler = async function () {
  const queryPromise = new Promise(function (resolve, reject) {
    const dbConfig = {
      host: '127.0.0.1', // The host where MySQL server runs
      port: 3306, // Note that this is not an identifying property. Identifying properties must be strings
      user: 'root',
      database: 'movies'
    };
    const mysql = require('mysql');
    const mysqlClient = mysql.createPool(dbConfig);
    mysqlClient.getConnection(function (err, connection) {
      if (err) {
        reject({
          statusCode: 500,
          body: 'Couldn\'t make the connection to mysql client.'
        });
        return;
      }
      // Create an Exit Call
      var exitCall = tracer.startExitCall({
        exitType: 'DB',
        exitSubType: 'DB',
        identifyingProperties: {
          'HOST': '127.0.0.1', // The host where MySQL server runs
          'PORT': '3306',
          'DATABASE': 'movies',
          'VENDOR': 'MYSQL'
        }
      });
      connection.query('SELECT * FROM ClassicHits', function (err, results) {
        connection.release();
        if (err) {
          reject({
            statusCode: 500,
            body: 'Mysql query failed'
          });
          return;
        }

                // Stop the Exit Call
        tracer.stopExitCall(exitCall);
        resolve({
          statusCode: 200,
          body: JSON.stringify(results)
        });
      });
    });
  });
  return queryPromise;
}
```

Exit Call Types and Identifying Properties

Each exit call in the Controller has a distinct type determined by a set of identifying properties. The exit call types, subtypes, and identifying properties are listed below. For exit call subtypes, you can use the examples in the table or an empty string. Identifying properties must be strings.

Exit Call Type	Exit Call Subtype	Identifying Properties
HTTP	HTTP	HOST PORT URL QUERY STRING
DB	DB	URL HOST PORT DATABASE VERSION VENDOR
CACHE	CACHE	VENDOR SERVER POOL
CUSTOM	CASSANDRA CQL COUCHBASE AMAZON WEB SERVICES MONGODB	Any user-defined set of properties.

Transaction Correlation through API-defined Exits

If you manually create an exit call for the Node.js Serverless Tracer, you need to locate the correlation header. This is required if your function does not use HTTP or inter-AWS Lambda exit calls.

The code snippet below demonstrates how you can create the correlation header:

```
module.exports.myLambdaHandler = function () {
    // Start Exit Call
    var exitCall = tracer.startExitCall({
      // Fill in with the exit call information
    });

    //Get your correlation header
    if (exitCall) {
        var correlationHeader = tracer.createCorrelationInfo(exitCall);
    }

    /*
     * Your exit call code.
     * Pass the correlation header in the exit call for downstream correlation.
     */

    // Stop the exit call
    if (exitCall) {
        tracer.stopExitCall(exitCall);
    }
}
```

Transaction Error Reports

By default, exceptions are automatically detected by the tracer. If your business logic defines the transaction as "in error" for any other reason, you can customize the error report method to mark the transaction in error.

The following code snippet demonstrates how to create a transaction error report:

```
// Transaction Error Report
module.exports.apiTransactionErrorReport = function(event, context, callback) {
  setTimeout(function () {
    tracer.reportTransactionError(new Error('API Error'));
    callback(null, {
      statusCode: 200,
      body: 'Transaction Error Reported'
    });
  }, 100);
}
```

Exit Call Error Reports

By default, the tracer automatically discovers exit calls passed over HTTP or inter-AWS Lambda calls. Use the `reportExitCallError` method to report errors and exceptions that occur during exit call execution.

The following code snippet demonstrates how to create an exit call error report:

```
// Use the Exit Call API in your AWS Lambda handler function

module.exports.myLambdaHandler = async function () {
  const queryPromise = new Promise(function (resolve, reject) {
    const dbConfig = {
      host: '127.0.0.1', // The host where your MySQL server runs
      port: 3306, // Note that this is not an identifying property. Identifying properties must be strings
      user: 'root',
      database: 'movies'
    };
    const mysql = require('mysql');
    const mysqlClient = mysql.createPool(dbConfig);
    mysqlClient.getConnection(function (err, connection) {
      if (err) {
        reject({
          statusCode: 500,
          body: 'Couldn\'t make the connection to mysql client.'
        });
        return;
      }
      // Create an Exit Call
      var exitCall = tracer.startExitCall({
        exitType: 'DB',
        exitSubType: 'DB',
        identifyingProperties: {
          'HOST': '127.0.0.1', // The host where your MySQL server runs
          'PORT': '3306',
          'DATABASE': 'movies',
          'VENDOR': 'MYSQL'
        }
      });
      connection.query('SELECT * FROM ClassicHits', function (err, results) {
        connection.release();

                //Create an Exit Call error report
        if (err) {
          tracer.reportExitCallError(exitCall, 'MYSql error', 'Failed in making the query');
          reject({
            statusCode: 500,
            body: 'Mysql query failed'
          });
          return;
        }
        resolve({
          statusCode: 200,
          body: JSON.stringify(results)
        });
      });
    });
  });
  return queryPromise;
}
```

Override the Tracer's Default Behavior

If you prefer to customize the function/business transaction names or keep the tracer settings in your source code, override the environment variables to retrieve values at runtime from any desired source.

The following code snippet demonstrates how to override the default behavior of the tracer:

```
tracer.init({
  controllerHostName: 'controllerHost',
  controllerPort: 8080,
  accountAccessKey: 'controllerAccountAccessKey',
  accountName: 'controllerAccountName',
  applicationName: 'applicationName',
  tierName: 'tierName',
  serverlessApiEndpoint: 'serverlessApiEndpoint',
  proxyHost: 'proxyHost',
  proxyPort: '80',
  proxyUser: 'proxyUser',
  proxyPassword: 'proxyPassword',
  certificateFile: 'proxyCertFile',
  debug: true
});
```

Integrate the Node.js Serverless Tracer with End User Monitoring

This document contains links to AWS documentation. AppDynamics makes no representation as to the accuracy of Amazon documentation since Amazon controls its own documentation.

Serverless APM for AWS Lambda is designed to integrate with your existing End User Monitoring (EUM) configurations. EUM integration provides complete end-to-end visibility on the performance of your web and mobile applications, linking calls from an end-user device through your serverless functions to continue your business transactions

AWS Lambda functions can correlate EUM and AWS Lambda-originated business transactions, in conjunction with the following EUM agents:

- Browser Real User Monitoring (Browser RUM)
- Mobile Real User Monitoring (Mobile RUM)
- IoT Monitoring

This document assumes you are familiar with AppDynamics End User Monitoring. See EUM documentation for more information.

Before You Begin

Ensure your deployment has:

- An AWS Lambda function instrumented with the Node.js Serverless Tracer
- Active EUM licenses and JavaScript Agents

Add the EUM Environment Variable

By default, EUM integration is turned off for the Node.js Serverless Tracer.

To enable EUM integration, set the `APPDYNAMICS_ENABLE_EUM` environment variable to `true`. See Set Up the Serverless APM Environment for more details.

Configure Response Header Mappings

Configure response header mappings to pass ADRUM_*n* headers back to the browser. You can configure response header mappings:

- Using the AWS API Gateway
- Passing directly via AWS Lambda Proxy Integration
- Returning EUM metadata in your function code

Response Header Mappings in Amazon API Gateway

Add header mappings for each ADRUM_*n* header in the Amazon API Gateway. To configure response header mappings in the Amazon API Gateway, refer to the Amazon documentation.

The table below lists the required response headers and their mapping values:

Response Header	Mapping Value
`ADRUM_3`	`integration.response.body.headers.ADRUM_3`
`ADRUM_2`	`integration.response.body.headers.ADRUM_2`
`ADRUM_1`	`integration.response.body.headers.ADRUM_1`
`ADRUM_0`	`integration.response.body.headers.ADRUM_0`

Response Header Mappings in AWS Lambda Proxy Integration

As an alternative to custom integration, AWS Lambda proxy integration allows the client to call each function in the backend and returns output in a JSON format. Map these responses by adding each single-value ADRUM_*n* header directly to the headers field in your output object class. Refer to the Amazon documentation for details.

Response Header Mappings for All Other Invocations

To gather EUM metadata, you need to configure response header mappings. This code sample invokes an `appDEumHeaders` method on the tracer. This method returns an object containing the ADRUM_*n* headers. Pass this object to the downstream service for consumption by the downstream JavaScript Agent.

The code snippet below demonstrates how you can return EUM metadata:

```
module.exports.myLambdaHandler = function (event, context, callback) {
  setTimeout(function () {

        ////Call a transaction object
    let appDBusinessTxn = tracer.getCurrentTransaction();

        //Stop the transaction
    tracer.stopTransaction();

        //Return EUM Metadata
    let appDEumHeaders = tracer.getEumMetadata(appDBusinessTxn);
    callback(null, {
      eumMetadata: appDEumHeaders,
      data: 'Call to Lambda is a success'
    });
  }, 300);
}
```

Update the Front End Application

To see the correlation between EUM and AWS Lambda-originated business transactions, you must inject the JavaScript Agent into your browser's HTML, as described in Inject the JavaScript Agent.

Enable CORS Configuration

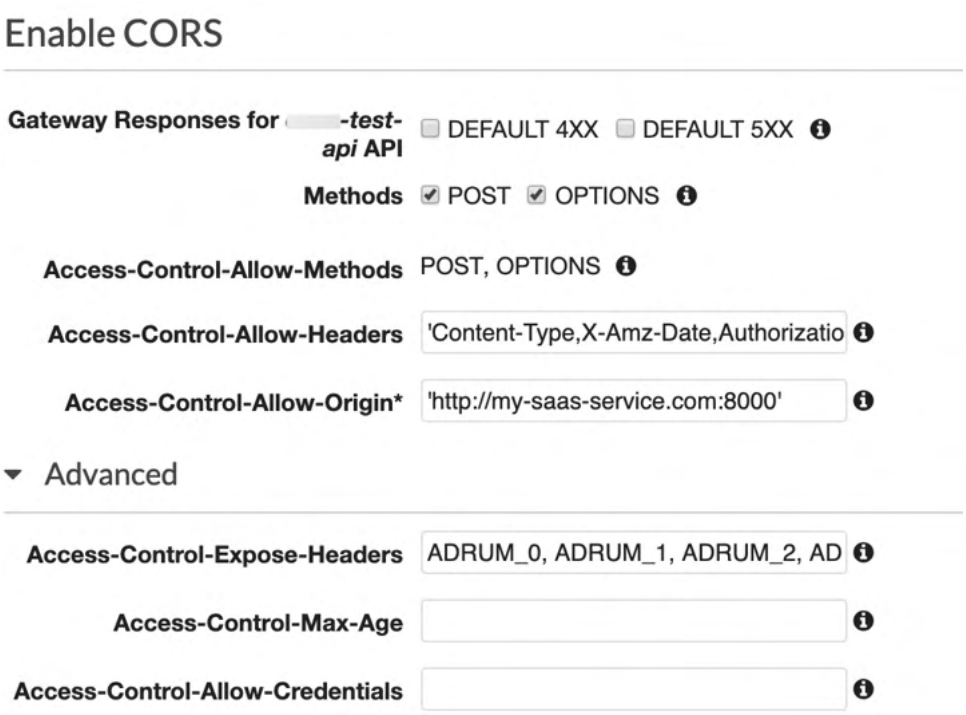
Browsers require cross-origin resource sharing (CORS) for functions that access responses from browser requests to a domain other than the base page's domain. You must explicitly grant cross-domain access to all applicable ADRUM_*n* headers.

To enable CORS configuration, define the required AWS response headers in the Amazon API Gateway:

- **Access-Control-Expose-Headers** - maps the custom ADRUM headers. These headers must have the names ADRUM_0 through ADRUM_3 to be compatible with the JavaScript Agent.
- **Access-Control-Allow-Methods** - check the header(s) with the appropriate method.
- **Access-Control-Allow-Origin** - set to the full URL of the web page that originated the request, including schema, hostname, and port.

See AWS documentation for advanced headers and additional details.

The screenshot illustrates how to enable CORS in the Amazon API Gateway:



After you have enabled CORS, you must define the response header mappings. To configure response header mappings in the Amazon API Gateway, refer to the Amazon documentation.

The table below lists response header mappings and example mapping values for CORS configuration:

Response Header	Example Mapping Value
`Access-Control-Expose-Headers`	`` `ADRUM_0,ADRUM_1,ADRUM_2,ADRUM_3` ``
`Access-Control-Allow-Credentials`	`` `true` ``
`Access-Control-Allow-Methods`	`` `POST,OPTIONS` ``
`Access-Control-Allow-Origin`	`` `http://my-saas-service.com:8000` ``

(Optional) If you connect your function to the Amazon API Gateway via a proxy integration, define the CORS response header mappings in your function code:

```
//Return EUM Metadata
    let appDEumHeaders = tracer.getEumMetadata(appDBusinessTxn);
    let returnHeaders = {
      'Access-Control-Allow-Origin': 'http://my-saas-service.com:8000',
      'Access-Control-Allow-Credentials': true,
      'Access-Control-Expose-Headers': 'ADRUM_0,ADRUM_1,ADRUM_2,ADRUM_3'
    };
```

Troubleshoot the EUM Integration

If you do not see a correlation between EUM data and business transactions, collect an HTTP Archive (HAR) file that captures the activity in your browser. Use the HAR file to confirm all necessary CORS headers are set and expected data populates in the ARUM_*n* headers.

The EUM documentation provides additional troubleshooting advice.

Python Serverless Tracer

This page provides requirements and an overview of the instrumentation process for Python functions during development. The preferred method is to use the AppDynamics AWS Lambda Extension. See Use the AppDynamics AWS Lambda Extension to Instrument Serverless APM at Runtime.

> This document contains links to Amazon Web Services (AWS) documentation. AppDynamics makes no representation as to the accuracy of Amazon documentation because Amazon controls its own documentation.

Before You Begin

Ensure that your setup meets the following requirements:

- Existing AWS Lambda functions implemented in Python >= 3.6
- Python Package Installer (the `pip` version must match the Python version of your AWS Lambda function)
- AppDynamics SaaS Controller >= 4.5.16

Install the Python Serverless Tracer

> The Python Serverless Tracer is available for Python >= 3.6. The version of the tracer you obtain from PyPi needs to match the version of Python you use to run your function code in AWS.

The Python Serverless Tracer is available for download in PyPi repository. You can install the Python Serverless Tracer locally or package it with your AWS Lambda function code.

Install Locally

Run a `pip install` command to install the tracer in your local environment:

```
pip install appdynamics-lambda-tracer
```

If you install the tracer locally, you need to package the tracer with your AWS Lambda function at runtime.

Package with AWS Lambda Functions

Run the following command to include the tracer in your function's package:

```
pip install --target ./package/ appdynamics-lambda-tracer
```

See AWS documentation to learn more about packaging dependencies in your functions.

Instrument Your Function Code

To instrument your AWS Lambda function, add these lines of code:

```
import appdynamics # Add AppDynamics libraries. Must be the first line of code

@appdynamics.tracer # Must come before the handler function
def my_handler(event, context):
        print("Hello world!")
```

Exit Call Instrumentation

By default, the Python Serverless Tracer automatically discovers HTTP, Amazon DynamoDB, and inter-AWS Lambda exit calls. Use the Python Tracer API to create an exit call if you want visibility into other types of external calls made by your function. See Python Serverless Tracer API.

> This document contains links to AWS documentation. AppDynamics makes no representation as to the accuracy of Amazon documentation because Amazon controls its own documentation

Python Serverless Tracer API

This page describes how to use the Python Serverless Tracer API to modify the behavior of the tracer within your AWS Lambda function.

Create Exit Calls

Exit calls are automatically instrumented for HTTP, Amazon DynamoDB, and inter-AWS Lambda calls. All other protocols need to manually create an exit call.

To create an exit call, use the `ExitCallContextManager` method in a `with` statement to wrap the exit call. The method takes the following *required* parameters:

- `exit_point_type`: Type of the exit call
- `exit_point_sub_type`: Subtype of the exit call
- `identifying_properties`: Properties defining the exit call

The `ExitCallContextManager` method automatically reports any runtime exceptions that occur within the method's scope as exit call errors.

The code sample below demonstrates how to use the exit call method for a MySQL database:

```
import appdynamics # Add AppDynamics libraries. Must be the first line of code

@appdynamics.tracer
def handler(event, context):
    with appdynamics.ExitCallContextManager(exit_point_type="DB", exit_point_sub_type="DB",
identifying_properties={"VENDOR": "MYSQL"}) as ec:
        data = fetch_from_db()
```

Exit Call Types and Identifying Properties

In the Controller, each exit call has a distinct type, determined by a set of identifying properties. Include at least one identifying property for each exit call type.

The `exitCall` types, subtypes, and identifying properties are listed below. For exit call subtypes, you can use the examples listed in the table or use an empty string.

Exit Call Type	Exit Call Subtype	Identifying Properties

`HTTP`	`HTTP`	`HOST`
		`PORT`
		`URL`
		`QUERY STRING`

`DB`	`DB`	`URL`
		`HOST`
		`PORT`
		`DATABASE`
		`VERSION`
		`VENDOR`

`CACHE`	`CACHE`	`VENDOR` `SERVER POOL`

`CUSTOM`	`Cassandra CQL`
	`Couchbase`
	`Amazon Web Services`
	`Mongo DB`
		Any user-defined set of properties

Create Exit Call Error Reports

The `ExitCallContextManager` method automatically reports errors. If you want to add custom error reports, use the `report_exit_call_error` method. The method takes the following parameters:

- `error_name:` Name of the error
- `error_message` (Optional): Descriptive error message
- `http_status_code` (Optional): The HTTP status code if available.

The code sample below demonstrates how you can use the exit call error report method:

```
import appdynamics
@appdynamics.tracer
def handler(event, context):
        #Create an exit call
    with appdynamics.ExitCallContextManager("DB", "DB", {"HOST": "ec2-12-123-123-12.us-west-2.compute.amazonaws.
com", "PORT": "3306", "DATABASE": "movies", "VENDOR": "MYSQL"}) as ec:
        movies = fetch_movies_from_mysql_db()
        if movies == None:
                        #Create an exit call error report
            ec.report_exit_call_error(error_name="DBError", error_message="Item not found") #ec is the object
returned by ExitCallContextManager above
```

Create Transaction Error Reports

By default, the Python Tracer automatically detects transaction error reports. You can customize these error reports using the `appdynamics.report_error(error_name: str, error_message: str)` method to report transaction errors. The method takes the following parameters:

- `error_name`: Name of the error
- `error_message` (Optional): Descriptive error message

The code sample below demonstrates how you can use the transaction error report method:

```
import appdynamics # Add AppDynamics libraries. Must be the first line of code
import requests #Add request library

@appdynamics.tracer # Must come before the handler function
def my_handler(event, context):
        try:
        r = requests.get('https://api.github.com/events')
    except Exception as e:
        appdynamics.report_error(error_name=type(e).__name__, error_message=str(e)) # Reports a transaction
error
```

Integrate the Python Tracer with End User Monitoring

Serverless APM for AWS Lambda is designed to integrate with your existing End User Monitoring (EUM) configurations. EUM integration provides complete end-to-end visibility on the performance of your web and mobile applications, linking calls from an end-user device through your serverless functions to continue your business transactions

AWS Lambda functions can correlate EUM and AWS Lambda-originated business transactions, in conjunction with the following EUM agents:

- Browser Real User Monitoring (Browser RUM)
- Mobile Real User Monitoring (Mobile RUM)
- IoT Monitoring

This document assumes you are familiar with AppDynamics End User Monitoring. See End User Monitoring.

Before You Begin

Ensure your deployment has:

- An AWS Lambda function instrumented with the Python Serverless Tracer
- Active EUM licenses and JavaScript Agents

Add the EUM Environment Variable

By default, EUM integration is turned off for the Python Serverless Tracer.

To enable EUM integration, set the `APPDYNAMICS_ENABLE_EUM` environment variable to `true`. See Set Up the Serverless APM Environment.

Configure Response Header Mappings

Configure response header mappings to pass ADRUM_*n* headers back to the browser. You can configure response header mappings:

- Using the AWS API Gateway
- Passing directly via AWS Lambda Proxy Integration
- Returning EUM metadata in your function code

Response Header Mappings in Amazon API Gateway

Add header mappings for each ADRUM_*n* header in the Amazon API Gateway. To configure response header mappings in the Amazon API Gateway, refer to the Amazon documentation.

This table lists the required response headers and their mapping values:

Response Header	Mapping Value
`ADRUM_3`	`integration.response.body.headers.ADRUM_3`
`ADRUM_2`	`integration.response.body.headers.ADRUM_2`
`ADRUM_1`	`integration.response.body.headers.ADRUM_1`
`ADRUM_0`	`integration.response.body.headers.ADRUM_0`

Response Header Mappings in AWS Lambda Proxy Integration

As an alternative to custom integration, AWS Lambda proxy integration allows the client to call each function in the backend and returns output in a JSON format. Map these responses by adding each single-value ADRUM_*n* header directly to the headers field in your output object class. See the Amazon documentation.

Response Header Mappings for All Other Invocations

To gather EUM metadata, you need to configure response header mappings. This code sample invokes an `appDEumHeaders` method on the serverless tracer. This method returns an object containing the ADRUM_*n* headers. Pass this object to the downstream service for consumption by the downstream JavaScript Agent.

The code snippet below demonstrates how you can return EUM metadata:

```
@appdynamics.tracer
def my_lambda_handler(event, context):


        #Get the current transaction object
    transaction = appdynamics.get_current_transaction()


        #Stop the transaction
    appdynamics.stop_transaction()


        #Return EUM metadata
    eum_headers = appdynamics.get_eum_metadata(transaction)
    return {
        "eum_metadata": eum_headers,
        "data": "call to Lambda is a success"
    }
```

Update the Front End Application

To see the correlation between EUM and AWS Lambda-originated business transactions, you must inject the JavaScript Agent into your browser's HTML, as described in the Inject the JavaScript Agent.

Enable CORS Configuration

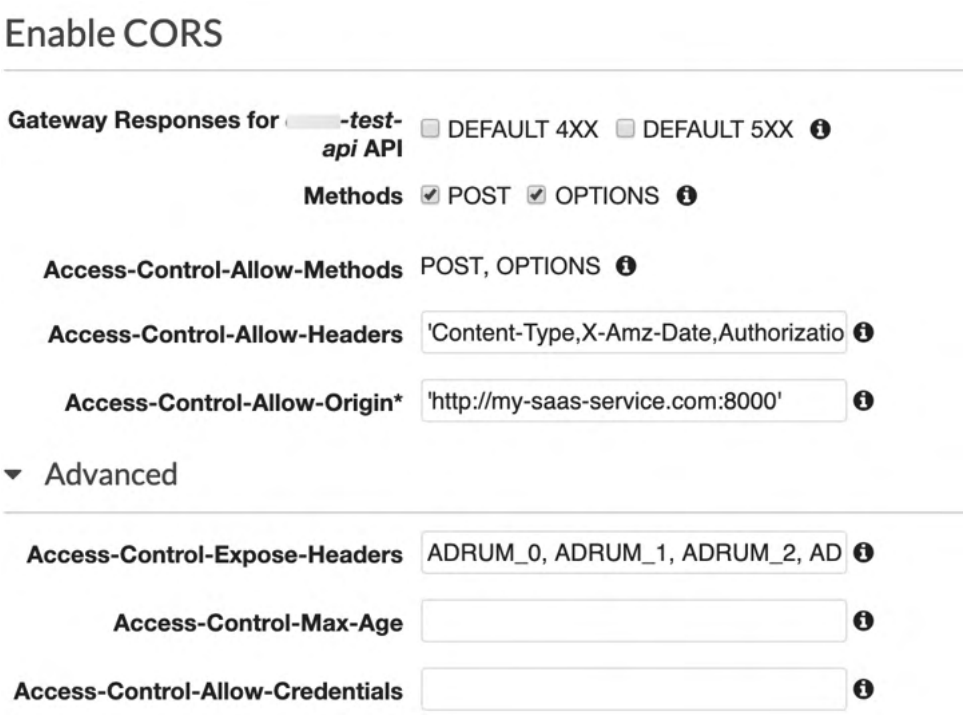
Browsers require cross-origin resource sharing (CORS) for functions that access responses from browser requests to a domain other than the base page's domain. You must explicitly grant cross-domain access to all applicable ADRUM_*n* headers.

To enable CORS configuration, define the required AWS response headers in the Amazon API Gateway:

- **Access-Control-Expose-Headers** - maps the custom ADRUM headers. These headers must have the names ADRUM_0 through ADRUM_3 to be compatible with the JavaScript Agent.
- **Access-Control-Allow-Methods** - check the header(s) with the appropriate method.
- **Access-Control-Allow-Origin** - set to the full URL of the web page that originated the request, including schema, hostname, and port.

See AWS documentation for advanced headers and additional details.

The screenshot illustrates how to enable CORS in the Amazon API Gateway:



After you have enabled CORS, you must define the response header mappings. To configure response header mappings in the Amazon API Gateway, see the Amazon documentation.

This table lists response header mappings and example mapping values for CORS configuration:

Response Header	Example Mapping Value
`Access-Control-Expose-Headers`	`` `ADRUM_0,ADRUM_1,ADRUM_2,ADRUM_3` ``
`Access-Control-Allow-Credentials`	`` `true` ``
`Access-Control-Allow-Methods`	`` `POST,OPTIONS` ``
`Access-Control-Allow-Origin`	`` `http://my-saas-service.com:8000` ``

(Optional) If you connect your function to the Amazon API Gateway via a proxy integration, define the CORS response header mappings in your function code:

```
# Return EUM Metadata
eum_headers = appdynamics.get_eum_metadata(transaction)
        return_headers = {
      'Access-Control-Allow-Origin': 'http://my-saas-service.com:8000',
      'Access-Control-Allow-Credentials': True,
      'Access-Control-Expose-Headers': 'ADRUM_0,ADRUM_1,ADRUM_2,ADRUM_3'
        }
```

Troubleshoot the EUM Integration

If you do not see a correlation between EUM data and business transactions, collect an HTTP Archive (HAR) file that captures the activity in your browser. Use the HAR file to confirm all necessary CORS headers are set and expected data populates in the ARUM_*n* headers.

The EUM documentation provides additional troubleshooting advice.

This document contains links to AWS documentation. AppDynamics makes no representation as to the accuracy of Amazon documentation since Amazon controls its own documentation.

Verify the Serverless Tracer Instrumentation

This document contains links to AWS documentation. AppDynamics makes no representation as to the accuracy of Amazon documentation because Amazon controls its own documentation.

After you set environment variables and instrument the Serverless Tracer, you can verify successful instrumentation.

Test the Function

Deploy your function to AWS and test in the AWS Management Console.

You need to create a test load on your function to see activity within AppDynamics. If your test log yields no errors, the tracer successfully instrumented.

To protect your production code, the tracer never fails your function. Your function may execute successfully yet fail to instrument the tracer. Check the logs of each AWS Lambda test in the AWS Management Console to confirm Serverless APM successfully instrumented.

View in the Controller

Your AWS Lambda function appears on the flow map after a few minutes of processing application load.

To view your function in AppDynamics:

1. Log in to your AppDynamics Controller.
2. Navigate to **Applications** and find your new application.

Apache Web Server Agent

Related pages:

- Apache Web Server Backend Detection
- Transaction Detection for Apache Web Servers

The Apache Web Server Agent discovers, maps and tracks metrics for business transactions, app services, and backends in your web application by injecting instrumentation into the Apache server at runtime.

About the Apache Web Server Agent

The Apache Agent automatically discovers incoming HTTP requests to Apache Web Server nodes as business transaction entry points. AppDynamics uses URI-based naming for transactions originating at Apache Agent nodes. The agent discovers exit points to Apache modules and resolves them to the downstream tier or backend. See Apache Web Server Backend Detection.

Business transaction naming is limited to URLs or the WEB type. This provides limited options for match and split rules.

The Apache Agent tracks the following metrics for web servers:

- Calls per minute and time spent in the web server tier
- Calls per minute and time spent in Apache module backends
- Number of errors

The agent identifies slow and stalled transactions according to dynamic baselines. When it detects an HTTP response code greater than 400, the agent identifies the transaction as an error transaction.

Transaction snapshots taken on Apache Web Server nodes include the following data:

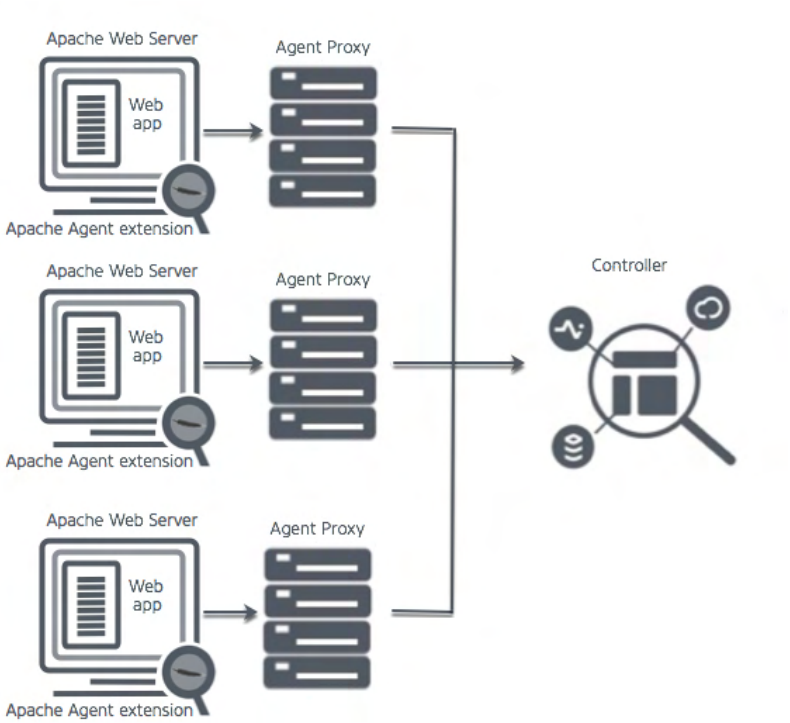
- URL
- HTTP error codes
- HTTP request data:
    - Cookie, including JSESSIONID
    - Referer
    - X-Forwarded-For
    - SM_USER
- Exit calls to Apache modules

Agent Deployment Architecture

The Apache Web Server Agent consists of these components:

- Apache extension
- AppDynamics agent proxy

The agent proxy is a Java process that handles communication between the Apache Web Server Agent and Controller. The proxy transmits the raw performance data from the agent to the Controller, which stores, analyzes, and presents the data.



As a separate process, the agent proxy must be started separately from the monitored web server. You can start the proxy manually or configure it to start automatically by setting `AppDynamicsLaunchProxy` in `appdynamics_agent.conf`. Whether the proxy is started automatically or manually, the commands that start the agent check whether the proxy is already running and do not attempt to start it again if it is.

Naming Apache Web Server Nodes

Each web server instance maps to a node in the AppDynamics model. When naming Apache Web Server nodes, be sure to use a meaningful name. Some options are:

- `hostName-appName-nodeName`
- `hostName-tierName-nodeName`
- `appName-nodeName`
- `tierName-nodeName`
- IP address
- fully qualified domain name

Supported Apache Web Servers

Related pages:

- Browser RUM Supported Environments

Apache Server Agent Support

Apache Web Servers

Supported Apache Web Server Version
• Apache HTTP Server 2.2.x • Apache HTTP Server 2.4.x • IBM HTTP Server >= 7.0 • Oracle HTTP Server >= 11g

Operating Systems

- Any Linux distribution based on glibc >= 2.12
- RHEL5 is not supported from 20.x onwards. Support is applicable for glibc2.5+ (RHEL6 or later).

> If you encounter an ELF warning message, it may be related to the incorrect version of glibc. Verify that your operating system has the suitable glibc version.

Automatically Discovered Business Transactions

The Apache Agent automatically discovers the following business transactions:

Type	Custom Configuration Options	Downstream Correlation
Web (HTTP)	Yes	Yes

By default the agent excludes requests for the following static file types:

bmp
cab
class
conf
css
doc
gif
ico
jar
jpeg
jpg
js
mov
mp3
mp4
pdf
png
pps
properties
swf
tif
txt
zip

Remote Service Detection

Apache Modules

The Apache Agent automatically detects loaded Apache modules as remote services. The agent excludes a list of common modules from detection.

- core.c
- http_core.c
- mod_access_compat.c
- mod_actions.c
- mod_alias.c
- mod_allowmethods.c
- mod_appdynamics.cpp
- mod_auth_basic.c
- mod_auth_digest.c
- mod_authn_alias.c
- mod_authn_anon.c
- mod_authn_core.c
- mod_authn_dbd.c
- mod_authn_dbm.c
- mod_authn_default.c
- mod_authn_file.c
- mod_authn_socache.c
- mod_authnz_ldap.c
- mod_authz_core.c
- mod_authz_dbd.c
- mod_authz_dbm.c
- mod_authz_default.c
- mod_authz_groupfile.c
- mod_authz_host.c
- mod_authz_owner.c
- mod_authz_user.c
- mod_autoindex.c
- mod_cache.c
- mod_cache_disk.c
- mod_cgi.c
- mod_data.c
- mod_dbd.c
- mod_deflate.c
- mod_dir.c
- mod_disk_cache.c
- mod_dumpio.c
- mod_echo.c
- mod_env.c
- mod_expires.c
- mod_ext_filter.c

mod_file_cache.c

mod_filter.c

mod_headers.c

mod_include.c

mod_info.c

mod_lbmethod_bybusyness.c

mod_lbmethod_byrequests.c

mod_lbmethod_bytraffic.c

mod_lbmethod_heartbeat.c

mod_log_config.c

mod_logio.c

mod_lua.c

mod_mem_cache.c

mod_mime.c

mod_mime_magic.c
mod_negotiation.c

mod_perl.c

mod_python.c

mod_remoteip.c

mod_reqtimeout.c

mod_rewrite.c

mod_setenvif.c

mod_slotmem_plain.c

mod_slotmem_shm.c

mod_so.c

mod_socache_dbm.c

mod_socache_memcache.c

mod_socache_shmcb.c

mod_speling.c

mod_ssl.c

mod_status.c

mod_substitute.c

mod_suexec.c

mod_systemd.c

mod_unique_id.c

mod_unixd.c

mod_userdir.c

mod_usertrack.c

mod_version.c

mod_vhost_alias.c

prefork.c

util_ldap.c

For End User Monitoring, the Apache Agent does not support automatic injection of the Javascript `adrum` header and footer to instrument web pages.

Third-Party Modules

The Apache Agent >= 20.5 requires the use of the latest version of SiteMinder 12.52 SP1 to eliminate a known bug in SiteMinder.

Site Minder

Product Name=CA SiteMinder Web Agent
FullVersion=12.52.110.2813
Version=12.52
Update=110
Build Number=2813

Install the Apache Agent

Related pages:

- Download AppDynamics Software
- Install the Machine Agent
- SELinux Installation Issues

To monitor the performance of your Apache HTTP Server, IBM HTTP Server (IHS) or Oracle HTTP Server, install the AppDynamics Apache Agent on the servers where you run Apache, IHS or OHS. The agent instruments the Apache server and sends performance data to a Java proxy, which in turn sends data to the AppDynamics Controller.

If you are using the Apache Monitoring Extension with the Standalone Machine Agent, you can continue to use it. You may need to restart the Standalone Machine agent after installing the Apache Agent.

Before Starting

Ensure that the web server version and operating system are supported. See Supported Apache Web Servers.

Also verify the following requirements on the machine on which you are installing the agent:

- Perform the installation under the same user and group that Apache is using when spawning worker processes and threads.
- Ensure that the machine on which the agent runs has random or urandom set appropriately. This is because certain functions in the Apache Agent are dependent on random number generation.
  Else, it can lead to abrupt behavior such as log files or directories may not be created, nothing is reported from the agent to the Controller, and so on.
- Ensure that the ulimit setting for open file descriptors is set such that neither the Apache worker process or the Proxy task does not exceed it to avoid resource contention. **
- Check the open file descriptor limit of the userid that the proxy and/or Apache worker process must have:
  For the MPM_Worker mode, the number of file descriptors must be set using the following calculation: **1024 + ServerLimit + (2 * ServerLimit * ThreadsPerChild)**.
  For all other modes, you must set the number of file descriptors using the calculation: **(Total number of concurrently active Apache workers * 2) + 1024.**
  The open file descriptor setting can be found in /etc/security/limits.conf, but the setting requires a host reboot to become effective, or set through the "ulimit -n" command for the current userid's session only.
- Check for modules with dependencies on `libstdc++.so.5`. See Special Considerations for Apache Web Servers with libstdc++.so.5 Modules. T his is most common with IHS and OHS.
- Ensure that the machine on which the agent runs has a locale environment set properly. As some functions in the Apache Agent have a dependency on the locale it could lead to a crash otherwise.
  Based on the agent's requirement, you can select one of the following and set the environment variable as:

```
export LANG=en_US.UTF-8
export LANGUAGE=$LANG
export LC_ALL=$LANG
```

Or

```
export LANG=en_US.UTF-8
```

Or

```
export LC_ALL=C
```

Download and Install the Apache Agent

To guarantee performance and stability, you should install one agent for each Apache instance. Do not exceed two Apache instances per agent.

1. Download the Apache Agent from the Getting Started wizard or https://accounts.appdynamics.com/downloads.
2. Extract the agent to `/opt`.

   ```
   tar -xzvf ./appdynamics-sdk-native.<version>.tar.gz -C /opt
   ```

   The agent installs to `/opt/appdynamics-sdk-native`. This is the `<agent_install_directory>`. If you are installing multiple agents, you must create separate install directories for each agent.
3. Grant the Apache, IHS, or OHS owner read and write permissions to the logs directory: `<agent_install_directory>/logs`.
4. Run `install.sh`. This script installs the agent proxy.

```
<agent_install_directory>/install.sh
```

5. For an agent to monitor your Apache instance, you must launch the proxy. If you are monitoring multiple Apache instances, we recommend that you manually launch each proxy, and set `AppDynamicsLaunchProxy` to `NO` for every Apache instance.

   The next step is to create the agent configuration file and configure the agent.

For cURL installation, see Download AppDynamics Software.

Configure the Apache Agent

1. Create a configuration file for the AppDynamics Apache Web Server agent, such as `appdynamics_agent.conf`, in the Apache, IHS, or OHS configuration directory. For example:

```
touch /etc/<path_to_webserver_dir>/conf/appdynamics_agent.conf
```

2. In the `appdynamics_agent.conf` file, configure the following settings. If you have multiple agents installed, you must edit the appdynamics_agent.conf file for each agent, specifying which directory the agent load modules are located. Ensure that each Apache instance references a different `appdynamics_agent.conf` file, and that each file specifies a different agent install directory.

For multiple Apache web server instances that share the same Apache Agent, ensure that Comm dirs, set using `AppdynamicsProxyCommDir` dire ctive, for each proxy process is different. Also ensure that the node names in each server configuration file, `appdynamics_agent.conf`, are also different.

- `LoadFile`: Loads the AppDynamics Agent SDK shared library. Assuming the agent is installed in `/opt`, the setting would be `/opt/appdynamics-sdk-native/sdk_lib/lib/libappdynamics_native_sdk.so`. Required.
  For example:

```
LoadFile /opt/appdynamics-sdk-native/sdk_lib/lib/libzmq.so.3
LoadFile /opt/appdynamics-sdk-native/sdk_lib/lib/libuuid.so.1
LoadFile /opt/appdynamics-sdk-native/sdk_lib/lib/libappdynamics_native_sdk.so
```

  libappdynamics_native_sdk.so has a runtime dependency on libzmq and libuuid. These libraries can be found in the same directory as ibappdynamics_native_sdk.so.
- `LoadModule`: Loads the AppDynamics Apache Agent shared library. The correct module is `libmod_appdynamics.so` for Apache 2.4 and `libmod_appdynamics22.so` for Apache 2.2. Required. IHS and OHS require the Apache 2.2 module.
  Apache 2.4 example:

```
LoadModule appdynamics_module /opt/appdynamics-sdk-native/WebServerAgent/Apache/libmod_appdynamics.
so
```

  Apache 2.2 example:

```
LoadModule appdynamics_module /opt/appdynamics-sdk-native/WebServerAgent/Apache
/libmod_appdynamics22.so
```

- `AppDynamicsEnabled`: Must be included and set to ON for web server monitoring to be enabled. Otherwise, monitoring is disabled by default. For example:

```
AppDynamicsEnabled ON
```

  To disable monitoring, set to `OFF` or remove the AppDynamics module `include` statement (see Configure the Apache Server for AppDynamics) from the Apache configuration file.

- `AppDynamicsControllerHost`: Hostname or IP address of the AppDynamics controller to connect to. Required.
  For example

```
AppDynamicsControllerHost mycontroller.saas.appdynamics.com
```

- `AppDynamicsControllerPort`: Controller HTTP(S) port. For SaaS, use 443 for HTTPS or 80 for HTTP. For on-premises, HTTP defaults to 8090 and HTTPS defaults to 8181. Optional. Configure if you do not want to use these defaults. If you are using HTTPS, the SSL setting (`AppDynamicsControllerSSL`) must also be set.
  For example:

```
AppDynamicsControllerPort 80
```

- `AppDynamicsControllerSSL`: Set to connect to the Controller over SSL. Default is `OFF`. Set to `ON` to enable SSL connection and also set the `AppDynamicsControllerPort` to an HTTPS port.
  Example:

```
AppDynamicsControllerSSL OFF
```

- `AppDynamicsAccountName`: AppDynamics account name. For on-premises controllers, find your credentials under  **> License** in the Controller. For SaaS Controllers, AppDynamics provides your credentials in your Welcome email. Required.
  Example:

```
AppDynamicsAccountName MyCompany
```

- `AppDynamicsAccessKey`: AppDynamics account access key. For on-premises Controllers, find your credentials under  **> License** i n the Controller. For SaaS Controllers, AppDynamics provides your credentials in your Welcome email. Required.
  Example:

```
AppDynamicsAccessKey zd8yjh5yuy5k
```

- `AppDynamicsProxyHost`: Proxy server hostname or IP address. Optional. Use if the agent connects to the Controller through a HTTP proxy server.
- `AppDynamicsProxyport`: Proxy server port. Optional. Use if the agent connects to the Controller through a HTTP proxy server
- `AppDynamicsApplication`: AppDynamics application name. See Overview of Application Monitoring. Required.
  For example:

```
AppDynamicsApplication MyWS2App
```

- `AppDynamicsTier`: AppDynamics tier name. See Overview of Application Monitoring. Required.
  For example:

```
AppDynamicsTier MyWS2
```

- `AppDynamicsNode`: AppDynamics node name. See Overview of Application Monitoring. The node name is required and each node name must be unique.
  For example:

```
AppDynamicsNode WS2_1
```

```
#Load the AppDynamics SDK.
LoadFile /opt/appdynamics-sdk-native/sdk_lib/lib/libzmq.so.3
LoadFile /opt/appdynamics-sdk-native/sdk_lib/lib/libuuid.so.1
LoadFile /opt/appdynamics-sdk-native/sdk_lib/lib/libappdynamics_native_sdk.so

#Load the Apache Agent. In this example for Apache 2.4
LoadModule appdynamics_module /opt/appdynamics-sdk-native/WebServerAgent/Apache/libmod_appdynamics.so

AppDynamicsEnabled On

#AppDynamics Controller connection.
AppDynamicsControllerHost mycontroller.saas.appdynamics.com
AppDynamicsControllerPort 80
AppDynamicsControllerSSL OFF

#Account credentials
AppDynamicsAccountName MyCompany
AppDynamicsAccessKey zd8yjh5yuy5k

#Configure Controller connection through an HTTP proxy server.
#AppDynamicsProxyHost <proxy host>
#AppDynamicsProxyPort <proxy port>

#Business application, tier, node
AppDynamicsApplication MyApache2App
AppDynamicsTier Apache2
AppDynamicsNode Apache2_1
```

Launch the Proxy

To launch the proxy manually, run `runSDKProxy.sh` using the user ID of the Apache worker threads.

```
nohup <agent_install_directory>/runSDKProxy.sh >>/dev/null 2><agent_install_dir>/logs/proxy.out &
```

It is recommended that you create a Unix System Service that automatically starts the proxy at system startup.

The Proxy expects the appropriate (64- vs. 32-bit) `libstdc+6` library location. If you are installing the Apache Agent on a 64-bit OS that supports running a 32-bit Apache HTTP Server, make sure that the downloaded Apache Agent matches (i.e., is 32-bit), and that the `libstdc+6` library is 32-bit as well. If you have both 64-bit and 32-bit versions of that library, make sure you are pointing to the correct version using the command `export LD_PRELOAD=<path to 32bit library>` before starting `runSDKProxy`.

Additional Directives

You may need to set the following optional directives unless instructed by AppDynamics support:

- `AppDynamicsBackendNameSegments-` The number of URL segments that are displayed in the backend naming. This is only applicable for requests routing through the mod_proxy, when Apache is acting as a reverse proxy server, also-known-as a gateway server. Specify the rule configuration for backend naming in the `appdynamics_agent.conf` file. The default value is 0 and only the IP address of the request URL is displayed. If you set a value of 2, then the first two segments along-with the IP address are displayed. Therefore for a request proxying to `http://1.1.1.1/services/admin/abc`, the backend isnamed as `http://1.1.1.1/services/admin`.

  For example:

```
AppDynamicsBackendNameSegments 2
```

- `AppDynamicsResolveBackends-` Controls the way Apache module backends appear in the Controller UI. If ON, metrics for Apache modules are shown as part of downstream tiers or backends. Module backends do not appear on flow maps.
  If OFF, Apache module backends appear in the flow maps with their own metrics. See 'Apache Modules' on Install the Apache Agent .
  Defaults to ON. Optional.
  For example:

  ```
  AppDynamicsResolveBackends OFF
  ```

- `AppDynamicsTraceAsError-` If `ON`, tracepoints are written as errors to the Apache logs, by default `error_log`. Default is `OFF`. Optional.
  For example:

  ```
  AppDynamicsTraceAsError OFF
  ```

- `AppDynamicsReportAllInstrumentedModules-` If `OFF`, the agent reports only modules that occur during the `HANDLER` stage of Apache request processing. If ON, the agent reports modules during all stages of Apache request processing. The default is `OFF`.

  For example:

  ```
  AppDynamicsReportAllInstrumentedModules OFF
  ```

- `AppDynamicsLaunchProxy-` If `ON`, the agent launches the proxy automatically when on startup. If `OFF`, you need to manually launch the proxy. If your system uses any command to do a graceful restart of Apache, such as log rotate, or if your system encounters heavy loads, you should set this property to `OFF`. The default is `OFF`.
  For example:

  ```
  AppDynamicsLaunchProxy OFF
  ```

  For information about how to examine and resolve proxy issues that may prevent an application agent from connecting to the Controller, see Dynamic Language Agent Proxy.

  You may want to launch the proxy manually for debugging. For example, to set a different proxy root directory or runtime directory or to output additional debugging information.
- `AppDynamicsProxyCommDir-` Path to the directory containing the domain control socket, which the agent uses to start an AppDynamics node. Defaults to `<agent_install_directory>/logs/appd-sdk`. If you modify this setting, make sure that the directory path does not exceed 107 characters, which is the Linux limit to the socket file name size. If the directory path is too long, you will get an error message when the agent attempts to start.

  ```
  AppDynamicsProxyCommDir <agent_install_directory>/proxy/altcommdir
  ```

- For Filtering sensitive-data and privileged information, see Filter Sensitive Data.

Configure the Apache Server for AppDynamics

1. Add an include for the AppDynamics Apache Agent configuration to the server configuration. Ensure that the AppDynamics module is loaded after any modules you want to be instrumented. Any modules loaded after the AppDynamics agent are excluded from instrumentation and monitoring. For example, add the following as the last line in the httpd`.conf` file—on Ubuntu and Debian, this file is called `apache2.conf`:

   ```
   #Include AppDynamics Apache Agent Configuration
   Include conf/appdynamics_agent.conf
   ```

   If you are running multiple instances of the web server, modify the `.conf Include` path to match the configuration file name for that instance.
   For example:

   ```
   #Include AppDynamics Apache Agent Configuration.
   Include conf/appdynamics_agent1.conf
   ```

   > For versions upto 4.x, if you set values for Apache thread limits such as ServerLimit, ThreadsPerChild, and ThreadLimit, you need to ensure that the concurrent thread limit does not exceed 4095. For versions, 20.x and later this limit is not applicable. Users can create as many threads as allowed by their system, depending on the available resources.

2. Restart the web server. For example, for Apache on CentOS:

   ```
   apachectl restart
   ```

3. Apply load to your web server to activate instrumentation.

The AppDynamics Apache Agent automatically detects the incoming HTTP calls as business transactions. It detects loaded modules as backends. Log in to the AppDynamics Controller to begin monitoring.

Map Virtual Hosts to AppDynamics Tiers

By configuring virtual hosts, Apache Web Server administrators can have a single Apache Web Server instance act as the entry point for what appears to be different websites from the end user's perspective.

In the AppDynamics application model, it will usually make sense to represent each virtual host configured in an Apache Web Server as its own tier. This tends to better represent the logical model of the environment and reduces the likelihood of quickly exhausting tier business transaction limits for an Apache Web Server that proxies a large application environment.

To associate different virtual hosts with different tiers, add the AppDynamicsApplicationContext directive to the virtual host configuration, supplying the application, tier and node name as arguments, as follows:

```
AppDynamicsApplicationContext <application> <tier> <node>
```

For example:

```
Listen 80
<VirtualHost *:80>
    DocumentRoot "/www/example1"
    ServerName site1.example.com
    ...
    AppDynamicsApplicationContext MyApp site1.example.com:80 node01
</VirtualHost>

<VirtualHost *:80>
    DocumentRoot "/www/example2"
    ServerName site2.example.org
    ...
    AppDynamicsApplicationContext MyApp site2.example.org:80 node01
</VirtualHost>
```

In the example above, the tier name is a combination of the virtual machine ServerName and port number. You are not required to include the ServerName and port number in the tier name, but they can be useful for identifying which virtual machines are creating business transactions. We recommend using a tier name that best represents the purpose of the virtual machine.

The Apache Web Server Agent associates incoming requests to the AppDynamics context based upon the virtual host server and port in the request.

Note that instead of using virtual hosts to multi-host from Apache, an administrator may run multiple instances of the Apache Web Server with a distinct `httpd.conf` file—on Ubuntu and Debian, this file is called `apache2.conf`—for each instance. In this case, you would instrument the Apache Web Server by creating an AppDynamics agent configuration file for each instance, and include each configuration file in the `httpd.conf` file of the corresponding instance. See Configure the Apache Agent.

Apache Agent Directories

This page describes the directories for the Apache Agent.

Agent Install Directory

The agent install directory <agent_install_directory> is the appdynamics-sdk-native directory where you installed the agent.

For example, if you un-tarred the agent to /opt, the agent home would be: /opt/appdynamics-sdk-native.

It contains the following files:

- install.sh to install the Apache Agent and the agent proxy.
- runSDKProxy.sh to start the agent proxy.

WebServerAgent Directory

The <agent_install_directory>/WebServerAgent directory contains the Apache Agent libraries.

The Apache subdirectory contains the Apache modules for the Apache Agent:

- For Apache 2.2: libmod_appdynamics22.so
- For Apache 2.4: libmod_appdynamics.so

Conf Directory

The <agent_install_directory>/conf directory contains the logging configuration files:

- A logging configuration template: appdynamics_sdk_log4cxx.xml.template
- A log configuration file: appdynamics_sdk_log4cxx.xml

Logs Directory

The logs <agent_install_directory>/logs directory contains the logs for the Apache Agent and the agent proxy.

Proxy Directory

The proxy <agent_install_directory>/proxy directory contains the agent proxy files.

Proxy Control Directory

The proxy control directory contains the domain control socket, which the agent uses to start an AppDynamics node. Defaults to <agent_install_directory>/logs/appd-sdk. You need to pass this path to the runProxy script if you are starting the proxy manually.

Troubleshoot Apache Agent Installation

If you installed the Apache Agent and are experiencing problems, try these troubleshooting suggestions.

Undefined Symbol Error

An error message similar to the following typically indicates that you have loaded the incorrect agent library for the version of Apache you are using.

```
ERROR
httpd: Syntax error on line 953 of /srv/jas/app/embcms/HTTPServer/cma1-dev2/conf/httpd.conf: Syntax error on
line 7 of /srv/jas/app/embcms/HTTPServer/cda0-dev2/conf/appdynamics.conf: Cannot load /srv/jas/data/AppDynamics
/WebAgent/WebServerAgent/Apache/libmodappdynamics.so into server: /srv/jas/data/AppDynamics/WebAgent
/WebServerAgent/Apache/libmod_appdynamics.so: undefined symbol: ap_log_error
```

Check the description of LoadModule on Install the Apache Agent to make sure that you are loading the correct module. For example, for Apache 2.2, your LoadModule directive should be:

```
LoadModule appdynamics_module /srv/jas/data/AppDynamics/WebAgent/ApacheAgent/Apache/libmod_appdynamics22.so
```

App Crash Caused by libstdc++ Mismatch

The AppDynamics agent uses libstc++ v6. If your application uses a different version of libstc++ and the modules were not loaded in the correct order, the wrong library routines could be called. If you get a dump with a long stack trace, this could be the cause. See Apache with libstdc++5 Considerations for information on how to prevent this.

Tier Name Not Specific to the Apache Agent

When the Apache Agent and Machine Agent are hosted on the same server, and the Machine Agent is started before the Apache Agent, then the tier is registered as a Java tier that prevents the Apache Agent from registering itself.

An error message is displayed in the Apache Agent logs:

```
Is your agent attempting to register to the wrong tier?
```

Workaround

1. Stop the Machine Agent and Apache Agent.
2. Delete the tier from the Controller UI.
3. Restart the Apache Agent first and allow it to register the tier.
4. Restart the Machine Agent.

Apache with libstdc++5 Considerations

Related pages:

- Install the Apache Agent

The Apache Agent depends on libstdc++6. Some servers, primarily IHS and OHS, have modules built with libstdc++.so.5. This conflict can prevent Apache from starting successfully.

Check for libstdc++5 Dependencies

To determine if your web server has libstdc++.so.5 dependencies:

1. Change directory to your web server's modules directory.
2. Run this command:

```
ldd *.so 2>&1 | grep -v 'execution permission' | grep 'libstdc++.so.5'
```

3. If this command returns any results—for example, `libstdc++.so.5 => /lib64/libstdc++.so.5`—your server has a dependency on libstdc++.so.5.

Resolution Overview

To avoid libstdc++5 conflicts, force libstdc++6 to load before libstdc++5 when Apache is started. The general steps for performing this configuration are:

1. Identify the paths to libstdc++.so.6.
2. Determine the correct libstdc++.so.6 path for your installation.
3. Use the `LD_PRELOAD` environment variable to have libstdc++6 preloaded.

These steps are detailed in the following sections.

Identify the Paths to libstdc++.so.6

From a command prompt, run:

```
ldconfig -p | grep libstdc++.so.6
```

This will return one or more paths. For example:

```
/lib64/libstdc++.so.6
/lib/libstdc++.so.6
```

Determine the Correct libstdc++.so.6 Path for the Installation

If more than one path is returned, determine which is the correct path for your Apache Server installation.

If you are running a 64-bit operating system, but a 32-bit version of IHS or OHS, choose the 32-bit version, which in our example would be:

```
/lib/libstdc++.so.6
```

Add the Export LD_PRELOAD Command

Open the envvars** file, which is located in the `/bin` directory of your IHS or OHS installation. Add the following line, using the path to the `libstdc++.so.6` path that you determined in the previous step.

```
export LD_PRELOAD=<path-to-libstdc++.so.6>
```

This will cause `libstdc++.so.6` to be loaded before `libstdc++.so.5`, and the conflict will be averted.

Upgrade the Apache Agent

To upgrade the Apache Agent, back up the current agent install directory elsewhere. Then download and install the new agent into the same directory as the old one was.

Then follow these steps to upgrade the agent:

1. Kill the `proxy` process.
   If you do not know the PID of the `proxy` process you can get it by running this command:

   ```
   ps -aux | grep "proxy"
   ```

   Then kill the process as follows:

   ```
   kill <PID>
   ```

2. Shut down the web server.
3. Move/rename the agent install directory, `/opt/appdynamics-sdk-native` to make a backup.
4. Download the new version of the agent and extract it into `/opt/appdynamics-sdk-native`.
5. Change directory to the agent install directory:

   ```
   cd <agent_install_directory>
   ```

6. Run the installer:

   ```
   ./install.sh
   ```

   It is not necessary to change the configuration when you use the same directory location as the old agent installation.

7. Restart the web server.

Uninstall the Apache Agent

1. Remove the include statement for the AppDynamics Apache Agent from the HTTP server configuration. For example in httpd.conf–on Ubuntu and Debian, this file is called apache2.conf—remove this line:

```
Include conf/appdynamics_agent.conf
```

2. Restart the web server.
3. Optionally remove the following files:
    - libmod_appdynamics.so–for Apache 2.4.x—or libmod_appdynamics22.so—for Apache 2.2.x—from the web server modules directory, if your administrator put it there
    - appdynamics_agent.conf or similar from the web server configuration directory
4. Kill the proxy process.
5. Uninstall the agent:

```
<agent_install_directory>/install.sh -u
```

6. Optionally remove any remaining AppDynamics files:

    - <agent_install_directory>
    - /tmp/appd-sdk

Apache Agent Logging

Related pages:

- Dynamic Language Agent Proxy

The Apache Agent generates these two logs:

- Agent log
- Proxy log

Agent Log

The agent log contains information about the transactions that the agent processes and sends to the proxy.

The log is located at `$<apache_agent_install>/logs/agent.log`. By default this is `/opt/appdynamics-sdk-native/logs/agent.log`.

The default pattern for agent log naming is the following:

- `agent.log`: the current log
- `agent.log.1`: most recent log
- `agent.log.2`: second most recent log
- `agent.log.3`: third most recent log
- `agent.log.4`: fourth most recent log
- `agent.log.5`: fifth recent log.

The agent creates and rotates a maximum of six log files. Maximum log size is 20 MB, which gives you a maximum of the most recent 120 MB of log data at one time.

Proxy Log

The proxy log contains information about the transactions between the proxy and the Controller.

The proxy log consists of a file named `proxyCore.log` located at `$<apache_agent_install>/logs/proxyCore_$date.log` and other files named `agent.<datestamp>.log` located at `$<apache_agent_install>/logs/<node_name>/agent.<datestamp>.log.`

See Dynamic Agent Proxy Logging.

Filter Sensitive Data for Apache Agent

Related pages:

- Install the Machine Agent

Add Sensitive Data Filter for Cookies

You can use sensitive Cookie filters to configure the agent to obfuscate sensitive information from the URLs in transaction snapshot details.

1. Edit the Apache Agent configuration file: `<path_to_webserver_dir>/conf/appdynamics_agent.conf`
2. Add sensitive cookie filter element as directives:
    - `AppDynamicsMaskCookie ON`
    - `AppDynamicsCookieMatchPattern pattern`
        - `AppDynamicsMaskCookie:` Specify if filtering is enabled or not. Set it `ON` for cookie filtering to be enabled. Default value is `OFF.`
        - `AppDynamicsCookieMatchPattern:` Specify the pattern that when matched, filters value of that cookie.
3. Enabling AppDynamicsMaskCookie masks values of all the cookies associated with the given request. To filter selective cookies, set the following:
    - Add `AppDynamicsCookieMatchPattern` and provide the full name of the cookie whose value needs to be masked as `AppDynamicsCookieMatchPattern <pattern>`.

        For example:

```
AppDynamicsCookieMatchPattern PHPSESSID
```

        A substring of a cookie name does not mask any value in the transaction snapshot. Ensure that you enter the Full name of the cookie for this directive.

    - For masking multiple cookies values simultaneously, provide names of all those cookies separated by '|' as a single string:

        `AppDynamicsCookieMatchPattern <pattern1|pattern2|pattern3>`

        For example:

```
AppDynamicsCookieMatchPattern PHPSESSID|X-CSRF-TOKEN|cookiekey
```

        If '|' is present in the cookie name itself, the agent cannot mask those cookies as '|' is used as a name separator in `AppDynamicsCookieMatchPattern` directive.

Add Sensitive Data Filter for the SM_USER

You can use SM_USER filter to configure the agent to obfuscate sensitive information from the URLs in transaction snapshot details.

1. Edit the Apache Agent configuration file: `<path_to_webserver_dir>/conf/appdynamics_agent.conf.`
2. Add sensitive SM_USER filter element as directive:

    - `AppDynamicsMaskSmUser ON`
    - `AppDynamicsMaskSmUser:` Specify if filtering is enabled or not. Set it `ON` for sm_user filtering to be enabled. Default value is `OFF`.

    Enabling this masks the values of all the SM_USER associated with the given request.

Filter Certain URL Segments or Query Parameters

By default, the AppDynamics Apache Agent sends transaction data to the Controller that your organization may classify as privileged information. Although such data is useful for diagnosis and troubleshooting, security considerations may require you to filter certain sensitive information from view in the Controller. You can use `Sensitive URL` filters to exclude sensitive information from a URL in snapshot details.

Add a Sensitive URL Filter

1. Edit the `appdynamics_agent.conf` configuration file in the path: `<path_to_webserver_dir>/conf/appdynamics_agent.conf`.
2. In the `appdynamics_agent.conf` file, configure the following settings:

    - `AppDynamicsDelimiter`: Specify the character that you want to use as URL segment endpoints. The agent splices the URL at each delimiter instance to create the segments. For HTTP, use the forward slash character "/". For the forward slash, the agent does not split on the slashes immediately following the protocol. For example, "https://myapp.example.com/" constitutes a single segment. By default, the delimiter is "/" but this is REQUIRED for successful filtering.

        For example:

```
AppDynamicsDelimiter /
```

Note that #' cannot be used as a delimiter as the configuration file cannot process it.

- `AppDynamicsSegment`: Specify a comma-separated list to indicate the segments that you want the agent to filter. Segment numbering starts from 1. If you specify 0 or negative values, the agent fails to redact the segments. This attribute is REQUIRED.

  For example:

```
AppDynamicsSegment 2,3
```

- `AppDynamicsMatchfilter`: The type of filter to be used to match the URL amongst the following: `NOT_EMPTY|EQUALS|STARTSWITH|ENDSWITH|CONTAINS|REGEX`. Default is `NOT_EMPTY,` but REQUIRED.

  For using this correctly, query parameters should not be considered for match-filtering. With an example of the call "https://myapp.example.com/sensitive/data?first_name=abc&last_name=xyz", to specify match-filter as STARTSWITH, it matches a specified string starting with the hostname “myapp.example.com” in this case. Similarly for ENDSWITH, it will correspond to the last segment leaving out the query parameters, “data” in this case, as query parameters are never reported in the snapshots.

  For example:

```
AppDynamicsMatchfilter CONTAINS
```

- `AppDynamicsMatchpattern`: Specify the string that you want to be filtered with the match-filter. This attribute is REQUIRED.

  For example:

```
AppDynamicsMatchpattern one
```

For example, the following configuration splits the URL on the "/" character and masks the third and fifth segments of the URL. In this case, the segmentation and obfuscation apply only to URLs containing "myapp":

```
AppDynamicsDelimiter /
AppDynamicsSegment 3,5
AppDynamicsMatchfilter CONTAINS
AppDynamicsMatchpattern myapp
```

The exit call to "https://myapp.example.com/customer/customerid/account/accountid/data?first_name=abc&last_name=xyz" breaks down to six segments: " https://myapp.example.com", "customer", "customerid", “account”, “accountid” and "data?first_name=abc&last_name=xyz". The Controller shows the masked values of the URL: “ /customer/*****/account/*****/data” in the snapshot details. "https://myapp.example.com" corresponds to segment number 1 and so on.

As the query parameters are never sent to the controller, so they are not filtered. In the transaction snapshots, the URLs are sent by default without the query parameters but now after masking the corresponding URL segments.

Filter Multiple URLs

Due to the limitations of the Apache configurations, if you want to filter multiple URLs separately, the arguments need to be written with '|' separated as described below:

```
AppDynamicsDelimiter /|/
AppDynamicsSegment 1,2,3|1,4
AppDynamicsMatchfilter CONTAINS|ENDSWITH
AppDynamicsMatchpattern One|.php
```

Each ‘|’ separated values correspond to an additional URL filter added. For ‘n’ number of separate URL filters, you need to have ‘n’ different ‘I’ configurations correspondingly. These filters behave independently on the URLs and will filter based on the configurations specified for each filter.

You must define all the configuration settings, though the configurations assume the default values. If you miss defining a particular setting in case of multiple filtering, the filtering fails.

CCPP SDK

Related pages:

- SELinux Installation Issues

This page provides an overview of the C/C++ SDK for C/C++ applications in AppDynamics.

The SDK Agent provides transaction/backend reporting, automatic tier mapping, automatic dynamic baselining, health rules, data collectors, and transaction snapshots. Altogether, these tools provide visibility into application load and response times, as well as any custom metrics you define.

To monitor the C and C++ applications, add some API calls to the source code. Once running, the SDK registers Business Transactions with the Controller. You can then see your application flow map monitor performance in the Controller.

CCPP SDK Supported Environments

C/C++ Supported Platforms

Operating Systems

- Any Linux distribution based on glibc >= 2.5 and the x86 32-bit or x86 64-bit architecture
- Windows Server >= 2012 R2
    - Visual Studio >= 2015

To develop with the C++ SDK, you need Visual Studio >= 2015 . However, the SAP ABAP Agent on Windows leverages the same SDK and does not require development work.

You must install the Visual Studio C++ Redistributable; the full development environment is not required.

Use the CCPP SDK

On this page:

Related pages:

- C/C++ SDK Reference
- Overview of Application Monitoring
- Business Transactions

This page provides an overview of how to use the C/C++ SDK.

The SDK provides routines for creating/managing business transactions, transaction snapshots, backends, exit points, and custom metrics.

To instrument your applications with the C/C++ SDK, obtain the most current version of the C/C++ SDK. See C/C++ SDK.

Add the AppDynamics Header File to the Application

After downloading the SDK, you are ready to add AppDynamics instrumentation to your C/C++ application. The first step is to include the AppDynamics header file at the top of your `main` function file:

```
#include <path_to_SDK>/sdk_lib/appdynamics.h
```

Initialize the Controller Configuration

Controller information settings permit the agent to connect to the Controller. Some settings are required for all applications, while others are needed only for certain types of application environments. For example, if the agent needs to connect to the Controller via an internal proxy in your network, you need to set up the connection settings for the proxy. See C/C++ SDK Reference for a list of the settings and information about which ones are required.

In your application, assign values to the required settings.

For example, to set the Controller connection information:

```
const char APP_NAME[] = "SampleC";
const char TIER_NAME[] = "SampleCTier1";
const char NODE_NAME[] = "SampleCNode1";
const char CONTROLLER_HOST[] = "controller.somehost.com";
const int CONTROLLER_PORT = 8080;
const char CONTROLLER_ACCOUNT[] = "customer1";
const char CONTROLLER_ACCESS_KEY[] = "MyAccessKey";
const int CONTROLLER_USE_SSL = 0;
```

To create the default application context, declare an `appd_config struct` with the Controller settings.

For example:

```
struct appd_config* cfg = appd_config_init(); // appd_config_init() resets the configuration object and pass
back an handle/pointer
appd_config_set_app_name(cfg, APP_NAME);
appd_config_set_tier_name(cfg, TIER_NAME);
appd_config_set_node_name(cfg, NODE_NAME);
appd_config_set_controller_host(cfg, CONTROLLER_HOST);
appd_config_set_controller_port(cfg, CONTROLLER_PORT);
appd_config_set_controller_account(cfg, CONTROLLER_ACCOUNT);
appd_config_set_controller_access_key(cfg, CONTROLLER_ACCESS_KEY);
appd_config_set_controller_use_ssl(cfg, CONTROLLER_USE_SSL);
```

Initialize the SDK

In your `main` function, call these initialization functions:

1. Initialize the `config` structure as shown in the previous section.
2. Initialize the SDK by passing the configuration structure to `appd_sdk_init()`.

If `appd_sdk_init()` returns zero, the SDK is initialized successfully. Non-zero indicates failure because the SDK could not reach the Controller.

The following example illustrates how to initialize the SDK:

```
int initRC = appd_sdk_init(cfg);

if (initRC) {
      std::cerr <<  "Error: sdk init: " << initRC << std::endl;
      return -1;
}
```

All SDK calls including `appd_sdk_init()` and `fork()` should only occur in the forked process, never in the parent process. Each child call should behave independently and cannot share handles between the process and other SDK-instrumented child processes.

Create Alternate Application Contexts

The SDK presumes a default application context that has no name associated with it.

To create the default application context, use the `appd_config_init()` command which returns a pointer to the default configuration structure. Then, use setter commands to modify the fields within the structure. Once all relevant fields are set, use the `appd_sdk_init()` call to initialize the SDK with the default application context.

All other application contexts are created with an associated name and a collection of separate setter methods.

To create an alternate application context, pass the `appd_context_config_init()` command the name of your alternate application context. This will return a pointer to the alternate application context configuration structure. Then, use the `_context_` setter commands to modify the fields within the structure. Once all relevant fields are set, use the `appd_sdk_add_app_context()` call to initialize the SDK, passing it the alternate application context structure pointer.

```
// create the alternate application context
appd_context_config* cfg2 = appd_context_config_init("My_Context");
...

// add additional setter commands to initialize the alternate application context
appd_context_config_set_controller_account(cfg2, ac.second.ac_account_get().c_str());
appd_context_config_set_controller_access_key(cfg2, ac.second.ac_access_key_get().c_str());
appd_context_config_set_controller_host(cfg2, ac.second.ac_host_get().c_str());
appd_context_config_set_controller_port(cfg, ac.second.ac_port_get());
appd_context_config_set_controller_use_ssl(cfg2, ac.second.ac_use_ssl_get() ? 1 : 0);
appd_context_config_set_app_name(cfg2, ac.second.ac_app_get().c_str());
appd_context_config_set_tier_name(cfg2, ac.second.ac_tier_get().c_str());
appd_context_config_set_node_name(cfg2, ac.second.ac_node_get().c_str());

// initialize the alternate application context
appd_sdk_add_app_context(cfg2);
```

Create Business Transactions

Define a Business Transaction by enclosing the code that constitutes the request that you want to monitor between `appd_bt_begin()` and `appd_bt_end()` calls. `appd_bt_begin()` returns a handle to use in subsequent routines that affect that business transaction.

If you are creating a business transaction that correlates with an upstream business transaction, pass the correlation header of the upstream transaction so the new transaction that you are creating can correlate with it. See `appd_exitcall_get_correlation_header()` in C/C++ SDK Reference. If the transaction does not need to correlate with another transaction, pass `NULL` for the correlation header parameter.

You can optionally store the business transaction handle in the global handle registry with a guid for easy retrieval later using `appd_bt_store()`. Retrieve the handle from the global handle registry using `appd_bt_get()`.

The following example demonstrates setting a business transaction:

```
// start the "Checkout" transaction
appd_bt_handle btHandle = appd_bt_begin("Checkout", NULL);

// Optionally store the handle in the global registry
appd_bt_store(btHandle, my_bt_guid);
...

// Retrieve a stored handle from the global registry
appd_bt_handle myBtHandle = appd_bt_get(my_bt_guid);
...

// end the transaction
appd_bt_end(btHandle);
```

Between starting and ending the transaction, you can perform operations such as adding errors to the business transaction, defining the transaction snapshot attributes, adding backends and exit calls, and so on.

When the business transaction ends, via a call to `appd_bt_end()`, the agent stops reporting metrics for the business transaction.

Create Business Transactions With an Alternate Application Context

You can create a business transaction with an alternate application context using `appd_bt_begin_with_app_context()` and `appd_bt_end()` calls. In addition to the parameters passed to `appd_bt_begin()`, the `appd_bt_begin_with_app_context()` also takes an application context name string, as defined by the call to `appd_context_config_init()`.

`appd_bt_begin_with_app_context()` returns a handle to use in subsequent routines that affect that business transaction.

The following example demonstrates setting a business transaction with an alternate application context:

```
// create the alternate application context
appd_context_config* cfg2 = appd_context_config_init("My_Context");
...


// add additional setter commands to initialize the alternate application context
appd_context_config_set_controller_account(cfg2, ac.second.ac_account_get().c_str());
appd_context_config_set_controller_access_key(cfg2, ac.second.ac_access_key_get().c_str());
appd_context_config_set_controller_host(cfg2, ac.second.ac_host_get().c_str());
appd_context_config_set_controller_port(cfg, ac.second.ac_port_get());
appd_context_config_set_controller_use_ssl(cfg2, ac.second.ac_use_ssl_get() ? 1 : 0);
appd_context_config_set_app_name(cfg2, ac.second.ac_app_get().c_str());
appd_context_config_set_tier_name(cfg2, ac.second.ac_tier_get().c_str());
appd_context_config_set_node_name(cfg2, ac.second.ac_node_get().c_str());

// initialize the alternate application context
appd_sdk_add_app_context(cfg2);


// start the "Checkout" transaction with the alternate application context
appd_bt_handle btHandle = appd_bt_begin_with_app_context("My_Context", "Checkout", NULL);

// end the transaction
appd_bt_end(btHandle);
```

When the business transaction ends, via a call to `appd_bt_end()`, the agent stops reporting metrics for the business transaction. Between starting and ending the transaction, you can perform operations such as adding errors to the business transaction, defining the transaction snapshot attributes, adding backends and exit calls, and so on.

Add Business Transaction Errors

Use `appd_bt_add_error()` to report business transaction errors. If you set this function parameter, `markBTAsError` the transaction is reported as an error transaction when the error occurs. For the function `{{appd_bt_add_error,}},` the Controller database truncates log messages that exceed 5,000 characters.

The SDK provides an `enum` to classify the error level as `APPD_LEVEL_NOTICE`, `APPD_LEVEL_WARNING`, or `APPD_LEVEL_ERROR`.

Manage Business Transaction Snapshots

When the agent is monitoring a business transaction, it automatically creates transaction snapshots, which describe instances of the business transaction at certain points in time. Transaction snapshots are extremely useful for troubleshooting poor performance because they contain a lot of detail.

You do not have to modify anything to create these snapshots, other than create the business transaction, but you can add calls to:

- Find out if a snapshot is being taken
- Provide additional data in a snapshot
- Set the URL for a snapshot

Determine if the Agent is Taking a Snapshot Now

The agent does not constantly collect screenshot due to high cost. By default, it collects a snapshot every ten minutes, but this schedule is configurable. See Configure Snapshot Periodic Collection Frequency.

You can determine if a snapshot is happening using `appd_bt_is_snapshotting()`, which returns non-zero if the agent is collecting a snapshot. The main reason to do this is to avoid the wasted overhead for collecting user data for a snapshot or setting the snapshot URL if no snapshot is currently being collected.

Add Business Transaction User Data

You can optionally add data to transaction snapshots. For example, you might want to know which users are getting a lot of errors, from which regions users are experiencing slow response times or which methods are slow. In the Controller UI, the data appears in the USER DATA tab of the transaction snapshot.

If a snapshot is occurring, use `appd_bt_add_user_data()` passing a key and value for the data that you want the snapshot to collect. For the function `{{appd_bt_add_user_data,}}`, the Controller database truncates log messages that exceed 5,000 characters.

Add Snapshot URL

A snapshot URL allows Controller users to share a snapshot with others. You can set a URL for the current snapshot using `appd_bt_set_url()`.

Set Snapshot Example

```
void setSnapshotAttributes(appd_bt_handle btHandle, int minute, int halfsec)
{
    // do this only if the agent is collecting a snapshot
    if (appd_bt_is_snapshotting(btHandle))
    {
        char nameBuf[30];
        char valueBuf[30];
        // add custom data to the snapshot
        snprintf(nameBuf, sizeof(nameBuf), "BT:%p\n", btHandle);
        snprintf(valueBuf, sizeof(valueBuf), "Minute:%d Second:%d\n", minute, halfsec/2);
        appd_bt_add_user_data(btHandle, nameBuf, valueBuf);

        static int snapCount = 0;
        int switchVal = snapCount % 4;

        // report errors, but only ERROR_LEVEL errors are marked as error transactions
        if (switchVal)
        {
            appd_error_level errorLevel;
            bool markBtAsError;
            switch (switchVal)
            {
            case 1:
                errorLevel = APPD_LEVEL_NOTICE;
                markBtAsError = false;
                snprintf(nameBuf, sizeof(nameBuf), "NOTICE BT:%p M:%d S:%d\n", btHandle, minute, halfsec/2);
                break;
            case 2:
                errorLevel = APPD_LEVEL_WARNING;
                markBtAsError = false;
                snprintf(nameBuf, sizeof(nameBuf), "WARNING BT:%p M:%d S:%d\n", btHandle, minute, halfsec/2);
                break;
            case 3:
                errorLevel = APPD_LEVEL_ERROR;
                markBtAsError = true;
                snprintf(nameBuf, sizeof(nameBuf), "ERROR BT:%p M:%d S:%d\n", btHandle, minute, halfsec/2);
                break;
            }
            appd_bt_add_error(btHandle, errorLevel, nameBuf, markBtAsError, markbtaserror);
        }
        snapCount++;
        // set the snapshot url
        snprintf(nameBuf, sizeof(nameBuf), "http://bt-%p.com", btHandle);
        appd_bt_set_url(btHandle, nameBuf);
    }
}
```

Create Backends

A backend is a database or a remote service such as a message queue, HTTP service or cache service that your application uses. A backend component is not itself monitored by the application agent, but the agent monitors calls to it from instrumented servers. You need to create backends in the instrumented environment so that the agent can discover them. This involves:

- Declaring the backend
- Setting its identifying properties
- Optionally configuring how the backend is presented in the AppDynamics UI
- Adding the backend to the instrumented application

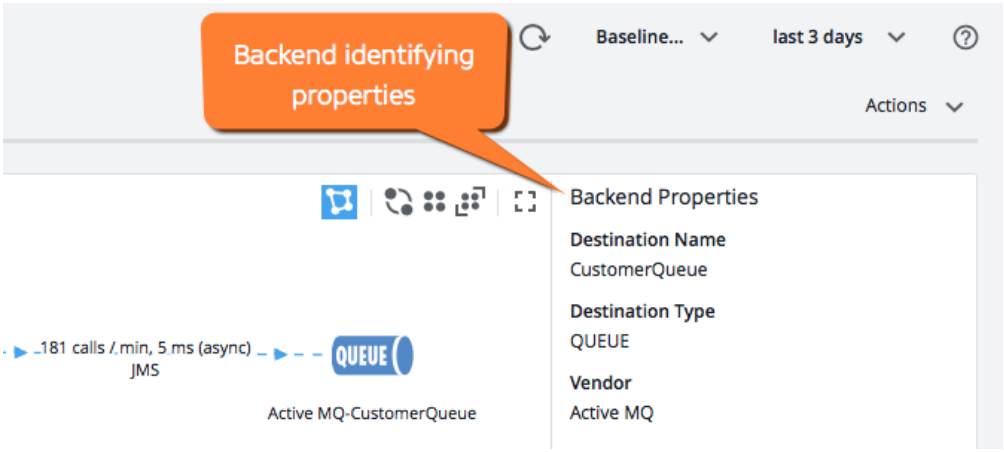
Declare the Backend

You must declare a backend using `appd_backend_declare()` before the agent can detect it. After you declare a backend, you don't need to declare it again if the backend is used by other business transactions in a single SDK instance. A backend must be of one of the supported types listed under 'Exit Call Types' in C/C++ SDK Reference.

Identify the Backend

A backend also has identifying properties, which you set using `appd_backend_set_identifying_property()`. The properties vary depending on the type of the backend and the types of information that you want to display. The Controller displays identifying properties in backend dashboards. You must set at least one identifying property for the type of any backend that you plan to add.

The following shows the backend properties in the Controller UI for an ActiveMQ:



Resolve to a Tier

By default, the Controller doesn't display a detected backend as a separate entity in flowmaps, but the agent reports its metrics as part of the downstream tier. If you want to display the backend as a separate component and not resolved to the tier, use `appd_backend_prevent_agent_resolution()`. See Resolve Remote Services to Tiers.

Add to Application

After you declare and configure the backend, add it to the application so the agent can detect it.

Backend Example

The following listing shows an example of setting up a database backend.

```
// declare a backend, only once for this SDK instance
const char backendOne[] = "first backend";
appd_backend_declare(APPD_BACKEND_HTTP,  backendOne);

// set the host property
rc = appd_backend_set_identifying_property(backendOne, "HOST", "sqs-us-west-hostname");
if (rc) {
   std:cerr << "Error: appd_backend_set_identifying_property: " << rc << ".";
   return -1;
}

// do not resolve the backend to the tier
rc = appd_backend_prevent_agent_resolution(backendOne);
if (rc) {
   std:cerr << "Error: appd_backend_prevent_agent_resolution: " << rc << ".";
   return -1;
}

// add the backend
rc = appd_backend_add(backendOne);
if (rc)
{
   std:cerr << "Error: appd_backend_add: " << rc << ".";
   return -1;
}
```

Manage Exit Calls

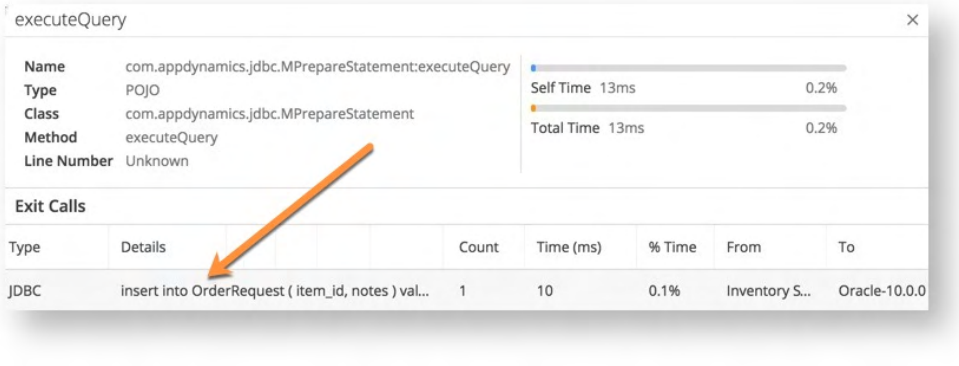
When an application makes a call to another component—a detected backend or another application server—the agent reports metrics on those calls.

Define an exit call by enclosing the code that constitutes the exit call between `appd_exitcall_begin()` and `appd_exitcall_end()` calls. `appd_exitcall_begin()` returns a handle to use in subsequent routines that affect that exit call. An exit call occurs in the context of a business transaction.

You can optionally store the exit call handle in the global handle registry with a guid for easy retrieval later with `appd_exitcall_store()`. Retrieve the handle from the global handle registry with `appd_exitcall_get()`.

Pass the business transaction handle and the destination backend to `appd_exitcall_end()`.

You can optionally add details to an exit call as any arbitrary string. The details are reported in the exit call details in the transaction snapshots in the Controller UI:



You can also add errors to an exit call using `appd_exitcall_add_error()`. Use the enum for the error levels. You can also add an error message.

Simple Exit Call Example

```
// start the exit call to backendOne
appd_exitcall_handle ecHandle = appd_exitcall_begin(btHandle, backendOne);

...

// optionally store the handle in the global registry
appd_exitcall_store(ecHandle, my_ec_guid);

...
// retrieve a stored handle from the global registry
appd_exitcall_handle myEcHandle = appd_exitcall_get(my_ec_guid);

// set the exit call details
rc = appd_exitcall_set_details(ecHandle, "backend ONE");
if (rc) {
   std:cerr << "Error: exitcall details1";
   return -1;
}

// add an error to the exit call
appd_exitcall_add_error(ecHandle, APPD_LEVEL_ERROR, "exitcall1 error!", true);

// end the exit call
appd_exitcall_end(ecHandle)
```

Correlate with Other Business Transactions

A correlation header contains the information that enables the agents to continue the flow of a business transaction across multiple tiers.

An SDK can correlate with other SDKs as well as other AppDynamics agents—such as Java, .NET or PHP—that perform automatic correlation for certain types of entry and exit points.

Correlate with an Upstream Tier

When your instrumented process receives a continuing transaction from an upstream agent that supports automatic correlation:

1. Using a third-party `http_get_header()` function, extract the header named `APPD_CORRELATION_HEADER_NAME` from the incoming HTTP payload.
2. Pass the header to `appd_bt_begin()`. For example:

```
const char* hdr = http_get_header(req, APPD_CORRELATION_HEADER_NAME);
appd_bt_handle bt = appd_bt_begin("fraud detection", hdr);
```

If the header retrieved by the `http_get_header()` function is valid, the business transaction started by the `appd_bt_begin()` call will be a continuation of the business transaction started by the upstream service.

Correlate with a Downstream Tier

The downstream agent is watching for a correlation header named `singularityheader` in the HTTP payload.

If your SDK agent is making an exit call to a downstream agent that supports automatic correlation:

1. Set the name of the correlation header using `APPD_CORRELATION_HEADER_NAME`.
2. Begin the exit call.
3. Retrieve the correlation header from the exit call using the `appd_exitcall_get_correlation_header()` function.
4. Using third-party HTTP functions, prepare an HTTP POST request.
5. Inject a header named `APPD_CORRELATION_HEADER_NAME` with the value of the correlation header retrieved in step 3 into the outgoing payload of the HTTP request.
6. Make the request. For example:

```
const char* const APPD_CORRELATION_HEADER_NAME = "singularityheader";

appd_exitcall_handle inventory = appd_exitcall_begin(bt, "inventory");
const char* hdr = appd_exitcall_get_correlation_header(inventory);
http_request req;
http_init(&req, HTTP_POST, "https: //inventory/holds/%s", sku);
http_set_header(&req, APPD_CORRELATION_HEADER_NAME, hdr);
http_perform(&req);

...
```

cURL Downstream Correlation Example

The following example uses the cURL library to correlate with a downstream transaction.

```
#include <curl/curl.h>
. . .
. . .
// get the correlation header into the response
class LibcurlURLGet
{
public:
    LibcurlURLGet(const std::string& url) : url(url), curlHandle(NULL) {}
    std::string GetResponse(const char* correlationHeader)
    {
        CURLcode res;
        Response response;
        curlHandle = curl_easy_init();
        if (curlHandle)
        {
            curl_easy_setopt(curlHandle, CURLOPT_URL, url.c_str());
            curl_easy_setopt(curlHandle, CURLOPT_WRITEFUNCTION, WriteDataCallback);
            curl_easy_setopt(curlHandle, CURLOPT_WRITEDATA, &response);
            if(correlationHeader && strlen(correlationHeader))
            {

// start new added code

                std::string singularityHeader(APPD_CORRELATION_HEADER_NAME);
                singularityHeader.append(": ");
                singularityHeader.append(correlationHeader);

// end new added code

                curl_slist* slistNewHeaders = 0;
                slistNewHeaders = curl_slist_append(slistNewHeaders, correlationHeader);
                curl_easy_setopt(curlHandle, CURLOPT_HTTPHEADER, slistNewHeaders);
```

```
            }
            res = curl_easy_perform(curlHandle);
            if(res != CURLE_OK)
            {
            }
        }
        curl_easy_cleanup(curlHandle);
        return response.GetResponseString();
    }
private:
    class Response
    {
    public:
        std::string GetResponseString()
        {
            return sResponse;
        }
        void AppendResponse(void* ptr, size_t sizeBytes)
        {
            sResponse.append((char*)ptr, sizeBytes);
        }
    private:
        std::string sResponse;
    };
    static size_t WriteDataCallback(void* ptr, size_t size, size_t nmemb, Response* response)
    {
        response->AppendResponse(ptr, size * nmemb);
        return size * nmemb;
    }
    std::string url;
    CURL* curlHandle;
};

...
// start an exit call from the current transaction
appd_exitcall_handle ecHandle1 = appd_exitcall_begin(btHandle1, tier2);

...
// before exiting get the correlation header
const char* corrHeader1 = appd_exitcall_get_correlation_header(ecHandle1);

// put the correlation in the HTTP response
std::string response = urlGet.GetResponse(corrHeader1);

...
// start the next business transaction to correlate with the previous one
appd_bt_handle btHandle = appd_bt_begin("Verify", response)
```

Create Custom Metrics

You can create custom metrics to supplement built-in metrics such as Business Transaction call count and response time. You can also create metrics for multiple application contexts.

Custom metrics are dependent on the application context; in situations with multiple contexts, you must specify the correct context name for custom metrics to apply to the associated application context. See Create Alternate Application Contexts.

> ⚠ If you create a custom metric in a node, the custom metric will appear in every other node in the tier.

For example, the default application context uses a null pointer or empty string as its name. The `appd_sdk_init` method tells the SDK to connect to the Controller and create the default application context (i.e. no name is specified).

```
struct appd_config* cfg = appd_config_init();
appd_config_set_controller_host(cfg, host.c_str());
appd_config_set_controller_port(cfg, port);
appd_config_set_controller_account(cfg, account_name.c_str());
appd_config_set_controller_access_key(cfg, access_key.c_str());
appd_config_set_controller_use_ssl(cfg, useSSL?1:0);
appd_config_set_app_name(cfg, app_name.c_str());
appd_config_set_tier_name(cfg, tier_name.c_str());
appd_config_set_node_name(cfg, node_name.c_str());
appd_sdk_init(cfg);
```

To specify an alternative application context, you must provide a name string (not a default null pointer). Use the `appd_sdk_add_appd_context` method to create an alternative application context. In the example below, we create an alternative application context called "context2." You will use that name string to create custom metrics, not the actual configuration structure.

```
const char* name = "context2";
struct appd_context_config* cfg2 = appd_context_config_init(name));
appd_context_config_set_controller_account(cfg2, ac.second.ac_account_get().c_str());
appd_context_config_set_controller_access_key(cfg2, ac.second.ac_access_key_get().c_str());
appd_context_config_set_controller_host(cfg2, ac.second.ac_host_get().c_str());
appd_context_config_set_controller_port(cfg, ac.second.ac_port_get());
appd_context_config_set_controller_use_ssl(cfg2, ac.second.ac_use_ssl_get() ? 1 : 0);
appd_context_config_set_app_name(cfg2, ac.second.ac_app_get().c_str());
appd_context_config_set_tier_name(cfg2, ac.second.ac_tier_get().c_str());
appd_context_config_set_node_name(cfg2, ac.second.ac_node_get().c_str());

appd_sdk_add_app_context(cfg2);
```

The Custom Metrics path may or may not go the same Controller; similar to business transactions in separate nodes, your program can process two business transactions from different nodes: one aligns with the default application context and the other with the alternative application context. You must specify that your method strings begin with `Custom Metrics|`, not `Custom Metric|`.

You can use the following methods to create a custom metric:

- `appd_custom_metric_add()`
- `appd_custom_metric_report()`

The `appd_custom_metric_report` creates a default application context and an alternative application context named `my_second_custom_metric`.

In the Controller, the custom metrics folder appears under the associated tier as "Custom Metrics" in the **Metric Browser** with two entities: `my_default_custom_metric` and `my_second_custom_metric`.

The following example illustrates a call with both method signatures.

```
appd_custom_metric_add("app_context", "Custom Metrics|Memory|Total Memory Usage",
APPD_TIMEROLLUP_TYPE_AVERAGE, APPD_CLUSTERROLLUP_TYPE_INDIVIDUAL,
APPD_HOLEHANDLING_TYPE_RATE_COUNTER);
```

The method passes the application context ("Total Memory Usage") and the path to the metric (**Custom Metrics** > **Memory** in the **Metric Browser**) to the Controller. The additional parameters define how the metric is processed. With the metric declared, your code can then report data to the metric with the `appd_custom_metric_report()` method, as follows:

```
appd_custom_metric_report("app_context", "Custom Metrics|Memory|Total Memory Usage", 1234);
```

See C/C++ SDK Resource Management to simplify your instrumentation on C++ applications.

Metric Handling Parameters

Like built-in metrics, the Controller applies some standard processing functions to custom metric values. These functions can define, for example, how the metric is rolled up over time. How the Controller processes a custom metric depends on the nature of the metric; for a metric that represents the state of the machine, the Controller captures the last observed value for the metric. For a call rate metric, the Controller uses an average value. The following section provides information about the parameters you can use to define metric handling by the Controller.

Time Rollup Type

The `time_rollup_type` parameter tells the Controller how to roll up values for the metric over time. There are three ways in which the Controller can roll up metrics, as follows:

- It can calculate the average of all reported values in the time period. An example of a built-in metric that uses this is the `Average Response Time` metric.
- Sum of all reported values in that minute. This operation behaves like a counter. An example metric is the `Calls per Minute` metric.
- Current is the last reported value in the minute. If no value is reported in that minute, the last reported value is used. An example of a metric that uses this would be a machine state metric, such as `Max Available (MB)` metric.

Cluster Rollup Type

The `appd_cluster_rollup_type` parameter tells the Controller how to aggregate metric values for the tier—a cluster of nodes.

- Individual: Aggregates the metric value by averaging the metric values across each node in the tier. For example, `Hardware Resources|Memory|Used %` is a built-in metric that uses the individual rollup type.
- Collective: Aggregates the metric value by adding up the metric values for all the nodes in the tier. For example, `Agent|Metric Upload|Metrics uploaded` is a built-in metric that uses the collective rollup type.

Hole Handling Type

A particular metric may not report data for a given minute. The `appd_hole_handling_type` parameter tells the Controller how to set the metric count for that time slice. The count is set to zero if the hole handling type is `REGULAR_COUNTER`, and set to one if `RATE_COUNTER`. In effect, `REGULAR_COUNTER` does not affect aggregation while `RATE_COUNTER` does.

For example, consider four time slices with the data 9, 3, 0, 12. Notice that the third time slice is zero filled, with no metric being reported for that time slice. The sum of the values is 9+3+0+12 = 24.

If the metric is `REGULAR_COUNTER`, the count for the third times slice would be zero, and therefore the overall count would be 1+1+0+1 = 3. If the metric is RATE_COUNTER the overall count would be 1+1+1+1 = 4. In the case of the `REGULAR_COUNTER`, the average would be 24/3 = 8, while for the `RATE_COUNTER` the average would be 24/4 = 6.

Built-in metrics that use the regular counter hole-handling type include `Average Response Time`, `Average CPU Used`, and `Number of Stalls`. Built-in metrics that use the rate counter hole-handling type include `BT Calls per minute` and `Errors per minute`.

Generate Call Graphs

You can use the C/C++ SDK to instrument methods so that they are reported and displayed in the call graph. You can instrument methods in one of the following ways:

- Use the `APPD_AUTO_FRAME` macro, C++ only
- Use `appd_frame_begin` and `appd_frame_end`

Instrument Methods with the `APPD_AUTO_FRAME` Macro

You can use the `APPD_AUTO_FRAME` macro to instrument applications built on C++ frameworks. The macro is equivalent to calling the `appd_frame_begin` and `appd_frame_end` methods.

1. Specify the business transaction that you want to instrument calls for, and configure the root of the call graph.

```
appd::sdk::BT bt("mybt");
APPD_AUTO_FRAME(bt);
method1(bt);
method2(bt);
```

2. Instrument the methods as shown below.

```
void method1(appd::sdk::BT& bt)
{
 APPD_AUTO_FRAME(bt);
 // Code for method1...
}
void method2(appd::sdk::BT& bt)
{
 APPD_AUTO_FRAME(bt);
 // Code for method2...
}
```

The methods must have access to the business transaction so the business transaction can be passed into the method as an argument.

Instrument Methods with the `appd_frame_begin` and `appd_frame_end` Functions

You can instrument methods for applications that are built on non-C++ frameworks. To instrument a method, you call `appd_frame_begin` at the beginning of the method, and `appd_frame_end` at the end of the method. When you do this, the duration of the method call is automatically calculated and reported in the call graph.

If your method makes an exit call, you can instrument the exit call by wrapping the exit call code between `appd_exitcall_begin` and `appd_exitcall_end`.

The code samples below show three methods that are each instrumented using `appd_frame_begin` and `appd_frame_end`, and then called in a root method, `main()`. The second sample method, `method2()`, contains an exit call.

```
void method1( )
{
 appd_frame_handle frameHandle = appd_frame_begin(btHandle, APPD_FRAME_TYPE_CPP, nullptr, APPD_FUNCTION_NAME,
__FILE__, __LINE__);
 // code for method1
 appd_frame_end(btHandle, frameHandle);
}
```

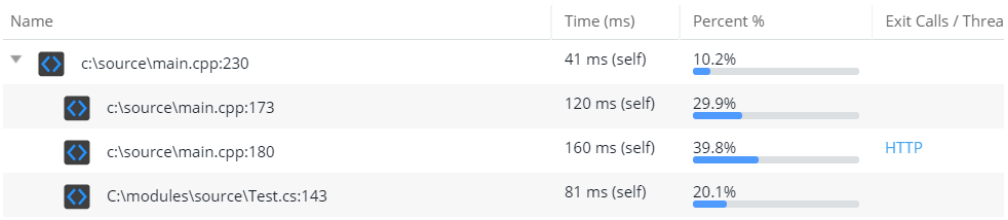
```
void method2()
{
 appd_frame_handle frameHandle = appd_frame_begin(btHandle, APPD_FRAME_TYPE_CPP, nullptr, APPD_FUNCTION_NAME,
__FILE__, __LINE__);
 // code for method2
appd_exitcall_handle exitCallHandle = appd_exitcall_begin(btHandle, BACKEND_NAME);
 // code for exit call
 appd_exitcall_end(exitCallHandle);
 appd_frame_end(btHandle, frameHandle);
}
```

```
void method3()
{
 // To illustrate instrumenting a method where this SDK cannot be used
 // method3 is a wrapper for the call of the actual method which will show up in the call graph
 appd_frame_handle frameHandle = appd_frame_begin(btHandle, APPD_FRAME_TYPE_CPP, "Test", "Compute", "C:
\\modules\\source\\Test.cs", 143);
 // call the wrapped method
 appd_frame_end(btHandle, frameHandle);
}
```

The `main()` method below is the root of the call graph. It specifies the business transaction that invokes the three methods above by passing the transaction into `appd_bt_begin` and `appd_bt_end`.

```
int main(int argc, char **argv)
{
 // initialize AppDynamics
 btHandle = appd_bt_begin(TRANSACTION_NAME, "");
 appd_frame_handle frameHandle = appd_frame_begin(btHandle, APPD_FRAME_TYPE_CPP, nullptr, APPD_FUNCTION_NAME,
__FILE__, __LINE__);
 // code for main
 method1();
 method2();
 method3();
 appd_frame_end(btHandle, frameHandle);
 appd_bt_end(btHandle);
}
```

The sample methods above are represented by the call graph shown below:



Generate Pre-populated Call Graphs (C++ Only)

The previous section describes instrumenting your C/C++ application by calling the instrumentation methods in your code. However, if you do not have the option of modifying your source code, you can generate call graphs that are populated with data from sources such as log files.

To generate a pre-populated call graph:

1. Create the root of the call graph by instantiating a `CallGraph` class.
   The `CallGraph` class takes the following parameters:

   - `bt`: The business transaction.
   - `class_name`: The name of the class that contains the root of the call graph.
   - `method_name`: The name of the method that represents the root of the call graph.
   - `file_path`: The file path to the class.
   - `line_number`: The line number of the method.
   - `time_msec`: The time taken by this frame (method) in milliseconds.
   - `frame_type`: The type of the frame (the language for the method). Currently, `APPD_FRAME_TYPE_CPP` is the only option.

   The following example demonstrates a `CallGraph` instantiation.

   ```
   appd::sdk::CallGraph callGraph(bt, "Class1", "main", "/src/file1.cc", 276, 100, APPD_FRAME_TYPE_CPP);
   ```

2. Create the call graph tree by calling `add_child` on the root and on any of the added children:

   ```
   callGraph.root() .add_child("Class2", "method1", "/src/file2.cc"), 101, 40, APPD_FRAME_TYPE_CPP) .
   add_child("Class2", "method2", "/src/file2.cc"), 523, 30, APPD_FRAME_TYPE_CPP); auto& cge1 = callGraph.
   root() .add_child("Class3", "method1", "/src/file3.cc"), 27, 30, APPD_FRAME_TYPE_CPP); cge1.add_child
   ("Class3", "method2", "/src/file3.cc"), 430, 15, APPD_FRAME_TYPE_CPP);
   ```

3. Call `add_to_snapshot`on the call graph. For this to work, the business transaction must be snapshotting.

   ```
   callGraph.add_to_snapshot();
   ```

Analytics Support for C++ SDK

Two of the existing API methods, which were previously only used for adding data to a snapshot, will now send Analytics data if Analytics is enabled for the current Business Transaction on the Controller:

- Set URL for a snapshot: the URL is set for a snapshot if one is occurring. The data should be either 7-bit ASCII or UTF-8. You can call this function when a snapshot does not occur. When the given Business Transaction is not snapshotting, this function returns immediately. However, if extracting the data to pass to this function is expensive, you can use `appd_bt_is_snapshotting` to check if the Business Transaction is snapshotting before extracting the data and calling this function.

  ```
  void appd_bt_set_url(appd_bt_handle bt, const char* url);
  ```

  - param bt: The business transaction to add the user data to, if it is taking a snapshot.
  - param url: The value of the URL for the snapshot as 7-bit ASCII or UTF-8.
- Add user data to a snapshot: the data should be either 7-bit ASCII or UTF-8. You can call this function when a snapshot does not occur or if Analytics is not enabled. If the data is only for snapshotting and extracting the data to pass to this function is expensive, you can use `appd_bt_is_snapshotting` to check if the Business Transaction is snapshotting before extracting the data, and calling this function.

  ```
  void appd_bt_add_user_data(appd_bt_handle bt, const char* key, const char* value);
  ```

- param bt: The business transaction to add the user data to, if it is taking a snapshot.
- param key: The name of the user data to add to the snapshot as 7-bit ASCII or UTF-8.
- param value: The value of the user data to add to the snapshot as 7-bit ASCII or UTF-8.

These API methods existed previously, but their behavior did not extend to reporting data to Analytics. To correctly connect to Analytics, initialize using the following configuration APIs:

```
void appd_config_set_analytics_host(struct appd_config* cfg, const char* host);
```

- param host: The host for the Analytics Agent, which defaults to `localhost`.

```
void appd_config_set_analytics_port(struct appd_config* cfg, const unsigned short port);
```

- param port: The port that the Analytics Agent listens on, which defaults to 9090.

Terminate the Agent

Just before the application exits, terminate the SDK:

```
appd_sdk_term();
```

Enable SSL for CCPP SDK

Related pages:

- Use the C/C++ SDK

This page explains how to configure the C/C++ Agent to connect to the Controller using SSL.

To enable the SSL on the C/C++ Agent:

1. Open your application to the file where you configure the Controller settings.
2. Set `const int CONTROLLER_USE_SSL = 1;`
3. In the `appd_config` struct, include the following method:

```
appd_config_set_controller_use_ssl(cfg, CONTROLLER_USE_SSL);
```

If you are using a self-signed certificate, you must do the following:

1. Get the Controller certificate by running the following command:

```
openssl s_client -connect <hostname>:<port> -showcerts < /dev/null | openssl x509 > cert.pem
```

2. Update the `certificate_file` Controller setting to point to absolute path to certificate.

```
appd_config_set_controller_certificate_file(cfg, <Path to Cert File>);
```

Check the Certificate `CommonName`

The certificate `CommonName` must match the Controller hostname. If they do not match, you must re-configure the certificate on the Controller.

You can check the `CommonName` by running the following command:

```
openssl s_client -connect <CONTROLLER_HOST>:<CONTROLLER_PORT> -showcerts
```

The `CommonName` appears above the `BEGIN CERTIFICATE` line.

CCPP SDK Reference

On this page:

- appd_exitcall_set_details
- appd_exitcall_store
- appd_frame_begin
- appd_frame_end

Related pages:

- Use the C/C++ SDK
- C++ SDK RAII (Resource Management Is Initialization)

This page describes the functions, structures, and environment variables defined in the C/C++ SDK to instrument C/C++ applications.

The API includes functions to create business transactions, transaction backends/exit points, and define custom metrics.

Basic Types

The C/C++ SDK defines the following opaque types:

- `appd_bt_handle`: A handle to an active business transaction
- `appd_exitcall_handle`: A handle to an active exit call
- `struct appd_config*`: A pointer to a configuration object
- `struct appd_context_config*`: A pointer to a context configuration object

appd_config_init

Initializes an empty configuration structure. Call this function before before calling `appd_sdk_init()`.

Format

```
APPD_API struct appd_config * appd_config_init ()
```

Parameters

- `cfg`: AppDynamics configuration object.

appd_config_set_analytics_enabled

Set an enable/disable flag for Analytics Agent reporting. When disabled, Analytics-related logging messages are suppressed.

Format

```
APPD_API void appd_config_set_analytics_enabled(struct appd_config* cfg, const unsigned short enable)
```

Parameters

- `cfg`: AppDynamics configuration structure for the C/C++ SDK.
- `enable`: The enable/disable flag for the Analytics Agent. If the value is non-zero, the agent is enabled. If the value is zero (default setting), then the agent is disabled.

appd_config_set_app_name

Set business application name.

Format

```
appd_config_set_app_name(cfg, APP_NAME);
```

Parameters

- cfg: AppDynamics configuration object.
- `APP_NAME`: Name of the application.

appd_config_set_tier_name

Set tier name.

Format

```
appd_config_set_tier_name(cfg, TIER_NAME);
```

Parameters

- cfg: AppDynamics configuration object.
- `TIER_NAME`: Name of the tier.

appd_config_set_node_name

Set node name.

Format

```
appd_config_set_node_name(cfg, NODE_NAME);
```

Parameters

- `cfg` – AppDynamics configuration object.
- `NODE_NAME` – Name of the node.

appd_config_set_controller_host

Set hostname of the Controller.

Format

```
appd_config_set_controller_host(cfg, CONTROLLER_HOST);
```

Parameters

- `cfg`: AppDynamics configuration object.
- `CONTROLLER_HOST`: Name of the Controller host.

appd_config_set_controller_port

Set the port number on which the Controller is listening.

Format

```
appd_config_set_controller_port(cfg, CONTROLLER_PORT);
```

Parameters

- `cfg`: AppDynamics configuration object.
- `CONTROLLER_PORT`: Port number that the Controller is listening on.

appd_config_set_controller_account

Set the account name for connecting to the Controller.

Format

```
appd_config_set_controller_account(cfg, CONTROLLER_ACCOUNT);
```

Parameters

- `cfg`: AppDynamics configuration object.
- `CONTROLLER_ACCOUNT`: Account name that the Controller is connected to.

appd_config_set_controller_access_key

Set the access key for connecting to the Controller.

Format

```
appd_config_set_controller_access_key(cfg, CONTROLLER_ACCESS_KEY);
```

Parameters

- `cfg`: AppDynamics configuration object.
- `CONTROLLER_ACCESS_KEY:` Access key for connecting to the Controller.

appd_config_set_controller_use_ssl

Specify whether SSL should be used to communicate with the Controller.

Format

```
appd_config_set_controller_use_ssl(cfg, CONTROLLER_USE_SSL);
```

Parameters

- `cfg`: AppDynamics configuration object.
- `CONTROLLER_USE_SSL:` Whether to use SSL to connect with the Controller. Set to a non-zero integer for true. Set to the integer zero for false. Note that SaaS Controllers require this to be set to non-zero.

appd_config_set_controller_http_proxy_host

Optional. Set the hostname of the HTTP proxy if using an HTTP proxy to talk to Controller.

Format

```
appd_config_set_controller_http_proxy_host(cfg, CONTROLLER_HTTP_PROXY_HOST);
```

Parameters

- `cfg`: AppDynamics configuration object.
- `CONTROLLER_HTTP_PROXY_HOST`: Hostname of the HTTP proxy if using an HTTP proxy to talk to Controller.

appd_config_set_controller_http_proxy_port

Optional. Set the port number of the HTTP proxy. Default is 80.

Format

```
appd_config_set_controller_http_proxy_port(cfg, CONTROLLER_HTTP_PROXY_PORT);
```

Parameters

- `cfg:` AppDynamics configuration object.
- `CONTROLLER_HTTP_PROXY_PORT`: Port name of the HTTP proxy. Default is 80.

appd_config_set_controller_http_proxy_username

Optional. Set the username to connect to the HTTP proxy with.

Format

```
appd_config_set_controller_http_proxy_username(cfg, CONTROLLER_HTTP_PROXY_USERNAME);
```

Parameters

- `cfg`: AppDynamics configuration object.
- `CONTROLLER_HTTP_PROXY_USERNAME`: Username to connect to the HTTP proxy with**.**

appd_config_set_controller_http_proxy_password

Optional. Set the password to connect to the HTTP proxy with.

Format

```
appd_config_set_controller_http_proxy_password(cfg, CONTROLLER_HTTP_PROXY_PASSWORD);
```

Parameters

- `cfg`: AppDynamics configuration object.
- `CONTROLLER_HTTP_PROXY_PASSWORD`: Password to connect to the HTTP proxy with.

appd_config_set_controller_http_proxy_password_file

Optional. Set the file containing password to connect to the HTTP proxy with.

Format

```
appd_config_set_controller_http_proxy_password_file(cfg, CONTROLLER_HTTP_PROXY_PASSWORD_FILE);
```

Parameters

- `cfg`: AppDynamics configuration object.
- `CONTROLLER_HTTP_PROXY_PASSWORD_FILE`: File containing password to connect to the HTTP proxy with.

appd_config_set_controller_certificate_file

Optional. Set the the CA certificate file name. Set this if you choose to use your own certificate file.

Format

```
appd_config_set_controller_certificate_file(cfg, CONTROLLER_HTTP_CERTIFICATE_FILE);
```

Parameters

- `cfg`: AppDynamics configuration object.
- `CONTROLLER_CERTIFICATE_FILE`: Name of the certificate file. Defaults to the included `ca-bundle.crt` file.

appd_config_set_controller_certificate_dir

Optional. Set the full path to the CA certificate file. Set this if you have multiple certificate files.

Format

```
appd_config_set_controller_certificate_dir(cfg, CONTROLLER_HTTP_CERTIFICATE_DIR);
```

Parameters

- `cfg`: AppDynamics configuration object.
- `CONTROLLER_CERTIFICATE_DIR`: Full path to the certificate files.

appd_config_set_flush_metrics_on_shutdown

Specify whether to collect metrics in the final minute before SDK shutdown—via `appd_sdk_term`. By default, metrics reported in the minute before shutdown are lost. If you enable metric flushing, then you can retain the metrics reported in the final minute before shutdown.

Format

```
APPD_API void appd_config_set_flush_metrics_on_shutdown(struct appd_config* cfg, int enable);
```

Parameters

- `cfg`: AppDynamics configuration object.
- `int`: Set to a non-zero integer to enable flushing metrics to the Controller. Default is 0.

By setting this flag, the `appd_sdk_term()` is blocked until the underpinning agent logic has uploaded the metric data. By not using this flag, any metrics collected since the last Controller update is discarded. Depending on the load on the Controller, flushing the metrics can take anywhere between 60 and 120 seconds on average or longer. Do not use this flag if you need the termination/tear-down logic to complete immediately.

appd_config_set_logging_min_level

Set the minimum level of logging that is allowed. If `APPD_LOG_LEVEL_TRACE`, all log messages are allowed. If `APPD_LOG_LEVEL_FATAL`, only the most severe errors are logged. The default is `APPD_LOG_LEVEL_INFO`.

Format

```
appd_config_set_logging_min_level(cfg, LOGGING_MIN_LEVEL);
```

Parameters

- `cfg`: AppDynamics configuration object.
- `LOGGING_MIN_LEVEL`: The minimum level of logging that is allowed.
- `APPD_LOG_LEVEL_TRACE`
- `APPD_LOG_LEVEL_DEBUG`
- `APPD_LOG_LEVEL_INFO`
- `APPD_LOG_LEVEL_WARN`
- `APPD_LOG_LEVEL_ERROR`
- `APPD_LOG_LEVEL_FATAL`

appd_config_set_logging_log_dir

Set the directory to log to. The process running the SDK must have permissions to create this directory, if it does not already exist, to list the files within it, and to write to the files within it.

Format

```
appd_config_set_logging_log_dir(cfg, LOGGING_LOG_DIR);
```

Parameters

- `cfg`: AppDynamics configuration object.
- `LOGGING_LOG_DIR:` The directory where you want to store log files. Defaults to `/tmp/appd`.

appd_config_set_logging_max_num_files

Set the maximum number of log files allowed per tenant. Log files are rotated when they reach this number.

Format

```
appd_config_set_logging_max_num_files(cfg, LOGGING_MAX_NUM_FILES);
```

Parameters

- `cfg`: AppDynamics configuration object.
- `LOGGING_MAX_NUM_FILES`: The maximum number of log files allowed per tenant. Default is 10.

appd_config_set_logging_max_file_size_bytes

Set the maximum size of an individual log file, in bytes. Log files are rotated when they reach this size.

Format

```
appd_config_set_logging_max_file_size_bytes(cfg, LOGGING_MAX_FILE_SIZE_BYTES);
```

Parameters

- `cfg`: AppDynamics configuration object.
- `LOGGING_MAX_FILE_SIZE_BYTES:` The maximum size of an individual log file, in bytes. Default is `5 * 1024 * 1024`.

appd_config_set_init_timeout_ms

Set the number of milliseconds you want `appd_sdk_init` (an asynchronous action) to wait until it has received controller configuration and is ready to capture business transactions. Set this if you want to capture short-running business transactions that occur at application startup and you don't mind the delay of waiting for the Controller to send the configuration.

Format

```
appd_config_set_init_timeout_ms(cfg, INIT_TIMEOUT_MS);
```

Parameters

- `cfg`: AppDynamics configuration object.
- `INIT_TIMEOUT_MS`: The number of milliseconds you want `appd_sdk_init` (an asynchronous action) to wait until it has received controller configuration and is ready to capture business transactions.
    - X: Wait up to X milliseconds for Controller configuration.
    - 0: Do not Wait for Controller configuration.
    - -1: Wait indefinitely until Controller configuration is received by agent
    - Default is 0.

appd_config_getenv

Set the environment variable for configuring the SDK. Environment variables are not read by default. You must call this function to configure the SDK via environment variables.

For a list of available environment variables, see Environment Variables.

Format

```
APPD_API void appd_config_getenv(struct appd_config * cfg, const char * prefix);
```

Parameters

- `cfg`: AppDynamics configuration object.
- `prefix:` The prefix of the environment variable name, where the environment variable name is <prefix>.<base>.

appd_context_config_init

Initializes and returns an empty context configuration structure.

Format

```
APPD_API struct appd_context_config* appd_context_config_init(const char* context_name)
```

Parameters

- `context_name:` Name for this context.

appd_context_config_set_controller_host

Add the Controller host to the application context.

Format

```
APPD_API void appd_context_config_set_controller_host (struct appd_context_config * context_cfg, const char * host)
```

Parameters

- `context_cfg`: An AppDynamics context configuration object indicating the business application, tier and node name for this context.
- `host`: Name of the Controller host.

appd_context_config_set_controller_port

Add the Controller port to the application context.

Format

```
APPD_API void appd_context_config_set_controller_port (struct appd_context_config * context_cfg, const unsigned short port)
```

Parameters

- `context_cfg`: An AppDynamics context configuration object indicating the business application, tier, and node name for this context.
- `port:` Port number that the Controller is listening on.

appd_context_config_set_controller_account

Add the Controller port to the application context.

Format

```
APPD_API void appd_context_config_set_controller_port (struct appd_context_config * context_cfg, const char * acct)
```

Parameters

- `context_cfg`: An AppDynamics context configuration object indicating the business application, tier, and node name for this context.
- `account:` Account name that the Controller is connected to.

appd_context_config_set_controller_access_key

Add the Controller access key to the application context.

Format

```
APPD_API void appd_context_config_set_controller_port (struct appd_context_config * context_cfg, const char * key)
```

Parameters

- `context_cfg`: An AppDynamics context configuration object indicating the business application, tier, and node name for this context.
- `key:` Access key for connecting to the Controller.

appd_context_config_set_controller_use_ssl

Add the Controller access key to the application context.

Format

```
APPD_API void appd_context_config_set_controller_port (struct appd_context_config * context_cfg, unsigned int * ssl
)
```

Parameters

- `context_cfg`: An AppDynamics context configuration object indicating the business application, tier, and node name for this context.
- `ssl`: Set to a non-zero integer for true. Set to the integer zero for false. Note that SaaS Controllers require this to be set to non-zero.

appd_context_config_set_controller_http_proxy_host

Optional. Set the hostname of the HTTP proxy if using an HTTP proxy to talk to controller.

Format

APPD_API void appd_context_config_set_controller_http_proxy_host(struct appd_context_config * *context_cfg*, const char * *host*);

Parameters

- `context_cfg`: An AppDynamics context configuration object indicating the business application, tier, and node name for this context.
- `host`: Hostname of the HTTP proxy if using an HTTP proxy to talk to Controller.

appd_context_config_set_controller_http_proxy_port

Optional. Set the hostname of the HTTP proxy if using an HTTP proxy to talk to Controller.

Format

```
APPD_API void appd_context_config_set_controller_http_proxy_host(struct appd_context_config * context_cfg, const unsigned short port);
```

Parameters

- `context_cfg`: An AppDynamics context configuration object indicating the business application, tier, and node name for this context.
- `port:` Port name of the HTTP proxy. Default is 80.

appd_context_config_set_controller_http_proxy_username

Optional. Set the username to connect to the HTTP proxy with**.**

Format

```
APPD_API void appd_context_config_set_controller_http_proxy_host(struct appd_context_config * context_cfg, const char * user);
```

Parameters

- `context_cfg`: An AppDynamics context configuration object indicating the business application, tier, and node name for this context.
- `user`: Username to connect to the HTTP proxy with.

appd_context_config_set_controller_http_proxy_password

Optional. Set the password to connect to the HTTP proxy with.

Format

```
APPD_API void appd_context_config_set_controller_http_proxy_host(struct appd_context_config * context_cfg, const char * pwd);
```

Parameters

- `context_cfg`: An AppDynamics context configuration object indicating the business application, tier, and node name for this context.
- `pwd`: Password to connect to the HTTP proxy with.

appd_context_config_set_controller_http_proxy_password_file

Optional. Set the password to connect to the HTTP proxy with.

Format

```
APPD_API void appd_context_config_set_controller_http_proxy_host(struct appd_context_config * context_cfg, const char * file);
```

Parameters

- `context_cfg`: An AppDynamics context configuration object indicating the business application, tier, and node name for this context.
- `file`: File containing password to connect to the HTTP proxy with.

appd_context_config_set_controller_certificate_file

Optional. Set the CA certificate file name. Set this if you choose to use your own certificate file.

Format

```
APPD_API void appd_context_config_set_controller_http_proxy_host(struct appd_context_config * context_cfg, const char * file);
```

Parameters

- `context_cfg`: An AppDynamics context configuration object indicating the business application, tier, and node name for this context.
- `file`: Name of the certificate file. Defaults to the included `ca-bundle.crt` file.

appd_context_config_set_controller_certificate_dir

Optional. Set the full path to the CA certificate file. Set this if you have multiple certificate files.

Format

```
APPD_API void appd_context_config_set_controller_http_proxy_host(struct appd_context_config * context_cfg, const char * dir);
```

Parameters

- `context_cfg`: An AppDynamics context configuration object indicating the business application, tier, and node name for this context.
- `dir`: Full path to the certificate files.

appd_custom_event_start

This function starts the definition of a custom event. Returns a handle to the defined event. By calling `appd_custom_event_end()`, the definition of the event is signaled as complete. The event handle returns null on failure.

Format

```
APPD_API appd_event_handle appd_custom_event_start(const char* application_context, enum appd_event_severity severity, const char* event_sub_type, const char* summary)
```

Parameters

- `application_context`: This string picks one agent out of multiple agents running in multi-tenant mode. If the value is null, the default agent is used.
- `severity`: This is an enum representing the severity of the event. Valid values are `APPD_EVENT_SEVERITY_INFO`, `APPD_EVENT_SEVERITY_WARNING`, and `APPD_EVENT_SEVERITY_ERROR`.
- `event_sub_type`: The string containing the subtype of the custom event. This can be used in the Controller to filter the custom events belonging to a specific subtype.
- `summary`: A string with a brief description of the event.

appd_custom_event_add_detail

This function adds a detail made up of a name and a value to the definition of a custom event. Returns a non-zero value on success, otherwise zero. This call can be made between the calls of `appd_custom_event_start()` and `appd_custom_event_end()`. This function can be called multiple times to add details to the event. The event handle returned by `appd_custom_event_start()` identifies the right event to add the detail to. If multiple calls to this function are made with the same detail name, then the detail value from the most recent call will be used to associate that detail name.

Format

```
APPD_API int appd_custom_event_add_detail(appd_event_handle event_handle, const char* detail_name, const char* detail_value)
```

Parameters

- `detail_name`: This string contains the name of the detail that needs to be added. If the value is null or empty, the API fails and returns zero.
- `detail_value`: This string contains the name of the value that needs to be added. If the value is null or empty, the API fails and returns zero.
- `event_handle`: Refers to the event to which the detail needs to be added to. If the value is null or invalid, the API fails and returns zero.

appd_custom_event_add_property

This function adds a property made up of a name and a value to the definition of a custom event. Returns a non-zero value on successful, otherwise zero. This call can be made between the calls of `appd_custom_event_start()` and `appd_custom_event_end()`. This function can be called multiple times to add properties to the event. The event handle returned by `appd_custom_event_start()` identifies the right event to add the property to. This property can be used in the Controller to filter the custom events. If multiple calls to this function are made with the same property name, then the property value from the most recent call will be used to associate that property name.

Format

```
APPD_API int appd_custom_event_add_property(appd_event_handle event_handle, const char* property_name, const char* property_value)
```

Parameters

- `event_handle`: Refers to the event to which the property needs to be added to. If the value is null or invalid, the API fails and returns zero.
- `property_name`: This string contains the name of the property to be added. If the value is null or empty, the API fails and returns zero.
- `property_value`: This string contains the value of the property to be added. If the value is null or empty, the API fails and returns zero.

appd_custom_event_end

Signals to the C++ SDK that the event definition is complete and queues the event to be sent to the Controller. Returns a zero value on success, otherwise a non-zero value.

Format

```
APPD_API int appd_custom_event_end(appd_event_handle event_handle)
```

Parameters

- `event_handle`: Used to refer to the event. If the value is null or invalid, the API fails and returns zero.

appd_eum_get_cookie

Returns an EUM–ADRUM cookie for a business transaction in an EUM-enabled browser application. You can call this function between `appd_bt_begin()` and `appd_bt_end()`.

At this time, `appd_eum_get_cookie` works only on snapshots that contain call graphs. See Correlate Business Transactions for EUM for more information.

Format

```
APPD_API const char* appd_eum_get_cookie(appd_bt_handle bt, int https, int short_form, const char* referrer_url,
const char* path)
```

Parameters

- `bt`: The handle to the business transaction that the EUM-ADRUM cookie is generated for.
- `https`: When set to a non-zero value, this flag indicates that the incoming request uses the HTTPS protocol. This flag is used to make the EUM cookie secure.
- `short_form`: When set to a non-zero value, this flag indicates that shortened names will be used for sub-cookie names inside the EUM cookie.
- referrer_url: The URL of the page from which user made the request. This can be set to `NULL` or `""` if the referrer URL is unknown.
- `path`: The URL path where the cookie will be stored. This can be set to `"/"` in most cases. If set to `NULL` or `""`, the default value `"/"` is used.

Returns

On success, the function returns a string that contains a cookie key and a cookie value separated by =. Do not free the memory associated with the returned cookie string; when you call `appd_bt_end()`, memory is automatically released.

If EUM is disabled or no configuration information has been received from the Controller, then NULL is returned.

appd_sdk_add_app_context

Initialize the AppDynamics SDK.

Format

```
APPD_API int appd_sdk_add_app_context (struct appd_context_config *  context_cfg)
```

Parameters

- `context_cfg`: An AppDynamics context configuration object indicating the business application, tier, and node name for this context.

appd_sdk_init

Initialize the AppDynamics C/C++ SDK. An instrumented application must call this function once, preferably during application startup.

When using `fork()`, the `appd_sdk_init()`, and all other SDK calls, should occur only in the forked process, never in the parent, and each child should behave as its own process, and never share handles between the process itself and other SDK instrumented child processes.

Format

```
APPD_API int appd_sdk_init (const struct appd_config *  config)
```

Parameters

- `config`: AppDynamics configuration settings that enable communication between the agent and the Controller.

Returns

Zero on success, otherwise a non-zero value. If the return value is not zero, a log message describes the error.

appd_sdk_term

Stop the C/C++ SDK. Ends all active business transactions for this agent and stops agent metric reporting to the controller.

If you execute the agent for less than a few minutes, the agent may not have enough time to fully report to the Controller, as this action is done once every 60 seconds. To ensure that any remaining data is transmitted to the Controller, wait at least one minute before issuing the `appd_sdk_term()` call.

Format

```
APPD_API void appd_sdk_term()
```

appd_bt_begin

Start a Business Transaction or continue an existing transaction.

Keep in mind that each application is limited to 200 registered business transactions, and each agent is limited to 50 registered business transactions. Like transactions discovered by other types of agents, business transactions that exceed the limit that are reported by the C/C++ SDK are included in the "all other traffic" grouping. Ensure that your agent does not create excessive business transactions and that your implementation does not introduce the possibility of business transaction explosion.

Format

```
APPD_API appd_bt_handle appd_bt_begin (const char *  name, const char *  correlation_header)
```

Parameters

- `name`: The name for the business transaction. In the case of a continuing transaction in the current business application with a valid correlation header, the SDK uses the name from the header. Do not use the following characters in transaction names: `{ } [ ] | & ;`
- `correlation_header`: A correlation header if this is a continuing transaction, else `NULL`. If specified, the correlation header has been generated by another AppDynamics agent and made available to this agent as a string. The correlation header provides information to enable this transaction to correlate with an upstream transaction.

Returns

An opaque handle for the business transaction that was started.

appd_bt_begin_with_app_context

Start a business transaction or continue an existing transaction in a multi-tenant Controller environment.

Format

```
APPD_API appd_bt_handle appd_bt_begin_with_app_context (const char *  context, const char *  name, const char *  correlation_header)
```

Parameters

- `context`: The application context name that this business transaction belongs to.
- `name`: The name for the business transaction. In the case of a continuing transaction in the current business application with a valid correlation header, the SDK uses the name from the header. Do not use the following characters in transaction names: `{ } [ ] | & ;`
- `correlation_header`: A correlation header if this is a continuing transaction, else `NULL`. If specified, the correlation header has been generated by another AppDynamics agent and made available to this agent as a string. The correlation header provides information to enable this transaction to correlate with an upstream transaction.

appd_bt_enable_snapshot

Enables a snapshot for a given business transaction. This call can be made between the calls of `appd_bt_begin()` and `appd_bt_end()`. Returns a non-zero value if the snapshot is successfully enabled, otherwise returns a zero.

Format

```
APPD_API int appd_bt_enable_snapshot(appd_bt_handle bt)
```

Parameters

- `bt`: The handle to the business transaction for which the snapshot needs to be enabled.

appd_bt_end

End the given business transaction.

Format

```
APPD_API void appd_bt_end (appd_bt_handle  bt)
```

Parameters

- `bt`: The handle to the business transaction to end.

appd_bt_get

Get a BT handle associated with the given GUID by `appd_bt_store`.

Format

```
APPD_API appd_bt_handle appd_bt_get (const char *  guid)
```

Parameters

- `guid`: The globally unique identifier that was passed to `appd_bt_store`.

Returns

The handle to the business transaction associated with the given GUID.

In the following cases the SDK logs a warning and returns a handle that you may safely use in other API functions but that will cause these functions to immediately return without doing anything:

- The SDK does not find a handle associated with the GUID.
- The call ended before the SDK could retrieve the handle.

appd_bt_is_snapshotting

Reports whether the agent is currently taking a transaction snapshot. Useful before calling `appd_add_user_data()` or `appd_bt_set_url()`, since those potentially expensive functions would do nothing if called when a snapshot is not being taken.

Format

```
APPD_API char appd_bt_is_snapshotting (appd_bt_handle  bt)
```

Parameters

- `bt:` The handle to the business transaction to check for snapshotting.

Returns

Non-zero if the given business transaction is taking a snapshot. Otherwise, zero.

appd_bt_set_url

Set URL for a snapshot (if one is being taken).

URL is set for a snapshot if one is occurring. Data should be either 7-bit ASCII or UTF-8.

It is safe to call this function when a snapshot is not occurring. When the given business transaction is NOT snapshotting, this function immediately returns. However, if extracting the data to pass to this function is expensive, you can use `appd_bt_is_snapshotting` to check if the business transaction is snapshotting before extracting the data and calling this function.

Format

```
APPD_API void appd_bt_set_url (appd_bt_handle  bt, const char *  url)
```

Parameters

- `bt`: The business transaction to add the user data to, if it's taking a snapshot.
- url: The value of the URL for the snapshot as 7-bit ASCII or UTF-8.

appd_bt_override_start_time_ms

Takes a time, in milliseconds, representing the number of milliseconds elapsed since Jan 1, 1970 UTC. The time specified for this function overrides the default start time, which is the time at which the `appd_bt_begin()` API is called according to the system clock. By overriding the start time with a specific value, the BT internal start time is disabled, and the specified start time is reported to the Controller.

This function is located in the `appdynamics_advanced.h` file, and is only used in special cases.

Format

```
APPD_API void appd_bt_override_start_time_ms(appd_bt_handle bt, unsigned int timeMS);
```

Parameters

- `bt`: The business transaction to override the timing of
- `timeMS` : Time in milliseconds since start of epoch (midnight, Jan 1, 1970 UTC).

appd_bt_override_time_ms

Takes a time, in milliseconds, that overrides the business transaction response time as reported for this business transaction to the Controller.

The C/C++ SDK maintains its own, internal timer for the response time for the business transaction. This timer reflects the elapsed time spent between the `bt_begin` and `bt_end` calls. In some cases, you may want to set the business transaction timer directly, overriding the default business transaction response timer. For instance, this may be useful for integrating external monitoring systems with AppDynamics.

It is important to note that when calling this function, the reported business transaction will be either the time you specify or the sum of all exit call timings for the transaction, whichever is greatest. This is because a business transaction cannot be reported to take less time than the total of the exit calls it contains.

This function is located in the `appdynamics_advanced.h` file, and is only used in special cases.

Format

```
APPD_API void appd_bt_override_time_ms(appd_bt_handle bt, unsigned int timeMS);
```

Parameters

- `bt`: The business transaction for which to set the response time.
- `timeMS`: The time the business transaction took, in milliseconds.

appd_bt_store

Store a BT handle for retrieval with `appd_bt_get`.

This function allows you to store a BT in a global registry to retrieve later. This is convenient when you need to start and end a BT in separate places, and it is difficult to pass the handle to the BT through the parts of the code that need it.

When the BT is ended, the handle is removed from the global registry.

Format

```
APPD_API void appd_bt_store (appd_bt_handle  bt, const char *  guid)
```

Parameters

- `bt:` The BT to store.
- `guid:` A globally unique identifier to associate with the given BT.

Example

```
int begin_transaction(uint64_t txid, uint64_t sku, float price)
{
    appd_bt_handle bt = appd_bt_begin("payment-processing", NULL);
    appd_bt_store(bt, std::to_string(txid).c_str());
    // ...
}
```

appd_bt_add_error

Add an error to a business transaction.

Errors are reported as part of the business transaction. However, you can add an error without marking the business transaction as an error (e.g., for non-fatal errors).

Format

```
APPD_API void appd_bt_add_error (appd_bt_handle  bt, enum appd_error_level  level, const char *  message, int  mark_bt_as_error)
```

Parameters

- `bt:` The handle to the business transaction to which the error is added.
- `level:` The error levels are `APPD_LEVEL_NOTICE`, `APPD_LEVEL_WARNING`, and `APPD_LEVEL_ERROR`.
- `message:` Error message for this error.
- `mark_bt_as_error:` If true, the business transaction experiencing this error is marked as an error transaction. In this case, the business transaction is counted only as an error transaction. It is not also counted as a slow, very slow or stalled transaction, even if the transaction was also slow or stalled. If false, the business transaction is not marked as an error transaction.

appd_bt_add_user_data

Attaches user data to transaction snapshots generated for the specified business transaction.

The user data is added to the business transaction but reported only when a transaction snapshot is occurring. In the Controller UI, the user data appears in the **Business Data** tab of the transaction snapshot details.

It is safe to call this function when a snapshot is not occurring. When the specified business transaction is not taking a snapshot, this function immediately returns.

If extracting the data to pass to this function is expensive, you can use `appd_bt_is_snapshotting()` to check if the business transaction is actually taking a snapshot before extracting the data and calling this function.

Data should be either 7-bit ASCII or UTF-8.

Format

```
APPD_API void appd_bt_add_user_data (appd_bt_handle  bt, const char *  key, const char *  value)
```

Parameters

- `bt`: The business transaction to add the user data to, if it's taking a snapshot.
- `key:` The name of the user data to add to the snapshot as 7-bit ASCII or UTF-8.
- `value:` The value of the user data to add to the snapshot as 7-bit ASCII or UTF-8.

appd_backend_add

Add a declared backend to the current business application. Use `appd_backend_declare` to declare a backend before adding it.

This function fails if the type of the backend being added does not have at least one identifying property.

Format

```
APPD_API int appd_backend_add (const char *  backend)
```

Parameters

- `backend:` Pointer to the backend.

Returns

Zero on success, otherwise a non-zero value. If the return value is not zero, a log message describes the error.

appd_backend_declare

Declare the existence of a backend. Call this only if the backend is not already registered with the C++ SDK instance.

Format

```
APPD_API void appd_backend_declare (const char *  type, const char *  unregistered_name)
```

Parameters

- `type:` The type of the backend, from these options:
    - `APPD_BACKEND_HTTP`
    - `APPD_BACKEND_DB`
    - `APPD_BACKEND_CACHE`
    - `APPD_BACKEND_RABBITMQ`
    - `APPD_BACKEND_WEBSERVICE`
    - `APPD_BACKEND_JMS`
- `unregistered_name:` The name of the backend. Must be unique to the C++ SDK instance.

appd_backend_prevent_agent_resolution

Call to prevent a downstream agent from resolving as this backend. If used, this must be called before `appd_backend_add()`.

Normally, if an agent picks up a correlation header for an unresolved backend, it will resolve itself as that backend. This is usually the desired behavior.

However, if the backend is actually an uninstrumented tier that is passing through the correlation header (for example, a message queue or proxy), then you may wish the backend to show up distinct from the tier that it routes to. If you call this function, correlation headers generated for exit calls to this backend in the SDK will instruct downstream agents to report as distinct from the backend.

For example: if you have Tier A talking to uninstrumented Backend B which routes to instrumented Tier C, if you do NOT call this function, the flow map will be A > C. If you DO call this function, the flow map will be A > B > C.

Format

```
APPD_API int appd_backend_prevent_agent_resolution (const char *  backend)
```

Parameters

- `backend:` Pointer to the declared backend.

Returns

Zero on success, otherwise a non-zero value. If the return value is not zero, a log message describes the error.

appd_backend_set_identifying_property

Sets an identifying property of a backend. Call once for every identifying property that you want to set. Call this function after `appd_backend_declare()` and before `appd_backend_add()`.

A backend's identifying properties uniquely identify the downstream component being called. In the Controller UI, these properties are visible in the upper right panel of the backend dashboards.

You must set at least one identifying property when you add a backend of one of the built-in exit call types. The valid properties vary by the following exit call type.

Exit Call Type	Valid Identifying Properties
APPD_BACKEND_HTTP	HOST, PORT, URL, QUERY STRING
APPD_BACKEND_DB	HOST, PORT, DATABASE, VENDOR, VERSION
APPD_BACKEND_CACHE	SERVER POOL, VENDOR
APPD_BACKEND_RABBITMQ	HOST, PORT, ROUTING KEY, EXCHANGE
APPD_BACKEND_WEBSERVICE	SERVICE, URL, OPERATION, SOAP ACTION, VENDOR
APPD_BACKEND_JMS	DESTINATION, DESTINATIONTYPE, VENDOR
APPD_BACKEND_WEBSPHEREMQ	HOST, PORT, DESTINATION, DESTINATIONTYPE, MAJOR VERSION, VENDOR

Format

```
APPD_API int appd_backend_set_identifying_property (const char *  backend, const char *  key, const char *  value)
```

Parameters

- `backend:` Pointer to the backend.
- `key:` Name of the property. For example, DESTINATION or PORT.
- `value:` Value of the property. For example, Order Queue or 3304

Returns

Zero on success, otherwise a non-zero value. If the return value is not zero, a log message describes the error.

appd_custom_metric_add

Report a custom metric.

Format

```
APPD_API void appd_custom_metric_add (const char *  application_context, const char *  metric_path, enum appd_time_rollup_type  time_rollup_type, enum appd_cluster_rollup_type  cluster_rollup_type, enum appd_hole_handling_type  hole_handling_type)
```

Parameters

- `application_context:` The application context for this custom metric.
- `metric_path:` The path of the custom metric. This path defines where the custom metric appears in the Metric Browser. Use a pipe character to separate branches in the path. For example, `Custom Metrics|MyCustomMetric` defines `MyCustomMetric` as a top-level metric in the tree.
- `time_rollup_type:` Specifies how to rollup metric values for this metric over time, from the following:
  - `APPD_TIMEROLLUP_TYPE_AVERAGE:` Compute the average value of the metric over time.
  - `APPD_TIMEROLLUP_TYPE_SUM:` Compute the sum of the value of the metric over time.
  - `APPD_TIMEROLLUP_TYPE_CURRENT:` Report the current value of the metric.
- `cluster_rollup_type`: Specifies how to rollup metric values for this metric across clusters, from the following:
  - `APPD_CLUSTERROLLUP_TYPE_INDIVIDUAL`: Roll-up the value individually for each member of the cluster.
  - `APPD_CLUSTERROLLUP_TYPE_COLLECTIVE`: Roll-up the value across all members of the cluster.
- `hole_handling_type`: Specifies how to handle holes—gaps where no value has been reported from this metric, from the following:
  - `APPD_HOLEHANDLING_TYPE_RATE_COUNTER`: Considers the gap or 0 value to be meaningful in the aggregation of the metric.

- `APPD_HOLEHANDLING_TYPE_REGULAR_COUNTER`: Does not consider the gap or 0 value to be meaningful in the aggregation of the metric.

For additional information on these parameters, see Create Custom Metrics.

appd_custom_metric_report

Report a value for a given metric.

Format

```
APPD_API void appd_custom_metric_report (const char *  application_context, const char *  metric_path, long  value)
```

Parameters

- `application_context`: The application context for this custom metric.
- `metric_path`: The path of the metric to report, as defined by `appd_custom_metric_add`.
- `value`: The value to report for the metric. The way the value is aggregated is specified by the roll-up parameters to `appd_custom_metric_add`.

appd_exitcall_add_error

Add an error to the exit call.

Format

```
APPD_API void appd_exitcall_add_error (appd_exitcall_handle  exitcall, enum appd_error_level  level, const char
*  message, int  mark_bt_as_error)
```

Parameters

- `exitcall`: Handle to the exit call.
- `level`: The level of this error are `APPD_LEVEL_NOTICE`, `APPD_LEVEL_WARNING`, and `APPD_LEVEL_ERROR`.
- `message`: Error message for this error.
- `mark_bt_as_error`: If true, the business transaction making the exit call that is experiencing this error is marked as an error transaction. In this case, the business transaction is counted only as an error transaction. It is not also counted as a slow, very slow or stalled transaction, even if the transaction was also slow or stalled. If false, the business transaction is not marked as an error transaction.

appd_exitcall_begin

Start an exit call to the specified backend as part of a business transaction.

Format

```
APPD_API appd_exitcall_handle appd_exitcall_begin (appd_bt_handle  bt, const char *  backend)
```

Parameters

- `bt:` Handle to the business transaction making the exit call.
- `backend`: The destination backend of the exit call. AppDynamics does not automatically detect backends for Go applications, so you must specify the destination backend.

Returns

An opaque handle to the exit call that was started.

appd_exitcall_end

Complete the exit call.

Format

```
APPD_API void appd_exitcall_end (appd_exitcall_handle  exitcall)
```

Parameters

- `exitcall`: Handle to the exit call being ended.

appd_exitcall_get

Get a handle to an exit call associated with a GUID via `appd_exitcall_store()`.

Format

```
APPD_API appd_exitcall_handle appd_exitcall_get (const char * guid)
```

Parameters

- `guid:` The globally unique identifier that was passed to `appd_exitcall_store()`.

Returns

The handle to the exit call associated with the given GUID.

In the following cases, the SDK logs a warning and returns a handle that you may safely use in other API functions but that will cause these functions to immediately return without doing anything:

- The SDK doesn't find a handle associated with the GUID.
- The call ended before the SDK could retrieve the handle.

appd_exitcall_get_correlation_header

Get the header for correlating a business transaction.

If a business transaction makes exit calls that you wish to correlate across, you should retrieve the correlation header and inject it into your exit call's payload.

The returned string is freed when the exit call ends. Do not free it yourself.

Format

```
APPD_API const char* appd_exitcall_get_correlation_header (appd_exitcall_handle  exitcall)
```

Parameters

- `exitcall`: Handle to the exit call.

Returns

On success returns a 7-bit ASCII string containing the correlation information. You can inject this string into the exit call's payload. A downstream agent can then extract the header from that payload and continue the business transaction.

On error, returns the default header that prevents downstream business transaction detection.

appd_exitcall_override_start_time_ms

Takes a time, in milliseconds, representing the number of milliseconds elapsed since Jan 1, 1970 UTC. The time specified for this function overrides the default start time, which is the time at which the `appd_exitcall_begin()` API is called according to the system clock. By overriding the start time with a specific value, the exit call's internal start time is disabled, and the specified start time is reported to the Controller.

This function is located in the `appdynamics_advanced.h` file, and is only used in special cases.

Format

APPD_API void appd_exitcall_override_time_ms(appd_exitcall_handle exitCall, int64_t timeMS);

Parameters

- `exitCall`: The exit call for which to override the start time
- `timeMS:` The time that you want to override the exit call start time

appd_exitcall_override_time_ms

Takes a time, in milliseconds, that overrides the exit call time reported to the Controller.

The C/C++ SDK maintains its own internal timer for the time it takes to complete the exit call. In some cases, you may want to disable the exit call's internal timer. For instance, this can be useful for reporting exit calls that are recorded in external monitoring systems and read into an SDK program.

This function is located in the `appdynamics_advanced.h` file, and is only used in special cases.

Format

```
APPD_API void appd_exitcall_override_time_ms(appd_exitcall_handle exitCall, int64_t timeMS);
```

Parameters

- `exitCall`: The exit call for which to set the completion time
- `timeMS`: The time that you want to override the exit call time.

appd_exitcall_set_details

Set the details string for an exit call. This can be used, for example, to add the SQL statement that a DB backend has executed as part of the exit call. This data is then visible in the exit calls details UI for the transaction snapshot.

Format

```
APPD_API int appd_exitcall_set_details (appd_exitcall_handle  exitcall, const char *  details)
```

Parameters

- `exitcall`: Handle to the exit call.
- `details`: An arbitrary string to add to the exit call.

Returns

Zero on success, otherwise a non-zero value. If the return value is not zero, a log message describes the error.

appd_exitcall_store

Store an exit call handle for retrieval with `appd_exitcall_get()`.

This function allows you to store an exit call in a global registry to retrieve later. This is convenient when you need to start and end the call in separate places, and it is difficult to pass the handle through the parts of the code that need it.

The handle is removed when the exit call (or the BT containing it) ends.

Format

```
APPD_API void appd_exitcall_store (appd_exitcall_handle  exitcall, const char *  guid)
```

Parameters

- `exitcall`: The exit call to store.
- `guid`: A globally unique identifier to associate with the given call.

Example

```
appd_exitcall_handle ec = appd_exitcall_begin(bt, "authdb");
appd_exitcall_store(ec, "login-exit");
```

appd_frame_begin

Records the start of a method call in the call stack. To track the duration of your method call, you must call `appd_frame_begin` near the start of the method code, and `appd_frame_end` when the method returns.

Format

```
APPD_API appd_frame_handle appd_frame_begin(appd_bt_handle bt, enum appd_frame_type frame_type,
const char* class_name, const char* method_name, const char* file, unsigned int line_number);
```

Parameters

- `bt`: The business transaction for the call graph.
- `frame_type`: The type of the frame. When used in C or C++ code, use `APPD_FRAME_TYPE_CPP`.
- `class_name`: The name of the class if this method is a member of the class. If not, then set to NULL.
- `method_name`: The name of the method.
- `file`: The path of the source file.
- `line_number`: The line number in the source file.

Returns

An opaque handle for the frame. NULL if an error occurred.

appd_frame_end

Records the end of a method call in the call stack. To track the duration of your method call, you must call `appd_frame_begin` near the start of the method code, and `appd_frame_end` when the method returns.

Format

```
APPD_API void appd_frame_end(appd_bt_handle bt, appd_frame_handle frame);
```

Parameters

- `bt`: The business transaction for the call graph.
- `frame`: The handle of returned by the corresponding `appd_frame_begin` call.

Environment Variables

You can configure the following environment variables for the C/C++ SDK.

Environment Variable	Value
`APP_NAME`	The name of the application.
`CONTROLLER_ACCESS_KEY`	The access key for connecting to the Controller.
`CONTROLLER_ACCOUNT`	The account name for connecting to the Controller.
`CONTROLLER_HOST`	The hostname of the Controller.
`CONTROLLER_PORT`	The port number on which the Controller is listening.
`CONTROLLER_HTTP_PROXY_HOST`	If using an HTTP proxy to talk to the Controller, the hostname of that HTTP proxy.
`CONTROLLER_HTTP_PROXY_PASSWORD`	The password for connecting to the HTTP proxy.
`CONTROLLER_HTTP_PROXY_PASSWORD_FILE`	The file containing password to connect to the HTTP proxy with.
`CONTROLLER_HTTP_PROXY_PORT`	The port number of the HTTP proxy.
`CONTROLLER_HTTP_PROXY_USERNAME`	The username to connect to the HTTP proxy with.
`CONTROLLER_USE_SSL` (optional)	To disable SSL, set to `off`, `0`, `f`, or `false`. To enable SSL, set to `on`, `1`, `t`, or `true`.
`CONTROLLER_CERTIFICATE_FILE`	The SSL certificate file name for connecting to the Controller.
`CONTROLLER_CERTIFICATE_DIR`	The directory where the SSL certificate is located.
`FLUSH_METRICS_ON_SHUTDOWN`	To enable metric flushing before shutdown, set to a non-zero integer. To disable metric flushing before shutdown, set to `0` (default).
`INIT_TIMEOUT_MS`	The number of milliseconds that you want `appd_sdk_init,` an asynchronous action, to wait until it has received Controller configuration and is ready to capture business transactions.
`LOGGING_LOG_DIR`	The directory that the logs are saved to.
`LOGGING_LEVEL`	The level of severity for which activity is logged. Options: all, trace, debug, info, warn, error.
`LOGGING_MAX_NUM_FILES`	The maximum number of log files that can be saved.
`LOGGING_MAX_FILE_SIZE_BYTES`	The maximum log file size that can be saved.
`NODE_NAME`	The name of the node.
`TIER_NAME`	The name of the tier.
`UNIQUE_HOST_ID` (optional)	The identifier of the physical host or virtual machine that the agent is installed on.

CPP SDK Resource Management

Related pages:

- C/C++ SDK Reference

This page describes the Resource Acquisition Is Initialization (RAII) functions defined in the C/C++ SDK to instrument C++ applications.

For C++ applications, you can use the `BT` and `ExitCall` classes to simplify your instrumentation. With these classes, you do not need to manually call `appd_bt_end()` and `appd_exitcall_end()`. When the `BT` or `ExitCall` object goes out of scope, the SDK calls its destructor and the business transaction/exit call automatically ends.

The following code example use these classes.

```
{
   appd::sdk::BT bt("mybt");
   appd::sdk::ExitCall ec(bt, "auth");
   if (!make_auth_call())
   {
      bt.add_error(APPD_LEVEL_ERROR, "Authorization failed");
      return -1;
   }
   issue_api_call();
}
```

The above code example is equivalent to this:

```
{
    appd_bt_handle bt = appd_bt_begin("mybt", NULL);
    appd_exitcall_handle ec = appd_exitcall_begin(bt, "auth");
    if (!make_auth_call())
    {
       appd_bt_add_error(bt, APPD_LEVEL_ERROR, "Authorization failed", 0);
       appd_exitcall_end(ec);
       appd_bt_end(bt);
       return -1;
    }
    issue_apI_call();
    appd_exitcall_end(ec);
    appd_bt_end(bt);
}
```

When the BT lifetime depends on the non-deterministic lifetimes of other objects, you can use a shared pointer to a `BT` to keep the `BT` alive for the lifetimes of its dependencies. In this example, the `BT` ends when the last reference to it ends.

```
{
   //Initialize the BT with a shared pointer
   auto bt = std::make_shared<appd::sdk::BT>("compute");
   auto prod = createProducer(bt);
   auto consumers = createConsumers(bt, NUM_WORKERS);
   //Variable BT goes out of scope with no further references.
   //BT ends automatically.
}
```

For managing exit calls with complex lifetimes, consider using smart pointers `std::unique_ptr<appd::sdk::ExitCall>` or `std::shared_ptr<appd::sdk::ExitCall>`.

appd:sdk::BT

A RAII pattern constructor for C++ applications. For a new Business Transaction, this constructor creates the business transaction object and a handle to that object. This function continuous existing transactions and adds the correlation header if one is passed.

When the business transaction object goes out of scope, the SDK automatically ends the business transaction. A business transaction object cannot be copied.

Format

```
appd::sdk::BT(const std::string& name)
appd::sdk::BT(const std::string& name, const std::string& correlation_header)
appd::sdk::BT(const char* name, const char* correlation_header=NULL)
```

Parameters

- `name:` Business transaction name, passed as either a `string&` or `char *`, depending on which constructor you choose. In the case of a continuing transaction in the current business application with a valid correlation header, the SDK uses the name from the header.
- `correlation_header`: Optional correlation header, passed as either a `string&` or `char *`, depending on which constructor you choose.

appd::sdk::Event

This class is used to create an event in C++ and provides a convenient way of creating events. See C/C++ SDK Reference.

Format

```
appd::sdk::Event::Event
(const std::string& application_context,
enum appd_event_severity severity,
const std::string& even_sub_type,
const std::string& summary,
const std::map<std::string, std::string>& properties,
const std::map<std::string, std::string>& details)
```

Parameters

- `application_context`: This string picks an agent out of multiple agents running in multi-tenant mode. If this value is empty, then the default agent is used.
- `details`: A map of name and value pairs. An empty map is a valid argument. The name and value associated with a detail cannot be empty; that is an error.
- `event_sub_type`: The string containing the subtype of the custom event. This can be used in the Controller to filter the custom events belonging to a specific subtype.
- `properties`: A map of name and value pairs. An empty map is a valid argument. The name and value associated with a property cannot be empty; that is an error.
- `severity`: This is an enum representing the severity of the event. Valid values are `APPD_EVENT_SEVERITY_INFO`, `APPD_EVENT_SEVERITY_WARNING` and `APPD_EVENT_SEVERITY_ERROR`.
- `summary`: A string giving a brief description about the event.

appd::sdk::Event::report

This method in Event class is equivalent to `appd_custom_event_end()` in functionality. See C/C++ SDK Reference.

This function returns boolean value as `true` if the definition of the event is successful and queuing of the event (to send to the Controller) is successful. Otherwise, returns false.

Format

```
appd::sdk::Event event(<arguments>);
bool status = event.report();
```

appd:sdk::ExitCall

A RAII pattern constructors for C++ applications. Constructs an ExitCall object and creates a handle to that object. You must declare a backend and assign it at least one identifying property before you use the backend to instantiate the ExitCall object.

When the ExitCall object goes out of scope, the SDK automatically ends the ExitCall.

An ExitCall object cannot be copied.

Format

```
appd::sdk::ExitCall(BT& bt, const char* backend)
appd::sdk::ExitCall(BT& bt, const std::string& backend)
```

Parameters

- `BT& bt`: Handle to the business transaction making the exit call.
- `backend`: The destination backend of the exit call. Passed as either a `string&` or `char *`, depending on which constructor you choose.

appd::sdk::Frame

Use the Frame class to enable call graphs for your C++ application. You can use the Frame class to instrument the methods that you want to track, as an alternative to calling `appd_frame_begin` and `appd_frame_end`. You must pass the following parameters into the Frame class:

Format

```
Frame(BT& bt, appd_frame_type frame_type, const char* class_name, const char* method_name, const char* file, unsigned int line_number)
```

Parameters

- `bt`: The BT object that owns this function call.
- `frame_type`: The type of the frame. When used in C++ RAII code, use `APPD_FRAME_TYPE_CPP`.
- `class_name`: The name of the class if this method is a member of the class, else `NULL`.
- `method_name`: The name of the method. You can use `__FUNCTION__` or `__PRETTY_FUNCTION__`; any available compiler macro.
- `file`: The path of the source file. You can use `__FILE__` or any available compiler macro.
- `line_number`: The line number in the source file. You can use `__LINE__` or any available compiler macro.

Install the CCPP SDK on Windows

Related pages:

- Download AppDynamics Software
- Install the C/C++ SDK on Linux

To install the C/C++ SDK on Windows:

1. Download the C/C++ SDK from the AppDynamics Downloads Portal that corresponds to the bit number for your operating system (32-bit or 64-bit). The incorrect bit–SDK version compatibility will cause a linker error about missing symbols.
2. Extract the downloaded SDK zip file to the directory where you want to place the SDK, such as `C:\AppDynamics`.
3. Add the SDK lib directory to the system PATH, for example, add `C:\AppDynamics\appdynamics-cpp-sdk\lib to %PATH%`.
4. Configure your Visual Studio project to have the include path and link the AppDynamics library in **Open Project** > **Properties**.
    a. From the sidebar, unfold **C/C++** and choose **General**. In the new pane, change **Additional Include Directories** to add: `C:\AppDynamics\appdynamics-cpp-sdk\include`.
    b. From the sidebar, unfold **Linker** and choose **Input**. In the new pane, change **Additional Dependencies** to add: `C:\AppDynamics\appdynamics-cpp-sdk\lib\appdynamics.lib`.
5. Modify your program to use the AppDynamics C/C++ SDK. See C/C++ SDK Reference and Use the C/C++ SDK.

Install the CCPP SDK on Linux

Related pages:

- Download AppDynamics Software
- Install the C/C++ SDK on Windows

To install the C/C++ SDK on Linux:

1. Download the C/C++ SDK from the AppDynamics Downloads Portal.
2. Extract the SDK to the desired SDK home directory location, such as in `/opt`:

   ```
   sudo tar xvzf appdynamics-sdk-native-64bit-linux-VERSION.tar.gz -C /opt
   ```

   Replace `VERSION` in the file name with the version of the downloaded SDK.
3. Update your build scripts to be able to access the AppDynamics SDK library files:
   a. Add `/opt/appdynamics-cpp-sdk/lib` to your library path. For example, `-L/opt/appdynamics-cpp-sdk/lib`.
   b. Add `-lappdynamics` to your link options.
4. Modify your program to use the AppDynamics C/C++ SDK. See C/C++ SDK Reference and Use the C/C++ SDK.
5. Ensure that the `ulimit` setting for open file descriptors on the operating system is set to a sufficient number. At a minimum, you will need two file descriptors for each active thread that calls the C++ SDK `appd_sdk_init()` function.

Go SDK

Related pages:

- SELinux Installation Issues

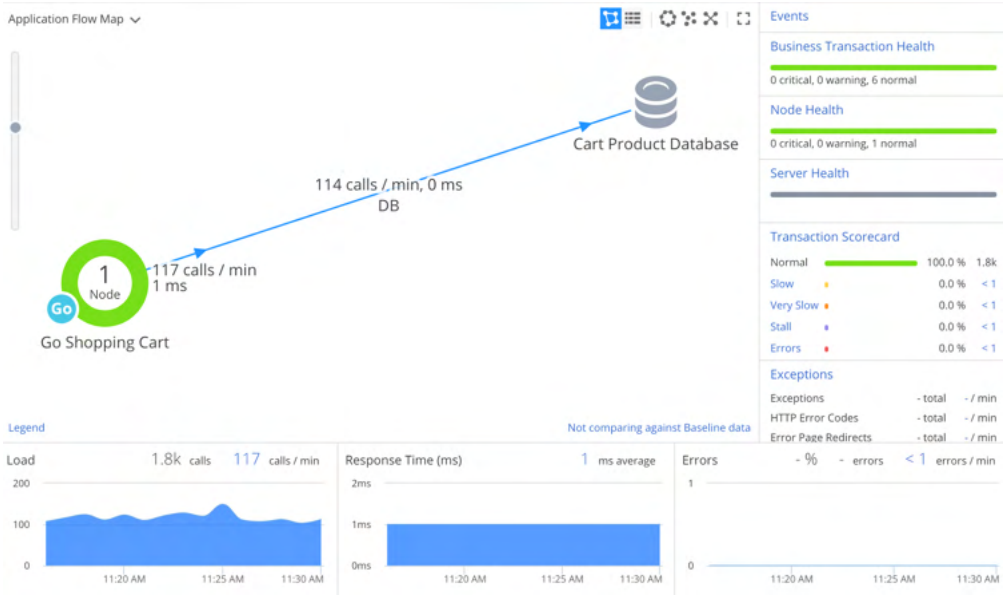
This page describes how to install and instrument the app server agent for Go applications.

AppDynamics monitors applications that have been created with the Go SDK.

The Go SDK can:

- Trace transactions through Go tiers
- Surface Go runtime errors
- Report backend calls made by Go services

Once the agent connects to the Controller, you can see flow maps, key performance indicators, errors, and more for the Go tier.



Limitations

The Go SDK uses cGo in the background. In Go, when you are calling out to C/C++, the code converts the current Goroutine to an OS thread. This is a limitation of the runtime, not the Go SDK. If you are running the agent with many Go routines and make Go SDK calls within the Go routines in your application code, there may be a risk of reaching the OS thread limit. If you encounter this limitation, contact AppDynamics Support.

Go Language Supported Environments

Go Supported Platforms

Language Support

All Go language versions are currently supported.

Operating Systems

- Any Linux distribution based on `glibc` >= 2.5, and the x86 32-bit or x86 64-bit architecture
- Mac OS X >= 10.8

Using Go SDK

On This Page:

This page provides instructions and an overview of the GO SDK, which provides routines for creating and managing Business Transactions, Transaction Snapshots, backends, and exit calls in AppDynamics.

Before You Begin

Before instrumenting your applications with the Go SDK, you will need the current version of the Go SDK. See Go Language Agent and Download AppDynamics Software to install the SDK.

Import the AppDynamics Package to the Application

After downloading the SDK, you are ready to add AppDynamics instrumentation to your Go application. The first step is to import the AppDynamics package:

```
import appd "appdynamics"
```

Initialize the Controller Configuration

Controller information settings permit the SDK to connect to the Controller. Some settings are required for all applications while others are required only for certain types of application environments. For example, if the SDK needs to connect to the Controller via an internal proxy in your network, set up the connection settings for the proxy. See Go SDK Reference for a complete list of settings and which ones are required.

Assign values to the required settings in your application. For example, to set the Controller connection information:

```
cfg := appd.Config{}

cfg.AppName = "exampleapp"
cfg.TierName = "orderproc"
cfg.NodeName = "orderproc01"
cfg.Controller.Host = "my-appd-controller.example.org"
cfg.Controller.Port = 8090
cfg.Controller.UseSSL = false
cfg.Controller.Account = "customer1"
cfg.Controller.AccessKey = "secret"
cfg.InitTimeoutMs = 1000  // Wait up to 1s for initialization to finish
```

Notice the `InitTimeoutMs` field. Once you initialize the configuration, you pass the configuration object to the call to initialize the agent via `InitSDK()`. The `InitTimeoutMs` field can have these values:

- When set to `0`, the default, the `InitSDK()` function operates as an asynchronous action, so that the initialization call does not block your program.
- Setting the value to `-1` instructs the program to wait indefinitely until the controller configuration is received by the agent, that is, the `InitSDK()` method returns control. This is useful when you want to capture short-running Business Transactions that occur at application startup and you do not mind the delay of waiting for the Controller to send the configuration.
- Alternatively, set it to a specific number of milliseconds to wait.

If you use a multi-tenant Controller (SaaS or on-premises multi-tenant), you need to create the context for the multi-tenant environments using the `AddAppContextToConfig()` method. You can then pass the context as a parameter to methods for performing particular operations, such as starting a Business Transaction or adding custom metrics. See Go SDK Reference for `AddAppContextToConfig()` and related methods.

Initialize the Agent

Initialize the agent by passing the configuration structure to `InitSDK()` in your main function.

If `InitSDK()` returns `nil`, the agent is initialized successfully. If an error returns, it is likely because the agent could not reach the Controller.

For example, to initialize the SDK:

```
if err := appd.InitSDK(&cfg); err != nil {
        fmt.Printf("Error initializing the AppDynamics SDK\n")
} else {
        fmt.Printf("Initialized AppDynamics SDK successfully\n")
}
```

Create Business Transactions

Define a Business Transaction by enclosing the code that constitutes the request you want to monitor between `StartBT()` and `EndBT()`calls. `StartBT()` returns a handle to use in subsequent routines that affect that business transaction.

If you are creating a Business Transaction that correlates with an upstream Business Transaction, pass the correlation header of the upstream transaction so the new transaction that you are creating can correlate with it. See Go SDK Reference for `GetExitcallCorrelationHeader()`. If the transaction does not need to correlate with another transaction, pass an empty string for the correlation header parameter.

You can optionally store the Business Transaction handle in the global handle registry with a GUID for easy retrieval later using `StoreBT()`. Retrieve the handle from the global handle registry `using GetBT()`.

For example, to set a Business Transaction:

```
// start the "Checkout" transaction
btHandle := appd.StartBT("Checkout", "")

// Optionally store the handle in the global registry
appd.StoreBT(btHandle, my_bt_guid)
...

// Retrieve a stored handle from the global registry
myBtHandle = appd.GetBT(my_bt_guid)

// end the transaction
appd.EndBT(btHandle)
```

Between starting and ending the transaction, you can perform operations such as adding errors to the business transaction, defining the transaction snapshot attributes, adding backends and exit calls, and so on.

When the business transaction ends via a call to `EndBT()`, the agent stops reporting metrics for the business transaction.

Add Business Transaction Errors

Use `AddBTError()` to report Business Transaction errors. If you set this function's `markBTAsError` parameter, the transaction is reported as an Error Transaction when the error occurs.

The SDK provides constants classifying the error level as `APPD_LEVEL_NOTICE`, `APPD_LEVEL_WARNING` or `APPD_LEVEL_ERROR`.

Manage Business Transaction Snapshots

The agent automatically creates Transaction Snapshots when monitoring a Business Transaction. Transaction snapshots are useful for troubleshooting poor performance because they describe instances of the business transaction at certain points in time.

Other than creating the Business Transaction, you do not have to modify anything to create these snapshots, but you can add calls to:

- Determine if a snapshot is being taken
- Provide additional data in a snapshot
- Set the URL for a snapshot

Determine if the Agent is Taking a Snapshot Now

Due to high cost, the agent does not constantly collect snapshots. By default, it collects a snapshot every ten minutes, but this schedule is configurable. See Configure Snapshot Periodic Collection Frequency.

You can determine if a snapshot is happening using `IsBTSnapshotting()`, which returns `true` if the agent is collecting a snapshot. This method avoids wasted overhead in collecting user data for a snapshot or setting the snapshot URL if no snapshots are being collected.

Add Business Transaction User Data

You can optionally add data to transaction snapshots. For example, you might want to know which users are getting a lot of errors, or from which regions users are experiencing slow response times, or which methods are slow. In the Controller UI, the data appears in the USER DATA tab of the transaction snapshot.

If a snapshot is occurring, use `AddUserDataToBT()` passing a key and value for the data that you want the snapshot to collect.

Add Snapshot URL

If a snapshot is occurring, you can set a URL for the current snapshot using `SetBTURL()`.

Set Snapshot Example

```
func setSnapshotAttributes(bt appd.BtHandle, key, value string) {
        if appd.IsBTSnapshotting(bt) {
                appd.AddUserDataToBT(bt, key, value)
                appd.SetBTURL(bt, "user/login")
        }
}
```

Create Backends

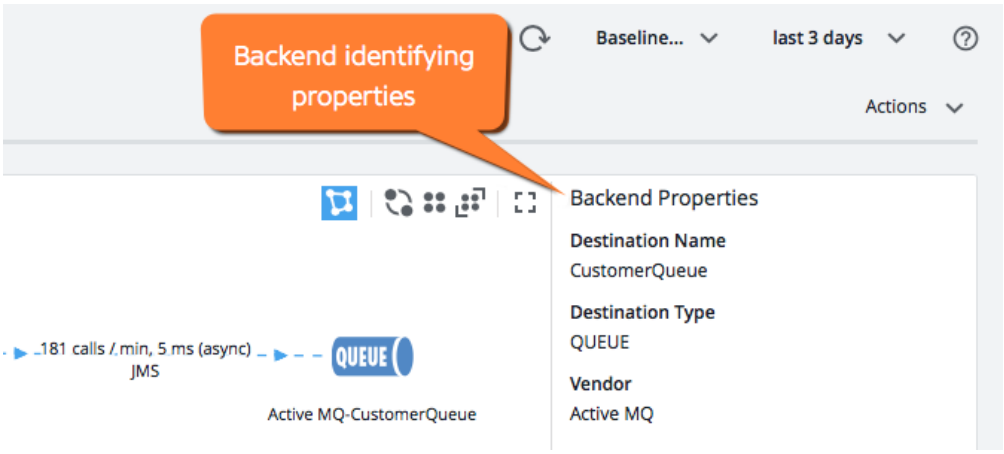
A backend is a database or a remote service such as a message queue, HTTP service, or cache service that your application uses. A backend component is not itself monitored by the application agent, but the agent monitors calls to it from instrumented servers. You need to create backends in the instrumented environment so that the agent registers them. Creating and adding the backend to the instrumented application involves:

- Naming the backend
- Setting its identifying properties
- Optionally, configuring how the backend is presented in the Controller UI

Identifying Properties

A backend has identifying properties. These vary depending on the type of the backend and the types of information that you want to display. The Controller displays identifying properties in backend dashboards. You must set at least one identifying property for the type of any backend that you plan to add.

The following shows the backend properties in the Controller UI for an Oracle database:



Resolve to a Tier

By default the Controller doesn't display a detected backend as a separate entity in flow maps, but the agent reports its metrics as part of the downstream tier. If you want to display the backend as a separate component and not resolved to the tier, set `resolve` to `false`. See Resolve Remote Services to Tiers.

Backend Example

The following listing shows an example of setting up a database backend.

```
backendName := "Cart Product Database"
backendType := "DB"
backendProperties := map[string]string {
        "DATABASE": "sqlite3",
}
resolveBackend := false

appd.AddBackend(backendName, backendType, backendProperties, resolveBackend)
```

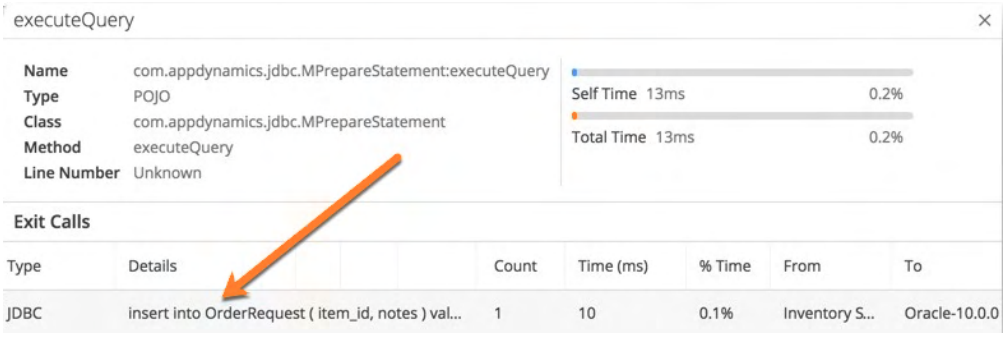
Manage Exit Calls

When an application makes a call to another component, such as a detected backend or another application server, the agent reports metrics on those calls.

Define an exit call by enclosing the code that constitutes the exit call between `StartExitCall()` and `EndExitcall()` calls. `StartExitcall()` returns a handle to use in subsequent routines that affect that exit call. An exit call occurs in the context of a Business Transaction.

You can optionally store the exit call handle in the global handle registry with a guid for easy retrieval later using `StoreExitcall()`. Retrieve the handle from the global handle registry `using GetExitcall()`.

You can optionally add details to an exit call as any arbitrary string with `SetExitcallDetails()`. The details are reported in the exit call details in the transaction snapshots in the Controller UI:



You can also add errors to an exit call using `AddExitcallError()`. Use the enum for the error levels. You can also add an error message.

Simple Exit Call Example

```
// start the exit call to backendName
ecHandle := appd.StartExitcall(btHandle, backendName)

...

// optionally store the handle in the global registry
appd.StoreExitcall(ecHandle, my_ec_guid)

...
// retrieve a stored handle from the global registry
myEcHandle := appd.GetExitcall(my_ec_guid)

// set the exit call details
if err := appd.SetExitcallDetails(myEcHandle, "Exitcall Detail String"); err != nil {
        log.Print(err)
}

// add an error to the exit call
appd.AddExitcallError(myEcHandle, appd.APPD_LEVEL_ERROR, "exitcall error!", true)

// end the exit call
appd.EndExitcall(myEcHandle)
```

Correlate with Other Business Transactions

A correlation header contains the information that enables the agents to continue the flow of a Business Transaction across multiple tiers.

An SDK agent can correlate with other SDK agents as well as other AppDynamics agents, such as Java, .NET, or PHP, that perform automatic correlation for certain types of entry and exit points.

Correlate with an Upstream Tier

When your instrumented process receives a continuing transaction from an upstream agent that supports automatic correlation:

1. Using a third-party `Header.Get()` function, extract the header named `APPD_CORRELATION_HEADER_NAME` from the incoming HTTP payload.
2. Pass the header to `StartBT()`. For example:

```
hdr := req.Header.Get(appd.APPD_CORRELATION_HEADER_NAME)
bt := appd.StartBT("Fraud Detection", hdr)
```

If the header retrieved by the `Header.Get()` function is valid, the business transaction started by the `StartBT()` call will be a continuation of the Business Transaction started by the upstream service.

Correlate with a Downstream Tier

The downstream agent is watching for a correlation header named `singularityheader` in the HTTP payload.

If your SDK agent is making an exit call to a downstream agent that supports automatic correlation:

1. Set the name of the correlation header using `APPD_CORRELATION_HEADER_NAME`.
2. Begin the exit call.
3. Retrieve the correlation header from the exit call using the `GetExitcallCorrelationHeader()` function.
4. Using third-party HTTP functions, prepare an HTTP POST request.
5. Inject a header named `APPD_CORRELATION_HEADER_NAME` with the value of the correlation header retrieved in Step 3 into the outgoing payload of the HTTP request.
6. Make the request. For example:

```
inventoryEcHandle := appd.StartExitcall(btHandle, "Inventory DB")
hdr := appd.GetExitcallCorrelationHeader(inventoryEcHandle)

client := &http.Client{
        CheckRedirect: redirectPolicyFunc,
}

req, err := http.NewRequest("POST", "https://inventory/holds/xyz", nil)
// ...
req.Header.Add(appd.APPD_CORRELATION_HEADER_NAME, hdr)
resp, err := client.Do(req)
// ...

...
```

Terminate the Agent

Terminate the agent right before the application exits:

```
appd.TerminateSDK()
```

Go SDK Reference

Use the AppDynamics Go SDK to instrument Google Go applications. The API includes functions for creating business transactions, transaction backends, exit points, and so on. See Use Go SDK.

Basic Types

The AppDynamics Go SDK defines these opaque types:

- `BtHandle`: A handle to an active business transaction
- `ExitcallHandle`: A handle to an active exit call

Config Struct

The `Config` struct contains settings used by the agent to connect to the AppDynamics Controller. The structure serves an equivalent function of the agent configuration files described in Agent-to-Controller Connections.

The `Config` struct is defined as follows:

```
type Config struct {
    AppName, TierName, NodeName string
    Controller Controller
    InitTimeoutMs int
    Initialized uint // a special field used by the underlying AppDynamics libraries. Do not use.
    Logging LoggingConfig
    UseConfigFromEnv bool
    EnvVarPrefix string
}
```

The structure is made up of these fields:

- `AppName`: Name of the business application to which this node belongs.
- `TierName`: Name of the tier to which this node belongs. A tier is a group of identical or similar nodes. An *originating tier* is the tier that receives the first request of a business transaction. A *downstream tier* is a tier that is called from another tier, thus continuing a business transaction.
- `NodeName`: The monitored application server to be monitored by this agent.
- `Controller`: Connection settings for the Controller. This struct is defined as:

  ```
  type Controller struct {
      Host string
      Port uint16
      Account, AccessKey string
      UseSSL bool
      HTTPProxy HTTPProxy
      CertificateFile
      CertificateDir
  }
  ```

  The `Controller` struct contains these fields:

  - `Host`: Controller host.
  - `Port`: Controller port; defaults to 8080.
  - `Account`: Name of your AppDynamics account.
  - `Access_key`: Key to your AppDynamics account. To find your access key, click the settings (gear) icon in the upper right corner of the AppDynamics UI, then click License.
  - `UseSSL`: Enable SSL communication with the controller; the default is false. Set to true to enable. If you set UseSSL to true, the default certificate, ca-bundle.crt, is used. To use your own certificate, specify your certificate file and directory in the CertificateFile and CertificateDir parameters.
  - `HTTPProxy`: If the agent needs to connect to the Controller via a local HTTP proxy, use this struct to specify connection settings for the proxy.
  - `CertificateFile` : The file name of the SSL certificate.
  - `CertificateDir` – The directory where the certificate is located.
- `InitTimeoutMs`: The initialization function, `InitSDK` relies on the Controller configuration to start business transactions. To prevent the initialization function from blocking your application, you can use this setting. It lets you instruct `InitSDK` whether to wait for the Controller configuration and if so how long to wait before starting business transactions. Valid values are:

  - N: Wait for up to N milliseconds for the Controller configuration.
  - 0: Do not wait for the Controller configuration. This is the default.
  - -1: Wait indefinitely until the Controller configuration is received by the agent.
- `Logging:` Logging settings for the controller. This struct is defined as:

```
const (
    APPD_LOG_LEVEL_DEFAULT LogLevel = iota
    APPD_LOG_LEVEL_TRACE
    APPD_LOG_LEVEL_DEBUG
    APPD_LOG_LEVEL_INFO
    APPD_LOG_LEVEL_WARN
    APPD_LOG_LEVEL_ERROR
    APPD_LOG_LEVEL_FATAL
)
type LoggingConfig struct {
    BaseDir             string
    MinimumLevel        LogLevel
    MaxNumFiles         uint
    MaxFileSizeBytes    uint
}
```

The `Logging` struct contains these fields:

  - `BaseDir:` The absolute path to the directory where log files are written. If left empty, the default is `/tmp/appd`.
  - `MinimumLevel`: One of the APPD_LOG_LEVEL_xxx constants that are shown above. The default is `APPD_LOG_LEVEL_INFO`.
  - `MaxNumFiles`: The maximum number of logging files to store on disk. Once this maximum is hit, older logs are rotated out.
  - `MaxFileSizeBytes`: The maximum size of the log files before they are rotated.
- `UseConfigFromEnv:` Set to `true` if you want the SDK to check for any configuration environment variables and use those configuration values for initialization. Note that because this happens on initialization, the environment variable settings override the configuration you set in your program.
- `EnvVarPrefix:` If `UseConfigFromEnv` is set to `true`, use this property to specify the prefix to use for environment variable names. The default prefix is `APPD_SDK`.

If the agent needs to connect to the Controller via a local HTTP proxy server, you need to configure the settings for the proxy server. You can do so using the `HTTPProxy` struct, defined as follows:

```
type HTTPProxy struct {
    Host string
    Port uint16
    Username, PasswordFile string
}
```

The `HTTPProxy` struct has the following fields:

- `Host:` Hostname or IP address of the HTTP proxy server
- `Port:` Port of the proxy server
- `Username:` Proxy server user name
- `PasswordFile:` Proxy server password file

ContextConfig Struct

You use the `ContextConfig` struct for calls that apply to multi-tenant Controller environments.

```
type ContextConfig struct {
    AppName string
    TierName string
    NodeName string
}
```

The `ContextConfig` struct has these fields:

- `AppName:` Name of the business application to which this node belongs.
- `TierName:` Name of the tier to which this node belongs. A tier is a group of identical or similar nodes. An *originating tier* is the tier that receives the first request of a business transaction. A *downstream tier* is a tier that is called from another tier, thus continuing a business transaction.
- `NodeName`: The monitored application server to be monitored by this agent.

AddAppContextToConfig

Add application context to the AppDynamics configuration for a multi-tenant Controller.

Format

```
func AddAppContextToConfig(cfg *Config, context string, contextCfg *ContextConfig) error
```

Parameters

- `cfg`: AppDynamics configuration object.
- `context:` A unique identifier used to refer to this context.
- `contextCfg:` An AppDynamics context configuration object indicating the business application, tier, and node name for this context.

InitSDK

Initialize the AppDynamics Go SDK. An instrumented application must call this function once, preferably during application startup.

Format

```
func InitSDK(cfg *Config) error
```

Parameter

`cfg:` AppDynamics configuration settings that enable communication between the agent and the Controller.

Returns

nil on success, otherwise an error value

TerminateSDK

Stop the AppDynamics Go SDK. Ends all active business transactions for this agent and stops agent metric reporting to the Controller.

Format

```
func TerminateSDK()
```

StartBT

Start a business transaction or continue an existing transaction.

Keep in mind that each application is limited to 200 registered business transactions, and each agent is limited to 50 registered business transactions. Unlike transactions discovered by other types of agents, business transactions that exceed the limit that are reported by the Go SDK are not included in the all other traffic grouping. This means that it's up to you to ensure that your agent does not create excessive business transactions. Use care to ensure that your implementation does not introduce the possibility of business transaction explosion.

Format

```
func StartBT(name, correlation_header string) BtHandle
```

Parameters

- `name:` The name for the business transaction. In the case of a continuing transaction in the current business application with a valid correlation header, the SDK uses the name from the header. Do not use the following characters in transaction names: { } [ ] | & ;
- `correlation_header:` A correlation header if this is a continuing transaction, else NULL. If specified, the correlation header has been generated by another AppDynamics agent and made available to this agent as a string. The correlation header provides information to enable this transaction to correlate with an upstream transaction.

Returns

An opaque handle for the business transaction that was started.

StartBTWithAppContext

Start a business transaction or continue an existing transaction in a multi-tenant Controller environment.

Format

```
func StartBTWithAppContext(context, name, correlation_header string) BtHandle
```

Parameters

- `context:` The application context name that this business transaction belongs to.
- `name:` The name for the business transaction. In the case of a continuing transaction in the current business application with a valid correlation header, the SDK uses the name from the header. Do not use the following characters in transaction names: { } [ ] | & ;
- `correlation_header:` A correlation header if this is a continuing transaction, else NULL. If specified, the correlation header has been generated by another AppDynamics agent and made available to this agent as a string. The correlation header provides information to enable this transaction to correlate with an upstream transaction.

EndBT

End the given business transaction.

Format

```
func EndBT(bt BtHandle)
```

Parameter

`bt:` The handle to the business transaction to end.

GetBT

Get a BT handle associated with the given guid by `appd_bt_store`.

Format

```
func GetBT(guid string) BtHandle
```

Parameter

`guid:` The globally unique identifier that was passed to `appd_bt_store`.

Returns

The handle to the business transaction associated with the given guid.

In the following cases the SDK logs a warning and returns a handle that you may safely use in other API functions but that will cause these functions to immediately return without doing anything:

- The SDK doesn't find a handle associated with the guid
- The call ended before the SDK could retrieve the handle

IsBTSnapshotting

Reports whether the agent is currently taking a transaction snapshot. Useful before calling `AddUserDataToBT` or `SetBTURL`, since those potentially expensive functions would do nothing if called when a snapshot is not being taken.

Format

```
func IsBTSnapshotting(bt BtHandle) bool
```

Parameter

`bt:` The handle to the business transaction to check for snapshotting.

Returns

true if the given business transaction is taking a snapshot. Otherwise, false.

SetBTURL

Set the URL for a snapshot, if one is being taken. It is safe to call this function when a snapshot is not occurring. When the given business transaction is not snapshotting, this function immediately returns. However, if extracting the data to pass to this function is expensive, you can use `IsBTSnapshotting` to check if the business transaction is snapshotting before extracting the data and calling this function. The `url` argument data should be either 7-bit ASCII or UTF-8.

Format

```
func SetBTURL(bt BtHandle, url string)
```

Parameters

- `bt:` The business transaction to add the user data to, if it is taking a snapshot.
- `url:` The value of the URL for the snapshot as 7-bit ASCII or UTF-8.

StoreBT

Store a BT handle in a global registry to retrieve later with `GetBT`. This is convenient when you need to start and end a business transaction in separate places, and it is difficult to pass the handle to the business transaction through the parts of the code that need it.

When the business transaction is ended, the handle is removed from the global registry.

Format

```
func StoreBT(bt BtHandle, guid string)
```

Parameters

- `bt:` The BT to store.
- `guid:` A globally unique identifier to associate with the given BT.

AddBTError

Add an error to a business transaction. Errors are reported as part of the business transaction. However, you can add an error without marking the business transaction as an error (e.g., for non-fatal errors).

Format

```
func AddBTError(bt BtHandle, level ErrorLevel, message string, mark_bt_as_error bool)
```

Parameters

- `bt:` The handle to the business transaction to which the error is added.
- `level:` The error level, from the following:
    - `APPD_LEVEL_NOTICE`
    - `APPD_LEVEL_WARNING`
    - `APPD_LEVEL_ERROR`
- `message:` Error message for this error.
- `mark_bt_as_error:` If true, the business transaction experiencing this error is marked as an error transaction. In this case, the business transaction is counted only as an error transaction. It is not also counted as a slow, very slow or stalled transaction, even if the transaction was also slow or stalled. If false, the business transaction is not marked as an error transaction.

AddUserDataToBT

Attaches user data to transaction snapshots generated for the specified business transaction.

The user data is added to the business transaction but reported only when a transaction snapshot is occurring. In the Controller UI, the user data appears in the USER DATA tab of the transaction snapshot details.

It is safe to call this function when a snapshot is not occurring. When the specified business transaction is not taking a snapshot, this function immediately returns.

If extracting the data to pass to this function is expensive, you can use `IsBTSnapshotting` to check if the business transaction is actually taking a snapshot before extracting the data and calling this function.

The data in the `value` argument should be either 7-bit ASCII or UTF-8.

Format

```
func AddUserDataToBT(bt BtHandle, key, value string)
```

Parameters

- `bt` –The business transaction to add the user data to, if it's taking a snapshot.
- `key` – The name of the user data to add to the snapshot as 7-bit ASCII or UTF-8.
- `value` – The value of the user data to add to the snapshot as 7-bit ASCII or UTF-8.

AddBackend

Add a backend to the business application.

This function fails if the type of the backend being added does not have at least one identifying property.

Format

```
func AddBackend(name, backendType string, identifyingProperties map[string]string, resolve bool) error
```

Parameters

- `name:` The name of the backend. Must be unique to the Go SDK instance.
- `backendType:` The type of the backend, from these options:
    - `APPD_BACKEND_HTTP`
    - `APPD_BACKEND_DB`
    - `APPD_BACKEND_CACHE`

    - APPD_BACKEND_RABBITMQ
    - APPD_BACKEND_WEBSERVICE
    - APPD_BACKEND_JMS
- identifyingProperties: A map of key/value pairs that contain the backend's identifying properties. The properties uniquely identify backend. In the Controller, these properties appear in the upper right panel of the backend dashboards. You must set at least one identifying property when you add a backend of one of the built-in exit call types. The valid properties vary per exit call type as indicated below.

Exit Call Type	Valid Identifying Properties
APPD_BACKEND_HTTP	"HOST", "PORT", "URL", "QUERY STRING"
APPD_BACKEND_DB	"HOST", "PORT", "DATABASE", "VENDOR", "VERSION"
APPD_BACKEND_CACHE	"SERVER POOL", "VENDOR"
APPD_BACKEND_RABBITMQ	"HOST", "PORT", "ROUTING KEY", "EXCHANGE"
APPD_BACKEND_WEBSERVICE	"SERVICE", "URL", "OPERATION", "SOAP ACTION", "VENDOR"
APPD_BACKEND_JMS	"DESTINATION", "DESTINATIONTYPE", and "VENDOR"

  For example, a key PORT would have a value of 3304.
- resolve: Set the property to false to prevent a downstream agent from resolving as this backend.
  If true, the default, an agent that detects a correlation header for an unresolved backend resolves itself as that backend. However, if the backend is actually an uninstrumented tier that is passing through the correlation header—for example, a message queue or proxy—you may wish the backend to show up distinct from the tier that it routes to. If false, correlation headers generated for exit calls to this backend will instruct downstream agents to report as distinct from the backend.

  For example, if you have Tier A connecting to uninstrumented Backend B, which routes to instrumented Tier C, the setting affects the flow map as follows:

    - True, the flow map will be A > C.
    - False, the flow map will be A > B > C.

Returns

nil on success, otherwise an error value

AddExitcallError

Add an error to the exit call.

Format

```
func AddExitcallError(exitcall ExitcallHandle, level ErrorLevel, message string, mark_bt_as_error bool)
```

Parameters

- exitcall: Handle to the exit call.
- level: The level of this error, from the following:
    - APPD_LEVEL_NOTICE
    - APPD_LEVEL_WARNING
    - APPD_LEVEL_ERROR
- message: Error message for this error.
- mark_bt_as_error: If true, the business transaction making the exit call that is experiencing this error is marked as an error transaction. In this case, the business transaction is counted only as an error transaction. It is not also counted as a slow, very slow or stalled transaction, even if the transaction was also slow or stalled. If false, the business transaction is not marked as an error transaction.

StartExitcall

Start an exit call to the specified backend as part of a business transaction.

Format

```
func StartExitcall(bt BtHandle, backend string) ExitcallHandle
```

Parameters

- bt: Handle to the business transaction making the exit call.
- backend:The destination backend of the exit call. AppDynamics does not automatically detect backends for Go applications, so you must specify the destination backend.

Returns

An opaque handle to the exit call that was started.

EndExitcall

Complete the exit call.

Format

```
func EndExitcall(exitcall ExitcallHandle)
```

Parameter

`exitcall:` Handle to the exit call being ended.

GetCRollupType

Specify how to roll up values for the metric over time.

Format

```
func GetCRollupType(rollUp RollupType)
```

Parameters

- `rollUp:` The approach for rolling up metrics. The following roll-up options are valid:
  - `APPD_TIMEROLLUP_TYPE_AVERAGE`: The average of all metric values across each node in the tier.
  - `APPD_TIMEROLLUP_TYPE_SUM`: Sum of all reported values in the minute.
  - `APPD_TIMEROLLUP_TYPE_CURRENT`: The last reported metric in the minute.

GetCClusterRollupType

Specify how to aggregate metric values for the tier (a cluster of nodes).

Format

```
func GetCClusterRollupType(rollUp ClusterRollupType)
```

Parameters

- `rollUp:` The approach for rolling up metrics. The following roll-up options are valid:
  - `APPD_CLUSTERROLLUP_TYPE_INDIVIDUAL`: The average of all metric values across each node in the tier.
  - APPD_CLUSTERROLLUP_TYPE_COLLECTIVE: The sum of all metric values for all the nodes in the tier.

GetExitcall

Get a handle to an exit call associated with a guid via `StoreExitcall`.

Format

```
func GetExitcall(guid string) ExitcallHandle
```

Parameter

`guid:` The globally unique identifier that was passed to `appd_exitcall_store()`.

Returns

The handle to the exit call associated with the given guid.

In the following cases the SDK logs a warning and returns a handle that you may safely use in other API functions but that will cause these functions to immediately return without doing anything:

- The SDK doesn't find a handle associated with the guid
- The call ended before the SDK could retrieve the handle

GetExitcallCorrelationHeader

Get the header for correlating a business transaction.

If a business transaction makes exit calls that you want to correlate across, retrieve the correlation header using this function and inject it into your exit call's payload.

The returned string is freed when the exit call ends. Do not free it yourself.

Format

```
func GetExitcallCorrelationHeader(exitcall ExitcallHandle) string
```

Parameter

`exitcall:` Handle to the exit call.

Returns

On success, returns a 7-bit ASCII string containing the correlation information. You can inject this string into the payload of the exit call. A downstream agent can then extract the header from that payload and continue the business transaction.

On error, returns the default header that prevents downstream business transaction detection.

SetExitcallDetails

Set the details string for an exit call. This can be used, for example, to add the SQL statement that a DB backend has executed as part of the exit call. This data is then visible in the exit calls details UI for the transaction snapshot.

Format

```
func SetExitcallDetails(exitcall ExitcallHandle, details string) error
```

Parameters

- `exitcall:` Handle to the exit call.
- `details:` An arbitrary string to add to the exit call.

Returns

Nil on success, otherwise an error value

StoreExitcall

Store an exit call handle in a global registry for later retrieval with `GetExitcall`. This is useful when you need to start and end a call in separate places, and it is difficult to pass the handle through the parts of the code that need it.

The handle is removed when the exit call, or the BT containing it, ends.

Format

```
func StoreExitcall(exitcall ExitcallHandle, guid string)
```

Parameters

- `exitcall:` The exit call to store.
- `guid:` A globally unique identifier to associate with the given call.

Example

```
ExitcallHandle ec = StartExitcall(bt, "authdb");
StoreExitcall(ec, "login-exit");
```

Install the Go SDK

Related pages:

- Download AppDynamics Software

To install the Go SDK:

1. Download the Go SDK distribution from the AppDynamics Downloads Portal.
2. Extract the Go SDK Zip in your Go workspace.

After installing the Go SDK, you are ready to instrument your Go application with the API.

IBM Integration Bus Agent

The IBM Integration Bus (IIB) Agent allows you to see the activity flow in your IIB instance. You can use the AppDynamics IIB Agent to do the following:

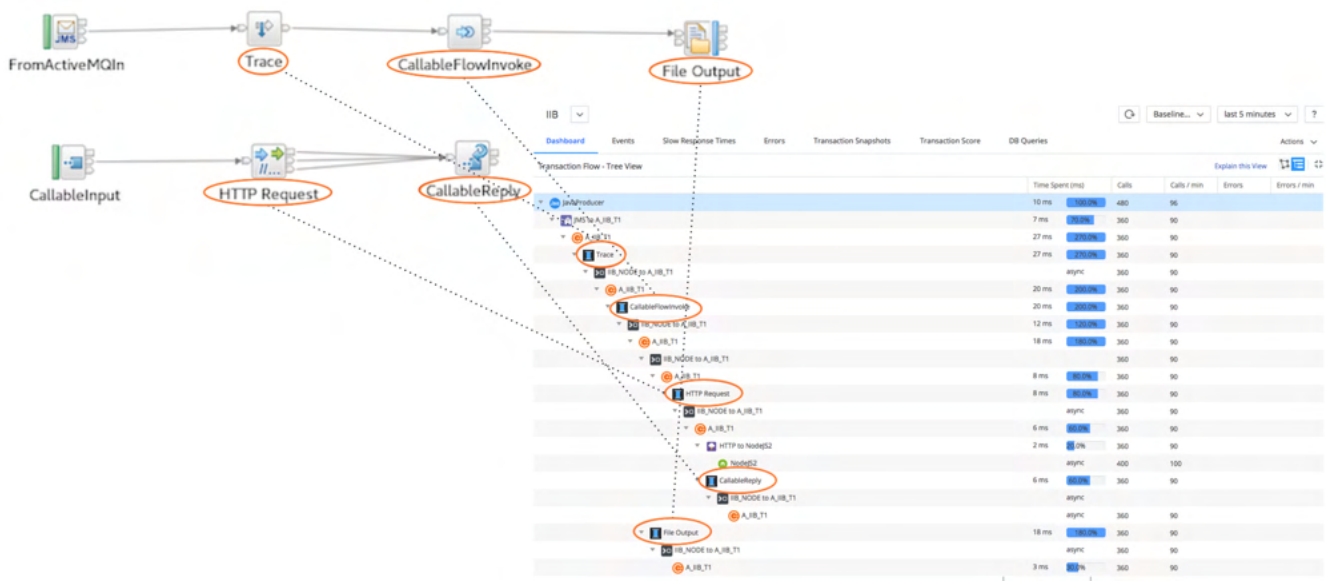
- Correlate business transactions that flow through IIB to identify where time is spent processing transactions in the larger end-to end architecture.
- Measure where business transaction processing time is spent within IIB flows, allowing operations staff and flow developers to identify any bottlenecks within the flows.

> This page describes IBM IIB in the context of it being monitored by AppDynamics. For a full description of IBM IIB and its capabilities, refer to the IBM product documentation.

How It Works

The agent supports inbound and outbound correlation for HTTP, JMS, and IBM MQ messaging nodes. You can identify business transactions that originate in the broker itself by the transaction name, which contains the message flow name and the message flow node name where the BT originated.

To measure where the time is spent within the flows, the agent models each message flow node as a thread (with thread names corresponding to the message flow node name). You can view the per-node timings in the tree view of the Business Transactions dashboard for any business transaction that passed through IIB, as shown below:



The IIB Agent also reports message flow node timing data, represented as thread segments in transaction snapshots, for business transactions that flow through IIB. Message flow node timing data is displayed in the Waterfall View.

Each broker is represented in the AppDynamics flow map as a tier, which is named as you specified in the configuration. Individual broker processes register with AppDynamics as nodes. The node names are generated from the Broker name and the Execution Group name.

IIB Supported Environments

IIB Agent Support

This page provides describes the supported versions, operating systems, and node types of the IIB agent. The AppDynamics IIB agent supports the Linux and AIX operating systems.

Operating Systems

The IIB agent supports these operating systems:

- Linux x86-64
- AIX v7.1 and v7.2

IIB Versions

The IIB Agent supports these versions of IIB for the Linux and AIX operating systems:

Operating System	IIB Version
Linux (Any Linux distribution based on glibc >= 2.5)	• IIB v9 • IIB v10 • ACE v11
AIX 7.1 (minimum patch level 7100-03-09-1717 or 7100-04-04-1717)	• IIB v9 • IIB v10
AIX 7.2	• IIB v9 • IIB v10 • ACE v11 (requires ACE Fix Pack >= v11.0.0.9)

IIB Node Types

The agent can continue business transactions detected upstream at these node types:

- SOAPInput
- HTTPInput
- JMSInput
- MQInput

The agent can detect and tag exit calls for downstream correlation at these node types:

- SOAPRequest
- HTTPRequest
- JMSOutput, JMSReply
- MQOutput, MQReply

> For MQ, we use the MQRFH2 message header to provide correlation. Any applications consuming MQ messages from IIB with the IIB agent must support the MQRFH2 header.

The agent can detect database backend calls for these node types:

- DatabaseRetrieve
- DatabaseRoute

All nodes are represented within AppDynamics Business Transactions as Threads. You can view the per node timings in the tree view of the Business Transaction dashboard and in the transaction snapshots.

Install the IIB Agent

Related pages:

- Download AppDynamics Software
- mqsichangeflowuserexits command
- cciRegisterUserExit
- SELinux Installation Issues

To begin monitoring IIB, you must install the IIB agent.

Install the Agent

1. Download the agent from https://download.appdynamics.com/download/.
2. Extract the IIB agent zip file into your chosen installation directory.

For cURL installation, see Download AppDynamics Software.

Configure the IIB Agent

The page guides you on how to integrate Appdynamics IIB Agent on Linux and AIX. The installation steps for IIB agent for IIB v9 and v10 are the same for Linux and AIX. The installation steps for IIB ACE 11 are provided in a separate tab.

> ACE is currently supported on Linux and AIX v7.2.

1. Specify the settings for the agent-controller communication. In the `controller-info.xml` file, set these properties:
    - `account-name`: Account name of the controller login.
    - `account-access-key`: Access key. This key is used for verification with the controller.
    - `application-name`: Name of the application that the IIB server belongs to.
    - `controller-host`: Hostname of the controller.
    - `controller-port`: Port number of the controller.
    - controller-proxy-host: Host name or IP address of any proxy required for the agent to connect to the controller.
    - controller-proxy-port: Port number of any proxy required for the agent to connect to the controller.
    - controller-proxy-username: Username for authenticating with the proxy.
    - controller-proxy-password: Password for authenticating with the proxy user.
    - controller-proxy-passwordfile: Full path name of a file containing the password for authenticating with the proxy user.
    - `controller-ssl-enabled`: SSL login. Set `1` to enable, `0` to disable.
    - `controller-cert-file`: Full path to the PEM format X509 certificate for SSL.
      See, Enable SSL for the C/C++ SDK for more information on obtaining the file.
    - `log-dir`: Path to the directory containing the IIB agent log files. The default path is `/tmp/appd`. Logs that are written before this configuration are logged in this path.
    - `log-level`: Level of the logs. Set this property to `trace|debug|info|warning|error`. `Error` is the highest priority and `trace` is the lowest priority.
    - `tier-name`: Name of the tier representing the broker.
    - `user-exit`: Exit name of the user. This must be in the alphanumeric format, as provided to the `mqsichangebroker` command.
    - <log-settings> </log-settings>: Log settings of the agent. For more information, see Configure Agent Log File Size.
    - disable-correlation: Enable or disable correlation of specific backends. Set 1 to enable, 0 to disable.
      disable-http-correlation: Set 1 to enable, 0 to disable.
      disable-jms-correlation: Set 1 to enable, 0 to disable.
      disable-mq-correlation: Set 1 to enable, 0 to disable.
    - flow-level-visibility-enabled: Flow level visibility option. Set 1 to enable, 0 to disable. The default value is 0. See IIB Agent Flow Level Visibility.
2. Run the following command to stop the broker. To configure user exits, your broker must be stopped.

    ```
    mqsistop <broker_name>
    ```

3. Install the IIB agent user exit. Ensure that the IIB agent user exit is enabled by default, for all the flows. Perform the steps based on the version applicable to you.

    > The agent starts and stops with the broker, if it is installed on the broker.

4. Run the following command to start the broker.

    ```
    mqsistart <broker_name>
    ```

For IIB 10, you can monitor a subset of your message flows. You can exclude flows that are of less importance from being monitored by running the following command:

```
mqsichangeflowuserexits <broker_name> -e <integrationServerName> -f <MessageFlow> -k <application_name> -i
<user_exit_name>
```

Partial Instrumentation of Brokers

Configure Agent Log File Size

AppDynamics IIB Agent creates a separate log file for each execution group. Therefore, each log continuously grows, even if the execution group is not instrumented.

Hence, a need to stop pushing the updates to the logs for non-instrumented execution groups arises. AppDynamics IIB Agent now allows you to configure the log file size according to your requirement.

The following setting in the configuration file, `controller-info.xml` allows you to configure the log file size of the agent:

```
<log-settings>
<log-setting broker="" execution-group="">
<max-file-size></max-file-size>
</log-setting>
</log-settings>
```

The settings are defined as follows:

Setting	Description
<log-settings></log-settings>	Header tag
<log-setting broker="" execution-group="">	Broker name and the execution group name, which must be strings
<max-file-size> < /max-file-size>	The maximum size of the log file. The format is <number><unit>. For example, 5 MB. Valid units are "MB" or "KB". The accepted range in KB is [200KB, 999KB] and in MB it is [1MB, 999MB]. Every execution group has its own log file.

However, you need to note the following guidelines for the configuration:

- If a broker is not named, the setting applies to all the brokers.
- If an execution group is not named, the setting applies to all the execution groups.
- An execution group name must be accompanied by a broker name.

If you include the log setting, log-rotation is set to 1, else the default value of 5 is considered as the log rotation.

For example, in the following configuration, there are three instances of log setting:

```
<controller-info>
<controller-host></controller-host>
<controller-port></controller-port>
<controller-ssl-enabled></controller-ssl-enabled>
<account-name></account-name>
<account-access-key></account-access-key>
<application-name></application-name>
<tier-name></tier-name>
<user-exit></user-exit>
<log-level></log-level>
<enable-extended-logging></enable-extended-logging>
<controller-cert-file></controller-cert-file>
<controller-cert-dir></controller-cert-dir>
<log-settings>
<log-setting broker="" execution-group="">
<max-file-size>200KB</max-file-size>
</log-setting>
<log-setting broker="Broker1" execution-group="EG1">
<max-file-size>2MB</max-file-size>
</log-setting>
<log-setting broker="Broker1" execution-group="">
<max-file-size>3MB</max-file-size>
</log-setting>
</log-settings>
</controller-info>
```

The order of preference is mentioned in the following table. The preference order is defined as `1 2 3 4`. Hence, if an order is invalid or does not match, then the next valid order is considered.

Order Number	Preferences	Order Template
1	Execution-Group	`<log-setting broker="Broker1" execution-group="EG1">`
2	Broker	`<log-setting broker="Broker1" execution-group="">`
3	Across multiple-broker	`<log-setting broker="" execution-group="">`
4	Default	Agent default is applied which is 10MB

The following are the valid values, for different execution groups when used with the above example configurations.

Broker Name	Execution Group Name	Valid Values (max-file-size)
Broker1	EG1	2MB
Broker1	All other EGs	3MB
All other Brokers	All other EGs	200KB

Troubleshooting

If you experience any functional issues with the agent, review the following information and supply it to the AppDynamics Support.

- The agent log. The IIB agent writes status information into the configured log directory, or `/tmp/appd` if the configuration has not been read at the time the message is logged. The status information can help diagnose issues with the agent.
- The system log. The IIB broker writes status messages into the system log. The messages include information about the loading and operation of user exits, such as the AppDynamics IIB agent.
- Abend files in the IIB errors directory. The default location is `/var/mqsi/common/errors`.
- Core files, if they exist.

If the user-exit name provided in the `controller-info.xml` file is not alphanumeric, the broker cannot load the agent and fails to start.

If you have an issue with the agent or if you want to disable it, perform the following:

1. Stop the broker `mqsistop <broker-name>.`
2. Perform the following based on the version applicable to you.

3. Start the broker `mqsistart <broker-name>.`

Disable the IIB Agent

Turn off MQRFH2 Injection Added by AppDynamics Monitor Agent Tool

For IIB Agents that use the MQ mechanism, the MQRFH2 message header is used to provide correlation to downstream tiers. Any applications consuming MQ messages from an upstream IIB agent must support the MQRFH2 header standards to provide end-to-end correlation.

In a situation where the downstream agent reports the error MQRC_HEADER_ERROR (MQRC 2142) while parsing the message from an IIB agent via MQ, you can perform one of the following:

- Update the downstream consumer to accept the `MQRFH2` header structure by upgrading the downstream software to a version that supports `MQRFH2`.
- Disable MQ correlation in IIB Agent by setting <disable-mq-correlation>1</disable-mq-correlation> in the controller-info.xml file and restart the agent.
- Change the `MQ PROPCTL` property to `NONE` on the destination queue. See https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_7.5.0/com.ibm.mq.mig.doc/q008230_.htm.

Troubleshooting for IBM MQ .NET Clients

AppDynamics Monitor Agent tool adds an additional RFH header to the MQ message before adding the message to the queue. Because of this additional RFH header, IBM MQ .NET Clients may report an `MQRC_HEADER_ERROR (MQRC 2142)` while parsing the message. Following is the sample RFH header that gets added to MQMessage.

```
RFH ….x…″…3…MQSTR …….P…<usr><singularityheader
dt=″string″ >notxdetect
=True</singularityheader></usr>
```

The RFH2 structure begins with <usr><singularityheader dt=”string” >. This RFH header is added by the AppDynamics Monitor Agent tool and not by MQ.

Perform these steps to disable the additional header on the tool:
1. Go to **Configuration > Instrumentation > Backend Detection.**
2. Click on your application name, go to the .NET tab and click **Queues**.
3. Click **Edit Automatic Discovery** and in the popup, clear the **Correlation Enabled** checkbox.

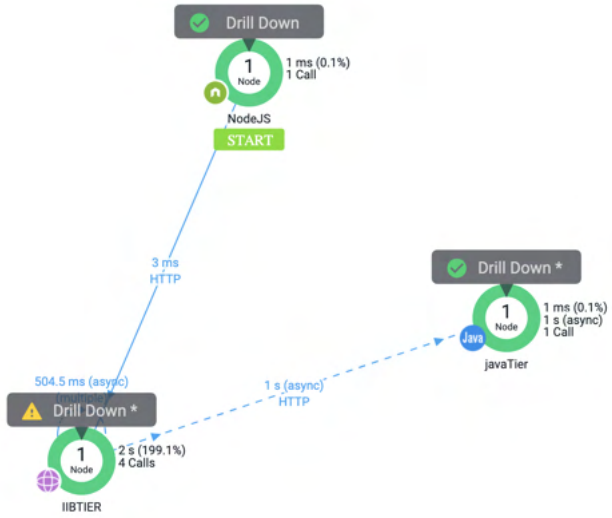
IIB Agent Flow Level Visibility

Flow Level Visibility feature captures the time spent on each message flow and any exit calls made from the message flow. Unlike the existing agent flow, Flow level does not capture the time spent on every node in the message flow. All the other features such as BT correlation, snapshots, exit calls, and so on remain unchanged. The difference is only visible in the Tree View in Business Transaction and Water Fall View in Transaction Snapshots. This difference is demonstrated with an example below:

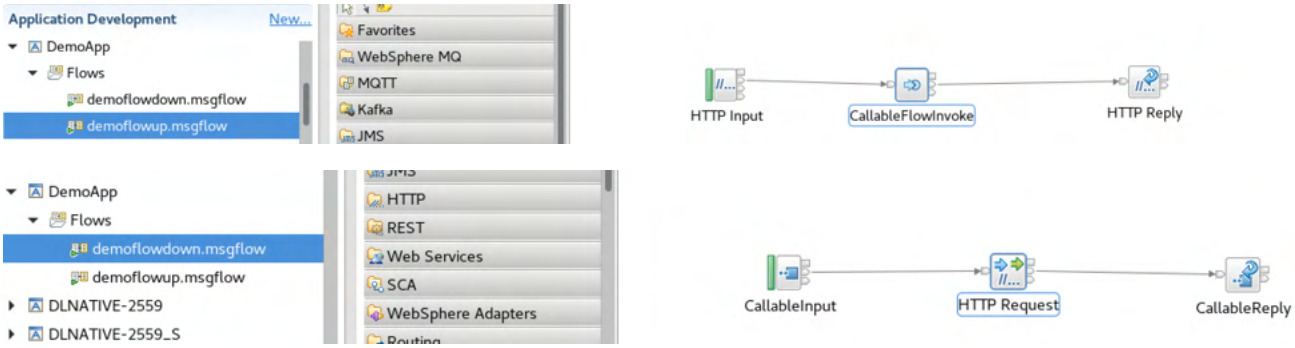
Existing IIB Agent Flow

In the existing IIB Agent default visibility, the agent gathers timings for each node in an IIB flow.

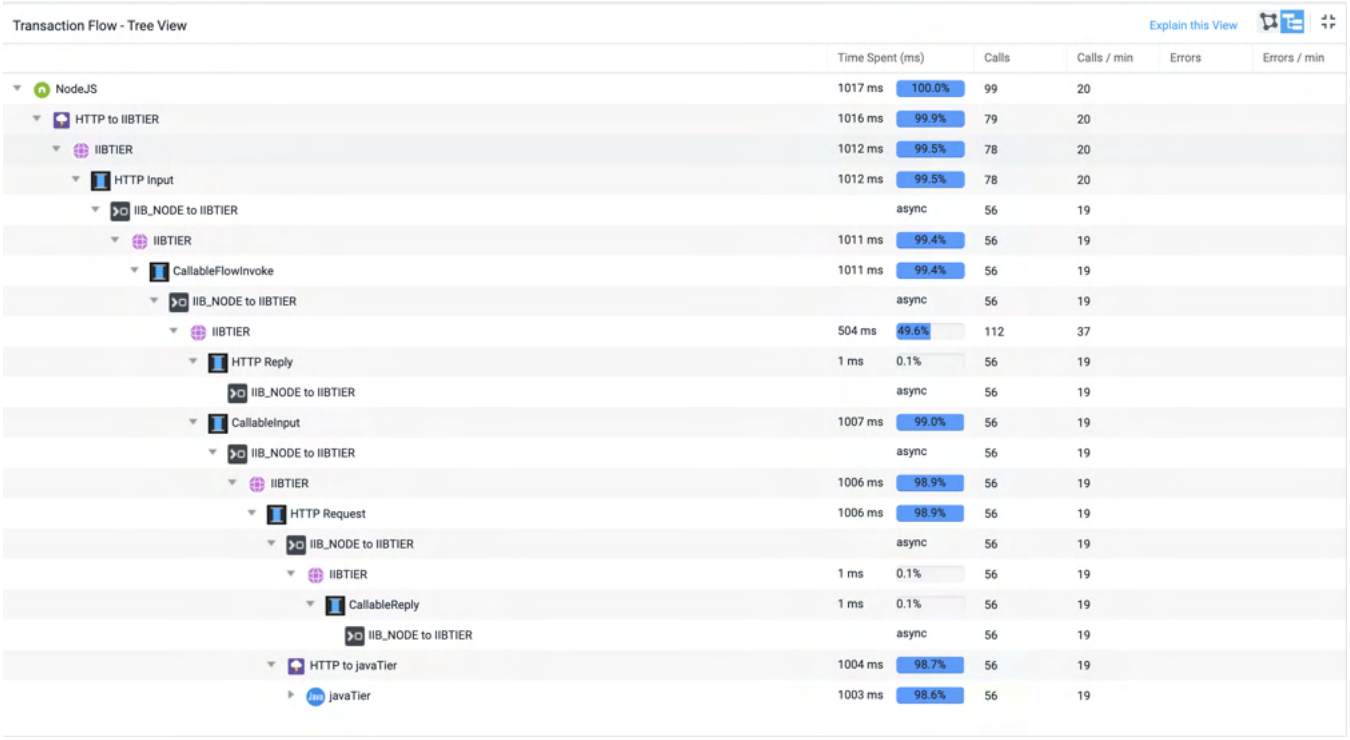
Consider the following illustration that shows the information captured by the existing framework for the flow **NodeJS--->IIB--->Java**:

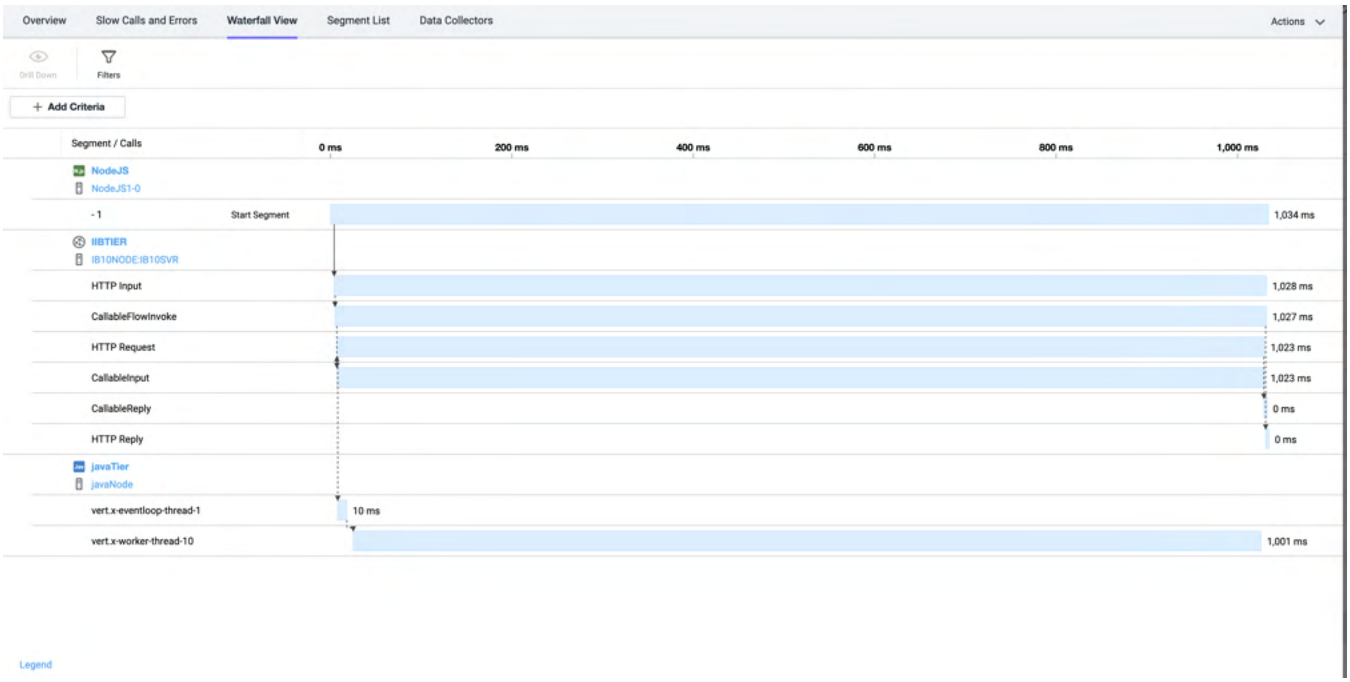


The broker within the IIB tier runs the following two message flows: demoflowup and demoflowdown. demoflowup makes a call to demoflowdown, which in turn, makes a backend call.



The following image displays the business transaction tree view and the transaction snapshot waterfall view for transactions flowing through the IIB application:

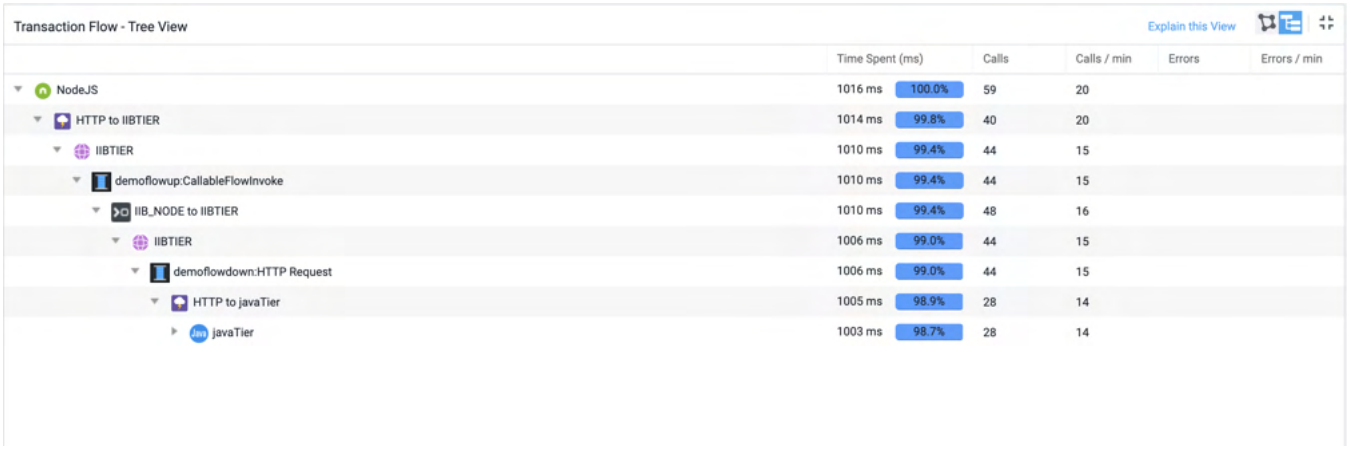


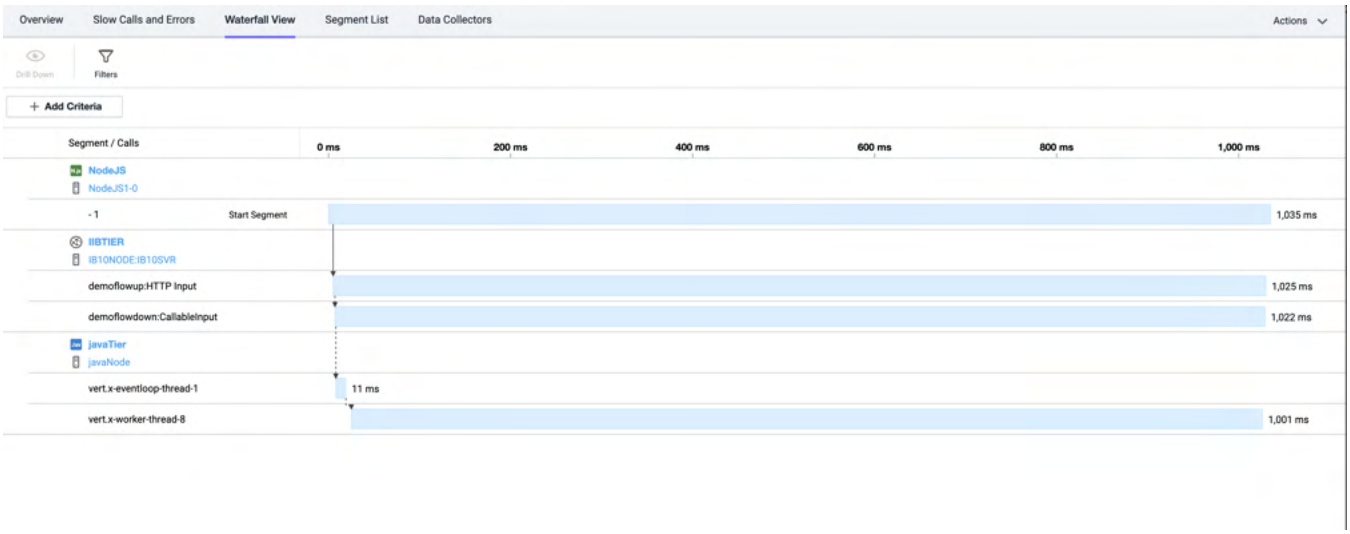


Flow Level Visibility

To reduce the cost of collecting the node level data, a new mode for the agent to provide flow-level visibility is introduced. In this mode, timings are collected only for flow start and end and any external calls made from within a flow. Therefore, the expected CPU cost of instrumenting the flows with the agent is reduced, especially for complex flows.

The business transaction tree and snapshot views from the sample application are provided below to illustrate the difference in data granularity with flow-level visibility enabled.





When you set the flow-level-visibility-enabled flag as 0 or 1, you might need to delete the BT in the tree view so that the tree view gets updated appropriately.

Deploy AppDynamics for Azure

AppDynamics integrates seamlessly with Microsoft Azure. AppDynamics offers visibility into .NET code execution and Microsoft Azure services out of the box, allowing you to troubleshoot performance bottlenecks and optimize the performance of your Microsoft Azure applications.

AppDynamics for Azure App Service

To install and deploy AppDynamics for Azure App Service:

- For Ops teams: Install the AppDynamics Azure Site Extension for .NET or Java:
  - See Install the AppDynamics Azure Site Extension for .NET
  - See Install the AppDynamics Site Extension for Java
- For Dev teams: Install the **AppDynamics.Agent.Azure.AppService.Windows** NuGet package and configure the .NET agent in Azure App Service. See Install the AppDynamics .NET Microservices Agent and Install AppDynamics for Azure App Service.

AppDynamics for Azure Cloud Services and Azure Service Fabric

To install and deploy AppDynamics for Azure Service Fabric, see Install AppDynamics for Azure Service Fabric.

If you are instrumenting Cloud Services, see Install AppDynamics for Azure Cloud Services.

Upgrade the AppDynamics for Windows Azure NuGet Package

If you are currently using the AppDynamics.WindowsAzure NuGet Package, you will need to upgrade to a new NuGet package, depending on your Azure platform. See Upgrade AppDynamics.WindowsAzure NuGet Package.

Install AppDynamics for Azure App Service

Related Pages:

- SELinux Installation Issues

There are two ways to install the .NET microservices agent into Azure App Service:

- Use the AppDynamics Azure Site Extension for .NET or Java:
  - See Install the AppDynamics Azure Site Extension for .NET
  - See Install the AppDynamics Site Extension for Java
- Use the AppDynamics for Windows Azure NuGet Package: See Install the AppDynamics .NET Microservices Agent

Azure Site Extension is used by Ops teams that may not have access to source files, or would not like to modify or recompile them, yet still want to monitor their Azure projects and solutions.

The NuGet Package is used by Dev teams to deploy the .NET agent so they can include the agent binaries, scripts, and configurations in their project and configure it using Visual Studio or other IDE.

Install the AppDynamics Azure Site Extension for .NET

You can use the Microsoft Azure Portal to add the AppDynamics Azure Site Extension to your Azure App Service web app. Azure Site Extension is used by Ops teams that may not have access to source files, or would not like to modify or recompile them, yet still want to monitor their Azure projects and solutions.

Prepare to Install

To install the AppDynamics for Microsoft Azure Site Extension, you need:

- Connection information for your AppDynamics Controller. See Agent and Controller Compatibility.
- A Microsoft Azure account.
- An Azure web app to monitor.

If you are upgrading from a previous version, see Upgrade the AppDynamics Azure Site Extension.

Add the AppDynamics Azure Site Extension

Add the AppDynamics Azure Site extension as you would any site extension for any Azure web app.

1. Log in to the Microsoft Azure Portal.
2. Browse to your web app.

   > To configure the .NET Agent using environment variables, add the environment variables before you install the AppDynamics Azure Site Extension. See Configure the Agent Using Environment Variables.

3. From the **DEVELOPMENT TOOLS** list, click **Extensions**.
4. Click **+Add** to install the version of the **AppDynamics Azure Site Extension** you want to add to your web app.
   After you install the AppDynamics Azure Site Extension, it appears in the installed extensions list.

Add the AppDynamics Azure Site Extension Using an ARM Template

You can deploy the AppDynamics Azure Site Extension to Azure App Services using an Azure Resource Manager (ARM) template.

The following procedure uses Visual Studio Community 2017.

To create and deploy an ARM template:

1. From your web application in Visual Studio, select **File > New Project**.
2. Click **Cloud**, then click **Azure Resource Group**, then click **OK**.
3. From the **Select Azure Template** dialog, click **Web App**, then click **OK**.
4. Under your newly-created Resource Group, click the Website.json file.
5. Under JSON outline in the left pane, right-click **resources**, then select **Add New Resource**.
6. In the Add Resource dialog box, select **Application Settings for Web Apps**, enter a name, then click **OK.**
7. Under the properties section, enter your AppDynamics Controller information. The following example shows the Website.json file with the application properties for AppDynamics.
8. Add a new apiVersion section with your Azure Site Extension details:

```
{
     "apiVersion": "2015-08-01",
     "name": "[variables('webSiteName')]",
     "type": "Microsoft.Web/sites",
     "location": "[resourceGroup().location]",
     "tags": {
       "[concat('hidden-related:', resourceGroup().id, '/providers/Microsoft.Web/serverfarms/', parameters('hostingPlanName'))]
       "displayName": "Website"
      },
      "dependsOn": [
        "[resourceId('Microsoft.Web/serverfarms/', parameters('hostingPlanName'))]"
      ],
      "properties": {
        "name": "[variables('webSiteName')]",
        "serverFarmId": "[resourceId('Microsoft.Web/serverfarms', parameters('hostingPlanName'))]"
      },
      "resources": [
        {
          "apiVersion": "2016-08-01",
          "name": "appsettings",
          "type": "config",
          "dependsOn": [
            "[resourceId('Microsoft.Web/Sites', variables('webSiteName'))]"
           ],
          "properties": {
            "appdynamics.controller.hostName": "<host_name>",
            "appdynamics.controller.port": "<port_name>",
            "appdynamics.controller.ssl.enabled": "true",
            "appdynamics.agent.accountName": "<account_name>",
            "appdynamics.agent.accountAccessKey": "[paraeters(<InKeyVault>)]",
            "appdynamics.agent.applicationName": "<application_name>",
            "appdynamics.agent.tierName": "<tier_name>",
            "appdynamics.agent.nodeName": "node_name>"
          }
        },
        {
          "apiVersion": "2015-08-01",
          "name": "AppDynamics.WindowsAzure.SiteExtension.4.5.release",
          "type": "siteextensions".
          "dependsOn":
            "[resourceId('Microsoft.Web/Sites', variables('webSiteName'))]"
          ]
        },
```

Here is the sample text for copying or pasting:

```
{
      "apiVersion": "2015-08-01",
      "name": "[variables('webSiteName')]",
      "type": "Microsoft.Web/sites",
      "location": "[resourceGroup().location]",
      "tags": {
        "[concat('hidden-related:', resourceGroup().id, '/providers/Microsoft.Web/serverfarms/', parameters
('hostingPlanName'))]": "Resource",
        "displayName": "Website"
      },
      "dependsOn": [
        "[resourceId('Microsoft.Web/serverfarms/', parameters('hostingPlanName'))]"
      ],
      "properties": {
        "name": "[variables('webSiteName')]",
        "serverFarmId": "[resourceId('Microsoft.Web/serverfarms', parameters('hostingPlanName'))]"
      },
      "resources": [
        {
          "apiVersion": "2016-08-01",
          "name": "appsettings",
          "type": "config",
          "dependsOn": [
            "[resourceId('Microsoft.Web/Sites', variables('webSiteName'))]"
          ],
          "properties": {
            "appdynamics.controller.hostName": "mycompany.saas.appdynamics.com",
            "appdynamics.controller.port": "443",
            "appdynamics.controller.ssl.enabled": "true",
            "appdynamics.agent.accountName": "mycompany",
            "appdynamics.agent.accountAccessKey": "[parameters('AppDAccessKeyInKeyVault')]",
            "appdynamics.agent.applicationName": "HelloWorldSecureAppDKey",
            "appdynamics.agent.tierName": "TestTier",
            "appdynamics.agent.nodeName": "TestNode"
          }
        },
        {
          "apiVersion": "2015-08-01",
          "name": "AppDynamics.WindowsAzure.SiteExtension.4.5.Release",
          "type": "siteextensions",
          "dependsOn": [
            "[resourceId('Microsoft.Web/Sites', variables('webSiteName'))]"
          ]
        },
```

Configure the Controller Connection

You have the following options to configure the .NET Agent to connect to the AppDynamics Controller:

- Configure the Controller connection using the Kudu console.
- Configure the Controller connection settings using environment variables.

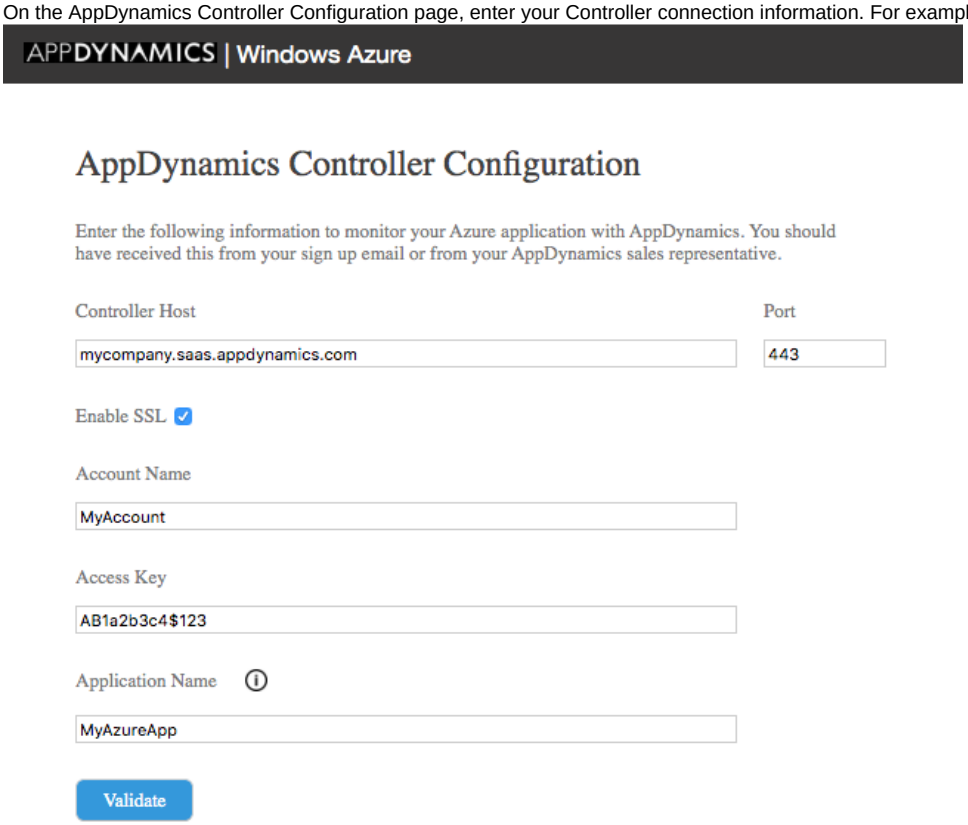
See Agent-to-Controller Connections.

Configure the Agent with the Kudu Console

When you add the AppDynamics Azure Site Extension to your web app, you can interactively configure the .NET Agent using the Kudu Console.

1. Navigate to the AppDynamics Controller Configuration page in the Kudu Console:
   http://{web app}.scm.azurewebsites.net/appdynamics/
   For example: https://myazureexample.scm.azurewebsites.net/appdynamics/

2. On the AppDynamics Controller Configuration page, enter your Controller connection information. For example:



3. Click **Validate** to test the connection to the AppDynamics Controller and save your settings.
4. Restart your web app.
   After you apply some load to your web app, you can view it on flow maps in the AppDynamics Controller UI.

Configure the Agent Using Environment Variables

You can use environment variables to configure the .NET Agent for unattended configuration.

To configure .NET Agents, add the environment variables before you install the AppDynamics Azure Site Extension:

1. Navigate to **SETTINGS > Application Settings** for your web app.
2. Add the .NET Agent environment variables under **App settings:**
   - appdynamics.controller.hostName: address for the AppDynamics Controller
   - appdynamics.controller.port: Controller port
   - appdynamics.agent.accountName: account name you use to log in to the Controller
   - appdynamics.agent.accountAccessKey: account key you use to log in to the Controller
   - appdynamics.agent.applicationName: business application name in the Controller
   - appdynamics.controller.ssl.enabled: set to `True` to enable SSL connection to the Controller; otherwise set to `False`.
   - appdynamics.proxy.hostName: name of the proxy host
   - appdynamics.proxy.authDomain: (Windows only) domain name of the proxy server
   - appdynamics.proxy.port: port number of the proxy
   - appdynamics.proxy.authName: name of the user who connects to the proxy
   - appdynamics.proxy.authPassword: password of the user who connects to the proxy
   - appdynamics.enable.tls12: set to True to enable transport layer security (TLS) 1.2; otherwise set to False.
3. Restart your web app. After you apply some load to your web app, you can view it on flow maps in the AppDynamics Controller UI.

Specify Which WebJobs to Monitor

Use the `APPDYNAMICS.PROCESSLIST` environment variable to specify which WebJobs the .NET Agent monitors.

Environment Variable: `APPDYNAMICS.PROCESSLIST`
Type: Strings separated by "|"
Default: None. When undefined, the agent instruments all WebJob running in the app service. When defined, the .NET Agent instruments only the specified processes.

Required: No

Usage Cases:

- Environment 1: An Azure app service with no WebJobs, where the Appdynamics .NET Agent site extension is installed and configured. After defining the environment variable `AppDynamics.ProcessList` with the value `w3wp.exe`, the `Daasrunner.exe` WebJob, which is a default WebJob that is added to all app services, is not instrumented.
- Environment 2: An Azure app service, with two WebJobs, `webjob1.exe` and `webjob2.exe`, where the Appdynamics .NET Agent site extension is installed and configured. After defining the environment variable `AppDynamics.ProcessList` with the value `w3wp.exe`, only the worker process serving the actual web app will be instrumented and no WebJobs will be instrumented. With the environment variable value set to `w3wp.exe|webjob1.exe`, the worker process serving the web app and the webjob1.exe are instrumented, but the webjob2.exe is not instrumented. With the `AppDynamics.ProcessList` removed from the App settings of the Azure app service, all the WebJobs, including `Daasrunner.exe` are are instrumented, including the worker process serving the web app.

Upgrade the AppDynamics Azure Site Extension

When you click on the Extensions tab for your web app, the Microsoft Azure Portal displays the currently installed version of the AppDynamics Azure Site Extension. The Update Available column of the installed extensions list indicates if there is a more recent minor release of the .NET Agent available. If so, you can click to update the extension from the list.

Upgrade a Major Version of the .NET Agent

AppDynamics maintains major release versions of the .NET Agent as separate site extensions. Therefore you need to uninstall the installed version of the AppDynamics Azure Site Extension before you upgrade to a new major release:

1. Log in to the Microsoft Azure Portal.
2. Stop your web app.
3. Click on the AppDynamics Azure Site Extension from the list of installed extensions and click **Delete** to uninstall it.
4. Install the new version of the AppDynamics Azure Site Extension as normal.

If you are upgrading from .NET Agent 4.2 and you used environment variable configuration, you must update your web app environment variables. See configure the agent using environment variables. Note that the `APPD_UNATTENDED` variable is no longer required.

Install the AppDynamics .NET Microservices Agent

Install the NuGet package **AppDynamics.Agent.Azure.AppService.Windows** to add the .NET microservices agent directly to projects you plan to deploy to Windows Azure. This package works for Azure App Service (formerly Azure Websites): Web Apps and API Apps.

Prepare to Install the NuGet Package

To install the AppDynamics for Windows Azure NuGet package you need:

- Connection information for your AppDynamics Controller 4.4
- Visual Studio >= 2012
- A Visual Studio solution to monitor
- Windows Azure SDK
- Windows Azure account

These instructions assume you are familiar with NuGet package management in Visual Studio and with the Microsoft Azure Portal. You can also try the step-through tutorial on how to instrument an App Service on the community knowledgebase.

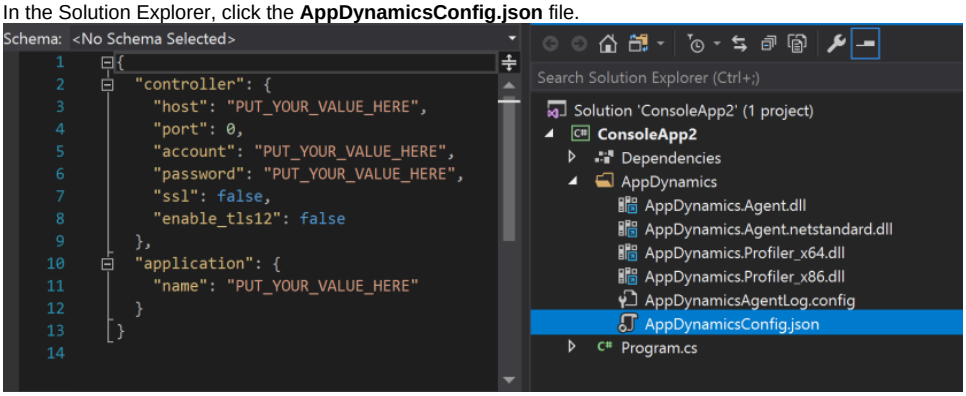
Add the .NET Microservices Agent to Your Azure Solution

Use the NuGet package manager to browse the nuget.org package source and install the **AppDynamics.Agent.Azure.AppService.Windows** package to your Azure solution in Visual Studio.

> For details about managing NuGet packages, see the documentation for your version of Visual Studio.

The **AppDynamics.Agent.Azure.AppService.Windows** package installation prompts you as follows:

1. The installer may ask you to verify your changes. If so, click **OK**.
2. Click **I accept** to agree to the terms of the license.
3. In the Solution Explorer, click the **AppDynamicsConfig.json** file.



4. Enter the connection information for your AppDynamics Controller in the area on the left.

For information about the AppDynamicsConfig.json file, see AppDynamicsConfig.json File.

Configure the Controller Connection

You have the following options to configure the .NET Agent to connect to the AppDynamics Controller:

- During development: As shown in the previous screenshot, you can enter your environment variables in the AppDynamicsConfig.json file, and save it in source control.
- During build: Define your msbuild parameters or environment variables that are passed to the AppDynamicsConfig.json file at build time. AppDynamicsConfig.json does exist at build time. So, if you're defining your msbuild parameters/env variables at build time, you will need to ignore it in source control so that the new AppDynamicsConfig.json file will be created.
- During runtime: Enter your environment variables in Azure.

By default, the .NET microservices agent names Azure tiers as site name. The .NET microservices agent names Azure nodes by machine name appended with the website hostname.

Monitor Virtual Applications of Azure Web Apps

Related links:

- Deploy AppDynamics for Azure

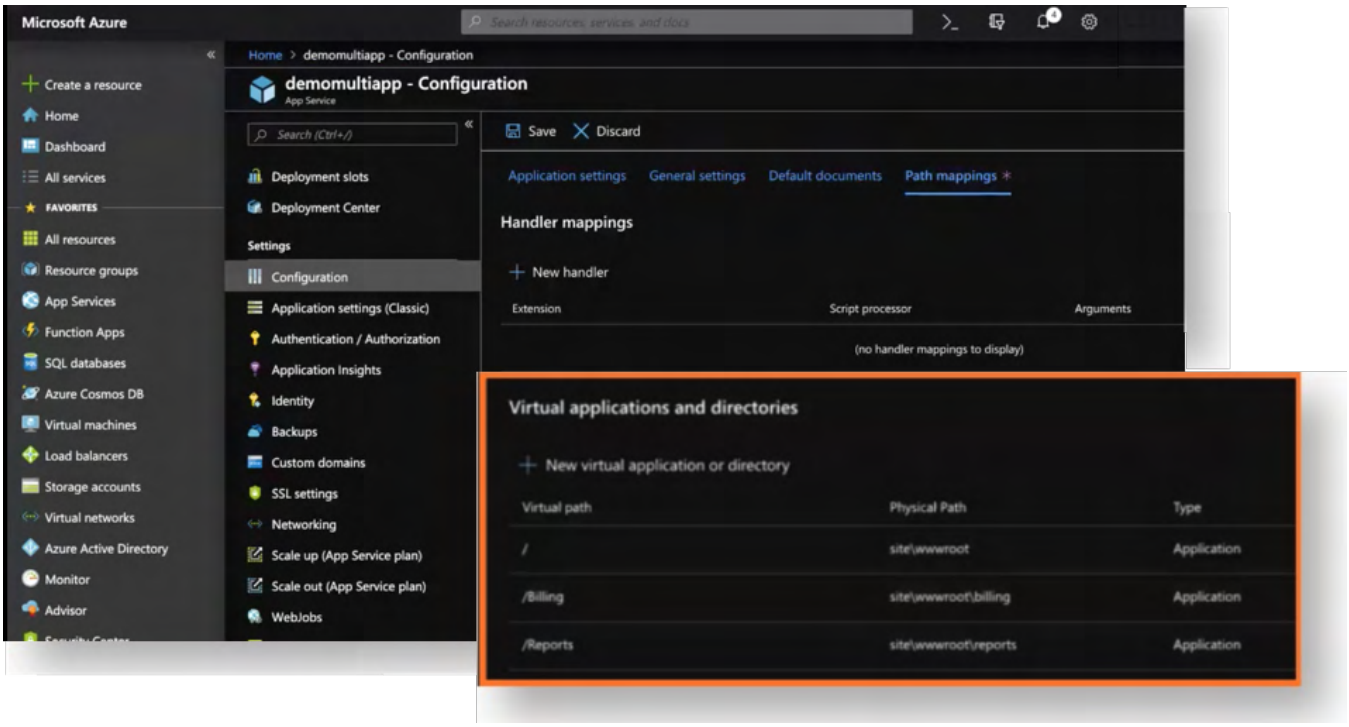
This page applies to the .NET Agent in the Azure App services environment. It describes how the .NET Agent by default names your virtual application tiers and nodes and also how you can customize tier names.

You can automatically name the tiers of virtual applications using either a default or custom method:

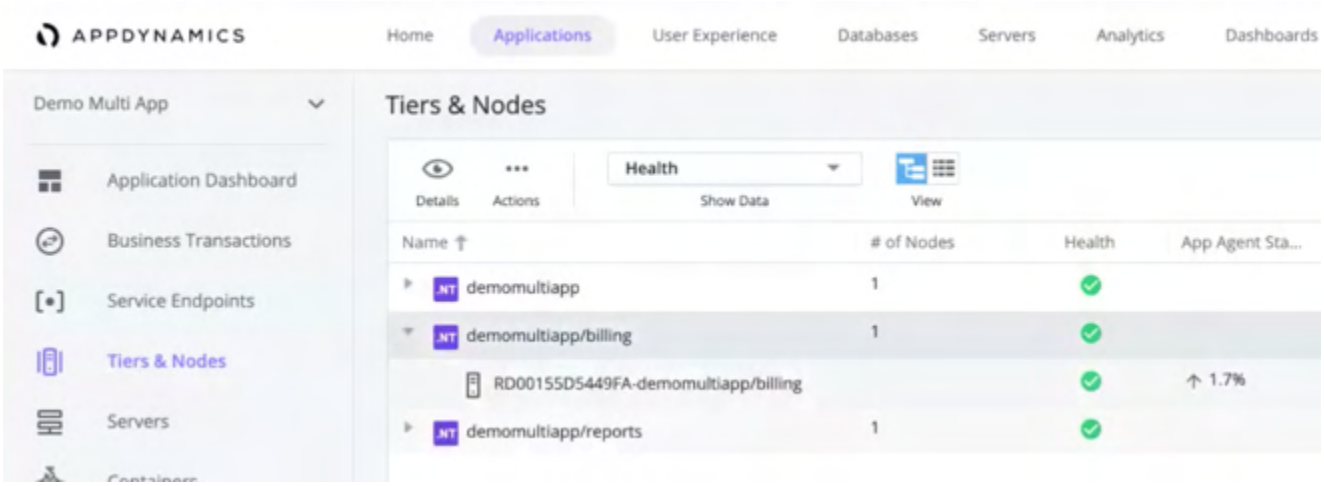
- Default: AppDynamics creates new tiers for each virtual directory. Each tier contains a node representing the virtual directory.
- Custom: When you specify a tier name in the `AppDynamicsConfig.json` file, all virtual directory applications will report as nodes under this tier.

Default Behavior

The following shows the Azure Web Apps configuration of the *demomultiapp* application with three virtual apps, the main application and two sub-applications.



Once you instrument the application with the .NET Agent, restart the application and apply load to it, the .NET Agent reports metrics for the application and virtual applications. In the Controller UI, each virtual application node appears under a separate tier. The name of the tier consists of the name of the application appended with the virtual path of the sub-application. For example, *demomultiapp/billing* as shown below. The name of the node is the same as the tier name prepended with the name of the virtual machine on which it runs.



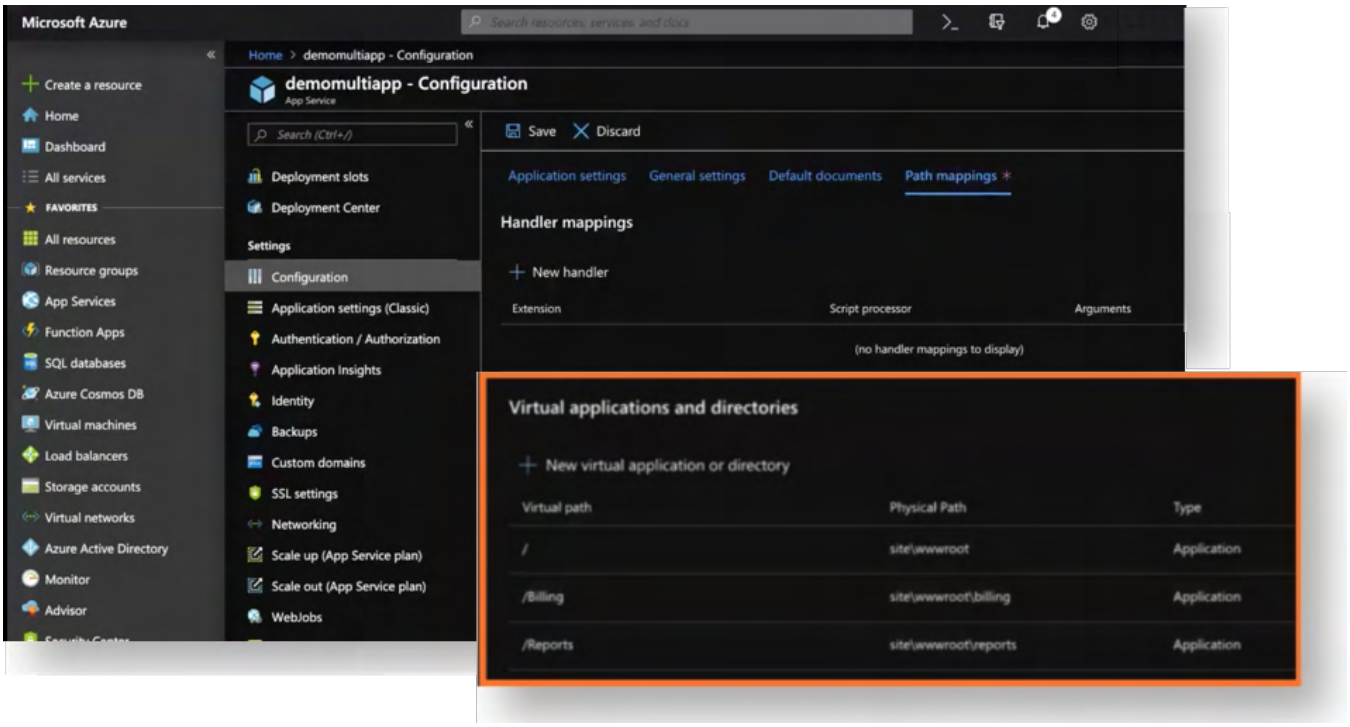
You can manually edit the tier name in the Controller UI.

Customize Tier Names

Rather than creating a tier for each virtual application, all virtual applications can report as nodes under a single tier. The tier name specified in the `AppDynamicsConfig.json` file sets the tier name.

- When you instrument your application using the .NET Agent site extension, the .NET Agent automatically instruments all virtual directories and applications. However, all AppSettings set at the root app are inherited by all virtual applications.
- For NuGet-based deployments, the agent uses the `AppDynamicsConfig.json` file deployed in the same path as the Profiler DLL and ignores the `AppDynamicsConfig.json` file settings for individual virtual apps in the deployment.

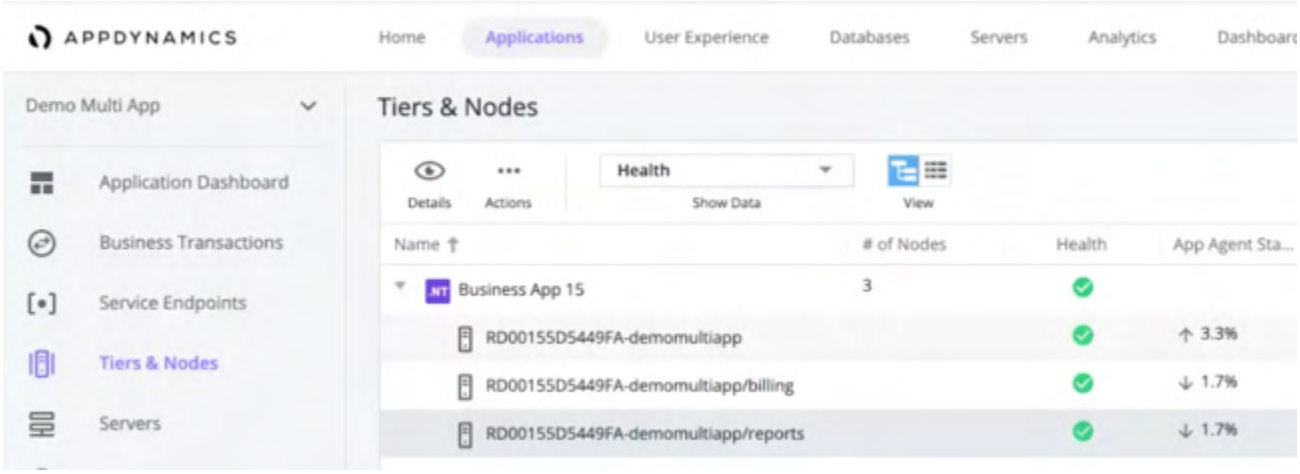
For example, in Azure Web Apps we have the following virtual app and sub-applications.



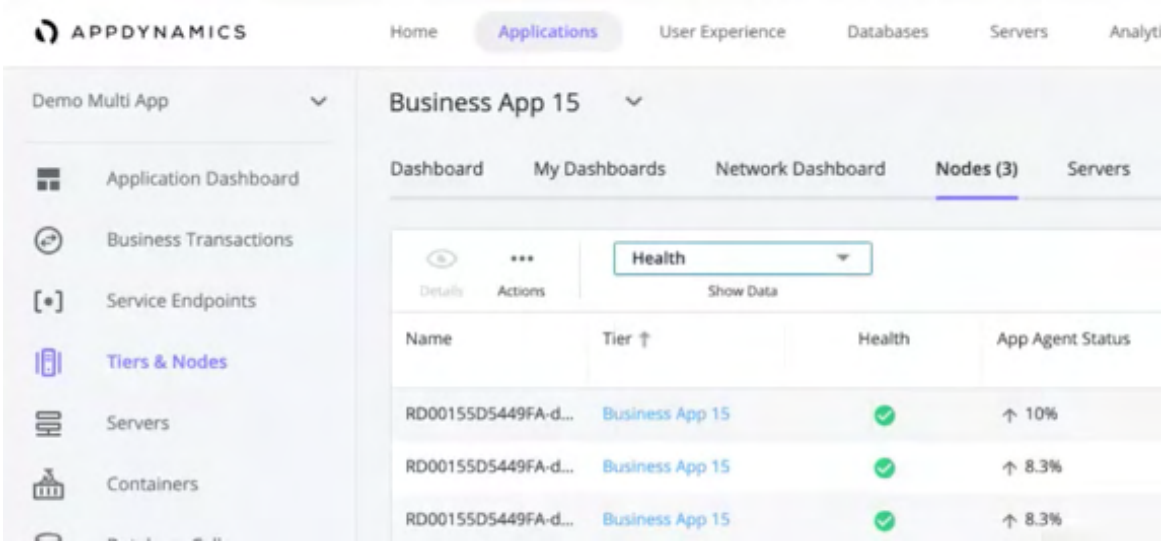
By default, the tier name of the application in the `AppDynamicsConfig.json` file is `null`. The following edited version of the `AppDynamics.Config.json` file shows a tier name of *Business App 15*. Note that the node name cannot be changed as its value is automatically updated as the application scales vertically and horizontally.

```
{
  "controller": {
    "host": "[redacted]",
    "port": [redacted],
    "account": "[redacted]",
    "password": "[redacted]",
    "ssl": [redacted],
    "enable_tls12": [redacted],
    "proxy": [redacted]
  },
  "application": {
    "name": "Demo Multi App",
    "tier": "Business App 15",
    "node": null
  },
  "instrumentors": {
    "customCorrelationConfig": null,
    "enable": [],
    "disable": []
  }
}
```

After the application is restarted, the virtual app tier name appears as *Business App 15*. When you expand the tier on the Tiers & Nodes Dashboard, you see the nodes of the virtual applications and the root application.



The Nodes Dashboard displays the nodes of the tier *Business App 15* in a similar manner.



Nodes of the virtual application are displayed in the Controller UI using the name of the virtual machine the app is running on, followed by the name of the virtual application as it appears in Azure Web Apps.

The current agent configuration file does not support targeting different virtual applications for customized tier naming, however, you can manually create .NET tiers and move the nodes to different tiers in the Controller UI.

Prevent Daas Instrumentation

You can prevent the agent from instrumenting the Daasrunner process by creating the environment variable `AppDynamics.Processlist` and setting it to `w3wp.exe`. This environment variable acts as an allowlist, where only the processes specified will be instrumented. Then delete the Daas tier and the Daasrunner node on the **Tiers & Nodes Dashboard**. You might also want to include WebJob processes in this list. See Install the AppDynamics Azure Site Extension for .NET.

Limitations

In this version, you cannot customize the tier names of virtual sub-applications in the `AppDynamicsConfig.json` file.

Also, this version does not support monitoring virtual apps together with deployment slots.

Instrument the .NET Agent with Azure Functions

AppDynamics serverless monitoring for Azure Functions gives you visibility into the performance of your applications that run as functions on Microsoft Azure. The .NET Agent supports Azure Functions hosted on App Service plans. Once installed, the .NET Agent automatically discovers business transactions in your Azure Functions.

You can monitor your applications running in Azure Functions exactly as you would any other application instrumented with the .NET Agent. Correlate business transactions between and through Azure Functions receive snapshots, create callgraphs, and integrate functions with End User Monitoring.

Before You Begin

Before you start instrumenting the .NET Agent with Azure Functions, make sure you meet the following requirements:

Microsoft Azure Requirements

- Applications hosted on the AppService plan
- Azure Functions versions 1.*x*, 2.*x*, and 3.*x*
- An instrumented Azure Function that is
    - implemented in .NET and
    - hosted on Windows OS

AppDynamics Requirements

- Microservice licenses
- .NET Agent version >= 4.5.16

Install AppDynamics Azure Site Extension

To add the AppDynamics Azure Site extension:

1. Log in to the Windows Azure Portal.
2. Browse to **Function App** and select the desired function.
3. Select **Platform Features** > **Site Extensions** > **+Add**.
4. Click **Choose Extensions** and select AppDynamics Azure Site Extension 4.5.
5. Click **Legal Terms**. Review AppDynamic's legal terms and conditions and select **OK**.
6. Click **OK** in the **Add Extension** menu to complete the installation.
7. Restart the function app.

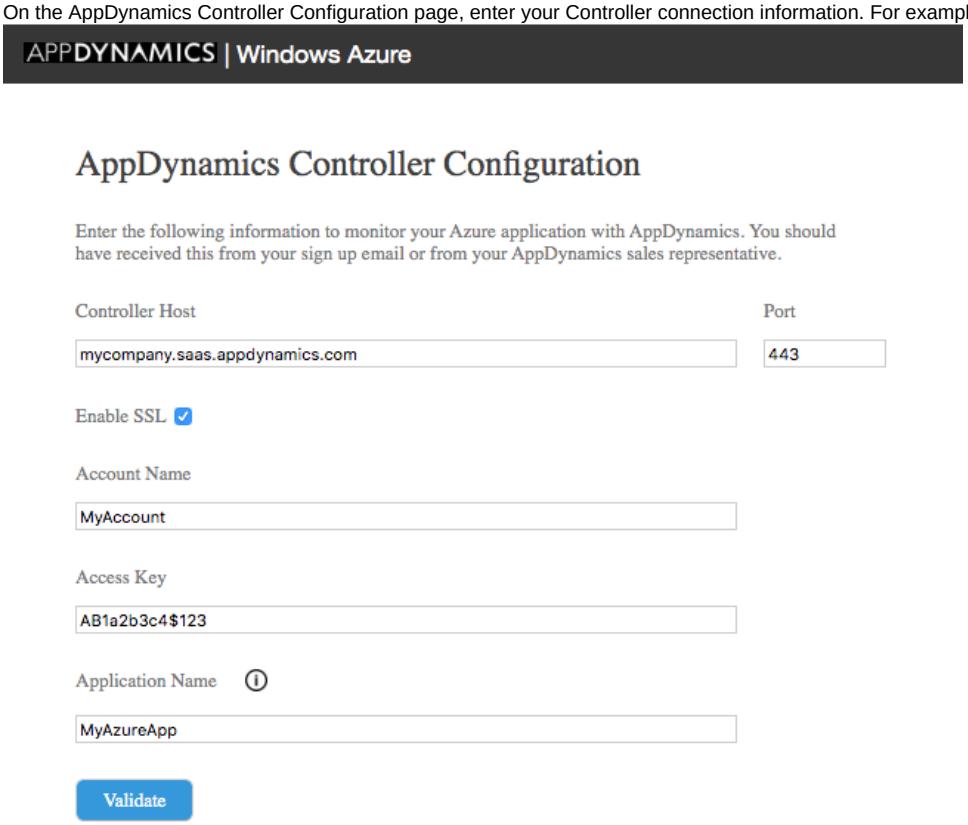
After you install the AppDynamics Azure Site Extension, it appears in the list of the installed extensions.

Connect the .NET Agent to the Controller

You can connect your Azure Function apps to your Controller using the Kudu Console.

1. Navigate to the AppDynamics Controller Configuration page in the Kudu Console:
   `http://{web app}.scm.azurewebsites.net/appdynamics/`
   For example: `https://myazureexample.scm.azurewebsites.net/appdynamics/`

2. On the AppDynamics Controller Configuration page, enter your Controller connection information. For example:



3. Click **Validate** to test the connection to the AppDynamics Controller and save your settings.
4. Restart your function app.

Verify Instrumentation

To verify Azure Function instrumentation, view the function on flow maps in the AppDynamics Controller UI.

Azure Function Names in the Controller UI

In the Controller, the business transaction names match the Azure Function name.

HTTP Trigger functions are named using two segments of the function URL. You can change the name of HTTP Trigger functions in custom transaction naming.

Automate .NET for Azure Functions Deployment

This page describes how to scale instrumentation of Azure Functions with the .NET Agent.

Add the AppDynamics Azure Site Extension Using an ARM Template

You can deploy the AppDynamics Azure Site Extension to Azure App Services using an Azure Resource Manager (ARM) template. This procedure uses Visual Studio Community 2017.

To create and deploy an ARM template:

1. From your web application in Visual Studio, choose **File > New Project**.
2. Click **Cloud**, then click **Azure Resource Group**, then click **OK**.
3. From the **Select Azure Template** dialog box, click **Web App**, then click **OK**.
4. Under your newly-created Resource Group, click the Website.json file.
5. Under JSON outline in the left pane, right-click **resources**, then choose **Add New Resource**.
6. In the Add Resource dialog box, select **Application Settings for Web Apps**, enter a name, then click **OK.**
7. Under the properties section, enter your AppDynamics Controller information. The following example shows the Website.json file with the application properties for AppDynamics.
8. Add a new resource for the Site Extension, as shown in the following example:

```
{
      "apiVersion": "2015-08-01",
      "name": "[variables('webSiteName')]",
      "type": "Microsoft.Web/sites",
      "location": "[resourceGroup().location]",
      "tags": {
        "[concat('hidden-related:', resourceGroup().id, '/providers/Microsoft.Web/serverfarms/', parameters
('hostingPlanName'))]": "Resource",
        "displayName": "Website"
      },
      "dependsOn": [
        "[resourceId('Microsoft.Web/serverfarms/', parameters('hostingPlanName'))]"
      ],
      "properties": {
        "name": "[variables('webSiteName')]",
        "serverFarmId": "[resourceId('Microsoft.Web/serverfarms', parameters('hostingPlanName'))]"
      },
      "resources": [
        {
          "apiVersion": "2016-08-01",
          "name": "appsettings",
          "type": "config",
          "dependsOn": [
            "[resourceId('Microsoft.Web/Sites', variables('webSiteName'))]"
          ],
          "properties": {
            "appdynamics.controller.hostName": "mycompany.saas.appdynamics.com",
            "appdynamics.controller.port": "443",
            "appdynamics.controller.ssl.enabled": "true",
            "appdynamics.agent.accountName": "mycompany",
            "appdynamics.agent.accountAccessKey": "[parameters('AppDAccessKeyInKeyVault')]",
            "appdynamics.agent.applicationName": "HelloWorldSecureAppDKey",
            "appdynamics.agent.tierName": "TestTier",
            "appdynamics.agent.nodeName": "TestNode"
          }
        },
        {
          "apiVersion": "2015-08-01",
          "name": "AppDynamics.WindowsAzure.SiteExtension.4.5.Release",
          "type": "siteextensions",
          "dependsOn": [
            "[resourceId('Microsoft.Web/Sites', variables('webSiteName'))]"
          ]
        },
```

Configure the Agent Using Environment Variables

If you want to configure the .NET Agent using environment variables, add the environment variables before you install the AppDynamics Azure Site Extension.

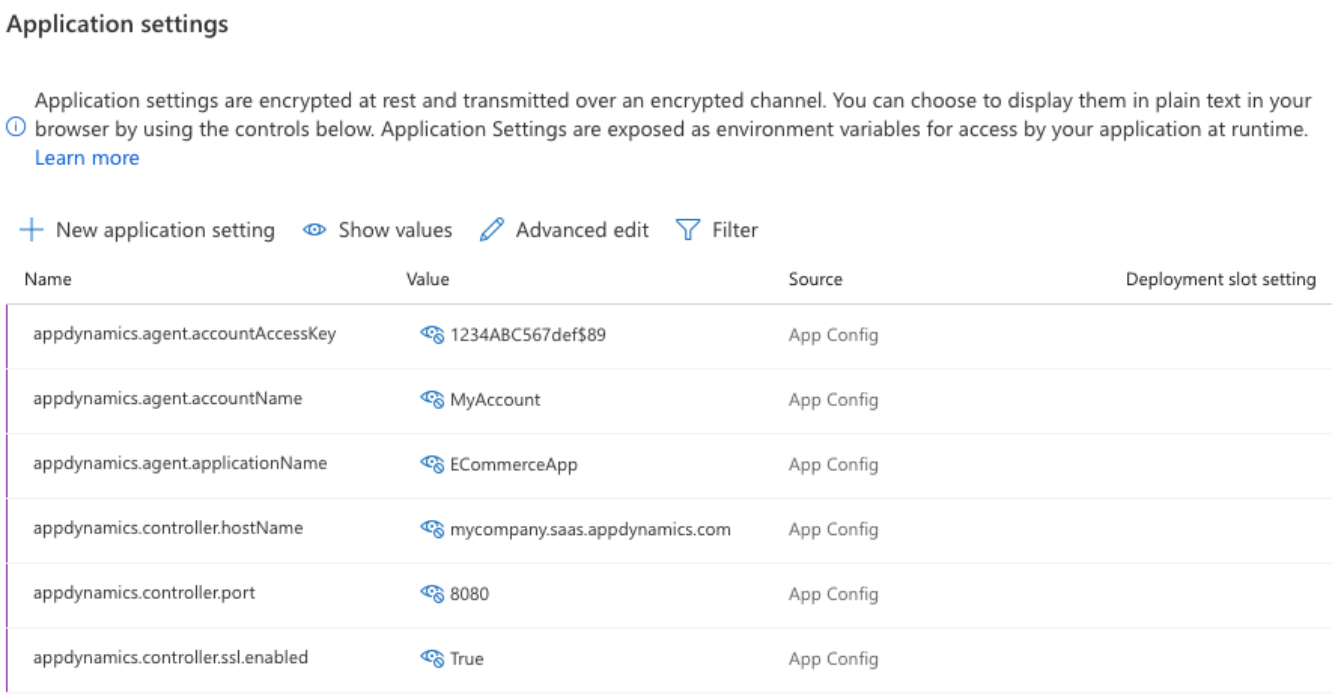
Configuring the .NET Agent using environment variables allows for unattended configuration. To configure agents in this manner, add the environment variables before you install the AppDynamics Azure Site Extension, as follows:

1. In the Azure Portal, open your Azure Function.
2. Navigate to **Function app settings > Manage application settings**.
3. Click **+New Application Setting** and enter your environment variables as name/value pairs.

The table below lists the available environment variables:

Environment Variable (Application Setting)	Description (Value)	Required
`appdynamics.controller.hostName`	Host associated with the Controller used by your Azure Function. Do not include http:// or https://	Yes
`appdynamics.controller.port`	Port associated with your Controller. If the port is left blank or is invalid, defaults to 443.	No
`appdynamics.agent.accountName`	Account name associated with the Controller used by your Azure Function.	Yes
`appdynamics.agent.accountAccessKey`	Access key for your Controller. See Agent-to-Controller Connections.	Yes
`appdynamics.agent.applicationName`	Name of the Azure Function App.	Yes
`appdynamics.controller.ssl.enabled`	Sets SSL connection to the Controller. Value "True" enables SSL, while value "False" disables SSL.	No

For example:



Install AppDynamics for Azure Service Fabric

The following instructions explain how to deploy the .NET microservices agent for AppDynamics for Azure Service Fabric.

Before you Install

From nuget.org, download the NuGet package `AppDynamics.Agent.Distrib.Micro.Windows` from https://www.nuget.org/packages/AppDynamics.Agent.Distrib.Micro.Windows/, then extract it to a folder. Then, use the following instructions to deploy the agent package.

Choose a procedure below to match your environment:

- Deploy the Package using Visual Studio
- Deploy the Package using another IDE

Deploy the Package using Visual Studio

1. From your editor, open `ServiceManifest.xml` in *each* folder in the published `application` root.
2. Add the following environment variables in each `CodePackage`:

```
<CodePackage ...>
  <EntryPoint>
    ...
  </EntryPoint>
  <EnvironmentVariables>
    <EnvironmentVariable Name="COR_ENABLE_PROFILING" Value="1" />
    <EnvironmentVariable Name="COR_PROFILER" Value="{39AEABC1-56A5-405F-B8E7-C3668490DB4A}" />
    <EnvironmentVariable Name="COR_PROFILER_PATH" Value="<PathToDeploymentFolder>/AppDynamics.
Profiler_x64.dll" />
  </EnvironmentVariables>
</CodePackage>
```

3. Make a copy of the `AppDynamicsConfig.json` file and rename it to `<executable_name>.AppDynamicsConfig.json`. For example, `<servicefabricapplicationname>.AppDynamicsConfig.json`.
4. Copy `AppDynamics.Agent.dll`, `AppDynamics.Profiler_x64.dll`, `AppDynamicsAgentLog.config`, `AppDynamicsConfig.json` from `<nuget_package>\content\AppDynamics` and add these files in the Visual Studio solution of each of the service projects at the top level, not under any subfolders.
5. Make a copy of `AppDynamicsConfig.json` and rename it to:`<executable_name>.AppDynamicsConfig.json`, for example, `<servicefabricapplicationname>.AppDynamicsConfig.json`. Put it into the root of each service project.

> When you rename the `.json` file to the application name, do not include `.exe` at the end of your application name.

6. Right-click on the AppDynamics related files and select **Copy Always** in **Properties**.
7. Update `<executable_name>.AppDynamicsConfig.json` with your configuration information:

```
{
 "controller":
 {
 "host":"",
 "port":0,
 "account":"",
 "password":""
 },
 "application":
 {
 "name":""
 }
}
```

Do not specify node; it will be assigned automatically; Specifying tier is optional, it could be assigned automatically.

For information about the `AppDynamicsConfig.json` file, see AppDynamicsConfig.json File.

The next steps are optional and should only be used if you need per service-instance control of the controller/application/tier.

1. Add the following environment variables to each CodePackage:

```
<EnvironmentVariable Name="appdynamics.controller.hostName" Value="< >" />
<EnvironmentVariable Name="appdynamics.controller.port" Value="<  >" />
<EnvironmentVariable Name="appdynamics.agent.accountName" Value="<  >" />
<EnvironmentVariable Name="appdynamics.agent.accountAccessKey" Value="<  >" />
<EnvironmentVariable Name="appdynamics.agent.applicationName" Value="<  >" />
<EnvironmentVariable Name="appdynamics_agent_tier_name" Value="<  >" />
```

2. Modify `ApplicationManifest.xml`. In each `ServiceManifestImport` and `CodePackage` folder add:
    - In `ServiceManifestImport`:

      ```
      <EnvironmentOverrides CodePackageRef="Code">
        <EnvironmentVariable Name="appdynamics.controller.hostName" Value="<AppD_ControllerHostName>" />
        <EnvironmentVariable Name="appdynamics.controller.port" Value="<AppD_ControllerPort>" />
        <EnvironmentVariable Name="appdynamics.agent.accountName" Value="<AppD_AccountName>" />
        <EnvironmentVariable Name="appdynamics.agent.accountAccessKey" Value="<AppD_AccountAccessKey>" />
        <EnvironmentVariable Name="appdynamics.agent.applicationName" Value="<AppD_ApplicationName>" />
        <EnvironmentVariable Name="appdynamics_agent_tier_name" Value="<AppD_Service_Name_TierName>" />
      </EnvironmentOverrides>
      ```

    - In `Parameters`:

      ```
      <Parameter Name="AppD_ControllerHostName" DefaultValue="<Your_Controller_Name>" />
      <Parameter Name="AppD_ControllerPort" DefaultValue="<Your_Controller_Port>" />
      <Parameter Name="AppD_AccountName" DefaultValue="<Your_Controller_Account_Name>" />
      <Parameter Name="AppD_AccountAccessKey" DefaultValue="<Your_Controller_Access_Key>" />
      <Parameter Name="AppD_ApplicationName" DefaultValue="<Your_Controller_App_Name>" />
      ```

    - For each service: `<Parameter Name="AppD_%SERVICE_NAME%_TierName" DefaultValue="<Tier_Name_For_Service>" />`
3. If you need to provide the AppDynamics configuration during application package deployment, you can now do it using the `-ApplicationParameter` switch of the `New-ServiceFabricApplication cmdlet`.

Deploy the Package using another IDE

1. From your editor, open `ServiceManifest.xml` in each folder in the published application root.
2. Add the following environment variables in each CodePackage:

```
<CodePackage ...>
  <EntryPoint>
    ...
  </EntryPoint>
  <EnvironmentVariables>
    <EnvironmentVariable Name="COR_ENABLE_PROFILING" Value="1" />
    <EnvironmentVariable Name="COR_PROFILER" Value="{39AEABC1-56A5-405F-B8E7-C3668490DB4A}" />
    <EnvironmentVariable Name="COR_PROFILER_PATH" Value=AppDynamics/AppDynamics.Profiler_x64.dll" />
   </EnvironmentVariables>
</CodePackage>
```

3. Using PowerShell, unpack `AppDynamics.Agent.dll`, `AppDynamics.Profiler_x64.dll`, `AppDynamicsAgentLog.config`, `AppDynamicsConfig.json` from <nuget_package>\content\AppDynamics to the code package folder that contains the executable you want to run.
4. Make a copy of the of the `AppDynamicsConfig.json` file and rename it to `<executable_name>.AppDynamicsConfig.json`. For example, <servicefabricapplicationname>.AppDynamicsConfig.json.

> When you rename the `.json` file to the application name, do not include .exe at the end of your application name.

5. Make a copy of `AppDynamicsConfig.json` and rename it to:<executable_name>.AppDynamicsConfig.json. For example, `<servicefabricapplicationname>.AppDynamicsConfig.json`. Put it into the root of each service project.
6. Update `<executable_name>.AppDynamicsConfig.json` with your configuration information:

```
{
 "controller":
 {
   "host":"",
   "port":0,
   "account":"",
   "password":""
 },
 "application":
 {
 "name":""
 }
}
```

Do not specify the node; it will be assigned automatically. Specifying the tier is optional; it could be assigned automatically.

For information about the AppDynamicsConfig.json file, see AppDynamicsConfig.json File.

The next steps are optional and should only be used if you need per service-instance control of the controller/application/tier.

1. Add the following environment variables to each CodePackage:

```
<EnvironmentVariable Name="appdynamics.controller.hostName" Value="" />
<EnvironmentVariable Name="appdynamics.controller.port" Value="" />
<EnvironmentVariable Name="appdynamics.agent.accountName" Value="" />
<EnvironmentVariable Name="appdynamics.agent.accountAccessKey" Value="" />
<EnvironmentVariable Name="appdynamics.agent.applicationName" Value="" />
<EnvironmentVariable Name="appdynamics_agent_tier_name" Value="" />
```

2. Modify `ApplicationManifest.xml`. In each `ServiceManifestImport` and `CodePackage` folder add:
   - In `ServiceManifestImport`:

```
<EnvironmentOverrides CodePackageRef="Code">
  <EnvironmentVariable Name="appdynamics.controller.hostName" Value="<AppD_ControllerHostName>" />
  <EnvironmentVariable Name="appdynamics.controller.port" Value="<AppD_ControllerPort>" />
  <EnvironmentVariable Name="appdynamics.agent.accountName" Value="<AppD_AccountName>" />
  <EnvironmentVariable Name="appdynamics.agent.accountAccessKey" Value="<AppD_AccountAccessKey>" />
  <EnvironmentVariable Name="appdynamics.agent.applicationName" Value="<AppD_ApplicationName>" />
  <EnvironmentVariable Name="appdynamics_agent_tier_name" Value="<AppD_Service_Name_TierName>" />
</EnvironmentOverrides>
```

   - In `Parameters`:

```
<Parameter Name="AppD_ControllerHostName" DefaultValue="<Your_Controller>" />
<Parameter Name="AppD_ControllerPort" DefaultValue="<Your_ControllerPORT>" />
<Parameter Name="AppD_AccountName" DefaultValue="<Your_Controller_AccountName>" />
<Parameter Name="AppD_AccountAccessKey" DefaultValue="<Your_Controller_AccessKey>" />
<Parameter Name="AppD_ApplicationName" DefaultValue="<Your_Controller_App_Name>" />
```

   - For each service: `<Parameter Name="<AppD_Service_Name_TierName>" DefaultValue="<Tier_Name_For_Service>" />`
3. If you need to provide the AppDynamics configuration during application package deployment, you can now do it using the `-ApplicationParameter` switch of the `New-ServiceFabricApplication cmdlet`.

Install AppDynamics for Azure Cloud Services

You can use Microsoft Visual Studio with the NuGet package **AppDynamics.Agent.Azure.CloudServices** to add the .NET Agent directly to Azure Cloud Services projects: Web Roles and Worker Roles.

Prepare to Install the NuGet Package

To install the AppDynamics for Windows Azure NuGet package you need:

- Connection information for your Agent-to-Controller Connections 4.4
- Visual Studio >= 2012
- A Visual Studio solution to monitor
  The user account running Visual Studio must have the following permissions to the solution:
  - **Read** and **Write** permissions to each **project directory**
  - **Read** and **Write** permissions to each **Visual Studio .NET C# Project** (*.csproj) file
  - **Read** and **Write** permissions to the **Service Definition** (ServiceDefinition.csdef) file
- Windows Azure SDK
- Windows Azure account

These instructions assume you are familiar with NuGet package management in Visual Studio and with the Microsoft Azure Portal. You can also try the step-through tutorial on how to instrument an App Service on the community knowledgebase.

If you are upgrading to a new version of the AppDynamics for Windows Azure NuGet Package, see Upgrade AppDynamics.WindowsAzure NuGet Package.

Add the .NET Agent to Your Azure Solution

Use the NuGet package manager to browse the nuget.org package source and install the **AppDynamics.Agent.Azure.CloudServices** package to your Azure solution in Visual Studio.

> For details about managing NuGet packages, see the documentation for your version of Visual Studio.

The **AppDynamics.Agent.Azure.CloudServices** package installation prompts you as follows:

- The installer may ask you to verify your changes. If so, click **OK**.
- Click **I accept** to agree to the terms of the license.

The package manager installs the .NET agent into your project.

The configuration file installs to `<project_root>/App_Data/AppDynamics/Config/config.xml`. Enter the connection information for your AppDynamics Controller in the config.xml file. For details about Controller connection settings, see Agent-to-Controller Connections.

NuGet - Solution

```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <controller host="" port="0" ssl="false" enable_config_deployment="false">
    <application name="" />
    <account name="" password="" />
  </controller>
  <machine-agent />
  <app-agents azure="true" azure-role-name="" azure-role-instance-id="">
    <IIS>
      <automatic />
    </IIS>
    <standalone-applications>
      <standalone-application executable="WaWorkerHost.exe">
        <tier name="" />
      </standalone-application>
    </standalone-applications>
  </app-agents>
</appdynamics-agent>

```

By default the .NET Agent names Azure tiers as role name for Cloud Services.

If you want to customize the tier name, edit config.xml under `<project_root>/App_Data/AppDynamics/Config.xml`. For more information and examples, see "Name IIS Tiers Manually" on Name .NET Tiers.

After you finish any configuration changes, publish your solution to Windows Azure. For Cloud Services solutions, you can put load on your published project and log on to the AppDynamics Controller and begin monitoring your solution.

Update .NET Agent Configuration for Cloud Services

If you have already published your solution, you can update the .NET Agent configuration for the currently installed version without upgrading the agent.

1. Edit the config.xml file to make configuration changes.
2. Edit the startup.cmd file under `<solution_root>/AppDynamics`.
3. By default, the APPD_AGENT_CONFIGUPDATE variable is set to true. If you want to modify the configuration on your Azure Cloud Services machine, and you want to prevent the AppDynamics script from overwriting it, set APPD_AGENT_CONFIGUPDATE to false.

Upgrade AppDynamics.WindowsAzure NuGet Package

If you are currently using the AppDynamics.WindowsAzure NuGet package to monitor your Azure solutions, you must upgrade to one of the newest packages on nuget.org, depending on your platform and the applications that you want to monitor. For a list of packages, see .NET Microservices Agent.

To upgrade your Azure NuGet package:

1. In Visual Studio or other IDE, uninstall the AppDynamics.WindowsAzure NuGet package.

   > For Azure App Service only: After you uninstall the NuGet package, you *must* delete the AppDynamics folder from the Azure portal and your Visual Studio solution. You may need to stop your application first before deleting this folder. During uninstallation, the webconfig modification that is responsible for filtering user requests against the AppDynamics folder is removed, but this does not remove the AppDynamics folder. You must manually delete this folder. You must also remove the AppDynamics-specific information from `apphostconfig.xdt`.

2. Install one of the new NuGet packages.

Release Notes and PDFs

Release Notes	Associated PDF
4.4.3 Azure Enhancements and Resolved Issues	

4.4.3 Azure Enhancements and Resolved Issues

Enhancements:

- Microsoft Orleans is supported on the .NET and .NET microservices agents.
- .NET Core 2.0 for Windows is supported on the .NET microservices agent.
- Support for Azure Service Fabric Remoting v2.

The following NuGet packages are now available. If you have used a previous version of an AppDynamics NuGet package, you will need to upgrade to the new version. See Upgrade AppDynamics.WindowsAzure NuGet Package.

AppDynamics.Agent.Distrib.Micro.Windows	AppDynamics NuGet package for .NET. This package *should not* be installed directly and is intended for download and file distribution. This package is used for Azure Service Fabric deployments. See Install AppDynamics for Azure Service Fabric for instructions.
AppDynamics.Agent.Windows	AppDynamics .NET Core microservices agent for Windows. Recommended for standalone installations. See Install the .NET Core Microservices Agent for Windows for deployment instructions. NOTE: This package does not support .NET Framework, only .NET Core for Windows.
AppDynamics.Agent.Azure.CloudServices	AppDynamics .NET agent for Azure Cloud Services. See Install AppDynamics for Azure Cloud Services.
AppDynamics.Agent.Azure.AppService.Windows	AppDynamics .NET microservices agent for Azure App Service. This package is intended for applications deployed to Azure App Service (Azure Web Apps and Azure API Apps). See Install the AppDynamics .NET Microservices Agent and Install AppDynamics for Azure App Service for instructions.

Machine Agents

Related Pages:

- SELinux Installation Issues

Machine Agents and Network Agents give you end-to-end visibility into the hardware and networks on which your applications run. These agents can help you identify and troubleshoot problems that can affect application performance such as server failures, JVM crashes, and network packet loss.

You use the Machine Agent to collect basic hardware metrics. The functionality provided by the Machine Agent includes:

- Reporting basic hardware metrics from the server OS, for example, %CPU and memory utilization, disk and network I/O
- Reporting metrics passed to the Controller by extensions
- Running remediation scripts to automate your runbook procedures. You can optionally configure the remediation action to require human approval before the script is started.
- Running JVM Crash Guard to monitor JVM crashes and optionally run remediation scripts

If you have a Server Visibility license, the Machine Agent provides the following additional functionality:

- Extended hardware metrics such as machine availability, disk/CPU/virtual-memory utilization, and process page faults
- Monitor internal or external HTTP and HTTPS services
- Group servers together so that health rules can be applied to specific server groups
- Define alerts that trigger when certain conditions are met or exceeded based on monitored server hardware metrics

SELinux Installation Issues

SELinux is a security mechanism that works on top of the native file and directory read/write/execute permissions within the Linux file system. It is available for most Linux distributions and is installed by default in newer RHEL (Red Hat Enterprise Linux) & Fedora distributions.

As SELinux may prevent the installation and/or operation of any software being executed, ensure that you create appropriate policy file for it.

> Ensure that you consult with your security team to determine the correct level of access for the APM.

SELinux allows you to set a finer granularity of restrictions on access and execution. This control is represented by "policy files", typically created and maintained by the SecOps team within your organization. For more details about SELinux, see https://selinuxproject.org/page/Main_Page.

The policy files are found in `/etc/sestatus.conf` by default. To determine if SELinux exists on your system, run the `getenforce` command which returns the string `Enforcing` if it is active.

Alternatively, you can run this command:

```
sestatus
```

which generates this output:

```
SELinux status: enabled
SELinuxfs mount: /selinux
Current Mode: permissive
Policy version: 16
sestatus
```

If `SELinux status` is `disabled`, it indicates that the system has not installed the package. However, if the status returned is `enabled`, but the `Current Mode` is `permissive`, then SELinux policy files are not enforced. To install and test the APM Agent:

- Set the mode to permissive and then enable it
- Follow the SELinux guidelines to create the appropriate policy statements for the agent in question

> For more details on how to customize your policy files, see https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/5/html/deployment_guide/sec-sel-policy-customizing.
>
> To enable SELinux, use the command `setenforce 1` to enable enforcing mode; to disable SELInux use `setenforce 0` (set to `permissive` mode).
>
> For more details about enabling/disabling SELinux, see: https://docs.fedoraproject.org/en-US/Fedora/11/html/Security-Enhanced_Linux/sect-Security-Enhanced_Linux-Working_with_SELinux-Enabling_and_Disabling_SELinux.html

Administer App Server Agents

These pages provide general information about installing App Server agents.

For Agent-to-Controller version compatibility information, see Agent and Controller Compatibility.

Planning Agent Deployment

When planning your deployment, you should consider whether to install agents manually or automatically.

If deploying a relatively small number of agents, you may choose to install the agent manually.
For larger environments involving hundreds of agents, you should develop an automated deployment strategy.

For specific agents, see these links:

- Java: Java Agent or Install the Java Agent
- .NET: .NET Agent or Install the .NET Agent for Windows
- Node.js: Node.js Agent or Install the Node.js Agent
- PHP: PHP Agent or Install the PHP Agent
- Python: Python Agent or Install the Python Agent
- Database Agent: Administer the Database Agent
- Standalone Machine Agent: Install the Machine Agent

For automated deployment guidelines, see Controller Deployment.

Encrypt Agent Credentials

Related pages:

- Encrypt Credentials in .NET Agent Configuration
- Java Agent Configuration Properties

This page provides an overview of securing agent credentials in AppDynamics.

AppDynamics agents store several types of credential information on disk, including:

- Controller account access key
- Controller keystore/agent truststore password
- Proxy server password

Secure Credential Store

You can use the Secure Credential Store to encrypt credentials in the agent configuration for environments where security policies require secure credentials on disk.

The Secure Credential Store is comprised of two components:

- `scs-tool.jar`: A utility to create the secure credential store, encrypt credentials, and obfuscate the credential store password.
- Secure credential keystore: A keystore for the secret encryption key.

The Secure Credential Store encrypts plain text using the strongest encryption available according to the system's encryption jurisdiction policy.

For the .NET Agent, see Encrypt Credentials in .NET Agent Configuration.

After you set up the Credential Keystore, you must specify the following settings.

For the Analytics Agent:

- `ad.secure.credential.store.filename`
- `ad.secure.credential.store.password`

See Analytics Agent Rules.

For the Java, Machine, and Database Agents:

- `<controller-ssl-enabled>`
- `<controller-keystore-filename>`
- `<controller-keystore-password>`

See Java Agent Configuration Properties, Machine Agent Properties, and Database Agent Properties.

Initialize the Secure Credential Store

Before you can encrypt or obfuscate passwords, you must run the Secure Credential Store utility to create the keystore for your secret encryption key. The agent distribution includes the Secure Credential Store utility in the following locations:

- Java Agent: `<javaagent_home>/verX.X.X.X/utils/scs/scs-tool.jar`
- Machine Agent: `<machine_agent_home>/lib/secure-credential-store-tool-1.3.0.0.jar`
- Database Agent: `<database_agent_home>/lib/scs-tool.jar`
- Analytics Agent: `<analytics_agent_home>/bin/tool/scs-tool.jar`

Run the Secure Credential Store utility `generate_ks` command with the following parameters:

- `filename`: Absolute path where the utility will create the secure credential keystore. Use this path for `<credential-store-filename>` in agent configuration.
- `storepass`: The secure credential keystore password. Use the obfuscated version of this password as the value for `<credential-store-password>` in agent configuration.

For example:

```
/<full path to application JRE>/bin/java -jar ./scs-tool.jar generate_ks -filename '/opt/appdynamics
/secretKeyStore' -storepass 'MyCredentialStorePassword'
```

The Secure Credential Store utility confirms it created and initialized the keystore:

```
Successfully created and initialized new KeyStore file: /opt/appdynamics/secretKeyStore
Verification - New KeyStore file: /opt/appdynamics/secretKeyStore is properly initialized.
```

Encrypt Passwords

To encrypt passwords using the secure credential store utility, run the `encrypt` command with the following parameters:

- `filename`: Absolute path to the secure credential keystore file.
- `storepass`: Password for the secure credential keystore. You can use either a plain-text password or a password that has been obfuscated as described in the following section.
- `plaintext`: Any plain text to encrypt. For instance, account access key or password.

The following example uses a plain-text password—`storepass` argument—for the secure credential keystore:

```
/<full path to application JRE>/bin/java -jar ./scs-tool.jar encrypt -filename '/opt/appdynamics
/secretKeyStore' -storepass 'MyCredentialStorePassword' -plaintext 'MyAccessKeyOrPassword'
```

The same example uses an obfuscated password:

```
/<full path to application JRE>/bin/java -jar ./scs-tool.jar encrypt -filename '/opt/appdynamics
/secretKeyStore' -storepass 's_gsnwR6+LDch8JBf1RamiBoWfMvjjipkrtJMZXAYEkw8=' -plaintext 'MyAccessKeyOrPassword'
```

The Secure Credential Store utility writes out an encrypted password for use in agent configuration files:

```
r9iDWPzHRCNDM1B6KTag4A/cA5B4pouVPkv48ovRm6c=
```

Obfuscate the Secure Credential Store Password

In order to access the secret key in the secure credential keystore, the agent needs the obfuscated credential store password.

Run the secure credential store utility `obfuscate` command with the following parameter:

- `plaintext`: The plain text secure credential keystore password.

For example:

```
/<full path to application JRE>/bin/java -jar /opt/appdynamics/scs-tool.jar obfuscate -plaintext
'MyCredentialStorePassword'
```

The Secure Credential Store utility writes out an obfuscated password for use in the `<credential-store-password>` in agent configuration.

For example:

```
s_gsnwR6+LDch8JBf1RamiBoWfMvjjipkrtJMZXAYEkw8=
```

Encrypt a Plain Text Property

After you obfuscate the Secure Credential Store password, you can encrypt plain text properties.

The following example demonstrates how to encrypt properties in the Analytics Agent:

```
$ /<full path to application JRE>/bin/java -jar scs-tool.jar encrypt -filename /opt/appdynamics/secretKeyStore -
storepass 'Welcome' -plaintext 'MyAccountAccessKey'
```

The property generates an encrypted credential:

```
-001-24-pFoSE/xdPcinkBj9iiKvpQ==Rznx8Kt3sPZHQnKfYyubVuhorrBEbYFtDTPm8c/1kFO+Z2eR2WEHtBRg4vy1GyvJ
```

Sample Agent Configuration

The following example demonstrates the agent configuration properties for the Secure Credential Store. For more information, see the agent-specific configuration property documentation.

Java Agent Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<controller-info>
        ...
    <!-- Encrypted account access key -->
    <account-access-key>r9iDWPzHRCNDM1B6KTag4A/cA5B4pouVPkv48ovRm6c=</account-access-key>

    <!-- Encrypted Controller keystore / agent trust store password -->
    <controller-keystore-password>Tw49bd0hdCMBoQ5pfMMuYA/cA5B4pouVPkv48ovRm6c=</controller-keystore-password>

    <!-- Enable the Secure Credential Store -->
    <use-encrypted-credentials>true</use-encrypted-credentials>

    <!-- Path to they secure credential keystore -->
    <credential-store-filename>/opt/appdynamics/secretKeyStore</credential-store-filename>

    <!-- Obfuscated secure credential keystore password -->
    <credential-store-password>n/8GvAZsKk4gM3Z6g+XQ1w==</credential-store-password>
        ...
</controller-info>
```

Analytics Agent Configuration

```
ad.credential.store.filename=/opt/appdynamics/secretKeyStore
ad.credential.store.password=s_gsnwR6+LDch8JBf1RamiBoWfMvjjipkrtJMZXAYEkw8=
```

Encrypt Data on the Analytics Agent

You can encrypt any data on the Analytics Agent using `secure://<your-encrypted-credentials>`. You can encrypt data in the Analytics Agent properties file or in System Properties.

The following example demonstrates how to encrypt `http.event.accessKey` in the Analytics Agent properties file.

```
http.event.accessKey=secure://-001-24-Dr9FQGC179o4vPnuljnx8A==ZGVw/P4OONvpUidIhJ2u78FpRVVW8fbgr8J1HBHXwnE=
ad.secure.credential.store.filename=/opt/appdynamics/secretKeyStore
ad.secure.credential.store.password=s_gsnwR6+LDch8JBf1RamiBoWfMvjjipkrtJMZXAYEkw8=
```

Manage App Agents

From the AppDynamics Agents page, you can manage, reset, and disable app agents connected to the Controller, and view agent status and information.

Permissions

To manage app agents, you must belong to a role with the Account level permission:

- Administer users, groups, roles, authentication, and so on
- View license
- View AppDynamics agents
- Use Agent Download Wizard

The predefined Account Owner role includes the required permission.

To access app agent administration settings, click **Settings** > **AppDynamics Agents** > **App Server Agents**.

Associate App Agents with a Business Application

If you start an application server with an app agent without specifying the business application, the agent appears in the App Server Agents tab as an unassociated agent. You can associate an agent with an application manually:

1. From the agents list in the App Server Agents tab, select the unassociated app agent from the list.

   In the **View** menu, click **Agents not associated with an Application** to filter the list of App Server Agents.

2. Click **Associate with Application**. This button is enabled only for app agents that are connected to the Controller but not associated with a business application.

Reset App Agents

To reset app agents, users need the "Agent Advanced Operation" permission.

Resetting the app agent causes the agent to register itself with the Controller in the same way it does at JVM or application startup, but without restarting the JVM or application. Resetting the app agent purges in-memory data for the agent. It also applies certain configuration changes to the app agent, such as app agent node property changes.

Resetting an app agent causes the Controller to:

- Purge data such as in-memory business transactions and metrics, exit calls, and registration information for the agent
- Reset the business transaction limit counter to zero for the selected agent

Resetting an agent does not re-instrument or remove existing instrumentation. After a reset, the agent re-registers business transactions and backends and it creates new metrics. You may lose a few minutes worth of data between the reset and re-registration operations.

You may need to reset agents after you have reconfigured business transaction or backend detection and the agent is not applying the updated rules:

1. Delete all existing transactions and backends. See information on deleting unwanted business transactions in Transaction Detection Rules, Monitor Databases, and Remote Services.
2. Choose a reset option:
   - To reset all the app agents for a business application, click **Reset Agents**.
   - To disable a individual app agents, select one or more agents and click **Reset Selected App Agent(s).**

Enable and Disable App Agents

Disabling an app agent stops the agent from reporting metrics to the Controller. Disabling an agent can help you diagnose agent installation or application issues. It also lets you compare the difference in overhead between when the agent when capturing data versus when it is not, for example, without removing the agent. Disabling the agent does not require you to shut down or restart the application server.

Disabling an app agent does not stop the agent from operating or remove the bytecode instrumentation added by the agent. To fully remove the agent including remove bytecode instrumentation, follow the steps to uninstall the app agent or fully disable agent instrumentation under the Install App Server Agents page specific for the agent type. Uninstalling an app agent or fully disabling an app agent in this manner typically requires an application restart.

All agents connected to the Controller count against the agent license limits for that Controller. Even though it only reports minimal app server-related data, a disabled app agent is still connected to the Controller and consumes an app agent license.

To enable or disable all the app agents for a business application, the App Server Agents tab, click **Enable Agents** or **Disable Agents**.

Permission:

You must have the application-level permission, *configure agent properties* to enable and disable agents.

You can enable or re-enable individual agents by selecting them and clicking **Enable Selected App Agent** or **Disable Selected App Agent**. Alternatively, enable or disable app agents individually from the Agents tab on the Node Dashboard:

- To disable the agent, click Agent is **Off**.
- To enable the agent, click Agent is **On**.
  It takes about a minute for the operation to take effect.

By default turning the agent **Off** completely disables monitoring. For Java agents, un-check **Disable all monitoring including JVM and JMX metrics** on the Disable This App Agent pane to continue collecting JVM and JMX metrics such as heap memory, memory pools, garbage collection, and thread count.

Delete App Agents

Deleting an app agent removes the agent and any associated data from the Controller database. This does not affect the instrumentation of the application server. If the application server is still running with an app agent, or if it is subsequently restarted with an app agent, the agent re-registers and appears again in the Controller.

You can delete an app agent by clicking **Delete Agent from System**.

To completely remove an app agent completely so that it does not register again, uninstall it. See uninstallation instructions for your agent platform under Install App Server Agents.

Agent Log Files

Related pages:

- Java Agent Logging
- .NET Agent on Windows Logging
- PHP Agent Logging
- Node.js Agent Logging
- Python Agent Debugging and Logging
- Apache Agent Logging
- Dynamic Agent Proxy Logging
- Business Transactions Logging
- Bytecode Transformer Logging
- REST Logging
- Analytics Agent Logging

You can use agent logs to resolve agent configuration and application instrumentation issues. The Controller can generate and archive agent log files that you can submit to AppDynamics Support for troubleshooting assistance. See Request Agent Log Files.

Agent Log File Information

The beginning of the log file shows the startup of the agent monitoring services and the configuration settings. The agent log also contains the sequence of agent runtime activity and exceptions that are encountered. You can use this information to troubleshoot deployment issues.

Examples of log information:

- Agent version and build date
- JVM runtime version (Java only)
- Configuration changes
- Backend detection
- Exceptions
- Output from logging session requests

Logging Levels

The logging levels, listed in order from collecting the most information to the least, are:

- ALL: Logs all events.
- TRACE: Reports finer-grained informational events than the debug level that may be useful to debug an application.
- DEBUG: Reports fine-grained informational events that may be useful to debug an application.
- INFO: Default log level. Reports informational messages that highlight the progress of the application at coarse-grained level.
- WARN: Reports on potentially harmful situations.
- ERROR: Reports on error events that may allow the application to continue running.

Not all agents log at all levels.

Log Structure

This section describes how the agent logs are rolled over for these agents:

- Java Agent
- .NET Agent
- Proxy
- BytecodeTransformer
- Business Transactions
- JMX
- REST
- Machine Agent
- Dynamic Services (used internally)

When the maximum file size is reached, a new log file is created. The first file is named ...0.log, second file is ...1.log, and so on. There is a maximum of five files per set and a maximum of five sets for each instrumented node. If the maximum number of sets is reached, when a new set is created, the oldest set is deleted.

While the agent rotates away old log files, it retains the initial log file. The first log file contains information that reflects the specific context in which the agent was started, along with other information that can be useful for troubleshooting and record keeping. On agent restart, a new set is created and the oldest set is deleted

Each set includes not only the application agent log but also, depending on which logs exist, the `ByteCodeTransformer` log, the `REST` log and the `BusinessTransaction` log. A single set might consist of:

- `agent.<timestamp>.#.log`
- `ByteCodeTransformer.<timestamp>.#.log`
- `REST.<timestamp>.#.log`
- `BusinessTransaction.<timestamp>.#.log`

where # is the number of the set.

For example, these logs in the logs directory were created on April 3 or the first set, set 0. The agent is a Java Agent; this is indicated by the prefix *agent*. Other app agents use different naming conventions. See the agent-specific information.

```
agent.2015_04_03__14_49_38.0.log
agent.2015_04_03__14_49_46.0.log
agent.2015_04_03__14_51_04.0.log
BusinessTransactions.2015_04_03__14_49_38.0.log
BusinessTransactions.2015_04_03__14_49_46.0.log
BusinessTransactions.2015_04_03__14_51_04.0.log
ByteCodeTransformer.2015_04_03__14_49_40.0.log
ByteCodeTransformer.2015_04_03__14_49_47.0.log
ByteCodeTransformer.2015_04_03__14_51_04.0.log
REST.2015_04_03__14_49_38.0.log
REST.2015_04_03__14_49_46.0.log
REST.2015_04_03__14_51_04.0.log
```

On the next agent restart a new set is created. The logs in this set, set 1, will have names such as:

```
agent.2015_04_03__14_53_05.1.log
. . .
```

and the next set, set 2,

```
agent.2015_04_04__15_12_06.2.log
. . .
```

and so on through the five potential sets.

Agent-Specific Information

AppDynamics app server agents have different logging locations and different maximum log sizes.

Bytecode Transformer Logging

Related pages:

- Agent Log Files
- Java Agent Logging
- .NET Agent on Windows Logging
- Dynamics Agent Proxy Logging

The Bytecode Transformer log is generated by the Java Agent, the .NET Agent and the proxy. It contains the information associated with the AppDynamics platform bytecode instrumentation (BCI) engine. The BCI engine is used to inject interceptors. The BCI engine logs the classes being loaded in the application.

This log is useful for troubleshooting issues that depend on bytecode instrumentation including entry points, exit points, missing metrics, loggers, errors, exceptions, and asynchronous threads.
This log includes:

- Agent version and build date
- Each Java class that was examined and whether the methods were considered for bytecode instrumentation
- Each method that was instrumented

For the Java Agent and the proxy, the log is named ByteCodeTransformer*Year_mon_day_hr_min.#*.log, where # is the logging set number.

For the .NET Agent, the log is named `ByteCode.txt`.

See Agent Log Files for details about how the sets are organized that roll over.

REST Logging

Related pages:

- Agent Log Files
- Java Agent Logging
- .NET Agent on Windows Logging
- Dynamic Agent Proxy Logging

The REST log contains information about the AppDynamics agent and Controller communication. The log contains request and response payload data between the agent and the Controller, and the raw details of communications between the agent and the Controller. By default, the REST log contains INFO level logging for all registration requests up to a limit of one MB every five minutes.

You can use this log to troubleshoot issues with detecting and reporting business transactions, backends, events, and metric reporting.

The communication between the agent and Controller consists of:

- Configuration data
- Node identification
- Machine registration
- Business transaction registration
- Metric registration
- Event registration
- Snapshot capture/upload to controller database
- Metric data upload to controller database
- Event data upload to the Controller database
- Output from Logging Session requests

The naming convention for the REST log is `REST.<timestamp>.#.log` where # is the set.

See Agent Log Files for information about the structure of the log files into sets that roll over.

Within a set, a REST log file can reach a maximum of 5 MB.

Metrics Limits

To ensure that metric information that is most relevant to your application displays, agents have limits for the number of metrics they can store. Different limits apply to app agents and machine agents:

- For an app agent, the default maximum number of registered metrics is 5000.
- For the Standalone Machine Agent, the default maximum number of registered metrics is 450. This includes both basic hardware metrics and the expanded metrics available with Server Visibility.

If the limit is reached, an error event is generated of type `AGENT_METRIC_REG_LIMIT_REACHED` with a summary of *Metric registration limit of n reached*. No new metrics are created until the agent restarts. If necessary, you can increase the default limit.

Agent Metric Limits

You can increase or decrease the default metric registration limits for machine agents or app agents.

> Changing these limits can affect the resource consumption of your deployment. Before you change this setting verify your application environment and Controller can handle the increased resource requirements.

For the Java agent, modify the limit using the `agent.maxMetrics` system property. For example, to increase the machine agent metric limit, specify the maximum number of metrics as an argument when starting the machine agent in the following format:

```
-Dappdynamics.agent.maxMetrics=<max-number-of-metrics>
```

For example, when starting the machine agent, increase the maximum number of metrics that can be registered to 2000 as follows:

```
nohup java -Dappdynamics.agent.maxMetrics=2000 -jar machineagent.jar &
```

For the .NET Agent, set the `maxMetrics` property as an environment variable. This setting only affects the app agent. For example:

```
appdynamics.agent.maxMetrics=5500
```

For the .NET Agent for Linux, use the node property below to set agent metric limits. See App Agent Node Properties (A) for more information.

```
appdynamics.agent.metricLimits
```

For the .NET Machine Agent, specify the maximum number of metrics using the Metrics element in the config.xml. See 'Machine Agent Element' on .NET Agent Configuration Properties. See also Administer the .NET Agent.

```
<metrics max-metrics="300"/>
```

Controller Metric Limits

The Controller applies its own limits on metric registration. For an on-premises Controller, you can view and modify the properties that control the metric registration limit in the Controller settings page in the Controller Administration Console.

The relevant Controller settings are:

- `metric.registration.limit`: The maximum number of metrics that can be registered for an account in the Controller. The default is 2 million.
- `application.metric.registration.limit`: The maximum number of metrics that can be registered for a business application in the Controller. The default is 1 million.

Historical and Disconnected Nodes

If a node has been out of contact with the Controller for a certain amount of time, the Controller marks the node as a historical node. The Controller suspends certain types of processing activities for the node, such as rule evaluation.

If the node resumes contact with the Controller before the node deletion period expires, the Controller restores it to an active state. Otherwise, the node is permanently removed from the Controller and the node level data is no longer accessible in the UI. Tier and application level historical metric data for the node remains available after the node is deleted.

By default, the Controller considers a node historical after about 20 days of inactivity and deletes the node after 30 days. For a highly dynamic application environment in which nodes are created and destroyed frequently, AppDynamics recommends that you shorten the node activity timeout period to allow recycled nodes to be treated as such in the Controller.

The node activity timeout period is determined by the node retention period or activity settings.

The names of historical nodes can be assigned to new nodes. Node name reuse is a Java Agent option that, when enabled, directs the Controller to reuse node names so that data generated by multiple, short-lived nodes in a given tier is associated with a single logical node.

Node Activity and Agent Licensing

For licensing purposes, the Controller releases the license for the agent if the Controller has not received data from the agent in the previous 5 minutes. This license availability behavior is not affected by the historical node status or node deletion timeout settings.

Configure Node Activity Settings

The node activity settings are account level settings that the root AppDynamics administrator can modify from the Administration Console:

- `node.permanent.deletion.period`: Time (in hours) after which a node that has lost contact with the Controller is deleted permanently from the system. The data is removed. If the agent starts reporting again after this period, it will start like a new node. Therefore, no historical data will be available at the node level. You will see historical data at the tier and app level, and cluster roll up will take place as normal.
  The default is 720 hours, the minimum value is 6 hours, and the maximum value for this setting is unlimited.
- `node.retention.period`: Time (in hours) after which a node that has lost contact with the Controller is deleted. In this case, the Controller UI will not display the node, however, the system will continue to retain it. If the agent starts reporting again within these hours, it will reappear in the UI and the counter will reset. The data is persisted.
  The default is 500 hours, the minimum value is 1 hour, and the maximum value for this setting is unlimited.
  Additional notes about the node retention period:
  - A node will be impacted by the node retention period even if a Machine Agent is associated with that node. If a Machine Agent associated with the App Server Agent is running, the node retention period will still be in effect.
  - You can have any number of Machine Agents running in background that will never appear in the Controller unless the Machine Agent:
    - runs on a host machine that has been shut down or cannot connect to the Controller.
    - is not instrumented or has been uninstalled.
    - has been stopped.
  - If you need a node to be considered for the node retention period, it should be marked as historical on shutdown:

    ```
    Dappdynamics.jvm.shutdown.mark.node.as.historical=true
    ```

Agent Behavior When Disconnected from the Controller

If there are network problems or agent errors, the Controller may become unreachable. The Controller server may also be down for a variety of reasons.

If the Controller is unreachable for one minute:

- The agent goes into standby mode during which it does not detect any transactions.
- Any collected snapshots and events are dropped and lost. Snapshots and events are dropped because they consume too much memory to cache.
- All metrics that have not been posted to the Controller are stored in memory. The memory impact of retaining metrics is minimal.
- New business transaction registrations that have not been posted to the Controller are stored in memory.
- The agent attempts to connect to the Controller every minute and resumes normal activity when it can download its full configuration.

If the Controller becomes reachable in the following minute or two:

- All metrics that have been stored in memory are posted to the Controller.
- New business transaction registrations that have been stored in memory are posted to the Controller.
- Snapshots and events collected in the 20 seconds prior to the reconnection are posted to the Controller.

If the Controller is not reachable after three failed attempts that are one minute apart:

- The agent is muted and all business transaction interceptors are disabled. The interceptors are still called when monitored application entry point methods are executed, but they are unproductive. No new business transactions are discovered or registered. Correlation exit points will set a header such as "`notxdetect=true`", which tells downstream tiers to also ignore the transaction.
- JMX metrics are stored in the application server memory and transmitted to Controller after reconnection; so, there are no gaps in the metric history.
- Periodic metrics for the last three minutes are stored in memory. Metrics older than three minutes are purged from memory.

- The agent configuration channel and the metric channel continue to attempt to connect to the Controller once each minute.

The agent attempts to connect to the Controller in seven one-minute intervals and in five minute intervals afterwards. If the Controller is not able to reconnect after five minutes, the license is freed for another agent to use.

If the connection is successful and the agent is able to download its full configuration and a license:

- All periodic metrics, such as JMX metrics and Windows performance counters for the last three minutes, are posted to the Controller. The Controller drops metrics that were collected too long ago in the past, such as when rollups are already completed.
- The agent is reactivated, business transaction interceptors are re-enabled, business transactions are monitored and possibly snapshotted, new business transactions will be discovered and registered, and downstream correlation is re-enabled.

Request Agent Log Files

You can use agent logging information to troubleshoot issues that are local to an AppDynamics app agent.

> The Controller cannot access the log files generated by the .NET Microservices Agent.

This page describes how to work with agent logs. To access agent logging operations in the Controller UI, navigate to **Node Dashboard > Agents > App Server Agent > Agent Operations**.

Permissions

To request agent thread dumps or agent debug logs, users need the *Agent Advanced Operation* permission.

Get Agent Log Files and Thread Dump Samples

Requesting log files causes the agent to send the contents of its logs directory as a zip file to the Controller. From there, you can download the zip to your local machine.

These agent-specific considerations apply to logging sessions:

- For the Database Agent, Java Agent or PHP Agent, you can get all logs or choose from these alternatives:
  - Output *from a specific logger* at a set level, for a fixed duration. For a list of supported loggers, see Java Supported Environments.
  - *Thread dump samples*. The agent takes a thread dump according to the specified collection interval and specified number to collect. After the interval elapses the thread dumps are uploaded by the agent as a zip file.
- For the Node.js agent, the proxy logs are available.
- For the Python Agent, if you choose *All logs in the logs directory*, you get both the proxy logs and the agent logs. The agent log contains the log messages that it is configured to log in the agent configuration file. See Python Agent Settings.

While the status is PENDING, you can click **Refresh** to update the status field. Once the request succeeds, as indicated by the status, right-click the agent operation name for your request and choose Download Data. Enter a name for your request to prepend the generated zip file with your name. For example, the name of `CurrentTime` produces a zip file named similar to `CurrentTime_Node_8000_1390503141046.zip`.

If SUCCESSFUL doesn't eventually appear in the status for the request, the request may have timed out. A request can timeout due to incorrect request parameters or if the server becomes unavailable. In this case, right-click the request and choose **Delete Data** and retry the request.

Start an Agent Logging Session

For Java and Database Agents, you can generate log information with the specific type of logging information. For the Node.js and Python agents, the log information generated is from the proxy only.

1. Confirm that you have enough disk space to create the logs before starting a logging session. Log file requests fail if there is not enough disk space available. The disk space required depends on application activity. If you are uncertain about the amount of space required, try conducting an agent logging session with the shortest duration and closely monitor the disk space on the monitored machine.
2. Click **Start Agent Logging Session** in the **Agent Operations** panel.
3. Choose the duration for the logging session and the debug logger types to initiate. The logger type determines the type of information captured with these options:

Logging Session Type	Type of Information
Application Wide Configuration	Application configuration changes
Business Transaction Registration	Transaction, exit identification, self-resolution
Current Time	Requests to synch up the clock skew between the controller and the agent
Events	Changes in application state that are of potential interest
Metric Data	Metric data
Metric Registration	Metric registration
One Way Agent	For system (machine) agent only, information from the capture feature
Request Segment data	Snapshot upload of request segment
System Agent Registration	System (machine) agent registration
System Agent Reregistration	System (machine) agent reregistration

System Agent Polling Handler	System (machine) agent configuration request
Task Execution	Relates to the machine agent where you can create a task to execute on a machine, such as to restart the application server and run a script
Top Summary Stats	Request to upload the 'most expensive backend' summary
Application Configuration	Configuration change requests
Transient Channel	The activity of the transient channel that is used to upload and download instructions for JMX console live data

4. Click **Start Agent Logging Session**. Each selected session type displays in the **Session** panel with a start and end time based on the selected **Duration** of the logging session.

Submitting Log Files to Support

When submitting log files to AppDynamics Support, ensure you send the entire contents of the agent logs directory available as a zip file for download when you request the agent logs.

App Agent Node Properties

Related Pages:

- Monitor JVMs
- Monitor .NET Nodes

You can register app agent node properties to customize app agent behavior at the application, tier, or node level.

App Agent Node Properties

App agent node properties control the features and preferences for the Java Agent and .NET Agent. Such agent-specific settings include limits on the number of business transactions, the minimum number of requests to evaluate before triggering a diagnostic session, and so on.

Node properties follow an inheritance model similar to instrumentation detection, so you can set an individual property globally for an application, tier, or node level.

It is possible to set node properties in the Controller UI for the PHP, Node.js, Python, Web Server Agent, and C/C++ SDK agents, however, doing so may lead to unexpected behavior or agent failure, and is therefore not supported for those agent types.

Even though it is possible to configure node properties in the `app-agent-config.xml` file in the agent home directory, AppDynamics recommends that you use the Controller UI to configure node properties. The Controller UI displays only those node properties that are registered to the agent.

The App Agent Node Properties Reference reference includes additional properties that do not display in the UI by default. You can register these properties yourself, but unregistered properties are intended for specific application or troubleshooting scenarios and can impact the performance of your deployment. You should register properties or configure properties directly in `app-agent-config.xml` only under the guidance of AppDynamics Support or as specifically instructed by the documentation.

Permissions

Users need the Configure Agent Properties permission to create, edit, or delete agent configuration.

Edit a Registered Node Property

In the Controller UI, you can access node properties for a particular node or for all nodes from the dashboard of any node in the application, as follows:

1. Access the node dashboard by going to the App Servers page. See Tiers and Nodes.
2. Expand the tier that contains the node on which you want to configure a node property and double-click the node.
3. In the node dashboard, click **Actions > Configure App Server Agent**.
4. Select **Use Custom Configuration**, and then double-click the property to modify.

After customizing a configuration, you can copy the configuration to other nodes, to the tier, or apply it to the entire application.

Add a Registered Node Property

You can register and configure unregistered App Agent Node Properties Reference as instructed by AppDynamics Support.

To register a node property, create a custom configuration for the node (see Edit Registered Node Property). Add properties by clicking the **+** icon at the top of the list of current node properties.

In the **Create Agent Property** panel, use the values from App Agent Node Properties Reference to provide values for the name, description, type, and value of the property.

App Agent Node Properties Reference

This reference page describes the app agent node properties by type. Additionally, the reference includes these pages containing information about the app agent node properties:

- App Agent Node Properties (A)
- App Agent Node Properties (B-C)
- App Agent Node Properties (D-E)
- App Agent Node Properties (F-I)
- App Agent Node Properties (J-L)
- App Agent Node Properties (M)
- App Agent Node Properties (N-R)
- App Agent Node Properties (S)
- App Agent Node Properties (T-Z)

Use caution when modifying the agent default settings. If increasing limits specified for an agent, you must carefully assess and monitor memory consumption by the agent after the change.

App Agent Node Properties by Type

This table groups the app agent node properties by type to browse properties by functionality and feature area.

Type	Property
Automatic Leak Detection	minimum-age-for-evaluation-in-minutes
	minimum-number-of-elements-in-collection-to-deep-size
	minimum-size-for-evaluation-in-mb
Bytecode injection (BCI)	bci-log-config
	bciengine-disable-retransformation
	enable-interceptors-for-security
	enable-json-bci-rules
	enable-xml-bci-rules
Backend Detection	enable-kafka-consumer
	msmq-correlation-field
	msmq-single-threaded
	nservicebus-single-threaded
Object Instance Tracking (OIT)	collection-capture-period-in-minutes
	enable-instance-monitoring
	enable-object-size-monitoring
	leak-diagnostic-interval-in-minutes
JMX Property	heap-storage-monitor-devmode-disable-trigger-pct
Transaction Monitoring	ado-new-resolvers
	aspdotnet-core-naming-controllerarea
	aspdotnet-mvc-naming-controlleraction
	api-thread-activity-timeout-in-seconds
	api-transaction-timeout-in-seconds
	async-tracking
	async-transaction-demarcator
	capture-404-urls
	capture-error-urls
	capture-raw-sql

	capture-set-status
	capture-spring-bean-names
	disable-custom-exit-points-for
	disable-exit-call-correlation-for
	disable-exit-call-metrics-for
	disable-percentile-metrics
	downstream-tx-detection-enabled
	enable-all-rsd-error-propagation
	enable-default-http-error-code-reporter
	enable-soap-header-correlation
	enable-spring-integration-entry-points
	enable-transaction-correlation
	end-to-end-message-latency-threshold-millis
	find-entry-points
	jdbc-callable-statements
	jdbc-connections
	jdbc-dbcam-integration-enabled
	jdbc-prepared-statements
	jdbc-resultsets
	jdbc-statements
	jmx-appserver-mbean-finder-delay-in-seconds
	jmx-operation-timeout-in-milliseconds
	jmx-rediscover-mbean-servers
	jrmp-enable
	log-request-payload
	max-async-task-registration-requests-allowed
	max-async-task-registrations-allowed
	max-business-transactions
	max-service-end-points-per-async-type
	max-service-end-points-per-entry-point-type
	max-service-end-points-per-node
	max-service-end-points-per-thread
	max-urls-per-error-code
	min-load-per-minute-diagnostic-session-trigger
	rmqsegments
	percentile-method-option
	rest-num-segments
	rest-transaction-naming
	rest-uri-segment-scheme
	slow-request-deviation
	slow-request-monitor-interval

	spring-batch-enabled
	socket-enabled
	spring-integration-receive-marker-classes
	spring-mvc-naming-scheme
	thread-correlation-classes
	thread-correlation-classes-exclude
	max-service-end-points-per-node
	max-service-end-points-per-thread
	max-urls-per-error-code
	websocket-entry-calls-enabled
Transaction Snapshots	adaptive-callgraph-granularity
	callgraph-granularity-in-ms
	dev-mode-suspend-cpm
	dont-show-packages
	enable-startup-snapshot-policy
	max-call-elements-per-snapshot
	max-concurrent-snapshots
	max-error-snapshots-per-minute
	max-jdbc-calls-per-callgraph
	max-jdbc-calls-per-snapshot
	min-duration-for-jdbc-call-in-ms
	on-demand-snapshots
	show-packages
	slow-request-threshold
	thread-cpu-capture-overhead-threshold-in-ms
Miscellaneous	collect-user-data-sync

App Agent Node Properties (A)

This reference page contains information about app agent node properties. The properties are listed in alphabetical order.

adaptive-callgraph-granularity

This property enables adaptive snapshots. The call graph granularity for adaptive snapshots is based on the average response time for the business transaction during the last one minute and is thus adaptive. The following distribution is used:

- Granularity of 10 ms for average response time of <= 10 seconds
- 50 ms for 10 to 60 seconds
- 100 ms for 60 to 600 seconds
- 200 ms for > 600 seconds

Type:	Boolean
Default value:	false
Platform(s):	Java, .NET

ado-new-resolvers

Enable database detection and naming for ODP.NET backends labeled `Unknown0`.

Type:	Boolean
Default value:	true
Platform(s):	.NET

agentless-analytics-disabled

Disable agentless Transaction Analytics on a particular node or tier.

Type:	Boolean
Default value:	false
Platform(s):	Java

always-add-eum-metadata-in-http-headers

By default, the Java Agent, .NET Agent, and Node.js Agent set business transaction correlation data in a cookie for HTTP responses, except when the JavaScript Agent has already set an `isAjax`:true header in the request. When it finds the `isAjax`:true header, the agent sets the correlation metadata in the XHR header.

For cross-origin AJAX requests, the JavaScript Agent does not set `isAjax`:true so that the app agent doesn't write correlation data to the header of those responses.

Set `always-add-eum-metadata-in-http-headers` to `true` to configure the app agent to write business transaction metadata to the XHR header and in a cookie even if the request is considered cross-origin.

Type:	Boolean
Default value:	false
Platform(s):	Java, .NET, Node.js

analytics-sql-cpm-limit

This property specifies the per minute upper limit on the number of SQL queries that collect parameter data for analytics. The number is a cumulative total. It is not the number of distinct SQL queries, but the overall number of invocations of the SQL queries that have been configured to collect analytics data.

Type:	Integer
Default value:	10000
Platform(s):	Java, .NET

api-thread-activity-timeout-in-seconds

This property provides a time-out value that comes into play when you have added global transactions to your application using APIs from the AppDynamics SDK. In the event that the added transaction spawns additional threads that do not return or complete, this property provides a safety valve time-out value. The value is in seconds. The removeCurrentThread method is invoked after the specified timeout period.

Type:	Integer
Default value:	300 (seconds)
Range:	Minimum =1; Maximum=3600
Platform(s):	Java

api-transaction-timeout-in-seconds

This property provides a time-out value that comes into play when you have added global transactions to your application using APIs from the AppDynamics SDK. In the event that the added transaction does not return or complete, this property provides a safety valve time-out value. The time-out value is in seconds. The `endTransaction` method is invoked after the specified time-out period.

Type:	Integer
Default value:	300 (seconds)
Range:	Minimum =1; Maximum=3600
Platform(s):	Java

appdynamics-agent-metricLimits

This property increases the metric limit for the .NET Agent for Linux.

Type:	Integer
Default value:	5000
Platform(s):	.NET Agent for Linux

apply-reactive-rules

This property provides a switch to flip the entire reactor instrumentation. If you set it to false, all rules pertaining to reactor thread correlation are unapplied.

Type:	Boolean
Default value:	true
Platform(s):	Java

apply-additional-reactive-rules

This property provides a switch to apply in process correlation rules for prospective thread hand-offs. By default these rules are disabled. These rules can only be applied if 'apply-reactive-rules' node property is set to true. If 'apply-reactive-rules' property is set to false, setting this property to 'true' will not apply any rules.

Type:	Boolean
Default value:	false
Platform(s):	Java

aspdotnet-core-legacy-instrumentation

When you set this property, ASP.NET Core entry instrumentation is restored to `RequestServicesContainerMiddleware.Invoke` for .NET Core 2.1 and 2.2 apps. You can use this when Business Transactions are missing from .NET Core apps after you upgrade an agent.

Type:	Boolean
Default value:	false
Platform(s):	ASP.NET Core

aspdotnet-core-naming-controllerarea

If true, causes the agent to identify ASP.NET Core on the full framework business transactions as Area/Controller/Action. See Name MVC Transactions by Area, Controller, and Action.

Type:	Boolean
Default value:	true
Platform(s):	.NET

aspdotnet-mvc-naming-controlleraction

If true, causes the agent to identify ASP.NET MVC business transactions as Controller /Action. See Name MVC Transactions by Area, Controller, and Action.

Type:	Boolean
Default value:	false
Platform(s):	.NET

aspdotnet-mvc-naming-controllerarea

If true, causes the agent to identify ASP.NET MVC business transactions as Area /Controller. See Name MVC Transactions by Area, Controller, and Action.

Type:	Boolean
Default value:	false
Platform(s):	.NET

async-tracking

Enable or disable detection of asynchronous exit points. See Asynchronous Exit Points for .NET.

Type:	Boolean
Default value:	true
Platform(s):	.NET

async-transaction-demarcator

This class name and method name combination marks the end of an asynchronous distributed transaction. Use the format ClassName/MethodName. For example, foo/bar where foo is the class name and bar is the method name.

Type:	String
Default value:	none
Platform(s):	Java

App Agent Node Properties (B-C)

This reference page contains information about app agent node properties. The properties are listed in alphabetical order.

bci-log-config

Use this property to configure the bytecode transformer log (BCT log). This log shows what AppDynamics instruments and what classes are loaded in the JVM. The initial file (the *0.log*) is saved to preserve the context of the server startup and is not rolled over. The subsequent files rotate. The format of the file name is ByteCodeTransformer.*<timestamp>*.*<N>*.log. The time stamp is represented as `YYYY_MM_DD_HH_mm_ss`. N increments starting from zero.

Type	String
Default value:	20,5,4
Platform(s):	Java

Examples

```
ByteCodeTransformer.2012_09_12_20_17_57.0.log
```

Because the file size is checked every 15 seconds, the files may be a bit larger than the specified threshold value before they are rolled over.

The first log file generated is named as follows: `ByteCodeTransformer.<timestamp>.0.log`

The format for this property value is illustrated by the default value, `20,5,4`. The numbered segments have the following meaning:

- 20 is the size, in MB, of the first log file, the *.0* version
- 5 is the size in MB for each subsequent rolling file
- 4 is the number of ByteCodeTransformer log files to keep

bciengine-disable-retransformation

Disable or enable bytecode retransformation. By default, the Sun and JRockit variant of the Java Agent applies configuration changes requiring bytecode retransformation, such as new POJO rules, without a restart to the JVM. Set `bciengine-disable-retransformation` to `true` to prevent the agent from performing automatic retransformation to apply such rules.

Type	Boolean
Default value:	false
Platform(s):	Java

boot-amx

If set to true, enables support for Glassfish AMX MBeans. See GlassFish Startup Settings fo r more information about Glassfish server support.

Type	Boolean
Default value:	false
Platform(s):	Java

callgraph-granularity-in-ms

Specifies the granularity for call graphs for this node. The global configuration is ignored if this property is used. This value is ignored if the adaptive-callgraph-granularity property is set to true. A default value of zero means the global configuration, from **Configuration > Instrumentation > Call Graph Settings** is used. Does not need a restart.

Type	Integer
Default value:	0
Range:	Minimum=0; Maximum=5000
Platform(s):	Java, .NET

capture-404-urls

This property disables or enables the capture of the URLs causing 404 errors. The URLs are reported as ERROR events every 15 minutes and are viewable through the Event Viewer. The JVM needs a restart if retransformation is not supported for the JVM version.

404 errors usually mean that no application code is executed, resulting in nothing to be captured in a snapshot. You can get insight into the 404 error by setting this property to true. It reports all the URLs which caused 404 error.

The `capture-404-urls` node property is *deprecated* in AppDynamics v. 3.6 and replaced with `capture-error-urls`.

Type	Boolean
Default value:	false
Platform(s):	Java

capture-error-urls

This property enables or disables the capture of the following HTTP errors:

- 401 - Unauthorized
- 500 - Internal Server Error
- 404 - Page Not Found
- All other error codes are put in a generic HTTP error code bucket.

For these four categories, the agent collects URLs, limited to 25 per category per minute, and sends an event out every 5 minutes.

You can see these URLs when you drill down on an error code by clicking Troubleshoot > Errors > Exceptions tab > HTTP Error Codes.

Type	Boolean
Default value:	true
Platform(s):	Java

capture-raw-sql

If `capture-raw-sql` is enabled, SQL calls with dynamic parameters—such as question mark parameters—are captured by the agent and shown in the Controller UI with the dynamic parameters bound to their runtime values.

Type	Boolean
Default value:	false
Platform(s):	Java, .NET

Examples

For example, consider Java code that constructs a SQL call as follows:

```
stmt = new PreparedStatement("select * from user where ssn = ?")
stmt.bind(1, "123-123-1234")
stmt.execute()
```

With capture-raw-sql enabled, AppDynamics captures the SQL call in the following form:

```
select * from user where ssn = '123-123-1234'
```

If `capture-raw-sql` is disabled, the SQL call appears with question mark parameters not bound to values.

Disabling `capture-raw-sql` and using question mark parameters in SQL prepared statements gives you a mechanism for preventing sensitive data from appearing in the Controller UI.

It is important to note that the sensitive values must be parameterized in the original, prepared statement form of the SQL statement, as shown above. The following statement would result in the potentially sensitive information—the ssn value—appearing in the UI whether `capture-raw-sql` is enabled or disabled.

```
stmt = new PreparedStatement("select * from user where ssn ='123-123-1234'")
```

If you change this node property in an environment that is using the IBM JVM, you need to restart the JVM. This is because the feature requires retransformation of certain JDBC classes, which is not possible with the IBM agent.

Setting the option as an agent property affects the SQL capture mode for the node. You can configure the behavior for all nodes using the Capture Raw SQL option described in Call Graph Settings.

capture-set-status

Directs the agent to capture errors where the webservice is using setStatus() to send back an error. By default, only `sendError()` is instrumented by the agent.

Type	Boolean
Default value:	false
Platform(s):	Java

capture-spring-bean-names

When a class is mapped to multiple Spring bean names, by default only the name of the first Spring bean found displays. This can be misleading. For example, when you see a call graph for web service A that has Spring beans from web service B. Setting this property to false shows only the class name when these conflicts occur and does not show the Spring bean name.

Type	Boolean
Default value:	true
Platform(s):	Java

check-bt-excludes-early

Reverses the default order in which Java and .NET agents evaluate rules. If true, exclude rules are evaluated before match rules.

Type	Boolean
Default value:	false
Platform(s):	Java, .NET

collect-user-data-sync

Collect user data from diagnostic POJO data collectors synchronously. Does not require a restart.

Type	Boolean
Default value:	true
Platform(s):	Java

collection-capture-period-in-minutes

Total interval in minutes since server restart for which collections are captured for leak evaluation. The property takes effect only after the node restart.

Type	Integer
Default value:	30
Range:	Minimum=5; Maximum=N/A
Platform(s):	Java

App Agent Node Properties (D-E)

This reference page contains information about app agent node properties. The properties are listed in alphabetical order.

disable-agent

This property enables/disables the agent. When this property is set to true, TransactionEntryPoints will not be monitored. No new BTs or metrics will be registered, metrics, and snapshots will not be reported. Agent background threads will not be stopped. When set to false, the agent becomes active immediately. It does not need a restart.

Type:	Boolean
Default value:	False
Platform(s):	Java

disable-agent-api

This property disables all calls to the agent-api library. The calls function as no-ops when disabled.

Type:	Boolean
Default value:	False
Platform(s):	Java

disable-custom-exit-points-for

Disables certain types of automatically detected exit points: SAP, Mail, LDAP, and so on. Type names are case sensitive. Use commas to separate multiple types.

Type:	String
Default value:	none
Supported values:	CASSANDRA, Coherence, DangaMemcache, EHCache, LDAP, Memcache, MongoDB, RABBIT_MQ, REDIS, RMI, SAP, THRIFT
Platform(s):	Java

disable-exit-call-correlation-for

Disable exit call correlation for a specific type of call. For example, KAFKA, WCF, WEB_SERVICE, HTTP, JMS, and RMI. By default, all exit call correlations are enabled.

Type:	String
Default value:	none
Platform(s):	Java, .NET

disable-exit-call-metrics-for

Disables exit call monitoring for a specific type of exit call; for example, HTTP, JMS, WEB_SERVICE. If this property is set, the average data—calls/min, avg response time—for the specific exit call type is not collected. However, for a snapshot, all details are collected. Set this property if the application makes a large number of exit calls per transaction and the avg metrics are not important.

Type:	String
Default value:	By default, all exit call metrics are enabled.
Platform (s):	Java, .NET

disable-percentile-metrics

App agents that support percentile metrics enable collection by default. Disable percentile metrics on the Configuration > Slow Transaction Thresholds window or set this node property manually to `true` to disable percentiles.

Changes to this property do not require an agent restart.

Type:	Boolean
Default value:	false
Platform(s):	Java, .NET

disable-service-monitoring-metrics

This property stops the BT-to-BT incoming cross-application metrics to be reported (but normal cross-application metrics continue to be reported, as expected).

Type:	Boolean
Default value:	True
Platform(s):	Java

disable-soap-header-correlation-non-http

This property controls correlation with web service transactions. When enabled, it prevents i njection of the correlation header into a SOAP message if the WCF transport is not over HTTP or HTTPS.

Type:	Boolean
Default value:	false
Platform(s):	Java, .NET

For more information, see WCF Entry Points

disabled-features

Specifies types of data for which agent reporting to suppress. Use this property provides to disable data collection mechanisms at the agent to limit data reported by the agent for security or privacy reasons. This agent configuration overrides any Controller configuration that affects the data.

You can disable

- `LOG_PAYLOAD`: Log payload, such as the node property `log-request-payload`
- `RAW_SQL`: Raw SQL statements
- `CUSTOM_EXIT_SNAP_DATA`: Snapshot data in custom exits
- `METHOD_INV_DATA_COLLECTOR`: Diagnostic data collectors, method invocation
- `HTTP_DATA_COLLECTOR`: Diagnostic data collectors, HTTP requests
- `INFO_POINT`: Information points
- `ALL`: All of the above
- `NONE`: None of the above. This is equivalent to the default agent behavior.

Type:	String
Default value:	none
Platform(s):	Java

Examples

Configure disabled features in the app-agent-config.xml in the versioned conf directory for the agent. Specify the data category as the value attribute of the disabled-features property. You can have multiple data categories excluded by listing each separated by commas. For example:

With `capture-raw-sql` enabled, AppDynamics captures the SQL call in the following form:

```
<app-agent-configuration>
  <configuration-properties>
        ....
        <property name="disabled-features" value="RAW_SQL,LOG_PAYLOAD"/>
  </configuration-properties>
  .....
</app-agent-configuration>
```

disable-ibmbpm-data-collectors

Sets whether data-collectors for IBM-BPM task business transactions should be disabled (v alue=true) or enabled (value=false).

Type:	Boolean
Default value:	false
Platform(s):	Java

disable-ibmbpm-usertask-bt-in-process-correlation

Sets whether business transactions in-process correlation for IBM-BPM UserTask business transactions should be disabled (value=true) or enabled (value=false).

Type:	Boolean
Default value:	false
Platform(s):	Java

disable-ibmbpm-usertask-bt-naming

Sets whether business transactions naming scheme for IBM-BPM UserTask Business Transactions should be disabled (value=true) or enabled (value=false).
When it is set to `true`, the Business Transactions would be named as per the default URL and not the meaningful names.

Type:	Boolean
Default value:	false
Platform(s):	Java

dev-mode-suspend-cpm

The maximum number of transactions monitored per minute during development mode before the system switches out of development mode into normal operation mode.

Type:	Integer
Default value:	500
Range:	Minimum=0; Maximum=N/A
Platform(s):	Java

dont-show-packages

Do not show these packages / class names in addition to the ones configured in the global call graph configuration, for the call graphs captured on this node. Does not need a restart.

Type:	String
Default value:	none
Platform(s):	Java, .NET

downstream-tx-detection-enabled

If the agent cannot reach the controller for a prolonged period, it turns off most services and notifies the continuing tiers that upstream transaction was detected and is not being monitored. Set this property to `true` to enable the continuing tiers to detect their own transactions in the event of network failure on the upstream tiers.

Type:	Boolean
Default value:	false
Platform(s):	Java, .NET

enable-all-rsd-error-propagation

This property enables the agent to recognize errors that occur in calls outside of or tangential to business transactions.

`false`: All business transaction errors are reported, but errors outside of the business transaction are not reported.

`true`: All business transaction errors are reported. In addition, errors outside of the business transaction are reported. For example, errors generated from tracking analytics for a business transaction are reported.

`full-disable`: No asynchronous errors are recognized.

Type:	Boolean
Default value:	false
Platform(s):	Java, .NET

enable-async-service-endpoints

By default the Java Agent automatically detects service endpoints for worker threads. Set this property to "false" to disable service endpoint detection for worker threads. This has the same effect as the *Automatic Service Endpoint Detection* checkbox for other types of service endpoints on Configuration > Instrumentation > Service Endpoints.

Type:	Boolean
Default value:	true
Platform(s):	Java

enable-async2-eum-context-lookup

When EUM is enabled on applications implemented using the Servlet3 async dispatch mechanism, it is possible that multiple EUM correlation headers are added to the response.

This node property prevents duplicate ADRUM headers being injected in the HTTP response from Java Agent.

Type:	Boolean
Default value:	false
Platform(s):	Java

enable-axon-entry

This node property is used to disable axon entry. By default, it is enabled.

Type:	Boolean
Default value:	true (By default, axon entry is enabled.)
Platform (s):	Java

enable-axon-exit

This node property is used to disable axon exit. By default, it is enabled.

Type:	Boolean
Default value:	true (By default, axon entry is enabled.)
Platform (s):	Java

enable-bt-block-wait-time-monitoring

This property controls capture of per BT block and wait time metrics. It is disabled by default.

Type:	Boolean
Default value:	false - (Disabled by default)
Platform(s):	Java

enable-bt-cpu-time-monitoring

This property controls whether the agent captures the CPU time taken by a business transaction.

Type:	Boolean
Default value:	true
Platform(s):	Java

enable-default-http-error-code-reporter

This property disables or enables automatic HTTP error code reporting for error codes between 400 to 505.

Type:	Boolean
Default value:	true
Platform(s):	Java, .NET

enable-info-point-data-in-snapshots

This property disables or enables the capture of information point calls in snapshots. When this property is set to true, information point calls appear in the User Data section of the snapshot.

Type:	Boolean
Default value:	false
Platform(s):	Java, .NET

enable-instance-monitoring

This property enables or disables Instance tracking on this node. Does not need a JVM restart.

Type:	Boolean
Default value:	false
Platform(s):	Java

enable-interceptors-for-security

This property enables or disables security interceptors on this node. Set this property to true in environments where the Java 2 Security Manager is enabled. If the Java 2 Security Manager is enabled, and this property is not set to true, then the agent will encounter SecurityExceptions, and will not be able to collect the data that it should. Does not need a JVM restart.

Type:	Boolean
Default value:	false
Platform(s):	Java

enable-json-bci-rules

Set this property to true to enable JSON bytecode instrumentation rules. AppDynamics instruments the `get` and `getString` methods within the package/class `org.json.JSONObject` when you set this value to true. Needs a JVM restart.

Type:	Boolean
Default value:	true (This only affects new applications; applications created with a 3.7.x controller will still have this property set by default to false.)
Platform(s):	Java

enable-kafka-consumer

Set the `enable-kafka-consumer` to `true` to enable Apache Kafka consumer entry points. For more information see 'Apache Kafka Backends' on Java Backend Detection.

Type:	Boolean
Default value:	false
Platform(s):	Java

enable-object-size-monitoring

This property is related to Automatic Leak Detection (ALD) and enables or disables Object Size monitoring on this node. Changing this property does not need a JVM restart. ALD is supported for JVM version 1.6 and up.

Type:	Boolean
Default value:	false
Platform(s):	Java

enable-soap-header-correlation

This property controls correlation with web service transactions. When enabled, a node which receives a web service transaction may correlate that transaction with any downstream transactions. The ability to correlate depends on the particular web services framework. Currently, correlation is supported only by Apache Synapse and CXF frameworks. When disabled, the agent will not perform correlation through any web service tiers.

Type:	Boolean
Default value:	false
Platform(s):	Java, .NET

enable-spring-integration-entry-points

This property disables or enables the default detection of Spring Integration entry points. Set to `false` to disable.

Default detection of Spring Integration entry points is based on MessageHandler. In cases where a lot of application flow happens before the first MessageHandler is executed:

- Set this property to `false`
- Configure suitable POJO entry points
- Specify the property spring-integration-receive-marker-classes

See also Spring Integration Support.

Type:	Boolean
Default value:	true
Platform(s):	Java

enable-spring-ws-dom-parser-rules

This property enables to split web service business transactions on the SOAP XML payload. Set the value to true to enable the property. You can modify namespace context mapping (mapping of the prefix to URI) to parse SOAP message property in the `app-agent-config.xml` file.

Type:	Boolean
Default value:	false
Platform(s):	Java

enable-startup-snapshot-policy

This property disables or enables the policy for start-up transaction snapshot. This means snapshots are collected for all BTs for all invocations for the first 15 minutes of application server startup.

Type:	Boolean
Default value:	false
Platform(s):	Java, .NET

enable-transaction-correlation

This property disables or enables transaction correlation. It does not require a restart.

Type:	Boolean
Default value:	true
Platform(s):	Java, .NET

enable-vertx-http

Enable or disable servlet HTTP entry points and exit points for Vert.x.

Type:	Boolean
Default value:	true
Platform(s):	Java

enable-vertx-message-entry

Enable or disable Vert.x verticle message entry points for continuing transactions.

Type:	Boolean
Default value:	true
Platform(s):	Java

enable-xml-bci-rules

This property enables Java XML Binding and DOM Parser bytecode instrumentation rules.
Set to `true` to enable. The change takes effect after a JVM restart.

Type:	Boolean
Default value:	true (This only affects new applications; applications created with a 3.7.x Controller will still have this property set by default to false.)
Platform(s):	Java

end-to-end-message-latency-threshold-millis

Enables end-to-end message latency monitoring for distributed asynchronous systems by setting up a threshold. Any message taking more time than the threshold is viewable through the Event Viewer.

Type:	Integer
Default value:	0
Range:	Minimum=0; Maximum=36000
Platform(s):	Java

exceptions-to-ignore

By default, configuring an exception to be ignored in the error detection settings prevents the ignored exception thus marking any Business Transaction that experiences it as in error. However, it does not prevent the occurrences of the exception being tracked in the Errors and Exceptions dashboard.

To completely ignore certain exceptions, provide their fully qualified class names in this node property.

For example, oracle.jdbc.DatabaseException.

Multiple exceptions can be added that are comma-separated.

It is recommended to use this property with caution because it removes all visibility of the impact of exceptions configured in this way.

Type:	String
Default value:	0
Platform(s):	Java

App Agent Node Properties (F-I)

This reference page contains information about app agent node properties. The properties are listed in alphabetical order.

find-entry-points

Set this property to `true` to log all potential entry points that are hitting instrumented exit points or loggers to the Business Transactions log file.

Use this property when you suspect that some traffic is not being detected as business transactions. You should only enable this property for debugging purposes. You should deactivate this property in a production setup.

Tip: For new applications, you can use the interactive Live Preview tools to discover entry points.

Type:	Boolean
Default value:	false
Platform(s):	Java, .NET

heap-storage-monitor-devmode-disable-trigger-pct

The maximum Java heap utilization percentage for development mode. If the heap utilization exceeds this value, development mode is automatically deactivated.

Type:	Integer
Default value:	90
Range:	Minimum=0, Maximum=100
Platform(s):	Java

ibmbpm-systemtask-bt-naming

Decides BT naming scheme for IBM-BPM System Task POJO business transactions.

The property value is comma-separated identifiers chosen from *project*, *bpd*, *task*, and *implementation*, and is order-sensitive.

Type:	String
Default value:	default
Platform(s):	Java

Example

For example:

value = `project, task` means that the POJO business transactions are named as: `<project-name> / <task-name>`
value = `task, project` means that the POJO business transactions are named as: `<task-name> / <project-name>`
value = `none` means that the POJO business transactions are not detected, in other words, deactivated.
value = `default` would mean that all the identifiers are used for BT naming are in the default order, that is, they are named as:

```
<project-name> / <bpd-name> / <task-name> / <implementation-name>
```

App Agent Node Properties (J-L)

This reference page contains information about app agent node properties. The properties are listed in alphabetical order.

jdbc-callable-statements

Use this property to indicate the implementation classes of the `java.sql.CallableStatement` interface that should be instrumented.

Type:	String
Default value:	none
Platform(s):	Java

Examples

For example, to instrument calls to Times Ten (an unsupported database), you could set this property to the following:

```
com.timesten.jdbc.JdbcOdbcCallableStatement
```

jdbc-connections

Use this property to indicate the implementation classes of the `java.sql.Connection` interface that should be instrumented.

Type:	String
Default value:	none
Platform(s):	Java

Examples

For example, to instrument calls to Times Ten (an unsupported database), you could set this property to the following:

```
com.timesten.jdbc.JdbcOdbcConnection
```

jdbc-dbcam-integration-enabled

Use this property to integrate the Java Agent with AppDynamics for Databases and Database Monitoring. Changes to this property do not require a JVM restart for JDK 1.6 and higher. Older 1.5 JVMs do not support class reloading, so for those environments, a restart is required.

Type:	Boolean
Default value:	false
Platform(s):	Java

More Info

AppDynamics for Databases: When this property is enabled, you can link to AppDynamics for Databases from a Transaction Snapshot Flow Map where an exit call is to an Oracle database, and analyze the SQL statements that were running at the time of the snapshot. This property works in conjunction with the AppDynamics for Databases license and database collector that has been previously set up. Integration must also be set up from the Admin pages of the Controller UI. For more information, see Integrate and Use AppDynamics for Databases with AppDynamics Pro.

Database Monitoring: Set this property to enable snapshot correlation between Java applications and Oracle databases. This property configures Globally Unique Identifier (GUID) session tagging between business transactions monitored by the Java Agent and Oracle databases monitored by Database Monitoring. Each snapshot is identified by a GUID which Database Monitoring instrumentation injects into the Oracle session using a standard JDBC API. This enables AppDynamics to collect the session properties, including the GUID. When the queries are correlated with the GUID, AppDynamics can correlate the backend database activity with the business transaction snapshot.

jdbc-prepared-statements

Use this property to indicate the implementation classes of the `java.sql.PreparedStatement` interface that should be instrumented.

Type:	String
Default value:	none
Platform(s):	Java

Examples

For example, to instrument calls to Times Ten (an unsupported database), you could set this property to the following:

```
com.timesten.jdbc.JdbcOdbcPreparedStatement
```

jdbc-resultsets

Use this property to indicate the implementation classes of the `java.sql.ResultSet` interface that should be instrumented.

Type:	String
Default value:	none
Platform(s):	Java

Examples

For example, to instrument calls to Times Ten (an unsupported database), you could set this property to the following:

```
com.timesten.jdbc.JdbcStatement
```

jdbc-statements

Use this property to indicate the implementation classes of the `java.sql.Statement` interface that should be instrumented.

Type:	String
Default value:	none
Platform(s):	Java

Examples

For example, to instrument calls to Times Ten (an unsupported database), you could set this property to the following:

```
com.timesten.jdbc.JdbcOdbcStatement
```

jmx-appserver-mbean-finder-delay-in-seconds

When an app server starts up, the associated MBean server starts and the MBeans are discovered. The timing of these activities varies by app server and by configuration. If this activity is not completed in the time that the AppDynamics agent is expecting to discover the MBeans, then the MBean Browser will not show them. Using this node property, you can delay the discovery of MBeans to make sure that agent discovers all the domains after complete startup of the app server. For example, you can set the delay to a time which is 1.5 times of the server startup time. The default delay for the AppDynamics agent is two minutes.

Type:	Integer
Default value:	120
Platform(s):	Java

jmx-operation-timeout-in-milliseconds

Controls the length of time the Java Agent waits before timing out an MBean operation. If you have MBean operations that run longer than the default of 10 seconds, you can increase the timeout value up to 5 minutes.

Type:	Integer
Default value:	10000 (10 seconds)
Platform(s):	Java

jmx-rediscover-mbean-servers

When an app server starts up, the associated MBean server starts and the MBeans are discovered. The timing of these activities varies by app server and by configuration. If this activity is not completed in the time that the AppDynamics agent is expecting to discover the MBeans, then the MBean Browser will not show them. Using this node property, you can trigger the rediscovery of MBeans to make sure that the agent discovers all the domains after complete startup of the app server. Set this property to `true` and reset the app agent, as described in Manage App Agents.

Type:	Boolean
Default value:	false
Platform(s):	Java /td>

jmx-rediscover-mbean-servers-periodic-in-min

This property helps to trigger the rediscovery of Mbeans periodically for every "x" mins (x being the value specified by this node property). When enabled, this property causes the agent to refresh its Mbean server cache for every specified amount of time. By default, no periodic rediscovery of Mbean servers is done. Set the value of this property to 0 to stop the periodic rediscovery task if enabled.

Type:	String
Default value:	0
Platform(s):	Java

jmx-query-timeout-limit

This property sets the timeout limit. Setting this property to negative will disable timeout. For example, if you set the property to –1, timeout is disabled.

Type:	Long
Default value:	5
Platform(s):	Java

jrmp-enable

This property enables or disables AppDynamics support for Sun RMI over Java Remote Protocol (JRMP). You should test Sun RMI JRMP support in a staging environment before using it on production systems. Enable Sun JRMP support by setting this property to `true`.

Type:	Boolean
Default value:	false
Platform(s):	Java

leak-diagnostic-interval-in-minutes

The interval at which diagnostic data, content summary and activity trace, is captured for leaking collections.

Type:	Integer
Default value:	30
Range:	Minimum=2; Maximum=N/A
Platform(s):	Java, .NET

log-request-payload

Set this property to true to log the request payload—HTTP parameters, cookies, session keys, and so on— as part of a transaction snapshot. The log-request-payload property includes logging of WCF HTTP parameters for .NET.

This property dumps all the HTTP data in the app agent logs. But data collectors "diagnostic data collectors" HTTP type are specific to show only the particular HTTP data and not everything, which is: cookies, header, session. The node property gives a complete HTTP dump.

Type:	Boolean
Default value:	false
Platform(s):	Java, .NET

logbased-visibility-log-check-interval-in-millis

How often the agent checks application log files for information related to garbage collection performance.

Type:	Integer
Default value:	1000 ms
Platform(s):	Java, .NET

App Agent Node Properties (M)

This reference page contains information about app agent node properties. The properties are listed in alphabetical order.

In general, be cautious when modifying the agent default settings. After increasing limits specified for an agent, you need to assess and monitor memory consumption by the agent after the change.

maximum-activity-trace-stack-depth

This property determines the depth of the stack trace to capture as part of an activity trace session. By default, the size of the code paths for OIT (Object Instance Tracking), ALD (Automatic Leak Detection) and MIDS (Memory Intensive Data Structures) are set to 10. To increase this limit, use this property.

Warning: A larger depth has higher overhead on the system. AppDynamics recommends that you increase the default value of this property only temporarily, and remove it or set it back to 10 once you get the desired output.

Type:	Integer
Default value:	10
Platform(s):	Java

max-analytics-collectors-allowed

This property sets an upper limit for Analytics collectors in the agent.

Type:	Integer
Default value:	2000
Platform(s):	Java, .NET

max-async-task-registration-requests-allowed

Adjust this property to increase or decrease the number of asynchronous task registration requests allowed.

Type:	Integer
Default value:	500
Platform(s):	Java

max-async-task-registrations-allowed

Adjust this property to increase or decrease the number of asynchronous task registrations allowed.

Type:	Integer
Default value:	500
Platform(s):	Java, .NET

max-business-transactions

Sets a limit on the number of business transactions discovered once an agent is started. The limit helps to ensure that the Controller I/O processing capability and agent memory requirements are appropriate for a production environment. See Business Transactions.

Warning: Changing this setting can affect the resource consumption of your deployment. Before you change this setting, verify that your application environment and Controller can handle any increased resource requirements.

Type:	Integer
Default value:	50
Range:	Minimum=N/A; Maximum=300
Platform(s):	Java, .NET, Node.js, SAP/ABAP, C++ SDK (any non-proxied Dynamic Language Agent)

max-call-elements-per-snapshot

This property represents the maximum number of elements that are collected for any call graph for a snapshot. When the limit is reached, the agent stops collecting more data for this call graph, reports what has been collected to that point, and marks the call graph with a warning that the limit was reached. AppDynamics does not recommend dramatically increasing this number as it may have overhead implications.

Type:	Integer
Default value:	5000
Platform(s):	Java, .NET

max-concurrent-snapshots

The maximum number of total snapshots that are allowed, including continuing transactions. When the queue goes over the value set, additional snapshots are dropped. This property is ignored in Development mode.

Type:	Integer
Default value:	20
Range:	Values must be positive integers. No other constraints.
Platform(s):	Java, .NET

max-correlation-header-size

The maximum size of the correlation header.

Type:	Integer
Default value:	4096
Range:	Values must be positive integers. No other constraints.
Platform(s):	Java

max-error-snapshots-per-minute

A limit for the number of snapshots per minute due to errors. For example, if too many error snapshots are being seen, then tweak this value to reduce noise in the snapshots.

Type:	Integer
Default value:	5
Platform(s):	Java, .NET

max-jdbc-calls-per-callgraph

The maximum number of JDBC/ADO.NET exit-call stack samples per call graph. Only queries taking more time than the value of `min-duration-for-jdbc-call-in-ms` are reported. Changing the value does not require a restart.

Type:	Integer
Default value:	100
Range:	Minimum=1; Maximum=1000
Platform(s):	Java, .NET

max-jdbc-calls-per-snapshot

The maximum number of JDBC/ADO.NET exit calls allowed in a snapshot. Calls after the limit are not recorded. Changing the value does not require a restart.

Type:	Integer
Default value:	500
Range:	Minimum=1; Maximum=5000
Platform(s):	Java, .NET

max-metrics-allowed

This property sets the upper limit to the number of metrics that can be registered by the agent.

Note: If the number of metrics—registered + unregistered—is greater than the new node property value, then the new metrics will not be created.

Type:	Integer
Default value:	5000 or as set with – `Dappdynamics.agent.maxMetrics=`
Platform(s):	Java

max-prepared-statement-sql-length

Maximum length of the SQL statement for SQL prepared statements. The length of the SQL statement is truncated if it exceeds the value set by this property.

Type:	Integer
Default value:	500
Range:	Minimum=0; Maximum=NA
Platform(s):	Java, .NET

max-service-end-points-per-async-type

Maximum total number of service endpoints that can be registered for each asynchronous entry point type, such as worker thread. Because a single transaction may spawn many threads, you may expect more asynchronous service endpoint types than for synchronous service endpoint types.

Type:	Integer
Default value:	40
Range:	Minimum=0; Maximum=N/A
Platform(s):	Java

max-service-end-points-per-entry-point-type

Maximum total number of service endpoints that can be registered for each entry point type, such as servlet, struts action, web service, and so on.

Type:	Integer
Default value:	25
Range:	Minimum=0; Maximum=N/A
Platform(s):	Java, .NET

max-service-end-points-per-node

Maximum total number of service endpoints that can be detected on a single node. Increasing the value of this property enables more service endpoints to be detected on a particular node.

Type:	Integer
Default value:	100
Range:	Minimum=0; Maximum=N/A
Platform(s):	Java, .NET

max-service-end-points-per-thread

Maximum number of service endpoints detected on a single thread of transaction execution.

If this property is set to the default value of one and two service endpoints are detected that impact one specific transaction, only one service endpoint will be evaluated at any time. If a second service endpoint is detected in the context of the first one, the second is ignored. But, if the second service endpoint starts after the first one ends, the second service endpoint will be evaluated.

Increase this property to monitor additional service endpoints on a thread. This number ensures a maximum limit on overhead and number of metrics due to service endpoints on each thread execution.

Type:	Integer
Default value:	1
Range:	Minimum=0; Maximum=N/A
Platform(s):	Java, .NET

max-urls-per-error-code

This property increases the number of URLs the agent can track that produced a certain error. Once the maximum has been reached, all remaining errors are classified as unknown.

Type:	Integer
Default value:	50
Platform(s):	Java

min-duration-for-jdbc-call-in-ms

A JDBC/ADO.NET call taking more time than the specified time (in milliseconds) is captured in the call graph. The query continues to show up in a transaction snapshot. Setting this value too low (< 10ms) may affect application response times. Changing the value does not require a restart.

Type:	Integer
Default value:	10
Range:	Minimum=0; Maximum=N/A
Platform(s):	Java, .NET

min-load-per-minute-diagnostic-session-trigger

This property indicates the number of requests per Business Transaction to evaluate before triggering a diagnostic session. This property is useful to prevent diagnostic sessions when there is not enough load.

Type:	Integer
Default value:	10
Range:	Minimum=0; Maximum=N/A
Platform(s):	Java, .NET

minimum-age-for-evaluation-in-minutes

Automatic Leak Detection (ALD) tracks all frequently used Collections. For a Collection object to be identified and monitored, it must meet the conditions defined by the ALD properties. This property is the first criteria that needs to be met. The value is the minimum age of the Collection in minutes. This property takes effect after the node restarts.

From the point the collection is captured, it is monitored if it is still available for the specified period without collecting garbage. If it survives, then it is evaluated for size checks. If it meets the criteria, then it is monitored for long term growth in size.

Warning: If you reduce the default, there may be a performance hit on the CPU and memory because ALD needs to process more collections.

Type:	Integer
Default value:	30
Range:	Minimum=5; Maximum=N/A
Platform(s):	Java

minimum-number-of-elements-in-collection-to-deep-size

Automatic Leak Detection (ALD) tracks all frequently used Collections. For a Collection object to be identified and monitored for it must meet the conditions defined by the ALD properties. This property sets the number of elements threshold.

Warning: If you reduce the default there may be a performance hit on the CPU and memory because AD is processing more collections.

Type:	Integer
Default value:	1000
Platform(s):	Java

minimum-size-for-evaluation-in-mb

Automatic Leak Detection (ALD) tracks all frequently used Collections. For a Collection object to be identified and monitored it must meet the conditions defined by the ALD properties. This property sets the minimum initial size in megabytes for a collection to qualify for monitoring. The collection must also survive for the period specified in the minimum-age-for-evaluation-in-minutes property.

Warning: If you reduce the default there may be a performance hit on the CPU and memory because AD is processing more collections.

Type:	Integer
Default value:	5
Range:	Minimum=1; Maximum=N/A
Platform(s):	Java

min-transaction-stall-threshold-in-seconds

For Executor mode only, the asynchronous transactions do not get checked for stall unless they run at least the specified number of seconds.

Type:	Integer
Default value:	60
Platform(s):	Java

msmq-correlation-field

By default, the .NET agent disables downstream correlation for MSMQ message queues. Register this node property on both the publishing and receiving tiers to enable downstream correlation for MSMQ and to specify the field where the agent writes correlation data.

The agent supports Extension or Label fields. By default, the agent writes correlation data to the Extension field, however, some frameworks built on MSMQ write data to the Extension field. Only use Label when the Extension field is not available because it is already in use by the framework. The NServiceBus implementation of MSMQ uses the Extension field, so for NServiceBus, use Label. See MSMQ Backends for .NET.

Type:	String
Supported values:	• `None`: Disable downstream correlation for MSMQ • `Label`: Store correlation information in the label field • `Extension`: Store correlation information in the extension field
Default value:	None
Platform(s):	.NET

msmq-single-threaded

Specify the threading architecture for the MSMQ message queue. The default value is `false`. For multithreaded queue implementations, change the value to `false`. See MSMQ Backends for .NET.

Type:	Boolean
Default value:	false
Platform(s):	.NET

App Agent Node Properties (N-R)

This reference page contains information about app agent node properties. The properties are listed in alphabetical order.

In general use caution when modifying the agent default settings. If increasing limits specified for an agent, you need to carefully assess and monitor memory consumption by the agent after the change.

normalize-prepared-statements

When this flag is set to `true`, any variables in prepared statement SQL would be substituted with '?' before query text is added to any snapshots.

Type:	Boolean
Default value:	true
Platform(s):	Java

nservicebus-single-threaded

Specify the threading architecture for the NServiceBus message queue. The value defaults to `true`. For multithreaded queue implementations, change the value to `false`. See NServiceBus Backends for .NET.

Type:	Boolean
Default value:	true
Platform(s):	.NET

osb-ignore-exit-types

This comma-separated property defines which exits protocols (as configured in the Transport Configuration pane) are excluded from detection as uncorrelated custom backends.

Set the property to 'all', if you do not want to detect any exit.

Type:	String
Default value:	http,jms
Platform(s):	Java

on-demand-snapshots

Collect snapshots for all Business Transactions executed in this node. Does not need a restart. This property is ignored in the Development mode.

Type:	Boolean
Default value:	false
Platform(s):	Java, .NET

percentile-method-option

You can choose one of two different algorithms to calculate percentiles in AppDynamics:

- P Square algorithm (default): This option consumes the least amount of storage and incurs the least amount of CPU overhead. The accuracy of the percentile calculated varies depending on the nature of the distribution of the response times. You should use this option unless you doubt the accuracy of the percentiles presented.
- Quantile Digest algorithm: This option consumes slightly more storage and CPU overhead but may offer better percentiles depending on how the response times are distributed.

Changes to this property do not require that you restart the agent.

Type:	Numeric
Supported Values:	• 1: P Square • 2: Quantile Digest
Default value:	1
Platform(s):	Java, .NET

queue-single-threaded

Specify the threading architecture for the IBM MQ message queue. The value defaults to `false`.

When the IBM MQ `Get()` method is called, the transaction/snapshot is ended prematurely. To prevent this, set the agent node property: queue-single-threaded=true.

Type:	Boolean
Default value:	false
Platform(s):	Java, .NET

reportingFrequencyInMillis

The agent property specifies the frequency of Request Segment Data (RSD) uploads from the agent.

Type:	Numeric
Default value:	10000 ms
Platform(s):	Java

rest-num-segments

The property, `rest-num-segments` specifies the *n* in the first-*n*-segments parameter in rest-uri-segment-scheme. If this property is 0 or less, then the value of this property is ignored. The value of this property is also ignored if rest-uri-segment-scheme=full.

Type:	String
Default value:	2
Platform(s):	Java

rest-transaction-naming

This node property determines the format in which REST-based business transactions are named. You can use variables to populate the name with values bound at runtime. Any characters in the property value that do not match a variable are treated as literal text in the business transaction name, so you can, for example, separate variables with a colon, slash, or another character.

The agent takes each parameter and fills in the proper value based on the annotations and properties of the Java class:

- `{class-name}`: The app agent will fill in the name of the Java class mapped to the REST resource.
- `{method-name}`: The method being called.
- `{class-annotation}`: Class annotation values.
- `{method-annotation}`: Method annotation applied to the method (not always present).
- `{rest-uri}`: URI of the REST resource. The REST URI is further configured using the following properties:
  - rest-uri-segment-scheme
  - rest-num-segments
- `{http-method}`: HTTP method of the request, GET, POST, and so on.
- `{param-%d}`: A parameter to the method identified by position. Replace %d with the position of the parameter (ZERO-based).

Type:	String
Default value:	{class-annotation}/{method-annotation}.{http-method}
Platform(s):	Java

Examples

See Using Default Settings and Using rest-transaction-naming Properties.

rest-uri-segment-scheme

The property, `rest-uri-segment-scheme` has three valid values: `first-n-segments`, `last-n-segments`, and `full`. This property indicates how many segments of the URI to use for the URI in {rest-uri}. This option is case-sensitive. If the value of this property is `full`, then the value of rest-num-segments is ignored.

Type:	String
Default value:	first-n-segments
Platform(s):	Java

rmqsegments

The App Agent for .NET (agent) automatically discovers RabbitMQ remote services. Use the rmqsegments node property to refine the queue backend name to include some or all segments of the routing key. You must be familiar with your implementation RabbitMQ exchanges and routing keys. See RabbitMQ Exchanges and Exchange Types.

The RabbitMQ routing key is a string. The agent treats dot-separated (".") substrings of the routing key as segments. Set the value for rmqsegments to an integer that represents the number of routing key segments to include in the name. For more details, see information on refining backend naming in RabbitMQ Backends for .NET.

Type:	Numeric
Value:	Integer representing the number of routing key segments to include in the name.
Default value:	0
Platform(s):	.NET

App Agent Node Properties (S)

This reference page contains information about app agent node properties. The properties are listed in alphabetical order.

show-packages

For the call graphs captured on this node, show the specified packages or class names in addition to the ones configured in the global call graph configuration. Does not need a restart.

Type:	String
Default value:	none
Platform(s):	Java, .NET

slow-request-deviation

The value in milliseconds for the deviation from the current average response time. This setting is used for evaluation of slow in-flight transactions. Also, see the `slow-request-threshold` property for more details.

Type:	Integer
Default value:	200
Range:	Minimum=10; Maximum=3600
Platform(s):	Java, .NET

slow-request-monitor-interval

In-flight requests are checked for slowness in the interval specified by this property. The value is specified in milliseconds.

Type:	Integer
Default value:	100
Range:	Minimum=0; Maximum=3600
Platform(s):	Java, .NET

slow-request-threshold

In-flight requests taking more time than this threshold (in ms) with a deviation greater than the `slow-request-deviation` property from the current average response time are monitored to capture hot spots.

Type:	Integer
Default value:	500
Range:	Minimum=0; Maximum=3600
Platform(s):	Java, .NET

socket-enabled

Use this property to enable NetViz monitoring of .NET and Java apps running on Windows.

Type:	Boolean
Default value:	false
Platform(s):	.NET

spring-batch-enabled

Use this property to enable or disable OOTB BT Detection for Spring Batch.

Type:	Boolean
Default value:	true
Platform(s):	Java

spring-integration-receive-marker-classes

Use this property to specify the class and method you have identified as suitable POJO entry points for Spring Integration.

Based on the MessageHandler interface, the App Agent for Java by default automatically discovers exits for all channels except *DirectChannel*. In cases where a lot of application flow happens before the first MessageHandler is executed,

- Set enable-spring-integration-entry-points=false
- Configure suitable POJO entry points,
- Declare each suitable POJO entry point class/method in this property spring-integration-receive-marker-classes

If the application code polls for messages in a loop, the span of each loop iteration is tracked as a transaction. Tracking begins when the loop begins and end it when the iteration ends. To safeguard against cases where `pollableChannel.receive()` is not called inside a loop, specify this property for each class/method combination that polls messages in a loop.

After setting this property, restart the application server for changes to this property to take effect.

Type:	Comma-separated string of fully-qualified class /method name, such as spring-integration-receive-marker-classes = ,<> ....
Default value:	none
Platform(s):	Java

Examples

For example, to enable tracking for the following:

```
class MessageProcessor
{
void process()
{
   while(true)
   {
          Message message = pollableChannel.receive()
   }
}
}
```

set this property as follows:

```
spring-integration-receive-marker-classes = MessageProcesser/process
```

See also Spring Integration Support.

spring-mvc-naming-scheme

Register this node property to modify the naming scheme for Spring MVC transactions. Bean ID cannot be used as a global naming type. Use the bean ID and method name for global.

Type:	String
Allowed values:	bean-id, simple-class-name, fully-qualified-class-name, business-interface-name, bean-method-name, ben-id-and-method-name, class-and-method-name
Default value:	none
Platform(s):	Java

App Agent Node Properties (T-Z)

This reference page contains information about app agent node properties. The properties are listed in alphabetical order.

thread-correlation-classes

For multi-threaded applications, use this property to configure classes to be included in Java thread correlation when simple prefix-matching (matching on `STARTSWITH`) is sufficient to identify the classes.

The `thread-correlation-classes` property specifies the classes to include for thread correlation. This property can be used together with the `thread-correlation-classes-exclude` property.

The configured correlation takes effect without requiring a restart of the managed application.
Also see, Configure the Thread Correlation in Java Agent.

Type:	Comma-separated string of fully-qualified class names or package names
Default value:	none
Platform(s):	Java

This property is not recommended for use in the Executor Mode.

thread-correlation-classes-exclude

For multithreaded applications, use this property to configure classes to be excluded in Java thread correlation when simple prefix-matching (matching on `STARTSWITH`) is sufficient to identify the classes. This property specifies the classes to exclude from thread correlation. This property can be used in conjunction with the `thread-correlation-classes` node property.

The configured correlation takes effect without requiring a restart of the managed application.
See Configure Multithreaded Transactions (Java only)

Type:	Comma-separated string of fully-qualified class names or package names
Default value:	none
Platform(s):	Java

This property is not recommended for use in the Executor Mode.

thread-cpu-capture-overhead-threshold-in-ms

Determines the timeout allotted for collecting and calculating Thread CPU Time for 1000 iterations. If the timeout is exceeded, the Java Agent automatically disables CPU time collection for threads. CPU usage information will then be absent from the BT overview or snapshots in the Controller UI. In addition, an INFO-level log similar to the following appears in the logs:

```
[Thread-0] 22 Oct 2013 14:19:26,346 INFO
JVMThreadCPUTimeCalculator - Disabling BT CPU
Monitoring. Time taken to calculate Thread CPU Time for
[1000] iterations is [15 ms] which is greater than the
allowed budget of [10 ms].
```

This issue may particularly affect JDK 1.6 on Linux due to the issue getCurrentThreadCpuTime is drastically slower than Windows Linux.

Type:	Integer
Default value:	10 ms
Platform(s):	Java

Examples

You can increase the `thread-cpu-capture-overhead-threshold-in-ms` property, but it is important to note that this may result in increased overhead on your application. We recommend you use this Java HotSpot VM option instead to speed up the API call itself:

```
-XX:+UseLinuxPosixThreadCPUClocks
```

Restart is needed after changing this value.

wcf-enable-eum

Enable and disable EUM correlation from a WCF node.

Type:	Boolean
Default value:	false
Platform(s):	.NET

Examples

The enable EUM correlation, the WCF application must also have the following entry in the `Web.Config`:

```
<serviceHostingEnvironment aspNetCompatibilityEnabled="true" />
```

See serviceHostingEnvironment.

websocket-entry-calls-enabled

When set to false, no WebSocket entry calls are detected.

Type:	Boolean
Default value:	true
Platform(s):	Java

Dynamic Language Agent Proxy

Related pages:

- Configure the Agent for PHP CLI Applications
- Use a Shared Proxy for PHP Agents
- Start the PHP Agent Proxy Manually
- Start the Proxy Manually for Node.js
- Share a Proxy Among Node.js Agents
- Start the Python Agent Proxy Manually

The AppDynamics proxy is a Java daemon process that handles communication between the Controller and these agents:

- Node.js
- PHP
- Python
- Web Server

The proxy reports data collected by the agent to the Controller, which stores, baselines, and analyzes it.

This page describes information you can use to examine and resolve proxy issues that may prevent an application agent from connecting to the Controller, or reporting data correctly. For detailed information about how the proxy works with a specific agent, see the appropriate *Related pages*.

Proxy Basics

Single or Multi-Tenant

For the Node.js and PHP agents, the proxy can be single-tenant (one proxy per agent) or multi-tenant (multiple agents communicating through a single proxy), depending on the way the agents are configured. The default setup for these agents is single-tenant, but there are circumstances for which a multi-tenant proxy is required or desirable. The Apache Web Server and Python agents always communicate through a multi-tenant proxy when there are multiple agents on a machine.

Automatic or Manual Startup and Shutdown

Typically the proxy is automatically started when the application agent starts up, but in some cases, you need to launch it manually. These cases vary depending on both the particular agent and the application environment.

After the proxy is started (automatically or manually) it registers with the Controller and requests the agent configuration. The agent must receive the configuration from the Controller via the proxy before it can report metrics.

If the proxy was automatically started, the agent is supposed to shut down the proxy as part of its own cleanup procedures. If the proxy was manually started, it must be shut down manually.

Proxy Logs

If the proxy is running, you can check the proxy log to examine connection issues.

The proxy log files are named `proxy.<timestamp>.log`.

If the proxy is not running, examine the agent logs for the typical Java startup signature to see if the proxy started.

See the agent-specific documentation for the location of these logs:

- Node.js: Default is `/tmp/appd/logs`. See Install the Node.js Agent.
- PHP: Default is `<php_agent_install_dir>/logs`. See Install the PHP Agent.
- Python: Default is `/tmp/appd/logs`. See Python Agent Debugging and Logging.
- Web Server: `<webserver_agent_install>/logs/proxy_$date.lo`

Proxy Not Started Issues

The agent will not work if the proxy did not start.

Determine whether the proxy started by running this command:

```
ps aux|grep java
```

If the proxy is running, you should see `java` and `proxy` in the output, something like this:

```
/usr/lib/appdynamics-php5/proxy/jre/bin/java -server -Xmx120m -classpath /usr/lib/appdynamics-php5/proxy/conf
/logging/*:/usr/lib/appdynamics-php5/proxy/lib/*:/usr/lib/appdynamics-php5/proxy/lib/tp/*:/usr/lib/appdynamics-
php5/proxy/* -Djava.library.path=/usr/lib/appdynamics-php5/proxy/lib/tp -Dappdynamics.agent.logs.dir=/usr/lib
/appdynamics-php5/logs -Dcomm=/tmp/ad-siJ4rp -DagentType=PHP_APP_AGENT -Dappdynamics.agent.runtime.dir=/usr/lib
/appdynamics-php5/proxy com.appdynamics.ee.agent.proxy.kernel.Prox
```

If the proxy did not start, the most common reason is insufficient permissions.

- The agent installation directory, and its proxy control subdirectory, must be readable and executable by all and writable by the directory owner:

  ```
  chmod -R 755 <agent_install_dir>
  ```

- The agent installation directory must be owned by the instrumented application user. Who this user is depends on the platform. It could be the Apache user, the python container user, the nginx user, the `node.js` process, etc.

  ```
  chown -R <appuser>:<appsuser> <agent_install_dir>
  ```

If the proxy did not start, set these permissions and try again.

Proxy Connection Issues

If the proxy started but you don't have a connection to the Controller, you may be using incorrect Controller information.

Examine the proxy log and verify that there are no typos in the Controller domain name or port and that SSL setting is enabled for an SSL connection. If there are mistakes, edit the Controller/port/SSL values in the configuration. These settings are on the first page of the Agent Download and Install Wizard or the agent settings (require statement for Node.js) if you installed the agent manually.

If you have verified that the Controller settings are correct and the proxy still does not connect to the Controller, telnet to the SaaS Controller:

```
telnet <your_account>.saas.appdynamics.com 443
```

If you cannot access the Controller through telnet, examine the agent-registered messages in the logs that indicate why the proxy is unable to connect, such as the existence of a firewall or other obstacle at your site. Work with your administrator to see if you can resolve the issue.

If you can reach the Controller through telnet, there may be a problem on the Controller.

Dynamic Agent Proxy Logging

Proxy logs are created for app server agents that use the Dynamic Agent Proxy to communicate with the Controller. See Dynamic Language Agent Proxy.

The PHP Agent, Node.js Agent (Java proxy mode), Python Agent, and the Web Server Agent generate logs for the Dynamic Agent Proxy. The logs determine why snapshots are not being taken and diagnose communication issues with the Controller.

The proxy logs consist of a top-level file named `proxyCore`*Year_mon_day_hr_min*`.#.log` and files named `proxy`*Year_mon_day_hr_min*`.#.log`, where # is the log set. See Agent Log Files for information about how the logs are organized into sets that roll over.

Other logs associated with each proxy log include:

- Bytecode Transformer log
- REST log
- Dynamic Services (used internally for debugging) log

Each proxy log has a maximum size of five MB.

Allowing for the possibility of several restarts and all the possible logs that could be generated (Proxy, BytecodeTransformer, REST, and Dynamic Services), the maximum number of logs is 20 files (four log files times five sets). The maximum size is 100 MB per set.

Business Applications

As described in Overview of Application Monitoring, a business application serves as the top-level container in the AppDynamics APM model. Business applications contain a set of related business transactions and service endpoints, along with the infrastructure components and artifacts that interact to provide the services.

Organize Business Applications

You can model your environment into one or more business applications. When you have multiple business applications within the same AppDynamics account, you can use the cross application flow map to view relationships between the different applications.

Since access permissions in the Controller UI can be assigned by business application, organizing business application by teams in your organization may make the most sense. Keep in mind that configuration settings such as Health Rules are scoped to a business application, avoid having more business applications than needed to ease configuration.

> When you model your environment, be sure to assign unique names to your business applications. Names must be unique among both Application Performance Monitoring and End User Monitoring.

Permissions

Creating applications requires the *Can Create Applications* permission. Permission to view, edit, or delete applications can be set as part of the default application permissions for a custom role or for specific applications. See Application Permissions.

View Applications

The Applications page, accessible from the menu bar in the Controller UI, provides a high-level view of the business applications in your environment. At a glance, you can see the performance for the applications.

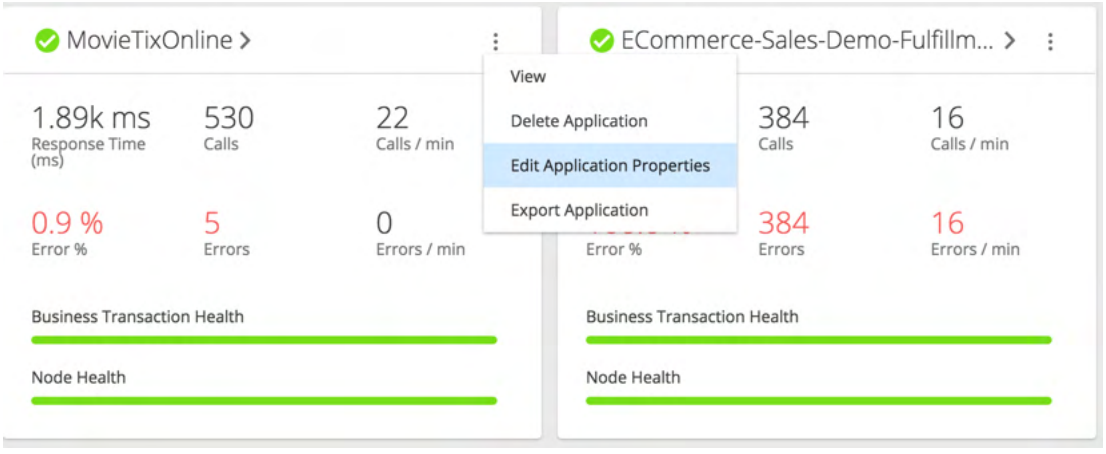
There are several views and sorting options in the page. To view relationships between business application (such as, when a business transaction in one invokes a service in another), select the **Flow View** icon.

Modify Applications

You can create new business applications from the Applications page, but it's not necessary to create the business application manually. The Controller creates a new business application automatically the first time an app agent registers itself using a new business application name.

Rename Applications

You can rename applications by selecting **Edit Application Properties** from the drop-down of an application.



> If you change the name of an application, you also need to change it in the agent configuration at `controller-info.xml`. If you do not change the controller-info.xml application name field, agents will re-create the old application name and report to the old application.

Deleting Applications

To delete an application:

Select the application in the **Applications** drop-down, and select **Actions > Delete**.

The Controller prompts you to acknowledge a list of related items that will be deleted including:

- Number of tiers and nodes
- Number of business transactions (see Business Transactions)
- Browser RUM Apps
- Mobile Apps

Application Performance Details

From the Applications page, click a listed application to configure and monitor the business application. In the business application instance Application Dashboard page, you can view:

- Top Business Transactions: The key performance indicators for the most expensive business transactions sorted by the various criteria, such as load, response time, errors and so on. Click View All in any of the panels in this tab to see a list displaying all the key performance indicators for the top business transactions in one panel.
- Transaction Snapshots: The transaction snapshots for the selected time range. From a transaction snapshot, you call drill down to the root cause of a performance problem.
- Transaction Analysis: Application performance over the selected time range as a graph. Use the graph to analyze the impact of different events on the application response time.
- Machine Snapshots: When an application has at least one .NET tier, this appears as a list of the machine snapshots for the selected time range. From a machine snapshot, you can drill down to view environmental conditions, such as running processes and memory usage for a particular machine. See Machine Snapshots for .NET.

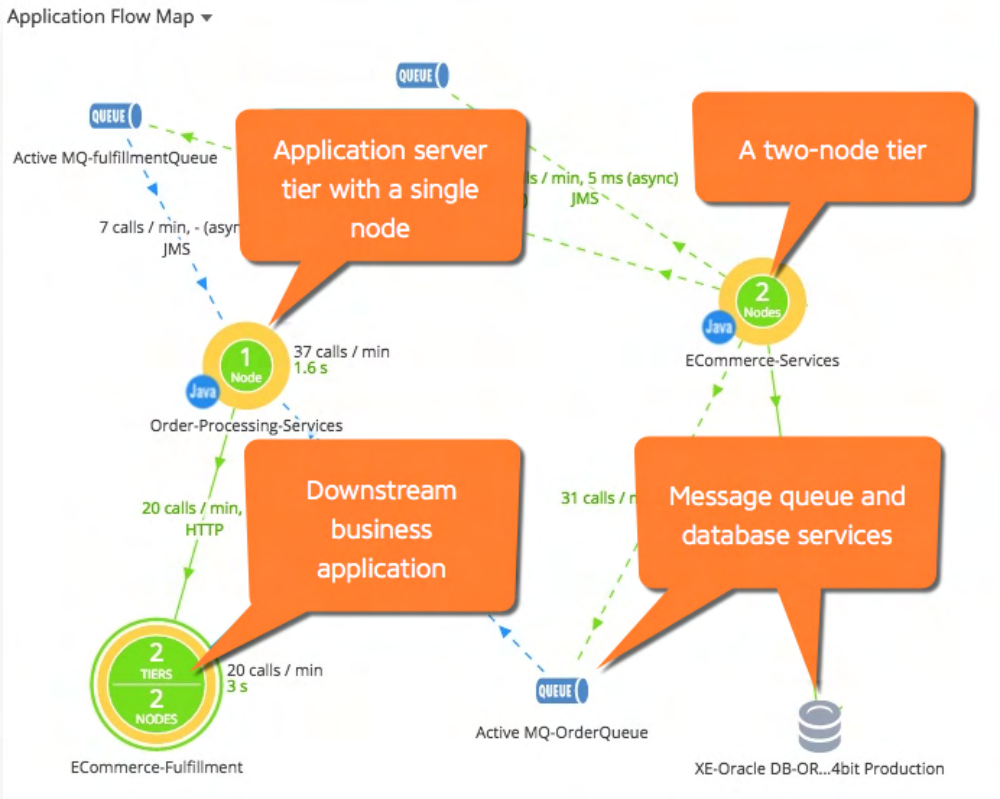
Flow Maps

Flow maps present a dynamic visual representation of the components and activities of your monitored application environment. See Overview of Application Monitoring.

This page describes how to view and use flow maps.

Flow Map Overview

Flow maps show the tiers, nodes, message queues, and databases in the environment, and the business transactions that flow through them. This sample shows a basic flow map for an e-commerce application. In the sample, three server tiers interact with databases and an Apache ActiveMQ message broker.



Connection Types

Flow lines represent connections between components in the flow map. Solid lines indicate synchronous connections, while dashed lines indicate asynchronous connections.

Many modern frameworks use asynchronous patterns even if you do not explicitly call an asynchronous function or method. For example, your application code may employ a synchronous call to a framework or an Object-relational mapping style API, but the framework itself invokes an asynchronous executor to handle the call. These types of asynchronous segments show up as a dotted line on the flow map. For more about asynchronous exit calls, see Trace Multithreaded Transactions for Java or Asynchronous Exit Points for .NET.

Request Times

The numbers above the flow lines indicate the calls made per minute to the tier and the average time taken for the request to be serviced; that is, the round-trip time for the request. The round-trip time includes time spent on the network, if applicable to your topology, and the time that the backend server or other process spends processing the request. The calls per minute for a given context, such as a tier, must be one or more for the flow map to display.

Performance Baselines

If performance baselines are set for transactions represented in the flow map, the flow lines use color to indicate the performance of the service represented by the flow line relative to the baseline. For example, a green flow line indicates that that response times in the time range do not differ significantly from the baseline. A yellow line indicates that response times are slower than the baseline. It takes some time for the Controller to establish baselines for a new installation. If there are no baselines for comparison, the flow lines are blue.

Live Entity Data

By default, the flow map only shows the nodes receiving performance data to optimize the rendering of the flow map and to enable you to quickly view the active nodes. You can set a filter to view the nodes not receiving performance data or all nodes. Choosing to view the nodes not receiving performance data can help you troubleshoot node issues.

When an entity is alive, it will affect other associated entities:

- When a business transaction entity is alive, the associated node, tier, and application will be alive.
- When a node is alive, the associated tier, and application will be alive.
- When a tier is alive, the associated application will be alive.

> If the Controller detects that a flow map is taking a long time to load, it does not load the flow map automatically. In this case, you can click **Show Flow Map** to display the flow map if you choose.

Types of Flow Maps

Flow maps appear in several of the built-in dashboards in the UI, and show different information depending upon the context in which they appear:

- *Cross Application flow maps* show exit calls between applications within the monitored environment. A pattern of such calls is known as cross-application flow.
- *Application flow maps* show the topology and activities within an application. It displays metric values across all business transactions in the application for the selected time range. For example, the application flow map displays calls per minute; average response time for calls made to databases and remote services; and business transaction errors per minute. These metrics are based on all calls made from a specific tier to a database or remote service across all business transactions.
    - When an application is one of several in a one-app-per-service architecture, business transactions can represent service endpoints. For these applications, the application flow map displays a *Cross-BT Hovercard* that shows calls to upstream and downstream business transactions which represent endpoints of other services.
- *Tier and node flow maps* display these metric values across all business transactions for the subset of the application flow related to the selected tier or node.
- *Business transaction flow maps* show the activity for a business transaction. The START label indicates the tier where the transaction starts (the originating tier). The business transaction flow map shows metrics that are calculated based on all executions of the business transaction during the selected time range.
- Snapshot flow maps illustrate the metrics associated with a single snapshot. The metrics values shown in the map are specific to a particular execution of the transaction.

AppDynamics shows cross-application flow on all flow maps where appropriate. For example, a tier flow map shows correlation when there are exit calls from the tier to another instrumented application.

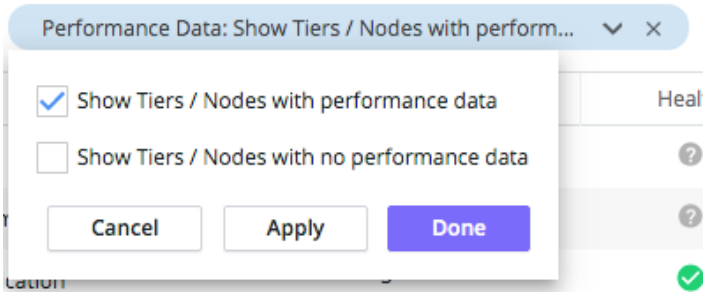
Context can determine the meaning of what flow maps represent. For example, consider the average response time (ART) for calls to a database.

- In the context of an application, ART averages all calls to the database that happen *within the application*. Suppose that the application flow map shows that ART for those calls is 20 milliseconds.
- In the context of a business transaction, ART represents average execution time *for the business transaction as a whole*. Suppose that in a BT within the same application, the database gets called twice each time the business transaction executes. In the business transaction flow map, ART represents the average duration of *a pair of* database calls, meaning that ART is 40 milliseconds.

Flow Map Interactions

On flow maps you can:

- Click items in the flow map to see key performance indicators in an informational popup. For some components of the flow map, like tiers, the popup displays additional details.
- Change the Time Range setting to have the flow map represent the activity of the system within the selected time frame.
- For SaaS Controllers, note that the flow maps show a maximum of the last 60 minutes of data, even if the time range in the UI is set to a greater range. This only applies to flow maps; the data in other graphs on the dashboard represent the selected time range.
- Click **Legend** to learn more about how flow maps represent data.
- Use a filter to view tiers and nodes (all associated entities) receiving performance data, not receiving performance data, or all nodes.
    - Select **Show Tiers / Nodes with performance data** to show entities and metrics for live entities over the specified time range.
    - Select **Show Tiers / Nodes with no performance data** to show metrics/entities of entities that are not alive over the specified time range.
    - Select both **Show Tiers / Nodes with performance data** and **Show Tiers / Nodes with no performance data** checkboxes to show metrics and entities without checking the liveness of entities.



- Drag and drop items to rearrange the flow map layout or use automatic arrangement options using the controls at the top right of the flow map:

  Use the controls to view the mapped components as a list, auto arrange the flow map—in which components are arranged for the fewest crossing flows—maximize the view, and more.

The Cross-BT Hovercard in the Application Flow Map

For an application in a one-app-per-service architecture, the application flow map displays a *Cross-BT Hovercard* showing calls from upstream and to downstream business transactions representing endpoints of other services.

This enables you to isolate the service which is the origin of any problem that arises, to one of three possibilities:

1. The service whose endpoint is a BT that you are monitoring ("your" service)
2. An upstream or downstream service, that "your" service communicates with through cross-BT calls
3. Yet another upstream or downstream service, that "your" service communicates with indirectly through a chain of cross-BT calls between several services

To access the Cross-BT Hovercard, select the line representing either the incoming or outgoing cross-application call, and then select the Business Transactions tab.

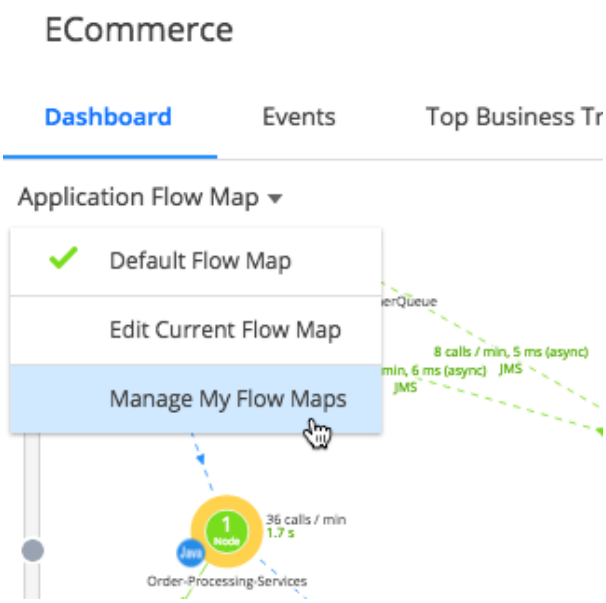
Manage Flow Maps

In a large scale deployment, the flow map may show hundreds of monitored nodes, not all of which may be of interest to specific users or teams. You can create flow maps that are targeted to specific areas of interest by creating custom flow maps.

The custom flow map can be configured to show only certain tiers or those based on performance thresholds, for example:

- Only tiers from where the load exceeds fifty calls per minute and the average response time exceed 10,000 ms
- Only backends receiving at least 400 calls per minute and generating more than 10 errors per minute

To create, copy, or delete a flow map, click the flow map menu and click **Manage My Flow Maps**.



When you create a flow map, the new flow map inherits the context of the flow map in which it was created, whether created from an application, business transaction, tier, or node flow map. To customize your new flow map, see Customize Flow Maps.

Customize Flow Maps

Deployment Support

Related pages:

- Flow Maps
- License Management
- Network Visibility

You can customize flow maps to show information based on performance criteria or to have specific tiers or databases and remote services visible.

You can customize a built-in flow map, but in most cases, you will likely want to customize a flow map you have created. To customize a flow map, choose **Edit Current Flow** from the flow map menu. You can make the flow map shared or private. When shared, your custom flow map appears in the flow map menu for all users.

Add ELBs to the Flow Map

> This document contains links to AWS documentation. AppDynamics makes no representation as to the accuracy of Amazon documentation because Amazon controls its own documentation.

AppDynamics users can now view Amazon-hosted entities and data in several AppDynamics products by connecting their Amazon account to AppDynamics. This topic explains how to connect and view Amazon CloudWatch Elastic Load Balancing (ELB) data on a Network Visibility flow map, and describes what metric data is reported. AppDynamics uses the Amazon CloudWatch API to obtain metrics in near real-time from your Amazon Web Services (AWS) resources and applications. This combination of AppDynamics and AWS cloud-native monitoring helps identify and diagnose a variety of issues from a single Controller.

Requirements

- AppDynamics SaaS Controller >= 4.5.13.
    - Role Delegation in AWS is available from >= 4.5.16.
- A Network Visibility license.
- An AWS Classic Load Balancer with at least one EC2 instance showing a status of InService. See Getting Started with Elastic Load Balancing.
- An AWS Identity and Access Management (IAM) role with permission to create a user and assign a role to that user.

AppDynamics provides two ways to connect to Amazon CloudWatch and start monitoring:

- Connect AppDynamics to Amazon CloudWatch using Role Delegation
- Connect AppDynamics to Amazon CloudWatch using Access Key Credentials

You can connect your AWS account using Role Delegation, granting additional permissions to a new or existing role. Alternatively, you can create Access Key Credentials, a new user with a new pair of access keys.

Connect AppDynamics to Amazon CloudWatch Using Role Delegation

Setting up Role Delegation requires configuring values and settings in your Controller and your AWS account:

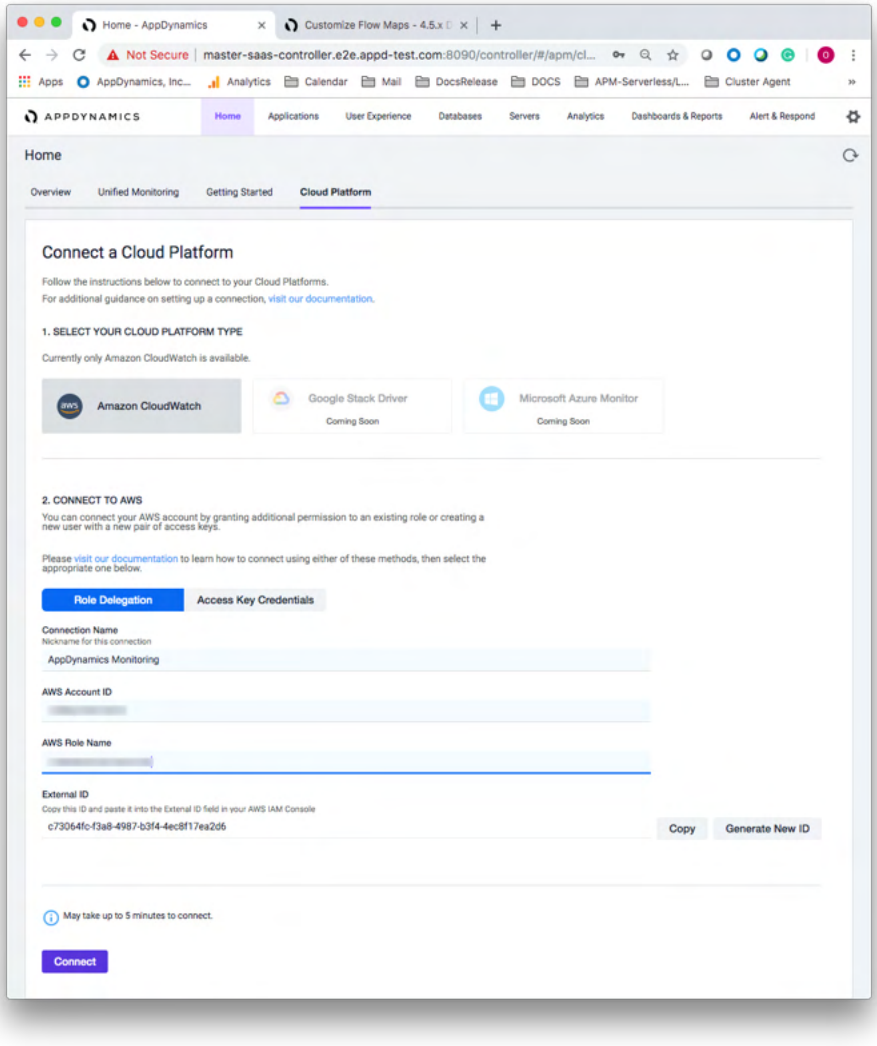
1. Create a Role Name and Copy External ID in the Controller
2. Create an AWS IAM Service Policy
3. Create an AWS IAM Role
4. Connect your AWS IAM Service Role to AppDynamics

Create a Role Name and Copy External ID in the Controller

To connect your AppDynamics Controller:

1. Go to https://accounts.appdynamics.com/subscriptions.

2. Sign in to your account.
3. Select the Controller that you want to connect to Amazon CloudWatch.
4. Navigate to **Home > Cloud Platform** tab.
5. Under **Connect a Cloud Platform**, select **Amazon CloudWatch**.
6. Click **Role Delegation**.



7. Enter your Amazon Credentials:
    a. Enter a Connection Name (in our example, we chose **AppDynamicsMonitoring**).
    b. Enter your organization's AWS Account ID without dashes.
    c. Enter an AWS Role Name, for example, **AppDynamicsMonitoringRole**, the name should not contain spaces or special characters.
    d. Under the **External ID** field, click **Copy**. Save this information somewhere, as you'll need this External ID to enter into your AWS IAM Console.

    > ⚠ If you close this window without capturing the External ID, you will have to generate a new ID for your **AppDynamicsMonitoring** account.

    Leave this browser window open. In a separate browser, log in to your AWS Account.

Create an AWS IAM Service Policy

In the AWS IAM Management Console, there are two ways to attach permissions policies, using JSON or the AWS UI. We recommend using JSON, as it contains the minimum required permissions. If you prefer you can also do this manually using the AWS UI, see Creating IAM Policies - AWS Identity and Access Management.

JSON

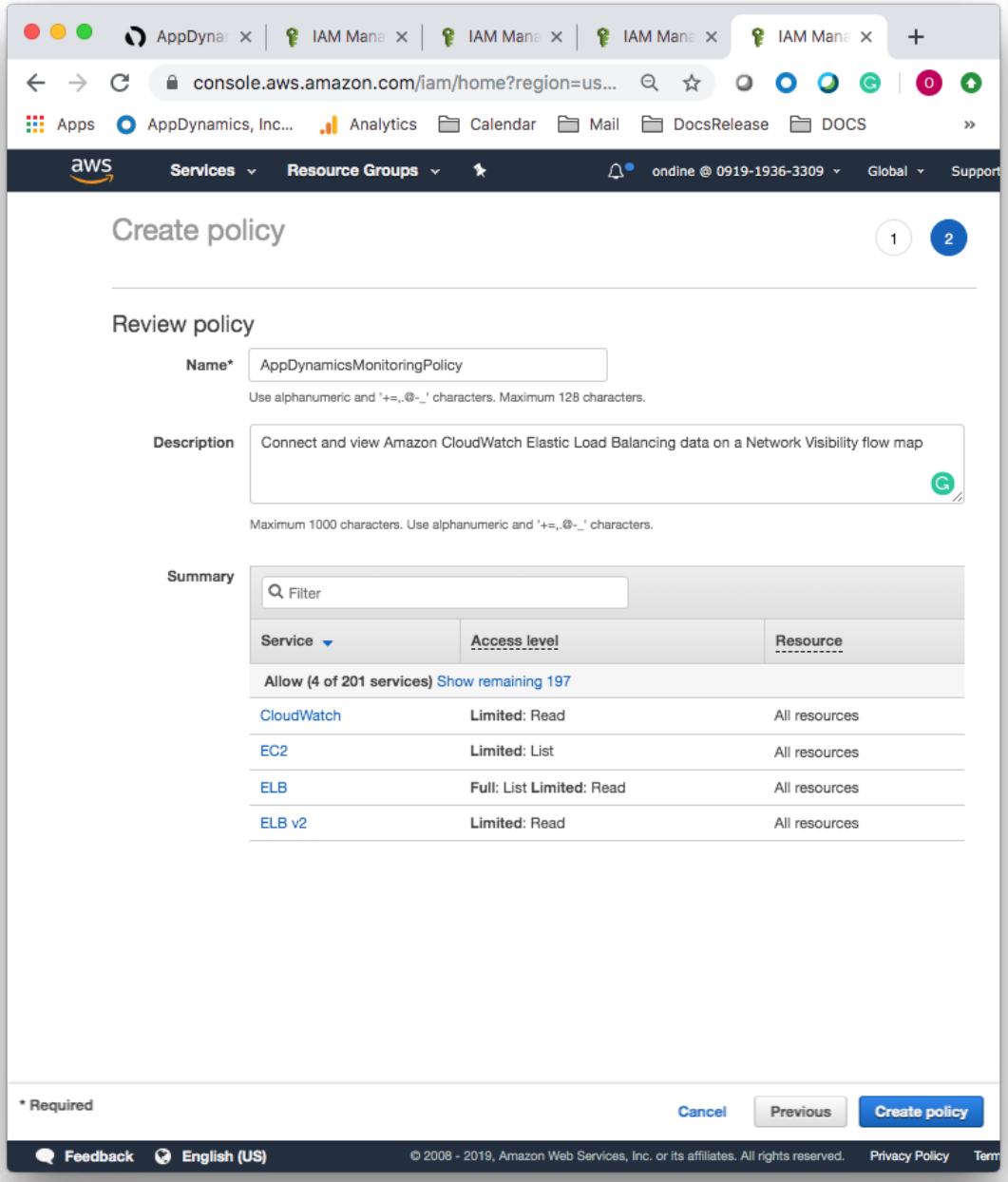
Navigate to the AWS Management Console and open the IAM console.

1. Click **Create policy**.
2. Click the **JSON** tab.

3. Copy the code below.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "elasticloadbalancing:DescribeLoadBalancers",
                "ec2:DescribeInstances",
                "cloudwatch:GetMetricData",
                "ec2:DescribeRegions",
                "elasticloadbalancing:DescribeInstanceHealth"
            ],
            "Resource": "*"
        }
    ]
}
```

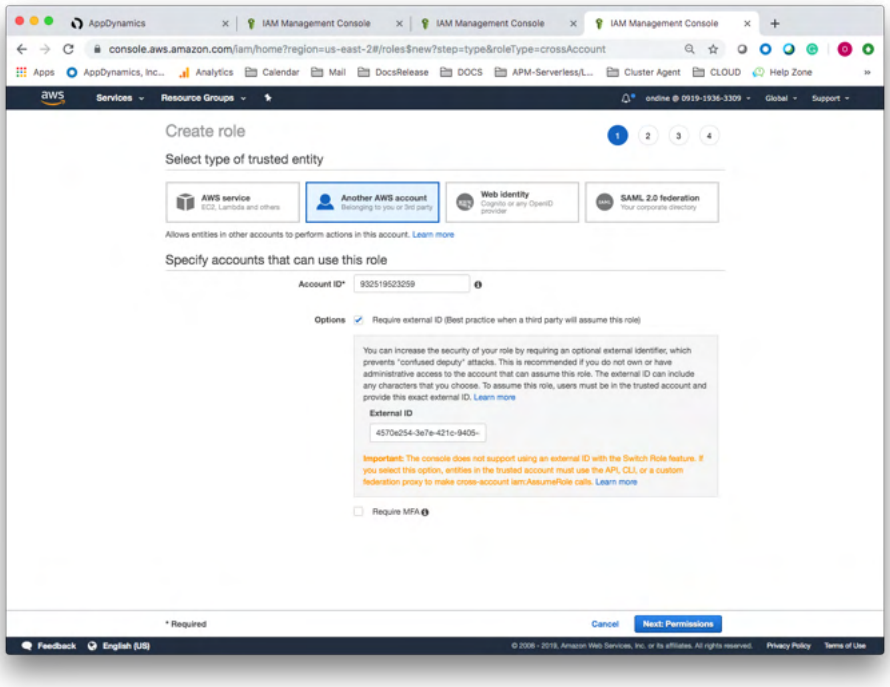
4. Paste the code under the **JSON** tab.
5. Click **Review policy**.
6. Enter a name for the policy, such as **AppDynamicsMonitoringPolicy,** and optionally, add a description**.**



Click **Create policy**. A success message displays: AppDynamicsMonitoringPolicy has been created.

Create an AWS IAM Role

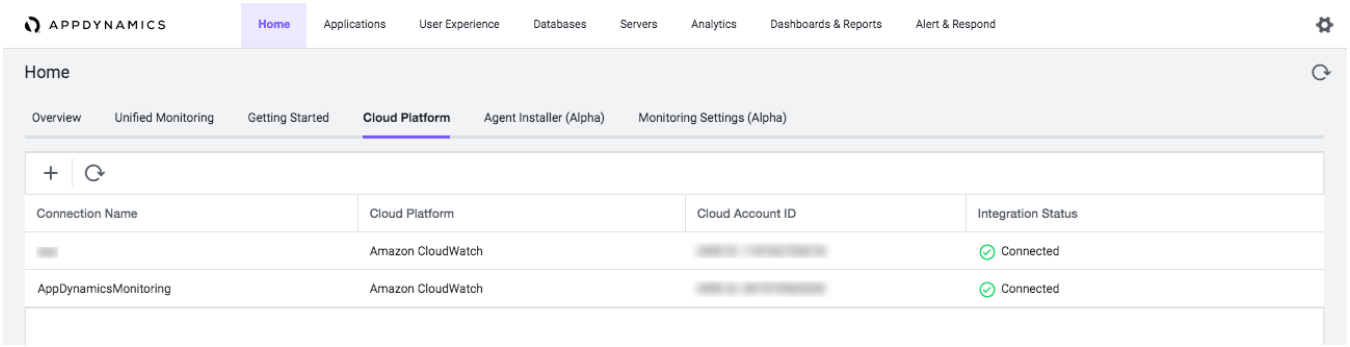
1. In the Identity and Access Management (IAM) console left navigation pane, select **Roles**.
2. Click **Create role**.
3. Select: **Another AWS account**.
4. For **Account ID**, enter AppDynamics Account ID: **932519523259**.
5. For **Options**, check **require external ID**.
    a. Navigate to your AppDynamics Controller browser window. Copy the External ID, if you have not already done so.



6. Return to the AWS Management Console browser window.
7. Paste or enter the **External ID** generated by the AppDynamics Controller. Leave the **Require MFA** option disabled.
8. Click **Next: Permissions**.
9. Under **Attach permissions policies**, search for and select **AppDynamicsMonitoringPolicy**.
10. Click **Next: Tags** (optional).
11. Click **Next: Review.**
12. Enter a name for the role, such as **AppDynamicsMonitoringRole**, and optionally, add a description**.**
13. Click **Create role**. A message displays: The role AppDynamicsMonitoring role has been created.

Connect your AWS IAM Service Role to AppDynamics

1. Return to the Controller browser window. The AppDynamicsMonitoring connection uses the **AppDynamicsMonitoringRole** to connect.
2. Make certain that the AWS Role Name in the Controller matches the role name with permissions policy created in the AWS IAM Management Console. In our example, we used the name: **AppDynamicsMonitoringRole**.
3. Click **Connect**. This initiates the AppDynamics data connection with Amazon CloudWatch.
4. The **Home > Cloud Platform > Integration Status** tab displays the connection status of your Amazon CloudWatch integration. After five minutes, use the refresh icon in your browser to verify the connection has succeeded.



Confirm that the Integration Status of your AWS account shows **Connected** before navigating to the Network Dashboard.

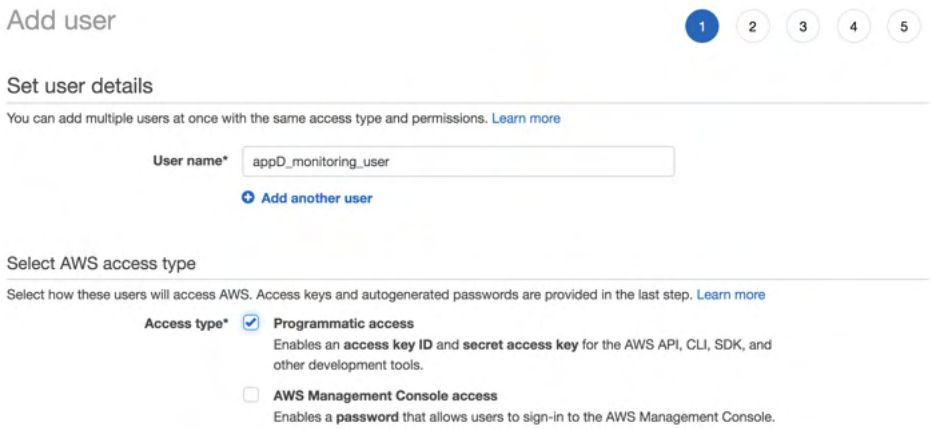
Connect AppDynamics to Amazon CloudWatch Using Access Key Credentials

To integrate AppDynamics with Amazon CloudWatch, create an AWS Identity and Access Management (IAM) user within your AWS account. Once you create the IAM user, attach read-only policies to limit the permissions that this policy grants to the user.

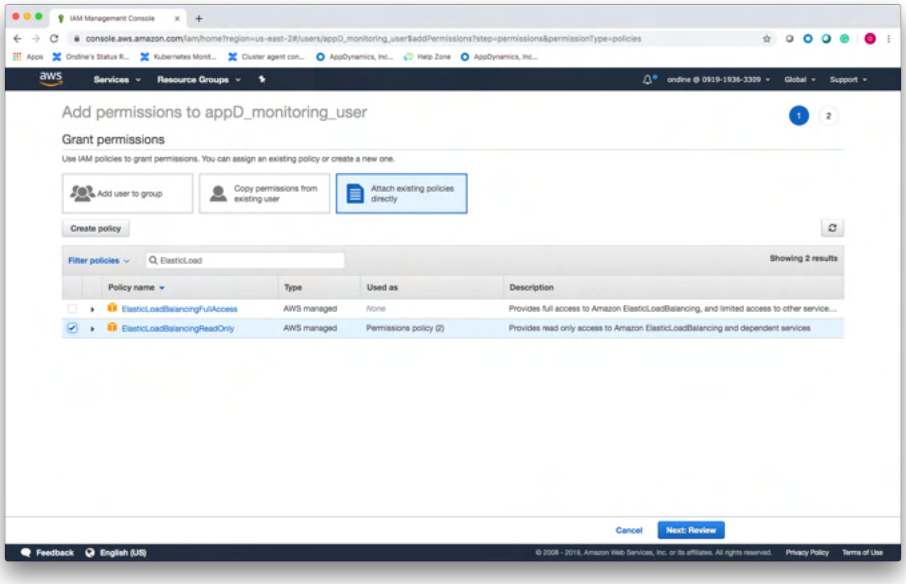
1. Create an AWS IAM User
2. Connect your AWS IAM User to AppDynamics

Create an AWS IAM User

1. Navigate and sign in to the AWS Management Console.
2. Open the IAM console.
3. In the Identity and Access Management (IAM) left navigation pane, select **Users**.
4. Click **Add user**. If you are unable to add a user, see Access Management in the Amazon documentation.
5. Enter the **User name:** `appD_monitoring_user` or a `username` of your choice.
6. Under **Select AWS access type**, check **Programmatic access**.



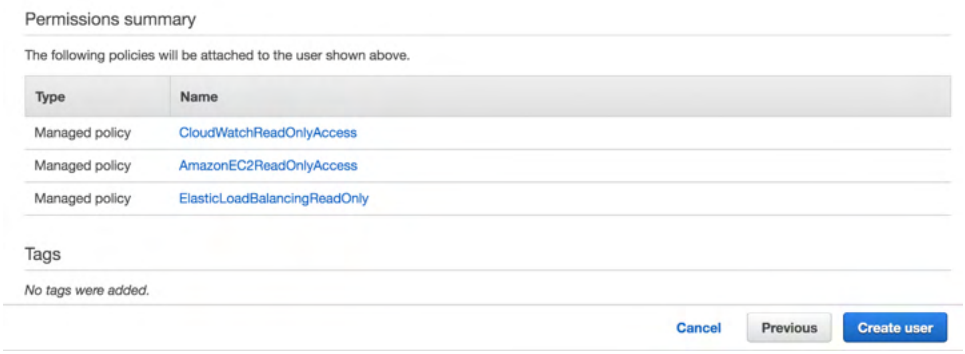
7. Click **Next: Permissions**.
8. Under **Grant permissions**, select **Attach existing policies directly**.



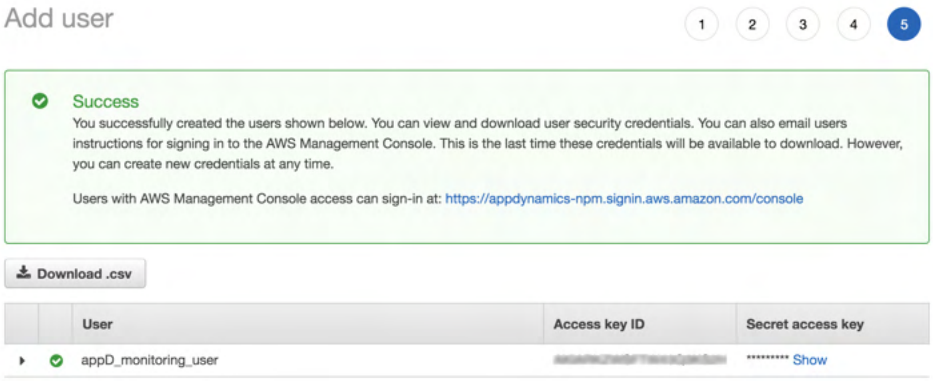
9. Choose appropriate policies for the **`appD_monitoring_user`** monitoring account. We recommend ReadOnlyAccess policies. You can also use custom policies specific to the active resource that you want to ingest Amazon CloudWatch metrics. See Access Management and Example IAM Identity-Based Policies. In our example, we chose these policies:
    - AmazonEC2ReadOnlyAccess
    - CloudWatchReadOnlyAccess
    - ElasticLoadBalancingReadOnly

    Set permissions boundary (optional).
10. Click **Next: Tags** (optional).

11. Click **Next: Review**.



12. Click **Create user**.



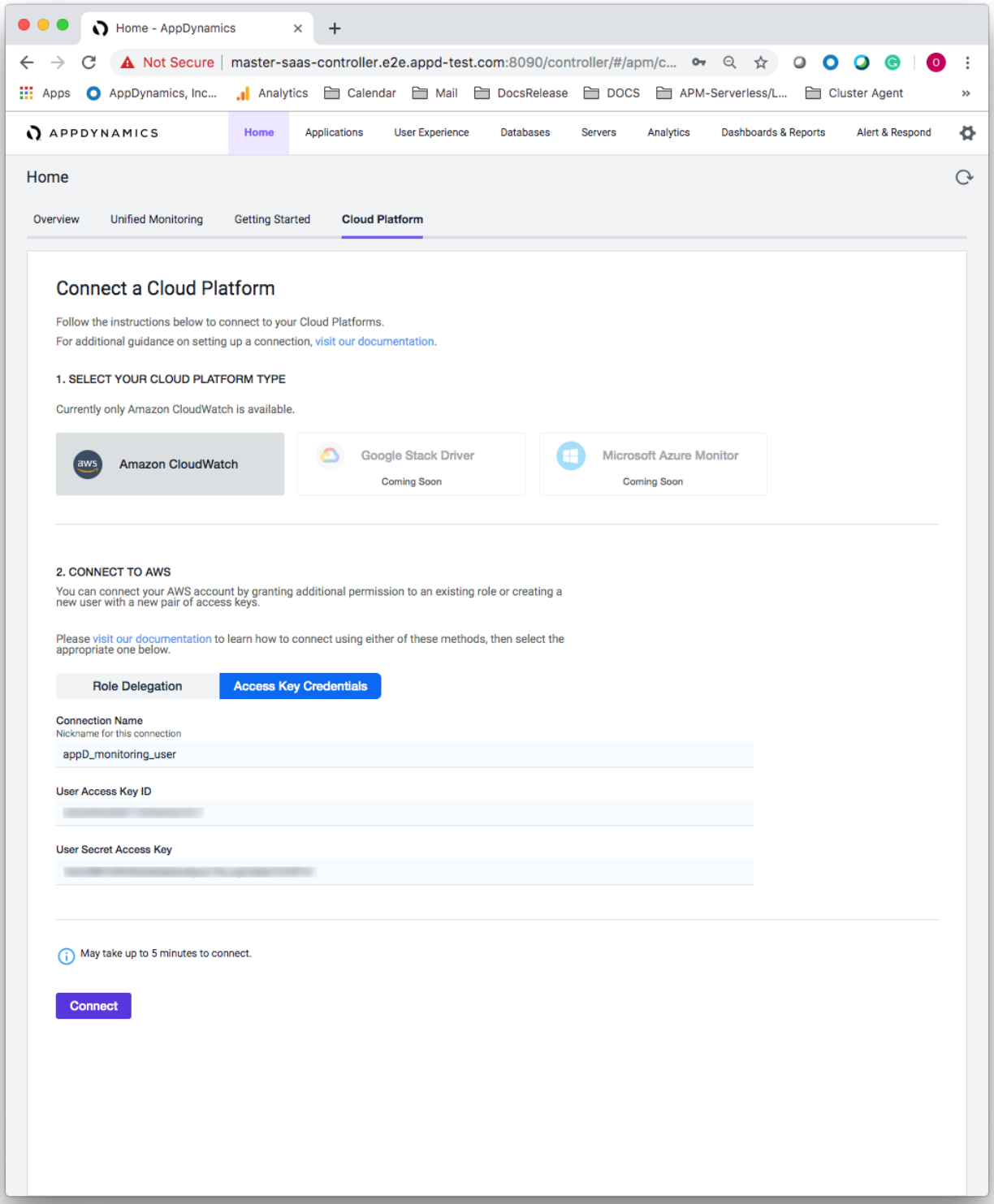
13. Note the **Access Key ID** and the **Secret Access Key** for the user, or click **Download .csv**. You'll need the keys to add Amazon CloudWatch to your AppDynamics account.

If you navigate away from this window without capturing these keys, you won't be able to access them again. You will have to create a new **`appD_monitoring_user`** monitoring account. See Creating IAM Users (Console).

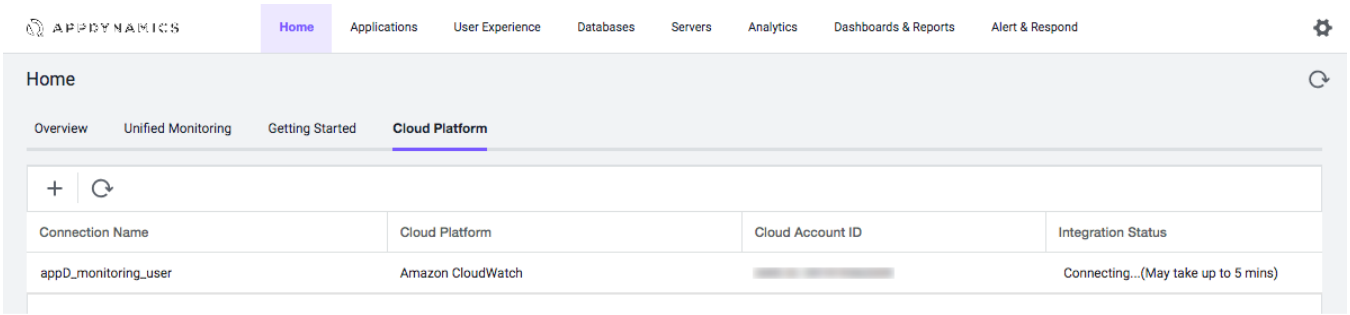
Connect your AWS IAM User to AppDynamics

To connect your AWS IAM user account to AppDynamics:

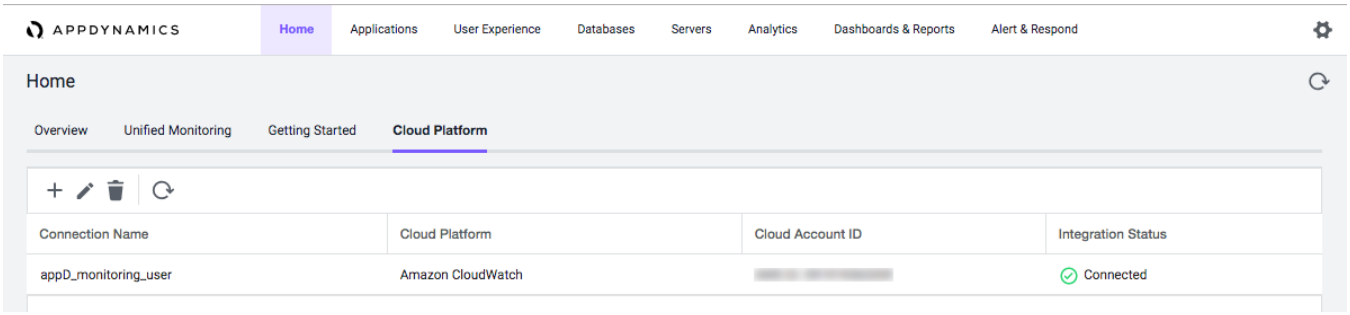
1. Go to https://accounts.appdynamics.com/subscriptions.
2. Sign in to your account.
3. Select the Controller that you want to connect to Amazon CloudWatch.
4. Navigate to **Home > Cloud Platform** tab.
5. Under **Connect a Cloud Platform**, select **Amazon CloudWatch**.
6. Click the **Access Key Credentials** tab.
7. Enter a Connection Name.
8. Enter your Amazon **appD_monitoring_user** Access Key ID.
9. Enter your Amazon **appD_monitoring_user** Secret Access Key.
10. Click **Connect**. This initiates the AppDynamics data connection with Amazon CloudWatch.



The **Home > Cloud Platform > Integration Status** tab displays the connection status of your Amazon CloudWatch integration.



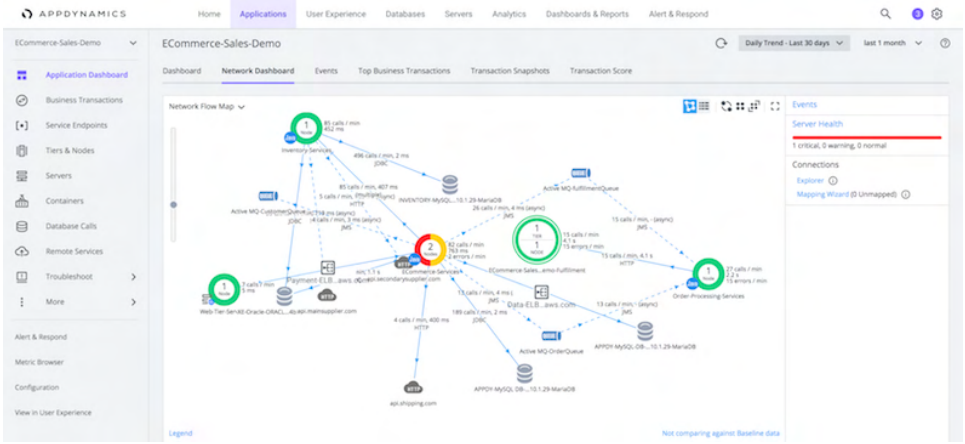
After five minutes, use the refresh icon in your browser to verify the connection has succeeded. Confirm that the Integration Status of your AWS account shows **Connected** before navigating to the Network Dashboard. See View and Monitor Elastic Load Balancing Data.



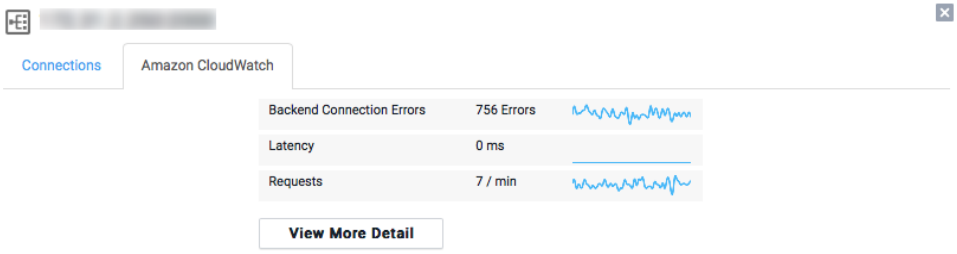
View and Monitor Elastic Load Balancing Data

To view and analyze Amazon CloudWatch metrics in the Controller UI:

1. Go to https://accounts.appdynamics.com/subscriptions.
2. Sign in to your account.
3. Select the Controller that you want to connect to Amazon CloudWatch.
4. Go to **Applications** and select the desired application.
5. Click **Network Dashboard.** A flow map displays.

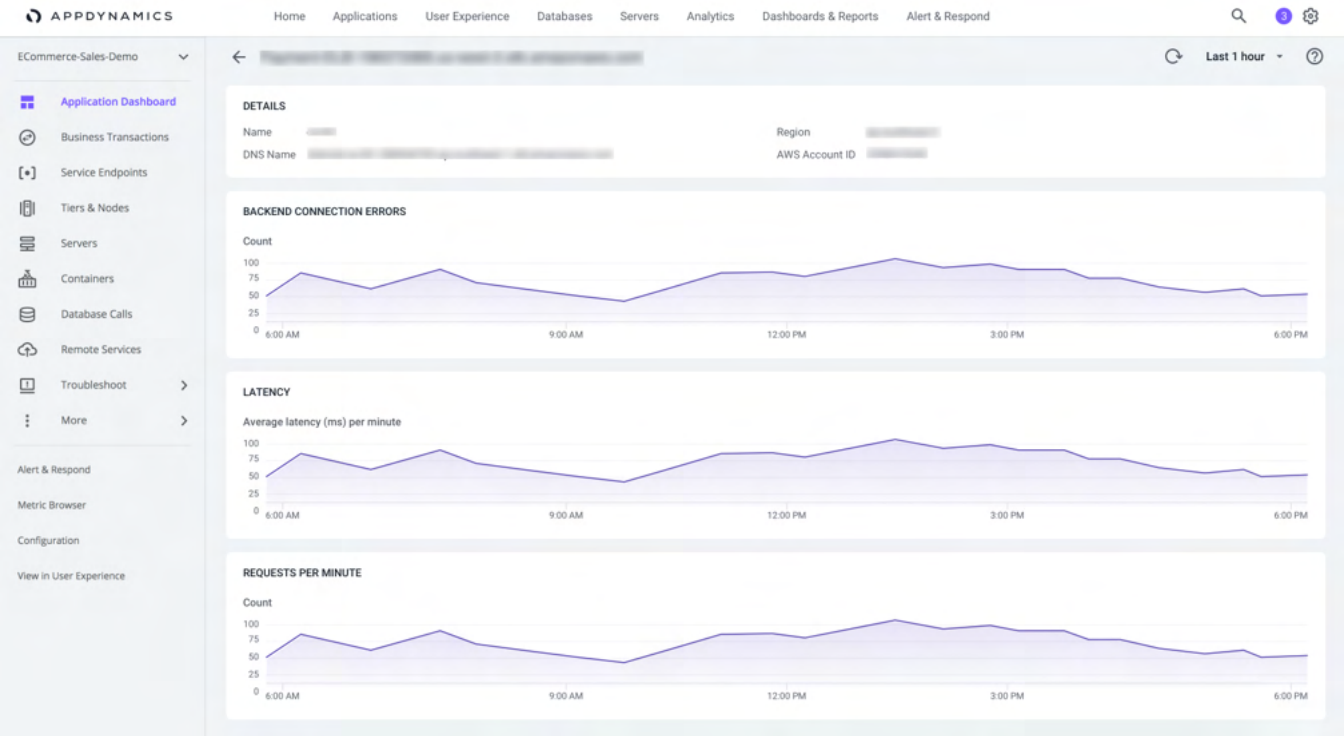


6. Click on an Amazon Elastic Load Balancer (ELB) icon. A dialog appears displaying **Connections** and **Amazon CloudWatch** tabs.



7. Click the Amazon CloudWatch tab. The following metrics appear:
   - **Backend Connection Errors**: Count of backend connection errors per minute.
   - **Latency**: Average request latency per minute.
   - **Requests**: Average requests per minute.

8. Click **See More Detail**.
9. Amazon CloudWatch ELB metrics appear in a time-series chart on the **Application Dashboard**.
10. To modify the time interval, in the top right corner of the page, click **Last 1 hour** drop-down list, and choose your desired time range.



Edit an Elastic Load Balancing Connection

As an account administrator, to edit an existing connection:

1. Go to https://accounts.appdynamics.com/subscriptions.
2. Sign in to your account.
3. Select the Controller that you want to connect to Amazon CloudWatch.
4. Navigate to the **Home > Cloud Platform** tab. A list of connections display.
5. Select the connection name that you want to edit.
6. Click the Edit (pencil) icon.
7. Click **Connect**.

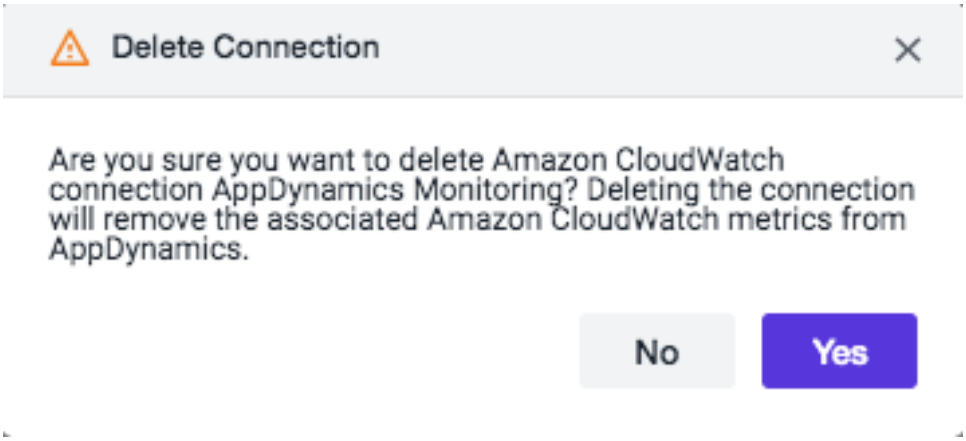
It may take several minutes for your edits to be synced. Refresh your browser after five minutes to verify the connection. Confirm that the Integration Status of your Amazon account shows success before navigating to the Network Dashboard.

Delete an Elastic Load Balancing Connection

As an account administrator, to delete an existing connection:

1. Go to https://accounts.appdynamics.com/subscriptions.
2. Sign in to your account.
3. Select the Controller that you want to connect to Amazon CloudWatch.
4. Navigate to the **Home > Cloud Platform** tab. A list of connections display.
5. Select the connection name that you want to delete.

6. Click the Delete icon . A dialog asks you to confirm that you understand the consequences of deleting the connection.



7. Click **Yes**. If the connection has any metrics, deleting the connection also deletes these metrics.

Troubleshoot AWS Integration Issues

Connection Status is Pending

Navigate to the **Home > Cloud Platform** list of connections. If a connection in the Amazon CloudWatch **Integration Status** column is stuck showing a "connecting" state, check the following:

- Confirm that you are using classic ELBs in your account.
- Confirm there is at least one EC2 instance with the status of InService connected to at least one classic ELB in that account.

If a connection in the Amazon CloudWatch **Integration Status** column is continuing to display "connecting":

- Manually refresh your browser to verify the connection.
- Wait an additional 5 minutes for the integration to connect.

Cross Application Flow

In the AppDynamics model, business applications usually represent a complete application environment. However, it is possible that different business applications share services or infrastructure components.

The cross application flow map displays dependencies and points of contact between business applications in the Controller. This page describes the cross application flow map.

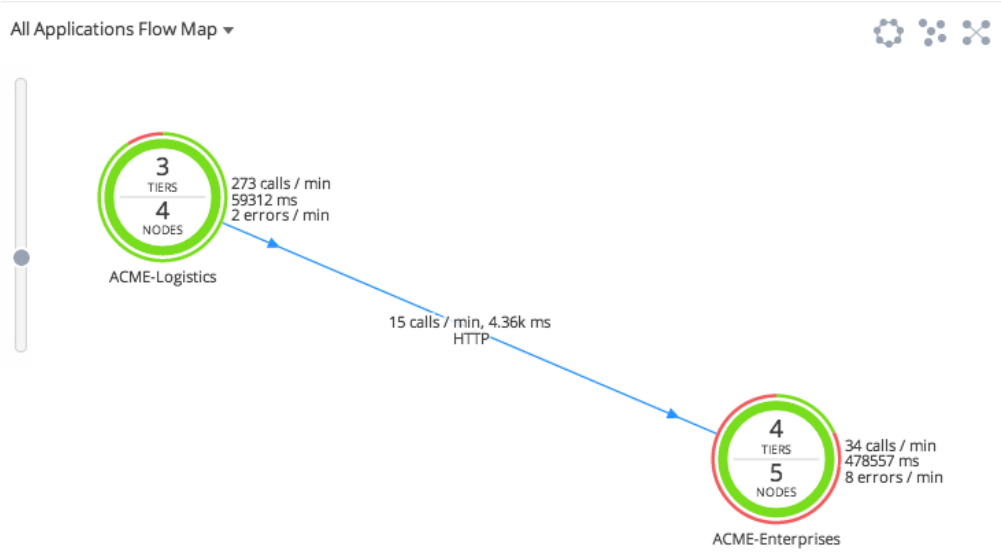
To view the cross application flow map, the user must have the account-level View Business Flow permission.

Access the Cross Application Flow Map

The Applications tab in the AppDynamics Controller UI provides several views of the business applications in your account: a card view, list view, and flow view. To access the cross application flow map, click the flow view link in the tab.

Use the Cross Application Flow Map

Click **View All Applications** on the Controller UI Home page to view the cross application flow.



In the context of a given business application, the call to another application is considered an exit call. If the other application is an AppDynamics-instrumented business application, the call is shown as cross application flow. For an example, see the All Applications Flow Map on Flow Maps.

All users of one application can see correlated applications on the flow map. However, to drill into metrics and snapshots for a correlated application, a user must be a member of a role with view permissions to the correlated application. See Create and Manage Custom Roles.

Hide Components or Business Application on the Flow Map

You can hide backend components, applications, or queues in the right-clicking menu for the component on the cross application flow map.

For applications monitored with the Java Agent, you can exclude a node from being resolved to a cross application flow by setting the following app agent node property to false:

```
support-cross-app-correlation=false
```

This property must be set in the downstream application and prevents calls from external apps registering as cross application flows.

You may choose to do this, for example, if you wanted the cross application call to be represented as a backend call in the calling applications flow map—and metric scheme—rather than as a call to another application. This ensures that metric data reflecting interactions with the tier in the external business application are retained as backend call metrics.

Note

The node property `support-cross-app-correlation=false` works only for the Java Agent.

Export and Import Business Application Settings

This page describes how to export and import the application configuration in the Controller. Application export/import allows you to back up just a single application configuration in the Controller, as opposed to the entire Controller instance, as described in Controller Data Backup and Restore.

For information on exporting other settings from the Controller, see Configuration Import and Export API. For JMX configurations, see Configure JMX Metrics from MBeans.

What Settings Are Exported?

You can export a business application configuration and import it as a new business application. The two applications can be on the same or different Controllers. If on different Controllers they must be using the same major version of AppDynamics.

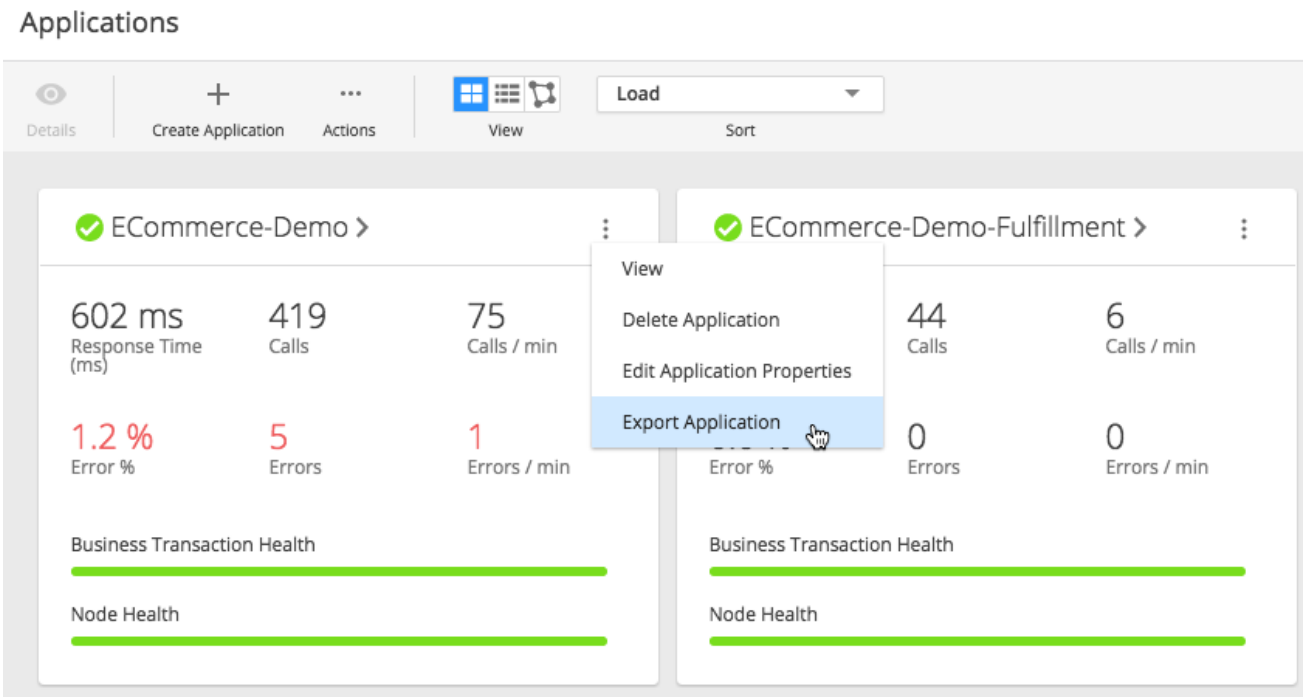
The export is an XML file that includes:

- Snapshot collection settings
- Call graph settings
- Error configuration
- Stall configuration and Business Transaction thresholds
- HTTP and SQL Data gatherer settings
- Tier and Node definitions
- Custom entry point configurations for Business Transactions
- Metric baselines
- Information point configurations

The configuration information does not include data such as Events, Health Rule Violations, and metrics. Nor does it include configuration artifacts not related to the business application, such as users, dashboards, policies, database, and remote services, or schedules.

Export a Business Application

To export the application, from the Applications home page, choose **Export Application** from the action menu for the business application you want to export.



Your browser downloads the XML-formatted configuration file for the application.

Import a Business Application

To import an application configuration, you must be logged in as a user with Administrator user role privileges. You also need access to a previously exported configuration file.

To import the configuration, follow these steps:

1. From the **Applications** page, click **Actions** > **Import**.

2. Select the XML file that contains the configuration you want to import. You can choose to import into an existing business application or create a new one.
3. Choose the configuration settings you want to import. Options include configuration settings for app agents, data collectors, health rules, MDS (scoped instrumentation settings), and more.
4. Click **OK** to start importing. The import process first validates the file. As it proceeds, it shows the status for each type of artifact in the **Import Application** dialog.
5. When done, close the **Import Application** dialog.

Application import does not work over HTTPS if the Controller uses a self-signed SSL certificate. Use a trusted, CA-signed certificate instead. See Controller SSL and Certificates.

Ingest OpenTelemetry Trace Data

Beta Disclaimer

This documentation mentions a product that is currently beta. AppDynamics reserves the right to change the product at any time before making it generally available as well as never making it generally available. Any buying decisions should be made based on features and products that are currently generally available. Please speak to your account representative if you are interested in participating in the beta, or in being notified when the feature becomes generally available.

OpenTelemetry is a single, standardized vendor-agnostic solution used to generate, collect, and export telemetry data. Telemetry data consists of traces, metrics, and logs. OpenTelemetry uses a set of APIs and SDKs to function as a pipeline to ingest and distribute data.

AppDynamics provides an OpenTelemetry Protocol compliant backend to ingest OpenTelemetry trace data generated from applications monitored using OpenTelemetry components. The ingested data is processed by the AppDynamics backend to provide monitoring and alerting capabilities in your application and cloud environments. This page describes how to configure AppDynamics and OpenTelemetry to send OpenTelemetry trace data to the AppDynamics backend.

Review AppDynamics and OpenTelemetry Requirements

AppDynamics

OpenTelemetry is not available for on-premises Controllers.

- To use AppDynamics OpenTelemetry, you need a SaaS Controller version >= 21.2.0.
- Active AppDynamics Pro edition license.
- API Key to authenticate with AppDynamics backend.

Users must have a role defined with the **View and Configure Licenses** permission for license management activity.

- To view your current AppDynamics build version in the Controller UI, go to > **About AppDynamics**.
- To view your current AppDynamics edition in the Controller UI, go to > **License**.
- To obtain your AppDynamics API Key, work closely with your account representative. To learn more, see `x-api-key` in the Attributes Description section.

OpenTelemetry

- Ensure that you have OpenTelemetry Collector >= 0.14.0 downloaded and installed on your deployment. See the OpenTelemetry Collector Getting Started documentation.
- Verify that the OpenTelemetry Collector can connect and report traces to the AppDynamics backend.

Configure the OpenTelemetry Collector YAML File

To send the distributed traces to the AppDynamics backend, you must configure the OpenTelemetry Collector YAML file.

1. Install the OpenTelemetry Collector.
2. Configure Your Applications to Send Service Resource Attributes.
3. Configure the OpenTelemetry Collector for AppDynamics.
4. Verify Configuration.

Install the OpenTelemetry Collector

Install the OpenTelemetry Collector in your computing environment. The OpenTelemetry Collector acts as the receiver in your pipeline and gathers telemetry data from various applications.

Configure Your Applications to Send Service Resource Attributes

AppDynamics relies on the `service.name` and `service.namespace` trace resource attributes to map the services to AppDynamics tiers and applications, respectively.

- `service.name`: Your AppDynamics tier name. Set the corresponding `service.name` resource attribute for every service being monitored.
- `service.namespace`: Your AppDynamics application name. Set the corresponding `service.namespace` trace resource attribute for every application being monitored.

You can set these `service.name` and `service.namespace` attributes using the `OTEL_RESOURCE_ATTRIBUTES` environment variable, or from inside your application code. Please consult the appropriate OpenTelemetry language-specific documentation for your code.

Configure OpenTelemetry Collector for AppDynamics

By default, the OpenTelemetry Collector enables and accepts OTLP (OpenTelemetry Protocol) trace data and sends it to the AppDynamics backend using the default configuration through the OTLP HTTP Exporter.

To configure your OpenTelemetry collector to report data to your AppDynamics Controller, edit your `otel-config.yml` configuration file using this information:

- Processors: (Resource and Batch): Convert data into chosen formats.
- Receivers: Send data to the OpenTelemetry Collector.
- Exporters: Send the converted data to the desired locations.
- Service: Ensures that all previously noted exporters, receivers, and processors are included in the service pipeline.
- Attributes Description: Sets the trace attributes for every service being monitored.

Processors

Resource

Use the resource processor to add the AppDynamics Controller account, host, and port.

- `appdynamics.controller.account`: Your AppDynamics Controller account name.
- `appdynamics.controller.host`: Your AppDynamics Controller hostname.
- `appdynamics.controller.port`: Your AppDynamics Controller port number.

Resource Processor

```
processors:
  resource:
    attributes:
          - key: appdynamics.controller.account
      action: upsert
      value: "acme"
          - key: appdynamics.controller.host
      action: upsert
      value: "acme.saas.appdynamics.com"
          - key: appdynamics.controller.port
      action: upsert
      value: 443
```

Batch

Use the batch processor to improve performance. The `batch` setting accumulates data received through the OpenTelemetry Observability Collector pipeline and sends it to the backend efficiently. The '`batch`' setting defaults the collection and processing to a timeout of 30 seconds, or when a batch size of 8,192 is reached.

- `timeout:` Time duration after which a batch is sent regardless of size. The default is 30 seconds.
- `send_batch_size`: Number of spans or metrics after which a batch is sent. The default is 8,192.

Batch Processor

```
processors:
  batch:
    timeout: 30s
    send_batch_size: 8192
```

Receivers

Receivers use OTLP to support transaction-oriented application data processing. For more information, see OpenTelemetry's OTLP Receiver documentation.

Specify your receivers endpoints:

```
receivers:
  otlp:
    protocols:
      grpc:
        endpoint: localhost:55680
      http:
        endpoint: localhost:55681
```

Exporters

Use the OTLP HTTP Exporter to send data, including the <x-api-key> in the header, to the AppDynamics endpoint <appd-endpoint>. To learn more, see <appd-endpoint> in the Attributes Description section. To obtain your <x-api-key>, work closely with your AppDynamics account representative.

```
exporters:
  otlphttp:
    endpoint: "<appd-endpoint>"
    headers: {"x-api-key": "<x-api-key>"}
```

Service

Ensure that all previously noted exporters, receivers, and processors are included in the service pipeline:

```
service:
  pipelines:
    traces:
      receivers: [otlp]
      processors: [resource, batch]
      exporters: [otlphttp]
```

Attributes Description

Key	Location	Type	Example	Required	Description
appd-endpoint	otlphttp exporter	string	• Asia Pacific (Sydney): https://syd-sls-agent-api.saas.appdynamics.com/ • EU (Frankfurt): https://fra-sls-agent-api.saas.appdynamics.com/ • US West (Oregon): https://pdx-sls-agent-api.saas.appdynamics.com/	Yes	Your AppDynamics endpoint where the OpenTelemetry Collector sends the ingested traces. AppDynamics endpoints are available in the following regions: Sydney, Frankfurt, and Oregon.
x-api-key	otlphttp exporter	string	<alpha numeric key>	Yes	Your AppDynamics API key must be defined as an HTTP header. [1]
appdynamics.controller.account	resource attribute	string	acme	Yes	AppDynamics Controller account name.
appdynamics.controller.host	resource attribute	string	acme.saas.appdynamics.com	Yes	AppDynamics Controller host used by OpenTelemetry. Do not include http:// or https://.
appdynamics.controller.port	resource attribute	integer	443	Yes	AppDynamics Controller port number.
service.name	resource attribute	string	shoppingcart	Yes	Logical name of the service; equivalent to your AppDynamics tier name. Set the corresponding service.name trace resource attribute for every service being monitored using either the SDK, or by setting the OTEL_RESOURCE_ATTRIBUTES environment variable. [2]

`service.namespace`	resource attribute	string	`Shop`	Yes	A namespace for the `service.name`; equivalent to your AppDynamics application name. Set corresponding `service.namespace` trace resource attribute for every service being monitored using either the SDK, or by setting the OTEL_RESOURCE_ATTRIBUTES environment variable. [3]

[1]: To obtain your unique `x-api-key`, you should work closely with your AppDynamics account team.
[2]: MUST be the same for all instances of horizontally scaled services.
[3]: A non-null, non-empty string value for `service.namespace` is required.

AppDynamics Configuration File Example

This example includes configuration for the processors, receivers, exporters, service, and attributes:

```
processors:
  resource:
    attributes:
    - key: appdynamics.controller.account
      action: upsert
      value: "acme"
    - key: appdynamics.controller.host
      action: upsert
      value: "acme.saas.appdynamics.com"
    - key: appdynamics.controller.port
      action: upsert
      value: 443
  batch:
    timeout: 30s
    send_batch_size: 8192
receivers:
  otlp:
    protocols:
      grpc:
        endpoint: localhost:55680
      http:
        endpoint: localhost:55681
exporters:
  otlphttp:
    endpoint: "https://pdx-sls-agent-api.saas.appdynamics.com"
    headers: {"x-api-key": "****************"}
service:
  pipelines:
    traces:
      receivers: [otlp]
      processors: [resource, batch]
      exporters: [otlphttp]
```

Verify Configuration

To validate that your configuration is working properly, review your OpenTelemetry Collector logs.

Visualize Your OpenTelemetry Data

AppDynamics Open Telemetry ingests and aggregates traces into Business Transaction flow metrics. A trace represents an end-to-end request which can be made up of single or multiple spans. Each span represents a single event within a component in the application triggered in the trace. The Business Transaction represents the entire graph of all traces sharing an origin span, and shows how these related traces flow between services.

Flow maps are a dynamic visual representation of your monitored environment's components and activities. Services monitored using OpenTelemetry are represented by the OpenTelemetry icon in the AppDynamics User Interface.

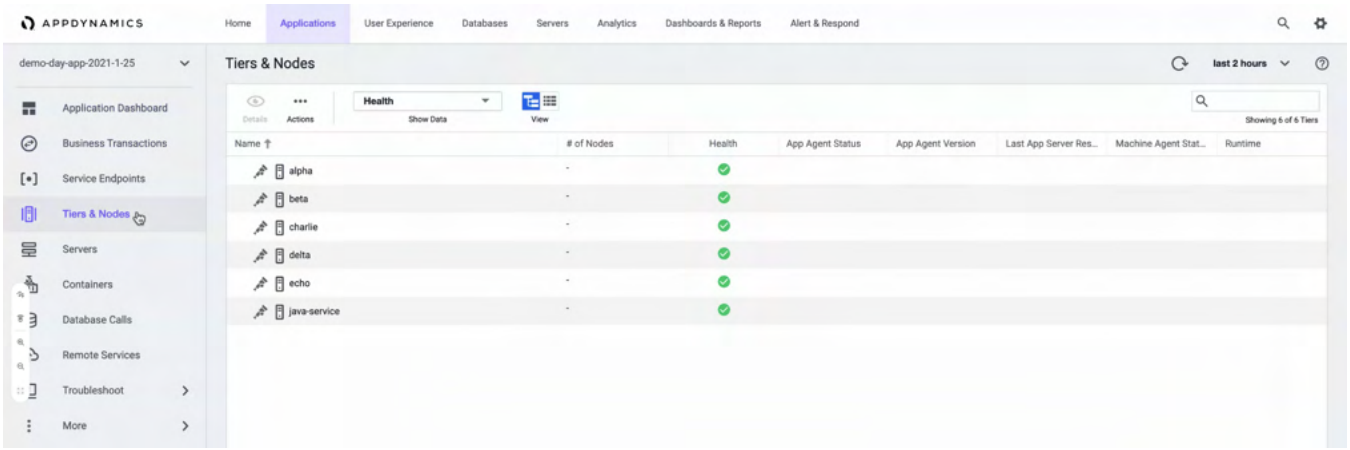
To view your OpenTelemetry data:

1. Select the OpenTelemetry-enabled Controller.
2. Navigate to the application name that you specified with the `service.namespace` resource attribute value.
3. Select **Application Dashboard > Dashboard > Application Flow Map**.

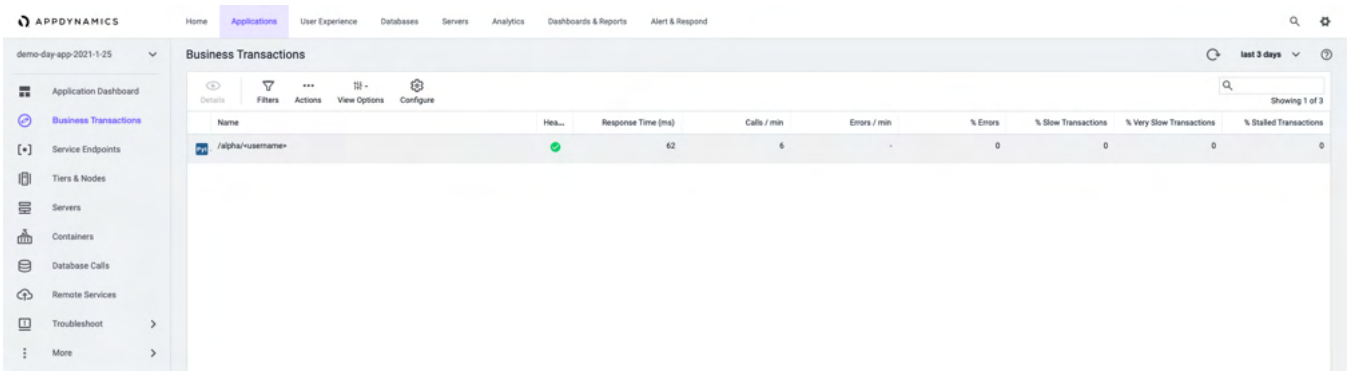
This example shows an application with OpenTelemetry enabled.

This example application consists of six tiers: **alpha**, **beta**, **charlie**, **delta**, **echo**, **java-service**. Next to each tier icon, a telescope icon displays. These telescope icons indicate that OpenTelemetry is successfully enabled for the tiers listed.

Under **Applications > Tiers & Nodes**, the tiers represent the services (defined by `service.name`) encountered by the ingested traces.



The **Applications > Business Transactions** view displays the Business Transactions. The names of the Business Transactions are derived from the name of the first span of each trace (also known as the root span). In this example, the first span is reported from the **alpha** tier and is named **/alpha /<username>**. The health status, response time, number of calls, and additional Business Transaction metrics are reported for the Business Transaction.



The AppDynamics Flow Map supports these programming language icons: C++, C#, Elixir, Erlang, Go, Java, JavaScript, Python, and Rust. C++ and C# programming languages default to a .NET icon in the Flow Map.

Troubleshooting

Flow Map

If you do not see the Flow Map in the Controller, review your configuration procedure. If the issue persists, please work with AppDynamics Support.

Business Transactions

Related pages:

- Organize Business Transactions
- Transaction Detection Rules
- Service Endpoints

In the AppDynamics application model, a business transaction represents the end-to-end, cross-tier processing path used to fulfill a request for a service provided by the application. This topic introduces and describes business transactions.

View Business Transactions

To view business transactions for a business application, click Business Transactions in the application navigation tree. The business transaction list shows key metrics for business transactions for the selected time range.

Only business transactions that have performance data in the selected time range appear in the list by default. You can show inactive business transactions for the time range by modifying the filter view options.

Other ways to modify the default view include showing transactions that belong to business transaction groups or transactions that exceed a configurable average response time. You can also choose which performance metrics appear for business transactions in the list from the View Options menu.

To see the actions you can perform on business transactions, open the More Actions menu. Actions include viewing health rule violations, configuring thresholds, renaming business transactions, grouping business transactions, starting a diagnostic session for the transaction, and classifying a business transaction as a background task.

Transaction Entry Points and Exit Points

When you install an app agent, the agent detects incoming calls and registers transactions based on the default transaction detection rules. Automatic detection rules describe entry points for transactions based on supported frameworks.

Usually, more than one tier participates in the processing of a transaction. A request to the originating tier may invoke services on:

- Another instrumented tier, called a downstream tier.
- Remote services that are not instrumented.

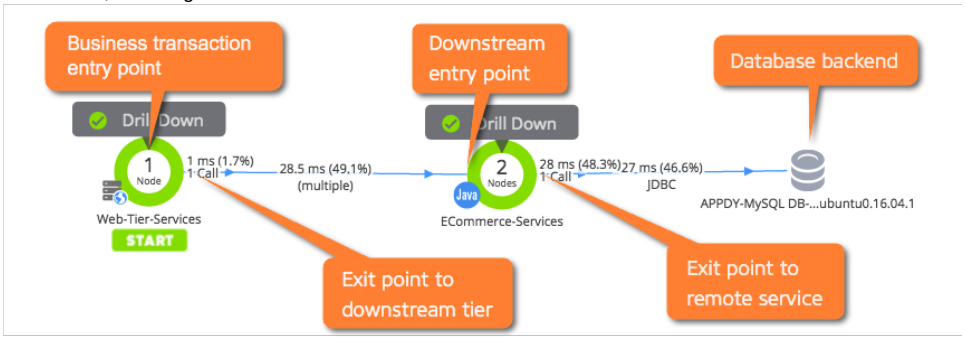
Outbound requests from an instrumented application tier are called exit points. Downstream tiers may, in turn, have exit points that invoke other services or backend requests.

App agents tag exit point calls with metadata describing the existing transaction. When an agent on a downstream tier detects an entry point that includes transaction metadata from another AppDynamics app agent, it treats the entry point as a continuation of the transaction initiated on the upstream tier. This linking of upstream exit points to downstream entry points is called correlation. Correlation maintains the client request context as it is processed by various tiers in your business application.

Sample Business Transaction

Consider, for example, the fictional ACME Online application. The application exposes a checkout service at `http://acmeonline.example.com/checkout`. A user request to the service triggers the following distributed processing flow and actions:

1. The business transaction entry point at the originating tier is `/checkout` URI, which is mapped to a Servlet called CheckoutServlet.
2. The request results in the originating tier invoking the createOrder method on a downstream tier, the ECommerce-Services server.
3. The inventory tier application calls a backend database, which is an exit point for the business transaction. The request context is maintained across tiers, including calls to backend tiers.



4. Any user request on the entry point is similarly categorized as this business transaction, the Checkout business transaction.

To enable detection of all the components in a distributed business transaction, downstream agents must be at the same AppDynamics release or newer than upstream agents. This is true whether the tiers are all built on the same platform—for example all Java—or multiple platforms—a Node.js tier calling a Java tier, calling a .NET tier and so on.

Refine Business Transactions

While the default rules can go a long way towards getting you a useful list of business transactions to track, an important part of implementing AppDynamics is verifying and refining the business transactions used to monitor your application. The business transaction you are monitoring should reflect those operations that are important to your application and business. It is important to consider the limits on business transactions, and apply your refinements accordingly.

Refining your business transaction list requires a solid understanding of the important business processes in your environment. Identify the 5 to 20 most important operations in the application. These are the key operations that must work well for the application to be successful.

Important services can be indicated by the number of calls or calls per minute received by the business transactions generated for the services. You can refine the list of transactions you want to monitor by locking down critical transactions and enabling automatic cleanup of stale transactions. For the Java and .NET environments, you can use interactive Live Preview tools to help identify important transactions.

You can add business transactions manually from a virtual business transaction called *All Other Traffic*, which is populated with transactions once the business transaction registration limits are reached, as described below.

To customize the business transaction list, you can use either of these approaches:

- You can modify existing business transactions by grouping, renaming, or removing the business transactions. Most of these operations are available from the business transaction list. Use this approach to apply relatively minor, small scale changes to the current business transaction list. For more information, see Organize Business Transactions.
- You can affect how business transactions are created by modifying the automatic discovery rules. You can modify rules to similarly achieve business transaction grouping and naming, and to exclude transactions. Discovery rules also enable you to define new entry points for business transactions. Discovery rule modification is a powerful mechanism for changing transaction discovery on a larger scale. For more information, see Transaction Detection Rules.

Business Transaction Limits

When reviewing and refining your business transaction limits, it is important to consider the business transaction limits for the Controller and app server agents. Business transaction limits prevent boundless growth of the business transaction list.

The default limits are:

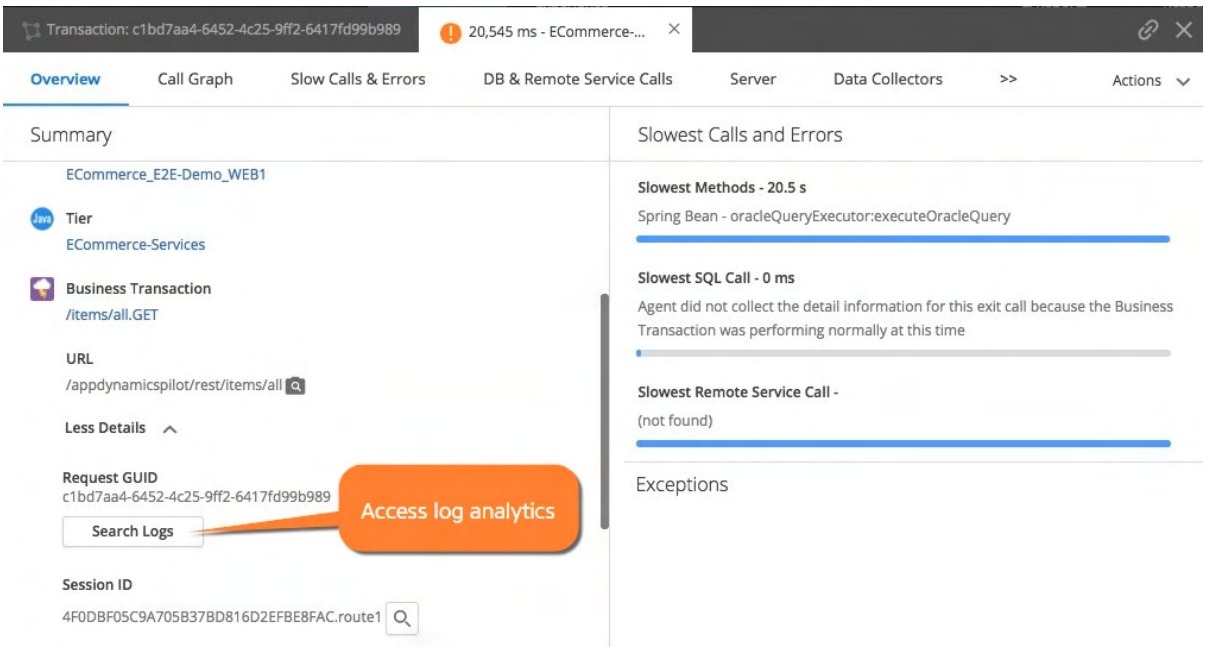
- Business Application Limits: Each application is limited to 200 registered business transactions.
- App Server Agent Limits: Each agent is limited to 50 registered business transactions.

There is no limit at the tier level.

Also note that the app agent limit applies to each app agent, not to the machine. If you have multiple app agents on a single machine, the machine the business transactions originating from the machine could be up to the number of agents times 50.

Correlate Business Transaction Logs

For those times when tracing application code doesn't provide enough clues to track down the cause of a problem, AppDynamics provides visibility into the transaction logs that can be correlated to specific business transaction requests. Log correlation visibility requires a license for both Transaction Analytics and Log Analytics. See Business Transaction and Log Correlation.



Business Transaction Health Rules

You can create health rules to monitor the parameters that represent the normal or expected operations for your business transactions (BT). The parameters rely on metric values, for example, the average response time.

When the performance of a business transaction violates the conditions set by a health rule, a health rule is violated. Alerts notify you of any violation and initiate remediation actions to mitigate the violation event. It is important that you configure alerts appropriately to ensure that you do not miss any alerts or receive false alerts. Alert Sensitivity Tuning (AST) helps you configure alerts with appropriate sensitivity. AST provides historical data for the metric or the baseline being configured and helps you visualize the impact of the alerting configuration. See Alert Sensitivity Tuning.

Create a BT Health Rule and Fine-Tune Metric Evaluation

You can create a health rule to monitor BT parameters and fine-tune the sensitivity of a BT health rule using AST. See Create a Health Rule and Fine-tune Metric Evaluation for instructions.

Organize Business Transactions

Organize and manage registered business transactions on the business transaction list to ensure that you are monitoring the most critical transactions. You can rename transactions, delete transactions, exclude transactions, or create custom transaction detection rules.

Especially for busy applications with lots of transactions, identify the best strategy for maximizing visibility into business-critical operations. Consider these strategies:

- Set transaction detection priorities and delete old transactions to ensure that AppDynamics instruments your most important transactions.
- Combine multiple transactions into rules and exclude unimportant transactions to ensure that you stay within the business transaction limits.
- Rename transactions to informative names and group together similar transactions so that your metrics are easier to read.

Rename Business Transactions

AppDynamics names the business transactions it discovers based on the default naming scheme for the application type of the entry point for the transaction. See Business Transactions.

You can change the name for the business transaction by selecting the business transaction and choosing Rename from the More Actions menu. This change only affects the label for the business transaction in the user interface. You can also modify the default naming scheme for subsequently discovered business transactions, as described in the discussion on naming in Transaction Detection Rules.

> By default, AppDynamics uniquely names business transactions according to the tier name and entry point type, along with an internal name. When you rename a business transaction, ensure its new name is unique; otherwise, there will be multiple business transactions reporting metrics under the same name.

Group Business Transactions

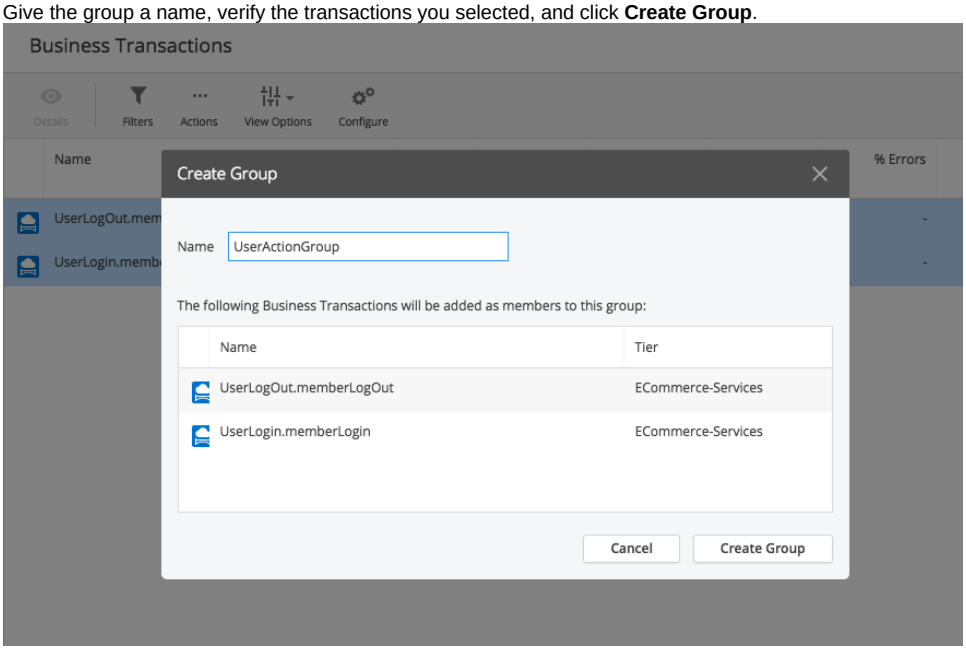
A transaction group lets you collect multiple related transactions into a single organizational unit in the AppDynamics model of your environment. For example, you may wish to group business transactions all of which have entry points in a specific WAR file. Groups could also be used to collect transactions of a specific organization, region, or category.

After grouping transactions, you can see aggregate metrics for the group. Metrics for individual transactions that make up the group remain available.

Grouping transactions does not affect the overall count of registered business transactions. To manage your transaction count, see Custom Match Rules.

To create a business transaction group:

1. Select the transactions you want to group from the Business Transactions list. Use the control or shift keys to multi-select business transactions.
2. Right-click the selected transactions and choose **Create Group**.
3. Give the group a name, verify the transactions you selected, and click **Create Group**.



Delete or Exclude Business Transactions

Excluding and deleting business transactions reduce the business transaction registration count. You can choose to exclude a short-lived business transaction temporarily or delete a stale business transaction permanently.

Exclude a Business Transaction

In effect, excluding the transaction disables the business transaction for metric processing purposes; it works even if the processing path that the business transaction represents remains active in the monitored application. Both serve to reduce the business transaction count considered against transaction registration limits.

> Excluding a business transaction in the business transaction list is not to be confused with using custom exclude rules to control business transaction detection. While the two approaches achieve similar goals, custom exclude rules operate at the transactions detection point and are most useful for preventing discovery and registration of a range or multiple ranges of business transactions at a time. See Custom Match Rules f or more information.

To exclude a business transaction:

1. Select your transaction from the list of business transactions.
2. Right-click the selection and click **Exclude Transactions**.
3. On the Exclude Business Transactions page, select the transaction and click **Exclude Transactions**.

Restore an Excluded Business Transaction

When you exclude a business transaction, the accumulated metrics for the transaction along with its underlying configuration is retained. You can restore the excluded transaction, if required.

1. From the Business Transactions page, click **Actions > View Excluded Transactions**.
2. From the Excluded Business Transactions page, select your transaction, and click **Unexclude Selected**.

Delete a Business Transaction

When you delete a business transaction from the list, the accumulated transaction metrics are removed. However, the deleted transaction is rediscovered if the corresponding function in the application is used. To delete a business transaction permanently, modify the discovery rules after deleting a transaction.

A business transaction can be considered stale in these scenarios:

- You have decommissioned an application, and you want to delete the historical data associated with business transactions that made up the application.
- You have changed business transaction discovery rules to either eliminate or reorganize business transactions and need to remove unwanted or otherwise obsolete business transactions.

To delete a business transaction:

1. Select your transaction from the list of business transactions.
2. Right-click the selection and click **Delete Transactions**.
3. From the Delete Business Transactions page, select the transaction and click **Delete Business Transactions**.

Automatic Cleanup of Stale Business Transactions

Automatic cleanup keeps less active transactions from cluttering your business transaction list. Enabling automatic cleanup deletes business transactions t hat are flagged as stale. You can set the activity threshold for a business transaction after which the transaction is considered stale.

To configure automatic cleanup threshold:

1. From the left pane, click **Configuration > Instrumentation > Transaction Detection > More**.
2. Under Business Transaction Automatic Cleanup, check **Enable Business Transaction Automatic Cleanup**.
3. Update the automatic cleanup threshold that qualifies a business transaction as stale:
    a. Specify your threshold time in, **Monitor Business transactions for __ minutes since creation**.
    b. Specify the number of calls in, **Remove Business transactions that have less than or equal to __ calls**.

Since creation, a business transaction is monitored for 15 minutes when automatic cleanup is enabled with default settings. If there are no calls reported within the first 15 minutes since the business transaction is registered, the business transaction is qualified as short-lived and the automatic cleanup deletes that business transaction. Automatic cleanup keeps a business transaction intact if it is reporting data for days.

However, automatic cleanup ignores the business transactions that are:

- Marked as *Permanent*
- Older than three days

Business Transaction Purger

There are Controller configurations for the business transaction purger:

- `max.bt.to.purge.per.execution`: the number of business transactions that can be purged when the purger executes once. The default is 1000, min is 0, and the max is 100000.
- `max.bt.to.purge.per.transaction`: in order to reduce db lock, the purger tries to purge business transactions in multiple transactions if there are too many business transactions to purge in each execution. This configuration specifies the number of business transactions that can be purged in each transaction. The default is 10, min is 0, and the max is 1000.

Manage Business Transaction Detection

If your system has hundreds of business transactions, you can leverage the rule system to fine-tune which transactions you want to detect and monitor. Creating rules gives you the exact transaction visibility that you want. For example, you can create rules to:

- Detect transactions that are not automatically detected.
- Prioritize monitoring some transactions over others.
- Stay under the transaction limit. When you combine multiple transactions in a rule, the individual transactions no longer count towards the transaction limit; the rule is counted as a single transaction.
- Exclude transactions that you are not interested in monitoring.

Suppose John is creating transaction detection rules for an ECommerce application. John discovers that the most critical transaction, `user-checkout`, is grouped into *All Other Traffic*, which obscures its visibility. To ensure that AppDynamics collects detailed metrics for `user-checkout`, John creates a rule that detects `user-checkout`, and sets the rule priority to a high value, 50. For less critical transactions, John creates a rule that detects multiple transactions: `createaccount`, `billing`, and `delivery-options`. He sets the rule priority to a lower value, 15. Combining multiple transactions into one rule helps John stay within the transaction limit. See Transaction Detection Rules.

All Other Traffic Business Transactions

As an organizational unit in the AppDynamics model, business transactions are meant to be used for the most important transactions in your environment. The Controller limits the number of business transactions (50 for a node and 200 for a business application). This limit helps control resource utilization on the Controller, and also helps you focus on the most important services performed by your application environment.

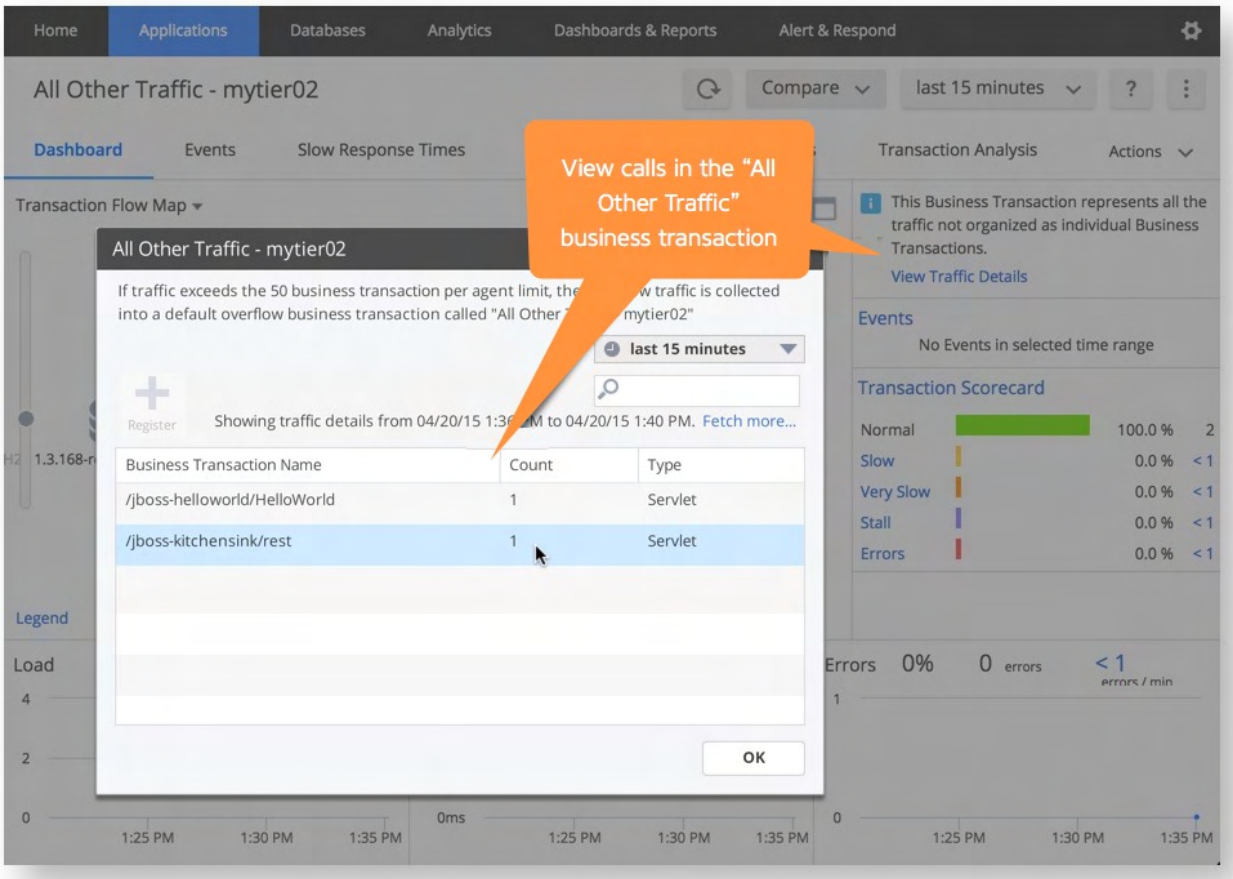
Once the business transaction registration limit is reached or transaction lockdown is enabled, newly detected transactions are not registered as business transactions, but they are monitored and reported by AppDynamics. The transactions are grouped to a virtual business transaction named `All Other Traffic - tier_name`. An *All Other Traffic* group exists for each tier on which the limit is reached.

All Other Traffic appears alongside other business transactions in the Business Transactions list. Like you can for any business transaction, you can view the dashboard and key performance metrics for All Other Traffic in the Metric Browser.

When refining business transactions, it's likely that you'll want to move certain transactions from the All Other Traffic bucket to make them first-class business transactions. You can keep within the limits by deleting or excluding other business transactions, such as those that have little or no load, by using a custom exclude rule. If the number of business transactions is under the limit and lockdown is enabled, you can promote a transaction from the list of All Other Traffic by registering it from the Traffic Details window.

To view incoming calls categorized in the All Other Traffic transaction:

1. Open the dashboard for the All Other Traffic business transaction.
2. Click **View Traffic Details**.



The **Traffic Details** panel lists transaction entry points that were hit by incoming requests after the registration limits were exceeded or after business transaction detection was locked down. The **Business Transaction Name** column contains auto-generated names for the transactions. The **Call** column shows the number of instances of the transaction and the **Type** column shows the entry point type.

If the *Fetch more* link appears, click to view more calls. You retrieve up to 600 calls each time you click the link. If the *Fetch more* link does not appear, there are no more calls to retrieve for the selected time range.

> The name used for All Other Traffic in the REST API is `APPDYNAMICS_DEFAULT_TX`. For an example, see Use the AppDynamics REST API.

Lock Down Business Transactions

A production implementation of AppDynamics usually involves an initial period of assessing, organizing and refining business transactions by customizing your default business transaction discovery rules.

Once you arrive at the set business transactions to monitor, you may want to *lock down* business transactions. Locking down business transactions prevents application changes, upgrades to the agent software, or other changes from affecting the number or selection of business transactions you monitor as the primary, first-class business transactions in the AppDynamics model of your environment.

With business transaction lockdown enabled, the Controller puts newly discovered transactions into the *All Other Traffic* transaction collection for the tier. You can promote a business transaction from the *All Other Transactions* list by registering it as a first-class business transaction manually.

Business transaction lockdown gives you a way to register business transactions individually, by manual selection. In certain scenarios, it may make sense to enable business transaction lockdown before discovery occurs and registering transactions manually. For example, this may be useful if your environment would otherwise generate a large number of business transactions and you only want to monitor a relative few as first class business transactions.

To enable business transaction lock down, select the **Enable Business Transaction Lock down** option in the application instrumentation settings page.

After locking down business transactions, you can promote a transaction manually by accessing the Traffic Details dialog from the All Other Transactions d ashboard. In the dialog, select the transaction and click **Register** to promote the transaction. If the agent is at the business transaction registration limit, you will need to delete a registered business transaction before registering one from the *All Other Traffic* list.

If you enable automatic cleanup, business transactions that you lock down but that do not meet the activity threshold are still deleted. To override automatic cleanup for a business transaction, right-click the transaction and select **Mark As Permanent**.

Business Transaction Performance

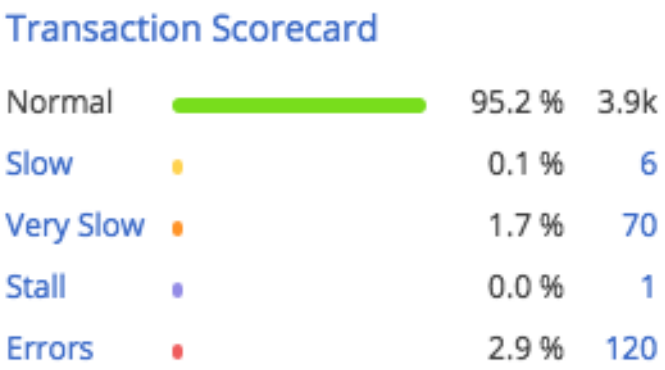
This page describes different options for viewing business transaction performance.

After verifying and refining your business transaction scheme, you can focus on monitoring business transaction performance. The monitoring and troubleshooting tools within the AppDynamics Controller UI provide a business transaction-oriented view of performance. For example, transaction snapshots tie together all the code call graphs across the monitored tiers that participated in processing a particular instance of a business transaction.

The business transaction list contains a high-level look at business application performance by transaction. You can click a particular business transaction to access additional information on the business transaction, such as information on slow performing instances of this transaction, errors, and the transaction analysis view.

Performance at a Glance

The transaction scorecard summarizes the performance of a business transaction at the application, tier, or node level within a specified time range. It covers the number and percentage of business transaction instances (calls) that are normal, slow, very slow, stalled, or errors based on the configured thresholds for these criteria.



The call counts in the scorecards in different dashboards have different meanings:

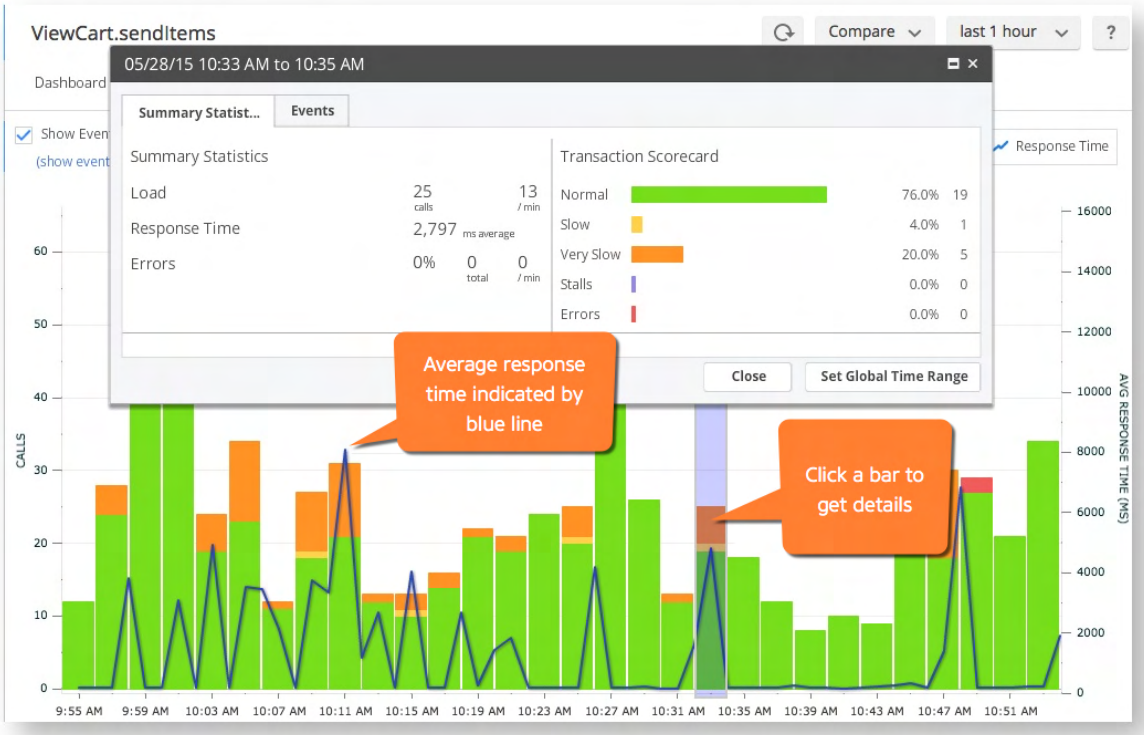
- In the application dashboard, the business transaction health scorecard aggregates the metrics for all the completed business transactions in the application, including the default business transactions. The number of calls is based on the number of business transactions that are completed. The percentages for those calls are based on the average for all the completed business transactions in the application. For example, if the scorecard displays 3% for very slow transactions, on average 3% of the calls for completed business transactions in the application were slow.
- In a tier dashboard, the transaction scorecard aggregates the metrics for all the nodes in the tier. The number of calls is based on the number of requests that the tier handled, including continuing transactions, threads spawned, and so forth.
- A node dashboard is similar to the tier dashboard in that it is based on the number of requests the node handled. The transaction scorecard displays the metrics for the single node.
- In a business transaction dashboard, the transaction scorecard displays the metrics for the entry point of a business transaction.

Transaction Score View

For a more detailed view of the performance of a transaction over time, navigate to the business transaction dashboard and click the Transaction Score tab. The graph in the tab shows the user experience for the selected business transaction, showing its performance relative to the performance thresholds as a bar chart.

For each time slot, the bar reflects the total calls, while the color segments—green, yellow, and so on—indicate the relative number of those calls that were normal, slow, very slow, stall, or errors.

The solid blue line in the graph indicates the average response time for the transaction over time, while the dotted line shows the baseline, if available.


ViewCart.sendItems
Compare
last 1 hour
Dashboard
05/28/15 10:33 AM to 10:35 AM
Summary Statist...
Events
Summary Statistics
Load
25 calls
13 / min
Response Time
2,797 ms average
Errors
0%
0 total
0 / min
Transaction Scorecard
Normal 76.0% 19
Slow 4.0% 1
Very Slow 20.0% 5
Stalls 0.0% 0
Errors 0.0% 0
Close
Set Global Time Range
Response Time
Average response time indicated by blue line
Click a bar to get details
CALLS
AVG RESPONSE TIME (MS)
9:55 AM
9:59 AM
10:03 AM
10:07 AM
10:11 AM
10:15 AM
10:19 AM
10:23 AM
10:27 AM
10:31 AM
10:35 AM
10:39 AM
10:43 AM
10:47 AM
10:51 AM

Automated Transaction Diagnostics

Deployment Support

This capability is available in the SaaS >= 20.10.1.

AppDynamics continuously collects Transaction Snapshots that capture all events in a transaction path for use in performance troubleshooting and root cause analysis. Although there are Rules and Limits in place to prevent excessive resource consumption, you can still have hundreds of individual snapshots to sort through as you investigate a particular performance issue.

The Automated Transaction Diagnostics (ATD) capability allows you to be proactive rather than reactive when it comes to anomalous transaction behavior. ATD uses machine learning to capitalize on the collected snapshots by automatically recognizing anomalies not yet severe enough to trigger alerts but significant enough to cause future production issues. In seconds you have a prioritized list of causal factor transactions that you can use to manage problematic business transactions before they become visible production slowdowns or outages for your customers.

ATD's guided root-cause analysis clearly exposes the offending anomalies along with the contributing tiers, exit calls, or inter-tier network issues. This allows a broader range of less specialized users to more accurately analyze the behavior and significantly reduce repair time.

Technical Overview

AppDynamics uses a systematic approach to identify problematic transactions, determine the probable root cause, and provide a list of snapshots used in the diagnosis. The diagnostic process:

- Analyzes business transaction response time data to determine if there was an issue and how long it lasted.
- Analyzes metrics data across the business transaction path to identify the suspected cause and type of issue.
- Uses snapshot data to provide detailed root cause analysis that diagnoses the business transaction anomaly.

Automated Transaction Diagnostics Workflow

The Transaction Diagnostics panel becomes visible after you select a business transaction.

The diagnostic process detects performance issues, automatically identifies the contributing transactions, and provides a report that simplifies root cause analysis allowing you to spend your time fixing the problems before they become production issues.

1. From the Home or Applications tab, choose the application you want to analyze.
2. Select **Business Transactions**.
3. Double-click a transaction you want to investigate.
4. From the Dashboard tab, review **Transaction Diagnostics** in the right panel.
    a. The server may take a few seconds to display issues. If the server finds no issues, a message displays that none were identified in that time range. Try selecting a longer time range of up to eight days.
    b. The performance issues listed may not directly correspond to the business transaction flow map being shown if the health rule was violated in the past.
5. Select a highlighted anomaly to open the **Transaction Diagnostics** details page.

The interactive graph consists of two charts. The upper graph displays transaction response times that correspond to the time range you choose in the lower navigation bar. The yellow plot bands highlight periods of abnormally slow response times.

1. Use the slider in the lower navigation chart to select a time range from five minutes to eight days.
2. Click any plot band in the navigation bar to view details for that time range. The chart automatically adapts to display the incidents for your time range choices and includes a baseline for the transaction.
3. Review the **Details of Performance Issue** section to view the **Suspected cause.** If you wish to manually review snapshots from the anomalous period, they are available in the lower **Snapshots of Impacted Transactions** list.
4. Double-click a snapshot in the list to view the transaction details.

Transaction Thresholds

A threshold is a boundary of acceptable or normal business transaction performance. AppDynamics provides default thresholds against which it compares the performance of every business transaction. Each transaction is classified into a user experience: normal, slow, very slow, stall or error transaction.

The user experience reflects the performance of a transaction instance relative to the usual performance of the business transaction. AppDynamics makes some intelligent guesses and calculations based on existing traffic flow to establish the default user experience profiles.

This page provides information about refining the criteria by which the classification is applied, according to your requirements.

Transaction Threshold Values

As soon as a transaction starts, the particular thread representing that transaction is monitored for a stall. Every five seconds, the in-flight transactions are evaluated to determine if they have met or exceeded the stall threshold.

If a transaction execution finishes before the stall threshold value, the execution time is first compared against the very slow threshold, then the slow threshold, and so on, and marked accordingly.

If a transaction hits the stall threshold (takes more than 300 deviations above the average for the last two hours) or the set stall threshold, a stall transaction event is generated. Whether the transaction eventually finishes successfully or times out, it is considered a stall for performance monitoring purposes.

If the standard deviation is large (that is, > 50% of average), the moving average is used in the calculation.

Transactions errors are not determined by thresholds. Instead, transactions are marked as errors when the HTTP response status code is in the 400–505 range or when any exception occurs that is not excluded in the custom error configuration.

Static and Dynamic Thresholds

Thresholds can be based on a static value or dynamic value. A dynamic threshold is based on performance for a most recent period of time, the previous two hours, by default. A dynamic threshold can be specified using either a percentage deviation or a standard deviation measure.

To understand how a transaction is compared to this moving window, consider a two-hour moving average window from 11:00 AM to 1:00 PM. All transactions that come in during 1:00 PM and 1:01 PM are measured against the response time and standard deviation of this window. For the next minute, between 1:01 and 1:02 PM, the window moves to 11:01 AM to 1:01 PM. The moving average itself is calculated based on the exponential moving average formula.

Note that:

- Data for minutes where there is no load is not counted.
- The data is maintained locally on each node for the starting tier for a business transaction. The Controller does not hold any of this data.

Percentage Deviation Threshold

Percentage deviation defines a threshold based on the moving average of the requests over a certain interval. The default time interval is 2 hours. If the average response time for the last two hours is X milliseconds, and if the request takes the percentage deviation over X ms, the transaction violates threshold.

For example, a transaction occurs at 10:00:01 AM. The average of the requests between 8:00:00 and 10:00:00 is 100 ms and the slow threshold is defined as 20% over the threshold. If the transaction takes 120 ms, it will be considered a slow transaction.

Standard Deviation Threshold

Standard deviation defines a threshold based on the moving average over a certain interval. The default time interval is two hours. This means if the average response time for the last two hours is X milliseconds, and if the request takes the standard deviation over X ms, it violates the threshold.

To understand how a standard deviation threshold works, consider a moving average that is 1500 ms with a standard deviation of 100 ms. If you set the threshold to 3, as in the following example, it means that the threshold will be three times the standard deviation. In other words, a transaction that takes more than 1500 + (3*100) or 1800 ms will violate the threshold.

Alternatively, you can set the threshold to a static value or use a percentage of the average.

You can also disable stall detection. If you disable stall detection, a transaction that might qualify as a stall is reported as very slow if it completes. If it does not complete, nothing is reported for the incomplete transaction.

Configure Thresholds

You can configure thresholds in **Configuration > Slow Transaction Thresholds**. You will need Configure Business Transaction application permissions.

Thresholds apply to various levels, by application or business transaction, including background task transactions. You can also configure thresholds for diagnostic sessions.

You can specify thresholds for User Transactions—regular business transactions—or background transactions tab, depending on the type of entity for which you want to configure thresholds.

> ⚠ Thresholds for Browser Real User Monitoring (RUM) are configured separately. For information about these thresholds, see Configure Browser RUM Performance Thresholds.

You can configure thresholds for slow, very slow, and stalled transactions. When a transaction exceeds a threshold, AppDynamics starts capturing snapshots of it. Because snapshots are not normally captured while performance is within normal range, the snapshots may not contain the full call graph for the transaction.

Dynamic Baselines

This page describes how AppDynamics automatically calculates the baseline performance for your applications. Once it establishes these prevailing performance characteristics, it can detect anomalous conditions for your application. AppDynamics uses baselines to:

- Color the Flow Map.
- Show transaction score dashboards.
- Compare metrics in metric graphs and the metric browser.
- Evaluate health in health rules (see Health Rules).

How Baselines Work

AppDynamics calculates baselines by using the underlying hourly data. A baseline can be calculated for every metric. Baselines have two main variables:

- The time over which accumulated data should be considered in calculating the baseline.
- Seasonality of the baseline.

Baselines Help Identify Trends

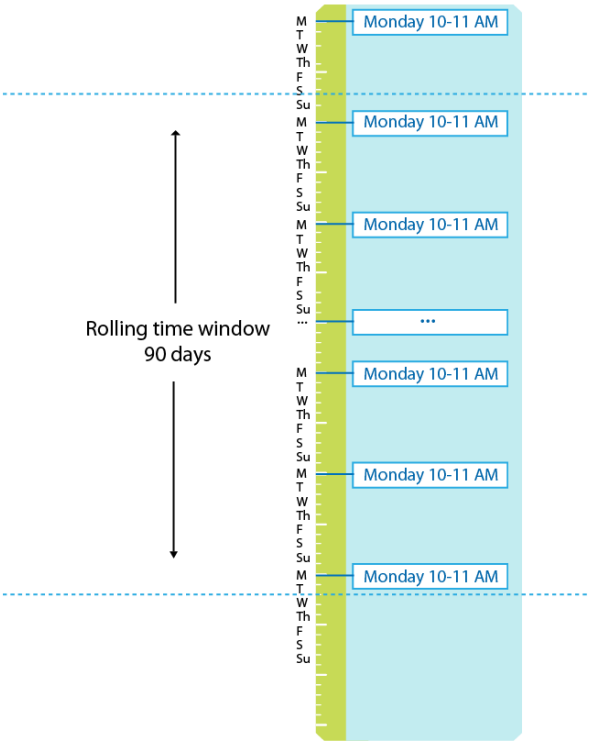
Performance expectations can differ between hours of the day, days of the week, or even months of the year. For example:

- A retail application may experience heavier traffic on the weekend than the rest of the week.
- A payroll application may experience a higher load at the beginning and end of the month compared to the rest of the month.

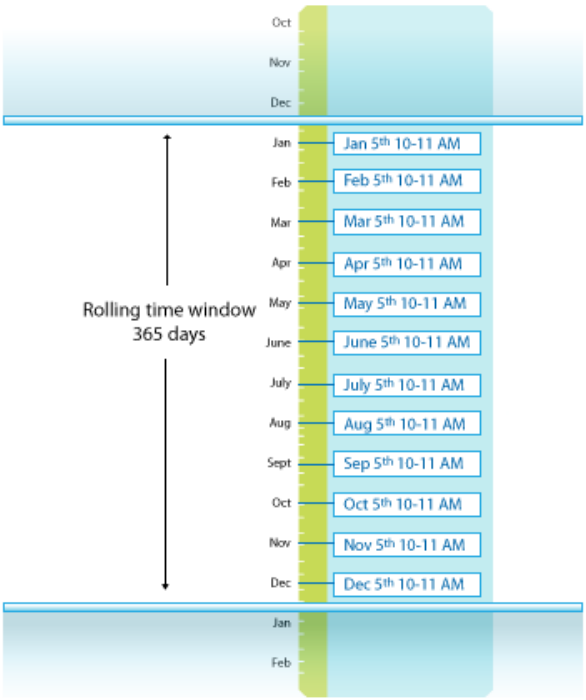
To account for this variation, you can use a rolling time period as the baseline context. A rolling time period establishes the baseline against data from the current hour taken at a daily, weekly, or monthly interval over the course of the time period.

For example, you have a baseline that uses the weekly trend with the time period configured for 90 days; this is one of the default baseline configurations available in AppDynamics.

At a given day and time, the baseline in effect is one that considers only the data accumulated on the same hour and day of the week over the last 90 days. This is illustrated in the following diagram.



A monthly trend calculates the baseline from data accumulated at the same hour but only on the same day of the month. So, for example, on January 5th at 10:30 AM, the baseline is established based on data accumulated at the same hour on the 5th of each month for the prior 365 days:



Baselines are not available immediately upon startup. It takes time and application load for the AppDynamics platform to collect data and create its initial baselines. The time depends on the type of baseline being used, whether daily, weekly, monthly, or none. Using none requires several hours before a baseline is available, daily takes several days, weekly takes several weeks, and so on.

> By default, baselines are not computed for metrics if there are fewer than 20 calls per minute. To configure the calls per minute threshold, contact your AppDynamics account representative.

How Standard Deviations are Calculated

The following standard formula is used to calculate the standard deviation:

```
standard deviation = sqrt [(B - A^2/N)/N]
```

where:
A = sum of the data values
B = sum of the squared data values
N = number of data values

The standard deviations calculations differ based on the type of metric used:

- If the time rollup type = Sum or Average
- If the cluster rollup type = Individual or Collective
  The cluster roll-up type is applicable only at the tier or app level (node-level metrics will have the same calculations).

The following example shows a standard deviation using metric types of:

- Time rollup type = Average
- Cluster rollup type = Collective

```
Metric - Calls Per Minute (Average, Collective)

(Hourly Metric Table)
```

Time	Sum	Count	Weight-Value	Weight-Value-Squared
09:00	180	2	180	300
10:00	300	2	300	780
11:00	150	2	150	195

For these metric values, the baseline of `type = All Data` lasts three hours (at 12:00) using the above values for calculations.

In the case of daily, weekly, or monthly seasonality, the data points considered are values for the same hour for all days, the same day of the week, or the same day of the year, respectively.

The baseline calculations, in this case, would be:

`Value` = `(180+300+150)/360 = 1.75`

`Standard deviation` = `sqrt(((300+780+195) - (((180+300+150)^2)/360))/360) = 0.69`

View Baselines

A baseline deviation is the standard deviation from a baseline at a point in time, represented as an integer value. You can set health rule conditions based on baseline deviations. For example, you can configure a warning condition as two standard deviations from the baseline and a critical condition as four standard deviations from the baseline. You can view built-in baselines and add new ones by navigating to **Configuration > Baselines**.

The daily trend is the default baseline. This is the baseline used by health rules if another baseline is not specified during health rule creation.

You can choose another as the default by selecting the baseline in the baseline list and choosing **Set as Default**.

> ⚠ Changing the default baseline changes the actual baseline for all existing health rules that rely on this default and do not specify another baseline. Be aware of your existing baselines and health rule definitions before you select this option.

Configure the Time Period

When configuring a baseline, you set the trend time period and the trend. The base time period comes in two forms:

- Fixed time range: from some specific date and time to a second specific date and time. For example, if you have a release cycle at a specific time you might limit your data collection to that specific time.
- Rolling time range: in which the most recent X number of days is used. This is the more common choice.

Transaction Snapshots

This page describes how to view transaction snapshot data to monitor and troubleshoot business transaction performance. To learn about when AppDynamics takes transaction snapshots or how to configure transaction snapshot settings, see Transaction Snapshot Collection.

AppDynamics monitors every execution of a business transaction in the instrumented environment, and the metrics reflect all such executions. For troubleshooting purposes, AppDynamics takes snapshots of specific instances of a transaction. A transaction snapshot gives you a cross-tier view of the processing flow for a single invocation of a transaction.

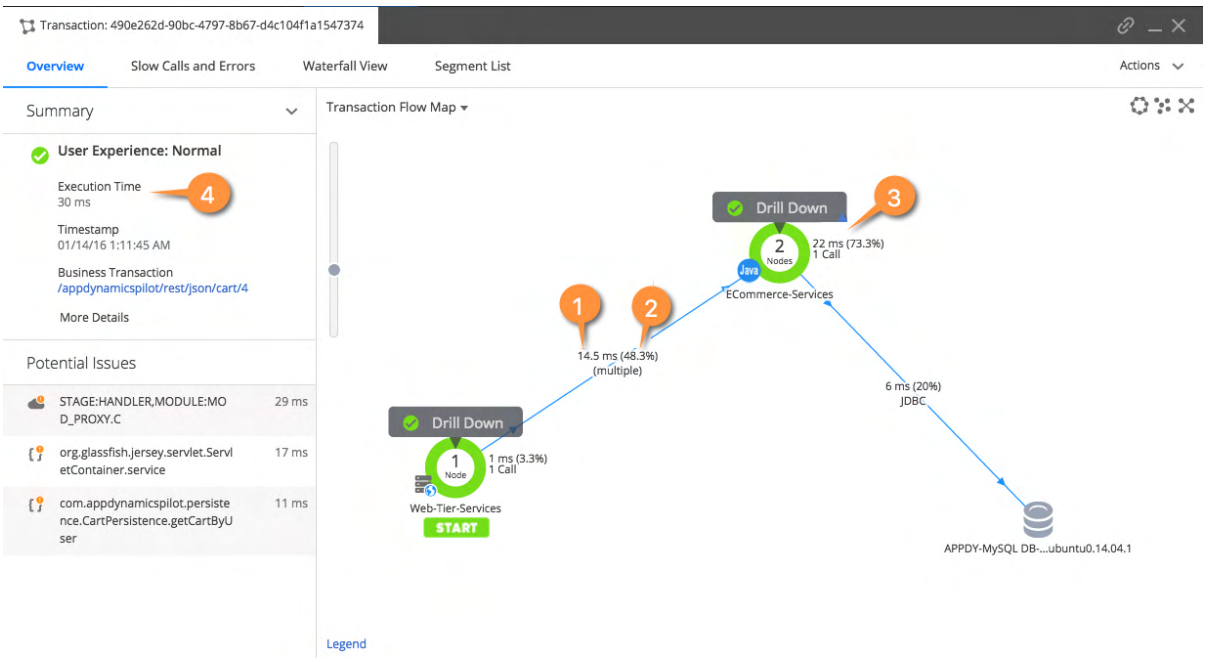
Call drill downs, where available, detail key information including slowest methods, errors, and remote service calls for the transaction execution on a tier. A drill down may include a partial or complete call graph. Call graphs reflect the code-level view of the processing of the business transaction on a particular tier.

View Transaction Snapshots

You can access business transaction snapshots from these locations in the AppDynamics Controller:

- **Troubleshooting** > **Slow Response Times** or **Troubleshooting** > **Errors** (left navigation tree for a business application)
- Transaction Snapshots tab on the Business Transaction Dashboard

Double-click a business transaction snapshot to display the snapshot viewer. This image and its accompanying table identify the metrics available in the transaction flow map:



Callout	Metric Name	Explanation
1	Tier Response Time (ms)	The total response time for the call as measured at the calling tier. This includes the processing time on the called tier as well as on any tiers and backends it calls in turn.
2	Percentage of Time Spent (%)	The time spent downstream processing at all downstream tiers and backends as measured by the calling tier and represented as a percentage of the entire execution lifespan of a business transaction. This metric does not include the processing time of asynchronous activities, if any.
3	Asynchronous Activity Processing Time (ms)	The processing time of all asynchronous activities at this tier. This metric does not contribute to the overall tier response time because the activity is asynchronous by nature. This metric is calculated by adding the execution times of all asynchronous activities at a tier and the time spent in communication between other tiers and backends as follows: *Asynchronous Activity Processing Time = Asynchronous-activity-1-processing-time + Asynchronous-activity-2-processing-time + so on.* In the Metric Browser, you can view the Average Async Processing Time metric, which shows the average of the async activity processing time over the selected time range.

4	Execution Time (ms)	Total time spent processing by the business transaction in all affected tiers and communication with other tiers and backends. This metric does not include the processing time of the asynchronous activities. However, in the case of Wait-for-Completion, the originating business transaction will take longer to process the request due to blocking and waiting for all the activities to complete before proceeding. The formula for this metric is calculated by summing up the processing times of a Business Transaction at a particular Tier /communication between Tiers/Backends as follows: *Execution Time = Time-spent-processing-in-Tier-1 + Time-spent-processing-in-Tier-2 + Time-spent-communicating-with-Tier-2 + so on.*

The **Potential Issues** panel highlights slow methods and slow remote service to help you investigate the root causes for performance issues. Click an item in the **Potential Issues** list to view the call in the call graph.

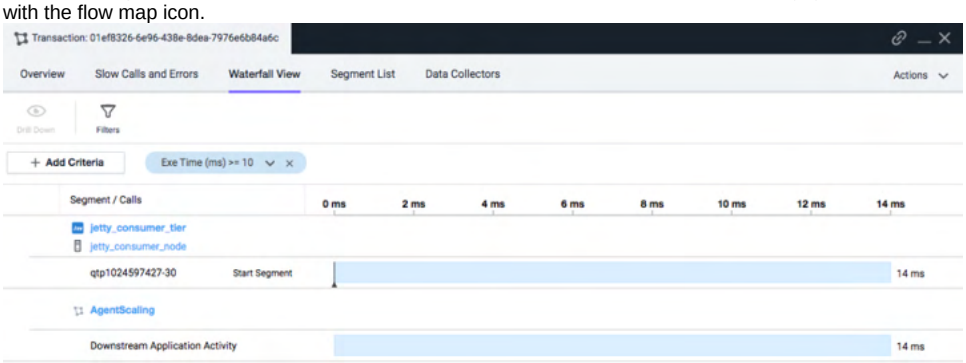
In the flow map for a business transaction snapshot, a tier with a drill-down link indicates AppDynamics has taken a call graph for that tier.

To drill into a call graph for a correlated application in a transaction with cross-application flow, you need *view* permissions to the destination busines

If you use the Java Agent and monitor Oracle databases with Database Monitoring and you have configured snapshot correlation between the Java Agent and the Database Agent, the flow map may also include database drill-downs.

The flow map or **Overview** is one of several views of the business transaction in the snapshot viewer. Other views include:

- **Slow Calls and Errors**, which presents information on the slowest database and remote service calls, slowest methods, and errors. You can gain further insight into these slow calls and errors either by viewing their details on by drilling down into their call graphs.
- **Waterfall View**, which shows the execution of an individual business transaction broken into execution segments. Each segment (shown as a blue bar) represents the time spent executing code on a particular thread within an instrumented application runtime, or the time spent waiting for responses from un-instrumented backends. Handoffs between segments are shown as solid lines for synchronous requests, and dotted for asynchronous requests. The waterfall view allows you to quickly determine which calls consumed the transaction time for a given snapshot. You can click a segment to view the resource wait time for its business transaction. Additionally, you can review downstream applications indicated with the flow map icon.



- **Segment List**, which shows the various legs of the transaction in descending order of duration and give access to their snapshots and allows you to drill down into their details.

Call Drill Downs

A call drill down contains details for that business transaction execution on a particular tier. It provides the code-level information for the transaction.

The contents of a transaction snapshot containing asynchronous segments look slightly different if you access the snapshot through the **Business Transaction** view or through the **App**/**Tier**/**Node** views:

- Initially the **Business Transaction** view only displays the originating segments for the transaction. You have the option to drill down into the asynchronous segments as desired.
- The **App**/**Tier**/**Node** views surface all the segments that are relative to that specific entity. When you access one of these views, you can see all segments, originating and asynchronous.

Node Drill Down

The node drill down organizes diagnostic data among these tabs:

- The Overview tab includes a problem summary, execution times, CPU time stamp, tier, node, process ID, thread name, etc.
- The Call Graphs tab lists call graphs showing the execution flow for the transaction on a given tier. For details, see Call Graphs.
- The Slow Calls and Errors tab lists all the slow method calls and calls that resulted in an error. You can use the *Hot Spots* slider to sort calls by execution time with the most expensive calls in the snapshot at the top.
- The Error Details tab exposes exception stack traces and HTTP error codes.
- The DB & Remote Service Calls tab shows all SQL query exit calls to databases and exit calls to other remote services such as web services, message queues, or caching servers. See Database queries and batching for more information about how AppDynamics handles SQL exit calls.

- The Server tab displays graphs for hardware—CPU Memory, Disk IO, Network IO— Memory—Heap, Garbage Collection, Memory Pools—JMX, and more. If you have Server Visibility, you will have access to full performance details for the server hardware and operating system
- For customers using Network Visibility, the Network tab shows charts related to the impact of the network on the transaction and other pertinent data. For information on network KPIs and troubleshooting see KPI Metrics in Network Dashboard and Application Flow Map and KPI Metrics in Right-Click Dashboards.
- The Data Collectors tab shows pertinent application data for the transaction snapshot. For configuration options, see Data Collectors.

  HTTP Data: HTTP payloads contain basic data such as the URL and session ID, and additional data for Servlet entry points, Struts, JSF, Web Services, etc. You can use HTTP data collectors to specify which query parameter or cookie values should be captured in the transaction snapshot.

  Cookies: The snapshot can use cookie values to help identify the user who initiated the slow or error transaction.

  User Data: User data from any method executed during a transaction, including parameter values and return values, to add context to the transaction. You can use method invocation data collectors to specify the method and parameter index.
  In cases where an exit call is made just before a business transaction starts, exit call information can show up in this field, particularly if the transaction is marked as slow or having errors. Please note that sensitive information on the exit call may be shown in this situation.

- The More tab shows how metrics for the node that deviate the most from the established baselines as Node Problems. It also shows all the Service Endpoints invoked during the snapshot and the Servlet URI and Process ID of the transaction.

Database Queries and Batching

AppDynamics normalizes SQL queries and by default does not display raw/bind values. You can configure SQL capture settings to monitor raw SQL data in the queries. Individual calls that take less than 1 second are not reported.

When returning data to a JDBC client, database management systems often return the results as a batched response. Each batch contains a subset of the total result set, with typically 10 records in each batch. The JDBC client retrieves a batch and iterates through the results. If the query is not satisfied, the JDBC client gets the next batch, and so on.

In the **SQL query** panel, a number followed by an X in the **Query** column means that the query ran the number of times indicated within a batch. The value in the **Count** column indicates the number of times that the batch job executed.

Database Drill down

Database drill downs provide this transaction information:

- **Queries**: lists the queries consuming the most time in the database as top SQL statements and Stored Procedures. Comparing the query weights to other metrics such as SQL wait times may point you to SQL that requires tuning.
- **Clients:** displays the hostname or IP addresses of the Top N clients using the database. A database client is any host that accesses the database instance.
- **Sessions:** displays the Session ID of the Top N sessions using the database sorted by time spent.
- **Schemas:** shows the names of the Top N busiest schemas on the database server.

Use the Transaction Snapshot List

You can view transaction snapshots generated in the UI time range from the **Transaction Snapshots** tab of the application, tier, node, or business transaction dashboards. From there you can:

- **Compare Snapshots** shows the performance of calls in two snapshots as a side-by-side comparison.
- Identify the most expensive calls / SQL statements in a group of Snapshots shows the calls that take the most time across the snapshots you have selected. You can select up to 30 snapshots.
- Find snapshots using the filter options.

The Controller purges transaction snapshots after two weeks by default, but you can configure the snapshot retention period. You can archive a snapshot to preserve it beyond the normal snapshot lifespan; for example, to retain a snapshot associated with a particular problem for future analysis. To archive a snapshot, select it from the list and choose **Actions** > **Archive**.

To archive a snapshot, you need the **Application level - Can create applications** permission.

The file cabinet icon in the far right column indicates that the snapshot is an archive snapshot ().

To display only archived snapshots in the snapshot list, filter the snapshot list and check **Only Archived**.

Call Graphs

Related pages:

- Call Graph Settings
- Data Collectors

A call graph in a transaction snapshot shows you business transaction processing information on a particular tier that participated on the business transaction. A call graph lists the methods in a call stack and provides information about each call.

A call graph can tell you general information such as the total execution time, the node name, the time stamp for the start of execution, and the unique identifier for the business transaction instance. Call graphs help you diagnose performance issues and optimize the flow of a complex business transaction.

Call graphs are also captured by the Node.js agent in process snapshots, which describe an instance of a CPU process on an instrumented Node.js node. See Event Loop Blocking in Node.js.

Call Graph Types

A call graph can be one of these types:

- Full call graphs capture the entire call sequence for the business transaction invocation. In this case, call graphs exist for each monitored node involved in the processing of the business transaction. Periodically collected and diagnostic snapshots are always full call graphs.
- Partial call graphs represent the subset of the call sequence for processing a business transaction, typically from the point at which the transaction has been determined to be slow or have errors. Partial call graphs are identified as such in the transaction snapshot.

View Call Graphs

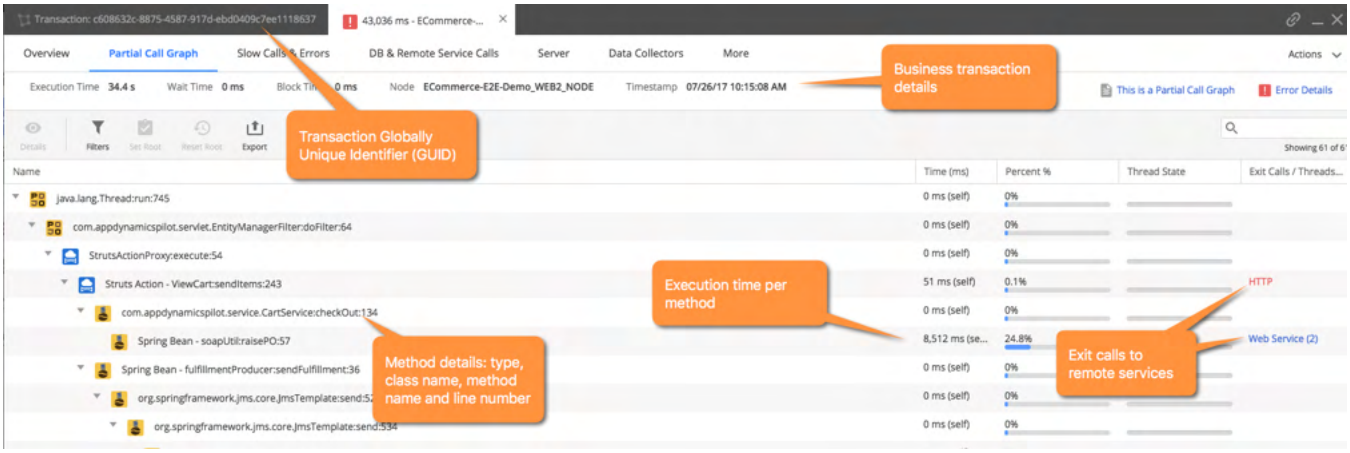
To view a call graph:

1. Open the dashboard for the business transaction for which you want to see a call graph and click the Transaction Snapshots tab.
2. In the transaction snapshot list, double click a snapshot.
3. Click the Drill Down link for the tier on which you want to see a call graph. An asterisk next to the link means that multiple calls were made at this tier for the transaction, which a dialog offers as choices to drill down into.

In the panel, the method execution sequence shows the names of the classes and methods that participated in processing the business transaction on this node, in the order in which the flow of control proceeded.

For each method, you can see the time spent processing and the line number in the source code, enabling you to pinpoint the location in code that could be affecting the performance of the transaction.

The call graph panel includes UI controls for navigating large call graphs. For instance, you can use the filter field to have only a particular method or types of methods displayed. When you find a method of interest, you can set it as the root method in the tree to view only the portion of the call graph that occurs from that point forward.



The call graph displays exit call links for methods that make calls to outside applications and services such as database queries and web service calls. Exit calls that do not exceed a minimum time threshold may not be represented.

SQL Query Exit Calls in Call Graphs

Often times application code executes database calls in multiple phases. For example:

- Construct and execute a SQL query
- Iterate over the result set

For example:

```
stmt = con.createStatement();
# Execute the query
ResultSet rs = stmt.executeQuery(query);
# Iterate through the results
while (rs.next()) {
        String PizzaType = rs.getString("PIZZA_TYPE");
    float price = rs.getFLoat("PRICE");
}
```

The Java Agent represents JDBC SQL query executions and the iterations over the result set separately in the call graph:

- For the execute call, the call graph shows the SQL query and the execution time. You can access the exit call details via a database link in the **External Call** column.
- For the iteration over the shows a count of the number of iterations and the total time. You can access the exit call details via a result set iteration link in the **External Call** column.

The .NET Agent does not separate ADO.NET SQL query execution exit calls from iteration over the result set in the call graph.

The Python Agent and the Node.js Agent sometimes separate out iterations as separate exit calls based upon the database driver used to make the call.

> AppDynamics does not always represent every database call as a separate exit call. Therefore you may see a greater number of database calls than the number of exit calls in the call graph.

Minimum Capture Threshold Times for SQL Calls

The query and result set iteration are both subject to the minimum capture threshold times for SQL calls, 10 ms by default. If the execution time for a query or result set iteration does not exceed the threshold it does not appear in the call graph. This means a given database interaction may be represented in a call graph in one of three ways depending on which parts of the operation took longer than 10 ms, if any:

1. Only the query appears (callout 1 in the image below)
2. Only the ResultSet iteration appears (callout 2)
3. Both appear (callout 3)

Click the link to view more details for the SQL exit call, or the result set iteration.

Maximum Number of SQL Queries in Snapshots

The maximum number of SQL queries in snapshots is set to 500 by default. For Java Agents, you can increase the value of the `max-jdbc-calls-per-snapshot` node property if you do not see expected SQL queries in a call graph.

Request Root Node

The call graphs in transaction snapshots of the Node.js, PHP, and Python agents have an artificial root node named `{request}` .

For Python and PHP, the purpose of this node is to reconcile time differences between the total business transaction time and the sum of the observed call graph nodes, to act as a single entry point for the call graph, and to contain any exit calls which could not be attributed to any other nodes.

For Node.js, the `{request}` node in a business transaction call graph signals the availability of a process snapshot call graph that intersects with the transaction snapshot. If there is no `{request}` node, no process snapshot is available. See Process Snapshots and Business Transaction Snapshots.

Node.js Call Graphs

When the Node.js agent captures a complete call graph, the call graph tab of the transaction snapshot displays the total execution total time of the transaction, including wait times, at the top the call graph.

The execution time shown in the call graph pane is the sum of the times of the individual method calls inside the call graph. This value may be less than the total execution time because it does not include wait times between calls.

Display Excluded Classes in Call Graphs

The sequence of method invocations in a call graph may involve hundreds of classes. These classes include user application classes, framework classes (such as Tomcat or Websphere runtime classes), Java/J2EE core libraries, .NET Framework classes, and more.

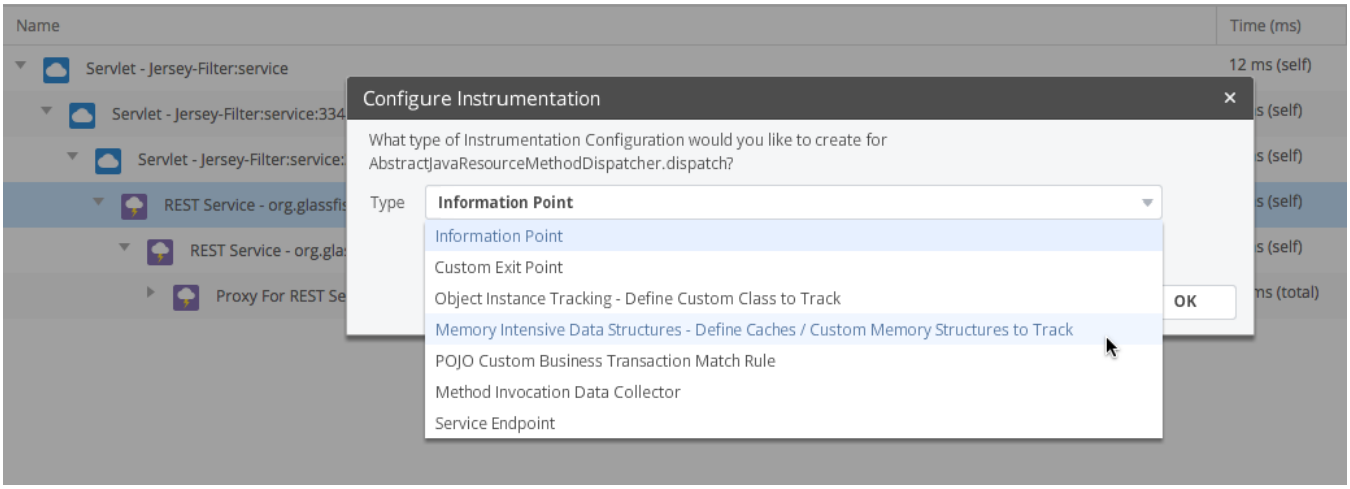
Not all of those classes may be relevant to the type of troubleshooting you normally do. In most cases, your primary interests are the custom application classes you have built on top of the framework or language platform. Other classes may be needed, but only on rare occasions.

To make the call graph more readable and to avoid the overhead of processing the timing information for non-user-application classes, other classes are not shown in the call graph.

If packages (Java) or namespaces (.NET) have been excluded, you can click the *packages/namespaces have been excluded* link to view the packages, and optionally configure the classes to be included in call graphs.

Configure Instrumentation

You can right-click any item in a call graph and select **Configure Instrumentation for this Class/Method**.



The Configure Instrumentation panel presents a drop-down from which you can select the type of configuration that you want to create for the method.

Diagnostic Sessions

By default, AppDynamics retains transaction snapshots that it captures at regular intervals and when a transaction is detected to be performing abnormally. The snapshots may have full, partial, or no call graphs. See Transaction Snapshots.

Diagnostic sessions let you trigger a timed session in which AppDynamics captures transaction snapshots for a business transaction at a more frequent interval that you set. In the session, AppDynamics captures full call graphs for the transactions.

Diagnostics sessions can also be triggered automatically by a health rule violation or in response to an abnormal amount of slow or error transactions. By default, if more than 10% of the requests in a minute are slow, AppDynamics starts a diagnostic session.

If the diagnostic session is triggered manually, the diagnostic session collects snapshots on all the nodes that the selected business transaction passes through. If the diagnostic session is triggered to start automatically, the diagnostics session collects snapshots on the triggering node.

Trigger a Diagnostic Session Manually

To start a diagnostic session manually, you need the **Start Diagnostic Sessions** permission.

You can start a diagnostic session from the business transaction list. In the list, select the transactions and start the session from the **More Actions** menu.

To view the transaction snapshots gathered in the session, in the business transaction dashboard, choose **Transaction Snapshots > Diagnostic Sessions**.

Configure Automatic Diagnostic Sessions for Slow and Error Transactions

You configure settings that trigger automatic diagnostic sessions from the transaction threshold configuration page. You can configure the threshold settings that trigger a diagnostic session as well as the snapshot collection frequency and duration once the session is triggered.

For performance reasons, you want to limit the duration and frequency of diagnostic sessions to the minimum required time to obtain the maximum amount of information for troubleshooting purposes. To avoid continuous collection when there are ongoing performance problems, configure the wait period between sessions and increase the time as needed.

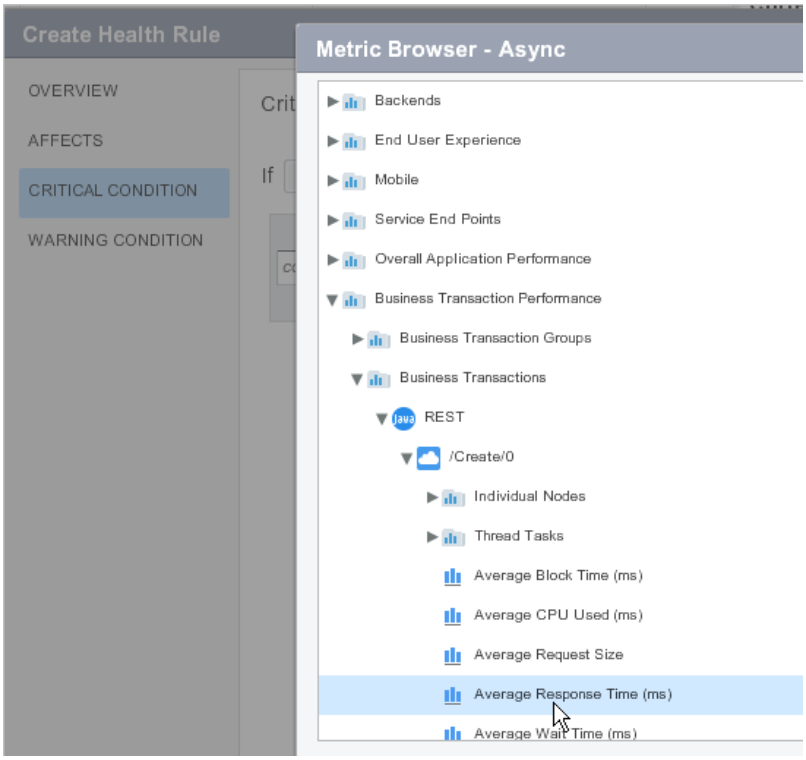
Use Diagnostic Sessions for Asynchronous Activity

Diagnostic sessions are usually triggered based on the overall performance of a business transaction. However, the overall performance of the business transaction may not reflect the execution time of asynchronous activity. It's possible that the originating transaction executes within normal bounds, while the asynchronous activity takes much longer.

To initiate diagnostic sessions for these scenarios, you can use health rules based on a performance metric (such as response time) of the asynchronous activity. Then create a policy that triggers a diagnostic session when the health rule violates. The general steps to do this are:

1. Create a custom health rule based on the asynchronous metric, such as average response time. The metrics for thread tasks are visible in the metric browser under the **Thread Tasks** node for transactions with asynchronous activity. Each thread task has an individual node (usually its

simple class name). Remember to select **Custom** as the type for the health rule.



2. Create a policy that is based on the baseline of the asynchronous metric of interest, for example, the average response time.
3. Configure the policy to trigger a diagnostic session on the affected business transaction. See Business Transactions.

Transaction Snapshot Collection

This page describes scenarios that trigger AppDynamics to take a Transaction Snapshot and how to configure transaction snapshot settings.

AppDynamics takes transaction snapshots of select instances of a transaction and provides a cross-tier view of the processing flow for a single invocation of a transaction. For information about the content of a transaction snapshot, see Transaction Snapshots.

Subject to the guidelines and limits described in the following sections, snapshots are taken in these cases:

- The app agent determines the user experience for the business transaction to be slow or the transaction incurred an error.
- The app agent collects snapshots during periodic snapshot collection.
- The app agent collects snapshots during a diagnostic session.

Snapshot and Call Graph Retention Rules

For a given transaction instance, a call graph may be available for some tiers but not all. The following guidelines describe the rules for when the app agent captures transaction snapshots for the originating and downstream tiers in a transaction. They also describe the type of call graphs a snapshot might include depending on the cause for the snapshot. The guidelines apply to business transaction correlation as well as cross-application flow.

- Any tier (originating or continuing) takes a snapshot when it recognizes that it is experiencing slow, very slow, or stalled response times or has errors.
- An originating tier takes a transaction snapshot:
    - When the agent starts a diagnostic session on the originating tier because it has detected a pattern of performance problems. You can also manually start a diagnostic session from the **Business Transaction Dashboard**. See Diagnostic Sessions.
    - When the agent identifies slow, very slow, or stalled response times, or errors on the originating tier. These snapshots may have partial call graph information because they start at the time when the transaction slowed or experienced an error.
    - Based on the periodic collection schedule. By default, the agent captures one snapshot every 10 minutes.
- The downstream tier captures snapshots when the tier immediately upstream to it tells it to take a snapshot. An upstream tier might direct its downstream tier to take a snapshot under these circumstances:
    - The upstream tier is taking a snapshot for a diagnostic session.
    - The upstream tier is taking a snapshot based on the periodic collection schedule.

Within the guidelines, snapshot retention is also subject to snapshot retention limits.

Transaction Snapshot Limits

Snapshot retention limits prevent excessive resource consumption at the node level:

- Originating transaction snapshots are limited to a maximum of 20 originating—5 concurrent—snapshots per node per minute.
- Continuing transaction snapshots are limited to a maximum of 200—100 concurrent—snapshots per node per minute.

AppDynamics applies snapshot retention limits to error transactions as well. As a result, not every error occurrence that is represented in an error count metric, for example, will have a corresponding snapshot. For error transactions, the following limits apply:

- For a single transaction, there is a maximum of two snapshots per minute.
- Across transactions, the maximum is limited to five snapshots per minute (specified by the node property `max-error-snapshots-per-minute`).

Configure Snapshot Periodic Collection Frequency

By default, AppDynamics collects a snapshot every 10 minutes.

Navigate to **Configuration > Slow Transaction Thresholds**. You can modify this default in the **Slow Transaction Thresholds** configuration page.

The value will apply to subsequently created business transactions, but if you check **Apply to all Existing Business Transactions**, all existing business transactions are affected by the change as well.

If you have a high load production environment, it is important that you do not use low values for snapshot collection, or configure collection on a very frequent basis. When there are thousands or millions of requests per minute, collecting snapshots too frequently may result in many extra snapshots that are not highly useful. Either turn off the periodic snapshots and apply to all Business Transactions, or choose a very conservative (high) rate depending on the expected load. For example, if you have a high load on the application, choose every 1000th executions or every 20 minutes, depending on the load pattern.

End-to-End Latency Performance

Works with:

The response time metric for a business transaction can tell you a lot about the performance of the transaction. In asynchronous applications, however, it may not tell you everything.

Consider a method that spawns other threads, possibly on other tiers, and then immediately returns control to the calling thread. The return of control stops the clock on the transaction in terms of measuring response time, but meanwhile, the logical processing for the transaction continues.

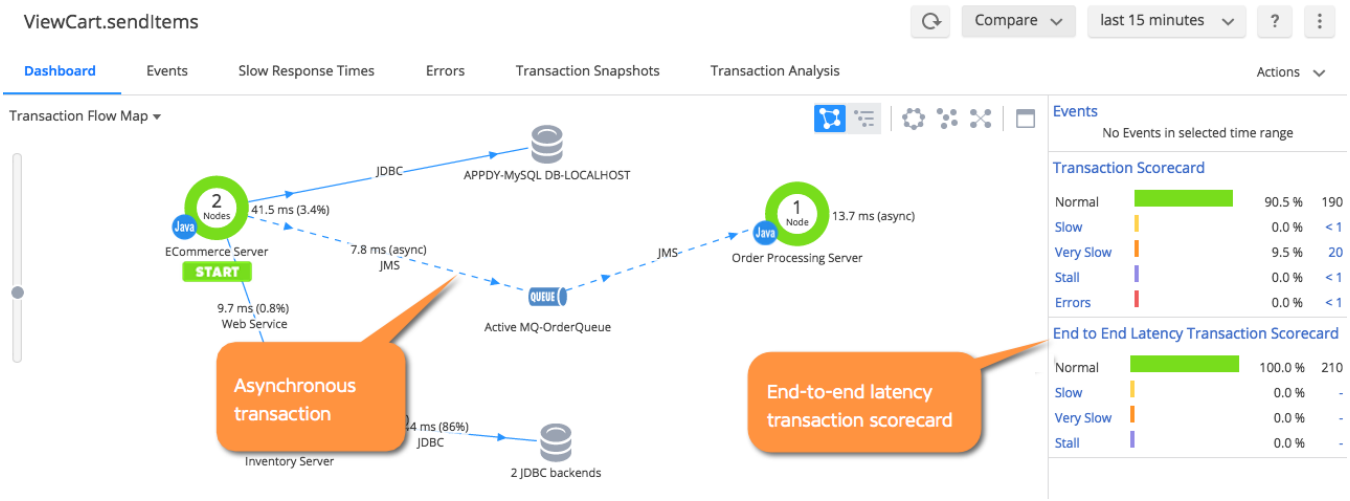
End-to-end latency metrics can reflect the response times for such asynchronous transactions.

Before You Begin

To monitor end-to-end metrics in AppDynamics, you must first configure the endpoints for your asynchronous transactions. See Asynchronous Transaction Demarcators.

Monitor End-to-End Performance

After you enable end-to-end latency transaction monitoring, end to end metrics appear in the metric browser for individual nodes and the overall application. The business transaction flow map also shows the *End to End Latency Transaction Scorecard*.



The End to End Latency Transaction Scorecard shows health based on the end-to-end asynchronous transaction time. The scorecard criteria—normal, slow, very slow and stall—for asynchronous transactions are the same as for the standard transaction scorecard. See Transaction Thresholds for more information. Note that the Transaction Scorecard reflects the data for the originating business transaction only, not end-to-end transactions.

End-to-End Performance Metrics

The performance metrics for end-to-end message transactions, which can be found in the overall application metric tree in the metric browser, are:

- Average End to End Latency: The average time in milliseconds spent processing end to end message transactions over the selected time frame.
- Number of Slow End to End Messages: The number of end to end transactions that exceeded the slow threshold for the business transaction over the selected time frame. The slow user experience is based on the slow transactions thresholds applicable to the general business transaction. Separate thresholds do not apply to end-to-end latency.
- Number of Very Slow End to End Latency Times: The number of end to end transactions that exceeded the very slow threshold for the business transaction over the selected time frame. The very slow user experience is based on the very slow transactions thresholds applicable to the general business transaction. Separate thresholds do not apply to end to end latency.

If you enable **End to End Latency Time (ms)** view option for the **Business Transactions** list, the waterfall view shows the end-to-end transaction performance for asynchronous transactions.

Monitor Background Tasks

Related pages:

- POCO Transaction as a Background Task - .NET

A background task is a processing activity such as an administrative task or internal process that you want to monitor separately from primary business transactions, such as a cron job that backs up a server.

While the process may be important to your environment, it is likely that you do not want to monitor them in the same way as primary business transactions. Background task performance metrics do not count toward the average response time of the tier or node.

During business transaction discovery, AppDynamics attempts to identify processes in the environment that are background tasks and classify them accordingly. You can modify the default configuration based on your requirements and manually change business transactions to background tasks.

View Background Tasks

Although background tasks are monitored separately, they display with the other business transactions in the Business Transactions list. Background tasks are identified in the list by the background task icon ().

This information is available on background tasks:

- The Business Transactions list shows currently executing background tasks and their statistics.
- The Metric Browser shows the response time for each background task.
- Transaction snapshots are generated for each execution of a background task. However, if a particular job runs frequently, the snapshot may not capture all details for each execution.

The tier and application-level metrics do not reflect background task activity. In addition, background tasks are not reflected on the application dashboard.

Reclassify a Business Transaction as a Background Task

You can reclassify any existing business transaction as a background task by right-clicking it in the business transaction list and choosing **Set as Background Task** option. Choosing **Set as User Transaction** turns a background task into a primary business transaction.

Enable Automatic Discovery for Background Tasks

At transaction discovery, a transaction is categorized as a background task based on a setting in a custom match rule that matches the transaction.

AppDynamics includes preconfigured match rules for Java frameworks that are likely to be background tasks, including for JCronTab, JavaTimer, Cron4J and Quartz frameworks. The match rules are disabled by default.

You can enable the match rules, modify them, or create your own custom match rules to have matched transactions categorized as background transactions.

To enable background task detection configuration, from the application or custom tier level configuration, edit an existing match rule or create a new one. Enable the rule, and check the Background Task option to business transactions discovered by this rule categorized as background tasks.

Configure Thresholds for Background Tasks

You can configure performance thresholds for background tasks separately from primary business transactions. From the **Configuration > Slow Transaction Thresholds** page, click the Background Tasks Thresholds tab to access settings specific for background tasks.

AppDynamics recommends that you configure a threshold that is suitable for the background task in your environment. Use static thresholds for slow and very slow background tasks, if they have infrequent load patterns such as once every night. This is because the dynamic moving average-based thresholds are more suitable for production load scenarios and will automatically classify a background process as slow or very slow.

Business Transactions Logging

Related pages:

- Agent Log Files
- Java Agent Logging
- .NET Agent on Windows Logging
- Dynamic Agent Proxy Logging

The Business Transactions log contains this information:

- Discovered business transactions
- Discovery stack trace for each entry point
- Dropped business transactions, which show up in the "All Other Traffic" business transaction when there is business transaction overflow. Only executed methods show up in the business transaction log. If the business functionality is not exercised by the application, the code is not executed, and the methods are not examined as potential entry points.

For the Java Agent and the proxy, the name of the Business Transactions log is `BusinessTransactions`*`Year_mon_day_hr_min`*`.#.log` where # is the log set.

For the .NET Agent, the name of the Business Transactions log is `BusinessTransactionsLog.#.txt` where # is the log set.

See Agent Log Files for information about how the logs are organized into sets that roll over. Within a set, the Business Transactions log file can reach a maximum of five MB.

Service Endpoints

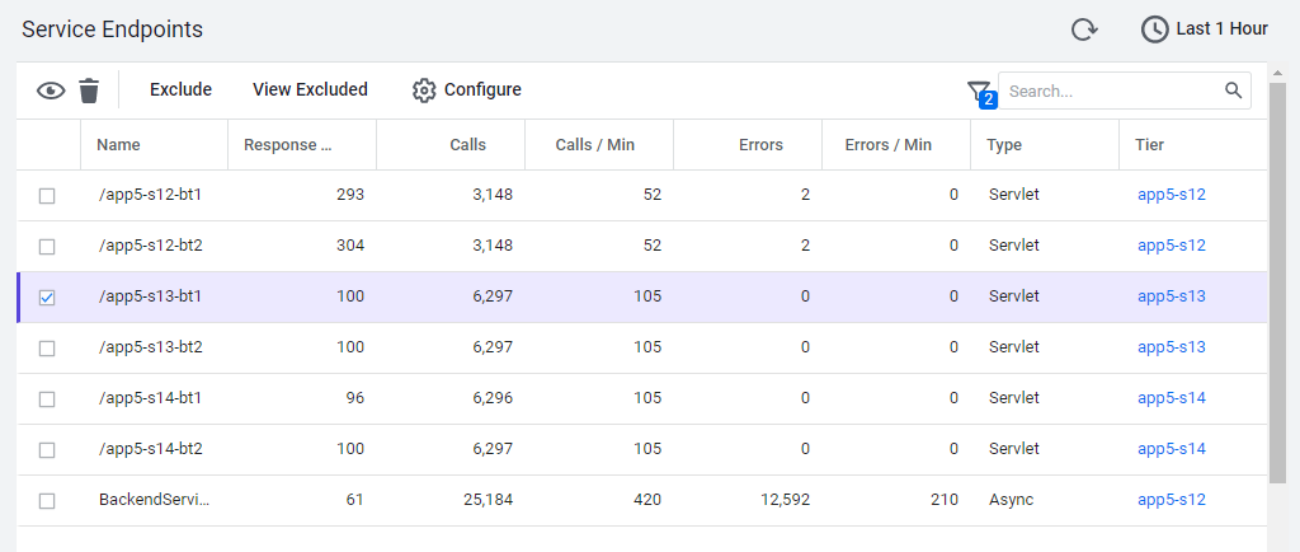
Works with:

A business transaction offers a view of application performance that cuts across the environment, reflecting all services that participate in fulfilling the transaction. However, another way of viewing application performance focuses on the performance of a particular service independently of the business transactions that use it.

Like business transactions, service endpoints provide key performance indicators, metrics, and snapshots. However, they provide that information exclusively in the context of that service, omitting business transaction context or downstream performance data.

Monitor Service Endpoints

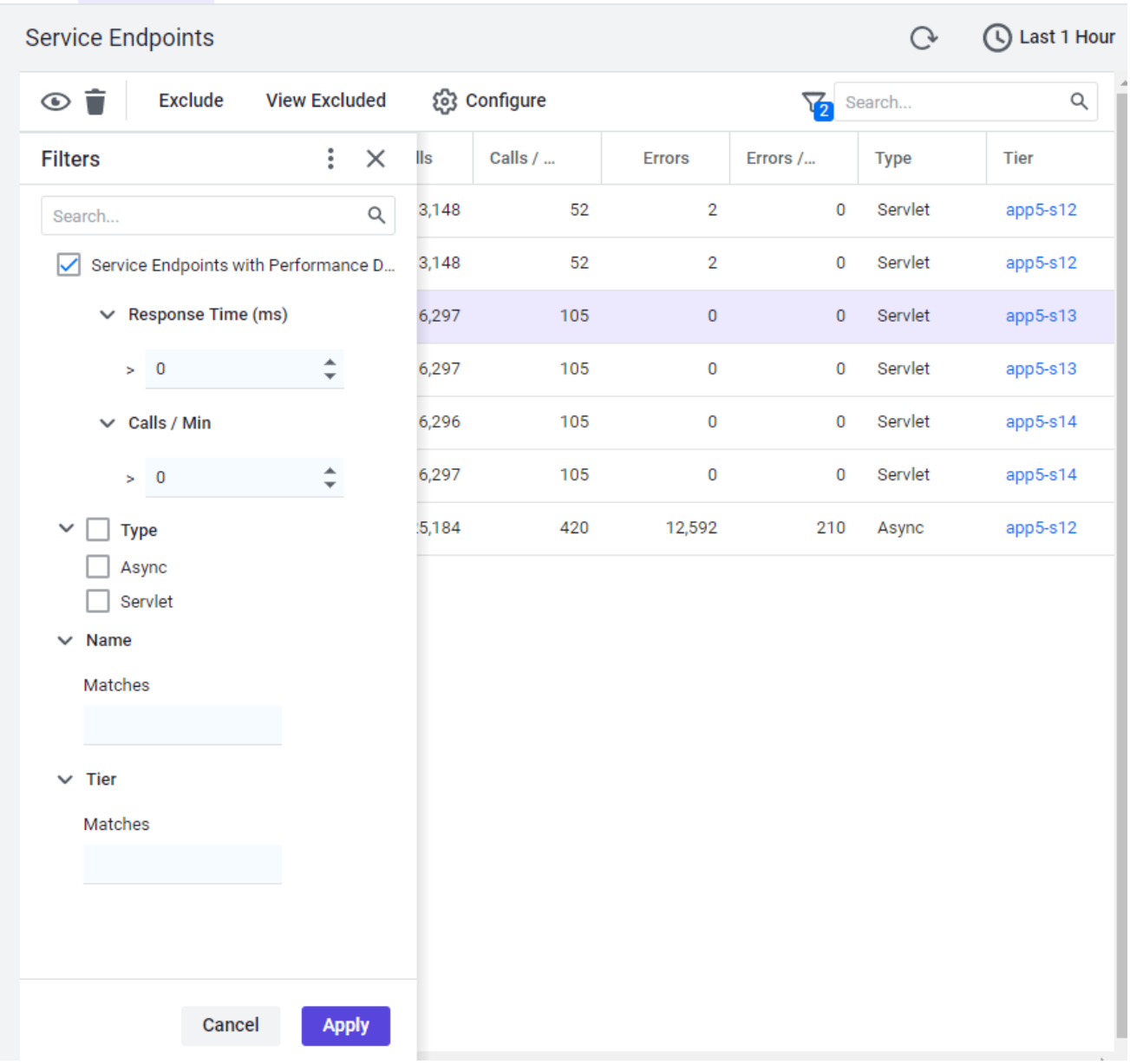
From the **Applications** navigation menu in the Controller, select **Service Endpoints** to view a list of service endpoints with key performance metrics.



Name	Response ...	Calls	Calls / Min	Errors	Errors / Min	Type	Tier
/app5-s12-bt1	293	3,148	52	2	0	Servlet	app5-s12
/app5-s12-bt2	304	3,148	52	2	0	Servlet	app5-s12
/app5-s13-bt1	100	6,297	105	0	0	Servlet	app5-s13
/app5-s13-bt2	100	6,297	105	0	0	Servlet	app5-s13
/app5-s14-bt1	96	6,296	105	0	0	Servlet	app5-s14
/app5-s14-bt2	100	6,297	105	0	0	Servlet	app5-s14
BackendServi...	61	25,184	420	12,592	210	Async	app5-s12

Service endpoints that contain performance data within the selected time range display in the list.

Select the **Filter** icon (on the top right of the page) to set various filtering options for active and inactive service endpoints within the designated time range. Other view options include showing transactions of certain types or ones that exceed a configurable average response time. After you set the desired filters, select **Apply**.



Additionally, you can view, delete, and exclude selected service endpoints by selecting the options from the top menu bar. Select **Exclude** to hide the service endpoint from the list.

To completely disable metric collection for the service endpoint, you can create an exclude rule by selecting **Configure**. See Service Endpoint Detection.

Diagnostic sessions are not intended to run in the context of a service endpoint alone. As a result, you cannot directly start a diagnostic session from a service endpoint. If you run a diagnostic session on the business transactions that include calls to the service endpoint, the transaction snapshots from these diagnostic sessions display in the service endpoint dashboard.

Service Endpoint Health Rules

You can create health rules to monitor the parameters that represent the normal or expected operations for your Service Endpoints. The parameters rely on metric values, for example, the average response time. When the performance of a service endpoint violates the conditions set by a health rule, a health rule is violated. Alerts notify you of any violation and initiate remediation actions to mitigate the violation event. It is important that you configure alerts appropriately to ensure that you do not miss any alerts or receive false alerts. 'Alert Sensitivity Tuning' (AST) helps you configure alerts with appropriate sensitivity. AST provides historical data for the metric or the baseline being configured and hence helps you visualize the impact of the alerting configuration. To create a health rule to monitor service endpoint parameters and fine-tune the sensitivity of the health rule using AST, see Create a Health Rule and Fine-tune Metric Evaluation.

Service Endpoint Limits

A service endpoint adds a small amount of system overhead. Agents capture approximately three metrics per service endpoint, so monitoring each service endpoint results in additional metric traffic.

To prevent the possibility of a boundless expansion of service endpoints, the Controller and agent configurations apply the following limits:

Element	Limit	For more information
Application Agent	25 service endpoints per entry point type	See `max-service-end-points-per-entry-point-type` on App Agent Node Properties Reference.
Java Agent	40 asynchronous worker thread service endpoints	See `max-service-end-points-per-async-type` on App Agent Node Properties Reference.
Node	100 service endpoints	See `max-service-end-points-per-node` on App Agent Node Properties Reference.
Controller account	4000 service endpoints	For an on-premises Controller, the limit is configurable using the `sep.ADD.registration.limit` configuration property accessible in the Administration Console.
Execution thread	1 endpoint	See `max-service-end-points-per-thread` on App Agent Node Properties Reference.

Service Endpoint Metrics

Service endpoint metrics display under **Service Endpoints** in the metric browser. Service endpoint metrics are subject to the same operations as other metrics, including those around metric registration, metric rollups for tiers, and limits on the number of metrics. Custom metrics are not available for service endpoints.

Customize Service Endpoints

For Java applications, AppDynamics automatically detects and registers service endpoints. For either Java or .NET, you can configure custom service endpoint detection rules. See Service Endpoint Detection.

Tiers, Nodes, and Naming

Related pages:

- Monitor Windows Hardware Resources
- Troubleshooting Machine Agent Installation
- Java Agent Configuration Properties
- Configure the .NET Machine Agent
- Node.js Setting Reference
- Node Naming in a PHP Environment

This page describes to view, create, and name tiers and nodes. In the Controller UI, you can use **Tiers & Nodes** to view and organize tiers, and the monitored nodes within each tier.

View All Tiers and Nodes

To view the list of tiers and nodes, click **Tiers & Nodes** in the left application panel. The tier list shows all tiers and health data for each tier.

In the tiers list, the **Machine Agent** column indicates whether the Standalone Machine Agent is reporting to the Controller. If the Machine Agent is not reporting, the column is empty. In this case, either an agent is not installed or it is not reporting.

You can change descending and ascending order of the list by clicking on any of the columns.

View Tier and Node Dashboards

In **Tiers & Nodes,** select a tier or node and click **Details**. The tier and node dashboards contain a flow map view at the tier or node level, including widgets for health statuses, load time, response time, and errors. The Memory tab shows memory information relevant to your application environment. Note that the specific information shown for each environment can vary, depending on the environment. PHP memory management is different than other languages. The relevant memory metrics supplied by the App Agent for PHP are in the Metric Browser.

Create a Tier

You can create a tier in the AppDynamics model of your environment by specifying the new tier in the agent configuration of your instrumented application (recommended method). However, you can also create tiers manually by navigating to **Tiers & Nodes > Actions > Create App Server Tier**.

After creating a tier, you can add nodes to the tier through the app agent configuration, or by reassigning an existing node to the new tier.

Rename Node, Tier, or Business Application in the UI

To rename a node, tier, or business application, see interacting with flow maps.

The UI prevents you from using certain special characters for nodes, tiers, and business applications names. Besides numbers, and upper and lower case characters, you can use these special characters in the names :

```
: ~ ! @ # $ % ^ * ( ) - _ + = / \ \ , . [ ] { } | ? .
```

Moving and Renaming Nodes

This page describes how to move and rename nodes. In the Controller UI, you can move and rename nodes between business applications or tiers to reflect changes in your environment model. See Overview of Application Monitoring.

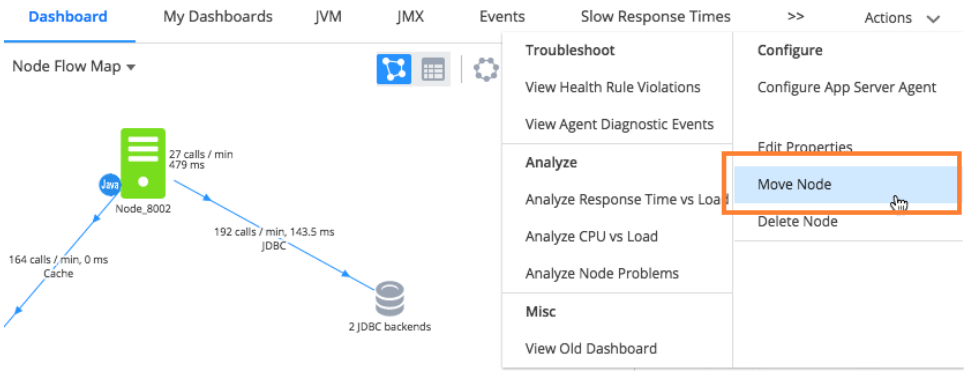
Move a Node

When you move a node to another tier or application in the Controller UI:

- All historical data is removed for that node, with no possibility for data recovery. Moving nodes in the UI is primarily intended for refining the AppDynamics application model of your environment, or other non-production scenarios. To retain historical data for the node, you can either change the tier name (without changing the application or node name) in the agent's `controller-info.xml` file or move the node to a different tier of the same application without data loss.
- The tier or application association in the agent's configuration file no longer reflects the actual tier or application association. The UI association is maintained unless the `force-agent-registration` flag is set to true in the agent's `controller-info.xml`.

You can move a node to a new application or tier with two methods:

- Controller UI: In the node dashboard, click **Actions** > **Move Node**. You do not need to restart the JVM using this method.



- `controller-info.xml`: Update the tier or application setting in the agent's `controller-info.xml` configuration file and restart the JVM.

Force Node Registration

If you moved a node in the UI and you want to move it again elsewhere using `controller-info.xml`, set the `force-agent-registration` property in the configuration file to `true` and restart the JVM. See Force Agent Registration Property.

Data Retention

When you move a node to another tier or application, all node-level historical data (metrics, events, and snapshots) is lost. The historical data is not recoverable. If retaining historical data for the node is important to you, you can change the tier, application or node name in the agent's `controller-info.xml` file.

Any tier-level data that was generated by the moved node is retained in the former tier. Similarly, when moving nodes between tiers in the same application, the tier-level data generated by the node is retained in the former tier.

Moving a Java Node and Machine Agent

If your JVM machine has both a Java Agent and a Machine Agent, you cannot change the associations in the Machine Agent `controller-info.xml` file. You can only change these associations either through the UI or by modifying the Java Agent `controller-info.xml` file described in Java Agent Configuration Properties.

Rename a Node

To rename a node, change the name in Controller UI and then in the app agent configuration file. Changing the node name in the configuration file ensures that the new name is retained through the next agent restart.

If you rename a node from the app agent configuration file only, the effect is that a new node is created with the restarted agent reporting data as that node. AppDynamics retains the historical data previously collected for the node, but the data is associated with the old node name. For most purposes, renaming the node involves changing it in the UI first, and then in the configuration file.

To change the name in the UI, follow the instructions for moving the node, but choose **Edit Properties** from the **Actions** menu, and enter the new name in the **Node Properties** dialog.

For information on changing the node name in the configuration files, see the agent installation or configuration documentation for your agent under Install App Server Agents.

For information on renaming a business application, see Business Applications.

Troubleshoot Node Problems

The page describes how to use the **Node Problem** viewer to troubleshoot node problems.

AppDynamics categorizes the ten items that deviate the most from the baseline performance as node problems. The Machine Agent must be installed on the machine hosting the node that you are troubleshooting.

Use the Node Problem Viewer

If accessed from the **Snapshot** viewer, the **Node Problem** viewer displays node problems for the time range of the snapshot. If accessed from the Node dashboard, the **Node Problem** viewer uses the time range set in the Node dashboard. You can edit the time range in the **Node Problem** viewer and then apply the new time range. You can also define and save a custom time range.

The right panel of the **Node Problem** viewer displays the metrics that deviate the most from their baselines for the specified time range.

On the left, you can set filters on what data appears. Options include:

- The baseline to use to define node problems.
- The type of metrics, whether custom, hardware metrics, JVM, and so on.
- Specify whether to display problems with values that are:
    - Higher than baseline
    - Lower than the baseline
    - Higher and lower than baseline

  For example, if you are only interested in CPU that is too high, set the **Analysis Type** to **Higher for Hardware Resources**. However, if you want to monitor the load on your machine continuously because low CPU usage would also be a problem, set the **Analysis Type** to **Higher and Lower** . The right panel shows the ten items that deviate the most from the baseline.

After changing an option, click **Analyze Node Problems** to apply the change.

Monitor JVMs

This page provides an overview of some of the tools AppDynamics provides for monitoring Java applications and troubleshooting common issues.

JVM Key Performance Indicators

A typical JVM may have thousands of attributes that reflect various aspects of the JVM's activities and state. The key performance indicators that AppDynamics focuses on as most useful for evaluating performance include:

- Total classes loaded and how many are currently loaded
- Thread usage
- Percent CPU process usage

On a per-node basis, AppDynamics reports:

- Heap usage
- Garbage collection
- Memory pools and caching
- Java object instances

You can configure additional monitoring for:

- Automatic leak detection
- Custom memory structures

View JVM Performance

You can view JVM performance information from the Tiers & Nodes dashboard or from the Metric Browser.

In the Tiers & Nodes dashboard, see the following tabs for JVM-specific information:

- The Memory subtab of the Tiers & Nodes dashboard allows you to view various types of JVM performance information: Heap and Garbage Collection, Automatic Leak Detection, Object Instance Tracking, and Custom Memory Structures.
- The JMX subtab of the Tiers & Nodes dashboard allows you to view information about JVM classes, garbage collection, memory, threads, and process CPU. In the JMX Metrics subtab metric tree, click an item and drag it to the line graph to plot current metric data.

In the Metric Browser, click **Application Infrastructure Performance** and expand the JVM folder for a given node to access information about Garbage Collection, Classes, Process CPU, Memory, and Thread use.

Alert for JVM Health

You can set up health rules based on JVM or JMX metrics. Once you have a health rule, you can create specific policies based on health rule violations. One type of response to a health rule violation is an alert. See Alert and Respond for how to use health rules, alerts, and policies.

You can also create additional persistent JMX metrics from MBean attributes. See Configure JMX Metrics from MBeans.

JVM Crash Guard

Using the Machine Agent, when a JVM crash occurs on a machine or node, you can be notified almost immediately and take remediation actions. A JVM crash is because it may be a sign of a severe runtime problem in an application. Implemented as part of JVM Crash Guard, JVM crash is an event type that you can activate to provide the critical information you need to expeditiously handle JVM crashes.

Memory Management

Memory management includes managing the heap, certain memory pools, and garbage collection. This section focuses on managing the heap. You can view heap information in the metric browser or in the Memory tab for a given node as directed in **Monitoring JVM Information**.

The size of the JVM heap can affect performance and should be adjusted if needed:

- A heap that is too small will cause excess garbage collections and increases the chances of `OutOfMemory` exceptions.
- A heap that is too big will delay garbage collection and stress the operating system when needing to page the JVM process to cope with large amounts of live data.

See Garbage Collection.

Detect Memory Leaks

By monitoring the JVM heap and memory pool, you can identify potential memory leaks. Consistently increasing heap valleys might indicate either an improper heap configuration or a memory leak. You can identify potential memory leaks by analyzing the usage pattern of either the survivor space or the old generation. To troubleshoot memory leaks, see Java Memory Leaks.

Detect Memory Thrash

Memory thrash is caused when a large number of temporary objects are created in very short intervals. Although these objects are temporary and are eventually cleaned up, the garbage collection mechanism might struggle to keep up with the rate of object creation. This might cause application performance problems. Monitoring the time spent in garbage collection can provide insight into performance issues, including memory thrash. For example, an increase in the number of spikes for major collections affects the JVM's ability to serve Business Transaction traffic and might indicate potential memory thrash.

The **Tiers & Nodes > Memory > Object Instance Tracking** subtab helps you isolate the root cause of possible memory thrash. To troubleshoot memory thrash, see Java Memory Thrash.

Monitor Long-lived Collections

AppDynamics automatically tracks long-lived Java collections (HashMap, ArrayList, and so on) with Automatic Leak Detection. Custom Memory Structures that you have configured, display in **Tiers & Nodes** > **Memory** > **Custom Memory Structures**.

AppDynamics provides visibility into:

- Cache access for slow, very slow, and stalled business transactions
- Usage statistics (rolled up to Business Transaction level)
- Accessed keys
- Deep size of the internal cache structure

Automatic Leak Detection for Java

This page describes how to view the Automatic Memory Leak dashboard and enable automatic leak detection. See Java Memory Leaks.

JVM Requirements

- Oracle JVM >= 1.5
- JRockit JVM >= 1.5
- IBM JVM >= 1.6

Automatic Memory Leak Dashboard

The Automatic Memory Leak dashboard shows:

- **Collection Size**: The number of elements in a collection.
- **Potentially Leaking**: Potentially leaking collections are marked as red. You should start diagnostic sessions on potentially leaking objects.
- **Status**: Indicates if a diagnostic session has been started on an object.
- **Collection Size Trend**: A positive and steep growth slope indicates a potential memory leak.

To identify long-lived collections, compare the JVM start time and Object Creation Time.

You must have the Configure Agent Properties permission to activate or deactivate automatic leak detection. See Create and Manage Custom Roles.

Enable Automatic Leak Detection

1. In the left navigation pane, click **Tiers & Nodes**.
2. In the right pane, expand the tier of the node you want to configure.
3. Select the node and click **View Dashboard**.
4. Click **Memory** > **Automatic Leak Detection**.
5. Click **On**.
6. Click **Start On Demand Capture Session** to detect leaking collections.

Test before implementing

Leak Detection may affect performance, so test this functionality in a preproduction environment or on a single node first.

AppDynamics begins to automatically track the top 20 application classes and the top 20 system (core Java) classes in the heap.

Troubleshoot Leak Detection

If you cannot see any captured collections, ensure that you have the correct configuration for detecting potential memory leaks. As described in the on-screen instructions you may need to lower the minimum collection size from the default, 5 MB:

To configure the agent of a specific node to capture smaller collections:

1. On the **Tiers & Nodes** list, expand a tier and double-click the node you want to configure.
2. Ensure the **App Server Agent** tab is active and click **Configure**.
3. Select the Application, Tier, and Node and then click **Use Custom Configuration**.
4. Search for `minimum-size-for-evaluation-in-mb` in the node properties list. Set the value to a smaller size.
5. Choose to overwrite the node configuration.

Object Instance Tracking for Java

This page describes how to configure and use object instance tracking for Java applications. For more information about why you may need to configure this, see Java Memory Thrash.

When object instance tracking is enabled, AppDynamics tracks the top 20 application and top 20 system (core Java) classes in the heap by default. You can also track specific classes.

> AppDynamics does not perform allocation tracking for core Java classes by default since doing so would add significant system overhead. However, it is possible to track core classes on a short-term basis (for example, while troubleshooting) or in pre-production environments. For more information, see Enabling Allocation Tracking for Core Java Classes.

Activating object instance tracking increases the amount of information captured by the agents, resulting in additional overhead. AppDynamics recommends using object instance tracking only while troubleshooting potential memory leaks. It does not normally need to be enabled during normal operation.

> You need the **Configure Agent Properties** permission to set object instance tracking. You need the **Configure Memory Monitoring** permission to configure the custom classes to track. See Create and Manage Custom Roles.

Prerequisites for Object Instance Tracking

- Review Java Supported Environments for information about platform support for object instance tracking.
- For JVMs prior to Java 9, object instance tracking uses tools.jar. If your application runs with the JDK, tools.jar should be already available. If you are running with the JRE, you must add tools.jar to `<JRE_HOME>/lib/ext` and restart the JVM. You can find `tools.jar` in `<JAVA_HOME>`/lib /tools.jar. For Java 9 onwards JEP220 moved the capabilities needed by OIT to the Core Java runtime, therefore `tools.jar` is no longer necessary.
- In some cases, you might also need to copy `libattach.so` (Linux) or `attach.dll` (Windows) from your JDK to your JRE.
- Depending on the JDK version, you may also need to specify the classpath as shown below (along with other -jar options).

> `jdk.jcmd` is no longer a required module for the custom runtime. However, it is required for the proper functioning of object instance tracking.

Specify the Classpath

When using the JDK runtime environment, set the classpath using the `-classpath` option for the application. For example:

- On Windows:
  ```
  java -classpath <complete-path-to-tools.jar>;%CLASSPATH% -jar myApp.jar
  ```
- On Unix:
  ```
  java -Xbootclasspath/a:<complete-path-to-tools.jar> -jar myApp.jar
  ```

Enable Object Instance Tracking

To start an object instance tracking session, follow these steps:

1. In the left navigation pane, click **Tiers & Nodes**.
2. In the right pane, expand the tier node and open the Node Dashboard for the node on which you want to enable object instance tracking.
3. Click the Memory tab.
4. Click the Object Instance Tracking subtab.
5. Click **ON**.

Tracked classes now appear in the Object Instance Tracking table. You can drill down to the tracked classes to see details.

Track Specific Classes

For performance reasons, only the top 20 application classes and the top 20 system (core Java) classes in the heap are tracked automatically.

Use the Configure Custom Classes to Track option on the Object Instance Tracking subtab to specify instances of specific classes to track. Note that the classes you configure here are tracked only if their instance count is among the top 1000 instance counts in the JVM.

To track instances of custom classes:

1. On the Object Instance Tracking tab, click **Configure Custom Classes to Track.**
2. On the Instrumentation page that displays, click the tier you want to customize.

3. In the Object Instance Tracking section for the tier, click **Add**.
4. Enter the fully-qualified class name of the class to track and click **Save**.

The class you added is now tracked during object instance tracking sessions.

Custom Memory Structures for Java

This page describes how to monitor custom memory structures for Java.

AppDynamics automatically tracks long-lived Java collections (HashMap, ArrayList, and so on) with Automatic Leak Detection. To track specific classes, you can use the Custom Memory Structures capability in the Controller UI.

You can use this capability to monitor a custom cache or other structure that is not a Java collection. For example, you may have a custom cache or a third-party cache such as Ehcache. In a distributed environment, caching can easily become a prime source of memory leaks. In addition, custom memory structures may or may not contain collections of objects that would be tracked using automatic leak detection.

> Monitoring custom memory structures with the Java Agent can result in increased CPU utilization. AppDynamics recommends you enable memory structure monitoring on a short-term basis only while troubleshooting or in pre-production environments.

To configure custom memory structures, ensure custom memory structures are supported in your JVM environment. See JVM Support.

- To activate or deactivate object instance tracking, you need the Configure Agent Properties permission.
- To configure the custom classes to track, you need the Configure Memory Monitoring permission. See Create and Manage Custom Roles.

Custom Memory Structures and Memory Leaks

Typically custom memory structures are used as caching solutions. In a distributed environment, caching can easily become a source of memory leaks. AppDynamics helps you to manage and track memory statistics for these memory structures.

AppDynamics provide visibility into:

- Cache access for slow, very slow, and stalled business transactions.
- Usage statistics rolled up to the Business Transaction level.
- Keys being accessed.
- Deep size of internal cache structures.

Automatic Leak Detection Versus Monitoring Custom Memory Structures

Automatic leak detection captures memory usage data for all map and collection libraries in a JVM session. However, custom memory structures may not contain all collection objects. For example, you may have a custom cache or a third-party cache such as Ehcache for which you want to collect memory usage statistics.

Using custom memory structures, you can monitor any custom object created by the app and the size data can be traced across JVM restarts. Automatic leak detection is typically used to identify leaks, while custom memory structures are used to monitor large coarse-grained custom cache objects.

The following provides the workflow for configuring, monitoring, and troubleshooting custom memory structures. You must configure custom memory structures manually.

1. On the **Tiers & Nodes** dashboard, use the **Automatic Leak Detection, On Demand Capture Session** feature to determine which classes aren't being monitored, for example, custom or third-party caches such as EhCache.
2. Configure **Custom Memory Structures** and then restart the JVM if necessary.
3. Enter the fully-qualified classname on the Create New Instance Tracker window and click **Save**.
   AppDynamics automatically tracks long-lived Java collections (HashMap, ArrayList, and so on) with Automatic Leak Detection.
4. Turn on **Custom Memory Structures** monitoring to detect potential memory leaks in the custom memory structures you have configured.
5. Drill down into leaking memory structures for details that will help you determine where the leak is.

To identify custom memory structures:

1. Navigate to **Memory** > **Automatic Leak Detection** and click **On**.
2. Click **Start On Demand Capture Session** to capture information on which classes are accessing which collections objects. Use this information to identify custom memory structures.

AppDynamics captures the top 1000 classes, by instance count.

Identify Potential Memory Leaks

Start monitoring memory usage patterns for custom memory structures. An object is automatically marked as a potentially leaking object when it shows a positive and steep growth slope. The **Memory Leak** dashboard provides the following information:

- **Heap & Garbage Collection**—Provides heap and garbage collection metrics.
- **Automatic Leak Detection**—Provides memory usage data for all map and collection libraries in a JVM session.
- **Object Instance Tracking**—Provides tracking data for the top 20 application and top 20 system (core Java) classes in the heap.
- **Custom Memory Structures**—Allows you to track specific classes and monitor a custom cache or other structure that is not a Java collection.

The **Custom Memory Structures** dashboard provides the following information:

- **Class**—The name of the class or collection being monitored.
- **Deep Size (bytes)**—The upper boundary of memory available to the structure. The deep size is traced across JVM restarts
- **% of Current Used Heap**—The percentage of memory available for dynamic allocation.
- **Potentially Leaking**—Potentially leaking collections are marked as red. We recommend that you start a diagnostic session on potentially leaking objects.
- **JVM Start Time**—Custom Memory Structures are tracked across JVM restarts.
- **Status**—Indicates if a diagnostic session has been started on an object.
- **Deep Size**—A positive and steep growth slope indicates a potential memory leak.

After the potentially leaking collections are identified, start the diagnostic session.

Diagnose Memory Leaks

On the **Custom Memory Structures** dashboard, select the class name to monitor and click **Drill Down.**

Isolate Leaking Collections

Use **Content Inspection** to identify to which part of the application the collection belongs. It allows monitoring histograms of all the elements in a particular memory structure. Start a diagnostic session on the object and then follow these steps:

1. Select the Content Inspection tab.
2. Click **Start Content Summary Capture Session**.
3. Enter the session duration. Allow at least 1-2 minutes for the data to generate.
4. Click **Refresh** to retrieve the session data.
5. Click a snapshot to view the details about that specific content summary capture session.

Access Tracking

Use **Access Tracking** to view the actual code paths and business transactions accessing the memory structure. Start a diagnostic session on the object and then follow these steps:

1. Select the Access Tracking tab.
2. Select **Start Access Tracking Session**.
3. Enter the session duration. Allow at least 1-2 minutes for data generation.
4. Click **Refresh** to retrieve the session data.
5. Click a snapshot to view the details about that specific content summary capture session.

JVM Crash Guard

This page describes how to view and monitor JVM crashes with JVM Crash Guard. When a JVM crash occurs, you need to know as soon as possible. A JVM crash may be a sign of a severe runtime problem in an application and often calls for immediate remediation steps.

View JVM Crash Information

When a JVM crash occurs, a corresponding event type is generated in the Controller UI.

To analyze and troubleshoot the crash:

1. In the **Events** panel, double-click the JVM Crash of interest.
2. Examine any logs associated with the JVM Crash event. (The local log folder is determined by the type of JVM and how it is configured. For more information, refer to the documentation for the specific JVM.)

The JVM Crash panel also displays information about actions executed as a result of the crash. These are actions that you specify when creating a policy that is triggered by a JVM crash event. See Policies.

The JVM Crash event details include this information:

- Timestamp
- Crash reason
- Hostname IP address
- Process ID
- Application name
- Node name
- Tier name

In the JVM Crash Details tab, the **Crash Reason** details field (if available) indicates the root cause of the crash. For example, the field could contain `java.lang.OutOfMemoryError` or Segmentation Fault.

To facilitate the discovery and display of the reason for the JVM crash, JVM Crash Guard supports:

- Hotspot JVM error log analysis
- IBM JVM system dump log analysis
- Jrockit JVM error log analysis

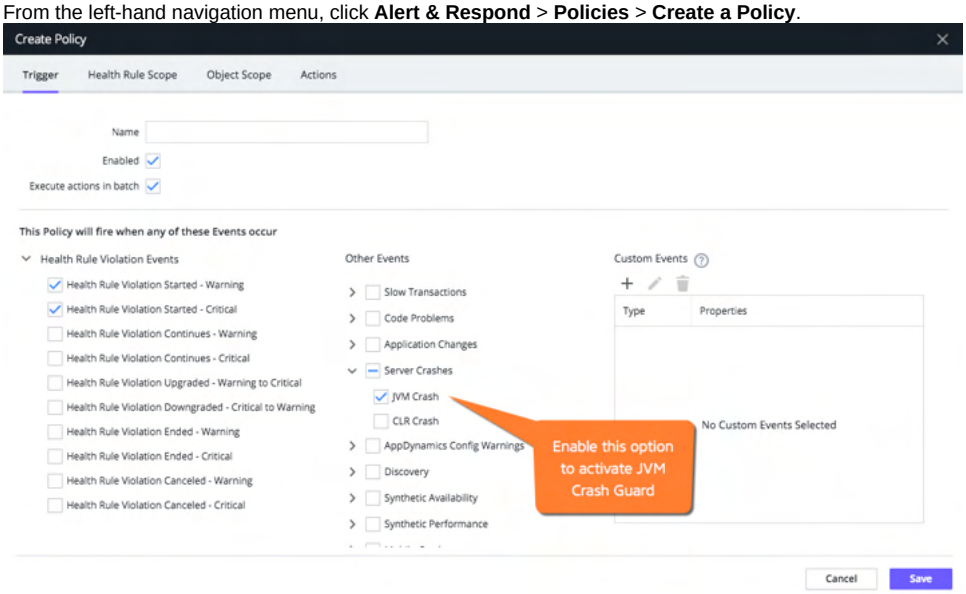
Enable Monitoring for JVM Crashes

Before you can monitor JVM crashes, you must:

- Install and enable a Machine Agent on the machine you want to monitor for JVM crashes. JVM Crash Guard works with the Machine Agent to trigger a policy when a JVM Crash event occurs.
- Ensure that the Machine Agent is running with the required privileges:
  - On Windows, the Machine Agent must run in Administrator root mode.
  - On Linux, JVM Crash Guard requires that the Machine Agent user be able to read all the processes in `/proc/*`. This may be the root user or another user with this privilege.
- Enable the JVM Crash Guard (it is disabled by default). Navigate to `extensions/CrashGuard/conf/crashGuardConfig.yml` and open the `crashGuardConfig.yml` file. Set `enabled` to `true`.

Once you have verified the requirements, follow these steps to create a policy for JVM crash events:

1. From the left-hand navigation menu, click **Alert & Respond** > **Policies** > **Create a Policy**.



2. In the **Other Events** section, expand the **Server Crashes** option and click **JVM Crash**. The JVM Crash event then becomes a trigger to fire a policy.
3. Proceed as usual to create the policy. See Policies.

> If an uninstrumented JVM crash occurs within less than a minute of a previous crash, it will not be reported by the Standalone Machine Agent. In some circumstances, the JVM may crash and then be restarted only to crash again within one minute. For this repetitive cycle crash and restart scenario, only the first JVM crash is reported by the agent.

Garbage Collection

This page describes how to monitor Garbage Collection for Java applications.

AppDynamics gathers Garbage Collection metrics and lets you analyze how periodic Garbage Collections affect the performance of your application. It is important to identify the impact of excessive Garbage Collection or memory-caused instability on the application. A typical Java application which runs on the Java Virtual Machine (JVM) creates objects such as strings, files, and arrays of primitives on the heap. The Java Garbage Collection is an automatic memory management process which finds and gets rid of the objects which are no longer used by the application.

The JVM periodically performs Garbage Collection to maximize available memory and the programmer need not explicitly mark the objects to be deleted. Garbage Collection requires a stop-the-world suspension of all application threads. This process affects the performance, especially for applications with large amounts of data, multiple threads, and high transaction rates.

See How to Master Your Java Memory for a review of how generational Garbage Collection works.

Before You Begin

If your application runs under JDK version 1.6.0_26-b03 and you have configured the monitored application to use G1GC, the Java Agent cannot capture memory statistics. To capture memory statistics, you can do any one of the following:

- Remove G1GC (`-XX:+UseG1GC`) from the application startup options, or
- Upgrade the JDK to >= 1.6.0_32.

In JVM >= 1.7, the agent attaches listeners to the Java event notification service to generate Garbage Collection metrics. In JVM < 1.7, the agent parses certain JVM log files to generate metrics, so you must verify that your JVM is generating the required logs. See Enable Log-based Garbage Collection.

Monitor Garbage Collection

The Java Agent reports certain Garbage Collection metrics at one-minute intervals. You can view these metrics on the Heap & Garbage Collection sub-tab of the Memory tab on the Node dashboard:

- **Heap utilization**: free, used, committed, and available.
  This is the most coarse-grained view of memory use.
- **Garbage Collection**: minor, major, and total on one timeline.
  This should give you some notion of the ratio of minor to major collections.
- **Minor Garbage Collection** on a timeline.
- **Major Garbage Collection** on a timeline.
- **Memory pool use**, including the use of all memory spaces: Young Generation, Old Generation, and PermGen.

For a finer-grained view of the impact of Garbage Collection on application performance, you can view Garbage Collection metrics in the Metric Browser. Navigate to **Application Infrastructure Performance > *tier* > JVM > Garbage Collection**.

In addition to the periodically-collected metrics, the Metric Browser shows the following metrics triggered by minor or major Garbage Collection events:

Metric	Triggering Event	Description
Allocated-Objects	Minor collection	The amount of memory allocated to the Young Generation. A high growth rate might indicate memory thrash. The allocation rate affects the frequency of minor collection events, which can impact application performance over time. The value for **Count** indicates how many collections were made.
Promoted-Objects	Major collection	The amount of memory in use for objects promoted from Young Gen space to Old Gen space. A high promotion rate is related to major collection events, which can have a significant impact on application performance due to the duration of a full major collection cycle. If the promotion rate is close to the allocation rate, this might indicate premature promotion. In this case, you might want to allocate more memory to Young Gen space. The value for **Count** indicates how many collections were made.
Freed Objects	Minor or Major collection	The amount of memory in use for objects being reclaimed in Young and Old space. In a normal system, the number of objects allocated and the free rate metrics should be roughly equal. The value for **Count** indicates how many collections were made.
LiveData	Major collection	The amount of memory in use that persists after major Garbage Collections. An upward slope in the size of live data indicates a possible memory leak.

Tune Garbage Collection in the JVM

After reviewing Garbage Collection diagnostic metrics, you can use these JVM arguments to tune space allocation in the JVM Garbage Collection memory pool. For example, you might want to increase the size of tenured space if your application needs to store many objects long term.

JVM Arguments	Meaning
`-Xms`	The amount of total memory allocated to young and old generation space.
`-XX:NewSize`	The amount of memory allocated to the young generation space.
`-XX:PermSize`	The amount of memory allocated to the permanent generation

Enable Log-based Garbage Collection

For applications running in JVMs < 1.7, Garbage Collection monitoring is based on periodically parsing certain Garbage Collection logs.

To set up Garbage Collection logging for your application, use these arguments:

```
-Xloggc:log-file-path
-XX:+Usecollector-type
-XX:+PrintGCDetails
```

The `log-file-path` option specifies where the log file is located. This table describes the possible values for collector types for the `-XX:+Use`*`collector-type`* option:

Collector Type	Effect
CMS Collector `-XX:+UseConcMarkSweepGC`	Good for applications that require low pause times and that can share resources with the garbage collector. Use the `-XX:ParallelCMSThreads=`<*n*> to set the number of threads to use.

Throughput or Parallel Collector
 	
`-XX:+UseParallelGC`
`-XX:+UseParallelOldGC`
	Can use multiple CPUs to speed up application throughput; good to use for work-intensive apps that can accept long pauses.

G1 Collector `-XX:+UseG1GC`	Available in Java 7 and designed to be a long term replacement for the CMS collector. This is a parallel, concurrent, and incrementally compacting low-pause collector.

To direct the Controller to display logged information, register the following node properties in the Controller UI:

```
enable-jmx-visibility=true
enable-log-based-gc=visiblity=true
```

To change the default log check interval, register the following node property:

```
logbased-visibility-log-check-interval-in-mills=1000
```

Specify Regular Expressions for Parsing Garbage Collection Logs

When you enable log-based Garbage Collection metrics, the agent uses built-in regular expressions to accommodate different JDK and Garbage Collection type combinations. If the built-in regular expressions don't return the Garbage Collection metrics for your system, you can register the following node properties to specify custom regular expressions for parsing the logs:

- `young-gc-custom-regex-1`
- `young-gc-custom-regex-2`
- `young-gc-custom-regex-3`
- `full-gc-custom-regex-1`

- `full-gc-custom-regex-2`
- `full-gc-custom-regex-3`

After you set a custom regular expression using a node property, the agent no longer uses any of the built-in regular expressions to parse the logs.

Monitor JMX

This page covers monitoring for Java Management Extensions (JMX). Java application environments usually support JMX or IBM Performance Monitoring Infrastructure (PMI). AppDynamics automatically discovers JMX and PMI attributes.

About JMX Monitoring Support

JMX uses objects called MBeans (Managed Beans) to expose data and resources from your application. In a typical application environment, there are three main layers that use JMX:

- JVMs provide built-in JMX instrumentation or platform-level MBeans that supply important metrics about the JVM.
- Application servers provide server or container-level MBeans that reveal metrics about the server.
- Applications often define custom MBeans that monitor application-level activity.

MBeans are typically grouped into domains to indicate where resources belong. Usually, in a JVM there are multiple domains. For example, for an application running on Apache Tomcat, there are *Catalina* and *Java.lang* domains. *Catalina* represents resources and MBeans relating to the Tomcat container, and *Java.lang* represents the same for the JVM Hotspot runtime. The application may have its own custom domains.

You can use MBean attributes to create persistent JMX metrics in AppDynamics as described here. In addition, you can import and export JMX metric configurations from one version or instance of AppDynamics to another.

See Writing PMI Applications Using the JMX Interface.

Requirements for JMX Monitoring

AppDynamics can capture MBean data if:

- The monitored system runs on Java >= 1.5.
- Each monitored Java process has JMX enabled. See JMX documentation.

MBean-Based Metrics

AppDynamics creates long-term metrics of the key MBean attributes that represent the health of the Java container. Depending on your application configuration, metrics may include:

- Session information such as the number of active and expired sessions, maximum active sessions, processing time, average and maximum alive times, and a session counter.
- Web container runtime metrics that represent the thread pool that services user requests. The metrics include pending requests and the number of current threads servicing requests. These metrics are related to Business Transaction metrics such as response time.
- Messaging metrics related to JMS destinations, including the number of current consumers and the number of current messages.
- JDBC connection pool metrics including current pool size and maximum pool size.

To view the JMX metrics discovered in a node, see the JMX tab on the Node Dashboard. For additional metrics not discovered automatically, you can confi gure JMX Metrics from MBeans yourself.

You can mask sensitive JMX and MBean data by setting the sensitive-data-filters property in the `app-agent-config.xml`. See Filter Sensitive Data.

View JMX Metric Data

You can view MBean-based metrics using the Node Dashboard and the Metric Browser. In addition, the MBean Browser enables you to view all the MBeans defined in the system.

To view JMX metrics in the Metrics Browser, click the **JMX** tab in the Node Dashboard. The JMX Metrics browser opens and displays the MBeans in a Metric Tree.

You can perform all the operations that are provided by the Metric Browser. See Metric Browser.

View Trending MBeans with Live Graphs

You can monitor the trend of a particular MBean attribute over time using the **Live Graph**.

To view live graphs:

1. In the Node Dashboard, click the JMX tab and then the MBean Browser sub-tab.
2. Select the domain for which you want to monitor MBeans.
3. In the domain tree, expand the domains to find and then select a MBean.
4. Expand the **Attributes** section and then choose an attribute of the MBean.
5. Click **Start Live Graph for Attribute** and then click **Start Live Graph**. The runtime values appear.
6. Select an attribute and click **Live Graph for Attribute** to see a larger view of a particular graph.

Working with MBean Values

When troubleshooting or monitoring a Java-based system, you may want to change the values of composite MBeans and execute MBean methods.

To change the value of an MBean attribute or invoke operation, you need the **Set JMX MBean Attributes and Invoke Operations** permission for the application. See Application Permissions.

To view and edit MBean attribute values:

1. From the **JMX** panel, select **MBean Browser**.
2. In the Domain tree, locate the MBean that interests you.
3. Select an editable attribute, one that has **Yes** in the **Editable** column, and then click **View/Edit Attribute**.
4. In the **MBean Attribute** panel that displays, you see the current value of the MBean Attribute.
5. You can change the value of an editable MBean Attribute by entering a new value in the **Value** field.

Invoke MBean Operations

Using the JMX viewer, you can invoke an MBean operation with standard Java language strings for parameters, and view return values.

To perform this action, a user must have the **Set JMX MBean Attributes and Invoke Operations** permission for the application

To invoke MBean operations:

1. Open the **MBean Browser**.
2. In the Domain tree, locate the MBean that interests you.
3. Open the **Operations** pane, scroll to find the operation that interests you, and double-click **Invoke Action**.
4. Enter the parameter values for the operation and then click **Invoke**. Scalar values for constructors of complex types, such as getMBeanInfo(java.util.Locale) allow you to enter "en-us".

A message appears indicating that the operation is in progress and the number of seconds elapsed. When the operation completes, the results display. The method return result from an invocation can also be a complex attribute. In this case, the name, description, type, and editable attributes of the method are also displayed in the MBean Operation Result area.

Available JMX Metrics

Java Management Extensions (JMX) is a public specification for monitoring and managing Java applications. Through JMX, AppDynamics can access Java class properties that collect management data, such as the resources your application is consuming.

For information on the specific metrics available for your environment, see the documentation provided by your vendor:

- Apache ActiveMQ, see ActiveMQ MBeans Reference, in the ActiveMQ Features documentation
- Apache Kafka, see Default JMX Metrics for Apache Kafka Backends
- Apache Solr, see Quick Demo in the Solr JMX documentation
- Apache Tomcat, see the descriptions of JMX MBeans for Catalina in the `mbeans-descriptor.xml` file for each package
- Cassandra, see Cassandra Metrics
- Coherence, see the Coherence MBeans Reference in Appendix A of the Coherence Management Guide
- GlassFish, see the Oracle Sun Glassfish Administration Guide where you can find a Metrics Information Reference
- HornetQ, see Using Management Via JMX
- JBoss, see An Introduction to JMX
- Kafka Broker, see Monitoring Kafka
- Oracle WebLogic Server, see Understanding JMX
- WebSphere PMI, see PMI data organization in the IBM Websphere Application documentation

Adding JMX Metrics

In addition to the preconfigured metrics, you can define a new persistent metric using a JMX Metric Rule that maps a set of attributes from one or more MBeans. See Configure JMX Metrics from MBeans.

Extending Monitoring with the Agent API

The Application Server Agent API lets you access metrics that are not supported by default or by MBeans. You can use the API to:

- Inject custom events and report on them
- Create and report on new metrics
- Correlate distributed transactions when using protocols that AppDynamics does not support

To learn more about the Application Server Agent API, see the Javadoc included with the Java Agent software at this location:

`<agent_home>/sdk/docs/index.html`

Default JMX Metrics for Apache Kafka Backends

This page describes default metrics for Apache Kafka Backends. The Java Agent includes rules for key metrics exposed by Apache Kafka producers and consumers. To monitor JMX metrics not collected by default, you can use the MBean browser to select the Kafka JMX metric and create a rule for it.

Kafka Producer JMX Metrics

- Response rate: the rate at which the producer receives responses from brokers
- Request rate: the rate at which producers send request data to brokers
- Request latency average: average time between the producer's execution of `KafkaProducer.send()` and when it receives a response from the broker
- Outgoing byte rate: producer network throughput
- IO wait time: percentage of time the CPU is idle and there is at least one I/O operation in progress
- Record error rate: average record sends per second that result in errors
- Waiting threads: number of user threads blocked waiting for buffer memory to enqueue their records
- Requests in flight: current number of outstanding requests awaiting a response
- Network IO rate: average number per second of network operations, reads or writes, on all connections

Kafka Consumer JMX Metrics

- Records lag max: maximum lag in terms of number of records for any partition within the timeframe
- Bytes consumed rate: average number of bytes consumed per second
- Fetch rate: number of fetch requests per second
- Records consumed rate: average number of records consumed per second
- Fetch latency max: maximum time this is taken for any fetch request

Kafka Server JMX Metrics

- `broker-request-total-time-ms`: Total end-to-end time in milliseconds.
- `broker-request-send-response-ms`: Responses dequeued are sent remotely through a non-blocking IO. The time between dequeuing the response and completing send is indicated by this metric.
- `broker-request-response-queue-ms`: Responses too are added to a queue. There is one queue per network processor. Network processor dequeues this response and sends it back. Time spent waiting in this queue is indicated by this metric.
- `broker-request-remote-time-ms`: If the request needs remote processing, the time spent in remote processing is indicated by this metric. For example, for the producer request, if acks is set to -1, the request is not completed until the acknowledgment is received from the followers. Time spent waiting for the followers in indicated by this metric. A fetch request can also be delayed if there is not enough data to fetch. That time too is accounted for by this metric.
- `broker-request-processing-ms`: The request is then processed by the KafkaAPI. The time spent in the processing is indicated by this metric. If this metric is high, debug the relevant request handler.
- `broker-request-queue-time-ms`: The request is added to a common queue. The items in the queue are processed by the request handler threads. The number of these handler threads can be configured through num.io.threads parameter. The average time, a request spends in this queue, is indicated by this metric. If this metric is high, increase the handler threads.

Monitor .NET Nodes

This page describes specific considerations for monitoring .NET applications and how to fine-tune the configuration for .NET frameworks and technologies. These include, for example, monitoring and configuration pages for your IIS applications, Windows services, and standalone applications.

Once you instrument your .NET application environment with the AppDynamics .NET Agent, you can start monitoring .NET application performance. See AppDynamics Essentials.

End-to-End APM for .NET

- Getting Started
- .NET Supported Environments
- .NET Agent
- Monitor CLRs
- Monitor Windows Hardware Resources

Monitor Windows Hardware Resources

This page describes how to monitor Windows hardware resources with the .NET Machine Agent. The AppDynamics .NET Agent includes an embedded .NET Machine Agent that runs as part of the AppDynamics.Agent.Coordinator service. The .NET Machine Agent regularly gathers system performance data metrics such as:

- CPU activity
- Memory usage
- Disk reads and writes
- Network traffic
- Percent free disk space and megabytes free

You can view Windows Hardware Resource metrics in the Metric Browser. When IIS is installed on the machine, the .NET Machine Agent also reports IIS, ASP.NET, and ASP.NET Application metrics. See Monitor IIS. The .NET Machine Agent also reports CLR metrics, which are based on performance counters. See Monitor CLRs.

For applications running on the Windows Azure platform, AppDynamics disables these hardware resource monitoring and machine agent features:

- CLR crash reporting
- Machine snapshots
- Performance counter metrics

Manage the .NET Machine Agent

You can manage the .NET Machine Agent in the Controller under **Settings** > **AppDynamics Agents** on the Machine Agents tab.

- To enable or disable all the machine agents, including the Standalone Machine Agent, for a business application, click **Enable Agents** or **Disable Agents**.
- To enable or disable an individual .NET Machine Agent, right-click the agent and click **Enable Selected Machine Agent** or **Disable Selected Machine Agent**.

  > Disabled agents maintain a 'heartbeat' connection to the Controller so you can enable it again. The agent persists its enabled/disabled state even after a restarted.

- To reset a .NET Machine Agent, right-click the agent and click **Reset Selected Machine Agent**. The reset operation purges all existing metrics for the agent. The agent restarts and begins gathering metrics again.

Machine Agent Tier

Immediately after you install and configure the .NET Agent, the .NET Machine Agent registers with the Controller and starts reporting performance data.

Frequently the machine agent metrics reach the Controller before the app agent has had time to instrument and register IIS applications, Windows services, or standalone applications. If there are no application tiers, then the machine agent registers as the Machine Agent tier.

Once the app agent begins reporting metrics for configured application tiers, the .NET Machine Agent reports to the application tiers and stops sending data to the Machine Agent tier. If you do not instrument any IIS applications, Windows services, or standalone applications on the server, the .NET Machine Agent always reports to the Machine Agent tier.

If you install the Standalone Machine Agent, a Java application, in a .NET environment, then you can add custom monitors and extensions such as the HTTP listener. See Configure the .NET Machine Agent.

Machine Snapshots for .NET

Sometimes environmental conditions cause trouble on the machines where your application runs. Issues that occur outside monitored application code don't show up on transaction snapshots. Machine snapshots provide critical details about CPU usage, memory usage, and the IIS queue on a server at a specific moment in time. Use the data in machine snapshots to uncover and resolve environmental problems.

AppDynamics generates machine snapshots to capture the state of a server at a specific moment in time. The machine snapshots show the processes running on the machine, the IIS application pool activity, and any related transaction snapshots. By default the .NET Machine Agent takes machine snapshots under these conditions:

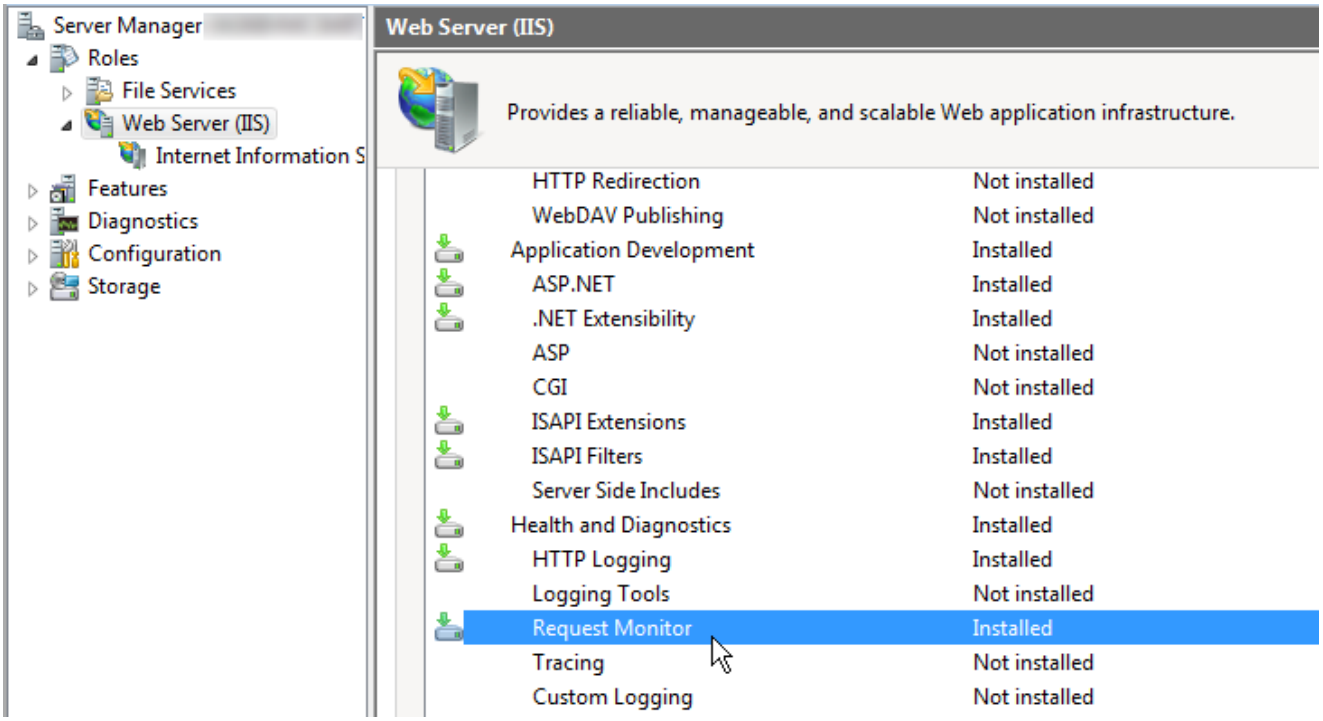
- Periodic collection: The agent takes one snapshot every 10 minutes.
- Breached thresholds: The .NET Machine Agent takes samples of machine statistics every 10 seconds within a 10-minute window. For each sample, the agent checks the CPU percent usage, the memory percent usage, and the oldest item in the IIS application pool queue. The agent flags a sample as a violation when the current usage meets or exceeds one of these thresholds:
  - CPU at 80% or higher
  - Memory at 80% or higher
  - IIS application pool queue item older than 100 milliseconds

  The agent takes a snapshot when it identifies 6 violations of a single type, such as CPU usage, within the window. The agent only takes one snapshot per window for breached thresholds.

To customize the periodic collection or threshold settings, see Configure Machine Snapshots for .NET.

Before Starting

The .NET Machine Agent requires >= IIS 7 to return machine snapshot data for IIS application pools. Additionally, you must enable the **Request Monitor** for the **IIS Health Monitoring** feature.



To enable Request Monitor from the Windows PowerShell command line, launch PowerShell as an administrator and run this command:

```
Install-WindowsFeature Web-Request-Monitor
```

Work With Machine Snapshots

The Machine Snapshot tab on the Application dashboard lists all the snapshots for the selected time range. Entries for individual snapshots show:

- Time the .NET Machine Agent took the snapshot
- Machine name for the snapshot
- Snapshot trigger, either periodic snapshot collection or threshold exceeded
- Percent CPU usage

- Percent memory usage

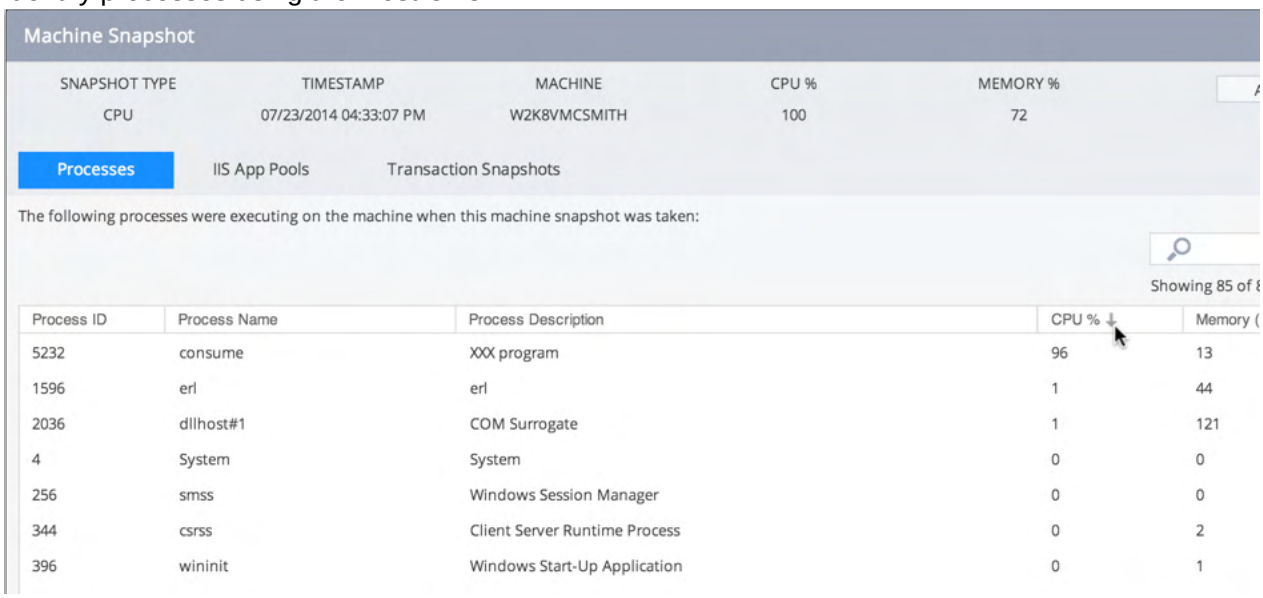
Use **Filters** to limit the snapshot list to a specific snapshot trigger:

- Periodic collection
- Memory events
- CPU events
- IIS events
- Machines where the agent took snapshots
- Archived snapshots

Double-click a snapshot to open the Machine Snapshot panel:

- The Processes tab shows information similar to that from the Windows Task Manager, including:
  - Process ID
  - Process name
  - Process description
  - Percent CPU used by the process
  - Percent memory used by the process

  Click a column heading to sort by the column. For example, click **CPU %** and sort descending to identify processes using the most CPU.



- The IIS App Pools tab shows information on the active IIS application pools:
  - Application pool name
  - Arrivals to the queue per second
  - Queue requests processed per second
  - Age of the oldest item in the queue

- The Transaction Snapshots tab displays all transaction snapshots involving the current machine for five minutes prior to and five minutes after the machine snapshot. This demonstrates how current environmental factors are affecting your business transaction performance. For example, when CPU usage is high, you may discover slow and stalled transactions. Double-click a transaction snapshot to open it.

Monitor CLRs

This page describes how to monitor CLRs with the .NET Machine Agent. The AppDynamics .NET Agent includes the .NET Machine Agent that runs as part of the AppDynamics.Agent.Coordinator service. The .NET Machine Agent regularly gathers CLR performance data and reports it back to the Controller.

CLR Events

The .NET Machine Agent monitors for CLR shutdown and restart events:

- The agent reports an App Server Restart event and indicates if the restart was graceful or not.
- For non-graceful shutdown, the agent reports a CLR Crash if it detects one.

> The default exit code for a graceful shutdown is "0". If your Windows service or standalone application uses a different exit code, see "Profiler - Successful Exit Code Element" on .NET Agent Configuration Properties.

CLR Metrics

CLR metrics provide insight into how the .NET runtime is performing. The AppDynamics preconfigured CLR metrics include:

- .NET CLR memory usage
- Total classes loaded and how many are currently loaded
- Garbage collection time spent, and detailed metrics about GC memory pools and caching
- Locks and thread usage
- Memory heap and non-heap usage, including the large object heap
- Percent CPU process usage

CLR Health Alerts

You can set up health rules based on infrastructure metrics. Once you have a health rule, you can create specific policies based on health rule violations. One type of response to a health rule violation is an alert.

In addition to the default metrics, you may be interested in additional metrics. You can specify additional performance counters to be reported by the .NET Machine Agent. See Manage Windows Performance Metrics.

Monitor CLR Crashes

This page describes monitoring for CLR crash events. The .NET Machine Agent monitors Windows for non-graceful CLR shutdowns on IIS, Windows services, and standalone applications. The agent raises CLR crash events in the Controller when such crashes occur. Use a policy to alert responsible parties when the agent reports a CLR crash.

Monitor for CLR Crash Events

Create a policy that sends notifications when a CLR crash event occurs.

1. Verify your **Email / SMS Configuration**. See Enable an Email Server.
2. Create a notification action configured to alert the parties who respond to CLR crashes. See Notification Actions.
3. Create a policy to trigger on **CLR Crash** under **Other Events** on the Create Policy panel. For the action, choose the notification action you created in step 2. See Configure Policies.

After you create the policy, the Controller sends a notification when the .NET Machine Agent raises CLR crash events.

Analyze and Respond to CLR Crashes

When a CLR crashes, the .NET Machine Agent raises an event in the Controller. The events page on the application dashboard displays the number of server crash events, including CLR crashes, during the selected time range.

1. Click the **Server Crashes** link in the application dashboard to display a list of all server crash events during the selected time range.
2. Optionally filter on the **CLR Crash** event type.
   Entries for individual CLR crashes display information about the crash event, including the process ID and the Windows event log ID.

   > After a CLR crash, IIS automatically tries to restart the CLR, so one issue frequently causes multiple CLR crash events.

3. Double-click a CLR Crash type event to display more information in the **CLR Crash** panel:
   - The Summary tab shows the Windows process ID of the w3wp process that crashed and the Windows event log ID for the crash. The summary also includes the affected tier and node names.

     > If multiple tiers have nodes on the same machine, you may see more than one tier in the CLR Crash window.

   - The Details tab displays essential information about the crash.
   - The Actions Executed tab shows any actions the event triggers. See Monitor for CLR crash events.
   - Read and add new comments on the Comments tab. For example, you can add a comment to the CLR crash event to notify colleagues that of the issue.

Disable CLR Crash Event Reporting

The .NET Machine Agent enables CLR crash event reporting by default. To disable it:

1. Edit the agent `config.xml` as an administrator. See Where to Configure App Agent Properties.
2. Set `enabled="false"` for the CLR Crash Reporting element. See CLR Crash Reporting Element.
3. Restart the AppDynamics.Agent.Coordinator service.

Crash Events Monitored by the .NET Machine Agent

The .NET Machine Agent listens on these Windows event logs for crash events:

- Application Log
- System Log

The agent listens for events logged at event-level `Warning` and higher on these sources:

- Application Error
- .NET Runtime
- WAS (Windows Process Activation Service)

The agent listens for events logged at event-level `Information` and higher on this source:

- Windows Error Reporting

The agent reports CLR crashes as events to the Controller:

- All w3wp process crashes
- Instrumented Windows service crashes
- Instrumented standalone application crashes

The Controller treats all events as `Warning` severity, which is not directly related to the Windows event level.

Object Instance Tracking for .NET

This page describes how to enable object instance tracking on a node. When you enable object instance tracking for a node, AppDynamics analyzes the heap to identify classes with the most instances on the heap. AppDynamics tracks the top 20 .NET framework classes and the top 20 application classes based upon the number of instances. Use object instance tracking to identify memory usage trends in classes with large numbers of instances.

Permissions

To enable object instance tracking, you need one of these permissions:

- Configure Memory Monitoring
  or
- Configure Agent Properties

See Create and Manage Custom Roles.

Before You Begin

- The .NET Agent only supports object instance tracking for >= .NET 4.
- The .NET Agent Coordinator Service must be the same architecture as the monitored application. For example, a 64-bit coordinator with a 64-bit IIS application.

Enable Object Instance Tracking on a Node

1. From the node dashboard, click the Memory tab.
2. Click the Object Instance Tracking subtab.
3. Click **ON**.
   Once the agent completes the heap analysis, AppDynamics begins to track the top 20 application classes and the top 20 system (Core .NET) classes in the heap.

   When you enable object instance tracking, the .NET application pauses during heap analysis and cannot process requests. Enable object instance tracking while you diagnose memory issues; and turn it off when you finish troubleshooting.

Identify Memory Usage Problems

Use these guidelines to identify memory usage problems:

- The Controller resolution for object instances is one minute, but the .NET Agent agent only sends data every ten minutes, so .NET memory appears as a series of peaks over time.
- It is normal for the Controller to display 0 for the Current Instance Count and Shallow Size in between instance count collection times.
- Hover over a peak to display information about the instance count.
- Look for trends where the peaks increase in size from left to right which may indicate a memory leak.

Track Object Instances for Custom Classes

If you want to track a class that doesn't appear in the top 20 on the Object Instance Tracking tab, you can configure a specific class to track.

1. Navigate to **Object Instance Tracking** > **Configure Custom Classes to Track** > **Configure Instrumentation** > **Memory Monitoring**.
2. Click **Add**. The Create New Instance Tracker panel opens.
3. Leave **Enabled** checked.
4. Enter the fully qualified class name for the instance to track.
5. Click **Save**.

Monitor IIS

This page describes how AppDynamics monitors Internet Information Services (IIS). The AppDynamics .NET Agent includes the .NET Machine Agent that runs as part of the AppDynamics.Agent.Coordinator service. The .NET Machine Agent regularly gathers IIS performance data and reports it back to the Controller as metrics.

IIS Events

The .NET Machine Agent monitors IIS for shutdown and restart events:

- The agent reports an App Server Restart event and indicates if the restart was graceful or not.
- For non-graceful shutdown, the agent reports a CLR Crash if it detects one.

Default IIS Metrics for .NET

You must install IIS on the machine to view the metrics for IIS, ASP.NET, and ASP.NET applications. The .NET Machine Agent uses Microsoft Windows Performance Counters to gather IIS metrics. In the Controller, you can view preconfigured metrics for IIS in the Metric Browser.

IIS Metrics

From the Metric Browser:

- To view the IIS metrics for a tier, expand **Application Infrastructure Performance** > **<Tier>** > **IIS**.
- To View the IIS metrics for a node, expand **Application Infrastructure Performance** > **<Tier>** > **Individual Nodes** > **<Node>** > **IIS**.

AppDynamics reports each metric for the entire tier, each individual application pool, and each individual node as follows:

- **Application Infrastructure Performance** > ***<Tier>*** > **IIS** = combined for all IIS processes in all Application Pools for this tier
- **Application Infrastructure Performance** > ***<Tier>*** > **Application Pools** > **<application pool name>** = combined for all processes in this specific Application Pool
- **Application Infrastructure Performance** > ***<Tier>*** > **Individual Nodes** > **<Node>** = metrics for the specific node

ASP.NET Metrics

To view the ASP.NET metrics in the Metric Browser, expand **Application Infrastructure Performance** > **<Node>** > **ASP.NET**.

AppDynamics reports these ASP.NET metrics:

- Application Restarts
- Applications Running
- Requests Disconnected
- Requests Queued
- Requests Rejected
- Request Wait Time
- Worker Process Restarts

ASP.NET Application Metrics

To view the ASP.NET Application metrics in the Metric Browser, expand **Application Infrastructure Performance** > ***<Node>*** > **ASP.NET Applications**.

AppDynamics reports these ASP.NET Application metrics:

- Anonymous Requests
- Anonymous Requests/sec
- Cache Total Entries
- Cache Total Hit Ratio
- Cache Total Turnover Rate
- Cache API Entries
- Cache API Hit Ratio
- Cache API Turnover Rate
- Errors Unhandled During Execution/sec
- Errors Total/sec
- Errors During Preprocessing
- Errors During Compilation
- Errors During Execution
- Errors Unhandled During Execution
- Errors Unhandled During Execution/sec
- Errors Total
- Errors Total/sec
- Output Cache Entries
- Output Cache Hit Ratio
- Output Cache Turnover Rate

- Pipeline Instance Count
- Requests Executing
- Requests Failed
- Requests In Application Queue
- Requests Not Found
- Requests Not Authorized
- Requests Succeeded
- Requests Timed Out
- Requests Total
- Requests/sec
- Session State Server Connections Total
- Session SQL Server Connections Total
- Sessions Active
- Sessions Abandoned
- Sessions Timed Out
- Sessions Total
- Transactions Aborted
- Transactions Committed
- Transactions Pending
- Transactions Total
- Transactions/sec

Monitor IIS Application Pools

You can monitor the health of IIS application pools for the instrumented .NET nodes in a tier. You can view the information by application pool, machine, and process IDs.

This view enables you to visualize key performance indicators for your infrastructure, such as node health and last CLR restart time.

To view the IIS application pools:

1. In the left navigation pane, click **Tiers & Nodes** > **Nodes**.
2. In the Show Data dropdown, select **IIS App Pools**.

If a machine or application pool name is not available for a .NET node, the .NET Agent creates the "Unknown App Pool" / "Unknown Machine" grouping.

Monitor Node.js Processes

This page describes how to monitor Node.js processes. The AppDynamics Node.js Agent helps you monitor Node.js applications in production to determine which applications are slower than normal or producing errors. It also provides tools for troubleshooting application problems so that you can take action before your users experience poor performance.

To access the data, log in to the Controller from a Web browser using your AppDynamics credentials. You can also access the data programmatically using the AppDynamics REST API.

Process Snapshots

The node dashboard for Node.js is similar to the node dashboards for other app agents except that it also includes a Process Snapshots tab, which you use to access process snapshots for the Node.js process.

In a single-threaded model, such as Node.js, one slow function forces other functions to wait. You can monitor Node.js processes using lists of process snapshots to identify which functions have high CPU times and which consume a lot of memory. From the list, you can select and examine process snapshots to identify exactly which functions in your code are blocking the CPU or leaking memory.

A process snapshot describes an instance of a CPU process on an instrumented Node.js node. It generates a process-wide call graph for a CPU process over a configurable time range. Process snapshots are independent of any running business transactions.

You can monitor process snapshots at the tier level or the node level.

Customizations

The Node.js Agent collects several metrics that do not appear in standard dashboards. You can see all the metrics in the Metric Browser, including special metrics related to Node.js processes in **Application Infrastructure Performance** > **Tier** > **Node.js.**

You can create custom dashboards that display any of these metrics using a variety of widgets to create a visual display of information customized for specific types of users in your organization: executives, ops, developers, QA and so forth.

You can also create health rules that stipulate certain levels of acceptable performance and then create policies that automatically send an alert or take an automated action when those rules are violated.

Object Instance Tracking for Node.js

This page describes how to enable object instance tracking for Node.js. The Node.js Agent provides object instance tracking (OIT) for monitoring memory usage. When you enable object instance tracking for a node, the Node.js Agent attempts to identify the object types with the most instances on the heap. The agent tracks the top 20 object types based upon the number of instances. Use object instance tracking to identify memory usage trends in objects with large numbers of instances.

Enable Object Instance Tracking on a Node

1. In the left navigation pane, click **Tiers & Nodes**.
2. In the **Tiers & Nodes** panel navigate to and select the node that you want to track.
3. Double-click or click **View Dashboard**.
4. In the node dashboard, click the Memory tab.
5. Click the **Object Instance Tracking** subtab.
6. Click **On** to start tracking. After completing the heap analysis, the agent begins tracking the top 20 object types in the heap.
7. When you are finished tracking, click **Off** because tracking can slow down the Node.js process. Use OIT for only for brief periods.

Although the Node.js OIT UI displays "Class" and "Classes" throughout, the Node.js Agent tracks object types, not classes. Substitute "object" for "class" wherever you see "class" for this feature.

Identify Memory Usage Problems

Use these guidelines to identify memory usage problems:

- The Node.js Agent attempts to report once per minute, but on a heavily loaded, memory-saturated application it could be less often.
- It is normal for the controller to display 0 for the Current Instance Count and Shallow Size between instance count collection times.
- Hover over a peak in the graph to display information about the instance count.
- Look for trends where the peaks increase in size from left to right. These may indicate a memory leak.

Track Object Instances for Custom Object Types

If you want to track an object that does not appear in the top 20 on the Object Instance Tracking tab, you can configure a specific object to track.

1. In the Object Instance Tracking tab, click **Configure Custom Classes to Track.**
2. In the **Define Custom Classes to Track** panel, click **Add**. The **Create New Instance Tracker** panel opens.
3. Leave **Enabled** checked.
4. Enter the object for the instance to track.
5. Click **Save**. The configured object is added to the objects to track when OIT is on.

Examine OIT Metrics in the Metric Browser

When enabled, OIT metrics are also reported under the **Application Infrastructure Performance** > ***<tier>*** > **Individual Nodes** > ***<node>*** > **Object Instance Tracking** branch in the Metric Browser.

Node.js Metrics

This page describes Node.js process metrics that are specific to the Node.js Agent.

You can view the metrics under these paths in the Metric Browser navigation tree:

- By tier at **Application Infrastructure Performance** > ***<TierName>*** > **Node.js**
- By node at **Application Infrastructure Performance** > ***<TierName>*** > **Individual Nodes** > **Nodejs_*<NodeName>*** > **Node.js**

CPU Usage Metric

- CPU Usage: % busy

Memory Metrics

Garbage Collection

- Full GC Per Min: Number of full garbage collection cycles per minute that the V8 JavaScript runtime has performed in the selected time range.
- Incremental GC Per Min: Number of incremental garbage collection cycles per minute that the V8 JavaScript runtime has performed in the selected time range.

Memory

- Heap size changed: Total amount of memory reclaimed by the full and incremental garbage collection cycles (as a percentage) in the selected time range.
- Current usage (V8 heap used in MB): Total size of the heap at the current time point. This reports how much memory the node process is using for data storage.
- RSS: Resident Set Size of the Node process. This reports the amount of memory (heap and stack) allocated for the process and in RAM, not swapped out.

I/O Metrics

Disk

- KB read per second: KB read from disk per second for the selected time range.
- KB written per second: KB written to disk per second for the selected time range.

Network

- Incoming: KB/sec received for the selected time range.
- Outgoing: KB/sec sent for the selected time range.

Socket.io Metrics

- Number of Connections: Number of currently open Socket.IO connections.
- Total Number of Connections: Number of connections that have been opened since the application started.
- Number of Messages Sent/Received: Number of messages that have been exchanged between the application and all connected Socket.IO clients.
- Size of Messages Sent/Received: Average size, in characters, of the messages exchanged. The underlying transport is text only, so non-string messages are JSON.serialized to determine their size.

Event Loop Metrics

- Average IO Time: Average number of milliseconds per event loop tick spent processing IO callbacks.
- Average Tick Length: Average amount of time between event loop ticks.
- Maximum Tick Length: Shortest amount of time between event loop ticks.
- Minimum Tick Length: Longest amount of time between event loop ticks.
- Tick Count: Number of times the event loop was ticked.

N|Solid Monitoring Data

This page describes available metrics and data for N|Solid monitoring. N|Solid is an enterprise-grade Node.js runtime produced by Nodesource that provides additional performance metrics about Node.js processes.

Metrics

When the Node.js Agent is installed on an N|Solid runtime, it collects these additional metrics and makes them available in the Controller so you can use them to diagnose latency issues and memory leaks.

You can view the metrics under these paths in the metric browser navigation tree, by node at **Application Infrastructure Performance** > ***<TierName>*** > **Individual Nodes** > **Nodejs_*<NodeName>*** > **Node.js**.

- NSolid -> 1-minute, 5-minute, and 15-minute load average
- NSolid -> System Uptime & Process uptime (ms)
- NSolid -> Memory -> JS heap -> Total Usage (MB)
    - The amount of the heap that is being used by JavaScript.
- NSolid -> Memory -> V8 heap -> Total Available (MB)
    - Total memory available to the V8 heap.
- NSolid -> Memory -> Total Available (MB)
    - Total memory available to the process.
- NSolid -> Memory -> Total Size (MB)
    - Total memory used by the process.
- NSolid -> Event Loop -> Active Handles:
    - The number of active handles the event loop will process. Handles tend to be longer-lived larger-scale asynchronous operations, such as open sockets or timers. An uptick in this metric could indicate a possible memory leak.
- NSolid -> Event Loop -> Active Requests:
    - The number of active requests the event loop will process. Requests tend to be shorter-lived smaller-scale operations such as writing to file handles and other file-related operations. This metric provides additional insight into load characteristics.
- NSolid -> Idle Percent:
    - Time spent waiting for I/O and not running Javascript.
- NSolid -> Estimated Lag
    - Average amount of time a I/O response may have to wait before being processed.
- NSolid -> Loops Per Second
    - Number of executions per second of the libuv event loop.
- NSolid -> Average Tasks:
    - Number of asynchronous Javascript tasks per turn of the loop.
- NSolid -> Total Count:
    - Total number of event loop turns.

Process Snapshots Data

You can view details on asynchronous activities in the Node.js event loop for N|Solid processes in **Process Snapshots** > **Async Activity**. These event loop activity categories display in Async Activity:

- Active Handles: active handles that the event loop will process
- Active Requests: the number of active requests the event loop will process
- Pending: lower level asynchronous activity

This information displays for each property:

- Type: The property type (e.g., TCP socket connection, setTimeout)
- Location: The source location associated with the activity, when available

Remote Services

This page describes the monitoring information available for remote services, also known as backends.

What is a Remote Service?

A remote service is a process that resides outside of the application server and provides a service to the application. An example of a remote service is a web service, message queue, or caching server.

AppDynamics automatically detects many common types of remote services when instrumented nodes make outbound requests. For more details on backend support by app agent type, refer to the supported environments page for your agent type under Install App Server Agents, such as Java Supported Environments.

To monitor call performance to a service, first make sure it shows up in the remote services list, which you can access by clicking the **Remote Services** link in the Applications menu. If a service you expect to see listed does not appear in the list, make sure the backend detection configuration is configured appropriately for your environment, as described in Backend Detection Rules.

Monitoring Information for Remote Services

AppDynamics monitors the overall performance of calls to a remote service, as well as the performance of those calls from within specific business transactions.

Metrics for remote services are presented in the Controller UI in the following locations:

- Business transaction metrics: The Transaction flow map shows the metrics for a specific business transaction for a specific service
- Tier metrics: The Tier flow map shows the metrics for all calls from a tier to the specified service
- Remote service metrics: The Application flow map and the Remote Services Dashboard show the overall remote service metrics across the application (all business transactions)

Remote Services Health Rules

You can create health rules to monitor the parameters that represent the normal or expected operations for your remote service. The parameters rely on metric values, for example, the average response time. When the performance of a remote service violates the conditions set by a health rule, a health rule is violated. Alerts notify you of any violation and initiate remediation actions to mitigate the violation event. It is important that you configure alerts appropriately to ensure that you do not miss any alerts or receive false alerts. Alert Sensitivity Tuning (AST) helps you configure alerts with appropriate sensitivity. AST provides historical data for the metric or the baseline being configured and hence helps you visualize the impact of the alerting configuration. To create a health rule to monitor remote service parameters and fine-tune the sensitivity of the health rule using AST, see Create a Health Rule and Fine-tune Metric Evaluation.

View Remote Service Performance on Flow Maps

Remote services detected during the specified time window appear on the Application Dashboard flow map. You can view the detected services in the context of the entire application's transaction flow. The application flow map displays calls per minute and average response time for calls made to remote services. These metrics include all calls made from a specific tier to a service across all business transactions. The tier and node flow maps display the same metric in their respective contexts.

The detected remote services show up on the Tier dashboard flow map. You can view the detected services in the context of the traffic on this specific tier.

For business transactions involving calls to remote services, the services appear on the Business Transaction dashboard flow map. You can view the detected services in the context of the traffic for this specific business transaction. The transaction flow map shows the average time spent in remote service calls for the business transaction. See Flow Maps.

View Discovered Remote Services

The Remote Services list shows all detected remote services along with key performance indicators. Services that are not active are removed after a configurable time period. See Stale Remote Service Removal.

From the Remote Services list, you can select a service and click **View Dashboard** to see the Remote Service Dashboard. The dashboard displays a Database Flow Map, backend properties, and graphs of the key performance indicators (KPIs). The properties indicate how the service is identified and determine how it shows in the flow map and how the metrics are aggregated. For a discussion of baselines and how they are used and configured, see Dynamic Baselines.

The Remote Services Dashboard has two tabs and an action option menu:

- Dashboard: Displays the flow map showing traffic from the calling tier to the remote service, the backend properties used for auto-detection and naming, and key performance indicators.
- Slowest Remote Services Calls: Lists up to ten calls to the service with the longest execution time, by tier and for all tiers.

- From the Action menu, you can also rename a backend, delete a backend, or resolve a backend to a tier.

Slow Remote Service Calls

AppDynamics displays a list of the slowest remote service calls with call details. Click **Troubleshoot > Slow Response Times > Slowest DB & Remote Service Calls** tab to view specific call details and related business transaction snapshots that can help you to troubleshoot.

The Slowest DB & Remote Service Calls tab lists up to ten calls to a remote service with the longest execution time over the selected time frame, by tier and for all tiers, and relevant metrics.

Max Time determines which calls are displayed in the **Slowest DB & Remote Service Calls** list. **Max Time** must exceed 50 ms before AppDynamics tracks the call as a potential candidate for this list. App agents aggregate and report call data to the Controller every 15 minutes.

If Query, Queue, or URL are not specified in the custom exit call, the details for the slowest service call will not be displayed.

Stale Remote Service Removal

A stale remote service is a service for which AppDynamics has previously captured metrics, but which has experienced no metric activity for a certain time period. If a backend is not called for 30 days (in other words, the call per minute metric is zero for 30 days) AppDynamics considers it a stale backend and removes it as a monitored artifact.

After a backend is removed, its metrics no longer appear in the Metric Browser and health rules that reference those metrics do not fire. You should remove any health rules conditions that reference metrics in stale backends.

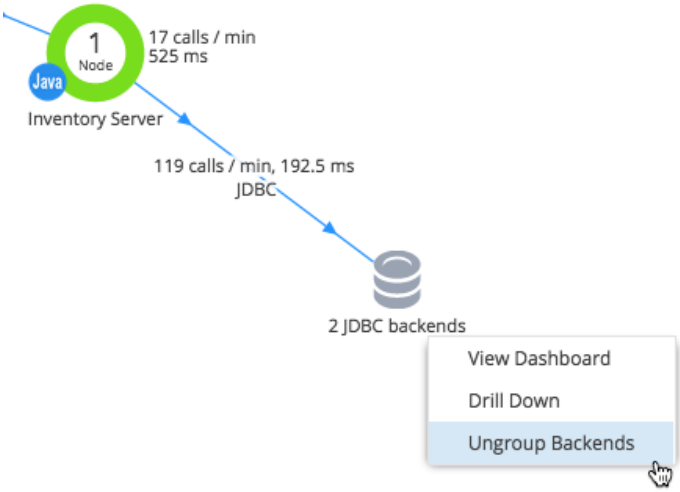
For an on-premises Controller, you can configure the interval by setting the `backend.permanent.deletion.period` property in the Administration Console. Reducing the period may be advisable for large installations, since the maximum number of backends removed in one pass is 50. The minimum interval is one week. Alternatively, you can disable automatic removal by setting the interval to 0.

Group Remote Services on Flow Maps

You can group remote services—or backends—in the flow map configuration to have two or more services of the same type to appear as a single icon in the flow map.

Backend grouping can improve readability and the focus of the flow map, for example, if individual databases or remote service instances are not important to the flow map users.

The following screenshot shows grouped databases on a flow map:

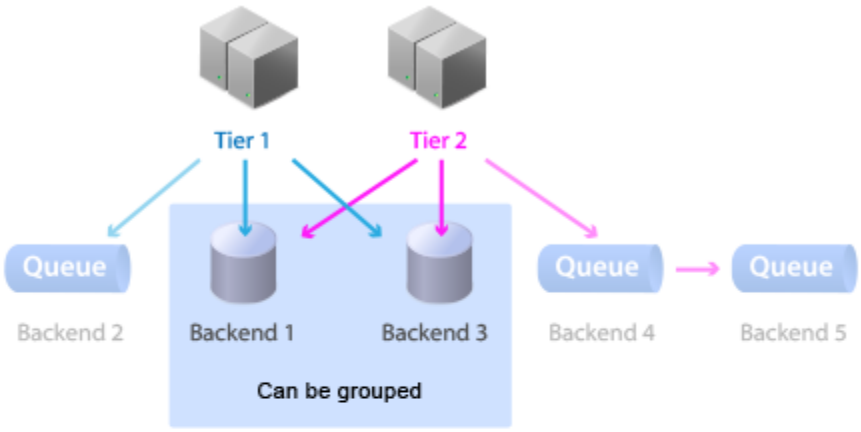


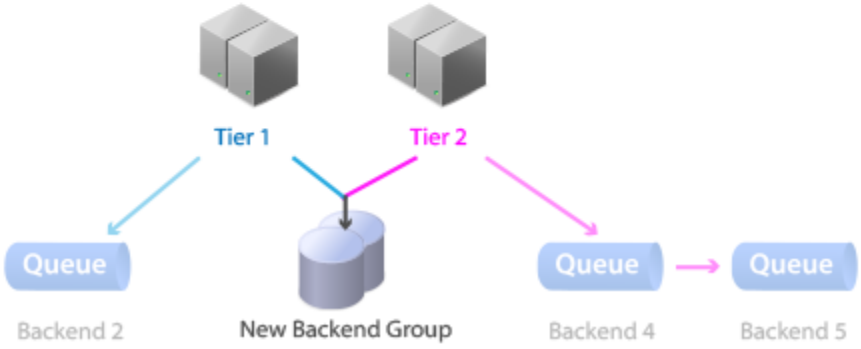
If you want to ungroup backends on a flow map, right-click the backend group and choose **Ungroup Backends**.

Grouped backend services must meet the following conditions:

- All backends must be of the same type
- All backends are called by the same tiers
- No backend in the group calls other backends or calls into other tiers

The following example shows the scenarios in which a set of backends could be grouped.





You can configure backend grouping in the Databases & Remote Services tab of the Configure Flow Map dialog.

You can set a few parameters around grouping, such as the minimum number of backends must be present before they are grouped. For example, if you set the minimum to 4 and only three backends are detected, they are not grouped.

You can exclude a specific backend server from ever being grouped by the flow map, regardless of the grouping configuration, by checking the **Ungroupable** checkbox for that backend in the Visibility and Grouping list.

If there are specific database or remote services that you do not want to see or group in the flow map, check the **Hidden** or **Ungroupable** check box.

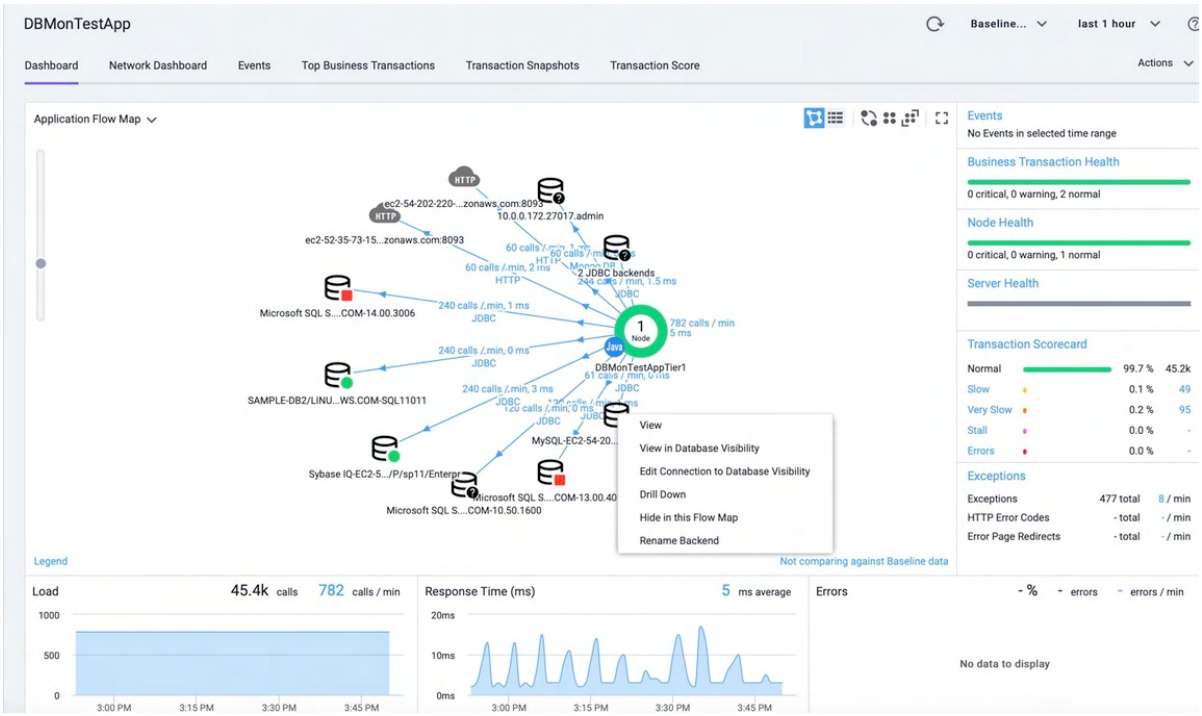
Monitor Databases

AppDynamics can monitor the performance of database calls made by instrumented applications.

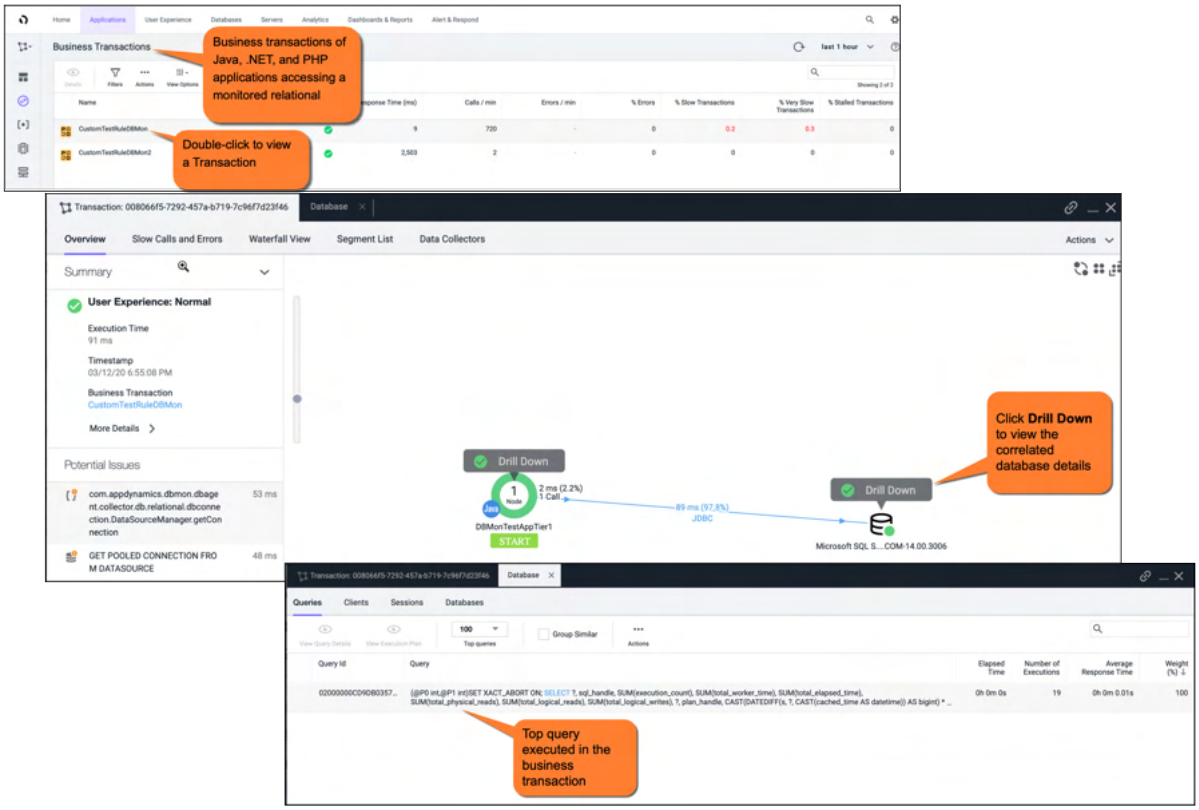
Measure Database Performance

AppDynamics collects metrics for database calls and response times at the following levels:

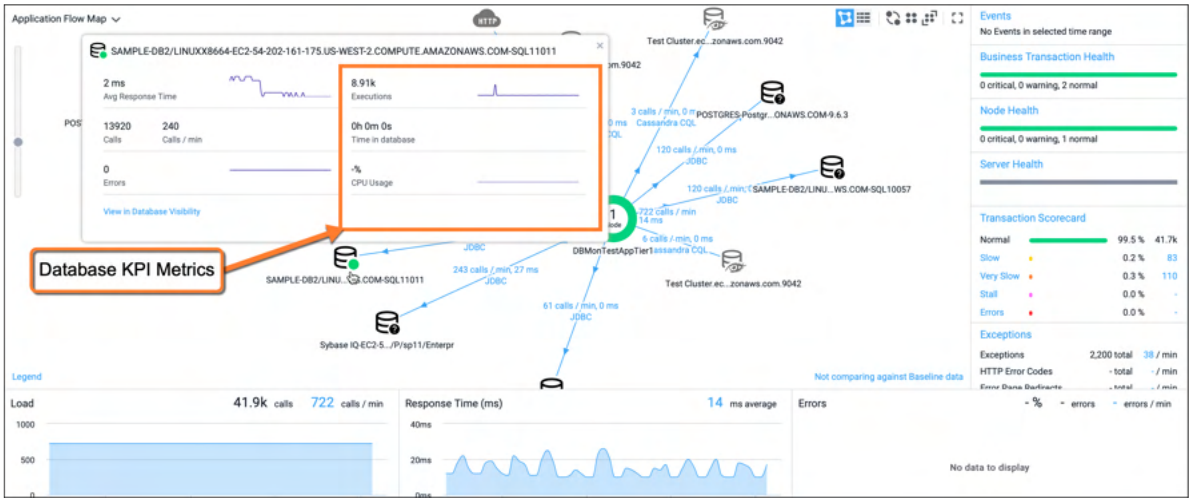
- Business transaction metrics: The metrics for a specific business transaction for a specific database are visible on the transaction flow map.
- Tier metrics: The metrics for all calls from a tier to the specified database are visible on the tier flow map.
- Database call metrics: The overall database access metrics across the application (all business transactions) are visible on the application flow map and the Database Calls dashboard.
- Integrated database metrics with **Database Visibility**:
    - When a database Collector has been set up in AppDynamics **Database Visibility**, you can link to that product from the Application and Database Calls dashboards. If you have already associated the database server with a Database Collector in Database Visibility, you can view the **Database Visibility** UI. If not, the disconnected database icon is displayed and you are prompted to link the database to an already configured server or cluster in **Database Visibility**. Also, you must have view permission to the corresponding collector configured in **Database Visibility**, to map/un-map it with the database server or cluster.



- A relational database backend of a Java application has the same hostname, port number, and database type as a database server already configured in a database Collector, the Oracle backend is automatically matched with the corresponding **Database Visibility** Collector. The Snapshot correlation view, which is available for relational database backends, shows the details of queries, clients, sessions, and schemas when the snapshot was captured.



- When a backend database is linked to a server or a cluster in **Database Visibility**, a click the database icon displays basic details with a link to **Database Visibility** UI. You can also view the database KPI metrics.



By default, many databases and data stores are automatically detected when calls are made from nodes instrumented with AppDynamics app agents. See Access Database Visibility from Application Monitoring Views.

To monitor call performance to a database, confirm that it appears in the Databases Calls list and has its own Database Calls dashboard. If a database is not appearing, check the configuration.

The database icons on the flow maps help you identify the status of the database. The following table lists the different database icons:

Database Icon	Status

	The database is healthy. No active health rule violations
	Health Rule Violation - Critical Condition
	Health Rule Violation - Warning Condition
	The database is either not linked to the appropriate server or cluster node. This icon also appears when the database is in the process of changing the state.
	The database is not connected to the server or cluster in **Database Visibility**. Click the database icon, then the **Connect** link to link the database backend to the appropriate server or cluster.

View Database Performance on Flow Maps

Databases detected during the specified time window show up on the Application Dashboard flow map, where you can view them in the context of the entire application's transaction flow. The application flow map displays calls per minute and average response time for calls made to databases. These metrics include all calls made from a specific tier to a database across all business transactions. The tier and node flow maps display a similar metric aggregating data from calls across all business transactions by tier or node respectively.

The detected databases appear on the Tier Flow Map, where you can view them in the context of the traffic on this specific tier.

For business transactions involving calls to databases, the databases appear on the Transaction Flow Map, where you can view them in the context of the traffic for this specific business transaction. The transaction flow map shows the average time spent in database calls for the business transaction.

Drill Down on Database Performance

In addition to seeing monitored databases in the flow map, you can view:

- Business transactions that make the most database calls. To see this, navigate to Database Business Transactions panel.
- Queries executed the most number of times by business transactions. To see this, double click on the business transaction and open the DB Queries tab.

Resolving Unexpected Databases on the Flow Map

AppDynamics can sometimes reveal unexpected connections from an application to a database on the flow map. If this occurs for you, try the following to determine why this database appears:

- From the left navigation menu, select **Tiers & Nodes > Databases**. Select the Slowest Database Calls tab and drill down into the snapshots to see the code that is calling the database. See To troubleshoot slow database and remote service calls.
- Run a diagnostic session to capture some transaction snapshots and look for calls to the database. See Diagnostic Sessions and Transaction Snapshots. If you have integrated AppDynamics for Databases, you can select a transaction snapshot that involves running SQL on an Oracle database and from the Transaction Flow Map you can link to AppDynamics for Database to see all queries executed in the SQL session that are associated with that transaction snapshot. See Use AppDynamics Pro with AppDynamics for Databases.
- Are any exceptions thrown when the database is seen? If so, look for error snapshots that point to the exception trace.

View Discovered Databases

The database list shows all detected databases along with key performance indicators. Stale databases can be configured to be automatically removed.

From the database list, you can select a database and click **View** to see the Database Calls dashboard. The dashboard displays a Database Flow Map, database properties, and graphs of the key performance indicators (KPIs). The database properties indicate how the agent identifies the database and control how it shows in the display map and how the metrics are aggregated. For a discussion of baselines and how they are used and configured, see Dyn amic Baselines.

The database dashboard has two tabs and an action options menu:

- Dashboard: Displays the flow map showing traffic from the calling tier to the database, the backend properties used for auto-detection and naming, and key performance indicators.
- Slowest Database Calls: Lists up to ten calls to the database with the longest execution time, by tier and for all tiers. See Slow Database Calls.
- The Action menu provides additional actions:
    - **Rename Backend**: Renames the database.

- **Resolve Backend to Tier**: Associates the database with the tier that you select so that the backend appears in the grid view of the tier and not as an independent component ("unresolved backend") on the application dashboard flow map. You can reverse this operation from the **Configure Backends resolving to this Tier** item in Actions menu in the Tier dashboard.
- **Delete Backends**: Removes instances of the database from the controller and all agents. An agent can re-discover the database and register it with the controller.

You can access the Database Server List by clicking **Servers > Databases**. See Backend Detection Rules.

Slow Database Calls

AppDynamics displays a list of the slowest database calls. For each call, you can view specific call details and related business transaction snapshots. The list shows up to ten database calls with the longest execution time over the selected time frame, by tier and for all tiers. Each call shows:

- **Call**: SQL Query
- **Avg. Time per Call (ms)**: the average time per call in milliseconds
- **Number of Calls**: the number of calls executed during the time range
- **Max Time (ms)**: the maximum execution time in milliseconds
- **View snapshots**: a link to view existing transaction snapshots

App agents aggregate and report call data to the Controller every 15 minutes. **Max Time** determines which calls are displayed in the **Slowest Database Calls** list. For example for JDBC calls, **Max Time** must exceed 10 ms before AppDynamics tracks the call as a potential candidate for this list.

Slowest database calls are defined as:

- Max Time greater than 10 ms
- Top ten slowest
- Reported every 15 minutes

View Slowest Database Calls

To view slowest database calls, click **Troubleshoot > Slow Response Times > Slowest DB and Remote Service Calls**.

On this page, you can do the following:

- If transaction snapshots are available for a call, you can click the **View Snapshots** link in the **Snapshots** column to select a snapshot and drill down to the root cause of the slowness.
- You can view explain plans by selecting a call and clicking **View Details**. In the dialog box, click **Explain Plans**. If parameter values are filtered out from the captured SQL, the Explain Plan feature is disabled.

NoSQL

AppDynamics displays NoSQL databases as Remote Services. See Remote Services.

Resolve Remote Services to Tiers

Resolving remote services to a tier can put the metrics for the services into an element of the AppDynamics application model that may provide a more useful view of the activities of the services in flow maps.

The tier may already exist in the business application model or you can create a new one. If it exists, after you resolve the backend to a tier it no longer appears as a backend icon in the flow maps.

AppDynamics automatically resolves some backends to tiers. In general, these are custom backends or HTTP, RMI, Thrift or web services, which are usually logically linked to a tier. If you want to see the backend on its own, you can unresolve it to display it on the flow map.

When to Resolve Backends to a Tier

You may want to resolve a backend to a tier if it represents a more logical view of your environment, for example, if a database runs on the same machine as an application server. Another reason to resolve backends to a tier is to combine similar backends.

For example, if there appear to be multiple databases that are actually the same database using different namespaces, the default flow map is likely to display the databases separately. You can resolve the database backends to a new tier of type "Database Server" so they display as one and the tier metrics are together. If a backend is taken out, the historical data for the backend is retained at the tier level.

Resolving Backends to Tiers

You can resolve a backend to a tier from the backend's dashboard page, either its Remote Service Dashboard or Database Dashboard. In the database dashboard, from the **Actions** menu, click **Resolve Backend to Tier** and select an existing tier or create a new one. Similarly, you can see backends that are associated with a tier and unresolve a backend from the action menu for the Tier dashboard.

Deleting the backend from the tier causes it to re-appear as an unresolved backend in the flow map or resolve it to a different tier.

Troubleshooting Applications

Related pages:

- Garbage Collection
- Error Detection

AppDynamics includes many tools and views designed to help you diagnose application-related problems.

Access Troubleshooting

When starting to troubleshoot an application problem, you should begun in the **Troubleshoot** section of the UI. You can access from the left-hand navigation pane of the Controller UI in an application context.

The area includes pages for analyzing slow response times, errors and exceptions, and health rule violations. It also provides access to war rooms, an area of the UI dedicated to troubleshooting a specific problem.

Additional Help

If slow response time persists even after you've completed the steps outlined above, you may need to perform deeper diagnostics. If you cannot find the information you need in the AppDynamics documentation, consider posting a note about your problem in the Community Discussion Boards. These discussions are monitored by customers, partners, and AppDynamics staff.

Slow Response Times

You may receive a notification based on a health rule violation, see performance indicators in flow maps or transaction scorecards that indicate slow response times. When you do, the following guidelines provide a strategy for troubleshooting and diagnosis.

The Slow Response Times Dashboard in the Troubleshoot menu lists the Business Transactions most responsible for slow Average Response Time (ART). From that dashboard, you can select a time range centered around a spike in ART that also captures more normal-looking time spans. This information is also available through the **Top Business Transactions** tab on the **By Contribution to App Average Response Time** tile.

Step 1: Check for slow or stalled business transactions.

AppDynamics uses transaction thresholds to detect slow, very slow, and stalled transactions. Follow these steps to determine if you complete Step 2 or move on to Step 3.

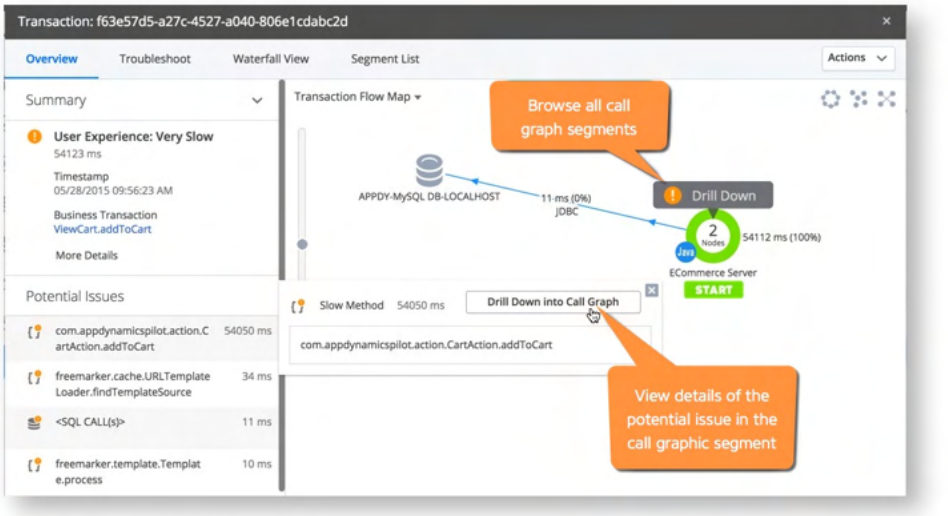
Do you have slow or stalled transactions?

1. Make sure that the time frame selected in the Controller UI encompasses the time when the performance issue occurred. If it's a continuing condition, you can keep the time frame relatively brief. Use the **Time Range** dropdown.
2. Click **Troubleshoot > Slow Response Times**.
3. Click the Slow Transaction**s** tab if it is not already selected.
4. Do you see one or more slow transaction snapshots on this page?

    - If **Yes** – Go to Step 2 to drill down to find the root cause.
    - If **No** – Go to Step 3 and check for slow backends.

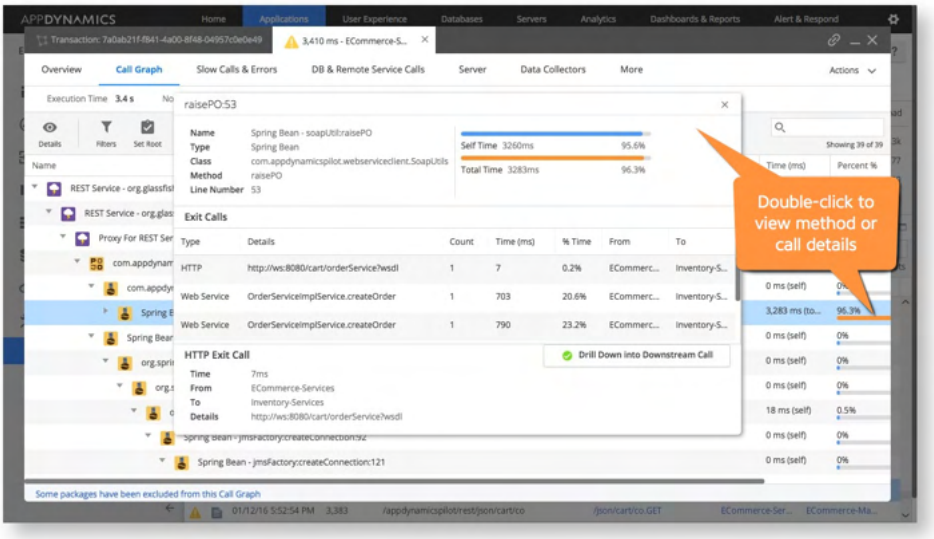
Step 2: Drill into slow or stalled transactions to determine the root cause.

1. From the left menu, navigate to **Troubleshoot** > **Slow Response Times**.
2. In the lower pane of the Slow Transactions tab, click the **Exe Time (ms)** column to sort the transactions from slowest to fastest.
3. Select a snapshot from the list and click **Details**.
4. Review the **Potential Issues** list to see the longest-running method and SQL calls in the transaction.
5. Click a potential issue and select **Drill Down into Call Graph** to go directly to that point in the call graph, or click **Drill Down** in the transaction flow map pane to see the complete set of call graph segments retained for this transaction.



6. View the **Time (ms)** column to see how long this method execution takes relative to the transaction execution time.

7. Click the **HTTP** link in the last column on the right to display the information details pain. In this example, the selected method is taking 96.3% of the transaction execution time.



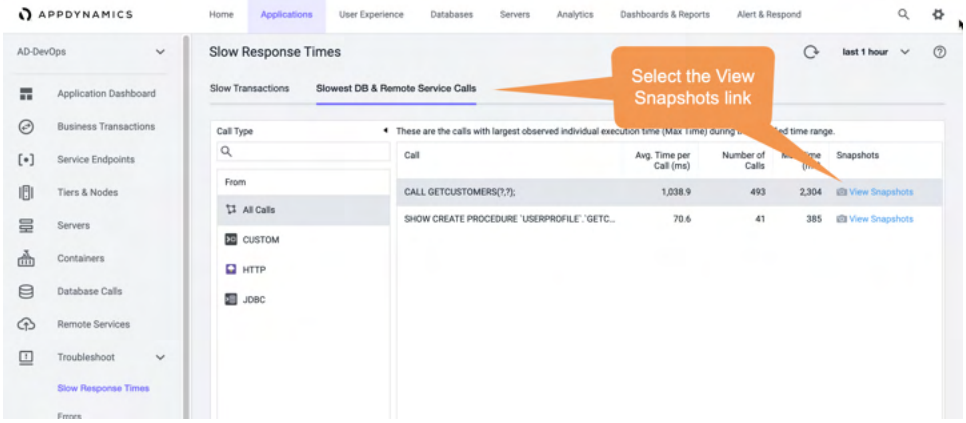
8. Note the *Class, Method,* and *Line Number* (if available) represented by the execution segment. This information provides a starting point for troubleshooting this code issue.

If there are multiple slow or stalled transactions, repeat this step until you have resolved them all and then continue to Step 3.

Step 3: Check for slow database or remote service calls on the backend.

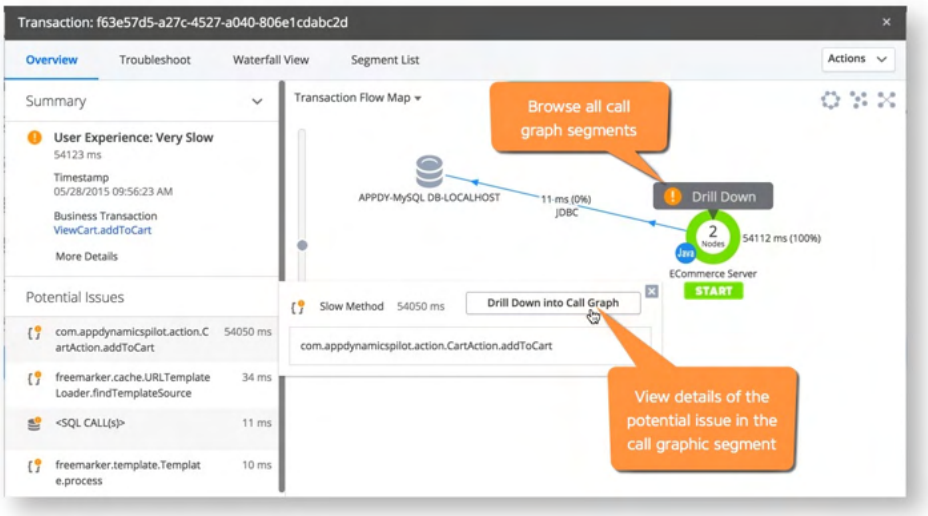
AppDynamics collects metrics about the performance of business transaction calls to the databases and remote servers from the instrumented app servers. You can drill down to the root cause of slow database and remote service calls.

1. Click **Troubleshoot > Slow Response Times**, then click the Slowest DB & Remote Service Calls tab.
2. Select a call from the list and click the **View Snapshots** link to show a list of Correlated Snapshots.



3. Click the **Exe Time (ms)** column to sort the transactions from slowest to fastest.
4. Select a transaction and click **Drill Down**.
5. Select a potential issue from the **Potential Issues** list

6. Click **Drill Down into Call Graph** to go directly to that point in the call graph, or click **Drill Down** in the flow map pane to see the complete set of call graph segments retained for this transaction.

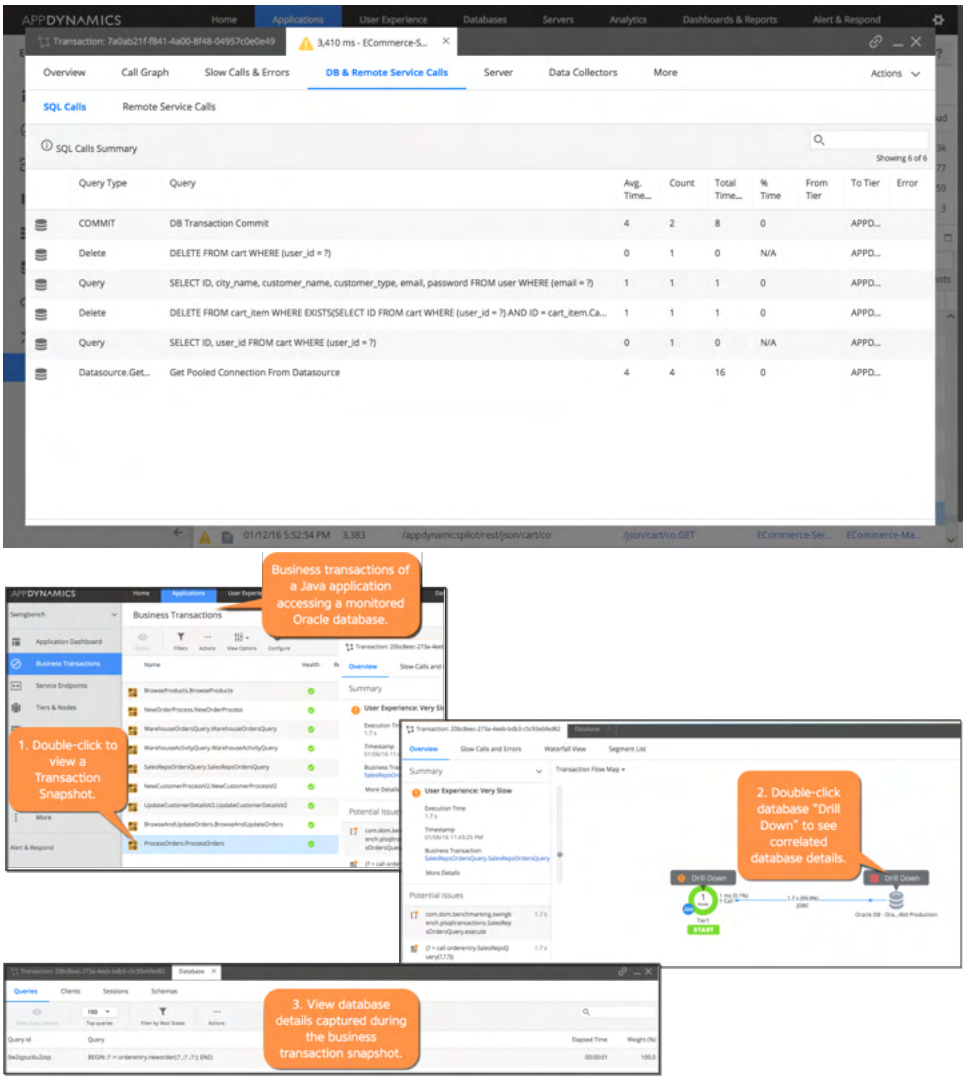


7. Review the **Time (ms)** column and select the transaction with the longest execution time for this method relative to the transaction execution time.
8. Select the DB & Remote Service Calls tab.
9. Do you see one or more slow calls on the **SQL Calls** or **Remote Service Calls** tabs?

    - If **Yes** – Go to Step 4 to drill down and find the root cause.
    - If **No** – Go to Step 5 to determine if the problem is affecting all nodes in the slow tier.

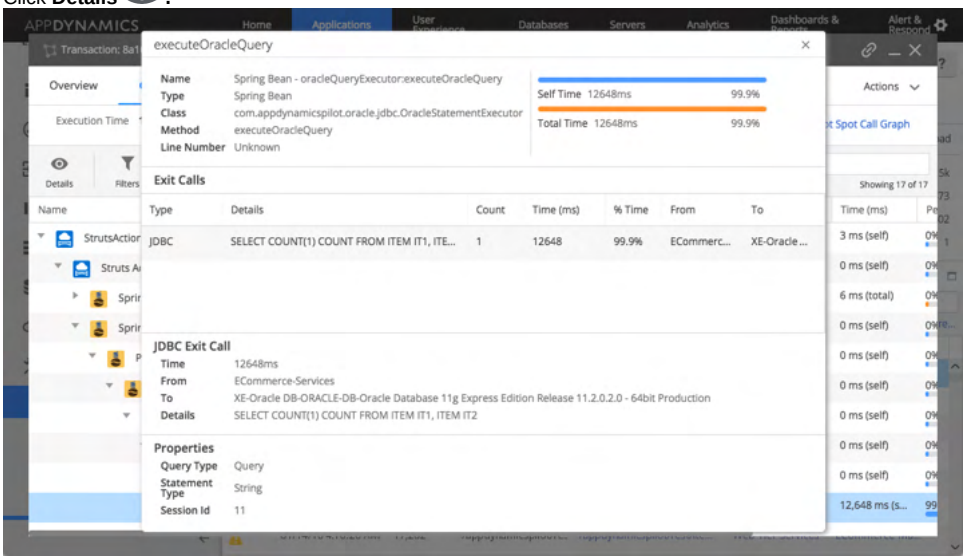
Step 4: Drill into SQL or Remote Service Calls to determine the root cause.

1. On the SQL Calls tab of the transaction snapshot, sort the SQL Calls by Avg. Time (ms).
    a. **Slow database call**– click the database call to gain information about the call.
    b. If you have **Correlated snapshots** between Java applications and Oracle databases – drill down into the Oracle database on the Transaction Snapshot to view database details captured during the snapshot.
    c. If you have **AppDynamics for Databases –** right-click the database on the Application, Tier, Node or Backend Flow Map, and choose **Link to AppDynamics for Databases**. You can use AppDynamics for Databases to diagnose database issues.
    d. If you have **Database Monitoring –** right-click the database on the Application, Tier, Node or Backend Flow Map, and choose **View** to review database problems.



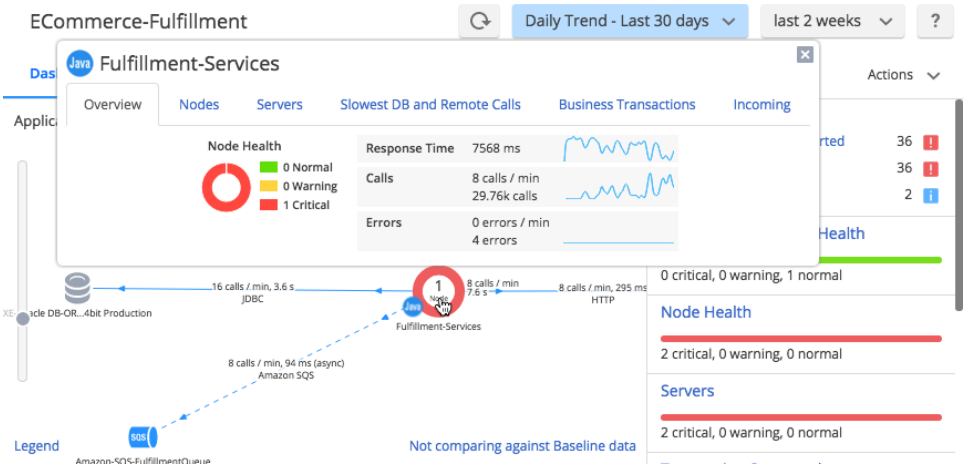
2. On the Remote Service Calls tab, sort the queries by Avg. Time (ms) .
3. Select the slow call.
4. Click **Drill Down into Downstream Call** to gain further insight into the methods of the service call.
5. Sort the methods by the **Time (ms)** column.
6. Select a slow method.

7. Click **Details** .



Step 5: Does the problem affect all nodes in the slow tier.

1. In the Application or Tier Flow Map, click the **tier** or **node** icon to see a quick overview of the health of each node in the tier.



2. Is the problem affecting all nodes in the slow tier? If all the nodes are yellow or red, the answer to this question is Yes. Otherwise, the answer is No.

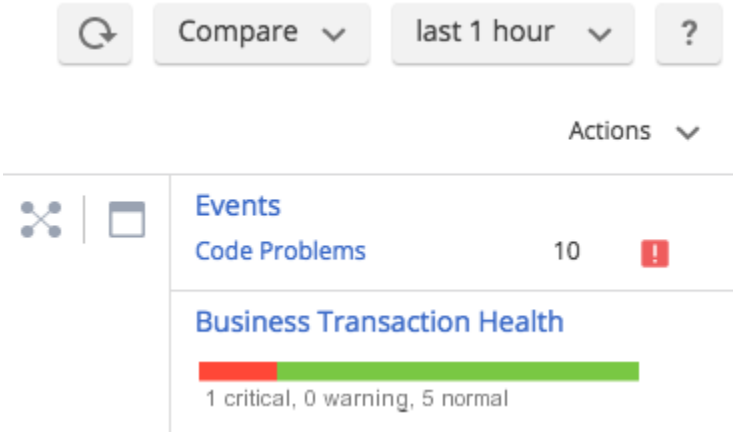
    - If **Yes** – Go to Step 6.
    - If **No** – The problem is either in the node's hardware or in the way the software is configured on the node. If only one node in a tier is affected, the problem is probably not related to the application code. Follow these steps to determine a hardware related problem.

        i. In the left navigation pane, click **Tiers & Nodes**.
        ii. Expand the Tier in the right pane and double-click the affected node to open its Node Dashboard.
        iii. Click the Memory tab
            1. Explore each of the available tabs to determine if you need to add memory to the node, configure additional memory for the application, or take some other corrective action.
        iv. Click the **Server** tab.
            1. If the Hardware tab indicates a hardware related problem, contact your IT department.

You have isolated the problem.

Step 6: Check to see whether the problem affects most business transactions

1. On the **Application Dashboard**, look at the *Business Transaction Health* pane on the right side of the screen.



2. Is the bar representing Business Transaction Health is primarily yellow or red? Yes or No?

    - If **No** – Then:

        i. Click **Business Transactions** in the left navigation pane.
        ii. Sort by Health, Transaction Score or other column headings to find the business transaction that is experiencing issues.
        iii. Double-click the problematic business transaction to see its dashboard, then use the tabs to diagnose the problem.

You have isolated the problem.

Additional Help

If you've tried to diagnose the problem using the previous steps and haven't found the problem, see additional information for your specific agent:

- Troubleshooting Applications
- Monitor Databases
- Troubleshoot Mobile Applications
- Slow Response Times for .NET

Errors and Exceptions

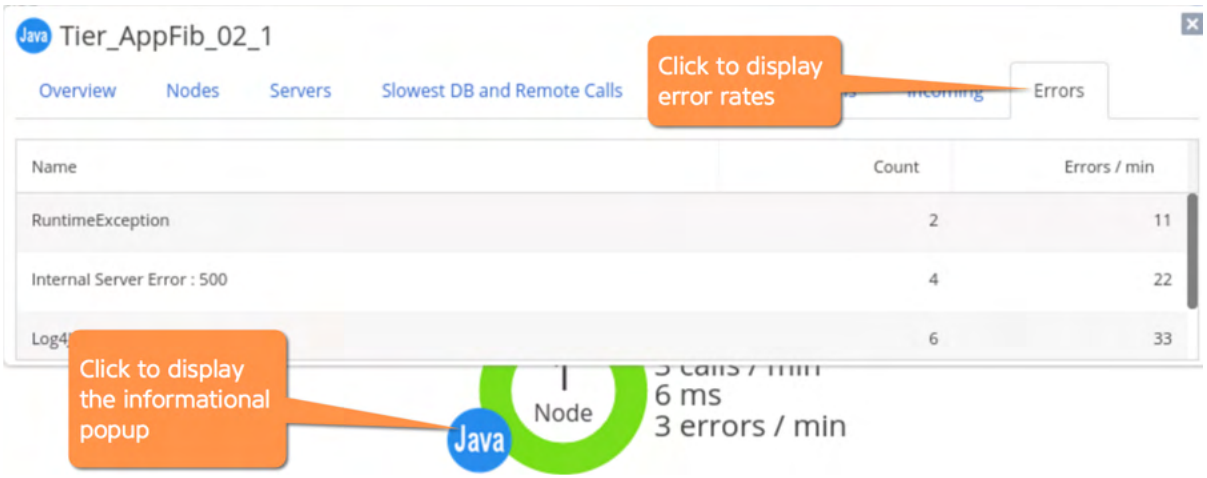
AppDynamics Application Intelligence Platform captures and presents information on business transaction errors in the monitored environment.

At a high-level, a business transaction is considered to have an error if its normal processing has been affected by a code-level exception or error event, including custom error events based on methods you specify.

View Error and Exception Information

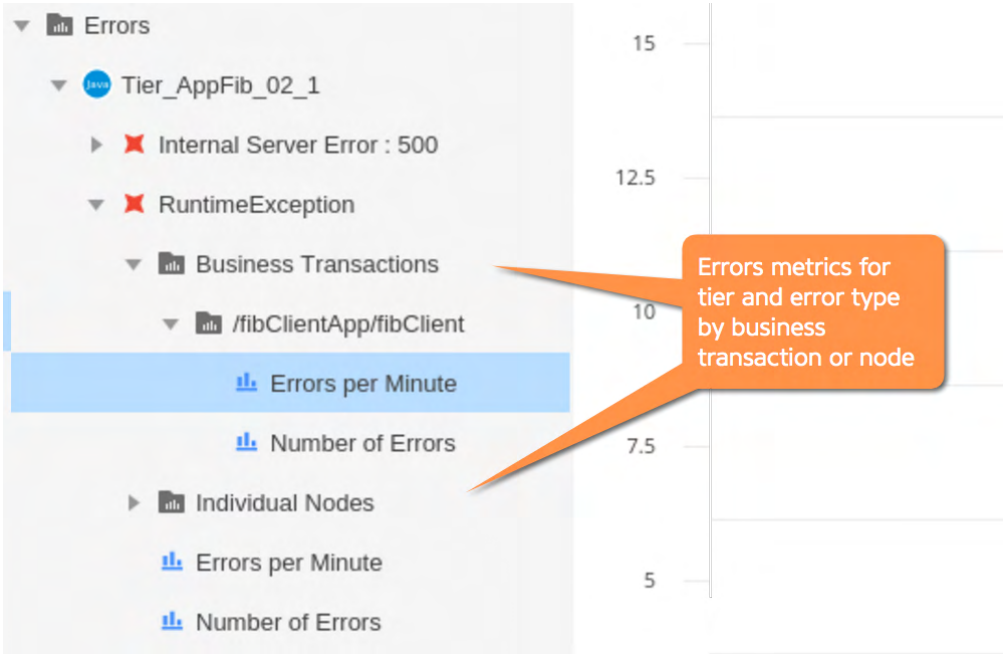
The Controller UI presents information on errors and exceptions in various places in the UI, including in transaction snapshots, metrics, and dashboards.

The informational popups for tiers in flow maps have an error tab that displays error rate metrics for the tier:



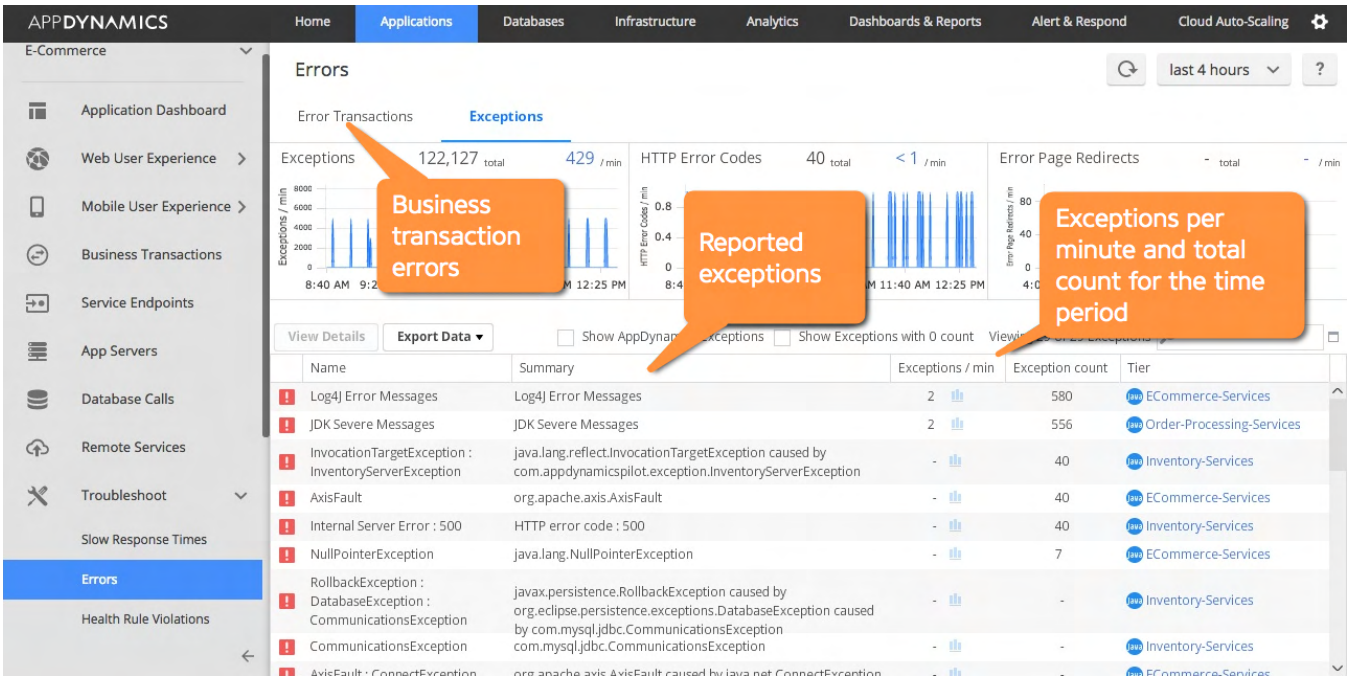
On the application and tier flow maps, the error rate is for all business transactions. On the business transaction flow map, errors apply only to the current business transaction.

The Metric Browser includes Error metrics:



The **Troubleshoot** > **Errors** page shows all error transactions. The page contains two tabs, one for transaction errors and one for Exceptions.

The tabs show information on the rate of errors or exceptions, and lets you drill down to the error or exception for more information, as shown:



Business Transaction Error

All transaction errors that have been detected according to the configured error detection rules in the selected time frame of the Controller UI appear in the Error Transactions tabs of the **Errors** page.

By default, AppDynamics considers a business transaction to be in error if it detects one of the following types of events in the context of the transaction:

- An unhandled exception or error. An exception that is thrown and never caught or caught after the business transaction terminates results in a transaction error, and the exception is presented in AppDynamics. An exception that is thrown and caught within the context of the business transaction is not considered a transaction error and the exception is not captured in AppDynamics.
- An exception caught in an exit call, such as a web service or database call.
- An HTTP error response, such as a status code 404 or 500 response.
- A custom-configured error method and error message.

Error detection configuration is described in Error Detection.

Errors that occur on a downstream tier that are not propagated to the originating tier do not result in a business transaction error. If the originating client receives a 200 success response, for example, the business transaction is not considered an error. The error contained within the downstream tier does count against the Error Per Minute metric for the continuing segment.

When a business transaction experiences an error, it is counted only as an error transaction. It is not also counted as a slow, very slow or stalled transaction, even if the transaction was also slow or stalled.

Code Exceptions in AppDynamics

Code exceptions are a common cause of business transaction errors. The Exceptions tab in the **Errors** page shows an aggregated view of the exceptions across all transactions. For purposes of this view, AppDynamics considers the following types of incidents to be exceptions:

- Any exception logged with a severity of Error or Fatal (using Log4j, java.util.logging, Log4Net/NLog, or another supported logger). This applies even if the exception occurs outside the context of a business transaction, in which case the exception type is specified as *Application Server*.
- HTTP errors that do not occur in the context of a Business Transaction.
- Error page redirects.

Exceptions that are thrown and handled within a business transaction are not captured by AppDynamics and do not appear in the Exceptions tab.

When troubleshooting errors, notice that the number of business transaction errors does not necessarily correspond to the number of exceptions in a given time frame. A single transaction that counts as an error transaction can correspond to multiple exceptions. For example, as the transaction traverses tiers, it can generate an exception on each one. Troubleshooting an error typically involves finding the exception closest to the originating point of the error.

If a stack trace for the exception is available, you can access it from the Exception tab in the Controller UI. A stack trace is available for a given exception if the exception was passed in the log call. For example, a logging call in the form of `logger.log(Level.ERROR, String msg, Throwable e)` would include a stack trace, whereas a call in the form of `logger.log(Level.ERROR, String msg)` would not.

Agent Errors

The Java Agent differentiates between agent internal errors and application errors. By default, agent internal errors no longer set off health rule violations. You can view agent internal errors in the following Metric Browser path: **Application Infrastructure Performance > <tier> > Agent > Internal Errors**.

Error and Exception Limits

AppDynamics limits the number of registered error types (based on error-logging events, exception chains, and so on) to 4000. It maintains statistics only for registered error types.

Reaching the limit generates the `CONTROLLER_ERROR_ADD_REG_LIMIT_REACHED` event. While it is possible to increase the limit, we recommend refining the default error detection rules to reduce the number of error registrations to have the error you are not interested in capturing ignored.

For more information, see information on configuring log errors and exceptions to ignore in Error Detection.

Configure Errors and Exceptions

AppDynamics automatically recognizes errors and exceptions for many common frameworks. You can customize the default error detection behavior as needed, for example, if you use your own custom error framework. See Error Detection.

Slow Response Times for .NET

You may notice that your application's response time is slow from these methods:

- You receive an alert: If you have received an email alert from AppDynamics that was configured through the use of health rules and policies, the email message provides a number of details about the problem that triggered the alert. See information about Email Notifications in Notification Actions. If the problem is related to slow response time, see Initial Troubleshooting Steps.
- You view the Application Dashboard for a business application and see slow response times.
- A user reported slow response time that relates to a particular business transaction, for example, an internal tester reports "Searching for a hotel is slow".

Initial Troubleshooting Steps

In some cases, the source of your problem might be easily diagnosed by choosing **Troubleshoot > Slow Response Times** in the left navigation pane. See Slow Response Times.

.NET Resource Troubleshooting

If you've tried to diagnose the problem using those techniques and haven't found the problem, use the following troubleshooting approaches to find other ways to determine the root cause of the issue.

Step 1 - CPU saturated?

Is the CPU of the CLR saturated?

1. Display the Tier Flow Map.
2. Click the Nodes tab, and then click the Hardware tab.
3. Sort by CPU % (current).

If the CPU % is 90 or higher, the answer to the question in Step 4 is Yes. Otherwise, the answer is No.

Yes – Go to Step 2

No – Review various metrics in the Metric Browser to pinpoint the problem.

In the left navigation pane, click **Servers > Tiers & Nodes > *slow tier***. Review these metrics in particular:

- ASP.NET -> Application Restarts
- ASP.NET -> Request Wait Time
- ASP.NET -> Requests Queued

- CLR -> Locks and Threads -> Current Logical Threads
- CLR -> Locks and Threads -> Current Physical Threads

- IIS -> Number of working processes
- IIS -> Application pools -> <*Business application name*> -> CPU%
- IIS -> Application pools -> <*Business application name*> -> Number of working processes
- IIS -> Application pools -> <*Business application name*> -> Working Set

You have isolated the problem and don't need to continue with the rest of the steps below.

Step 2 - Significant garbage collection activity?

1. Display the Tier Flow Map.
2. Click the Nodes tab, and then click the Memory tab.
3. Sort by Time Spent on Collections (%) to see what percentage of processing time is being taken up with garbage collection activity.

If Time Spent on Collections (%) is higher than acceptable (say, over 40%), the answer to the question in Step 5 is Yes. Otherwise, the answer is No.

Is there significant garbage collection activity?

Yes – Go to Step 3.

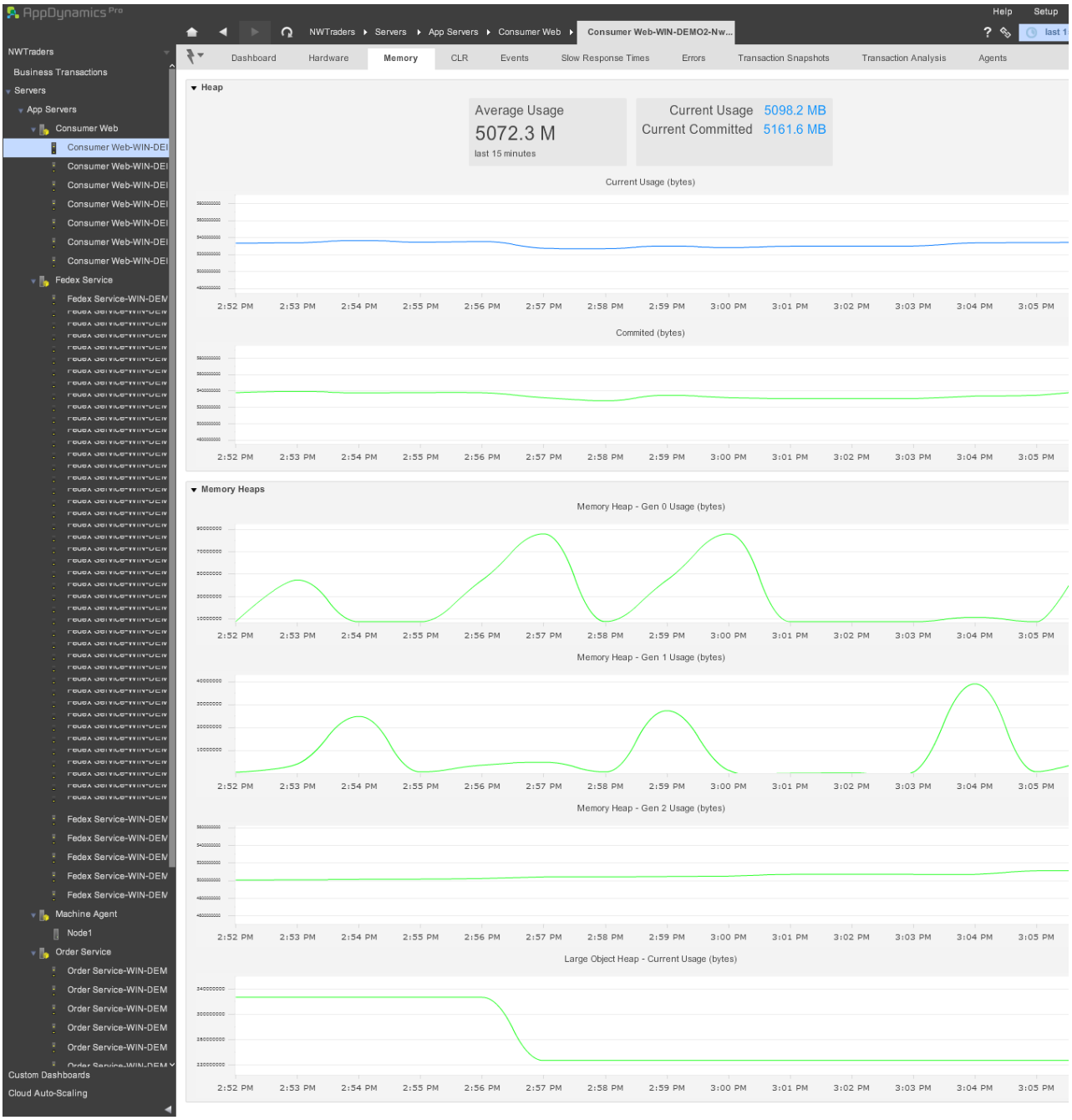
No – Use your standard tools to produce memory dumps; review these to locate the source of the problem.

You have isolated the problem and don't need to continue with the rest of the steps below.

Step 3 - Memory leak?

Is there a memory leak?

1. From the list of nodes displayed in the previous step (when you were checking for garbage collecting activity), double-click a node that is experiencing significant GC activity.
2. Click the Memory tab, then review the committed bytes counter and the size of the Gen0, Gen1, Gen2 and large heaps.



If memory is not being released (one or more of the above indicators is trending upward), the answer to the question in Step 6 is Yes. Otherwise, the answer is No.

Yes – Use your standard tools for troubleshooting memory problems. You can also review ASP.NET metrics; click **Tiers & Nodes > *slow tier* > ASP.NET**.

No – Use your standard tools to produce memory dumps; review these to locate the source of the problem.

Whether you answered Yes or No, you have isolated the problem.

Java Resource Issues

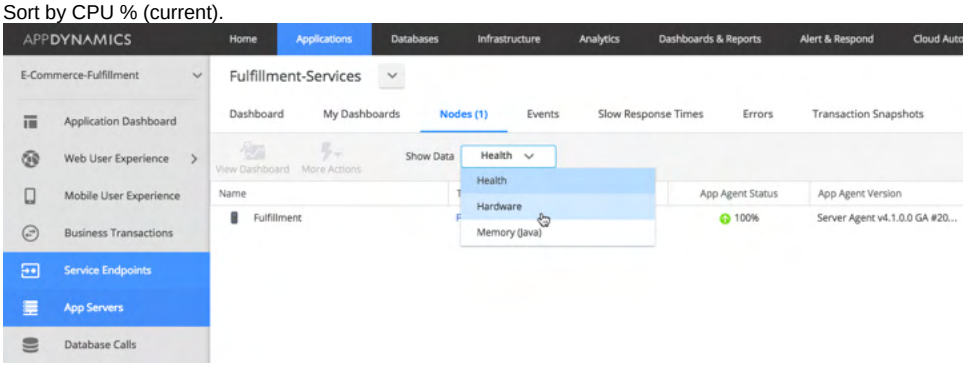
These troubleshooting guidelines may help you determine the root cause of many Java-related issues.

Step 1. CPU saturated?

Is the CPU of the JVM saturated?

How do I know?

1. Display the Tier Flow Map.
2. Click the Nodes tab, and then show **Hardware** data.
3. Sort by CPU % (current).



If the CPU % is 90 or higher, the answer to this question is Yes. Otherwise, the answer is No.

Yes – Go to Step 2.

No – The issue is probably related to a custom implementation your organization has developed. Take snapshots of the affected tier or node(s) and work with internal developers to resolve the issue.

Step 2. Significant garbage collection activity?

Is there significant garbage collection activity?

How do I know?

1. Display the Tier Flow Map.
2. Click the Nodes tab, and then click the Memory tab.
3. Sort by **GC Time Spent** to see how many milliseconds per minute is being spent on GC; 60,000 indicates 100%.
4. If GC Time Spent is higher than 500 ms, the answer to the question in Step 5 is Yes. Otherwise, the answer is No.

Result

Yes – Go to Step 3.

No – Go to Step 4.

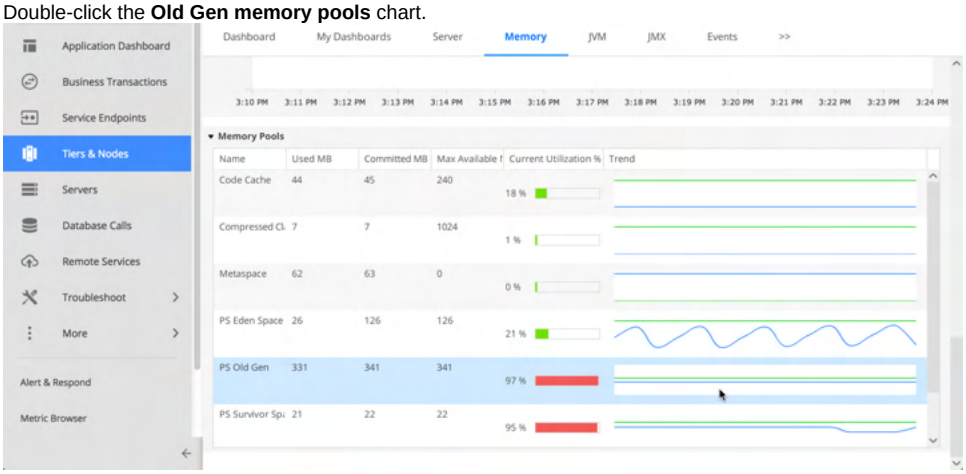
Step 3. Memory leak?

Is there a memory leak?

How do I know?

1. From the list of nodes displayed in the previous step (when you were checking for Garbage Collecting activity), double-click a node that is experiencing significant GC activity.
2. Click the Memory tab, then scroll down to display the Memory Pool graphs at the bottom of the panel.

3. Double-click the **Old Gen memory pools** chart.



If memory is not being released (use is trending upward), the answer to this question is Yes. Otherwise, the answer is No.

Result

Yes – Use various AppDynamics features to track down the leak. One useful tool for diagnosing a memory leak is object instance tracking, which lets you track objects you are creating and determine why they aren't being released as needed. Using object instance tracking, you can pinpoint exactly where in the code the leak is occurring. For instructions on configuring object instance tracking, as well as links to other tools for finding and fixing memory leaks, see Need more help?.

No – Increase the size of the JVM. If there is significant GC activity but there isn't a memory leak, then you probably aren't configuring a large enough heap size for the activities the code is performing. Increasing the available memory should resolve your problem.

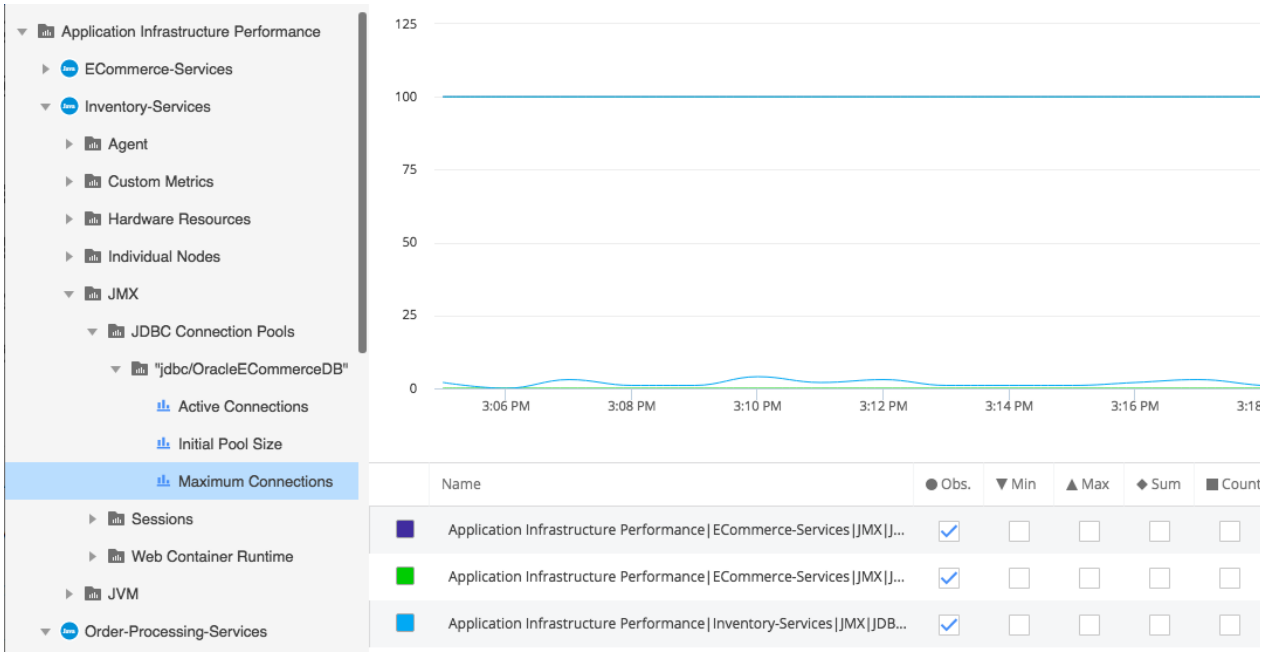
Whether you answered Yes or No, you have isolated the problem.

Step 4. Resource leak?

Is there a resource leak?

How do I know?

1. In the left Navigation pane, go to (for example) **Metric Browser > Application Infrastructure Performance > *TierName* > Individual Nodes > *NodeName* > JMX > JDBC Connection Pools > *PoolName***.
2. Add the **Active Connections** and **Maximum Connections** metrics to the graph.
3. Repeat as needed for various pools your application is using.



If connections are not being released (use is trending upward), the answer to the question in Step 7 is Yes. Otherwise, the answer is No.

Result

Yes – To determine where in your code resources are being created but not being released as needed, take a few thread dumps using standard commands on the problematic node. You can also create a diagnostic action within AppDynamics to create a thread dump; see *Thread Dump Actions* in Diagnostic Actions.

No – Restart the JVM. If none of the above diagnostic steps addressed your issue, it's possible you're simply seeing a one-time unusual circumstance, which restarting the JVM can resolve.

Java Memory Leaks

This page describes how to detect and troubleshoot Java memory leaks.

- To activate Automatic Leak Detection, you need the Configure Agent Properties permission.
- To start an On Demand Capture Session, you need the Advanced Agent Operation permission. See Create and Manage Custom Roles.

The garbage collection feature of the JVM greatly reduces the opportunities to introduce memory leaks into a codebase. However, because garbage collection does not eliminate memory leaks completely, AppDynamics includes Automatic Leak Detection for supported JVMs.

Automatic Leak Detection

You can access **Automatic Leak Detection** on the Memory tab of the Node Dashboard. **Automatic Leak Detection** is disabled by default because it increases overhead on the JVM. You should enable leak detection mode only when you suspect a memory leak problem. Turn off **Automatic Leak Detection** after you identify the cause for the leak.

Automatic Leak Detection uses **On Demand Capture Sessions** to capture actively used collections, any class that implements JDK Map or Collection interface during the capture period. The default capture period is 10 minutes.

AppDynamics tracks every Java collection that meets the following criteria:

- The collection has been alive for at least 30 minutes.
- The collection has at least 1000 elements.
- The collection Deep Size is at least 5 MB. The agent calculates Deep Size by traversing recursive object graphs of all the objects in the collection.

The following node properties define the defaults for leak detection criteria:

- `minimum-age-for-evaluation-in-minutes`
- `minimum-number-of-elements-in-collection-to-deep-size`
- `minimum-size-for-evaluation-in-mb`

See App Agent Node Properties.

The Java Agent tracks the collection and identifies potential leaks using a linear regression model. You can identify the root cause of the leak by tracking frequent access to the collection over a period of time.

After it qualifies a collection, AppDynamics monitors the collection size for a long-term growth trend. Positive growth indicates the collection is the potential source of a memory leak.

After AppDynamics identifies a leaking collection, the Java Agent automatically triggers diagnostics every 30 minutes. The diagnostics capture a shallow content dump and activity traces of the code path and business transactions that access the collection. You can drill down into any leaking collection monitored by the agent, to manually trigger Content Summary Capture and Access Tracking sessions.

You can also monitor memory leaks for custom memory structures. Typically custom memory structures are used as caching solutions. In a distributed environment, caching can easily become a prime source of memory leaks. It is therefore important to manage and track memory statistics for these memory structures. To do this, you must first configure custom memory structures. See Custom Memory Structures for Java.

Workflow to Troubleshoot Memory Leaks

You can use this workflow to troubleshoot memory leaks on JVMs that have been identified with a potential memory leak problem:

1. Monitor memory for potential JVM memory leaks.
2. Enable automatic leak detection.
3. Start an on demand capture session.
4. Detect and troubleshoot leaking conditions.

Monitor Memory for Potential JVM Leaks

Use the **Node** dashboard to identify the memory leak. A possible memory leak is indicated by a growing trend in the heap as well as the old/tenured generation memory pool.

An object is automatically marked as a potentially leaking object when it shows a positive and steep growth slope.

The **Automatic Memory Leak** dashboard shows:

- **Collection Size**—The number of elements in a collection.
- **Potentially Leaking**—Potentially leaking collections display as red. You should start diagnostic sessions on potentially leaking objects.
- **Status**—Indicates if a diagnostic session has been started on an object.
- **Collection Size Trend**—A positive and steep growth slope indicates a potential memory leak.

To identify long-lived collections, compare the JVM start time and Object Creation Time.

If no captured collections display, ensure that you have the correct configuration for detecting potential memory leaks.

Enable Memory Leak Detection

Memory leak detection is available through the **Automatic Leak Detection** feature. Once the **Automatic Leak Detection** feature is turned on and a capture session has been started, AppDynamics tracks all frequently used collections. Therefore, using this mode results in higher overhead.

1. Turn on **Automatic Leak Detection** mode only when a memory leak problem is identified
2. Click **Start On Demand Capture Session** to start monitoring frequently used collections and detect leaking collections.
3. After you identify and resolve the leak, turn the capture session and the leak detection modes off.
4. Start diagnosis on one individual collection at a time to achieve optimum performance.

Troubleshoot Memory Leaks

After detecting a potential memory leak, troubleshooting the leak involves performing these three actions:

- Select the Collection Object that you want to monitor
- Use Content Inspection
- Use Access Tracking

Select the Collection Object to Monitor

On the Automatic Leak Detection dashboard, right-click the class name and click **Drill Down**.

For performance reasons, start the troubleshooting session on a single collection object at a time.

Use Content Inspection

Content Inspection identifies which part of the application the collection belongs to so that you can start troubleshooting. It allows monitoring histograms of all the elements in a particular collection.

Enable Automatic Leak Detection by starting an On Demand Capture Session, select the object you want to troubleshoot, and then follow the steps listed below:

1. Click the Content Inspection tab.
2. Click **Start Content Summary Capture Session** to start the content inspection session.
3. Enter the session duration. Allow at least 1 – 2 minutes for data generation.
4. Click **Refresh** to retrieve the session data.
5. Click on the snapshot to view details about an individual session.

Use Access Tracking

Use Access Tracking to view the actual code paths and business transactions accessing the collections object.

As described above in Workflow to Troubleshoot Memory Leaks, enable Automatic Leak Detection, start an On Demand Capture Session, select the object you want to troubleshoot, and then follow the steps listed below:

1. Select the Access Tracking tab
2. Click **Start Access Tracking Session** to start the tracking session.
3. Enter the session duration. Allow at least 1-2 minutes for data generation.
4. Click **Refresh** to retrieve session data.
5. Click the snapshot to view details about an individual session.

The troubleshooting information pane shows the Java stack trace associated with the session. By default, the stack trace is shown to a depth of 10 lines. If you would like to temporarily increase the number of lines captures, you can use the `maximum-activity-trace-stack-depth` Java Agent property described on App Agent Node Properties Reference.

Increasing the stack trace depth can consume a significant amount of system resources. You must remove the property or set it back to the default value of 10 after you have captured the desired information.

Java Memory Thrash

Memory thrash is caused when a large number of temporary objects are created in very short intervals. Although these objects are temporary and are eventually cleaned up, the garbage collection mechanism may struggle to keep up with the rate of object creation. This may cause application performance problems. Monitoring the time spent in garbage collection can provide insight into performance issues, including memory thrash.

For example, an increase in the number of spikes for major collections either slows down a JVM or indicates potential memory thrash. Use object instance tracking to isolate the root cause of the memory thrash. To configure and enable object instance tracking, see Object Instance Tracking for Java.

AppDynamics automatically tracks object instances for the top 20 core Java (system) classes and the top 20 application classes.

The Object Instance Tracking subtab provides the number of instances for a particular class and graphs the count trend of those object in the JVM. It provides the shallow memory size (the memory footprint of the object and the primitives it contains) used by all the instances.

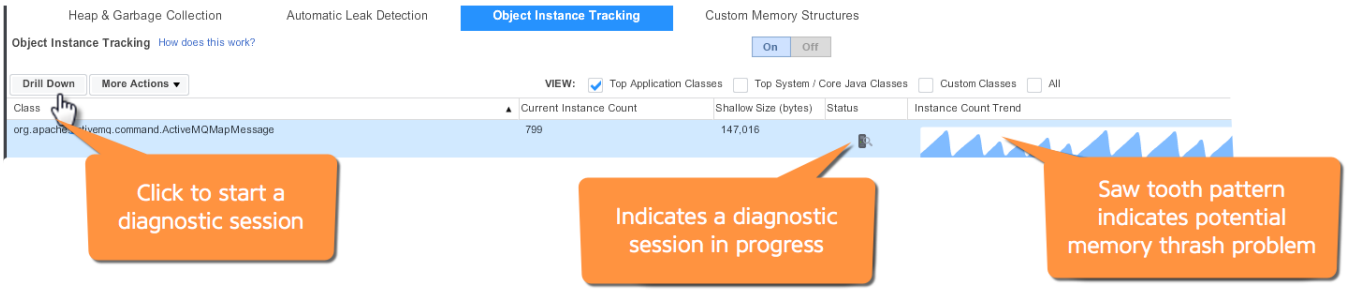
Analyze Memory Thrash

Once a memory thrash problem is identified in a particular collection, start the diagnostic session by drilling down into the suspected problematic class.

Select the class name to monitor and click **Drill Down** at the top of the Object Instance Tracking dashboard or right-click the class name and select the **Drill Down** option.

> For optimal performance, trigger a drill-down action on a single instance or class name at a time.

After the drill down action is triggered, data collection for object instances is performed every minute. This data collection is considered to be a diagnostic session and the Object Instance Tracking dashboard for that class is updated with this icon , to indicate that a diagnostic session is in progress.



The Object Instance Tracking dashboard indicates possible cases of memory thrash. Prime indicators of memory thrash problems indicated on the Object Instance Tracking dashboard are:

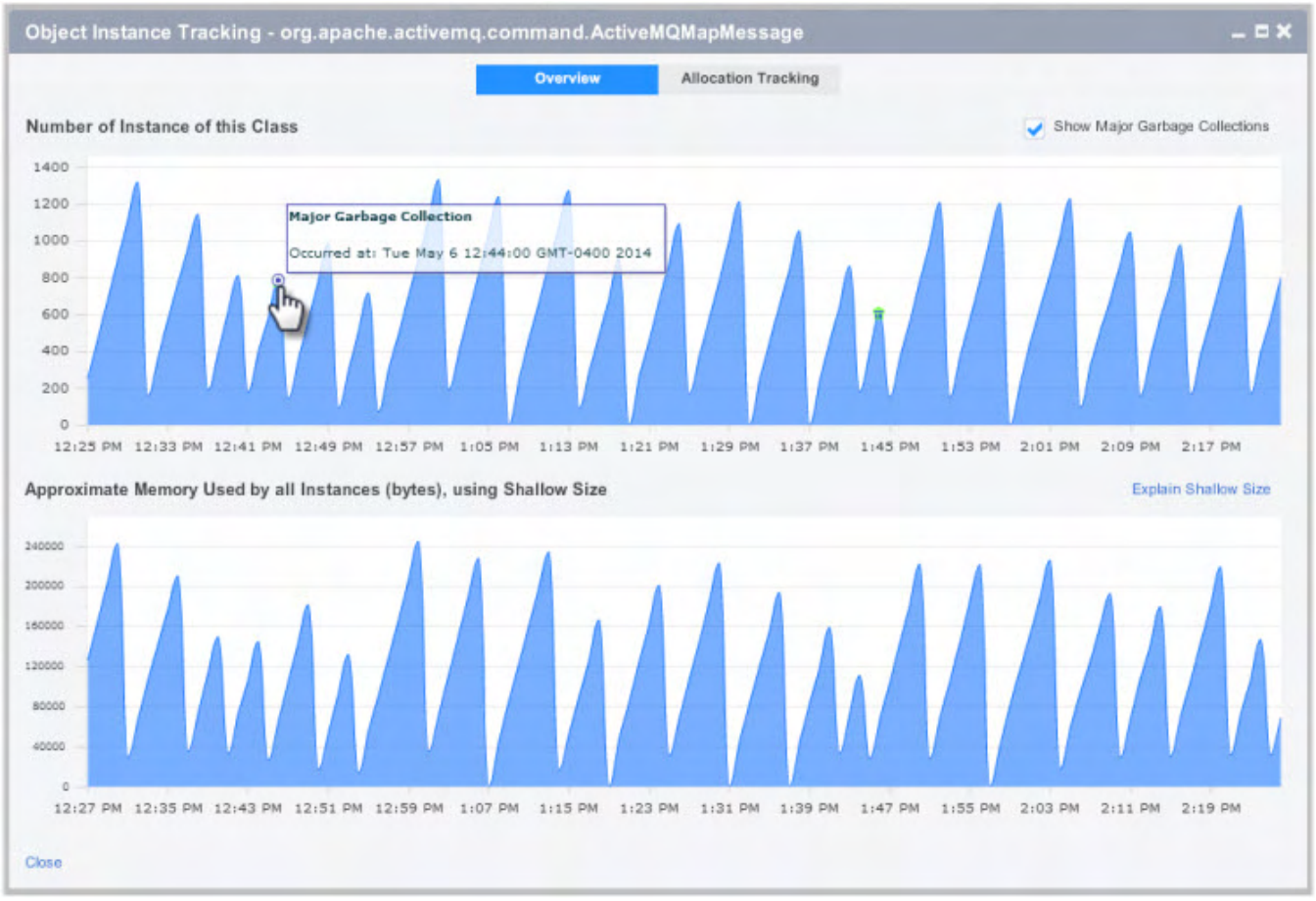
- **Current Instance Count:** A high number indicates the possible allocation of a large number of temporary objects.
- **Shallow Size**: Is the approximate memory used by all instances in a class. A large number for shallow size signals potential memory thrash.
- **Instance Count Trend:** A saw wave is an instant indication of memory thrash.

If you suspect you have a memory thrash problem at this point, then you should verify that this is the case. See To verify memory thrash.

Verify Memory Thrash

Select the class name to monitor and click **Drill Down** at the top of the Object Instance Tracking dashboard. On the Object Instance Tracking window, click **Show Major Garbage Collections**.

The following Object Instance Tracking Overview provides further evidence of a memory thrash problem.



If the instance count doesn't vary with the garbage collection cycle, it is an indication of a potential leak and not a memory thrash problem. See Java Memory Leaks.

Troubleshoot Java Memory Thrash Using Allocation Tracking

Allocation Tracking tracks all the code paths and those business transactions that are allocating instances of a particular class. It detects those code path /business transactions that are creating and throwing away instances.

To use allocation tracking:

1. Using the **Drill Down** option, trigger a diagnostic session.
2. Click the Allocation Tracking tab.
3. Click **Start Allocation Tracking Session** to start tracking code paths and business transactions.
4. Enter the session duration and allow at least 1 to 2 minutes for data generation.
5. Click **Refresh** to retrieve the session data.
6. Click a session to view its details.
7. Use the Information presented in the Code Paths and Business Transaction panels to identify the origin of the memory thrash problem.

Monitor Java Object Instances

If the application uses a JRE (rather than a JDK), use these steps to enable object instance tracking:

1. Ensure the tools.jar file is in the `jre/lib/ext` directory.
2. On the Node Dashboard, click the Memory tab.
3. On the Memory tab, click the Object Instance Tracking subtab.
4. Click **On** and then **OK**.

See Object Instance Tracking for Java.

Code Deadlocks for Java

By default, the Java Agent detects code deadlocks. You can find deadlocks and see their details using the Events list or the REST API.

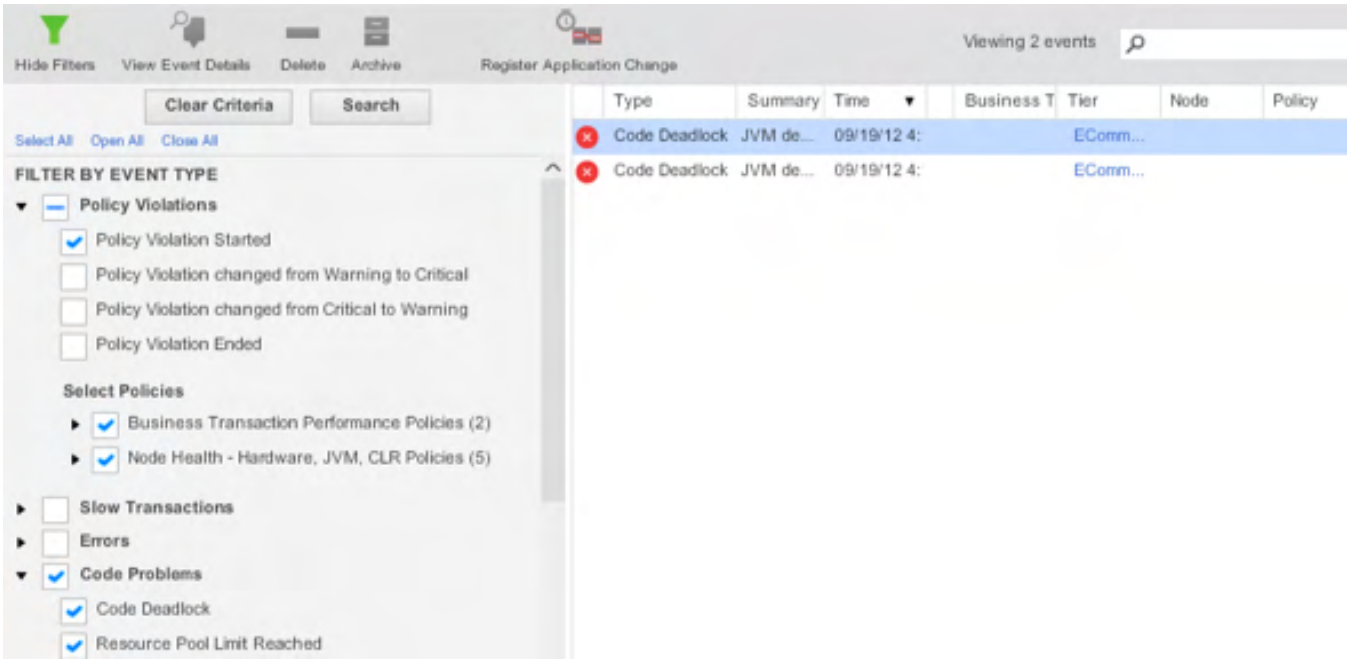
Code Deadlocks and Their Causes

In multi-threaded development environments, it is common to use more than a single lock. However, sometimes deadlocks will occur. Here are some possible causes:

- The order of the locks is not optimal
- The context in which they are being called (for example, from within a callback) is not correct
- Two threads may wait for each other to signal an event

Finding Deadlocks Using the Events List

Select **Code Problems** (or just **Code Deadlock**) in the Filter By Event Type list to see code deadlocks in the Events list. The following list shows two deadlocks in the ECommerce tier.



To examine a code deadlock, double-click the deadlock event in the events list and then click the Code Deadlock Summary tab. Details about the deadlock are in the Details tab. See Monitor Events.

Find Deadlocks Using the REST API

You can detect a DEADLOCK event-type using the AppDynamics REST API. For details, see the example Retrieve event data.

Thread Contention

Works with:

Thread contention arises when two or more threads attempt to access the same resource at the same time. This page describes how AppDynamics helps you diagnose and resolve thread contention issues.

Performance Issues Resulting from Thread Contention

Multithreaded programming techniques are common in applications that require asynchronous processing. Although each thread has its own call stack in such applications, threads may need to access shared resources, such as a lock, cache, or counter. See Enabling Thread Correlation.

While synchronization techniques can help to prevent interference between threads in such scenarios, they may nevertheless compete for access to shared resources. This can result in application performance degradation or even data integrity issues.

AppDynamics can help you identify and resolve problems relating to thread contention in business transactions and service endpoints. See Trace Multithreaded Transactions for Java.

Thread Contention Detection

AppDynamics detects thread contention based on the thread state of the instrumented application.

It identifies these block or waiting states in the JVM:

- Acquiring a lock (`MONITOR_WAIT`)
- Waiting for a condition (`CONDOR_WAIT`)
- Sleeping (`OBJECT_WAIT`)
- A blocking I/O operation

The `OBJECT_WAIT` state is triggered when the application makes one of the following calls:

- `Thread.sleep`
- `Object.wait`
- `Thread.join`
- `LockSupport.parkNanos`
- `LockSupport.parkUntil`
- `LockSupport.park`

The Controller alerts you to possible thread contention problems in the **Potential Issues** pane of the Business Transaction Flow Map. From there, you can use the browser to access additional information about blocked and waiting threads in business transactions or service endpoints, and determine the cause of the performance problem.

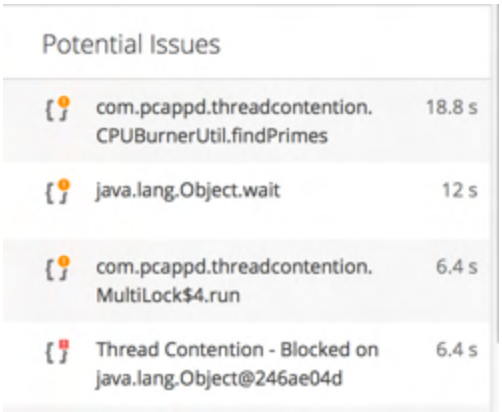
The following sections explain how you use the browser to surface contention information for business transaction and service endpoints.

Watch the Video: Monitoring and Analyzing Java Thread Contention

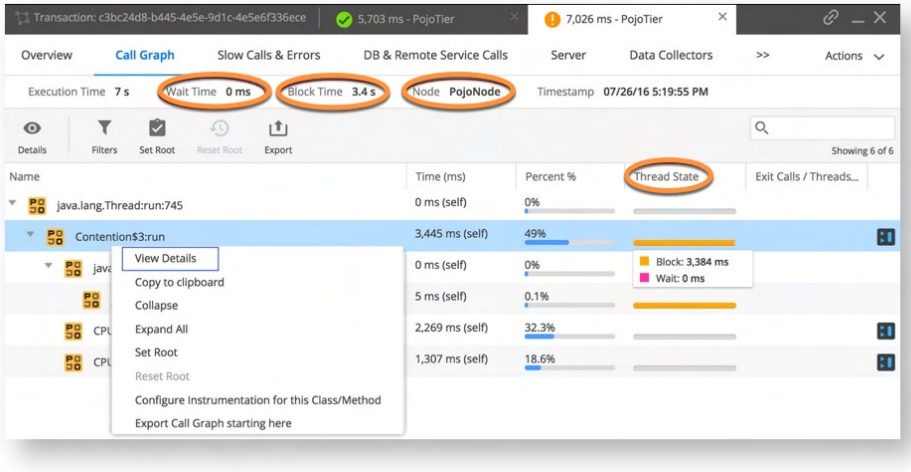
Thread Contention in Transaction Snapshots

To view information about thread contention:

1. In the transaction snapshot navigation page, look for items labeled as *Thread Contention* issues in the **Potential Issues** pane. The time column indicates blocked or wait time:



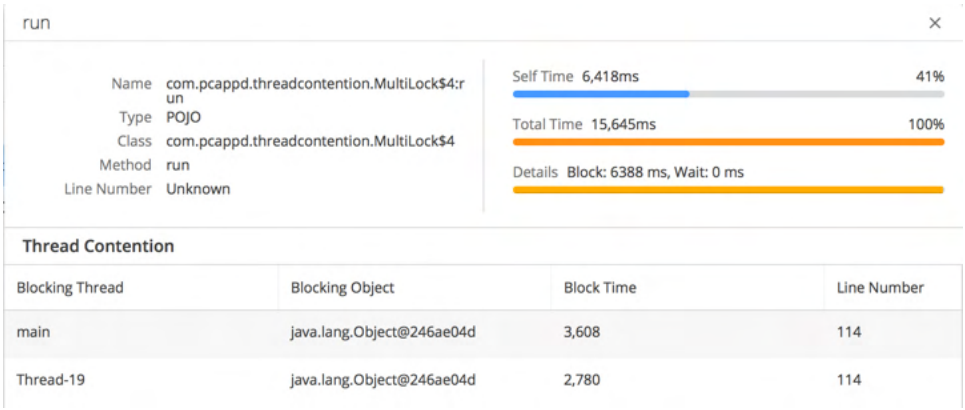
2. To display more information about the blocked method, click the thread contention item and select **Drill Down into Call Graph**:



The call graph shows the following information relevant to thread contention:

- In the **Call Graph** header, **Wait Time** and **Block Time** indicate aggregate measures for the thread in one segment of the business transaction.
- In the **Call Graph** header, **Node** specifies the name of the node hosting the contending threads, `PojoNode` in the example above.
- The **Time** column indicates the total self time for the method.
- The **Percent%** column shows the amount of time spent in the method as a percentage of overall time for the thread.
- The **Thread State** Column indicates the degree of thread contention issues for the method. Gray means no problems; yellow to red shading signals the severity of contention problems.
  (When you hover over the bar, a breakdown of the elements that make up the thread state is shown: This includes Block time and Wait time by default. To include Cpu Time in the Thread State detail, Dev mode must be enabled.)

3. Right-click on any method with a thread state that indicates block or wait times and select **View Details.** The Thread Contention details pane appears:



The Thread Contention details pane displays the name of the blocked method in the top left corner and adds the following information in the Threa d Contention table:

Element	Meaning
Blocking Thread	The thread holding a lock on the blocking object.
Blocking Object	The object that the blocked thread is waiting to access.
Block Time	The amount of time waiting to access the object.
Line Number	The line number in the blocked method where the blocking object is being accessed. With respect to the example above, `run` is attempting to access a locked object at line 114.

The order in which blocking threads are shown in the table is not significant; it does not imply a call order or time sequence.

In development mode, AppDynamics reports explicit locks such as `java.util.concurrent.locks.ReentrantLock` as Wait Time instead of Block Time in the Thread Details Call Graph view. Take this into consideration when monitoring business transactions and analyzing performance related to lock contentions.

Thread Contention in Service Endpoints

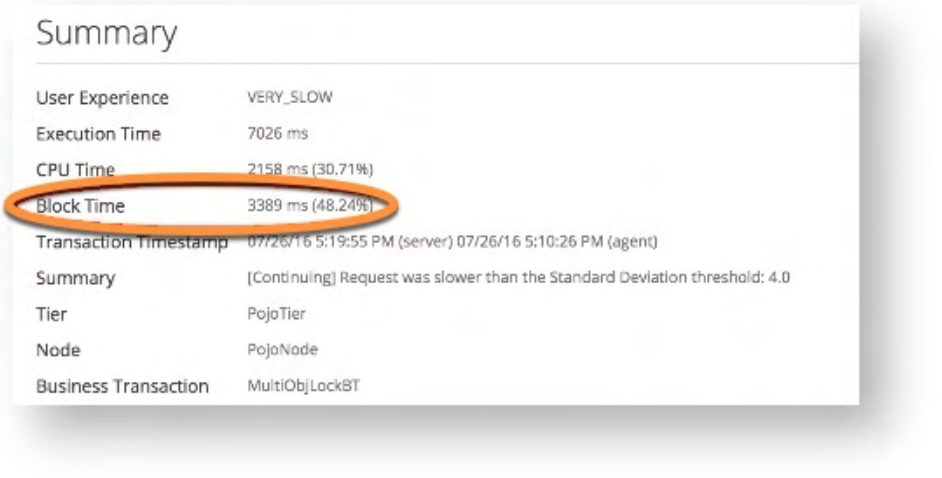
You can view thread contention information for service endpoint methods in AppDynamics. Call graphs identify service endpoint methods with this icon:  .

Select **More > Service Endpoints** from the menu bar to view thread contention information by service endpoint.

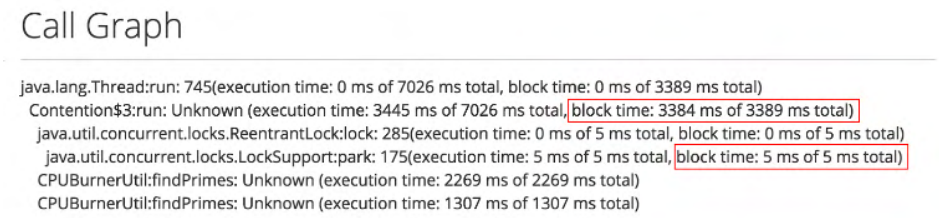
Export Contention Information

When you export the Call Graph for a Business Transaction, AppDynamics includes Transaction Contention information.

- The Summary pane includes **Block Time** data: the block time specified is the sum of all block times for the blocked methods shown in the CallGraph pane.



- The Call Graph pane lists block time by method:



Event Loop Blocking in Node.js

You can use process snapshots to examine Node.js event loop activity and identify functions with high CPU times that are blocking the event loop.

Latency in Node.js Event Loops

The event loop of a Node.js process is a single thread that polls for incoming connections and executes all application code. When a Node.js request makes a call to an external database, remote service or the filesystem, the event loop automatically directs the application's control flow to some other task, including other connections or callbacks.

CPU-intensive operations block the event loop, preventing it from handling incoming requests or finishing existing requests. A CPU-intensive operation in one business transaction may cause slowness in other business transactions.

Process Snapshots In AppDynamics

A process snapshot describes an instance of a CPU process on an instrumented Node.js node. It generates a process-wide flame graph for a Node.js process over a configurable time range.

Process snapshots provide visibility into the Node.js event loop across all business transactions for the duration of the process snapshot. Process snapshots are useful when the main troubleshooting tools (such as, business transaction snapshots) are inconclusive because the source of latency is a CPU-intensive operation in another business transaction. You can use lists of process snapshots to identify which functions have high CPU times. From the list, you can select and examine process snapshots to identify exactly which functions in your code are blocking the CPU.

For a given Node.js node or tier, you can access the list of process snapshots from the Process Snapshots tab of the node or tier dashboard. You can filter the process snapshot list to display only the snapshots that you are interested in. You can filter by execution time, whether the snapshot is archived, and the GUID of the request. If you access the list from the tier dashboard, you can also filter by node.

For more information on how process snapshots are generated and how to configure them, see Manage Node.js Process Snapshots.

To learn how process snapshots and business transaction snapshots are created, see Process Snapshots and Business Transaction Snapshots.

Process snapshots persist for 14 days, unless you archive them, in which case they are available forever.

A process snapshot contains these tabs:

- Overview
- Flame Graph
- Call Graph
- Allocation Call Graphs
- Hot Spots

Overview

Summarizes the snapshot. Contents vary based on the available information.

Usually contains at least the total execution time, tier and node of the process, timestamp, slowest method and request GUID.

Flame Graph

Provides a visualization of each stack frame’s frequency on the CPU over the duration of a process snapshot. The frame’s position relative to the bottom-most stack depicts the call-stack depth.

The flame graph contains the same information as the call graph, but allows you to quickly spot methods that are consuming more CPU resources relative to others.

The method corresponding to the stack frame on the top edge of the flame graph represents the method’s CPU resource consumption frequency.

To identify long-running CPU executions, look for long horizontal cells on the top edge of the flame graph.

A healthy Node.js process has minimal CPU-blocking activity; correspondingly, a flame graph for a healthy Node.js process has minimal long, horizontal cells along the top edge of its flame graph. See The Flame Graph.

Call Graph

Shows the total execution time and the percentage of the total execution time of each method on the process's call stack. The numbers at the ends of the methods are the line numbers in the source code. You can filter out methods below a certain time to simplify the graph and isolate the trouble spots.

The **Time** and **Percentage** columns identify which calls take the longest time to execute.

To see more information about a call, select the call and click **Details**.

Allocation Call Graph

Available only for process snapshots that are collected manually. See Manage Node.js Process Snapshots.

Shows the amount and percentage of the memory allocated and not freed by each method on the process's call stack during the process snapshot. You can use the **Method Size** slider to configure how much memory a method must allocate to be displayed in the allocation call graph. You can also filter out methods that consume less than a certain amount of memory to simplify the graph and isolate the trouble spots.

The **Size** and **Percentage** columns identify which calls consume the most memory.

The agent cannot report allocations made prior to the beginning of the allocation snapshot.

The allocation reported in the snapshot is the memory that is still referenced when the snapshot ends: memory allocated during the snapshot period minus memory freed during the snapshot period.

For more information about a call, select the call and click **Details**.

Hot Spots

This tab displays the calls by execution time, with the most expensive calls at the top. To see the invocation trace of a single call in the lower panel, select the call in the upper panel.

Use the **Method Time** slider in the upper right corner to configure how slow a call must be to be considered a hot spot.

Manage Node.js Process Snapshots

Related pages:

- Event Loop Blocking in Node.js
- Node.js Metrics
- Process Snapshots and Business Transaction Snapshots

This page describes how process snapshots are generated and viewed.

Automatic Process Snapshot Generation

When a business transaction snapshot is triggered by periodic collection or by a diagnostic session, a ten-second process snapshot is automatically started. By default, the agent starts no more than two process snapshots per minute automatically, but this behavior is configurable.

You can also start process snapshots manually on demand. See Collect Process Snapshots Manually.

Configure Automatic Collection

You can configure automatic process snapshot collection using these settings:

- `processSnapshotCountResetPeriodSeconds`: Frequency, in seconds, at which the automatic process snapshot count is reset to 0; default is 60 seconds.
- `maxProcessSnapshotsPerPeriod`: Number of automatic process snapshots allowed in `processSnapshotCountResetPeriodSeconds` seconds; default is 2 snapshots.
- `autoSnapshotDurationSeconds`: Duration of an automatically-generated process snapshot; default is 10 seconds.

To configure these settings, add them to the require statement in your application source code as described in Install the Node.js Agent. Then stop and restart the application.

Collect Process Snapshots Manually

If you want to generate some process snapshots now, you can start them manually.

1. Navigate to the dashboard for the tier or node for which you want to collect process snapshots.
2. Click the Process Snapshots tab.
3. Click **Collect Process Snapshots**.
4. If you are in the **Tier** dashboard, select the node for which you want to collect snapshots from the **Node** dropdown. If you are in the **Node** dashboard, you can only set up snapshot collection for that node.
5. Enter how many seconds you want to collect process snapshots for this node. The maximum is 60 seconds.
6. Click **Create**.

The agent collects process snapshots for the configured duration. Process snapshots that are started manually include an allocation call graph that shows how much memory has been allocated and not freed during the period recorded by the snapshot.

Process Snapshots and Business Transaction Snapshots

Related pages:

- Transaction Snapshots
- Manage Node.js Process Snapshots

This page explains the relationship between transaction snapshots and process snapshots created by the Node.js Agent.

V8 Sampler

Node.js is built on the V8 JavaScript engine, which includes a code sampler.

The Node.js Agent uses the V8 sampler to create process-wide process snapshots, which contain call graphs of the methods on the Node.js process's call stack.

Call Graph Data in Snapshots

Call graph data displays in business transaction snapshots as well as process snapshots.

When you view a business transaction snapshot, the displayed call graph specific to the transaction instance is derived from the concurrent process snapshot call graph.

When you view a process snapshot, the complete call graph of all the business transactions executed while the process snapshot was captured is displayed.

The call graph in a business transaction snapshot displays a view of the data from a concurrent process snapshot that is filtered to display only time in methods attributable to the specific business transaction. It is a subset of the concurrent process snapshot call graph.

For this reason, you might see an execution time for a method in a business transaction call graph that is less than the execution time for the same method in the concurrent process snapshot call graph. This would indicate that some calls to that method were made outside the context of the business transaction instance captured by the transaction snapshot.

The summary tab of a transaction snapshot includes a link to the process snapshot that was taken during the time covered by the transaction snapshot.

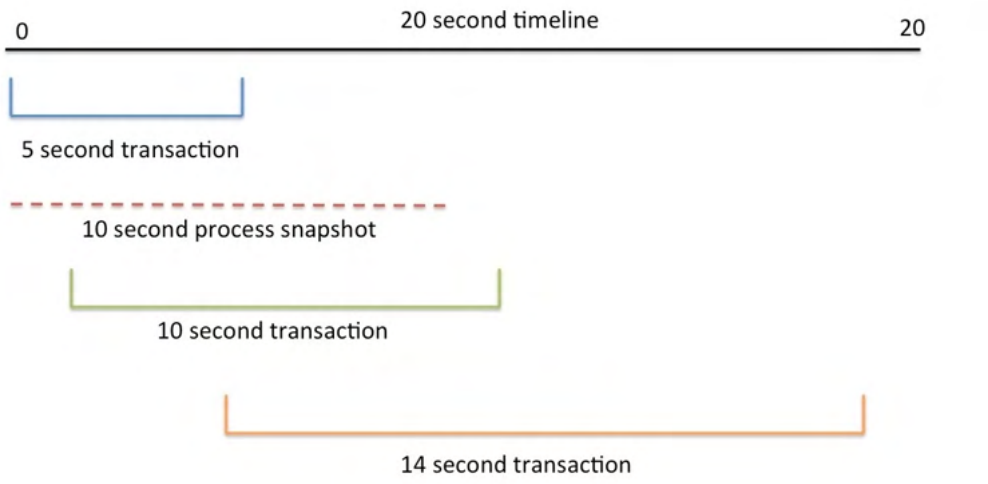
Business Transaction Snapshots Trigger Process Snapshots

To provide call graph data associated with business transaction snapshots, the agent starts a ten-second process snapshot whenever it starts a business transaction snapshot that is triggered by periodic collection or a diagnostic session if there is no existing process snapshot in progress for the current process. Process snapshots do not overlap. Periodic collection means that a business transaction is collected at periodic intervals, by default every ten minutes, but configurable. Diagnostic session means that either the agent has detected a pattern of possible performance issues and automatically started capturing transaction snapshots or a human has manually started a diagnostic session for the same reason.

Concurrent Business Transaction and Process Snapshots

The result presented is a process snapshot that ran concurrently with a business transaction. How well the two snapshots line up depends on the relative durations and start times of the transaction and the process snapshots.

In the scenario sketched below, all of the five-second blue transaction's calls, and most of the 10-second green transaction's calls are captured by a 10-second process snapshot, but only the about half of the 14-second orange transaction snapshot's calls.



If you find that your business transactions are running longer than your process snapshots, you can increase the default length of a process snapshot in the `autoSnapshotDurationSeconds` setting in the require statement.

Information Points

This page describes information points in AppDynamics. You can use information points to define custom metrics based on methods you configure. Information points are similar to data collectors; while data collectors show application data in the context of a business transaction, information points reflect data state across all invocations of a method, independent of business transactions. You can also apply computations to the values, for example, representing the sum or average for a method return value or input parameter.

To create, edit, or delete information points, you need the **Configure Information Points** permission.

Information Point Data: Code Metrics and Business Metrics

When you configure an information point, you automatically receive the KPI metrics (called *code metrics*) for the information point method.

The code metrics are:

- Total call count
- Calls per minute count
- Errors per minute
- Average response time

You can supplement the KPI metrics with custom business metrics for the information point.

Business metrics reflect the value of runtime data, such as the method parameter, return value, or a value captured by getter chain on the object on which the identified method was invoked. The business metric value represents either the sum or average of the values of the code point you identify as the information point.

Information points can give you significant insight into how the performance of an application corresponds to business performance. For example, depending on the nature of your application, you could use it to resolve business questions such as:

- What is the average value of the credit card total?
- How many credit cards did my application process in a certain time period, regardless of the business transaction?
- What was the average time spent processing a credit card transaction?

A example of a practical use of an information point is ignored exceptions. Exceptions, especially one that occurs frequently, can contribute to CPU spikes in a JVM. If you configure the exception to be ignored in AppDynamics, for example, if it is generated in the underlying application framework and does not have a direct bearing on your application performance, it may not be readily evident to you when the exception is affecting your application. An information point that counts the exception occurrence can help you identify the additional overhead.

Creating Information Points

To create information points, see Java and .NET Information Points and PHP Information Points.

Viewing Information Point Data

You can view information point values in various places in the Controller UI, including the following:

- Information Points page: The primary page for viewing and administering information points is the **Information Points** page. From there, click on an information point to view a histogram of the metrics for the information point for the selected time period.
- Metric Browser: The metrics also appear in the Metric Browser under the Information Points folder.
- REST API: Business metrics can be accessed from the AppDynamics REST API.
- Business Transaction Snapshots: Information point data does not appear in transaction snapshots by default, but you can configure it to appear by setting the `enable-info-point-data-in-snapshots` node property to `true`. When the `enable-info-point-data-in-snapshots` node property is set, information point calls appear in the User Data section of the snapshot. See App Agent Node Properties.

Java and .NET Information Points

Applies to:

This page describes how to create information points for Java and .NET applications. For PHP-based information points, see PHP Information Points.

Before Starting

Ensure that you have the ability to restart the application while minimizing user impact. To apply the configuration for a new information point for .NET and JVM 1.5 applications, you must restart the application server.

You must be logged in to the Controller as a user with Configure Information Points permissions. See Create and Manage Custom Roles.

Create an Information Point from a Transaction Snapshot

You can create an information point from the call graph of a transaction snapshot that includes the method.

When you locate an invocation of the method in the call graph, right-click it and select the option to configure instrumentation on this class or method. Your selection creates an information point

Create and Manage Information Points Manually

You can create, view, and modify information points on the **More > Information Points** page.

To create an information point, you must specify the class and method for which you want to collect information, along with other identifying information. For details on using the UI to configure code matching for an information point, see Data Collectors.

Information points are evaluated based on the execution of the method on which they are placed. Any match conditions apply to the state of the data at that time.

To collect business metrics from the method, from the **Edit Information Point** page, select **Add** to add a custom metric for the information point. You can configure and customize the type of data you want to collect for business metrics. Select **Save** when you are finished.

PHP Information Points

This page describes how to define information points for PHP applications.

Information points reflect key performance metrics and custom metrics for methods and code data points that you configure.

About PHP Information Points

To create an information point for PHP:

1. Define the information point using JSON notation.
2. In the definition, specify the method for the information point
3. Optionally, specify a point in the code that you want to capture as a custom metric, such as a parameter value or return value.
4. Navigate to **More** > **Information Points** and add an information point.
5. Choose **PHP** as the agent type and then paste the definition into the information point text box.

The elements of the definition correspond to configuration options available in the **Add Information Point** dialog for Java or .NET applications.

However, for PHP, the information point is defined in JSON syntax, and there are a few limits on configuration options. For one, class matching is limited to classname-based matches only. (Superclass matching or annotation matching are not available.) Also, you can only define a single match condition on a matched method. (A match condition refines the match of a method by testing a parameter or return value.)

If you do not configure a custom metric, the information point captures the generic KPIs for a matched method (response time, calls and calls per minute, errors an errors per minute). Custom metrics extend the information point by capturing the return or parameter value of a method, or the value of an invoked object.

Before Starting

To create information points in the Controller UI, you need to be logged in as a user with **Configure Information Points** permissions. See Create and Manage Custom Roles.

Method Match Conditions

A match definition specifies the method associated with the value you want to capture as an information point. The match definition must specify the method, but it may also specify the containing class and a refining match condition. The match condition can test the value of a method parameter, return value, or the return value of an invoked object.

Use this template to create a match definition:

```
{
  "probe": {
    "phpDefinition": {
      "classMatch": {
        "type": "MATCHES_CLASS",
        "classNameCondition": {
          "type": "EQUALS",
          "matchStrings": [
            "<class name>"
          ]
        }
      },
      "methodMatch": {
        "methodNameCondition": {
          "type": "EQUALS",
          "matchStrings": [
            "<method name>"
          ]
        }
      }
    }
  }
}
```

Edit the JSON objects in the following way:

- `classMatch` is optional. If you do not want to specify a class to match, delete the `classMatch` object from the `phpDefinition` object.
- If you are using the class match, set the value for the `classNameCondition` match string to the class name.
    - The class must be defined as a separate file from the file in which it is instantiated and invoked.
    - If your class names are defined in a namespace, escape the backslashes in the namespace with an escape backslash.
- `methodMatch` is required. Set the value for the `methodNameCondition` match string to the method name.

The following values are required for PHP information points:

- `classMatch` type: `MATCHES_CLASS`
- `classNameCondition` type: `EQUALS`
- `methodNameCondition` type: `EQUALS`

For example, the following JSON creates an information point on a class for which the class name equals `CheckoutManager` and the method name `processPayment`:

```
{
  "probe": {
    "phpDefinition": {
      "classMatch": {
        "type": "MATCHES_CLASS",
        "classNameCondition": {
          "type": "EQUALS",
          "matchStrings": [
            "Bundy\\ShoesBundle\\Entity\\CheckoutManager"
          ]
        }
      },
      "methodMatch": {
        "methodNameCondition": {
          "type": "EQUALS",
          "matchStrings": [
            "processPayment"
          ]
        }
      }
    }
  }
}
```

If creating an information point primarily to capture KPI metrics for the method, it is likely you will define the class for the method. However, if you are creating an information point to implement a code metric, you may only need to specify method name matching.

This example shows a method match that tracks how many times a method is called:

```
{
  "probe": {
    "phpDefinition": {
      "methodMatch": {
        "methodNameCondition": {
          "type": "EQUALS",
          "matchStrings": [
            "deleteCartItems"
          ]
        }
      }
    }
  }
}
```

Match Conditions

A match condition (defined by a `matchConditions` object) lets you refine the method matching criteria by specifying a match condition based on a parameter or return value or the value returned by an invoked object.

A match condition is optional, but there can be at most one for the information point. If you do not supply a match condition, the information point includes all invocations of the matched method.

The condition consists of a match type and the comparison operator that defines the data to be compared. The match types are:

- EQUALS
- NOT_EQUALS
- LT
- GT
- LE
- GE

- NOT
- AND
- OR

The comparison operator is an expression node that defines a left side (`lhs`) and a right side (`rhs`) of the operation. It contains:

- a type: `ENTITY`, `STRING`, or `INTEGER`
- a value, which is an `entityValue`, `stringValue`, or `integerValue`

If the type is `ENTITY`, the `entityValue` has one of the following types: `INVOKED_OBJECT`, `RETURN_VALUE`, or `PARAMETER`. If the type of the `entityValue` is `PARAMETER`, the `parameterIndex` indicates the 0-based numeric index of the parameter on which to match. So if the method takes two parameters, for example, the first being a string and second an integer, a `parameterIndex` of 0 matches against the string and a `parameterIndex` of 1 matches the integer value.

The information point in the following example only matches invocations of the `processPayment` method in which the second parameter (at parameter index 1) equals `VISA`.

```
{
  "probe": {
    "phpDefinition": {
      "methodMatch": {
        "methodNameCondition": {
          "type": "EQUALS",
          "matchStrings": [
            "processPayment"
          ]
        },
        "matchCondition": {
          "type": "EQUALS",
          "comparisonOp": {
            "lhs": {
              "type": "ENTITY",
              "entityValue": {
                "type": "PARAMETER",
                "parameterIndex": 1
              }
            },
            "rhs": {
              "type": "STRING",
              "stringValue": "VISA"
            }
          }
        }
      }
    }
  }
}
```

Metric Definitions

You can define custom metrics for data captured by an information point using the `metricDefinitions` object in your information point definition. Custom metrics appear in the Metric Browser and in the dashboard for the information point. You can create health rules based on a custom metric and retrieve custom metric values using the AppDynamics REST API.

There can be multiple custom metrics for a single information point. For example, one metric might be based on the average and one on the accumulated (sum) of the information point's values.

A `metricDefinitions` object consists of one or more definitions, each having the following structure:

- a name
- a rollup type (`AVERAGE` or `SUM`)
- data, which consists of a type, (`ENTITY`, `STRING`, or `INTEGER`) and a value (`entityValue`, `stringValue`, or `integerValue`)

  If the type is `ENTITY`, the `entityValue` has a type, which is `INVOKED_OBJECT`, `RETURN_VALUE`, or `PARAMETER`. If the type of the `entityValue` is `PARAMETER`, the zero-based `parameterIndex` indicates the parameter on which to base the match. A return value or parameter metric cannot be an array.

For example, the following `metricDefinitions` object defines two custom metrics: `VisaTotal`, which reports the sum of the Visa payments processed and `VisaAverage`, which reports the average value of the Visa payments processed.

```
"metricDefinitions": [
    {
      "name": "VisaTotal",
      "rollup": "SUM",
      "data": {
        "type" : "ENTITY",
        "entityValue": {
          "type": "RETURN_VALUE"
        }
      }
    },
    {
      "name": "VisaAverage",
      "rollup": "AVERAGE",
      "data": {
        "type" : "ENTITY",
        "entityValue": {
          "type": "RETURN_VALUE"
        }
      }
    }

  ]
```

Sample JSON Information Point Configuration

This is an example that you can copy and paste into the JSON text field in the PHP information point window. It produces two metrics, named `VisaTotal` and `VisaAverage`.

```
{
  "probe": {
    "phpDefinition": {
      "classMatch": {
        "type": "MATCHES_CLASS",
        "classNameCondition": {
          "type": "EQUALS",
          "matchStrings": [
            "CheckOutManager"
          ]
        }
      },
      "methodMatch": {
        "methodNameCondition": {
          "type": "EQUALS",
          "matchStrings": [
            "processPayment"
          ]
        },
        "matchCondition": {
          "type": "EQUALS",
          "comparisonOp": {
            "lhs": {
              "type": "ENTITY",
              "entityValue": {
                "parameterIndex": 1,
                "type": "PARAMETER"
              },
              "rhs": {
                "type": "STRING",
                "stringValue": "VISA"
              }
            }
          }
        }
      }
    },
    "metricDefinitions": [
      {
        "name": "VisaTotal",
        "rollup": "SUM",
        "data": {
          "entityValue": {
            "type": "RETURN_VALUE"
          },
          "type": "ENTITY"
        }
      },
      {
        "name": "VisaAverage",
        "rollup": "AVERAGE",
        "data": {
          "type": "ENTITY",
          "entityValue": {
            "type": "RETURN_VALUE"
          }
        }
      }
    ]
}
```

Configure Instrumentation

In AppDynamics, instrumentation refers to how app agents interact with your application software to gather performance data and report it back to the AppDynamics Controller. The first step to instrumenting your application is installing the app agents on the servers where the code runs.

App agents ship with default instrumentation settings that cover the most common types of application frameworks and programming patterns. If your requirements are more complex, you can customize the instrumentation behavior to suit your environment.

See Configure Instrumentation Overview for information on how AppDynamics applies instrumentation settings across your business application.

Access Instrumentation Settings

Navigate to **Configuration > Instrumentation** to customize:

- Transaction detection rules that define entry points for business transactions.
- Backend detection rules that define exit points for your business transactions.
- Error detection settings that define which transactions qualify as error transactions.
- Service endpoints to view metrics specific to services and transaction segments outside the context of a single business transaction.
- Diagnostic data collectors for an app agent to extract additional context about a specific transaction at transaction snapshot time. You can also configure data collectors for Application Analytics.
- Call Graph Settings to tune the amount of data the app agent collects for call graphs.
- JMX metrics that the Java agent collects for specific products and frameworks. See "Configure JMX Metric Rules" on Configure JMX Metrics from MBeans.
- Memory monitoring settings for agents that collect instance tracking data for specific classes or objects. See the following topics:
    - Object Instance Tracking for Java
    - Object Instance Tracking for .NET
    - Object Instance Tracking for Node.js
- Asynchronous transaction demarcators to set endpoints for asynchronous transactions.

Configuration Page Items

In addition to the instrumentation configuration options described on this page, there are many other ways to customize AppDynamics for your environment. For more information on the items on the **Configuration** page, review these pages:

- "Configure Thresholds" on Transaction Thresholds.
- "View Baselines" on Dynamic Baselines.
- Information Points
- Development Level Monitoring

If you have an End-User monitoring license, review these pages on **User Experience App Integration**:

- Correlate Business Transactions for Browser RUM
- Automatic Injection of the JavaScript Agent
- Assisted Injection

Configure Instrumentation Overview

The scope configuration model reduces the number of steps it takes to apply transaction detection rules to multiple tiers.

Under the scope configuration model, you can apply settings to groups of tiers, in addition to individual tiers and nodes. Groups of tiers are called scopes. Only transaction detection settings for applications created in AppDynamics >= 4.3 adhere to the scope model.

Live Preview Mode

You can interactively experiment with transaction detection rule configurations using Live Preview. Live Preview streams data from an active node, providing real-time data for the following configurations:

- All transaction detection rules that apply to a node. See Business Transaction Discovery Sessions.
- A single custom match rule applied to a node. See Custom Match Rule Live Preview.

The following limits apply to Live Preview:

- 100 sessions on the Controller
- 20 sessions on an account
- 10 sessions on a business application

Programmatic Expressions

You can specify the following types of programmatic expressions when configuring detection rules:

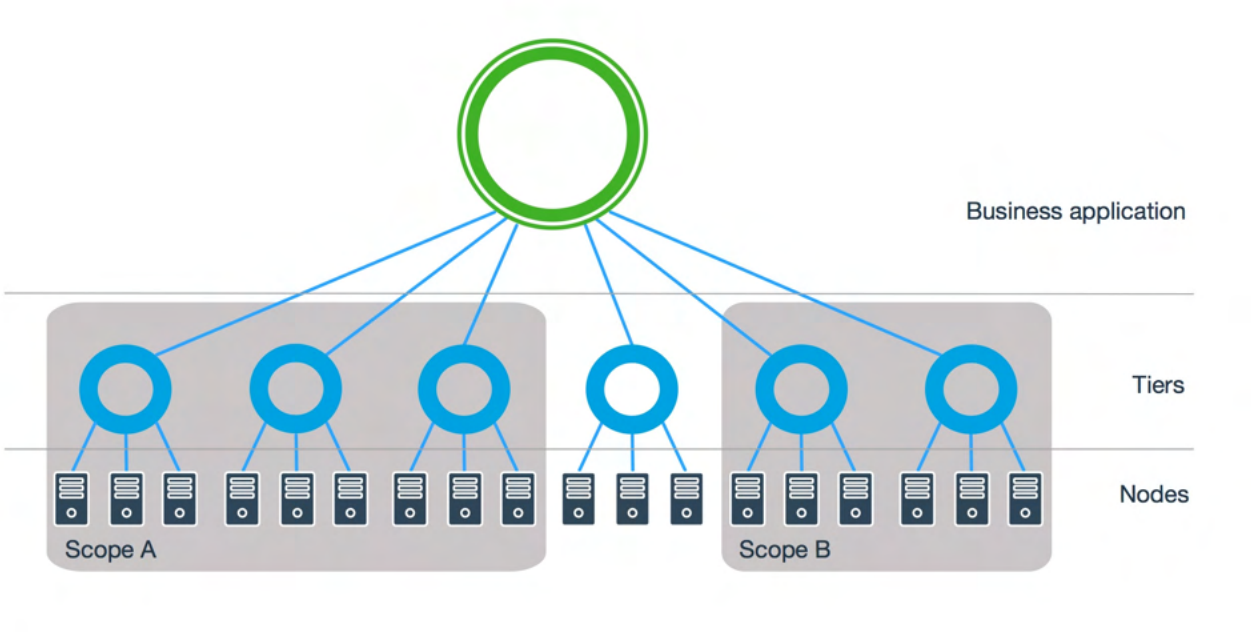
- Regular expression: A string or character search pattern
- Getter chains: Enable you to access data returned by methods in your application

Scope Configuration Model

The scope configuration model allows you to bundle multiple tiers into a scope, and apply transaction detection rules to that scope.

You can apply a rule to select tiers all at once. The Default Scope consists of all the tiers in the application.

In the diagram, you want to apply a rule to three of the six tiers in an application. Rather than manually applying the rule to each of the three tiers, you can bundle the tiers into a scope (Scope A) and apply the rule to the scope.



Creating a Custom Scope

You can create a custom scope by navigating to **Configuration > Instrumentation > Scopes**.

1. Click **Add**.
2. Enter a name for your scope.
3. In the **Include the following Tiers** dropdown, select one of these options:
    a. **All Tiers in the Application**: creates a scope that includes all the tiers in the application, except tiers that you specify. You can select tiers to exclude from the scope in the **Available Tiers** box and clicking the left-pointing arrow.
    b. **These Specific Tiers**: creates a scope that includes only tiers that you specify. You can select tiers to include in the scope in the **Availab le Tiers** box and clicking the left-pointing arrow.
4. Click **Save**.

To view the rules that are applied to a scope, select the scope in the Scopes tab. The rules applied to that scope display in the panel on the right.

Set the Scope for a Rule

When you create a new application, all rules for the application use the default scope. You can create additional scopes for that application, and choose which rules to apply to that scope. Each rule can only apply to one scope.

1. Click the Rules sub-tab.
2. Select the rule that you want to apply to a scope. The tiers that the rule is applied to are displayed in the panel on the right.
3. Click **Edit**.
4. Click **Change Scope**.
5. Select the scope that you want to apply this rule to.
6. Click **OK**.
7. Click **Save**.

To apply a rule to a scope, you need permissions to all the tiers in the scope. To apply a rule to the default scope, you need application-level permission.

Using Regular Expressions

Related pages:

- Transaction Detection Rules
- Custom Match Rules

Regular expressions display in various places in the AppDynamics configuration. These places include, for example, transaction detection rules, data collectors, EUM injection settings, health rules, and more. This page describes the use of regular expressions in AppDynamics.

Matching Guidelines

A match condition consists of a named property to match (such as a method name, Servlet name, URI, parameter, or hostname), a comparison operator, and a matching value. For complex match conditions, you can use a regular expression (often abbreviated to just regex).

Match rules are case sensitive. Also, matching is based on subsequence pattern matching. To match a complete string instead, you need to include "^" to match the start of the string and "$" to match the end of the string in your regular expression.

In the context of business transaction matching, the pattern does not include the protocol (e.g., "http://"), host, port, or query parameters in the URL. So for example, a URI of http://www.mysite.com:8000/account/settings.html?action=update would be matched by a business transaction regular expression with only "/account/settings.html".

Regular Expression Engines

The Java Agent uses Java libraries for regular expressions. See:

- Tutorial: http://download.oracle.com/javase/tutorial/essential/regex/index.html
- Javadoc: http://download.oracle.com/javase/1.5.0/docs/api/java/util/regex/Pattern.html

The .NET Agent uses the built in .NET regular expressions engine. See:

- MSDN: .NET Framework Regular Expressions

The Node.js Agent uses JavaScript regular expressions. See:

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp

The PHP Agent uses PHP's built-in PCRE regular expression engine and requires the same syntax, including delimiters (for example: /^Foo/). See:

- PCRE Manual

The Python Agent uses Python's regular expression syntax. See:

- https://docs.python.org/2/library/re.html

The Web Server Agent uses Perl regular expression syntax. See:

- http://www.boost.org/doc/libs/1_57_0/libs/regex/doc/html/boost_regex/syntax/perl_syntax.html

Regular Expression Examples

These examples show how to construct regular expressions to achieve different results.

Matching Non-Adjacent URL Segments

A typical use of regular expressions in the AppDynamics configuration is for business transaction custom match rules in which the expression is matched to a requested URI. In this context, it's common for an application's URI pattern to put information that would be useful for business transaction identification in different segments of the URI.

For example, given an example URL of `http://retailstore.example.com/store/jump/category/shoes/departments/view-all/cat630006p`, a business transaction might need to match on `/store/jump` and `all` to group user requests to view all of an available category.

A regular expression to match this case could be:

```
/store/jump.*\b?all\b
```

Matching Any Digit

Say you want to ignore numbers contained within a pattern. For example, consider the following URL examples:

- /group/1/session/

- /group/1/session/
- /group/31/session/
- /group/2/session/

Examples of matching regular expressions would be:

- ^/group/\d*/session/?$
- session
- /\d*/session/?$

Requiring a Digit

To group URLs that contain letters then numbers into one business transaction, such as the following:

- /aaa123.aspx
- /b1.aspx

You could use an expression such as the following:

```
/[a-z]+?[0-9]+?
```

Not matched would be a URL that does not have digits after the letters, for example: `/z.aspx`

Handling Letter Casing

Regular expression matching is performed in a case-sensitive manner. To match any case letters, you can use something similar to the following:

```
/[a-zA-Z]+?[0-9]+?
```

Or match in a case-insensitive manner using the `/` modifier. For example:

```
(?i)\w*cart\w*
```

Would match `addToCart` as well as `addTocart`.

Backend Discovery Rules

For an example of a JDBC backend regular expression, see the section JDBC with complex URLs in Example JDBC Backend Configuration.

You can find resources for testing your own regular expressions at http://www.regexplanet.com/advanced/java/index.html. There you can find regular expression test pages for many language engines, including Java, JS, .NET, PHP and more.

Performance Considerations

Although regular expressions are a powerful way to set AppDynamics configurations, you should consider the following to avoid performance issues:

- Do not use wildcard expressions unless absolutely needed. You do *not* need to use wildcard expressions (.*) before or after a match, and each wildcard regular expression results in approximately an order of magnitude performance impact.
- The number of configurations, the frequency the configuration is applied, and the content length of the target string: For example, if the configuration using a regular expression is applied in many places at high frequency for longer target strings, you might consider reducing the usage of the configuration or decrease the frequency of its application.

Using Getter Chains

Related pages:

- Transaction Detection Rules

Getter chains let you access method data in various contexts of the AppDynamics configuration. For example, getter chains enable you to name business transactions based on return values. This page describes how to use getter chains.

Getter Chains

You can use getter chains to:

- Create a new JMX Metric Rule and define metrics from MBean attributes. See Configure JMX Metrics from MBeans.
- Configure method invocation data collectors. See the Configuration Notes in Data Collectors.
- Define a new business transaction custom match rule that uses a POJO object instance as the mechanism to name the transaction. See POJO Entry Points.
- Configure a custom match rule for servlet entry points and name the transaction by defining methods in a getter chain. See "Split by POJO Method Call" on Split Servlet Transaction by Payload Examples.

As a best practice, you should use getter chains with simple getter methods only. Getter chains on processing-intensive methods, such as one that make numerous SQL calls, can result in degraded performance for the application and agent.

For example, the following shows a simple get method that returns a property for a class, such as `MyUser.Name`:

```
public class MyUser
{
   private String Name {get; set;}
   private String Url;
   public String GetUrl() {
      return this.Url;
   }
}
```

Special Characters in Getter Chains

Use the following special characters as indicated:

- Parentheses `( )` to enclose parameters
- Commas `,` to separate parameters
- Forward slashes `/` to separate type declarations from a value in a parameter
- Backslashes `\` to escape characters.
    - To search for a backslash `\`, escape the backslash with a backslash:

      ```
      GetString().Find(\\)
      ```

    - To split a string by a backslash `\`, escape the backslash in Java and within the Getter Chain:

      ```
      GetString.split(\\\\).[0]
      ```

- Dots `.` to separate methods and properties in the getter chain
- Dot `.` to represent "anything" must be escaped
- Curly braces `{ }` to delineate getter chains from static elements in custom expressions on HTTPRequest objects (including in the Java Servlet Transaction Naming Configuration window and in the Split Transactions Using Request Data tab of the servlet custom match and exclude rules)

Getter chains also treat spaces at the beginning or end of a string as special characters.

Escape Literal Characters

If the getter chain should treat a special character literally, escape it using a backslash. For parentheses, you only need to escape the closing parenthesis in a string.

- The following example shows the how to escape the dot in the string parameter.

  ```
  GetAddress().GetParam(a\.b\.c\.)
  ```

The agent executes `GetParam("a.b.c.")` on the result of `GetAddress()` and returns the value of the parameter.

- In the following example, the first dot is part of the string method parameter, which requires an escape character. The second and third dots don't require an escape character because they are used as method separators.

```
GetUser(suze\.smith).GetGroup().GetId()
```

- The following example shows how to escape the opening and closing parenthesis in a search for "()" within a string.

```
GetString.Find(\(\))
```

.NET Notes and Examples

These sections apply to getter chains used in .NET Agent configurations.

Escape backslashes within a getter chain for .NET with the double backslash.

Declare Parameter Types

The .NET Agent identifies parameter types as follows:

- Dictionaries, anything with property accessors, use a normal parameter set, which defaults to string.
- Arrays use single integers as parameters.

Therefore `0` means `string/0` if you have a hash, or anything that translates to `get_Item` (any kind of property). `0` means `int/0` if you have an array.

When your getter chain uses a method with parameters other than the default type, you need to declare the parameter type.

- The following example demonstrates how to declare the parameter types to resolve the overloaded `Susbstring()` method. The forward slash serves as the type separator.

```
GetAddress(appdynamics, sf).Substring(int/0, int/10)
```

  For instance, if `GetAddress(appdynamics, sf)` returns "303 2nd St, San Francisco, CA 94107", the full getter chain expression returns "303 2nd St".
- The following example shows how to declare the parameter types for a user-defined method that takes a float parameter, a boolean parameter, and an integer parameter. The forward slash serves as the type separator.

```
GetAddress(appdynamics, sf).MyMethod(float/0\\.2, boolean/true, boolean/false, int/5)
```

Access Indexed Properties and Dictionary Values

If your getter chain accesses a dictionary object, you can access the values using the following syntax:

- The following example returns the value for the key `suze.smith`.

```
UserDictionary.[string/suze\\.smith]
```

- The following example returns the value for the key suze.smith using the implied getter.

```
UserDictionary.get_Item(suze\\.smith)
```

Split by Character or Regular Expression Match in .NET Getter Chains

You can split values matched by character or as a string by matching a regular expression pattern. The result of a split operation is a string or character array that you can reference in your getter chain by an array index value.

The following examples illustrate how to use the character and string regular expression split operations in getter chains.

Split by Character

- The following chain splits a URL on the forward slash character. In this case, the first slash acts as a type separator. The getter chain returns the fourth item in the array.

```
GetUrl().Split(char[]//).[3]
```

  The agent returns "Search" when it applies the getter chain to the following URL: http://howdyworld.example.com/Search/Airfare
- In the following example, the split occurs on the forward slash character, and the result is the length of the resulting array.

```
GetUrl().Split(char[]//).Length
```

- This example illustrates a transaction splitting rule for URIs that use a semicolon delimiter. The getter chain splits on the forward slash, then splits the fourth element on the semicolon.

```
GetUri().Split(char[]//).[3].Split(char[]/;).[0]
```

  The agent returns `create-user` when it applies the getter chain to the following URL: `http://HowdyWorld.example.com/create-user;sessionid=BE7F31CC0235C796BF8C6DF3766A1D00?act=Add&uid=c42ab7ad-48a7-4353-bb11-0dfeabb798b5`

Split by Regular Expression

For more control, you can split values by string-based pattern matching. Pattern matching is particularly useful for situations that require complex matching, such as matching content within a request body.

The following example shows a getter chain that splits the value returned by `GetAddress()` and selects the seventh element in the resultant array:

```
GetAddress().Split(string/\\W+).[6]
```

Given an address such as `303 2nd St, San Francisco, CA 94107`, the example splits the value by word and references the sixth word in the array, `CA` in this case.

Compose Getter Chains for HTTP Requests

The .NET Agent requires special syntax for getter chains for HTTP Requests:

- For ASP.NET WebForms, MVC, and MVC WebAPI applications create getter chains based upon the `System.Web.HttpRequest` objects.
- For ASP.NET Core on the full framework, create getter chains based upon Microsoft.AspNetCore.Http.Internal.DefaultHttpRequest objects.

> If you have both ASP.NET and ASP.NET Core on the full framework apps in the same tier, you cannot use a getter chain to apply to both because the two frameworks use different objects to handle HTTP requests.

- Use the following syntax to delineate the boundaries of the getter chain:

```
${myobject.myproperty}
```

- The following example determines the user principal:

```
${Context.User.Identity.Name}
```

Places to use this syntax include:

- HTTP Request Data Collectors
- ASP.NET Transaction Detection custom expressions

Java Notes and Examples

The following sections apply to getter chains used in Java Agent configurations.

Values Passed in a Getter Chain

The value passed in a getter chain is always a string unless cast to another type.

The following cast types are supported:

- int
- float
- bool (the primitive Boolean value)
- boolean (a boxed boolean value; i.e. an object type that wraps a boolean)
- long
- object

The following section shows examples of how to refer to the types in parameters to getter chain methods. Notice that letter case is not important for the names of the types. Type casting is performed in a case-insensitive manner.

Java Getter Chain Examples

- Getter chain with integer parameters in the substring method using the forward slash as the type separator:

```
getAddress(appdynamics, sf).substring(int/0, int/10)
```

- Getter chain with various non-string parameter types:

```
getAddress(appdynamics, sf).myMethod(float/0.2, boolean/true, boolean/false, int/5)
```

- Getter chain with forward slash escaped; escape character needed here for the string parameter:

```
getUrl().split(\/) # node slash is escaped by a backward slash
```

- Getter chain with an array element:

```
getUrl().split(\/).[4]
```

- Getter chains that return Hashmap values:

```
get(object/myvalue).substring(int/0,int/10)
get(object/ACTION)
```

- Getter chain with multiple array elements separated by commas:

```
getUrl().split(\/).[1,3]
```

- Getter chain retrieves property values, such as the length of an array:

```
getUrl().split(\.).length
```

- Getter chain using backslash to escape the dot in the string parameter;
the call is getParam (a.b.c).

```
getAddress.getParam(a\.b\.c\.)
```

- In the following getter chain, the first dot requires an escape character because it is in a string method parameter (inside the parentheses). The second dot does not require an escape character because it is not in a method parameter (it is outside the parentheses).

```
getName(suze\.smith).getClass().getSimpleName()
```

The following getter chain is from a transaction splitting rule on URIs that use a semicolon as a delimiter.

```
getRequestURI().split(\/).[2].split(;).[0]
```

The call gets the URI using getRequestURI() and then splits it using the escaped forward slash. From the resulting array, it takes the third entry (as the split treats the first slash as a separator) and inserts what before the slash (in this case, nothing) into the first entry. Then it splits this result using the semicolon, getting the first entry of the resulting array, which in this case contains the API name.
For example, given the following URI:

```
/my-webapp/xyz;jsessionid=BE7F31CC0235C796BF8C6DF3766A1D00?act=Add&uid=c42ab7ad-48a7-4353-bb11-0dfeabb798b5
```

The getter chain splits on the API name, so the resulting split transactions are "API.abc", "API.xyz" and so on.

Tip: When using `string.split()`, remember that it takes a regex and you have to escape any special regex characters.

For example, if you want to split on the left square bracket ([):

```
Java syntax: split("\\[")
Getter chain syntax: split(\\\\[)
```

A Vertical Bar or Pipe (|) is a reserved character in Getter strings. For methods with a regular expression parameter, use Unicode as an alternative.

For example:

```
split(string/\\u007C).[0]   // return the first element of an string split by a Vertical Bar.
```

Blocklisted Classes or Packages and Methods of the Java Getter Chain

Certain packages and classes have been blocklisted from being accessed within getter chains.

The following methods and classes or packages have been blocklisted by the Java Agent for Getter Chains:

Methods:

- `getClassLoader`
- `loadClass`
- `invoke`

Classes or packages:

- `reflect`
- `nio.file`
- `ClassLoader`
- `io.File`
- `io.Socket`
- `net.Socket`
- `ProcessBuilder`
- `Runtime`

Transaction Detection Rules

Related pages:

- .NET Business Transaction Detection
- PHP Business Transaction Detection

AppDynamics uses different types of transaction detection rules to define entry points and name business transactions:

- Automatic transaction discovery rules include the default entry point types and naming configuration for each app agent type.
- Custom match rules offer finer control over transaction discovery and naming for a single entry point type.

In many cases, the built-in rules yield a useful set of business transactions. If not, you can create new rules or edit the existing rules to optimize the transaction detection for your environment.

Permissions

The *Configure Transaction Detection* permission is needed to customize transaction detection rules.

The *View Sensitive Data permission*, In combination with the *Configure Transaction Detection* permission, enables the use of Live Preview and Business Transaction Discovery features to stream live data from your application.

Live Preview

For Java and .NET environments, you can use interactive live preview tools to find business transaction entry points:

- Business transaction discovery sessions stream live data from the node to the Controller UI to help you make instrumentation decisions. These sessions display transactions based on all transaction detection rules active for the node.
- For certain types of entry points, custom match rule live preview enables you to preview transactions based on applying a single transaction detection rule on a node.

Business transactions within the context of the Live Preview, called transient transactions, do not persist after the Live Preview ends. Business transaction discovery sessions let you apply or discard rule changes when you end the session. Custom match live preview rules apply after you save the rule.

Some transient transactions may invoke exit calls to the same node where the Live Preview session is running. In certain cases, the agent may discover additional transient transactions and uninstrumented code on the downstream segment:

- The exit call and corresponding entry point are automatically discovered exit and entry points like HTTP or Web Service.
- Rare cases of custom activity in-process calls.

Manage Transaction Detection Rules

To customize transaction detection rules in the Controller UI, go to **Configuration > Instrumentation**. From the Business Transactions list, you can also click **Configure** to quickly access Transaction Detection settings.

You apply transaction detection rules to tiers using the flexible scope configuration model. From the Transaction Detection tab, you can manage rules as follows:

- Manage rules for individual scopes on the Transaction Detection tab.
- Manage rules application-wide on the Rules sub-tab. The Rules sub-tab also shows you the tiers where rules are applied and according to which scope.
- View or modify the rules for specific tiers on the Tiers sub-tab.

To modify an existing rule, you can double-click it to display the **Rule Editor**. Click **+/Add** to define a new rule.

Use **Live Preview** button to start a Business Transaction Discovery session for nodes running the Java and .NET Agent.

Transaction Detection Rule Priorities

AppDynamics app agents apply transaction detection rules of the same type in the following order of precedence:

1. Rule priority from highest to lowest. A priority of 0 is the lowest priority possible.
2. Creation date from oldest to newest.

For example, consider an HTTP request that matches a priority 2 custom match rule and a priority 6 custom match rule. The agent applies the priority 6 rule.

When an incoming request matches more than one type of detection rule, AppDynamics applies the detection rules in the following order of precedence:

1. Custom match *include* rules according to the include rule priority. If the request matches an include rule, the agent names the business transaction based on the rule.
2. Custom match *exclude* rules according to the exclude rule priority. If the request matches an exclude rule, the app agent excludes the transaction from discovery.

3. Default automatic detection rules. If the request matches a default automatic detection rule, it names the transaction according to the auto-detection naming scheme.

If two entry point types have the same priority where either can be used to define or register an entry point, then there is some precedence between them.

For example: If in Java just the Java Automatic Discovery Rule with all types is enabled, and there are entry points which can be considered as Web Services or Servlets - then the Servlets will take priority - a higher priority rule is needed if Web Services are the desired default.

To view the default order of transaction detection rules for a tier:

1. Navigate to **Configuration > Instrumentation > Transaction Detection > Tiers**.
2. Click a tier to see the rules ordered by precedence and priority.

For Java and .NET, you can set the `check-bt-excludes-early` node property to "true" to configure the app agent to evaluate custom match exclude rules before custom match include rules. This is a node-based configuration and does not affect the display order of rules in the Controller UI.

Export Detection Rules

You can migrate the rules between applications and tiers using the Controller API. See Configuration Import and Export API.

Automatic Transaction Discovery Rules

Each AppDynamics app agent type has a default automatic transaction discovery rule that applies to all the entry points for that agent. When an agent identifies a business transaction that matches an automatic transaction discovery rule, it names the business transaction based on naming rules for the entry point type.

Automatic Transaction Naming

The naming logic varies depending on the application technology. For example:

- For web-oriented technologies that have URI entry points such as Java Servlets, detection rules name the transaction using the first two segments of the URI by default. See URI Based Entry Points.
- For message-oriented technologies such as an asynchronous message listener or message-driven bean (JMS), the agent names the business transaction for the destination name (the queue name) or the listener class name (if the destination name is not available.)
- For Web services, the agent names the business transaction for the Web service name plus the operation name.

Customize Automatic Transaction Discovery Rules

You can customize automatic transaction discovery rules as follows:

- Add an automatic transaction discovery rule for an agent type.
- Enable or disable different types of automatically discovered entry points.
- Customize transaction naming for some entry points, such as most URI-based entry points.

To access the Rule Editor, double-click an Automatic Transaction Discovery rule for an app agent. Alternatively, you can create a new automatic transaction discovery rule for the agent type by clicking **Add**. Clicking the Rule Configuration tab displays the entry points for the automatic discovery rule.

When you edit, delete, or change the scope for an automatic transaction discovery rule, the Controller UI performs validation to ensure each tier has the correct automatic discovery rule available for its agent type.

If you are using the Java Agent you can preview the effects of your customizations before you apply them in production. See "Edit and Preview Transaction Discovery Rules" on Business Transaction Discovery Sessions.

Disable Transaction Monitoring for Entry Points

Automatic transaction detection rules have a **Transaction Monitoring Enabled** checkbox for each entry point type to control automatic discovery. By default, all entry point types are enabled. When you disable discovery for an entry point type, the agent stops counting, measuring, and recording all activity for that entry point type for the scope where the rule is applied.

> The **Transaction Monitoring Enabled** configuration affects custom match rules in the same scope for the entry point type. For example, if you disable POJO monitoring in the Java Default Automatic Transaction Discovery Rule for a scope, the Java Agent will not discover any POJO transactions for the scope.

Following are the cases when you might want to disable monitoring:

- You know that you don't want to monitor any business transactions of a specific entry point type.
- You want to monitor transactions from a downstream call.

For example, consider an application where Servlets implement Spring Beans and you are interested in monitoring the transaction starting at the Spring Bean level. Disable Servlet monitoring and the auto-detection rule for Spring Beans will be the entry point for the transaction.

Disable Automatic Transaction Detection for an Entry Point Type

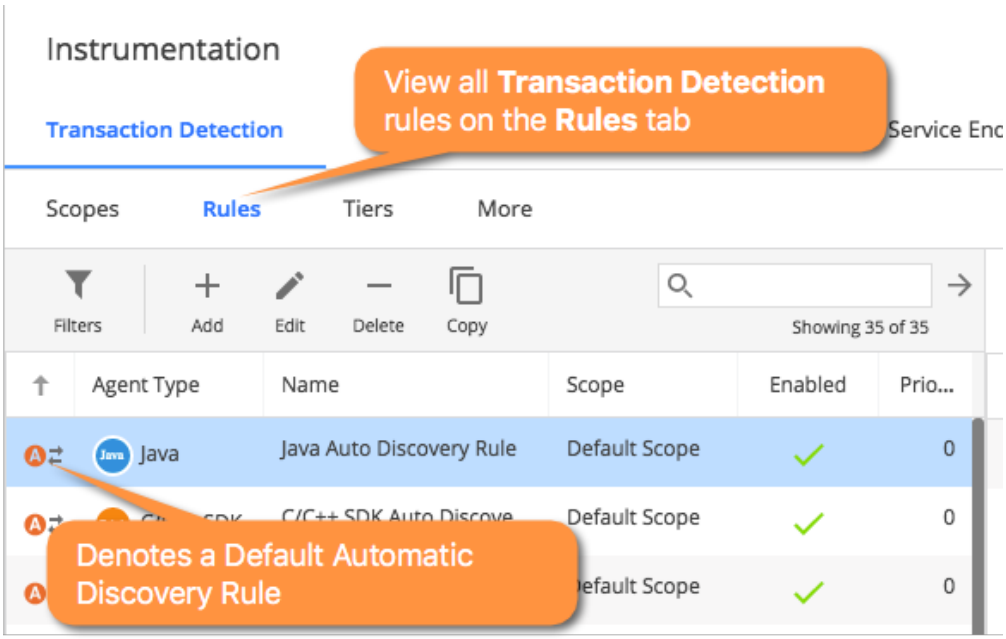
If you want to continue monitoring custom match rules, you can disable **Discover transactions automatically for <***entry point type***>**. When you disable automatic transaction discovery, the app agent stops reporting metrics for previously-detected transactions for the now disabled entry point type. The agent only detects transactions based on custom match rules. However, disabled transactions are not deleted or excluded. See Organize Business Transactions for information on deleting transactions.

- The agent does not discover any new transactions based on the Default Automatic Transaction Discovery Rule for the disabled type.
- The agent no longer auto-detects calls to methods and operations of the disabled entry point type.
- Custom match rules for the entry point type remain active and the agent reports metrics for transactions resulting from custom matches.
- Exit point detection remains enabled.

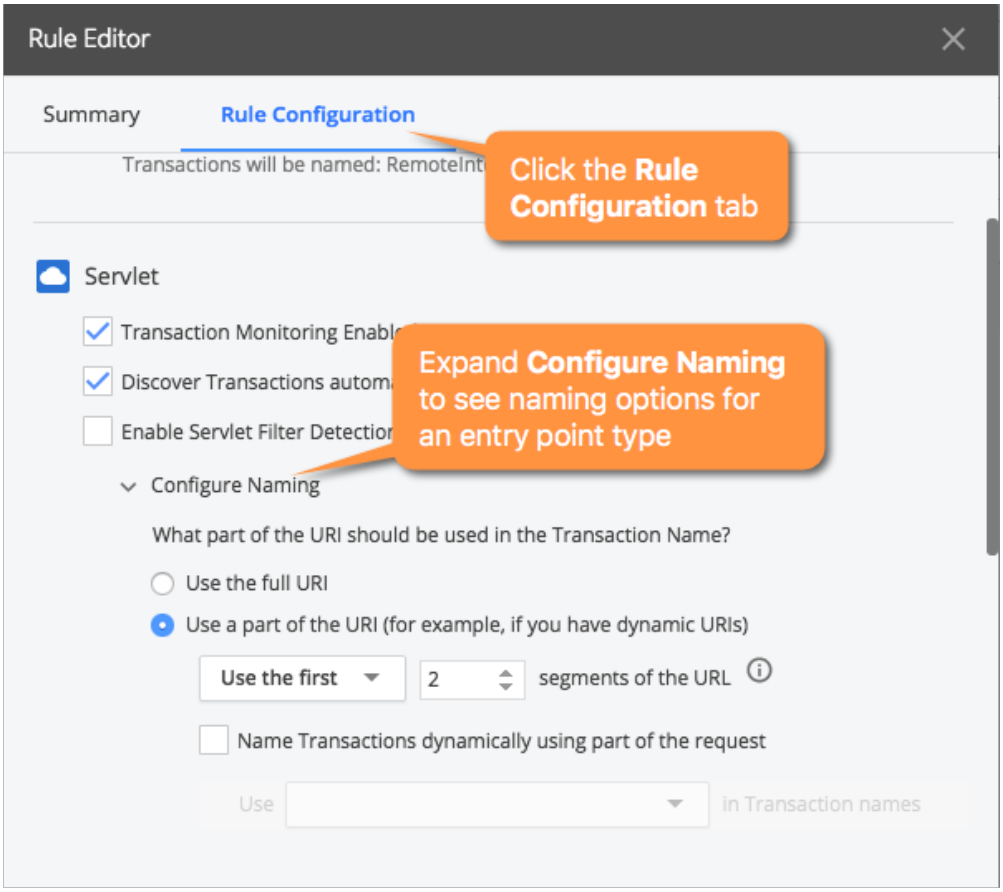
The *Discover Transactions automatically* configuration does not apply to POJO entry points or POCO entry points.

Customize Transaction Naming for an Entry Point

To view and edit the business transaction automatic discovery rules, go to **Configuration > Instrumentation > Transaction Detection**.



To customize transaction naming for an entry point type, double-click a Default Automatic Transaction Discovery rule for specific agent type. Click the Rule Configuration tab to see entry points for the rule. Entry point types that have customizable naming options have the **Configure Naming** expander, as shown below.



For more information specific to customizing naming for URI-based transactions, see URI Based Entry Points.

You can also customize transaction naming using a Custom Match Rule.

Transaction Detection for Apache Web Servers

The AppDynamics Apache Agent monitors web server entry points and names business transactions originating on Apache web server tiers based on the request URI. You can modify the default business transaction naming scheme if needed.

Default Automatic Naming for Web Transactions

By default, the AppDynamics auto-detection naming scheme identifies all web server transactions using the full URI before the query string. For example, the following URI represents a funds transfer operation for an online bank:

```
http://bank.example.com/Account/Transferfunds/California
```

Based on the default scheme, the business transaction would be named /Account/Transferfunds/California.

You can customize the auto-detected naming scheme by configuring identification based on:

- URI segments
- Headers, cookies, and other parts of HTTP requests

To customize auto-detected naming:

1. On the **Transaction Detection** page, select the application or tier to configure and click the Web Server tab.
2. Verify that Transaction Monitoring is enabled and click **Discover Transactions automatically for http web requests**.
   If you disable Discover Transactions automatically for WEB requests, the agent doesn't discover WEB transactions even if you configure custom naming.
3. Click **Configure Naming** for the Web type in the Entry Points panel.

Identify Transactions Using URI Segments

AppDynamics offers several options to automatically name web transactions based on the URI. Consider the following URL representing a checkout operation in an online store:

```
http://onlinestore.example.com/Web/Store/Checkout
```

You can configure AppDynamics to identify a more meaningful name using one of the following options:

- Click **Use the first** or **Use the last** to use the selected number of contiguous segments at the beginning or end of the URI. For example, to identify the checkout transaction using the last two segments of the URI: /Store/Checkout.
- If you need more flexibility, such as using non-contiguous segments in the name, click **Name Transactions dynamically using part of the requests** and specify the segments with the **Use URI segments in Transaction names** option.
- To name the transaction for specific URI segments, click **Use URI segment(s) in Transaction names**. This enables you to skip URI segments or use non-contiguous segments in the naming scheme.
  Enter the segment numbers separated by commas: 1, 3. For example, the following URL represents the checkout transaction requested by a customer with ID 1234:
  `http://onlinestore.example.com/Store/cust1234/Checkout`
  The checkout transaction is the same regardless of the customer, so it makes sense to omit the customer ID and name the transaction based on the first and third segments of the URI: /Store/Checkout.

Identify Transactions Using Headers, Cookies, and Other Parts of HTTP Requests

To identify business transactions using particular parts of the HTTP request, click **Name Transactions dynamically using part of the request** and configure the option that makes sense for your application.

Carefully consider your naming configuration choices. If you use a value such as the request originating address and you have many clients accessing your application, it's likely that you would quickly reach the maximum number of registered business transactions. See "About the "All Other Traffic" Business Transaction" in Business Transactions for information about this event.

The following provides sample results based on the configuration options:

- To name transactions based upon the parameter name, click **Use a parameter value in Transaction names** and enter the **Parameter Name**.
  For example, when you name the following transaction using the parameter name "category": http://example.com/Store/Inventory?category=electronics, AppDynamics names the transaction to include the category parameter value: /Store/Inventory.electronics.
- To use a header value in transaction names, click **Use header value in transaction names** and enter a **Header Name**.
  For example, if you name the transaction using a header such as "Version", AppDynamics names transactions with the header value as follows: /Store/Inventory.v2.5.
- To use a cookie value in transaction names, click **Use a cookie value in Transaction names** and enter the **Cookie Name**.
  For example, for a website that tracks a user's loyalty status in a cookie. Set the Cookie Name to "loyalty". AppDynamics names transactions for the loyalty cookie value: /Store/Inventory.Status=Gold.
- To Use a session attribute value in transaction names, Click **Use a session attribute in Transaction names** and enter the **Session Attribute Key**.
  For example, a website stores a customer's region in the session property. Set the Session Attribute name to "region". AppDynamics names transactions for the region session attribute value: /Store/Inventory.NorthAmerica.

- To use the request method in Transaction names, click **Use the request method (GET/POST/PUT)** in Transaction names.
  For example: /Store/Inventory.GET.
- To use the request host in Transaction names, click **Use the request host in Transaction names**. For example: `/Store/Inventory.192.0.2.0`
- To use the request originating address in Transaction names, click **Use the request originating address in Transaction names**.
  AppDynamics names transactions for the IP address of the request client. For example: `/Store/Inventory.192.0.2.10`.
- To use a regular expression on the URI to name the transaction, click **Apply a custom regular expression on the transaction name**.
  AppDynamics uses the following rules to name the transaction:
  - The Apache Agent uses Perl style regular expressions.
  - AppDynamics tests the regular expression against the segments specified in the configuration.
  - AppDynamics names the business transaction for the substring match.
  - If the regular expression pattern isn't found, the business transaction name follows the URI rules.
  - If you use groups in the regular expression, AppDynamics names the business transaction for the first matching group. If no matching groups are found in the pattern match, AppDynamics names the transaction for the fully matched substring.
    For example consider the following URL: `http://mywebapp.example.com/abc/;jsessionid=12345008;mykey=mytransaction;anotherkey=foo`.
    Use the first two segments of the URI with the following regular expression to name the transaction for the value of *mykey*: `.*mykey.(\w+).*`.
    In this instance AppDynamics names the transaction *mytransaction*.

Custom Match Rules and Exclude Rules

You can create custom match rules and exclude rules for Web type entry points for Apache Agent. The configuration parameters work the same as those for custom naming in this page. If an excluded request on the Web Server tier passes to another instrumented tier such as Java, PHP, or Python, the downstream agent will detect the transaction. See Custom Match Rules.

Validation for Automatic Transaction Discovery Rules

When you first create a business application in AppDynamics, the Default Scope ensures that all tiers for the application have a default automatic discovery rule for every agent type. Because the scope configuration model is highly configurable, the Controller UI enforces validations in the following cases:

- When you change the automatic discovery rules for a scope.
- When you assign or reassign tiers to a scope.

When you attempt to make an invalid change, the Controller prevents the change and displays a message in the Controller UI.

These validations ensure that after configuration updates every tier in your business application has the default automatic discovery rule for its agent type. They also ensure that there is an automatic discovery rule for all agent types to handle any prospective tiers you add to your application.

Existing Tiers

Every existing tier must belong to a scope that includes an automatic transaction detection rule for the tier's agent type. For example, a Java tier must belong to a scope with a Java automatic transaction detection rule.

The following rules apply:

- You can remove a tier from scope only when the tier has an auto-discovery rule from another scope.
- You can remove an automatic transaction detection rule from a scope only when tiers have auto-discovery rule from another scope.

Prospective Tiers

To ensure that any new tier you add to your application has an automatic transaction detection rule available, there must be at least one automatic transaction detection rule for each agent type in a scope of the following types:

- A scope that contains all tiers.
  OR
- A scope that contains all tiers excluding specific existing tiers.

Custom Match Rules

Related pages:

- Using Regular Expressions
- Custom Exclude Rule Examples
- .NET Agent for Linux Business Transaction Configuration

A custom match rule is a type of transaction detection rule that enables you to define criteria for business transaction discovery, and naming for an entry point type.

There are two types of custom match rules:

- A custom include rule defines an entry point to use in a new transaction.
- A custom exclude rule lets you prevent AppDynamics from detecting business transactions you are not interested in tracking. For example, application framework services, administrative console requests, or heartbeat pings, and so on.

Use Live Preview to experiment with certain custom match rule configurations and identify potential business transaction entry points:

- During a Business Transaction Discovery Session, you can use the Classes/Methods browser or Uninstrumented Code to create POJO and POCC custom match rules.
- When you create or edit a POJO or Servlet entry point, you can use Live Preview for the individual rule you are creating.

Transaction Naming

When an app agent detects a request that matches a custom include rule, it names the business transaction based on the rule name. You can further refine the transaction name using transaction splitting for URI based rules, for POJOs, or for POCOs.

If a request matches more than one custom include rule, it applies the rule with the highest number priority. See Transaction Detection Rules.

Create a Custom Match Rule

1. Navigate to **Configuration > Instrumentation** and select the **Transaction Detection** subtab.
2. Select the Rules subtab.

   > You can also add a rule within a specific scope on the Transaction Detection tab.

3. Select **Add** to open the Add Rule page with *Custom Match Rule* selected.
4. Select the **Agent Type** and the **Entry Point Type** and select **Next**.
5. On the Summary tab, make general rule configurations:
   - If you are creating a custom match exclude rule, click **Exclude Transactions discovered by this rule**.
   - Enter the rule *Name*. The app agent names requests that match the rule for the rule name.

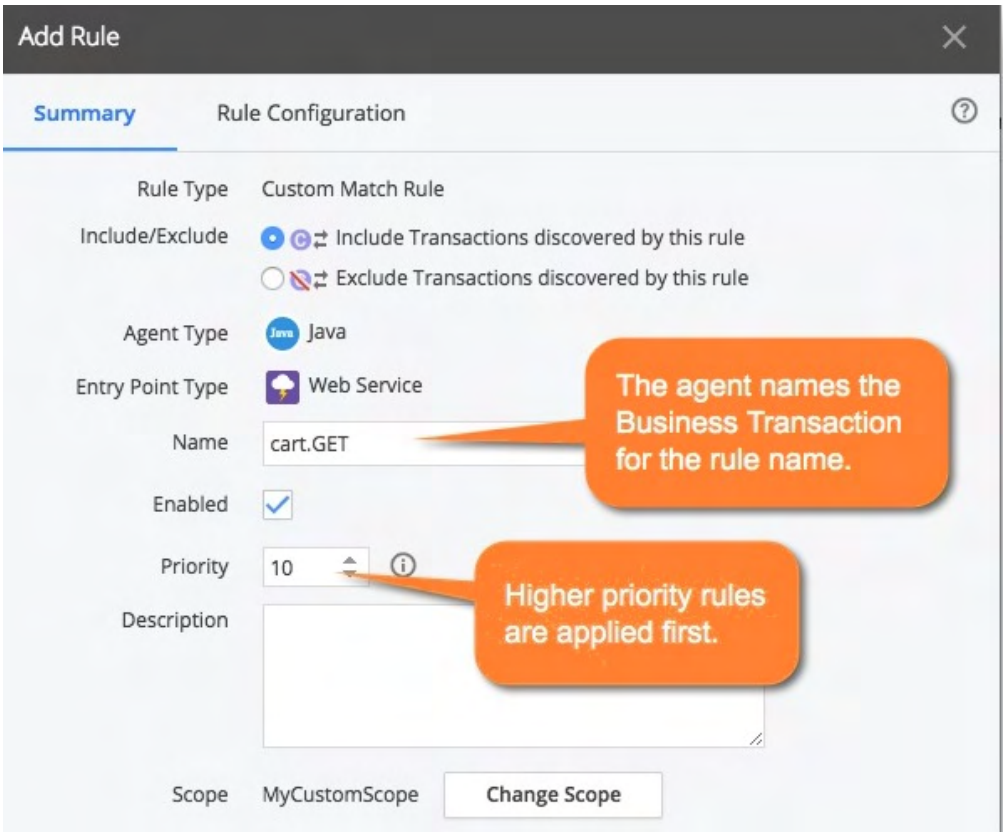
     > To avoid errors, do not include these special characters in the rule name: less than sign (<) and greater than sign (>).

   - Optionally disable the rule, set the rule *Priority* and assign the rule to a *Scope.*
     For information on rule priorities, see Transaction Detection Rules.
     For information on scopes, see Scope Configuration Model.
6. On the Rule Configuration tab, configure the match criteria based upon the entry point type. The PHP web choices, for example, let you specify HTTP-oriented match criteria, such as those for HTTP method, URI, and query parameters.

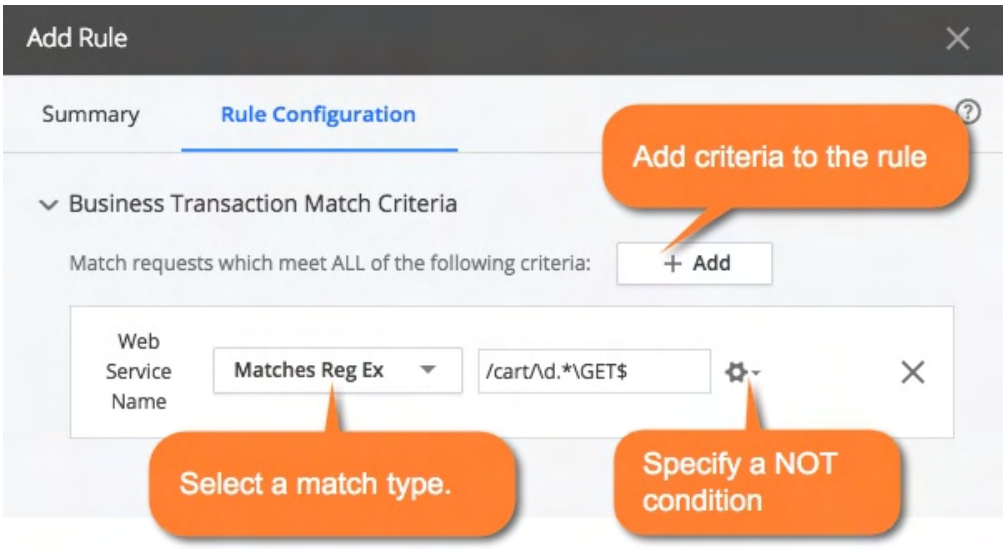
   > Servlet and POJO entry point types feature a *Live Preview* button to launch custom match rule live preview session. A live preview streams data from an active node so you can interactively experiment with rule configuration in the Add Rule window.

Custom Match Include Rule Example

This example shows a custom match include rule named *cart.GET* in a scope named MyCustomScope.



The "cart.GET" rule generates business transactions for Web Service GET requests where the URI begins with `/cart/` followed by a number. AppDynamics names the resulting business transaction `cart.GET`:



You can add conditions to the rule to match based on specific HTTP parameters or hostnames.

For certain HTTP-based request types, such as Servlets or ASP.NET, a match rule can have more than one HTTP Parameter match condition. A request must match all HTTP Parameter conditions to match this rule. There is an implicit AND operator between the parameters rather than an OR operator.

You can use more complex regular expression matching. For example, the following request URLs:

- `example.com/aaa123.aspx`
- `example.com/b1.aspx`

To match any incoming request URL to this business transaction, use:

```
example\.com/[a-zA-Z]]+?[0-9]+?
```

Any request to a URL that includes `example.com` followed by uppercase or lowercase letters and numbers would match this business transaction detection rule. A URL that has no digits after the letters, such as `example.com/z.aspx` does not match.

See Using Regular Expressions for more information on using regular expressions in the UI.

Transaction Splitting for URI Based Entry Points

You can configure business transaction splitting for these types of URI based entry points:

- Java Servlet
- ASP.NET
- PHP Web
- Node.js Web
- Python Web
- Web Server

When you enable transaction splitting, the agent resolves a subset of requests that match the custom include rule into a separate business transaction based on a dynamic part of a request. The app agent names the transaction:

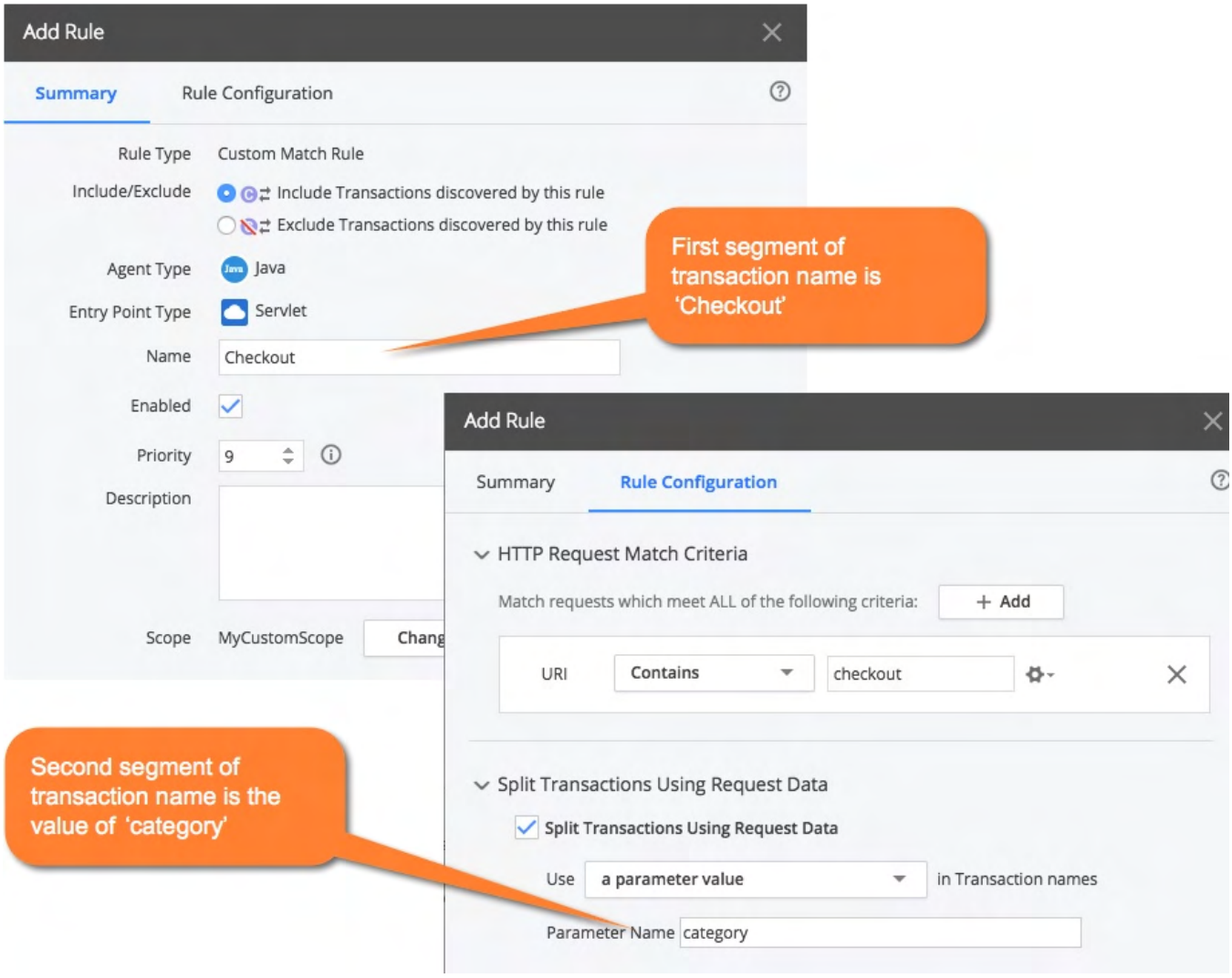
```
<custom match rule name>.<name derived from the split configuration>.
```

Consider an application with incoming requests:

- `http://nwtrader.com/checkout?category=electronics`
- `http://nwtrader.com/checkout?category=clothing`

This example shows how to configure transaction splitting to resolve two requests into two different business transactions, named:

- Checkout.electronics
- Checkout.clothing



You can split servlet transactions based upon the payload. See Split Servlet Transaction by Payload Examples.

Default Custom Exclude Rules

AppDynamics includes default exclude rules for the entry points for frameworks that are not usually of interest. Navigate to **Configuration** > **Instrumentation > Transaction Detection > Rules** and filter on **Rule Type: Custom Exclude** to view or modify the default custom match exclude rules. See Custom Exclude Rule Examples.

To configure exclude rules for Service Endpoints, see Service Endpoints.

Java Business Transaction Detection

This page lists different types of Java entry points for business transactions.

A tier can have multiple entry points. For example, a Java framework implementation may have a combination of pure Servlets or JSPs, Struts, Web services, and Servlet filters, all co-existing as monitored entry points on the same JVM.

The middle-tier components like EJBs and Spring beans are usually not considered entry points because they are normally accessed using either the front-end layers such as Servlets or from classes that invoke background processes.

For details on types of Java entry points and how to set up custom match rules, see:

- Servlet Entry Points
- Struts Entry Points
- Spring Bean Entry Points
- EJB Entry Points
- POJO Entry Points
- Web Service Entry Points
- Binary Remoting Entry Points for Apache Thrift
- CometD Support
- JAX-RS Support
- Spring Integration Support

Servlet Entry Points

AppDynamics automatically detects requests to servlet-based entry points and generates business transactions based on the requests. For general information on how to customize automatic transaction detection, see URI Based Entry Points.

Frameworks Supported as Servlets or Servlets Filter

AppDynamics supports many web frameworks based on servlets or servlet filters. The servlet configuration settings in AppDynamics apply to these frameworks as well as to plain servlets. Frameworks include:

- Spring MVC
- Wicket
- Java Server Faces (JSF)
- JRuby
- Grails
- Groovy
- Tapestry
- ColdFusion

Custom Match Rules for Servlet Transactions

Custom match rule let you control how business transactions are generated for Servlet-based requests.

Match Conditions

Match conditions can be based upon the URI, HTTP method, hostname, servlet name, or other characteristics of the request.

For HTTP Parameter conditions, you can add more than one. If you configure more than one HTTP Parameter match criteria, they must all be met by a request to be subject to this business transaction identification and naming rule, as must all conditions you configure for the rule.

Some of the options have NOT conditions that you can choose to negate the configured condition. Choose this option in the gear icon next to the condition.

Splitting Transactions using Request Data or Payload

If you match by URI, you can have the transaction identified (or split) based on values from the request, such as URI patterns, request data, or payload. See URI Based Entry Points for information on naming or splitting transactions by the request elements.

When you create the rule, it appears in the Custom Match Rule list, where you can enable, disable, edit or remove the rule.

See Split Servlet Transaction by Payload Examples for use case examples.

POST Request Body Parameter Matching Considerations

You can configure Servlet-based custom match that match POST requests based on HTTP body parameter values. To avoid interfering with the operation of the application, the Java Agent applies some special processing measures and limits on how it accesses the body parameters. This behavior can affect whether requests can be matched by POST body parameters for your particular application. If you are having trouble configuring matching by HTTP parameter in POST requests, you should understand this behavior.

Transaction Naming Match Criteria

For transaction match criteria, match conditions for HTTP parameters work under these conditions:

- If the parameter is a query string in the URL of the POST request, as opposed to the request body.
- If the Servlet Request body parameters have already been read and parsed by the application prior to the invocation of the servlet. A Servlet Filter, for example, may do this.

Otherwise, custom servlet match rules do not match HTTP parameter values in POST requests.

Transaction Splitting

For transaction split rules based on parameter matching in POST requests, the Java Agent defers transaction naming for incoming requests until the application servlet has accessed the request parameters.

The Java Agent considers the parameters "accessed" once the application invokes `getParameterMap()`, `getParameter()`, `getParameterNames()`, or `getParameterValues()` on the ServletRequest object. If the servlet does not call one of these methods, the agent will not apply the split rule and the transaction will not be named.

Exclude Rules for Servlets

To prevent specific Servlet methods from being monitored, add a custom exclude rule. The controls for selecting Servlets to exclude are the same as those for custom match rules.

Split Servlet Transaction by Payload Examples

AppDynamics lets you split servlet custom match rules according to the request payload. For each of these examples, you must create a servlet custom match rule and enable the URI option on the Rule Configuration tab.

Split by POJO Method Call

Using a Java method to name a transaction is useful when:

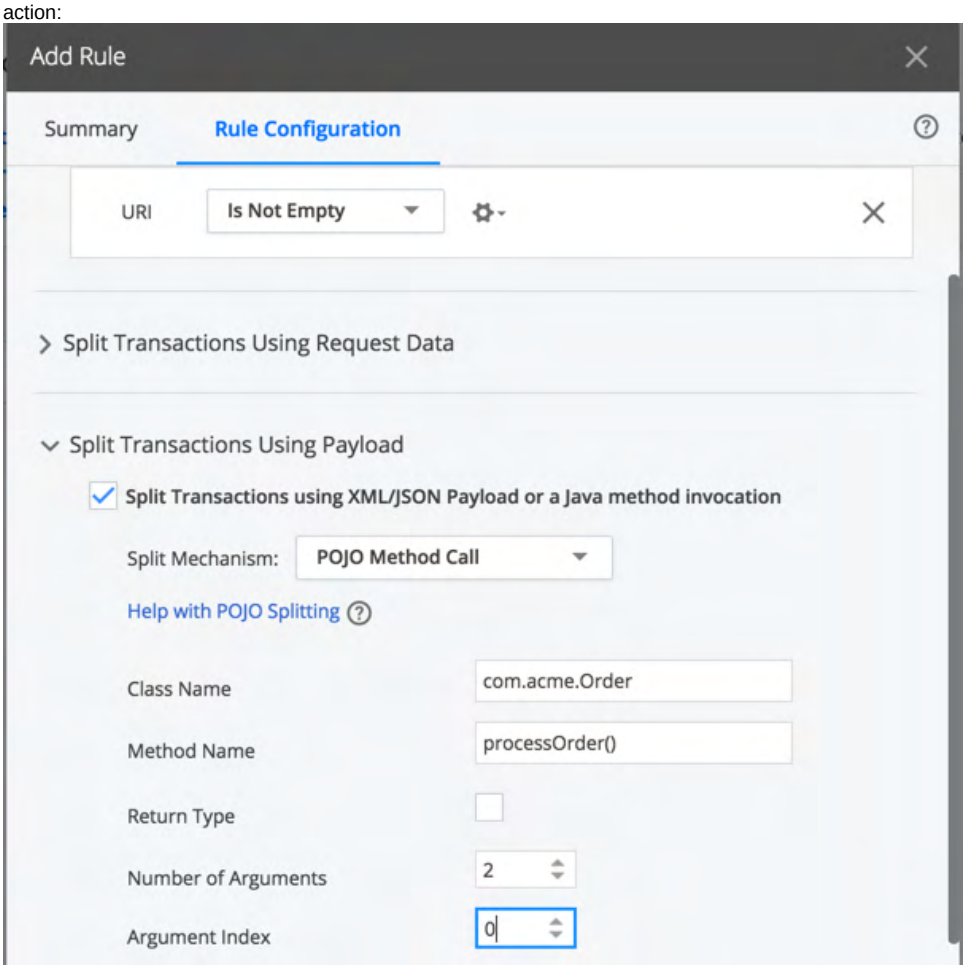
- You might not have a clear URI pattern, or
- You are using an XML/JSON framework not otherwise supported

For example, consider the `processOrder()` method in the `doPost()` method of the servlet at the following URL: `http://acmeonline.com/store`.

```
public void doPost(HttpServletRequest req, HttpServletResponse resp) {
   //process the data from the sevlet request and get the orderType and the items
   processOrder(orderType,item)
   ...
}
public void processOrder(String orderType,String item) {
   //process order
}
```

You want to derive transaction naming from the first parameter to the `processOrder()` method, `orderType`.

1. On the Rule Configuration tab for the custom match rule, enable the **Split Transactions using the XML/JSON payload or Java method invocation**.
2. Select **POJO Method Call** as the splitting mechanism.
3. For the method, choose the processOrder, and specify the parameter to use by numeric position in the parameter (0 index). The following screenshot displays the configuration of a custom match rule which will name all the qualifying requests into a `Store.order.creditcard` trans action:



In addition to the parameter, you can also specify either the return type or a recursive getter chain on the object to name the transaction. For example, if the method parameter points to a complex object like PurchaseOrder, you can use something like getOrderDetails().getType() to correctly name the transaction.

Split Transaction by JSP Name

You can identify transactions by JSP name, as follows:

1. On the Rule Configuration tab, enable the **Split Transactions using the XML/JSON payload or Java method invocation**.
2. Select **POJO Method Call** as the splitting mechanism.
3. Set the name of the class to `com.sun.faces.application.ViewHandlerImpl`.
4. Set the name of the method to `renderView()`.
5. Set the argument index to 1.
6. Define the Method Call Chain or getter as `getViewId()`. The agent appends the value to the name of the transaction as follows: *<Name of the Custom Rule>.<path to jsp>*.

You can later rename these business transactions to a more user-friendly name if you like.

Split Transactions on XPath Expression

You can access values in an XML payload for business transaction naming or splitting using an XPath expression. Consider the following example from an Ecommerce order transaction where the XML represents an order for three items. The order uses credit card processing, which is the distinguishing element for this body:

```
<acme>
   <order>
      <type>creditcard</type>
      <item>Item1</item>
      <item>Item2</item>
      <item>Item3</item>
   </order>
<acme>
```

The URL is:

```
http://acmeonline.com/store
```

The `doPost()` method of the Servlet is:

```
public void doPost(HttpServletRequest req, HttpServletResponse resp) {
   DocumentBuilderFactory docFactory =
   DocumentBuilderFactory.newInstance();
   DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
   Document doc = docBuilder.parse(req.getInputStream());
   Element  element = doc.getDocumentElement();
   //read the type of order
   //read all the items
   processOrder(orderType,items)
   ...
}
```

Imagine you want to differentiate "Order" transactions based upon the type of order. You can use the XPath expression `//order/type` on this XML payload, which in this example evaluates to `creditcard`.

1. On the Rule Configuration tab, click **Split transactions using XML/JSON Payload or a Java method invocation**.
2. Choose **XPath Expressions** for the Split Mechanism.
3. Enter the XPath expression that points to the value of the XML element to use for naming. In this example, `//order/type`

The agent appends the value of the XPath expression to the name of the business transaction, for example, `Store.order.creditcard`. Even though the agent doesn't name the transaction until after XML parsing, AppDynamics measures the duration of the business transaction to include the execution of the `doPost()` method.

You can use one or more XPath expressions to chain the names generated for the Business Transaction.

You can specify whether the request results in transaction splitting when the expression does not evaluate to a value.

Split Transactions on Java XML Binding

You can identify transactions for Java XML data binding frameworks for these types of frameworks:

- Castor
- JAXB
- JibX
- XMLBeans
- XStream

In the following example, the posted XML is unmarshalled to the `PurchaseOrderDocument` object, and the `getOrderType()` method should be used to identify the type of the order:

```
<acme>
   <order>
     <type>creditcard</type>
      <item>Item1</item>
      <item>Item2</item>
      <item>Item3</item>
   </order>
<acme>
```

The following snippet shows the `doPost()` method for the Servlet:

```
public void doPost(HttpServletRequest req, HttpServletResponse resp) {
   PurchaseOrderDocument poDoc = PurchaseOrderDocument.Factory.parse(po);
   PurchaseOrder po = poDoc.getPurchaseOrder();
   String orderType = po.getOrderType();

   //read all the items
   processOrder(orderType,items)
   ...
}
```

To split the transaction based upon the XML Binding:

1. On the Rule Configuration tab, check **Split transactions using XML/JSON Payload or a Java method invocation**.
2. Select **Java XML Binding** as the split mechanism.
3. Enter these values for the match criteria:
    - Unmarshaled Class Name: `PurchaseOrderDocument`
    - Method name: `getOrderType()`

The agent identifies the business transaction for this example as `Store.order.creditcard`:

This custom rule ensures that the Java agent intercepts the method in XMLBeans (which unmarshalls XML to Java objects). If the name of the transaction is not on a first level getter on the unmarshalled object, you can also use a recursive getter chain such as `getOrderType().getOrder()` to get the name.

Although the transaction name is not obtained until the XML is unmarshalled, the response time for the transaction is calculated from the `doGet()` method invocation.

Split Transactions on JSON Payload

You can access JSON payload for transaction identification purposes using the method where the Servlet unmarshalls the payload.

For example, the following JSON payload posts an `order` for an item `car` and uses `creditcard` for processing the order. The URL is `http://acmeonline.com/store`:

```
order :{
  type:creditcard,
  id:123,
  name:Car,
  price:23
}}
```

The following code snippet shows the `doPost` method of the Servlet:

```
public void doPost(HttpServletRequest req, HttpServletResponse resp) {
   //create JSONObject from servlet input stream
   String orderType = jsonObject.get("type");
   //read the item for the order
   processOrder(orderType,item)
   ...
}
```

After the application unmarshalls the posted JSON payload to the JSON object, the *type* key is available to identify the type of the order. In this case, this key uniquely identifies the business transaction.

To use the JSON payload for transaction identification you must set the `enable-json-bci-rules` node property to `true` on each node to enable this rule. To configure the rule:

1. On the Rule Configuration tab, check **Split transactions using XML/JSON Payload or a Java method invocation**.
2. For the JSON object key, enter the name of the JSON object. For example, `type`.

The agent automatically intercepts the `JSONObject.get("$JSON_Object_Key")` method to name the transaction. Although the agent doesn't obtain the transaction name until the JSON object is unmarshalled, the response time for the transaction will be calculated from the `doGet()` method.

Struts Entry Points

When your application uses Struts to service user requests, AppDynamics intercepts individual Struts Action invocations and names the user requests based on the Struts action names. A Struts entry point is a Struts Action that is being invoked.

AppDynamics supports the following versions of Struts:

- Struts 1.x
- Struts 2.x

Struts Action invocations are typically preceded by a dispatcher Servlet, but identification is deferred to the Struts Action. This ensures that the user requests are identified based on the Struts Action and not from the generic URL for Dispatcher Servlet.

The response time for the Struts-based transaction is measured when the Struts entry point is invoked.

Struts Request Names

When a Struts Action is invoked, by default AppDynamics identifies the request using the name of Struts Action and the name of the method. All automatically discovered Struts-based transactions are thus named using the convention `<Action Name>.<Method Name>`.

For example, if an action called ViewCart is invoked with the SendItems(), the transaction is named `ViewCart.SendItems`.

For Struts 1.x the method name is always `execute`.

You can rename or exclude auto-discovered transactions. See Organize Business Transactions.

Custom Match Rules for Struts Transactions

For finer control over the naming of Struts-based transactions, use custom match rules.

A custom match rule lets you specify customized names for your Struts-based requests. You can also group multiple Struts invocations into a single business transaction using custom match rules. See Custom Match Rules for information about accessing the configuration screens.

The matching criteria for Struts transactions are: Struts Action class names, Struts Action names, and Struts Action method names.

Exclude Rules for Struts Actions or Methods

To prevent specific Struts Actions and methods from being monitored, add a custom exclude rule. The criteria for Struts exclude rules are the same as those for custom match rules. In particular, you may need to exclude custom-built dispatch servlets, as described next.

Exclude Custom-built Dispatch Servlet

When a Struts action is called, it can demarcate a transaction as an entry point. AppDynamics instruments the Struts invocation handler to get the action name because the Struts action is not an interface. The invocation handler provides the Java Agent with the name of the action being invoked. If the dispatcher Servlet is custom-built and has not been excluded from instrumentation, the wrong entry point could be instrumented and the business transaction could be misidentified.

To address this issue, add a custom exclude rule for the dispatcher servlet or add a BCI exclude for it.

Spring Bean Entry Points

This page describes how to configure transaction entry points for Spring Bean requests.

Spring Bean-based Transactions

AppDynamics allows you to configure a transaction entry point for a particular method for a particular bean in your environment. The response time is measured from when the Spring Bean entry point is invoked (after receipt at a dispatcher servlet).

Default Naming for Spring Bean Requests

When you enable automatic discovery for Spring Bean based requests, AppDynamics automatically identifies all the Spring Beans based transactions and names these transactions using the following format:

```
BeanName.MethodName
```

By default, the transaction discovery for Spring Bean-based requests is turned off. You can enable it in any Automatic Transaction Discovery rule for Java. Select **Discover Transactions automatically for all Spring Bean invocations** on the Rule Configuration tab.

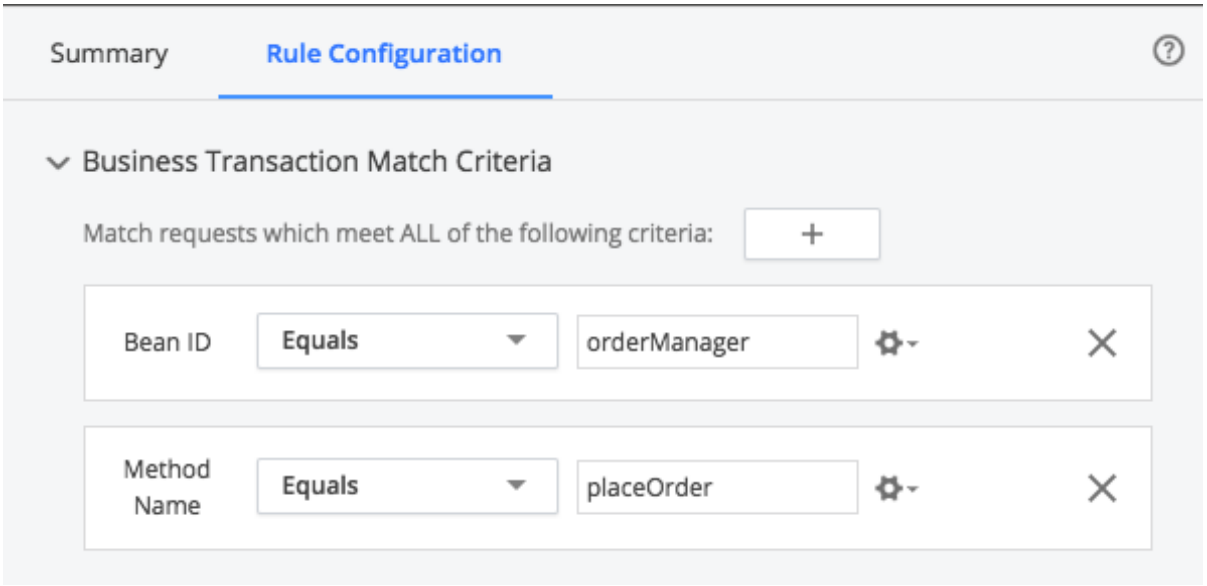
When a class is mapped to multiple Spring bean names, by default only the name of the first Spring bean found is used. This may not be ideal for your application such as cases where a call graph for web service A that has Spring beans from web service B. To contend with this scenario, you can remove the Bean name from the transaction name using the `capture-spring-bean-names` node property, as described on App Agent Node Properties (B-C).

Custom Match Rules for Spring Bean Requests

If you are not getting the required visibility with the auto-discovered transactions, you can create a custom match rule for a Spring Bean based transaction. You can match based on one or more of the following criteria:

- Bean ID
- Method Name
- Class Name
- Extends
- Implements

The following example creates a custom match rule for the `placeOrder` method in the `orderManager` bean.



Exclude Rules Spring Bean Transactions

To exclude specific Spring Bean transactions from detection, add a custom exclude rule.

The criteria for Spring Bean exclude rules are the same as those for custom match rules.

EJB Entry Points

AppDynamics allows you to configure an EJB-based transaction entry point on either the bean name or method name. The response time for the EJB transaction is measured when the EJB entry point is invoked.

Default Naming for EJB Entry Points

AppDynamics automatically names all the EJB transactions <*EJBName*>.<*MethodName*>.

Enabling EJB Transaction Detection

By default, automatic transaction discovery for EJB transactions is turned off. To get visibility into these transactions, enable the auto-discovery for EJB based transactions explicitly.

Before you enable auto-discovery for EJB based transactions, note that:

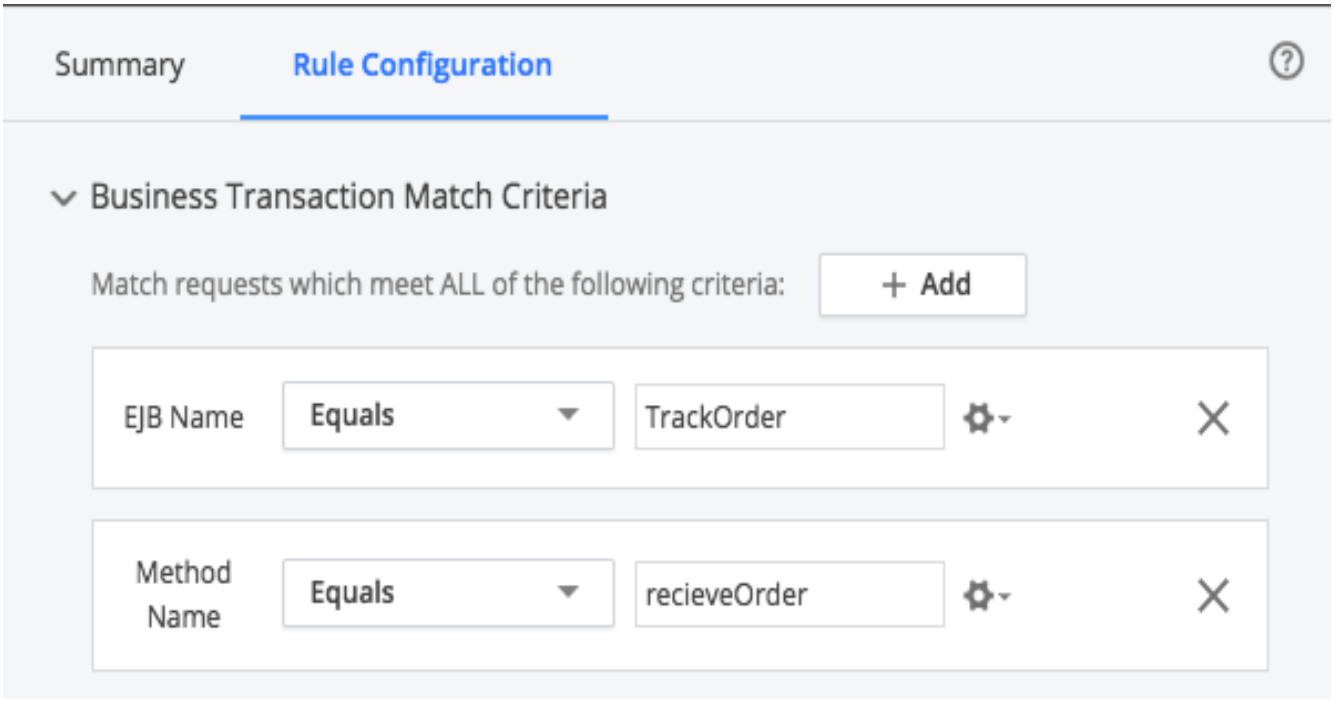
- If the EJBs use Spring Beans on the front-end, the transaction is discovered at the Spring layer and the response time is measured from the Spring Bean entry point.
- AppDynamics groups all the participating EJB-based transactions (with remote calls) in the same business transaction. However, if your EJBs are invoked from a remote client where the App Server Agent is not deployed, these EJBs are discovered as new business transactions.

You can enable auto-discovery for EJB transactions in any Automatic Transaction Discovery rule for Java. Check **Discover Transactions automatically for all Spring Bean invocations** on the Rule Configuration tab.

Custom Match Rules for EJB based Transactions

If you are not getting the required visibility with auto-discovered transactions, create a custom match rule for an EJB based transaction.

The following example creates a custom match rule for the receiveOrder method in the `TrackOrder` bean.



In addition to the bean and method names, other match criteria that could be used to define the transaction are the EJB type, class name, superclass name, and interface name.

Exclude Rules for EJB Transactions

To exclude specific EJB transactions from detection add a custom exclude rule. The criteria for EJB exclude rules are the same as those for custom match rules.

POJO Entry Points

This page describes how to create custom match rules for POJO (Plain Old Java Object) applications. See Custom Match Rule Live Preview for instructions about interactively working with live data to create a POJO.

About POJO Custom Match Rules

Custom match rules for POJOs let you configure business transaction detection in application environments that run pure Java applications or that use frameworks with entry points that are not automatically detected.

Unlike common frameworks, which are characterized by well-known entry points for applications, the logical entry point for a business transaction for a POJO entry point could be any method in the application.

To configure a custom POJO entry point, therefore, you need to identify the method that AppDynamics should consider the business transaction entry point. Keep in mind that the start and end of the execution of the method will correspond to the start and end of the business transaction, so the method should encapsulate the complete execution of the business transaction.

For example, consider the method execution sequence:

```
com.foo.threadpool.WorkerThread.run()
    calls com.foo.threadpool.WorkerThread.runInternal()
        calls com.foo.Job.run()
```

The first two calls to `run()` method are the blocking methods that accept a job and invoke it. The `Job.run()` method is the actual unit of work, because Job is executed every time the business transaction is invoked and finishes when the business transaction finishes.

Methods like these are the best candidates for POJO entry points. The response time for POJO transactions is measured from this entry point, and remote calls are tracked the same way as are remote calls for a Servlet's Service method.

Creating POJO Custom Match Rules

You can create a POJO entry point by adding a custom match rule with *POJO* as the Entry Point Type. The agent names business transactions for the custom match rule.

> In order for the agent to discover POJO transactions, your custom match rule must belong to a scope that also includes a Default Java Automatic Transaction Discovery rule with POJO transaction monitoring enabled.

You can classify matching transactions as background tasks by enabling the **Background Task** check box. When a request runs as a background task, AppDynamics reports only Business Transaction metrics for the request. It does not aggregate response time and calls metrics at the tier and application levels for background tasks. This ensures that background tasks do not distort the baselines for the business application. Also, you can set a separate set of thresholds for background tasks. For more information, see Monitor Background Tasks.

The custom match rule configuration offers several options for matching method invocations and splitting matched calls into separate business transactions, as described below.

> If you are running on IBM JVM v1.5 or v1.6, you must restart the JVM after defining the custom match rules.

Matching by Class and Method

You can specify matching criteria for the custom rule based on various forms of a method or class to which the method belongs.

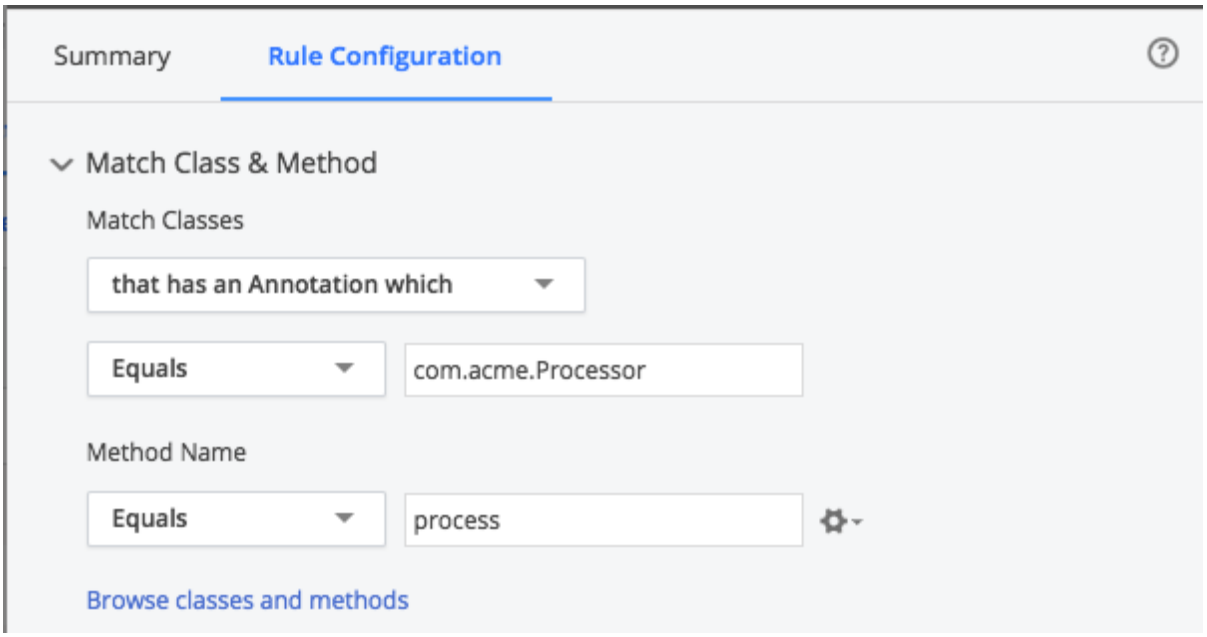
When specifying the method name matching criteria, use parameter matching to match against a particular method signature. For example, say you want t o instrument one of more methods in the following class:

```
class A
{
    public void m1();
    public void m1(String a);
    public void m1(String a, com.mycompany.MyObject b);
}
```

Configure instrumentation for each method based on its signature as follows:

- To instrument the first method signature (with no parameters), create a POJO-based business transaction match rule as follows:
  - Match Classes: with a Class Name that Equals A
  - Method Name: Equals m1()
- To instrument the second method, create a match rule as follows:
  - Match Classes: with a Class Name that Equals A
  - Method Name: Equals m1(java.lang.String)
- To instrument the third method, create a match rule as follows:
  - Match Classes: with a Class Name that Equals A
  - Method Name: Equals m1(java.lang.String, com.mycompany.MyObject)

You can also match methods that belong to classes with certain annotations. For example, say you want to match all classes that are annotated with `@com.acme.Processor`. The custom rule configuration can define the annotation as follows:



When calls to the process() method match, the result is a business transaction named for the custom match rule name.

Matching on Inner Classes and Inner Interfaces

You can match on inner classes and inner interfaces by adding a $ sign after the class name. For example, you could specify the following for the class name:

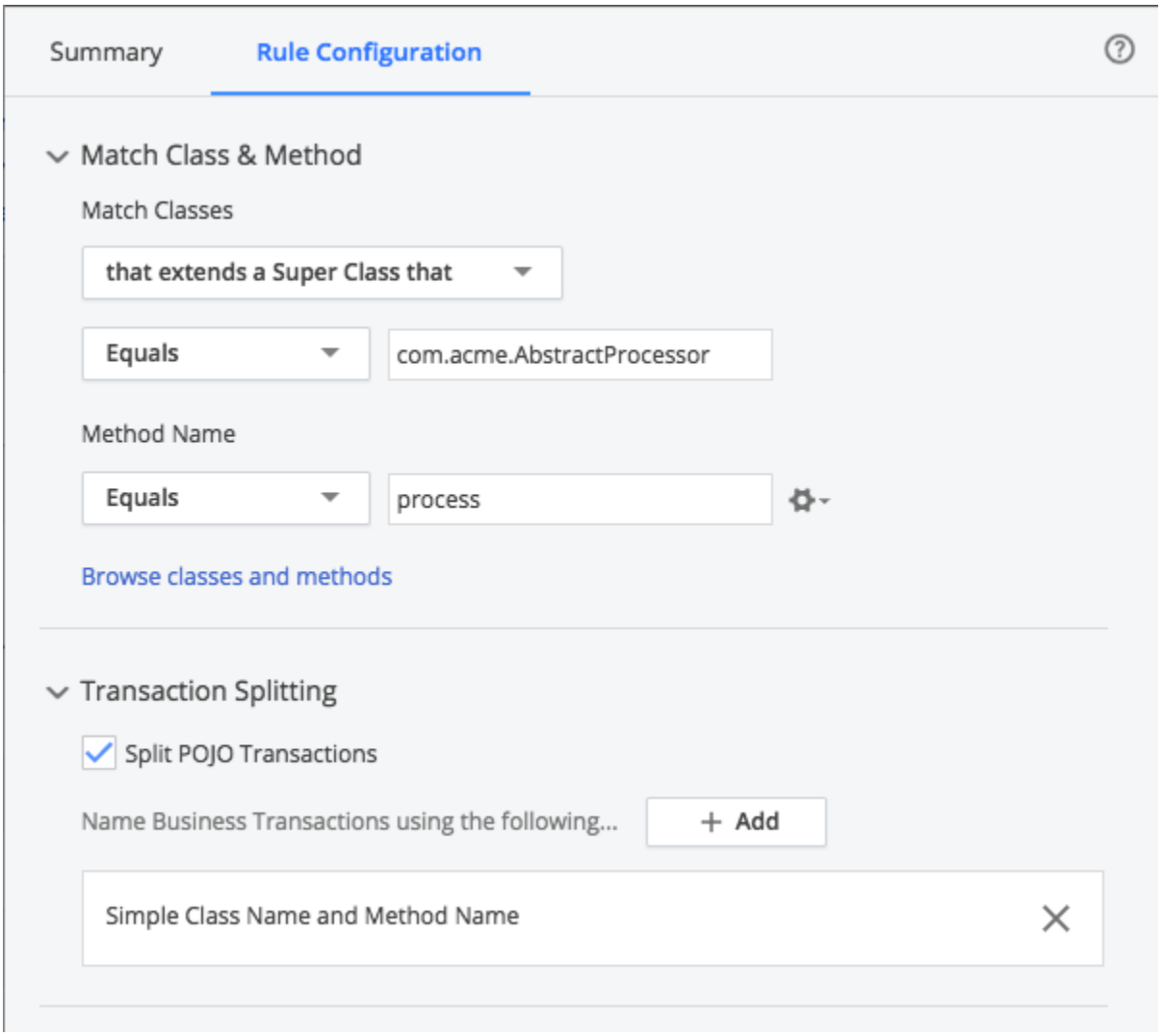
```
com.my.package.OuterClass$InnerClass
```

Splitting Matches into Separate Business Transactions

With only transaction matching configured, all requests matched by the match rule belong to the custom business transaction. Alternatively, you can create a match rule that generates multiple named business transaction based on criteria in the request.

For example, the following configuration defines a custom match rule that matches by superclass. In the example, the entry point matches the `process()` method defined in classes that extend the `com.acme.AbstractProcessor` superclass. In our example, the superclass is extended by the subclasses `SalesProcessor`, `InventoryProcessor`, `BacklogProcessor`.

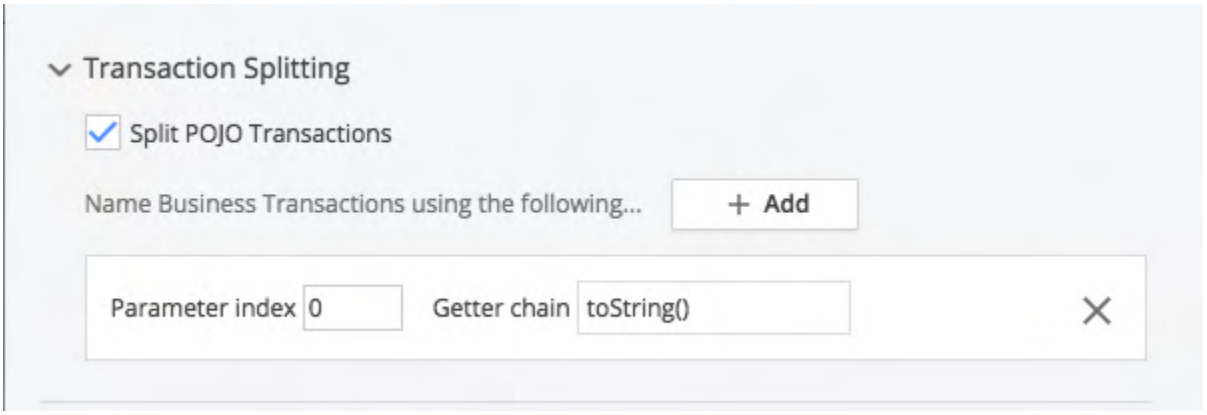
1. Create A custom rule that matches the superclass matches all of those subclasses:
2. Split the transaction on class name, you can have separate business transactions created for the respective subclasses. AppDynamics names the transactions with the class name, prepending the match rule name to the class name. For example, `Process.SalesProcessor`, `Process.InventoryProcessor`, and `Process.BacklogProcessor`.



It's possible that parameter values carry what would be meaningful identifiers for business transactions in your application. For example, say the `jobType` parameter in the following method may have the values of `Sales`, `Inventory` or `Backlog`.

```
public void process(String jobType,String otherParameters...)
```

You can split the transaction based on parameter value by indicating the zero-based position of the parameter in the method signature, 0 in the example:



The `toString()` method indicates how the value of the parameter should be rendered. See Using Getter Chains for more information. As indicated in the dialog, you can use other transaction splitting criteria as well, including thread ID, method name, class name, and so on.

The name of the rule is prepended to the dynamically-generated name to form the business transaction name.

Exclude Matches from Transaction Splitting

An exclude rule defines criteria which, when matched by a transaction instance, prevents transaction splitting from occurring as otherwise configured. In effect, this nullifies the effects of transaction splitting for the subset of transactions matched by the exclude condition.

Monitor Java Interface Static and Default Methods

As of Java 8, Java interfaces can include static and default methods. This page describes how to monitor these types of interface methods.

> Note that another Java language feature introduced in Java 8, lambda method interfaces, are not supported by the AppDynamics Java Agent.

Interface Static Methods

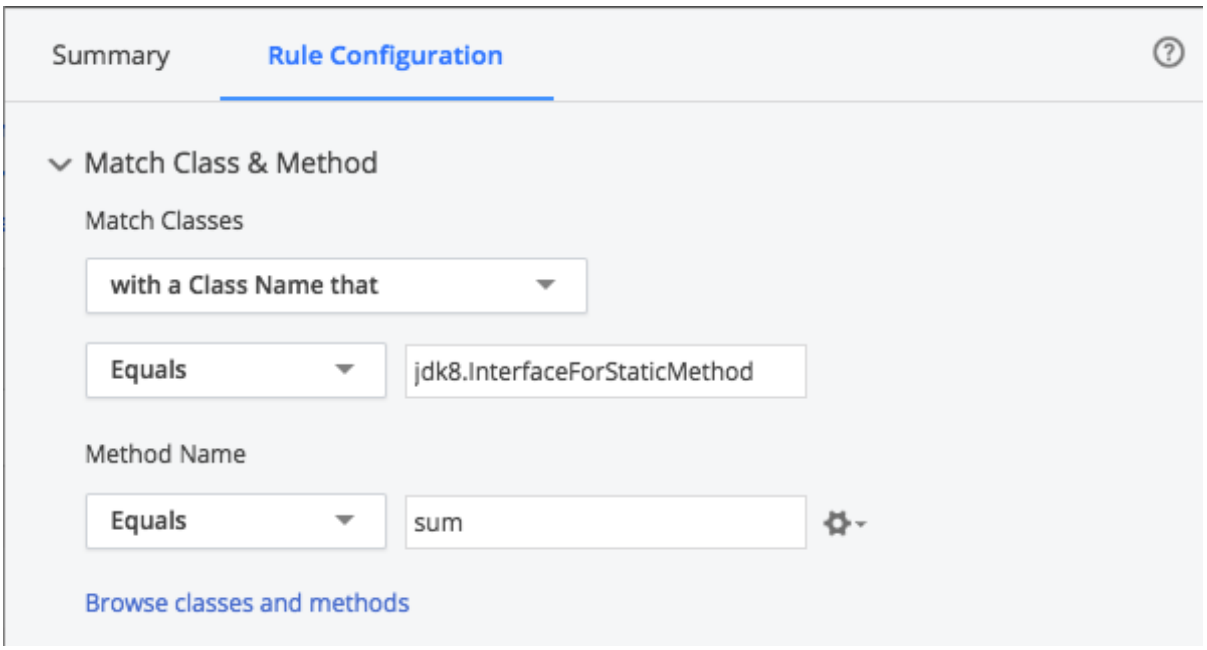
The following code snippet shows an example static interface method:

```
package jdk8;

interface InterfaceForStaticMethod
{
    static int sum(int acc, int x)
    {
        return acc + x;
    }
}
```

The actual bytecode for the method would reside in the compiled class file, so a class match against the interface class in the AppDynamics configuration would result in the instrumentation of that class for a business transaction.

To instrument such methods, create a POJO business transaction match rule for the `interfaceForStaticMethod`, such as:



In this case, the rule would match classes with a Class Name of `jdk8.InterfaceForStaticMethod`.

Interface Default Methods

Interfaces can also define default methods. As with interface static methods, the bytecode for default methods resides in the compiled class files.

When configuring instrumentation for a default method:

- If the targets are classes that do *not* override the default method, target the interface via a class match
- If the targets are classes that do override the default method, target the interface via an interface match

Instrument an Interface Default Method Directly

Given the following interface:

```
package jdk8;

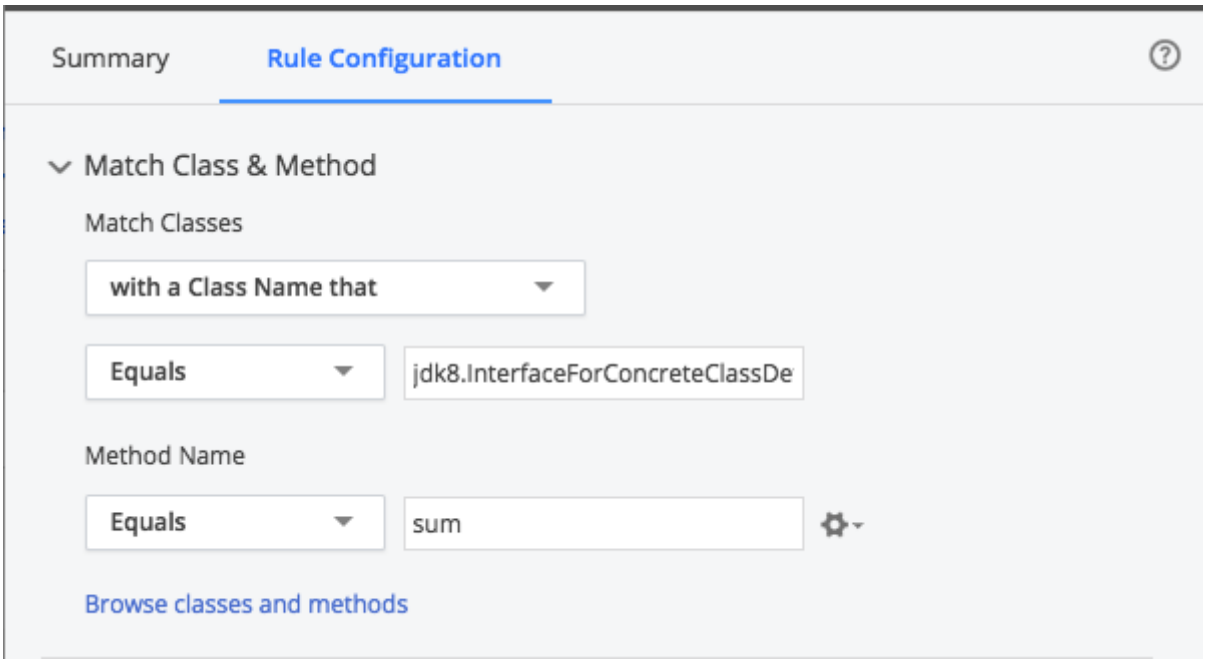
interface InterfaceForConcreteClassDefaultMethod
{
    default int sum(int acc, int x)
    {
        return acc + x;
    }
}
```

And its implementing class:

```
package jdk8;

class ClassForDefaultMethod implements InterfaceForConcreteClassDefaultMethod
{
}
```

In this example, the `ClassForDefaultMethod` does *not* override the default method. Therefore, you can use a class match against the interface such as:



Instrument an Overridden Default Method

Given the following interface:

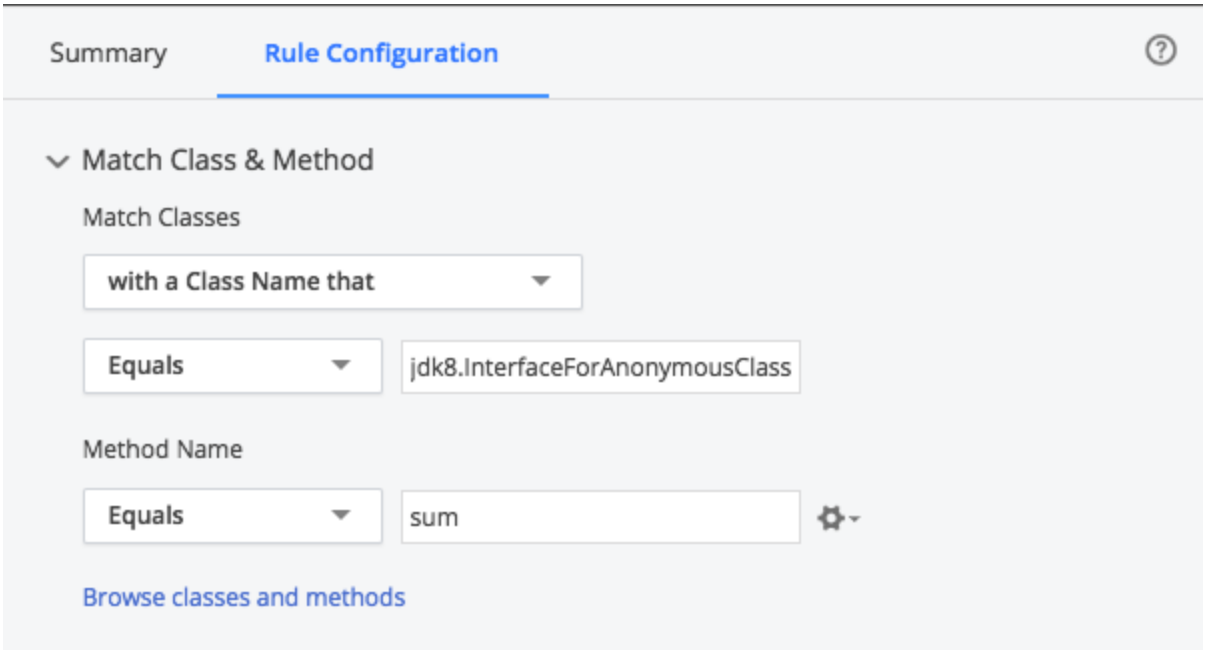
```
package jdk8;

interface InterfaceForAnonymousClassDefaultMethod
{
    default int sum(int acc, int x)
    {
        return acc + x;
    }
}
```

And:

```
InterfaceForAnonymousClassDefaultMethod i = new InterfaceForAnonymousClassDefaultMethod()
{
        @Override
        public int sum(int acc, int x)
        {
                return acc + x;
        }
};
```

Because the method is overridden, the target `bytecode` resides in the anonymous class, `InterfaceForAnonymousClassDefaultMethod`. So you need to create an interface match POJO business transaction match rule such as:



Instrument Java Lambda Expressions

This page describes Lambda expressions, a Java language feature introduced in Java 8 that enables a functional style of programming in Java.

Java Lambda

Here's a short example:

```
LambdaMethodInterface i = (acc, x) -> acc + x;
```

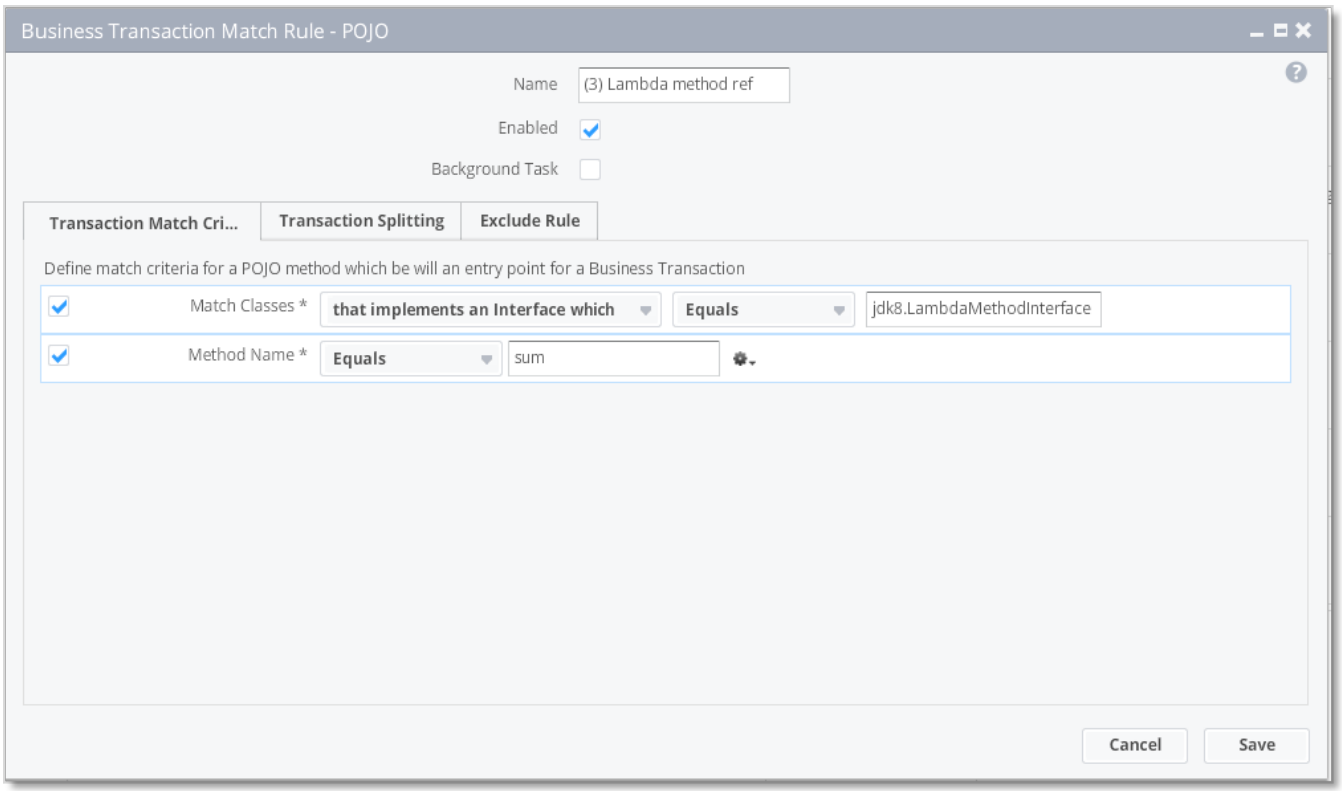
In the JVM, this creates an anonymous instance of the associated `FunctionalInterface`, as shown below:

```
package jdk8;

@java.lang.FunctionalInterface
interface LambdaMethodInterface
{
    int sum(int acc, int x);
}
```

Instrument Java Lambda Instances

To instrument lambda instances, create a POJO business transaction match rule for the `LambdaMethodInterface`, for example:



This sample rule matches classes that implement the interface name `jdk8.LambdaMethodInterface`.

Web Service Entry Points

When your application uses Web Services to service user requests, AppDynamics intercepts the Web Service invocations and names requests based on the Web Service action names and operation name. A Web Service entry point is a Web Service endpoint that is being invoked.

This is relevant only when the Web Service invocation is part of the entry point tier or in cases where the transaction should be continued in a downstream tier using a correlation header transported in the SOAP Envelope.

Web Service invocations are usually preceded by a dispatcher Servlet, but identification is deferred to the Web Service endpoints. This configuration ensures that the requests are identified or correlated based on the Web Service or its payload and not based on the generic URL for the dispatcher Servlet.

Default Naming

When the Web Service endpoint is invoked, the request is named after the Web Service name and the operation name.

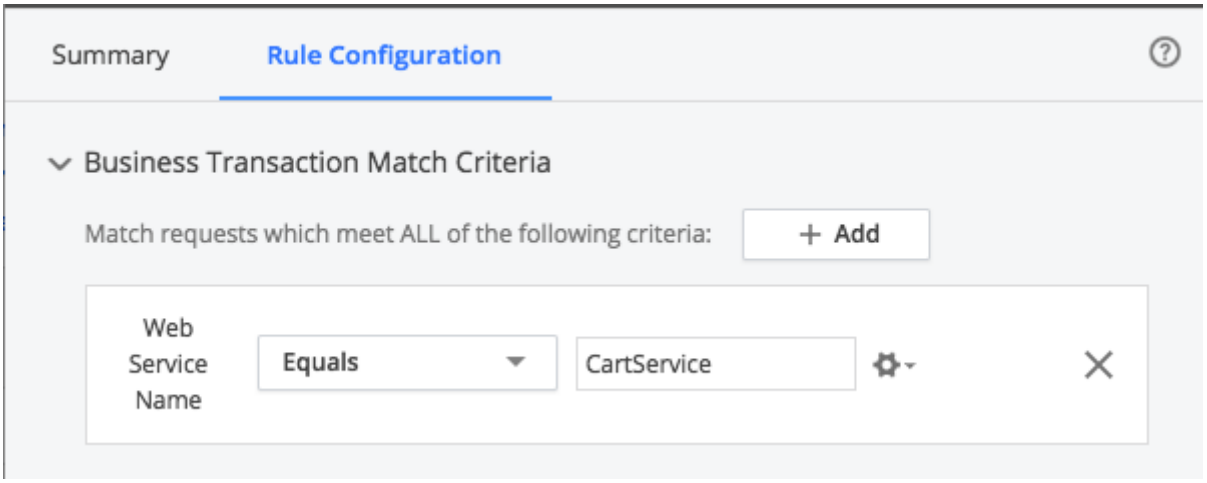
For example, if a service called `CartService` is invoked with the `Checkout` operation, the is named `CartService.Checkout`.

You can rename or exclude these automatically discovered transactions. See Organize Business Transactions.

Custom Match Rules for Web Services

You can aggregate different Web Service requests into a single business transaction using the web service name or the operation name. You do this by creating custom match rules for Web Services. See Custom Match Rules for information about accessing the configuration screens.

This example names all operations for the Web Service named `CartService`:



Exclude Rules

To exclude specific Web Services or operation names from detection, add an exclude rule. See Custom Match Rules. The criteria for Web Service exclude rules are the same as those for custom match rules.

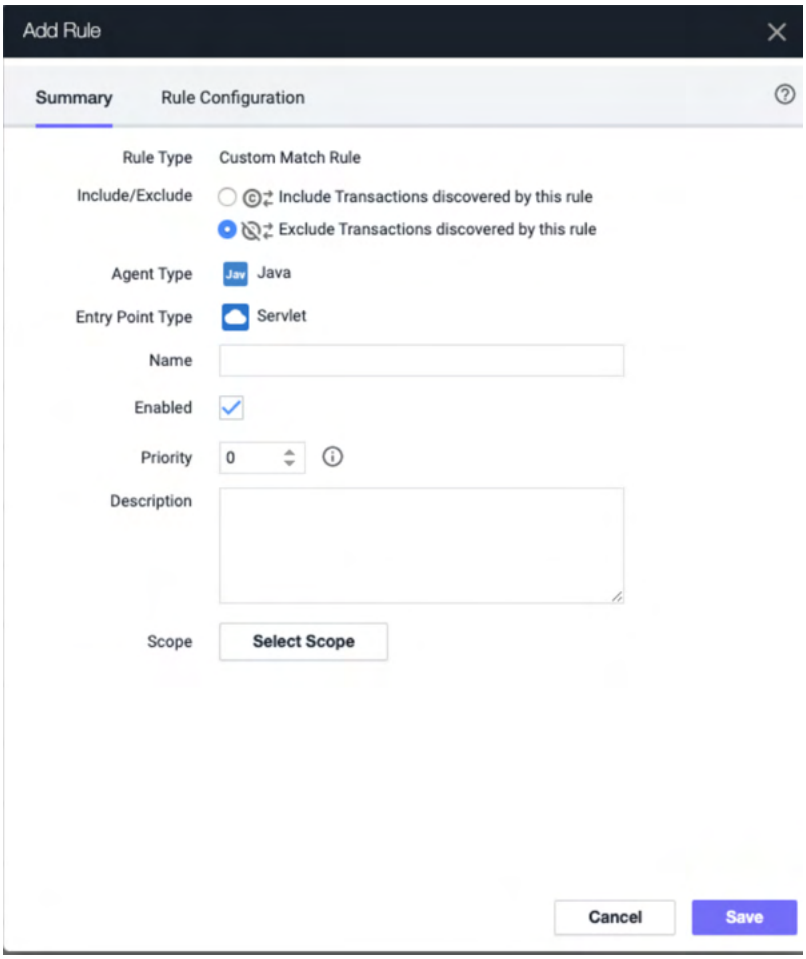
Transaction Splitting for Web Services Based on Payload

1. Disable the Web Service automatic transaction discovery.
2. Disable the following default exclude rules:
    - Apache Axis Servlet
    - Apache Axis2 Servlet
    - Apache Axis2 Admin Servlet
3. Add the custom match rule for Axis or Axis2 Servlet (based on the version being used) and split the transaction using payload or request depending on the pattern in your scenario.

Exclude Rule for JBoss Correlation

To detect Web Service entry and to support correlation in JBoss SOAP Web Service, you must create a Servlet exclude rule by excluding the class name, `org.jboss.wsf.stack.cxf.CXFServletExt` as follows:

1. In **Add Rule**, select **Custom Match Rule**.
2. Select **Servlet** as the **Entry Point Type**.
3. Select **Exclude Transactions discovered by this rule.**



4. Click **Save**.
5. In the Rule Configuration tab, specify the class name as `org.jboss.wsf.stack.cxf.CXFServletExt`.
6. Click **Save**.

Binary Remoting Entry Points for Apache Thrift

Apache Thrift is a binary remoting protocol. Cassandra uses the Thrift protocol to achieve portability across programming languages. Applications written in many different languages can make calls to the Cassandra database using the Thrift protocol.

AppDynamics is preconfigured to detect business transaction entry points for Cassandra with Thrift framework applications. Note that Binary Remoting entry points are not available as Service Endpoints.

AppDynamics measures performance data for Thrift transactions as for any other transaction. Thrift entry points are POJO-based. The response time for the transaction is measured from the POJO entry point, and the remote calls are tracked the same way as remote calls for a Servlet's Service method.

Default Naming for Binary Remoting (Thrift) Entry Points

When the automatic discovery for a request using Binary Remoting (Thrift) protocol is enabled, AppDynamics automatically identifies all the transactions and names them using the following format:

```
RemoteInterfaceClassName:MethodName
```

Enabling Detection for Binary Remoting (Thrift) Entry Points

Binary Remoting (Thrift) entry points are enabled by default, but if you are not seeing them in the Tier Flow Map, verify that they have been enabled in the Java automatic transaction detection rule. For more information, see Transaction Detection Rules.

When enabled, you can see Thrift calls from calling tiers to the Cassandra database in the List View of the Tier Dashboard. Since the transactions using the Thrift protocol are POJO-based, they appear as POJO in the Type column.

Creating Custom Match Rules for Binary Remoting (Thrift) Requests

If you are not getting the required visibility with the auto-discovered transactions, you can configure custom match rules and transaction splitting for specific classes and methods.

When creating the custom match rule, choose POJO as the rule type. The rule should be defined on the class and method you want to use as the entry point. Someone who is familiar with your application code may need to help you make this determination.

For information about and examples of the various POJO-based business transaction match rules you can create for Binary Remoting (Thrift), see POJO Entry Points. The rules for Binary Remoting (Thrift) entry points are the same as those for POJO entry points.

CometD Support

Related pages:

- Jetty Startup Settings
- Exclude Rules

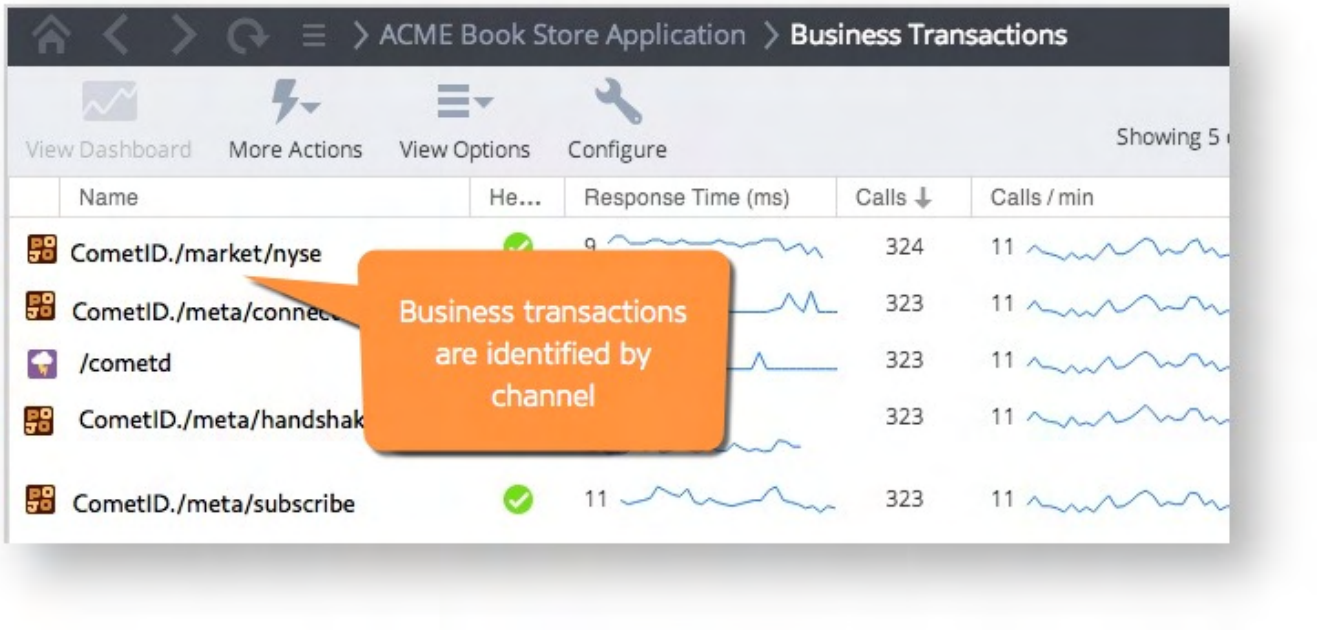
CometD is an HTTP-based event routing bus for transporting asynchronous messages over HTTP. CometD provides both a JavaScript API and a Java API and is often used for real-time interactive applications such as multi-player games and chat rooms.

CometD is distributed as a WAR file. When you instrument the CometD application container that runs the WAR (typically a Jetty server) with the Java agent, AppDynamics automatically detects and tracks CometD activity, as described in the following section.

About the Built-in Support

AppDynamics supports both the WebSocket and the HTTP long-polling client transports in CometD. It tracks CometD messages as they traverse channels using POJO rules that split business transactions based on channel name.

Accordingly, business transactions in AppDynamics correspond to CometD channels. The following screenshot shows a business transaction generated for the `/market/nyse` channel:



AppDynamics provides built-in exclude rules for the servlets that implement CometD transport functions, org.cometd.annotation.AnnotationComet and org.cometd.server.CometdServlet. For best results, you should exclude tracking for the container servlet as well, as described in the next section.

Create Container Transport Servlet Exclude Rules

CometD is implemented as a servlet. Since the servlet implements transport activity, you should create exclude rules for the servlets that perform the CometD transport function. This ensures that the results of business transaction discovery do not obscure the POJO-based BTs corresponding to CometD channels.

CometD is generally contained within a Jetty container, so the rules should exclude the Jetty container tracking in most cases. For Jetty, you should configure exclude rules for these Jetty servlets:

- EQUALS, `org.eclipse.jetty.server.handler.ContextHandler`
- EQUALS, `org.eclipse.jetty.servlet.ServletContextHandler`
- ENDSWITH, `jetty.plugin.JettyWebAppContext`

If using another type of container, you should create similar exclude rules for the equivalent supporting classes.

JAX-RS Support

The Java Agent supports Jersey 1.x and 2.x by default.

Business transaction entry points are named using the following app agent node properties:

- rest-num-segments
- rest-transaction-naming
- rest-uri-segment-scheme

Business Transaction Naming Scheme

By default, the Java Agent makes a best-effort basis to name JAX-RS-based transactions using the class annotation, method annotation, and HTTP method of the detected entry point, in the following form:

```
{class-annotation}/{method-annotation}.{http-method}
```

In some situations where the annotation values are not available, the agent may name the transaction as follows:

```
{fully qualified class name}.{method}
```

For example, when the JAX RS service endpoints don't have the class and method names in them explicitly, but the annotations are inferred in some other way.

For annotation inheritance cases, the agent attempts to fetch the annotations from the superclasses and interfaces as described by the JAX RS specification. The specification makes the following recommendation:

> *For consistency with other Java EE specifications, it is recommended to always repeat annotations instead of relying on annotation inheritance.*

Repeating annotations on the endpoint methods enables the Java Agent to name business transaction according to the default scheme.

rest-transaction-naming Variables

The `rest-transaction-naming` node agent property enables you to control naming for JAX-RS-based business transactions. You can use variables to have the transactions named with runtime values, for example, based on the method name, class name, parameter values, and more. For a full list of variables, see App Agent Node Properties Reference.

You can add a getter chain after any of the parameters to operate on the parameter values. To do so, add ':' followed by the getter chain. Standard getter chain escape requirements and rules apply.

Naming Examples

The following example shows naming based on a part of the value of the sixth parameter:

```
rest-transaction-naming={param-5:toString().toLowerCase().subString(5,20)}
```

The characters '{' and '}' are reserved. If used in getter chains or as text in the name the characters must be escaped with '\'. E.g. {{`class-name`}} would display as {`com.singularity.RestObject`}.

You can use characters in addition to parameters. For example, in the following example, the class name and method name are separated by a slash:

```
rest-transaction-naming={class-name}/{method-name}
```

Based on this example, when the Java Agent sees a REST resource with a class name of `com.company.rest.resources.Employees` with a `CreateNewEmployee` method, it names the business transaction `com.company.rest.resources.Employees/CreateNewEmployee`.

Spring Integration Support

In Spring-based applications, Spring Integration enables lightweight messaging and supports integration with external systems via declarative adapters.

The Java Agent automatically discovers entry points based on the `MessageHandler` interface. The Java Agent automatically discovers exit points for all Spring Integration Release 2.2.0 channels except `DirectChannel`.

The Java Agent supports tracking application flow through Spring Integration messaging channels. For pollable channels:

- A continuing transaction is tracked if the Spring Integration framework is polling for messages.
- If the application code polls for messages in a loop, the span of each loop iteration is tracked as a transaction. Tracking begins when the loop begins and end it when the iteration ends.

Entry Points

Originating transactions begin with `MessageHandler.handleMessage()` implementations. If the incoming message is already recognized by the Java Agent then a continuing transaction is started.

Exit Points

Exit points are based on `MessageChannel.send()`.

Most message channels exist inside the JVM. To represent this arrangement in a flow map, the Controller UI shows a link from the tier to the message channel component name (bean name) and then back to the tier.

Spring Integration Support Customizations

The following sections describe typical scenarios for tuning the default Spring Integration monitoring.

Track Application Flow Before Message Handler is Executed

In cases where a lot of application flow happens before the first MessageHandler gets executed, you should enable tracking the application flow as follows:

- Find a suitable POJO entry point and configure it.
- Set the enable-spring-integration-entry-points node property to `false`. This property is set to `true` by default.
- Restart the application server

Limit Tracking of Looping Pollable Channels

To safeguard against cases where `pollableChannel.receive()` is not called inside a loop, you can ensure that the Java Agent tracks a pollable channel loop only if it happens inside a class/method combination similar to that defined in the following example.

You can configure the spring-integration-receive-marker-classes node property for each class/method combination that polls messages in a loop, in which case only those class/methods identified in this node property are tracked.

```
class MessageProcessor {
   void process() {
      while(true) {
          Message message = pollableChannel.receive()
      }
   }
}
```

For example, for the loop above, set the `spring-integration-receive-marker-classes` node property as follows and restart the application server:

```
spring-integration-receive-marker-classes=MessageProcessor/process
```

The `spring-integration-receive-marker-classes` node property must be configured before the method `process()` gets executed for any changes to take effect. Restart the application server after setting this property.

.NET Business Transaction Detection

Related pages:

- Transaction Detection Rules
- Organize Business Transactions
- Configuration Import and Export API
- .NET Agent for Linux Business Transaction Configuration

The page lists different types of ASP.NET entry points for business transactions:

- POCO Entry Points
- ASP.NET Entry Points
- WCF Entry Points
- ASP.NET Web Service Entry Points
- ASP.NET Core Instrumentation Options
- Name MVC Transactions by Area, Controller, and Action

POCO Entry Points

Certain types of applications, such as Windows Services or standalone applications, may not have entry points that are automatically detected by the .NET Agent.

For these cases, you can create custom match rules for Plain Old CLR Objects (POCOs).

POCO Custom Match Rules Overview

To create a POCO custom match rule, define the custom match rule on the .NET class/method that is the most appropriate entry point for the business transaction. Someone who is familiar with your application code can help make this determination. You can also refer to the Custom Match Rules page.

You can optionally enable the Background Task option to have the transaction monitored as a background task. AppDynamics reports only Business Transaction metrics for background task transactions. It does not aggregate response time and call metrics at the tier and application levels for background tasks. See Monitor Background Tasks for more information.

After you create the rule, AppDynamics detects traffic on the matched entry point method and registers a business transaction for it, naming the transaction with the name of the custom match rule.

> By default, you have to wait one minute and then restart the application to apply instrumentation changes required for new POCO entry points. If you create an application-level POCO that applies to a tier that has not yet registered with the Controller, you may need to restart the application after the tier registers in order to see the business transaction.
>
> You can enable Runtime Reinstrumentation for the .NET Agent so that you don't need to restart your application after instrumentation changes.

As an alternative to defining POCO entry points as described here, you can use the transaction discovery tool to create match rules based on discovered transactions. See Business Transaction Discovery Sessions.

> The .NET Agent for Linux supports the configuration of simple POCO business transactions through the Controller UI. See .NET Agent for Linux Business Transaction Configuration

Define a POCO Entry Point

On an originating tier, a POCO entry point is the method that starts the business transaction. If the POCO entry point is on a downstream tier, it may correlate to an upstream exit point. When defining a POCO entry point, it is important to choose a method that begins and ends every time the business transaction executes. See Business Transactions.

Good candidates for POCO entry points include the following:

- A method in a socket application that executes every time a client connects
- A loop in a standalone application that batch processes records via a web service call. For example, an expense reporting system that loops through approved expenses to submit them for reimbursement.
- A Windows service that regularly executes a database call to check for new jobs to process. For example, consider the sample application logic below:

```
using System.Threading;
using System;
using System.Threading.Tasks;
using System.Configuration;
using System.Collections.Generic;

namespace JobProcessor {
    class JobProcessorCore {
        private void ProcessJob()
            {
            var logic = new JobManagement();
            while(running)
                {
                    var jobs = logic.GetJobs(); // Query in-memory Database for a list of jobs to process
                    foreach (var job in jobs)
                        {
                        logic.GetJobDetails(job, out type, out parameters); // Obtain the details on the
specific job at hand
                        ProcessJob(type, parameters);  // Execute the job in a separate thread
                        }

                    if (jobs.Count == 0)   // if there was no job to process, wait 1 minute
                        Thread.Sleep(60000);
                }
        }
    }
}
```

In the example, the POCO entry point would be defined on the namespace JobProcessor, class JobProcessorCore and method ProcessJob.

Task-Based POCO Entry Point Methods

AppDynamics automatically tracks POCO transactions for asynchronous, Task-based method entry points as end-to-end transactions. End-to-end latency metrics reflect the time it takes to complete the asynchronous actions associated with such methods.

To be tracked as an end-to-end transaction, the method you define as the POCO entry point must return a `Task` object and use the `async` modifier. In other words, the method signature should look similar to the following:

```
public async Task<int> SearchBestPrice()
{
   ...
}
```

AppDynamics records the response times for such methods as it does for other types of methods—the response time reflects the time from when the method is called to when it returns control to the calling thread. However, for asynchronous task-based methods, AppDynamics records the time it takes for the asynchronous task to be fulfilled as an end-to-end transaction latency metric.

> For .NET Agents >= 20.3.0, time spent on asynchronous tasks is reported in average response time (ART) but not reported in the end-to-end latency metric.

See End-to-End Latency Performance.

ASP.NET Entry Points

Related pages:

- Transaction Detection Rules
- Business Transactions

AppDynamics automatically detects entry points for client requests to ASP.NET applications. If the request occurs on an originating tier, the method or operation marks the beginning of a business transaction and defines the transaction name. In most cases, this type of entry point maps to a user request or action such as "Cart/Checkout". AppDynamics allows you to configure transaction naming based upon the ASP.NET request.

For information on how to configure default ASP.NET transaction detection, see URI Based Entry Points.

Custom Match Rules for ASP.NET Transactions

Custom match rules provide greater flexibility for transaction naming. When you define a match rule, AppDynamics uses the rule name for the business transaction name.

See Custom Match Rules for general information on how to create custom match rule.

The .NET Agent for Linux supports the configuration of simple ASP.NET business transactions through the Controller UI. See .NET Agent for Linux Business Transaction Configuration.

When AppDynamics detects a request matching your specified criteria, it identifies the request using your custom name. You can use the following criteria to match transactions:

Method: Match on the HTTP request method, GET, POST, PUT or DELETE.

With automatic discovery for ASP.NET transactions enabled, configuring the match on `GET` or `POST` causes the agent to discover both GET and POST requests. If you only want either `GET` or `POST` requests for the transaction, consider the following options:

- Disable automatic discovery for ASP.NET transactions.
- Create an exclude rule for the method you don't want: `GET` or `POST`.

URI: Set the conditions to match for the URI.

- For rules on regular expressions for .NET, see .NET Framework Regular Expressions.
- Optionally click the gear icon to set a NOT condition.
- You must set a URI match condition in order to use transaction splitting.

HTTP Parameter: Match on HTTP parameter existence or a specific HTTP parameter value.

Header: Match on a specific HTTP header's (parameter's) existence or a specific HTTP header value.

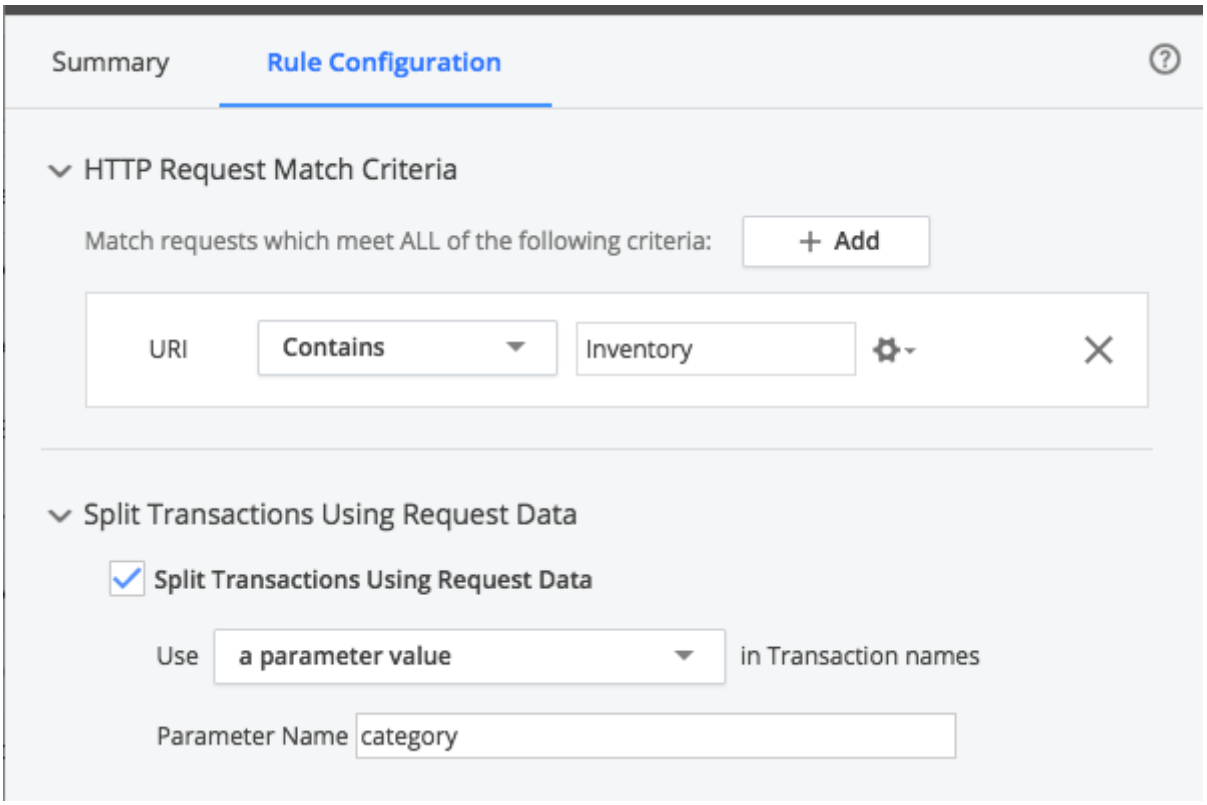
- Hostname: Match on the server hostname. Optionally click the gear icon to set a NOT condition.
- Port: Match on the server port number. Optionally click the gear icon to set a NOT condition.
- Class Name: Match on the ASP.NET class name. Optionally click the gear icon to set a NOT condition.
- Cookie: Match on cookie existence or specific cookie value.
- Hostname, Port, and Class Name options are non-functional in the .NET Agent 4.5.9 for Linux.

Split Custom ASP.NET Transactions

AppDynamics lets you further refine ASP.NET custom transaction names using transaction splitting.

- To use transaction splitting, you must specify URI match criteria for the custom match rule.
- The *Split Transactions Using Request Data* options work the like the automatic transaction detection configuration options described in URI Based Entry Points.

For example, you have a custom match rule named `MyTransaction` that matches the following URL: `http://example.com/Store/Inventory?category=electronics`. You can split the transaction on the parameter value as follows:



The .NET Agent names the resulting transaction `MyTransaction.electronics`.

WCF Entry Points

Related pages:

- Configure Business Transaction Detection for .NET
- Windows Communication Foundation (Microsoft Windows Reference)

AppDynamics automatically detects entry points for client requests to Windows Communication Foundation (WCF) services. If the request occurs on an originating tier, the method or operation marks the beginning of a business transaction and defines the transaction name. For information on originating tiers, see Business Transactions.

The .NET Agent detects async entry points for the following patterns:

- Task-based asynchronous operations
- IAsyncResult Begin-End Asynchronous pattern

Controlling Correlation Header Location

In 4.4 and earlier, when WCF calls execute on a downstream tier, AppDynamics includes them as part of the business transaction from the originating tier. If the downstream WCF service uses the NetTcpBinding in WCF, you must register the app agent node property `enable-soap-header-correlation` with a value of `true`. See 'enable-soap-header-correlation' on App Agent Node Properties Reference. Otherwise, the .NET Agent will detect the WCF call as the entry point for a new business transaction.

In the following table, when the node property is set to the default (false), the correlation message with header will only successfully deliver when sent over HTTP, not over SOAP with an HTTP wrapper or other protocol. When the node property is set to true, the correlation message with header will succeed in all scenarios.

	enable-soap-header-correlation	HTTP	SOAP (HTTP)	SOAP (other)
4.4 **default**	**false**	X	-	-
4.4	true	X	X	X

In 4.4.1 and later, the `enable-soap-header-correlation` node property and the default behavior for SOAP calls over non-HTTP protocols in WCF have changed. In addition, a new property, `disable-soap-header-correlation-non-http`, has been added. See `disable-soap-header-correlation-non-http` on App Agent Node Properties Reference. With the new default AppDynamics will, by default, include downstream WCF services using NetTcpBinding as part of business transactions.

The following table captures the changes and additions of the correlation data for the following use cases:

- When the node properties for both `enable-soap-header-correlation` and `disable-soap-header-correlation-non-http` are set to the default (false), the correlation message with header will only successfully deliver when sent over HTTP or SOAP wrapped in a protocol other than HTTP.
- When the node property for `enable-soap-header-correlation` is set to `true`, and the `disable-soap-header-correlation-non-http` node property is set to the default (`false`), the correlation message with header will succeed in all scenarios.
- When the node property for `enable-soap-header-correlation` is set to the default (`false`), and the `disable-soap-header-correlation-non-http` node property is set to `true`, the correlation message with header will only successfully deliver when sent over HTTP, not over SOAP with an HTTP wrapper or other protocol.
- When the node properties for both `enable-soap-header-correlation` and `disable-soap-header-correlation-non-http` are set to `true`, the correlation message with header will only successfully deliver when sent over HTTP or SOAP with an HTTP wrapper, not over SOAP with another protocol.

When upgrading from 4.4.0 to 4.4.1 or later, the `enable-soap-header-correlation` property retains its previous value and `disable-soap-header-correlation-non-http` defaults to `false`.

	enable-soap-header-correlation	disable-soap-header-correlation-non-http	HTTP	SOAP (HTTP)	SOAP (other)
4.5 **default**	**false**	**false**	X	-	X
4.5	true	**false**	X	X	X
4.5	**false**	true	X	-	-
4.5	true	true	X	X	-

By default, the .NET Agent does not correlate EUM transactions for WCF entry points. You can enable EUM correlation for WCF using the **wcf-enable-eum** node property. This feature is useful when an instrumented mobile application calls directly into a WCF entry point. See 'wcf-enable-eum' on App Agent Node Properties Reference.

Automatic Naming for WCF Transactions

By default, the AppDynamics auto-detection naming scheme identifies all WCF transactions using the service name and operation name:

```
ServiceName.OperationName
```

For example, a web service for a travel website books reservations from client front ends: `TravelService.BookTravel`. You can rename or exclude automatically discovered transactions from the Business Transactions List.

Custom Match Rules for WCF Transactions

Custom match rules provide flexibility for WCF transaction naming. When you define a match rule, AppDynamics uses the rule name for the business transaction name. See Custom Match Rules for more information.

When AppDynamics detects a request matching your specified criteria, it identifies the request using your custom name. You can match based upon:

Web Service Name

- For rules on regular expressions for .NET, see .NET Framework Regular Expressions.
- Optionally use the gear dropdown to set a NOT condition.

Operation Name

- For rules on regular expressions for .NET, see .NET Framework Regular Expressions.
- Optionally use the gear dropdown to set a NOT condition.

For example, you could report all operations for a TravelService, such as `SearchTravel` and `BookTravel`, as one business transaction.

ASP.NET Web Service Entry Points

AppDynamics automatically detects entry points for client requests to ASP.NET web services. If the request occurs on an originating tier, the method or operation marks the beginning of a business transaction and defines the transaction name. For more information on originating tiers, see Configure Business Transaction Detection for .NET.

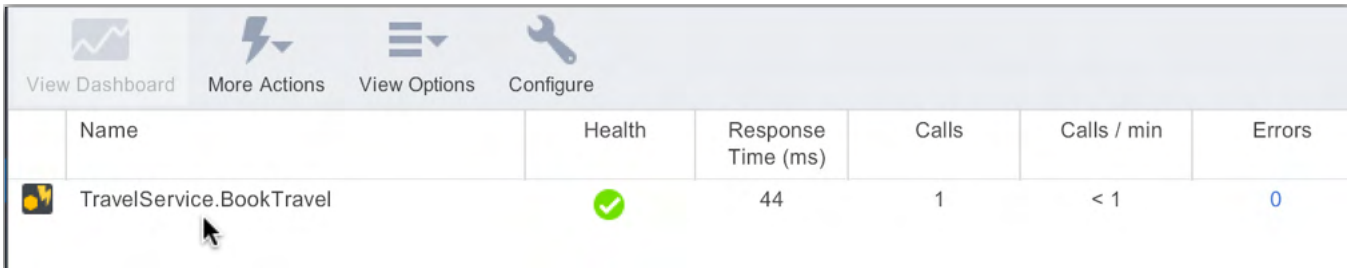
When web service calls execute on a downstream tier, AppDynamics includes them as part of the business transaction from the originating tier.

Automatic Naming for Web Service Transactions

By default, the AppDynamics auto-detection naming scheme identifies all web service transactions using the service name and operation name:

```
ServiceName.OperationName
```

For example, a web service for a travel website books reservations from client front ends.



You can rename or exclude automatically discovered transactions from the business transaction list.

Custom Match Rules for ASP.NET Web Service Transactions

Custom match rules provide flexibility for ASP.NET web service transaction naming. When you define a match rule, AppDynamics uses the rule name for the business transaction name.

To access the Custom Match Rules pane, see 'To create custom match rules for. NET entry points' on .NET Business Transaction Detection.

To create an ASP.NET web service custom match rule

1. In the **Custom Match Rules** pane, click the plus symbol (**+**) to add an entry point.
2. Click **Web Service** in the dropdown. Click **Next**.
3. Name the **New Business Transaction Match Rule**.
    - AppDynamics uses the rule **Name** to name the BT.
    - The Controller enables the rule by default. Disable it later if needed.
    - Set the **Priority** for the match rule. AppDynamics applies higher priority rules first.
4. Set one or more of the following match criteria. When AppDynamics detects a request matching your specified criteria, it identifies the request using your custom name.

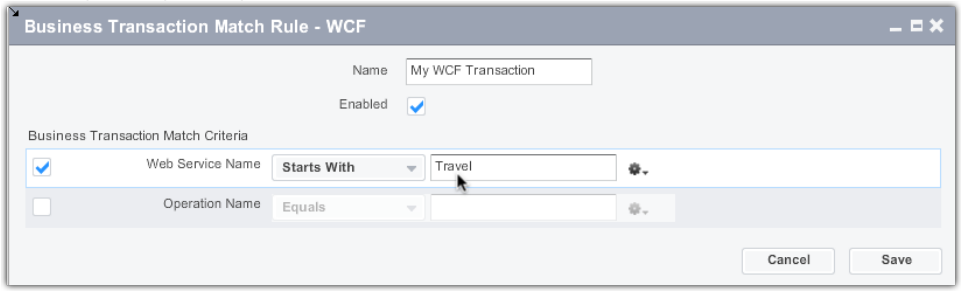
Web Service Name

- For rules on regular expressions for .NET, see .NET Framework Regular Expressions.
- Optionally use the gear dropdown to set a NOT condition.

Operation Name

- For rules on regular expressions for .NET, see .NET Framework Regular Expressions.
- Optionally use the gear dropdown to set a NOT condition.

For example, to report all operations for a TravelService, such as `SearchTravel` and `BookTravel`, as one business transaction:



5. Click **Create Custom Match Rule**.
   The rule appears in the **Custom Match Rule** list. The business application or tier you customized displays a green check in the **Select Application or Tier** pane.

After the agent receives the updated configuration, it discovers the new business transaction and displays it in the Business Transactions list.

ASP.NET Core Instrumentation Options

When you develop an ASP.NET Core application, a pipeline is built by connecting the configured middleware components together.

By default (for example, when you re-use the standard template application startup code), you do not have to define how to build the pipeline or the sequence of middleware components in this pipeline. Everything works as defined. For applications with more complex processing demands, you would typically modify the middleware pipeline and consider how the .NET Agent instruments ASP.NET Core applications. See middleware pipeline information from the Microsoft site.

The .NET Agent can instrument this middleware pipeline at different points along the pipeline. Each instrumentation option provides advantages and disadvantages:

- Resource invoker instrumentation
- Endpoint instrumentation
- HTTP instrumentation
- MVC routing instrumentation

Resource Invoker Instrumentation

- Default setting for ASP.NET Core 2.*x*.
- For ASP.NET Core 3.x and 5.0, set the node property `aspdotnet-core-instrumentation (string) = "ResourceInvoker"` to enable it.

This instrumentation occurs at the end of the pipeline just before it calls the controller action or page handler. This applies to Razor Page handlers, and MVC and Web API controller actions.

The advantage is that only actual action code is instrumented, and the .NET Agent instrumentation does not execute for any other web requests such as: static files and not found paths (`Error Code 404`).

This is the most lightweight instrumentation option. It is the optimal choice for most ASP.NET Core applications, especially where minimal or no customization was applied to the middleware pipeline from the default application code.

Endpoint Instrumentation

- Default setting for ASP.NET Core 3.x and 5.0 (as of .NET Agent 21.3).
- Not available for ASP.NET Core 2.x.

With the improved endpoint routing introduced in ASP.NET Core 2.2, all routing for Web API, MVC, and Razor Pages was merged into a new architecture enabling the .NET Agent to perform instrumentation just after the `EndpointRoutingMiddleware`.

The advantage of using Endpoint instrumentation (instead of Resource Invoker instrumentation) is that the Business Transaction is tracked through the execution of all middleware after the `EndpointRoutingMiddleware`. This provides more visibility into operations performed before the controller action is executed. This includes only middleware that participates in the routing process such as authentication and authorization, but excludes middleware such as health checks, serving up static files, and so on.

Endpoint instrumentation balances full pipeline visibility (which may introduce too much noise and overhead) with no visibility (using Resource Invoker instrumentation).

HTTP Instrumentation

- This setting is available for ASP.NET Core 2.x, 3.x, and 5.0 (as of .NET Agent 21.3).
- Set the node property `aspdotnet-core-instrumentation (string) = "HTTP"` to enable it.

This instrumentation option instruments the full middleware pipeline by instrumenting every single web request. It provides the most visibility of all options. However, when enabling this option, you must consider:

- MVC naming is not possible because the request URL has not been decoded by the routing middleware. It may not be an MVC request; if it is, then controller and action names have not been resolved. All business transactions will use the URL path as its name.
- Because the .NET Agent instrumentation runs and inspects every web request (even requests that result with `404 Errors`, and so on), it has a greater performance impact on the application. To understand the performance impact, you should test this option either in a pre-production environment, or on a limited node count before you enable the option for the entire tier or application.

MVC Routing Instrumentation

- This is the original instrumentation option introduced for ASP.NET Core 1.x. It is the default setting for ASP.NET Core 1.x, and a configurable option for 2.x.
- Set the node property `aspdotnet-core-instrumentation (string) = "Legacy"` to enable this for ASP.NET Core 2.x.

The MVC Routing instrumentation functionality has been deprecated with ASP.NET Core 3.0. It provides full pipeline visibility by instrumenting from the first middleware in the pipeline. However, when attempting to do MVC naming of incoming web requests, it may cause issues. It assumes that you built the middleware pipeline in a specific way, and contains very specific middleware request delegates.

If you are using ASP.NET Core 2.x, and need full pipeline visibility with MVC naming, then this is the only available option for you.

Name MVC Transactions by Area, Controller, and Action

Related pages:

- Business Transactions
- Transaction Detection Rules

By default, the AppDynamics .NET Agent names ASP.NET transactions based upon the URI. For ASP.NET Core on the full framework, the agent uses MVC area, Controller, and action naming by default, and can also be configured to use the URI. This page describes the MVC transaction naming configuration options for both ASP.NET and ASP.NET Core on the full framework.

> For transactions using the MVC naming conventions, the automatic transaction detection configuration settings apply to the MVC segments, not to the URI.

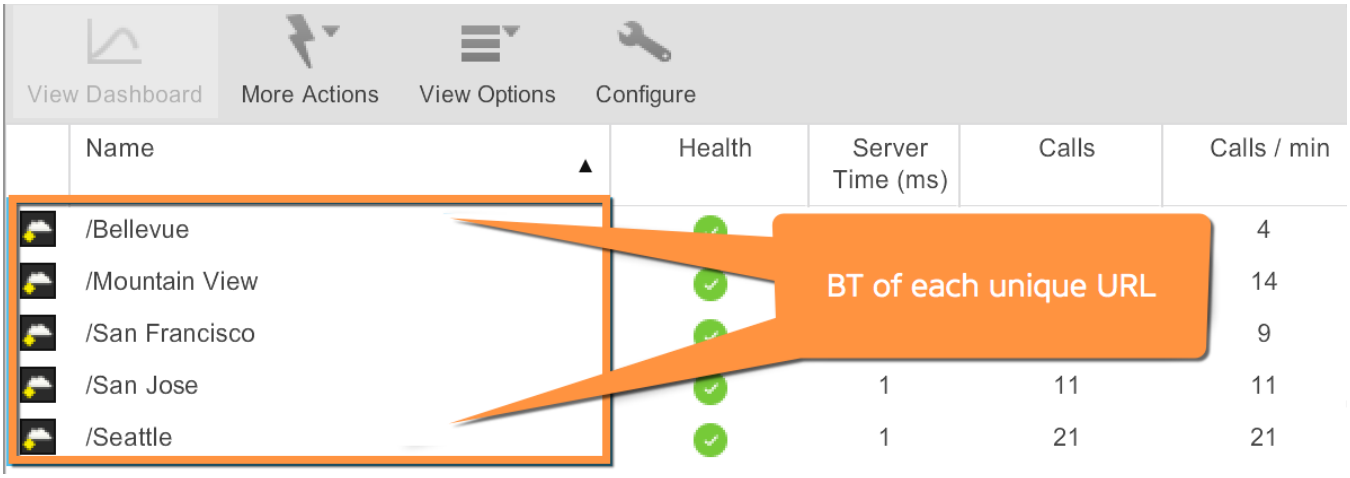
MVC Transaction Naming

For some MVC applications, URI-based naming may not produce meaningful business transaction names. Depending on the URI scheme, it might create many business transactions, pushing you quickly over your business transaction limit and causing most traffic to fall into the *All other traffic* business transaction.

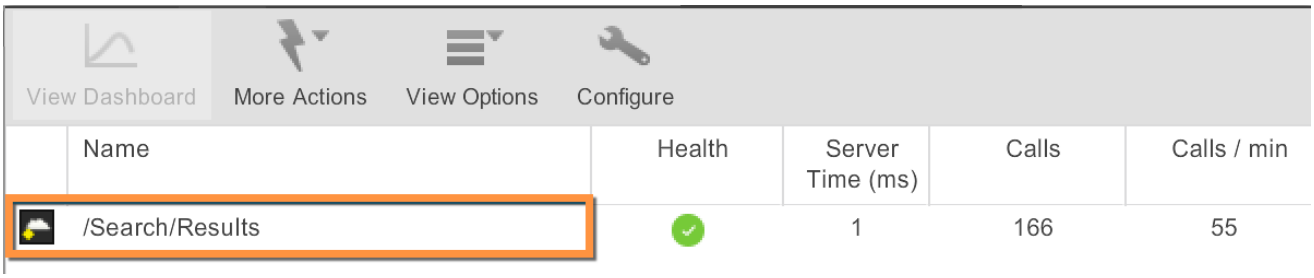
For example, consider an MVC application that indicates store locations in the URL as the result of a city location search, such as the following sample URL:

```
http://myapp.mycompany.com/Bellevue
```

Based on this URL, AppDynamics registers a business transaction named /Bellevue. Each search result with a unique city would similarly generate a unique business transaction.



Alternatively, you can configure the agent to identify transactions by the area, Controller, or action name. After you configure the agent to name transactions by Controller and action, the agent identifies the business transaction based on the Controller and action name, in this case, Search/Results.



In this example, the new business transaction combines search requests for all cities into one transaction, resulting in a more logical unit for measuring application performance.

Similarly, if your application uses areas to logically group Controllers, you can configure the use of area name in the business transaction.

For background information on ASP.NET Routing, see this Microsoft Developer Network article: msdn.microsoft.com/en-us/library/cc668201.aspx

Configure Node Properties for MVC Transactions

You can configure custom MVC transaction naming using node properties. This table describes how to configure node properties for MVC transactions based on the type of entry point.

> The node property supports: MVC 3, MVC 4, and WebAPI transactions.

Type of Entry Point	Name Business Transactions	Transaction Naming Configuration
ASP.NET hosted in IIS (`aspdotnet-mvc-`)	To name business transactions using the Controller and Action values	Register the `aspdotnet-mvc-naming-controlleraction` node property with a value = `true`
	To name business transactions using the Area and Controller values	Register the `aspdotnet-mvc-naming-controllerarea` node property with a value = `true`
	To include all three component values (Controller, Action, and Area) in business transaction names	1. Register both node properties 2. Configure the ASP.NET transaction naming to use three URI segments instead of two (default) See Automatic Transaction Discovery Rules.
ASP.NET Core on the Full Framework (`aspdotnet-core-`)	To name business transactions using the URI	Register both of these node properties with a value = `false` • `aspdotnet-core-naming-controllerarea **` • `aspdotnet-core-naming-controlleraction`
	To omit the Area value from your transaction names	Register the `aspdotnet-core-naming-controllerarea` node property with a value = `false`
.NET Agent for Linux	To use MVC Area, Controller, URI, and Action in business transactions name	Register the `aspdotnet-mvc-naming-controlleraction` node property with a value = `true`

To register a new node property, see App Agent Node Properties.

After you modify any transaction naming configuration, and traffic to the previously registered business transactions terminates, you can delete the old business transactions.

Node.js Business Transaction Detection

Related pages:

- Transaction Detection Rules

This page provides instructions about transaction detection and configuration for the Node.js Agent.

GraphQL Business Transactions

GraphQL is a query and schema definition language that allows servers to expose an API over a HTTP endpoint. You specify the required data in the request payload and send all the requests from the client to a server endpoint.

AppDynamics supports GraphQL when used with the `express-graphql` module, view GraphQL HTTP Server Middleware.

1. On the server side, you define a schema that describes the data and available operations on said data.
2. Then, you invoke operations defined by the schema to retrieve data (through URL requests that receive GraphQL queries) and mutate it.

In the current model, the Node.js Agent identifies an individual Business Transaction from the various requests coming to the server, but the agent does not capture the request payload.

GraphQL Business Transactions determines the business transaction through the incoming endpoint and incoming GraphQL payload, allowing further visibility into the GraphQL Node.js applications. Every GraphQL query that matches this URL is aggregated into one Business Transaction. With GraphQL Business Transactions, the Node.js Agent can be configured in the Controller to identify if an incoming business transaction matches a specific URI (for example, starting with `/QL`) and split that transaction with an operation into multiple transactions.

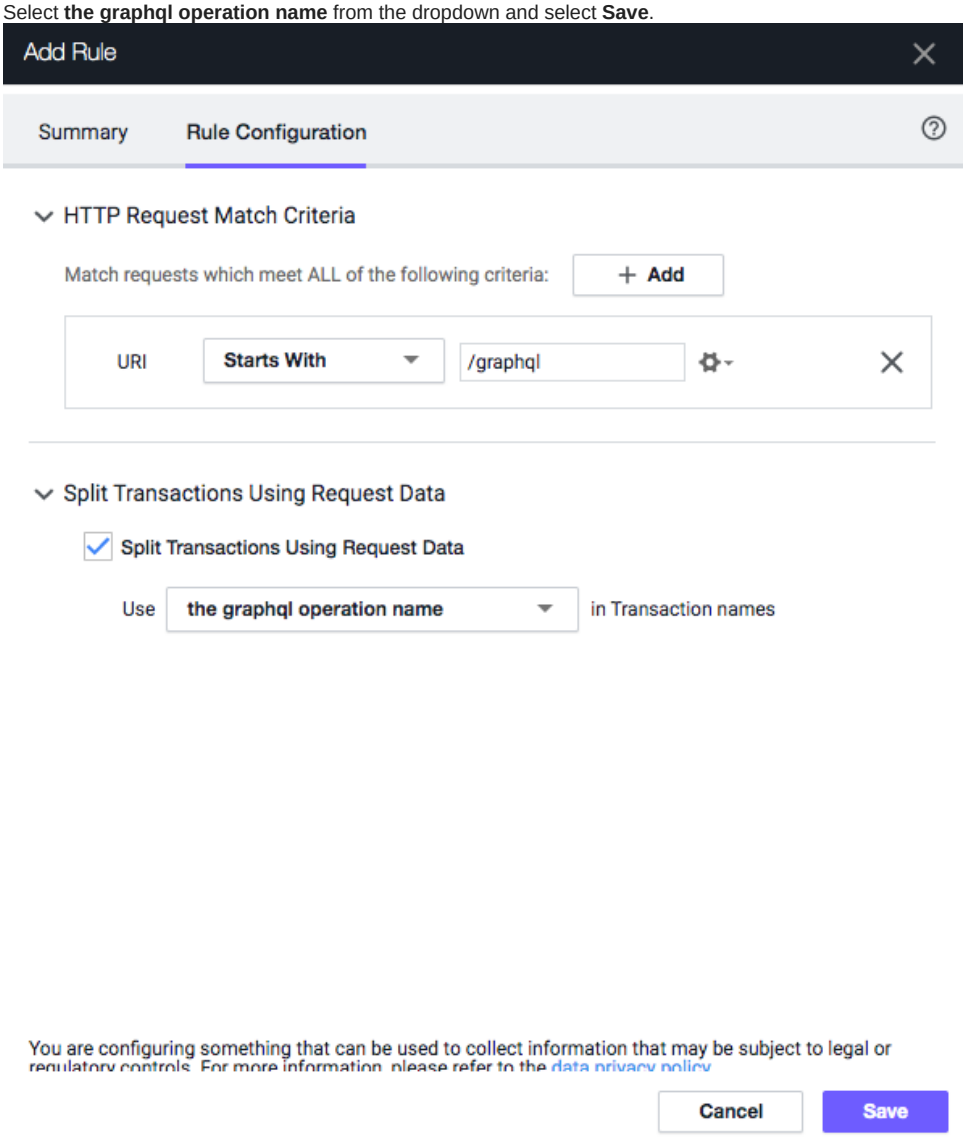
By default, GraphQL Business Transactions is disabled. To configure the Node.js Agent to enable GraphQL Business Transactions, set `enableGraphQL` to `true` in the call to `profile()`:

```
enableGraphQL: true,
```

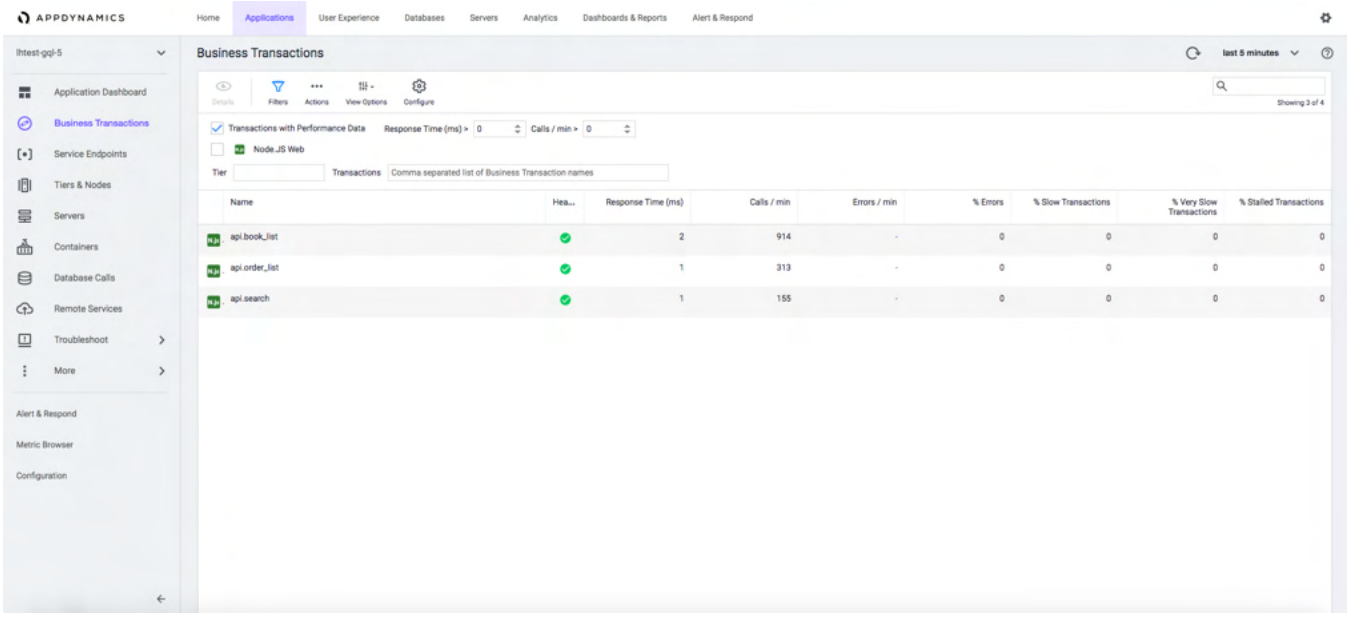
Once GraphQL Business Transactions is enabled, you can configure a GraphQL custom match rule for a Node.js application:

1. In the Controller, select **Configuration** > **Instrumentation** > **Transaction Detection**.
2. Click **Add Rule** > **Summary** > **Custom Match Rule**.
3. Select Node.js in the **Agent Type** dropdown. Click **Next**.
4. Enter a name for the match rule and ensure that the enabled checkbox is checked.
5. Click **Select Scope** and select the desired scope for the rule. Click **OK**.
6. Next, navigate to **Rule Configuration** > **HTTP Request Match Criteria** and click **Add**.
7. Select URI as the criteria and choose the desired operation from the dropdown list. Then, enter the desired value for the GraphQL incantation (for example, the URL prefix).
8. Navigate to **Split Transactions Using Request Data** and click the **Split Transactions Using Request Data** checkbox.

9. Select **the graphql operation name** from the dropdown and select **Save**.



Upon configuring the rule and sending a GraphQL queries in your application, the **Business Transactions** page displays separate transactions.



PHP Transaction Detection

Related pages:

- Configuration Import and Export API
- Configure PHP Custom Match and Exclude Rules

This page describes transaction detection and configuration procedures that related specifically to the PHP Agent. see Transaction Detection Rules.

PHP Entry Points

The entry point is where the business transaction begins, typically a URI or a page callback name, MVC controller action, page template name, or page template. The PHP Agent automatically detects and monitors the entry point types listed under PHP Frameworks and Protocols in PHP Supported Environments.

If the agent cannot detect the framework, the entry point type defaults to PHP Web. PHP Web automatically detects all HTTP requests to the application.

Enable Monitoring by Entry Point

You can enable and disable transaction monitoring for each entry point type.

When monitoring is disabled, the agent stops counting, measuring, and recording all activity on servers of that type.

Enable or disable monitoring by checking or clearing the **Transaction Monitoring** checkbox in the **Transaction Detection** panel for the PHP entry point type.

Transaction Naming

By default, PHP Agent names transactions are named after the first two segments in the service request URL. You can modify the default naming scheme in the same manner as for any web-based business transaction entry point.

For PHP-based entry point tiers that use a multi-layered framework architecture, the last framework that processes the transaction determines the name of the PHP business transaction. For example, for an incoming request processing path of PHP Web > MVC > Drupal 8, a new business transaction would be named based on the Drupal 8 invocation rather than the preceding layers in the request processing flow.

Virtual Host Naming by Entry Point

If you have multiple virtual hosts configured on a single web server, you can differentiate among the business transactions by adding the virtual host name as a prefix to the default business transaction name. Enable use of the prefix by checking the **Use Virtual Host in Business Transaction names** checkbox in the **Transaction Detection** panel for the PHP entry point type. Transactions discovered and named subsequent to this configuration will use the virtual host name as a prefix to the default name.

For example, an application has two exit.php actions running on the same physical host. One action runs on a virtual host named `phpagent1` and the other runs on a virtual host named `phpagent2`. Transaction naming will occur as follows:

- virtual host name is not used: agent identifies all requests to exit.php as a single business transaction named `exit.php`
- virtual host name is used: agent identifies requests as two different business transactions named `phpagent1 : exit.php` and `phpagent2 : exit.php`

Customize Transaction Detection and Naming

If the default contentions do not produce transactions to fit your needs, you can change the transaction configuration as follows:

- Create exclude rules to eliminate entry points that you do not need to monitor.
- Create custom match rules that specify matching rules for identifying and naming transactions.
- For PHP Web entry points, you can also configure transaction naming.

To access the transaction detection panes, select **Configuration > Instrumentation > Transaction Detection > PHP-Transaction Detection**.

Configure PHP Custom Match and Exclude Rules

Related pages:

- Transaction Detection Rules
- Using Regular Expressions

This page summarizes the criteria that you can use for creating custom match and exclude rules for each entry point type.

The criteria are the same for custom match and exclude rules.

If you disable automatic detection for an entry point type and then apply custom match rules, the agent discovers only requests defined by those match rules. For example, if you have a PHP MVC custom match rule that matches on “Action contains search”, and another rule that matches on "Action contains view", requests that contain "search" or "view" are the only MVC transactions that the agent detects.

On the other hand, if you do not disable automatic detection and you apply a custom match rule “Action contains search”, all the requests that match that criterion are grouped together as a single "search" transaction and then any other requests (those that do not contain “search”) are discovered using automatic detection.

Accessing Custom Rule Configuration

1. Click **Configuration Instrumentation > Transaction Detection > PHP-Transaction Detection**.
2. Scroll down to either the **Custom Match Rule** section or the **Exclude Rule** section.
3. Click **+** to create a new rule, then select from the dropdown the entry point type to which the rule applies.

PHP Web

Matching criteria can include one or more of the following:

- HTTP method
- URI segments
- HTTP parameter (value or existence)
- HTTP header (value or existence)
- Port
- Cookie (value or existence)

See Set Priority When Multiple Rules Apply in Custom Match Rules.

Transaction Splitting

For PHP Web, if a transaction is configured to identify the entry point based on the URI, you can optionally split the transaction into multiple transactions. For example, a login request may be automatically detected as a single transaction, but you want to split it into two transactions based on whether the request branches to a new-user or existing-user operation.

You can split a PHP Web transaction using request data from the transaction names such as URI segments, parameter values, header values, and cookie values.

PHP Web Splitting Example

For example, to split a custom match rule transaction "mobile checkout" by the mobile carrier parameter, so the agent detects separate transactions as follows:

- products/mobile/checkout?carrier=verizon
- products/mobile/checkout?carrier=sprint
- products/mobile/checkout?carrier=att

To split transactions based upon the carrier parameter:

1. On the Rule Configuration tab, set a URI match option.
2. Check **Split Transactions Using RequestData**.
3. Select **Use a parameter value in Transaction names**.
4. Enter "carrier" for the **Parameter Name**.

The agent creates the following transactions: "mobile checkout.verizon", "mobile checkout.sprint", and "mobile checkout.att".

PHP Web Custom Match Rules vs Auto-Detection Rules of Other Types

If a custom match rule configured for PHP Web matches the same criteria as an auto-detection rule for a business transaction of a different transaction type in the same application, the PHP Web custom match rule takes precedence and the business transaction appears as a PHP Web transaction type.

However, if a custom match rule for PHP Web and a custom match rule for another transaction type match the same criteria, precedence depends on the values of the rules' priorities. See Setting Match Rule Priority for information on rule priorities.

PHP MVC

By default, for most MVC frameworks, the business transaction is named using the controller:action. For modular MVC frameworks, the business transaction is named using the module:controller:action.

You can modify detection to match only a portion of the module, controller or view name (such as "Begins with" or "Contains") or on a regular expression, rather than the default which matches the entire name using "Equals" for the match condition.

You cannot configure a single match rule that matches on both the controller and the action. You can create MVC custom match rules using any of the following:

- controller name only
- action name only
- module name and action
- module and controller name

PHP Web Service

By default, for Web Services frameworks, the business transaction is named for the entire web service name and the entire operation name using "Equals" for the match condition.

You can modify detection with a custom rule that matches on only the web service name or only the operation name. You can match on a portion of either name or use a regular expression.

PHP Drupal

By default, AppDynamics automatically names Drupal transactions based on the entire page callback name of the Drupal module using "Equals" for the match condition.

You can modify detection with a custom match rule that uses just a portion of the page callback name or use a regular expression.

PHP Wordpress

By default, AppDynamics automatically names Wordpress transactions based on the entire Wordpress template name using "Equals" for the match condition.

You can modify detection with a custom match rule that uses just a portion of the template name, or use a regular expression.

PHP CLI

By default, AppDynamics automatically names PHP CLI transactions based on the last two segments of the script's directory path plus the name of the script.

The agent creates a business transaction instance every time the script is run.

You can modify detection with a custom match rule that matches on a different portion of the script path and name or uses a regular expression.

Custom Exclude Rule Examples

Custom exclude rules prevent detection of business transactions that match certain criteria. You might want to use exclude rules in the following situations:

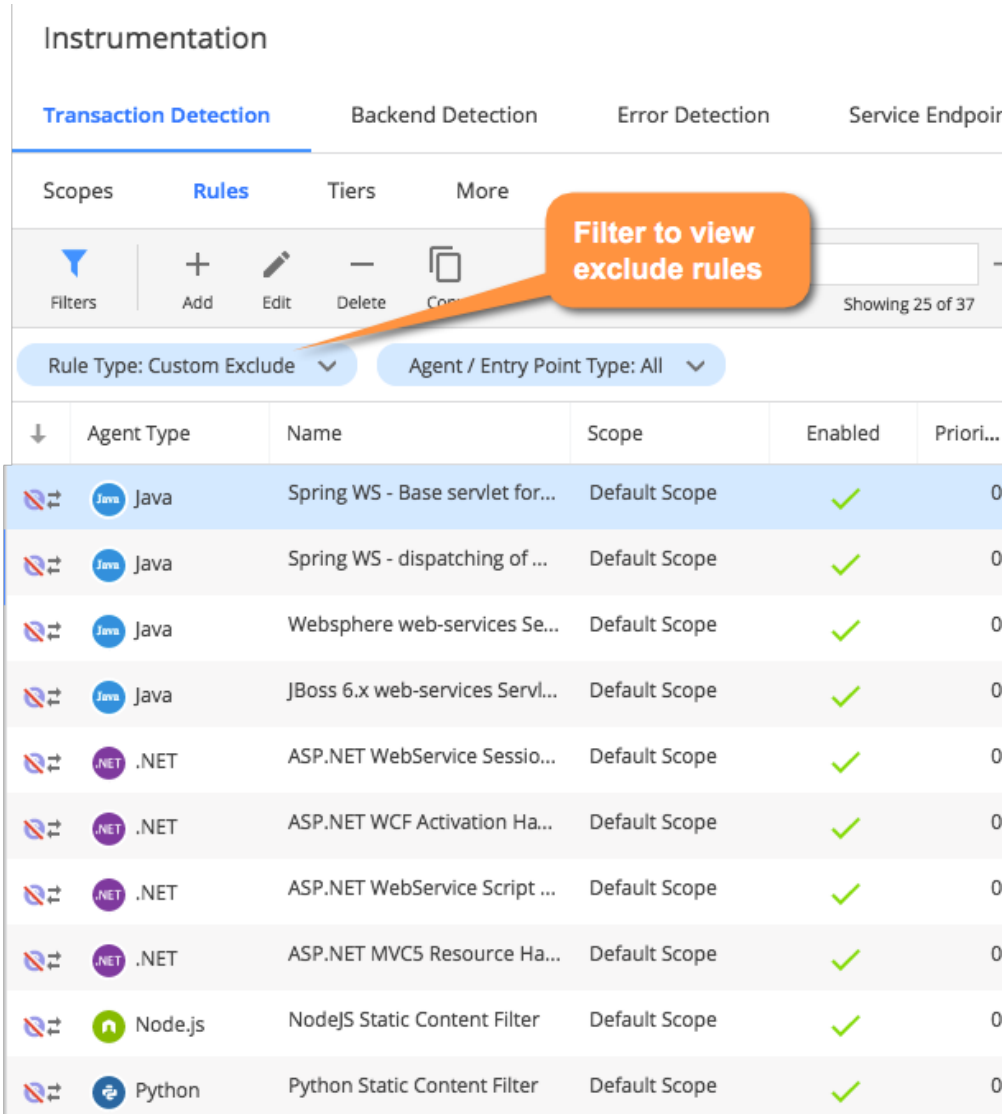
- AppDynamics detects business transactions that you do not want to monitor.
- To stay within agent and controller limits, you want to reduce the total number of business transactions.
- You need to suppress a default entry point to allow for the detection of a more precise entry point defined by a custom match rule.

You can customize existing rules or set up new rules. See Custom Match Rules.

Change the Default Exclude Rule Settings

AppDynamics has a default set of custom exclude rules for various agent types. Sometimes you need to disable a default rule or add a new custom exclude rule. Different entry point types include different match criteria that you can use to exclude transactions.

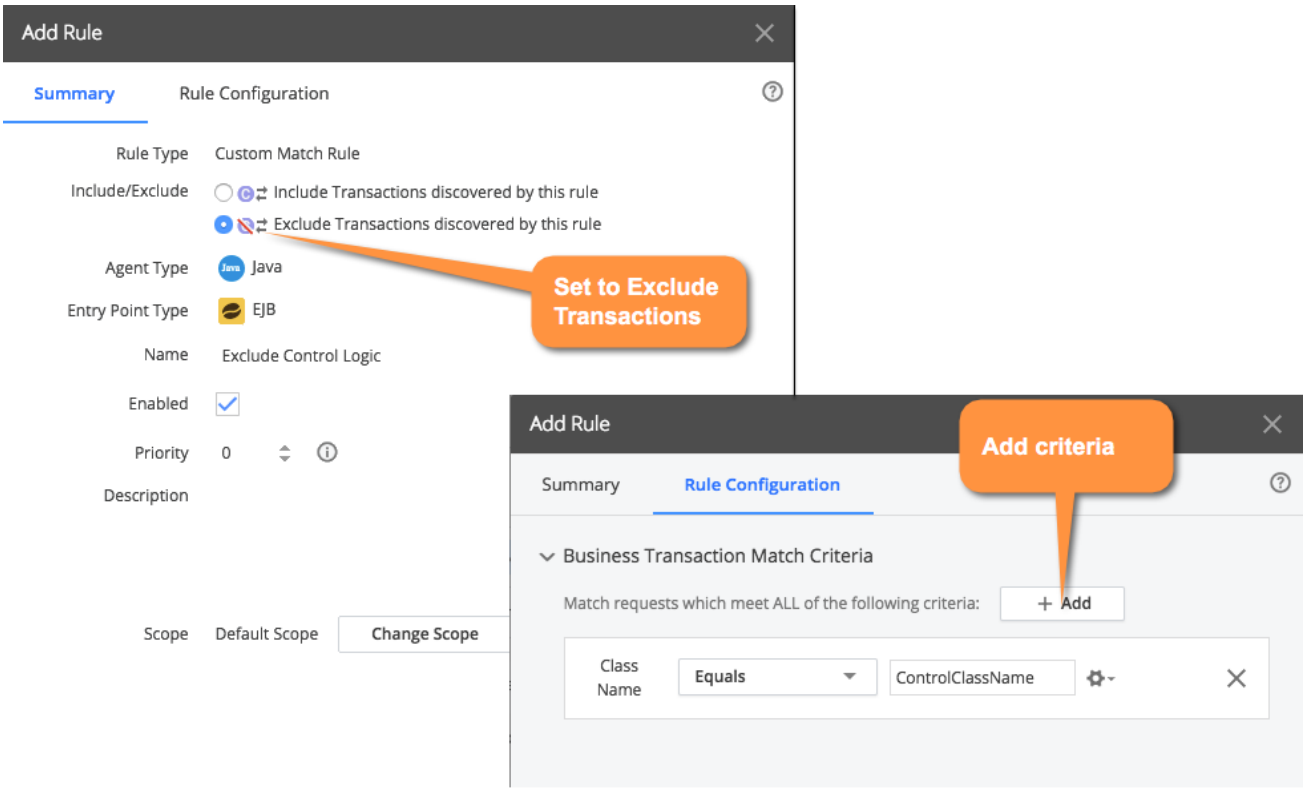
You can view the default rules on the **Transaction Detection** > **Rules** tab:



Define Transaction Names Using Downstream Application Logic

Sometimes an incoming request invokes control logic in your code that uses payload data to execute different business logic. In such cases, it makes sense to name your business transactions for the business logic class and method. You can use a custom exclude rule to prevent AppDynamics from naming the business transaction for control logic. Then you can identify the transaction by the business logic instead.

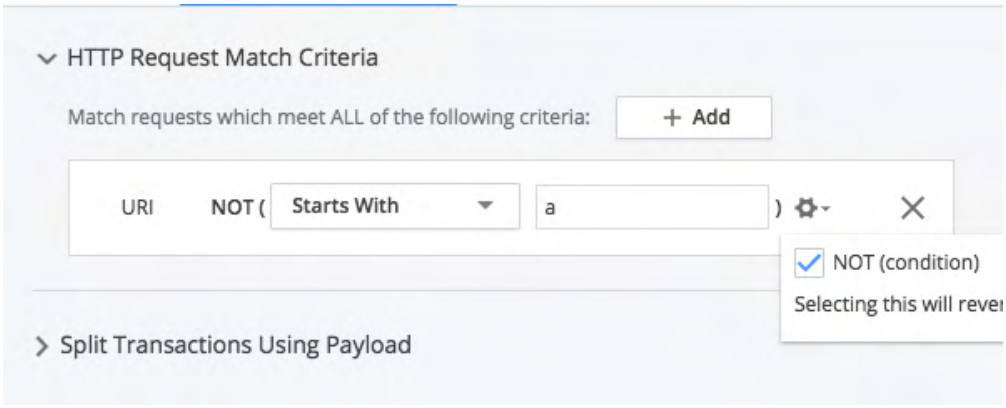
Consider an example where you have implemented both the control logic and business logic as EJBs. With **EJB** discovery enabled, AppDynamics discovers and names business transactions based on the control class and method. You can create an EJB exclude rule for the control class name using the **Class Name Equals** match criteria. After you create the rule AppDynamics discovers and names the business transactions for your business logic APIs:



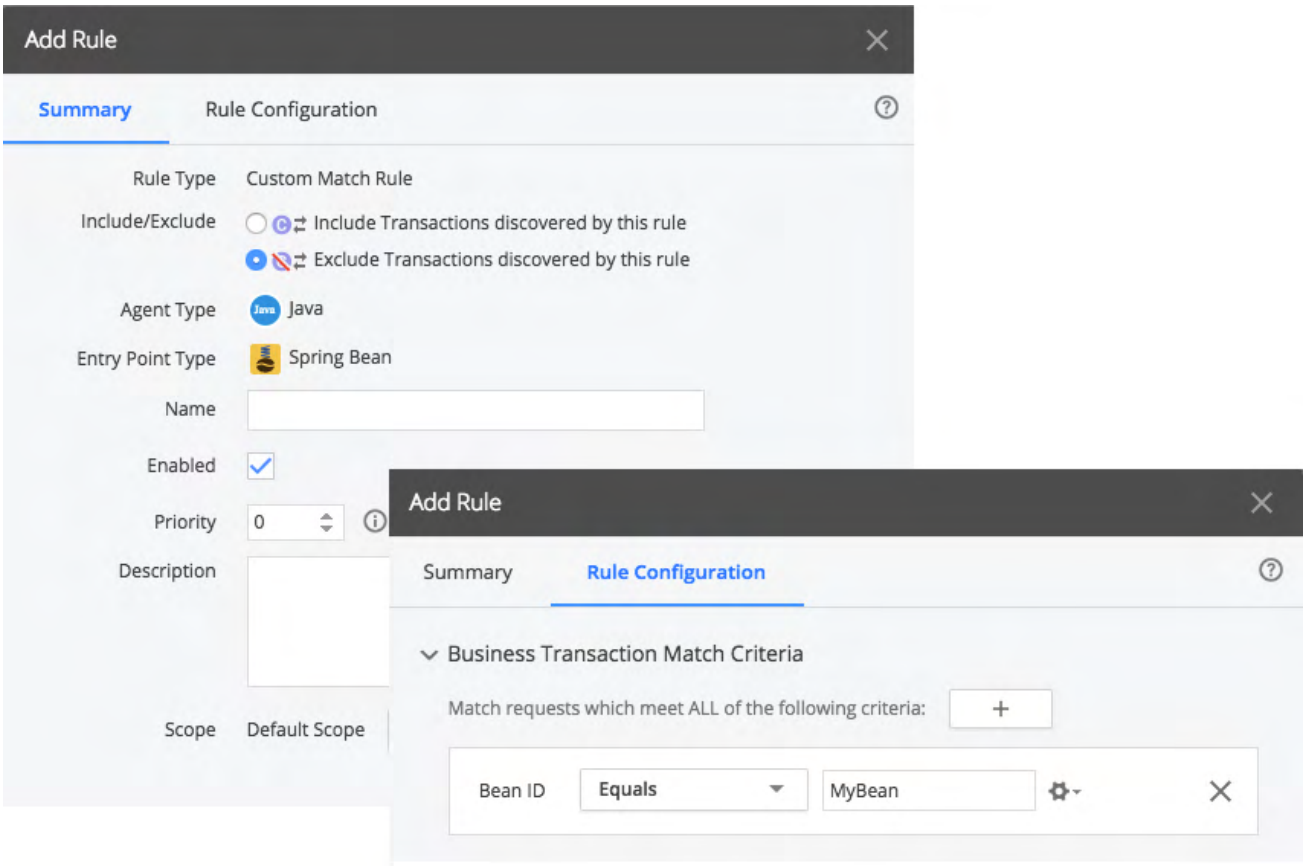
Use an Exclude Rule as a Filter

Custom exclude rules filter out unwanted requests from transaction discovery. You can specify match criteria that allow eligible requests but ignore everything else.

For example, you want to use default discovery rules. Your application receives URI ranges that start with : /a, /b ... /z. However, you only want to monitor URIs only when they start with /a . You can create a custom exclude rule that does a "Match : Doesn't Start With /a" as shown here:



In another example, imagine a specific an application that implements Spring Beans of the classes `java.lang.String` and `java.util.ArrayList`. This means that AppDynamics identifies all instances of these classes as Spring Beans with the same IDs. To fix the problem, you can define a custom exclude rule specific Spring Bean IDs:


Add Rule
Summary
Rule Configuration
Rule Type
Custom Match Rule
Include/Exclude
Include Transactions discovered by this rule
Exclude Transactions discovered by this rule
Agent Type
Java
Entry Point Type
Spring Bean
Name
Enabled
Priority
0
Description
Scope
Default Scope
Add Rule
Summary
Rule Configuration
Business Transaction Match Criteria
Match requests which meet ALL of the following criteria:
Bean ID
Equals
MyBean

Custom Match Rule Live Preview

This page provides instructions and information for the custom transaction detection rule Live Preview.

You must have Configure Transaction Detection and View Sensitive Data permissions to use Live Preview. See Create and Manage Custom Roles.

When you create or edit servlet or POJO custom match rules in AppDynamics, you can click **Live Preview** to inspect pertinent transaction data from a node.

Custom transaction detection rule Live Preview session lets you interactively preview the effect of rule updates on your business transactions. Live Preview also provides tools for the different rule types to let you inspect the live data streaming from your application.

When you launch Live Preview, select the node where you want to preview transactions. Nodes are organized by tier. You can see which nodes have active agents and if there is a live preview session already running on a node.

Preview Business Transactions

When you initiate a custom match rule live preview session, the Controller shows the rule configuration in the left pane. Click **Preview Business Transactions** in the right pane to see resulting transactions. You can edit the rule and click **Apply** to update the changes within the session.

When the agent discovers an entry point for a rule but the transaction payload contains correlation data from an existing transaction, Live Preview displays a masked transaction.

POJO Match Rule Tools

You can use these tools to inspect data and configure the entry point for a POJO custom match rule:

- **Classes/Methods browser**—lets you search for classes.
    - You can right-click a class or method in the result to set the class and method match criteria for the POJO.
    - You can view method invocations for a selected method.

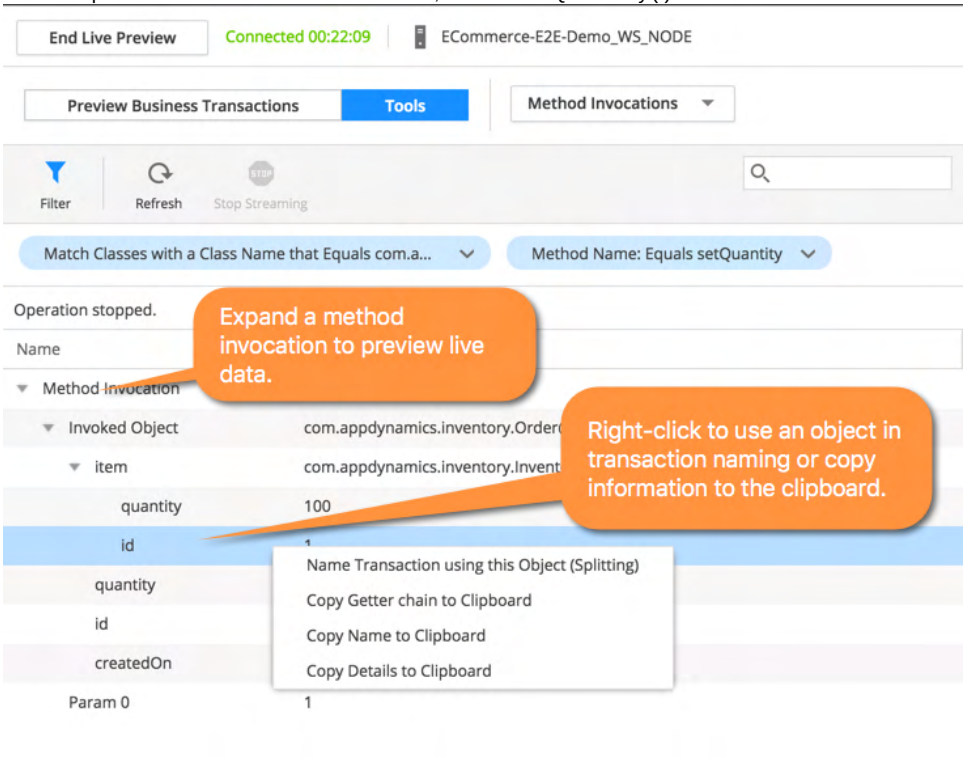
In live preview mode, the number of methods and classes that can be returned by an agent is now limit-bounded. The limits are:

- Maximum number of classes: 1000
- Maximum number of methods: 300

You can change limits by modifying the node properties `classmethodservice.max.no.of.classes.reported` and `classmethodservice.max.no.of.methods.reported`, respectively

- **Method Invocations**—lets you inspect live data from the server for the specified method.
    1. Expand a method invocation to see detailed information including invoked objects, parameters, and return values.
    2. Right-click on a data point to use it in transaction splitting.
       AppDynamics automatically creates a splitting rule following the `get<field>()` syntax.
       If your code follows another convention, you need to manually specify the getter method for the data point.

3. You can also copy a getter chain to the clipboard.
   The example below shows the data for a method, `Order.setQuantity()`:



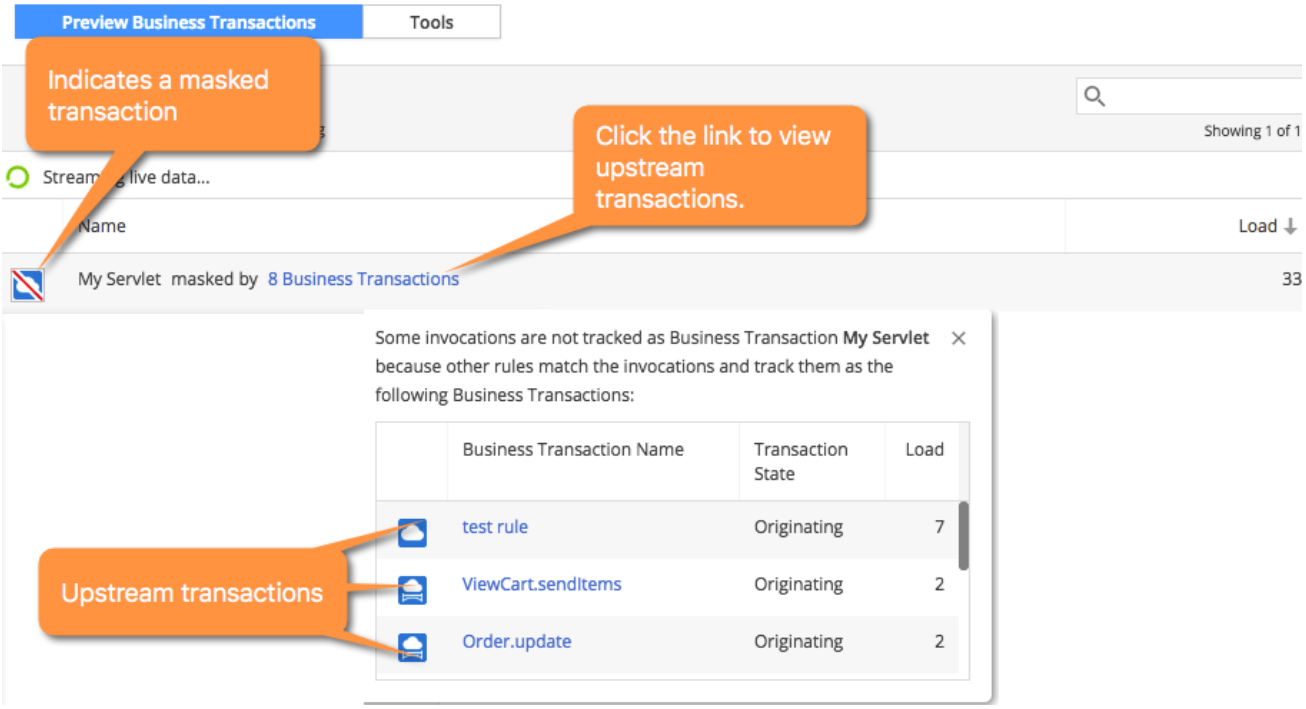
Servlet Match Rule Tools

When you use Live Preview to inspect data for a servlet custom match rule, AppDynamics shows live HTTP Requests that meet the rule criteria. You can click an individual Request URL to view detailed request data: headers, cookies, and parameters.

1. Right-click an individual data point, such as a header, to add the value to the match criteria or to set it as a transaction splitting option.
2. You can group the incoming HTTP requests by various parts of the request and its payload, such as the URI, an HTTP parameter, a header, etc. You can right-click a result from the **Values** column and add the value to the **HTTP Request Match** criteria.

Masked Transactions

Live Preview flags transactions that are downstream from another existing business transaction as masked transactions. To view the originating business transactions masking the current transaction, click the link indicating the number of transactions.



Click a business transaction name to view the business transaction dashboard for the masking transaction.

If you want to view metrics for the current entry point outside of a masking transaction or transactions, you can either:

- Deactivate the business transaction detection rules that created the upstream transactions, or
- Create a service endpoint for the entry point.

Otherwise, you can continue to track the performance of the masked entry point as a segment of the business transaction it currently belongs to.

URI Based Entry Points

For almost every web-based framework or platform, there is a URI based business transaction entry point for user requests. AppDynamics App Agents automatically discover the following URI-based entry points:

- Java Agent: Servlet entry points
- .NET Agent: ASP.NET entry points
- Node.js Agent: Node.js web entry points
- Python Agent: Python web entry points

The Web server Agent can detect Web request entry points, but automatic discovery is turned off by default so an agent on a downstream node can name the transaction.

URI based entry points all have similar configuration options for Automatic Transaction Discovery and Custom Match rules.

For additional functionality that applies to servlets, see Servlet Entry Points.

Default Naming

By default, when an app agent detects a URI request, it names the business transactions using the first two segments of the URI. For example, consider the following URI for a checkout operation in an online store: http://acmeonline.com/store/checkout.

The agent names the business transaction "/store/checkout" and assigns all requests that match that URI to the "store/checkout" business transaction.

If the first two segments of the URI do not contain enough information to effectively identify the business transaction in your environment, you can edit the a utomatic discovery rule to name the transaction based upon the URI and its context. Alternatively, you can create a custom match rule to specify a name.

For ASP.NET Core on the full framework, the default naming convention uses the Controller, action, and area. See Name MVC Transactions by Area, Controller, and Action for more information and configuration options.

Customize Automatic Naming Rules

When you edit or create a new Default Automatic Discovery Rule for URI based entry points, you can adjust naming to:

- Use the full URI to name transactions.
- Use part of the URI to dynamically name transactions.

Expand the **Configure Naming** option on the **Rule Configuration** tab to display the configuration options.

When you configure the rule to **Use the full URI**, the agent creates a business transaction name for the full request URI. There are no dynamic naming options. For example, for the URI http://acmeonline.com/store/coats/men, the agent names the transaction "`/store/coats/men`".

Use the full URI works well for Grails servlet transactions because the default naming scheme often doesn't differentiate between multiple transactions.

When you use parts of the URI to name transactions, you have several options to configure transaction naming:

- Use a Part of the URI
- Use Headers, Cookies, and Other Parts of HTTP Requests
- Use a Custom Expression

Use a Part of the URI

AppDynamics offers the following options to automatically name URI based transactions based upon the URI:

- Use the first or last URI segments
- Use specific URI segments

For the following URL the first two segments of the URI don't provide a significant name for the business transaction:

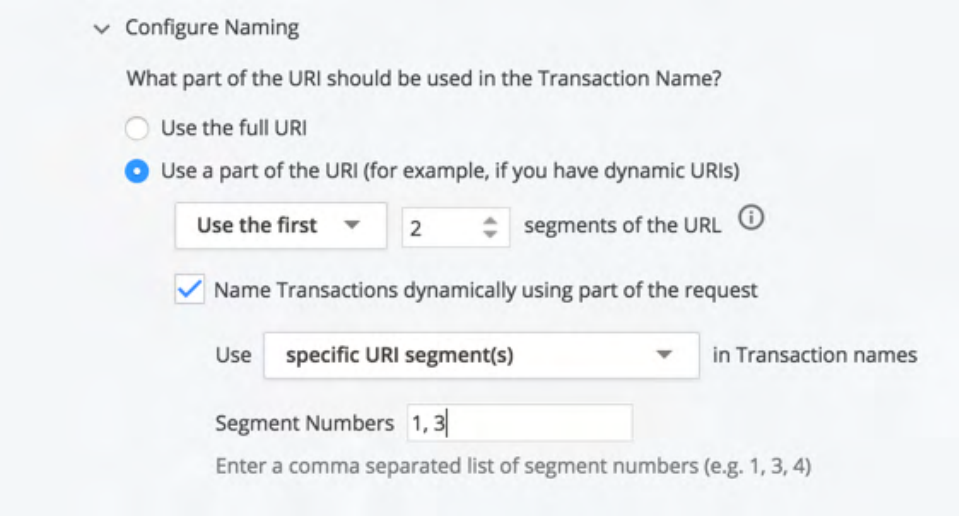
http://example.com/Web/Store/Checkout yields a default transaction name of "Web/Store".

You can use one of the following options to identify more meaningful transaction names:

- Click **Use the first** or **Use the last n** segments to use two contiguous segments at the beginning or end of the URI, where n is the number of segments. For example, if identify the checkout transaction using the last two segments of the URI, the agent names the transaction "/Store /Checkout".

- If you need more flexibility, such as using non-contiguous segments in the name, click **Name Transactions dynamically using part of the request**. Select **Use URI segments in Transaction names** option. To specify multiple segments, enter a comma-separated list of segment numbers.

  For example, the following URL represents the checkout transaction requested by a customer with ID 1234: `http://example.com/Store/cust1234/Checkout`
  The checkout transaction is the same regardless of the customer, so it makes sense to name the transaction based upon the first and third segments of the URI. AppDynamics names the transaction: "StoreCheckout".

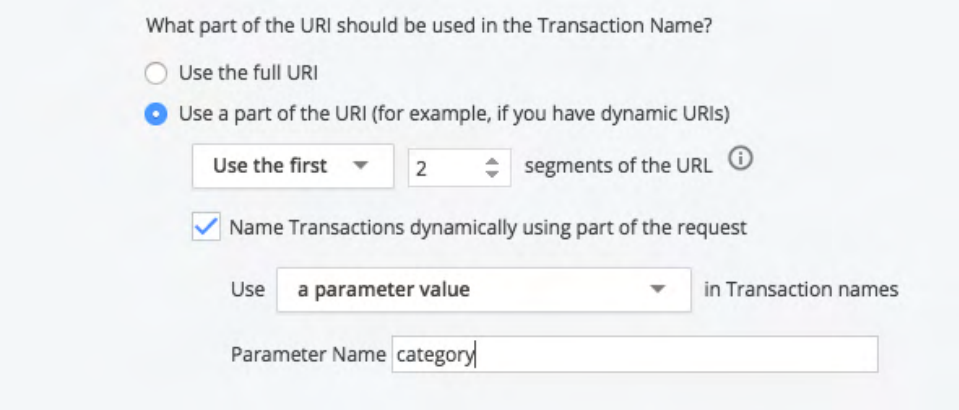


Use Headers, Cookies, and Other Parts of HTTP Requests

When you use the **Name Transactions dynamically using part of the request** option, you can choose to identify your URI based transactions using particular parts of the HTTP request. Note that not all agents offer all the naming options below.

> Carefully consider your naming configuration choices. If you use a value such as the request originating address and you have many clients accessing your application, you may see the all other traffic business transaction.

- To use HTTP parameter values in transaction names, select **Use a parameter value in Transaction names** and enter the **Parameter Name**.
  For example, consider the following URL: `http://example.com/Store/Inventory?category=electronics`



  AppDynamics names the transaction to include the category parameter value "/Store/Inventory.electronics".

- To use a header value in transaction names, select **Use header value in Transaction names** and enter a **Header Name.**
  For example, consider a site that uses the custom header "Version", AppDynamics names transactions with the header value "/Store/Inventory.v2.5".
- To use a cookie value in transaction names, select **Use a cookie value in Transaction names** and enter the **Cookie Name**.
  For example, a website tracks a user's loyalty status in a cookie. Set the Cookie Name to "loyalty". AppDynamics names transactions for the loyalty cookie value "/Store/Inventory.Status=Gold"
- To Use a session attribute value in transaction names, Click **Use a session attribute in Transaction names** and enter the **Session Attribute Key**.
  For example, a website stores a customer's region in the session property. Set the Session Attribute name to "region". AppDynamics names transactions for the region session attribute value "/Store/Inventory.NorthAmerica".
- To use the request method in Transaction names, click **Use the request method (GET/POST/PUT) in Transaction names**.
  For example AppDynamics names the transaction "/Store/Inventory.GET".
- To use the request host in Transaction names, click **Use the request host in Transaction names**.
  For example AppDynamics includes the request host id in the transaction name "/Store/Inventory.192.0.2.0".

- To use the request originating address in Transaction names, click **Use the request originating address in Transaction names**. AppDynamics names transactions for the IP address of the request client "/Store/Inventory.192.0.2.10".

Use a Custom Expression

You can use a custom expression on the URI to name:

- Java servlet transactions
- .NET ASP.NET transactions

Select **Name Transactions dynamically using part of the request** and choose **Use a custom expression in Transaction names**. Enter your custom expression getter chain as follows:

For Java, you can use getter chains to access values from the `HTTPServletRequest` object. For example, suppose you want to use a combination of the authenticated user and the value of an `HttpServletRequest`:

```
${getParameter(myParam)}-${getUserPrincipal().getName()}
```

The equivalent code expression would evaluate to:

```
request.getParameter("myParam")+"-"+request.getUserPrincipal()
```

You can create a custom expression on the `HTTPServletRequest` to identify all Servlet based requests (modify global discovery at the transaction naming level) or for a specific set of requests (custom rule).

A custom expression for a servlet transaction can have a combination of any of the following getter methods on request attributes:

Getters on Request Attributes	Transaction Identification
getAuthType()	Use this option to monitor secure (or insecure) communications.
getContextPath()	Identify the user requests based on the portion of the URI.
getHeader()	Identify the requests based on request headers.
getMethod()	Identify user requests invoked by a particular method.
getPathInfo()	Identify user requests based on the extra path information associated with the URL (sent by the client when the request was made).
getQueryString()	Identify the requests based on the query string contained in the request URL after the path.
getRemoteUser()	Identify the user requests based on the login of the user making this request.
getRequestedSessionId()	Identify user requests based on the session id specified by the client.
getUserPrincipal()	Identify user requests based on the current authenticated user.

For .NET, you can use getter chains to access properties and methods for the `HttpRequest` object for ASP.NET or `Microsoft.AspNetCore.Http.Internal.DefaultHttpRequest` for ASP.NET Core on the full framework.

- Enclose getter chains inside braces: `${}` . The custom expression may contain multiple getter chains.
- Use getter chain syntax.
- Use any `HttpRequest` request attributes or methods.

For example, consider the URL http://mystore.example.com/Store/Inventory-Furniture .The following custom expression uses two getter chains to access properties of the `HTTPRequest` object:

```
${Url.ToString().Split(Char[]/-).[2]}-${UserAgent}
```

- The first getter chain fetches the URL, splits it on the dash character ("-"), and uses the second string in the array.
- The second getter chain fetches the `HttpRequest.UserAgent` property.
- The literal dash character "-" separates the two getter chains.

The result is the following business transaction name:

Furniture-Mozilla/5.0 (Windows NT 6.3; Trident/7.0; rv:11.0) like Gecko

Message Queue Entry Points

Most frequently an instrumented application calls to a message queue as a remote service or backend. However, in cases when an application uses asynchronous message listeners or message-driven beans as the primary trigger for business processing on an originating tier, AppDynamics can intercept the message listener invocations and track them as business transactions.

AppDynamics discovers the following types of business transaction entry points:

- Java JMS entry points.
- .NET Message Queue entry points

Message queue entry points all have similar naming configuration options for Automatic Transaction Discovery and Custom Match Rules.

Default Naming

By default, the AppDynamics auto-detection naming scheme identifies all message queue transactions using the destination name. Otherwise, if the destination name is not available, AppDynamics uses the listener class name.

Custom Match Rule Options

When you create a JMS or Message Queue custom match rule, the agent names matching transactions for the custom match rule name. See Custom Match Rules.

You have the option to specify the following match criteria as a combination of the following entry point properties:

- The **Message Destination** as **Queue, Topic** or **Unknown**.
- **Message Property**. You can match against a property's existence or against a property's value.
- **Message Content**.

Business Transaction Discovery Sessions

Business Transaction Discovery Sessions provide interactive tools to help you create the optimal set of transaction detection rules and discover crucial business transactions.

During a discovery session, you can preview the effects of all the transaction detection rules on a node. This way you can create rules that capture the precise business transactions that impact your business and avoid rules that generate too many transactions.

Within the context of the discovery session, you can:

- Edit Java Agent transaction detection rules and Preview Business Transactions. (Java only)
- Identify entry points from Uninstrumented Code.
- Browse Classes/Methods to identify entry points.

During the business transaction discovery session, you can test out various transaction detection rule configurations without impacting your production environment.

Any business transactions discovered within the context of the discovery session, called transient transactions, do not persist after the session ends. When you end the session you have the opportunity to discard the rule changes or to apply them to your production configuration.

Permissions

You need the following permissions to launch a Business Transaction Discovery Session:

- Configure Transaction Detection
- View Sensitive Data

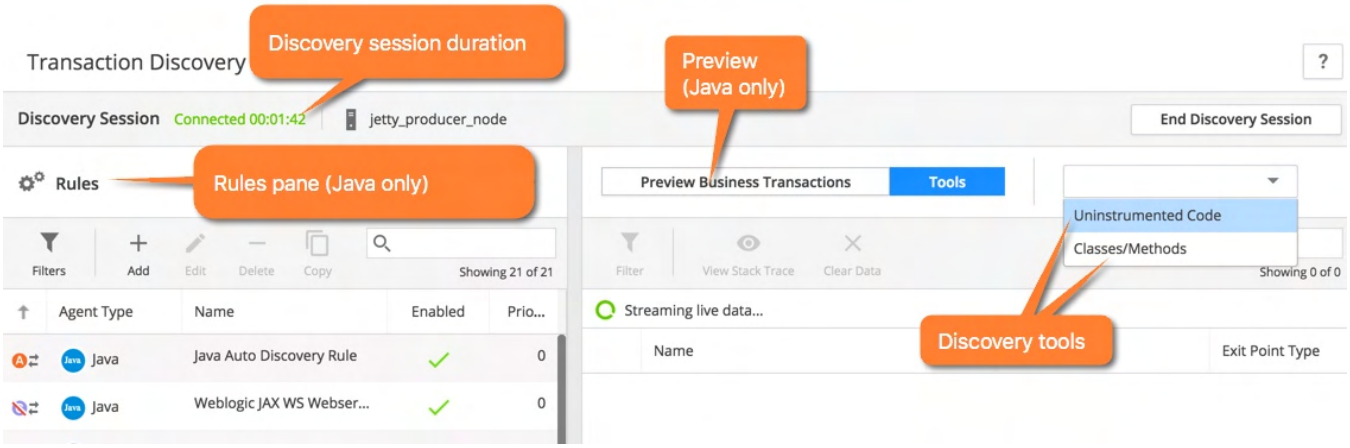
For information on AppDynamics Role Based Access Control, see Roles and Permissions.

Launch a Business Transaction Discovery Session

1. Navigate to **Configuration > Instrumentation** and click the Transaction Detection tab.
2. Click **Live Preview**.
3. Click **Start Discovery Session** to start a Business Transaction Discovery session.
4. Select the node to act as the source of the live data stream.

   Nodes are organized by tier. You can see which nodes have active agents and if there is a discovery session already running on a node.

The Transaction Discovery page displays information about the discovery session lets you navigate among the available interactive transaction discovery tools:



The transaction discovery tool works as it did in previous versions for upgraded business applications. See Business Transaction Discovery Sessions.

When you initiate a transaction discovery session on a Java node, the Rules pane displays the active transaction discovery rules that apply to the node. The right-hand pane lets you **Preview Business Transactions** or use the available interactive **Tools:** Uninstrumented Code detection and the Classes /Methods Browser.

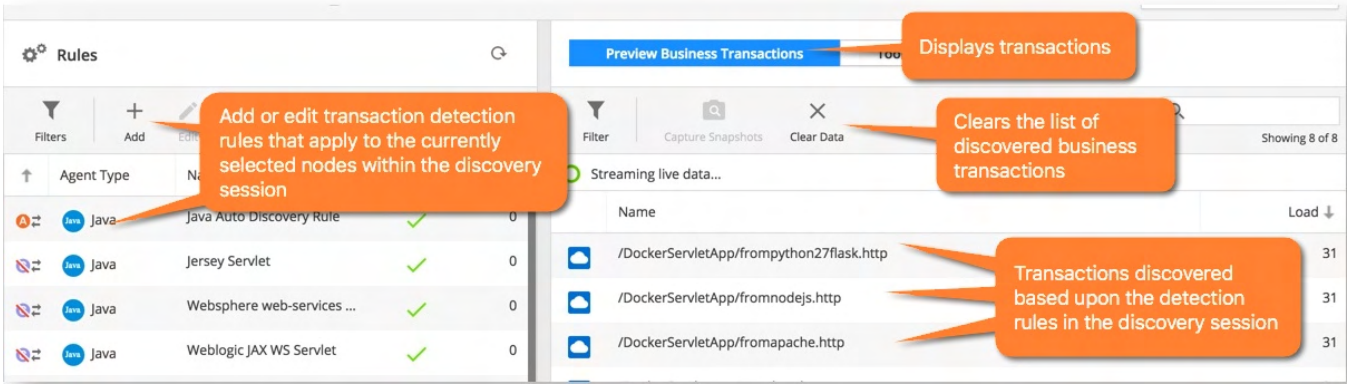
When you launch a discovery session on a .NET node, you see the available tools: Uninstrumented Code detection and the Classes/Methods Browser.

During a discovery session, you can use other features of AppDynamics while the discovery process continues in the background. This is useful for busy environments with many active transactions. You can give AppDynamics a few minutes to complete the discovery. When you are ready to return to the Transaction Discovery page click **Configuration > Instrumentation** and click the **Live Preview** button to resume your discovery session.

Edit and Preview Transaction Discovery Rules

You can add, edit, or delete any type of transaction detection rule for the Java Agent in the Rules pane. Click **Apply** to update the rule within the context of the discovery session.

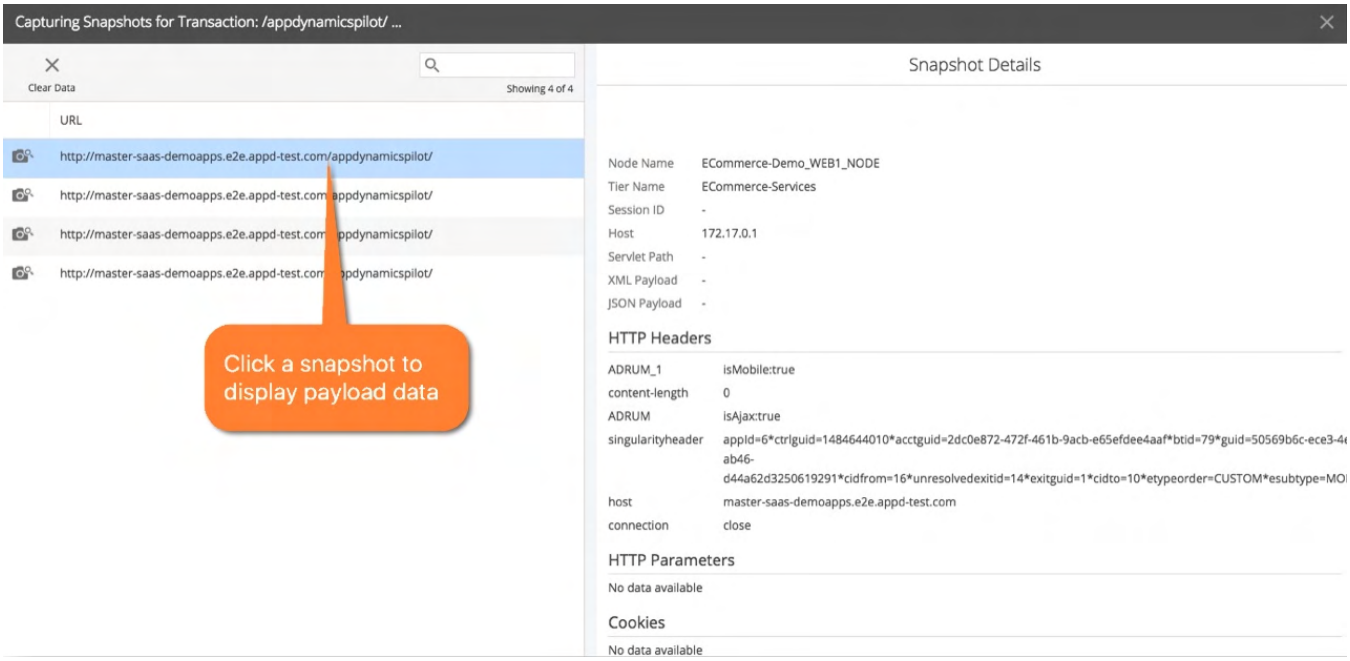
To see the effects of your changes, click **Preview Business Transactions**. As the app agent discovers business transactions based on the rule configuration, the transactions show up in a list in the right-hand pane:



You can select a transaction and click **Capture Snapshots** to collect transaction snapshots.

> The Capture Snapshots feature is only available for Servlets.

Click a specific snapshot to display the payload data for the business transaction in the Snapshot Details:

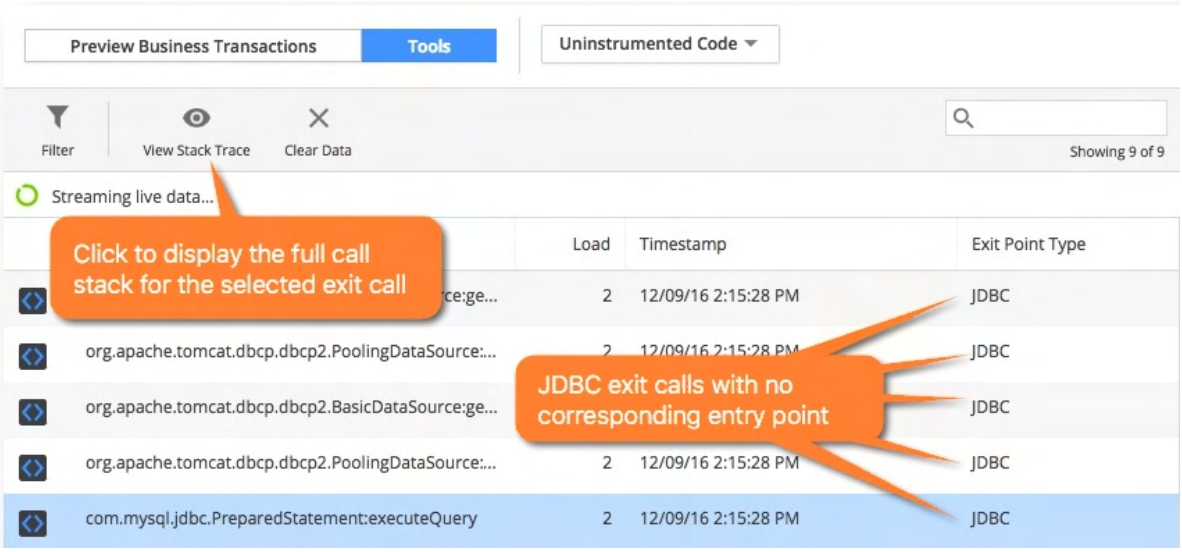


When you create custom match rules for servlets or for POJOs you can start a Live Preview session limited to the specific rule.

Inspect Uninstrumented Code to Discover Entry Points

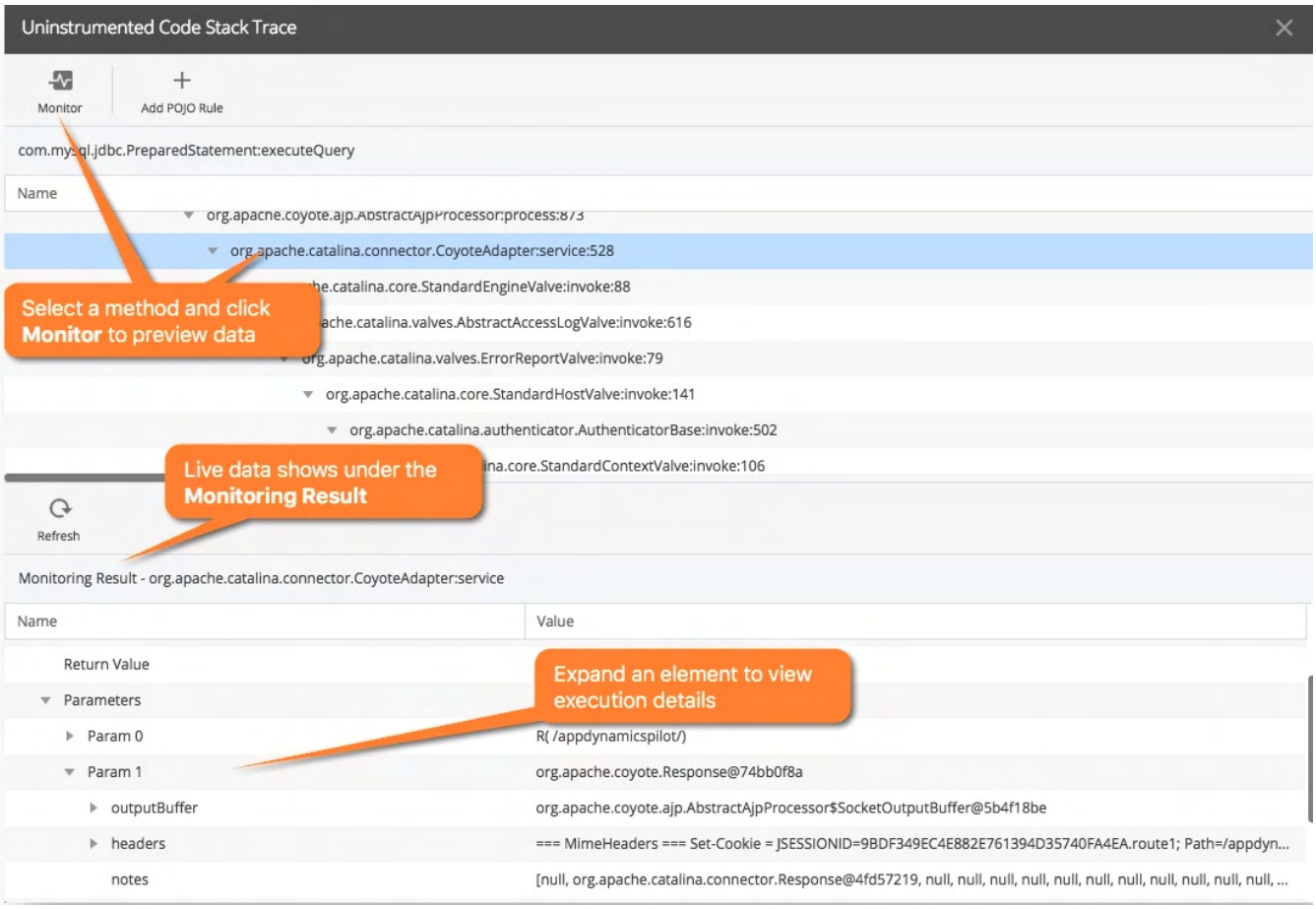
When the app agent doesn't automatically discover an entry point for a framework in your application, you can use Uninstrumented Code detection to find entry points for the framework. After you identify an entry point you want to track as a business transaction, you can create a custom match rule.

Uninstrumented Code detection identifies the support exit calls for an app agent on the node when the entry point is undetected. In the following example the Java Agent discovers some uninstrumented JDBC exit points:



You can select an exit point and click **View Stack Trace** to display the call stack for the exit point. Select any call in the stack trace and click **Monitor** to show interactive data for that specific call. When the agent returns the monitoring result, you can inspect the values for a specific method invocation including the **Invoked Object**, **Return Value**, **Parameters**, and more.

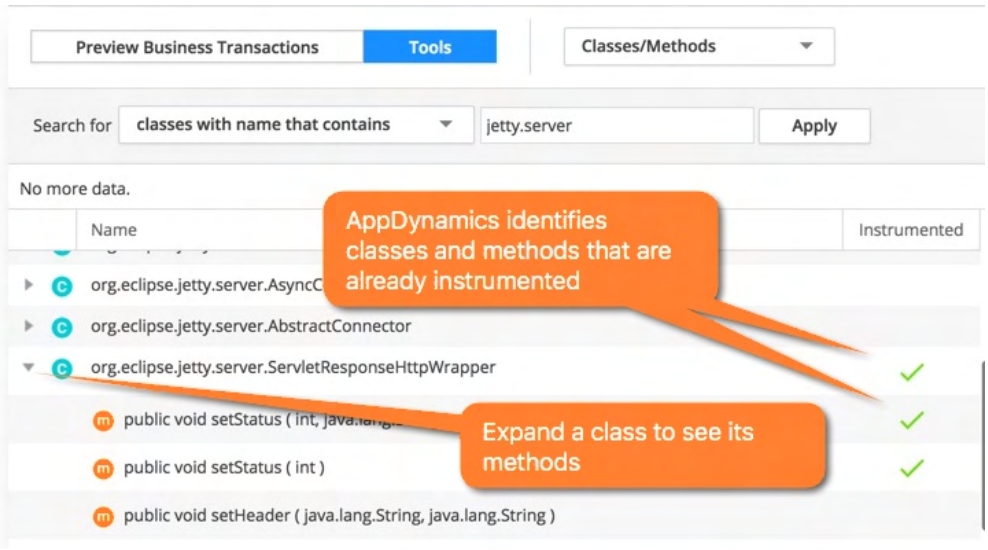
The example below shows the upstream stack trace for the `com.mysql.jdbc.PreparedStatement:executeQuery` JDBC exit call:



When you identify the method you want to instrument, you can click to add a POJO or POCO. The Controller UI automatically populates the Rule Configuration in the Add Rule editor with the class and method you want to instrument. Complete the custom match rule for the POJO or POCO as normal.

Search for Classes and Methods to Instrument

If you are familiar with your application code, you can use the Classes/Methods browser to search the running code for specific classes and methods to instrument. The example below illustrates how a search for `jetty.server` in a sample application returns both instrumented and uninstrumented classes and methods:



When you identify a method you want to instrument, right-click the method to open the Add Rule editor. AppDynamics automatically populates the class and method configuration based upon your search criteria from the Class/Method browser. You can right-click a class to instrument it, but you must add the method before you can save the POJO or POCO. Complete the custom match rule for the POJO or POCO as normal.

Backend Detection Rules

Related pages:

- Java Backend Detection
- .NET Backend Detection

In AppDynamics, databases and remote services (and in fact, any detected out of process components that are involved in Business Transaction processing) are collectively known as backends.

AppDynamics discovers backends from exit point instrumentation placed in the application code. An exit point is the precise location in the code where an outbound call is made from an instrumented node. AppDynamics automatically discovers a wide range of backends.

Default Automatic Backend Discovery

A discoverable backend is identified by its type and associated properties. The precise set of properties used to identify any given backend is dependent on its type. The type itself is defined in the built-in backend discovery rules for backend types supported out of the box. For exit calls to backends that are not instrumented with APM agents, AppDynamics uses the backend properties to identify and name the backend. Where a backend call is actually processed by a downstream tier also instrumented with AppDynamics, ensure that any given backend that is identified will always result in calls that will be processed by just one downstream tier.

This means, for example, that if tier A makes an HTTP call to an HTTP router on localhost:4040 which will forward the call to tier B or tier C depending on the URL, custom backend naming rules will be required to include enough segments of the request URL - as well as the host and port - such that requests destined for tier B will be associated with a backend with a different set of backend properties than those associated with tier C.

The **Automatic Backend Discovery** list in the **Configuration > Instrumentation >** Backend Detection tab shows the configurable backend discovery rules.

The automatic discovery rules vary according to the type of backend being identified, but generally include settings for enabling automatic discovery of the backend type, enabling correlation, and properties used to identify and name the backend. AppDynamics uses transaction correlation to track request processing across distributed tiers. Certain types of backends are automatically discovered as *high volume* exit points (sometimes called turbo exit points). These backend types include cache servers, EhCache, Danga Memcache, Memcached, and Oracle Coherence. For more about high volume exit points, see Exit Point Detection Rules.

When an exit point is identified for the first time, the exit call results in a backend discovery event.

Service Proxy

Sometimes an inaccurate identification of the backend destination causes each call that is sent (based on configuration) to a single backend, to arrive at one of several different downstream tiers.

For example, this situation may occur if the naming of the backend is based on the backend host and port where a particular backend's host:port combination actually identifies an Apache server. The Apache server then forwards requests to different downstream tiers based on the URL acting as a service proxy. When the AppDynamics Controller detects this situation, it will automatically instruct the upstream agent to identify the backend as a service proxy. Once that occurs, a service proxy icon will display on the flow map between the upstream tier and all of the possible target tiers shown downstream. See Service Proxy.

Permissions

To modify backend detection, you need the **Configure Backend Detection** permission.

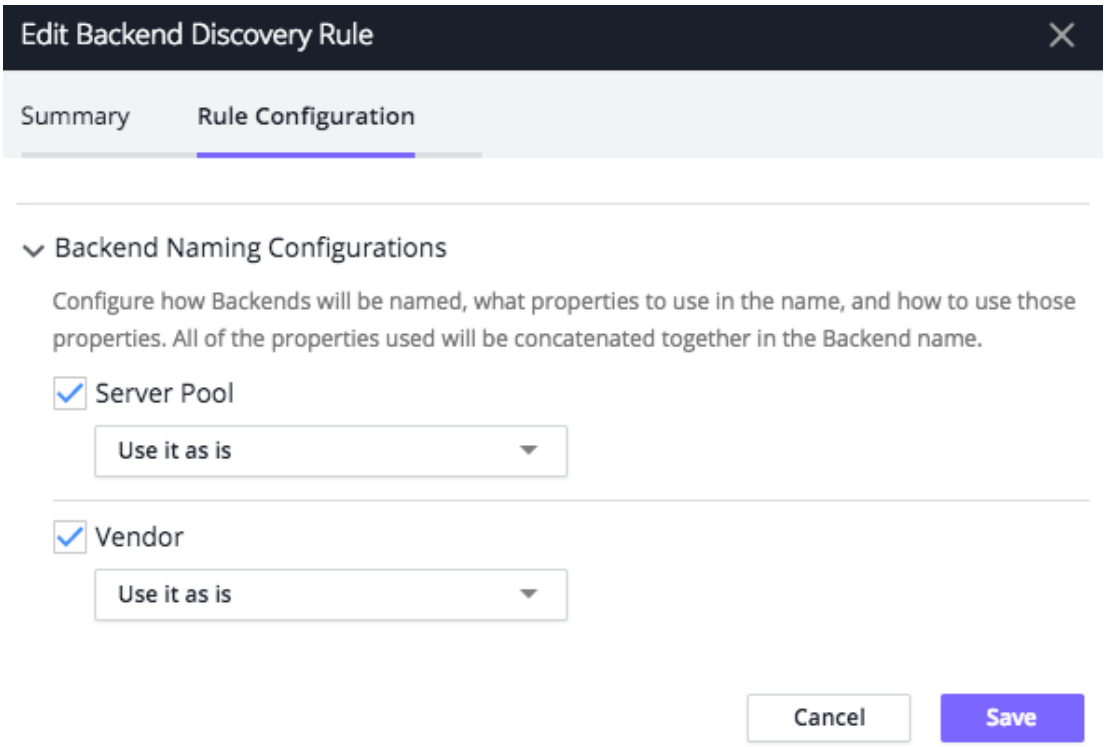
Modify Automatic Discovery

If you know that the application flow map should be showing a specific backend and you are not seeing it, you can modify the configuration to detect backends. You can:

- Edit the default backend discovery rules.
- Create new custom backend discovery rules.
- Create new custom exit points to provide visibility for backend types that are not automatically discovered.
- Disable automatic discovery of backends that are not of interest.
- Disable correlation for the backend.

In many cases, you can achieve the level of customization that you need by editing the default rules rather than creating new rules.

To edit a rule, in the Backend Detection tab, select the application and click **Edit Automatic Discovery**.



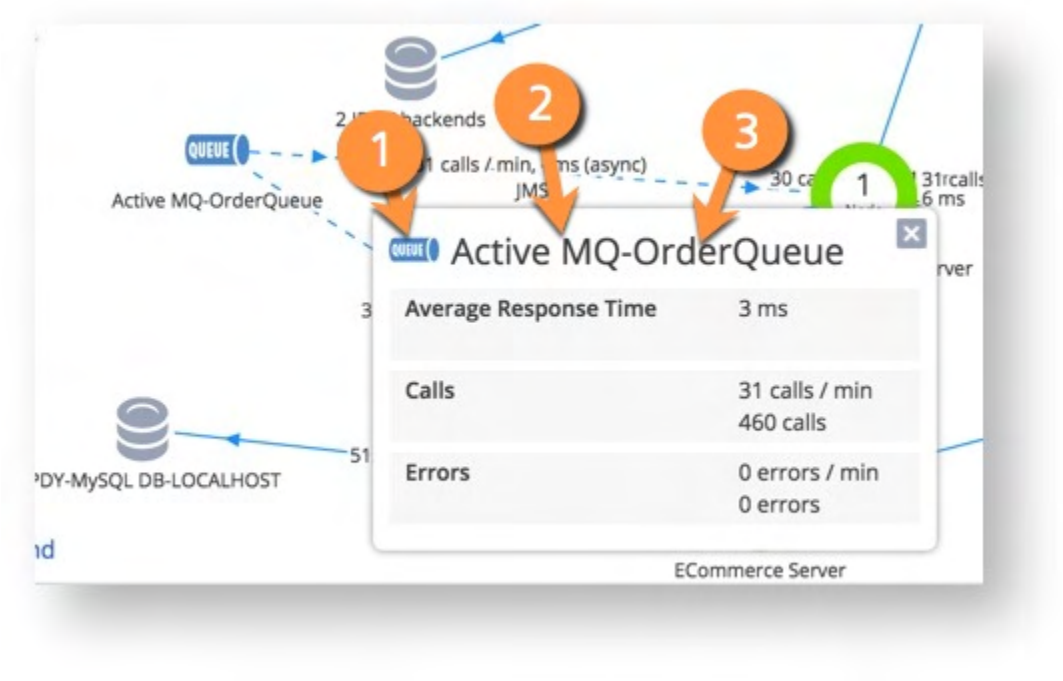
For certain backend types—particularly those named by URL—the name of the discovered backend instances include a port number. If the app agent is unable to discover the port number for some reason, the port number is displayed in the UI as "-1". This can happen, for example, when a port is not a public port or the API for discovering the port is not available on the application environment (often the case for an RMI backend type). Despite having an unknown port number, the backends are nonetheless uniquely identified.

Configurable Properties and Naming

For many of the automatically discovered backend types, AppDynamics uses a set of configurable properties to identify and name the backend. For example, the properties for a JMS messaging queue are something like this:

- DestinationType (for example, QUEUE)
- Destination (for example, OrderQueue)
- Vendor (for example, Active MQ)

The following informational popup shows how the values are used to identify a discovered backend. In the following example, callout 1 shows the value that corresponds to the destination type, callout 2 to the vendor name, and callout 3 to the destination queue name:



Aggregate or Split Backends

You can change the default configuration for the backend types shown in the UI to aggregate or split the backends. For example, you could disable the Vendor property for JMS backend detection to aggregate (join) all JMS backends with the same destination and destination type. The metrics for all JMS backends with the same destination and destination type would be aggregated as one backend.

For properties that you want to use, you can also configure how AppDynamics uses the property. For example, if the URL property is used to identify a backend and your URLs include a file name, the default configuration results in a separate backend for every file.

To reduce the number of backends identified using the URL property, configure which parts of the URL to use. For example, run a regular expression on the URL to discard the file name or any other parts of the URL that you do not want used for identification.

Add Custom Backend Discovery Rules

AppDynamics provides the additional flexibility to create new custom discovery rules for the automatically discovered backend types.

To create a custom discovery rule for an automatically discovered backend type, select the backend type in the **Automatic Backend Discovery** list. Click **Add** (the **+** icon) to create a new rule. Specify the following:

- **Name** for the custom rule.
- **Enabled** to enable or disable the rule.
- **Correlation Enabled** to specify whether or not to send business transaction correlation data to the backend**.**
- **Priority** for this custom rule compared to other custom rules for this backend type. The higher the number, the higher the priority. A value of 0 (zero) indicates that the default rule should be used.
- **Match Conditions** used to identify which backends are subject to the custom naming rules. AppDynamics applies the default rules to backends that do not meet all the defined match conditions.
- **Backend Naming Configuration** used to name the backends matching the match conditions. The naming configuration must include the property used by the match condition.

See specific exit points for examples.

All Other Traffic Backends

AppDynamics limits the number of backend systems that can be registered and tracked. Once the limit is reached (databases and remote services combined), additional backend are put in the "All other traffic for <type>" category.

The "type" is the backend type such as HTTP or Web Service. HTTP calls and JMS queues are the most common types that may reach the limit with the default configuration rules.

On a flow map, the category appears as follows:

Metrics for the backend calls grouped in the "All other traffic" category are aggregated. If the group includes backend instances that you want to be individually tracked, you can manage backend detection, as described in Manage Backend Discovery.

Backend Registration Limits

The limits on the number of backends that can be registered:

- The Controller enforces a limit of 1000 backend registrations in the business application for each type of backend (that is, 1000 HTTP backends, 1000 JMS backends, and so on). This setting is controlled by the `backend.registration.limit` Controller Setting. This limit applies whether the backends are resolved to a tier or unresolved.
- Agents apply a limit of 300 unresolved backends per application.

When the backend count exceeds the limit, the "All Other Traffic" category is automatically created. The logs indicate that a limit has been reached by a log entry such as the following:

```
[pool-1-thread-2] 21 Mar 2013 11:28:01,702 DEBUG AFastBackendResolver - Not discovering backend call [Exit
Point Type [WEB_SERVICE] Properties [{SERVICE_NAME=ZipCodeService002.wsdl}] Display Name [ZipCodeService002.
wsdl]] because maximum limit of discovered backends [300] reached.
```

Manage Backend Discovery

After reaching the backend registration limit, you should make sure that the systems being tracked as first-class registered backends are the ones that are important to you relative to those in the "All Other Traffic" category.

Also make sure that application environmental conditions are not causing an excess number of backends to be registered. These conditions may include:

- JMS queues that use the session ID in the destination. This causes each session to be identified as a separate backend.
- Calls to message queues hosted on the same server. In this case, you might not be interested in seeing each queue separately, and instead, want to aggregate everything for the same host and port to the same backend.
- Dynamic generation of IP addresses, such as when you host a service on Amazon Elastic Compute Cloud (Amazon EC2).
- Different JDBC drivers connecting to the same database may cause many backends. This can happen when there are slight variations in the driver metadata that AppDynamics uses to identify the database. For example, if two different JDBC drivers are used to access the same database, something as simple as a different spelling or format of the database name (ORACLE, oracle, or ORACL) can generate multiple database names when they are the actually the same physical database.

To manage backend registrations, you can change the configuration of the backend detection rules. For example, if a property causes excessive unique identification, consider removing the property or modifying its use in the detection and naming rule. If you created custom exit points, consider refining or removing them.

Delete Unnecessary Backends

Some backends may no longer have traffic. You can delete unused backends from the Remote Services List, any Remote Service dashboard, the Databases List, and any Database dashboard. If the backend has new traffic in the future, the app agent rediscovers it and reregisters it with the Controller.

You can also configure the Controller to automatically remove stale backends. See Stale Remote Service Removal.

Organize Backends in Flow Maps

Once you have configured detection for all the backends you require, you may need to organize the way they appear in the UI. See Group Remote Services on Flow Maps.

Exit Point Detection Rules

Custom exit points enable you to identify backend types that are not automatically detected. For example, you can define a custom exit point to monitor code calls to a file system read method.

After you have defined a custom exit point, the backend appears on the flow map with the type-associated icon you selected when you configured the custom exit point. See Custom Exit Points for Java.

Configuration Notes

You configure exit point detection in the **Configuration > Instrumentation** page. Select the Backend Detection tab and choose the application or tier on which to configure an exit point and the application type of the exit point, Java, .NET, and so on. From there you can add new exit points or modify existing ones.

The specific configuration options available differ based on the application type, but in general, you identify the exit point and specify the mechanism for it to be detected, such as a class and method name. These additional notes apply to exit point configuration:

- If the method you are using to identify the exit point is overloaded, you need to add the parameters that identify the signature for this form of the method.
- Match conditions let you restrict the method invocations for which you want AppDynamics to collect metrics based on runtime values tested by the condition. Match conditions can test parameter values or any value accessible by getter chain from the invoked object.
- Optionally, you can split an exit point based on a method parameter, the return value, or the invoked object.
- You can configure custom metrics and transaction snapshot data to collect for the backend.
- If you configure a custom backend for a method that encapsulates an automatically detected backend, such as an HTTP client call, you may lose correlation from the automatically detected exit call.
- Splitting a backend on return value may break correlation.

> ⚠ You are limited to 50 characters for the exit point value when adding a custom exit point or splitting/grouping exit points.

Adding a Custom Exit Point

When adding a custom exit point, you specify the class and method that identify an external call to the exit point. You can refine the call by specifying a return value, parameter, or getter chain match value.

The type you choose determines the icon that appears in the flow map for the custom exit point. If the type is not listed, select **Use Custom** and enter a string in the **Type** field to identify the exit point calls in the UI.

Note that you can also split the exit point based on these values. If you add a split condition for an exit point, it means that any exit point that matches the overall match condition for the custom exit point is further evaluated against the split condition. If its call matches the split condition, it is given the more specific name you configure in the split exit point configuration.

High Volume Exit Points

The **Is High Volume** option for custom exit points lets you create an optimized exit point configuration for backend systems with high-performance requirements, such as caching servers or in-memory databases.

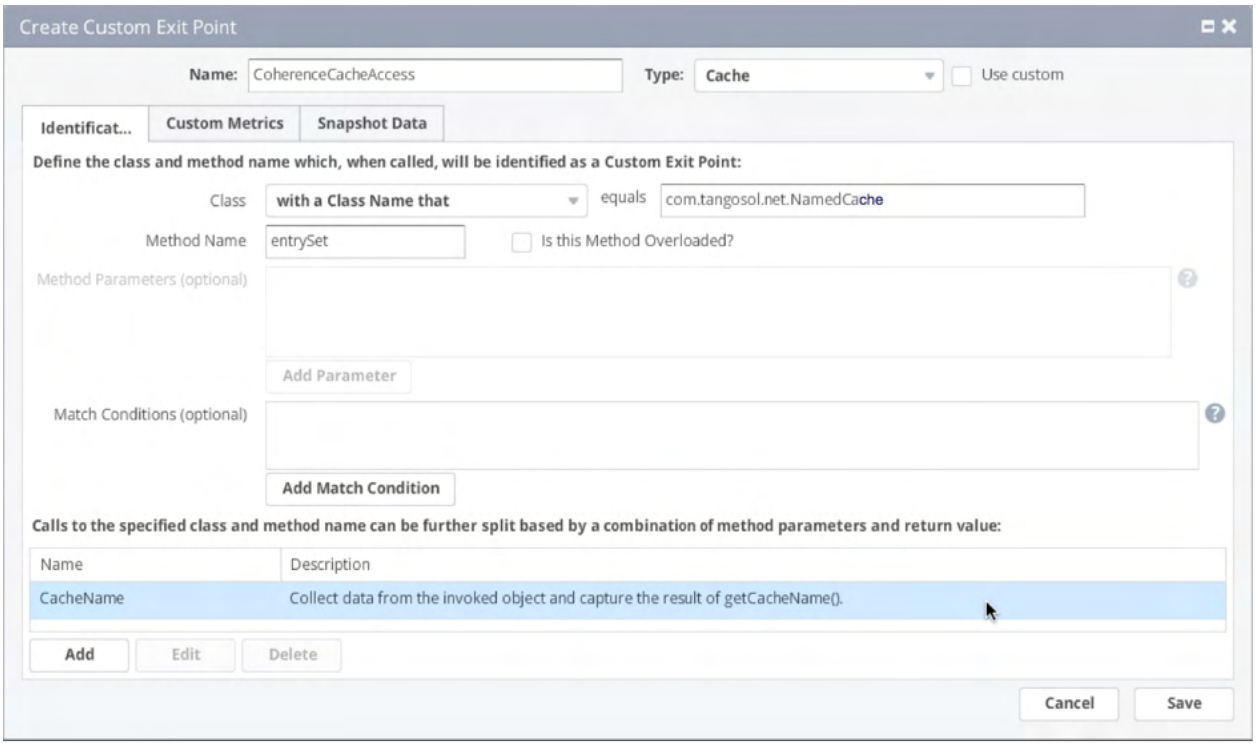
A "high volume" exit point keeps the overhead associated with the exit point to a negligible amount by bypassing certain processing operations on calls to this exit point. As a result, transaction correlation and error detail reporting are not available for the exit point, nor are match conditions in the exit point configuration.

With a high volume exit point, you do get the number of calls, number of errors, and average response time for calls.

Split and Group Exit Points

By splitting an exit point, you can use dynamic application values to identify an exit point. Specifically, you can use the value of either a method parameter, the method return value, or a value returned by a specified getter chain on the invoked object of the identified method. The configuration settings you use to configure exit point splitting are the same as used for Data Collectors. See Data Collectors for information on how to use the configuration UI.

A simple example of how you would use exit point splitting is for a cache exit point. In this example, say you configure splitting for an exit point that identifies a method for writing to the cache. The object on which the method for writing to the cache offers a method to get the current cache name, the `getCacheName()` method. By configuring a split on this method (as a getter chain on the invoked object), you can have the exit point named with the name of the cache node, as dynamically determined:



Splitting also lets you group exit points. Given an exit point for `NamedCache.entrySet()`, as shown in the example above, suppose we created another exit point. This one uses the `getAll()` method on the same class, `NamedCache`. It also has a split configuration that uses the `getCacheName()` method of the invoked object.

If the `getCacheName()` getter chain points to the same cache name when the `getAll()` and the `entrySet()` methods are invoked, they will be shown as calls to the same backend.

Java Backend Detection

Related pages:

- Exit Point Detection Rules
- Monitor Databases
- Remote Services

The AppDynamics Java Agent automatically discovers common types of remote services as backend systems based on the exit point associated with the backend. If AppDynamics does not automatically discover a backend system used in your environment, check whether it's an automatically discovered type (based on the list below) and compare the default exit points for the type to that used in your application environment.

Each automatically discovered backend type has a default discovery rule. Many backend types have a set of backend naming properties. Where individual properties are configurable, you can configure the property as follows:

- Use the property as is.
- Use segments of the property.
- Run a regular expression on the property. See Using Regular Expressions.
- Execute methods on the property. See Using Getter Chains.

For instructions to revise backend discovery rules, see Backend Detection Rules.

Amazon Simple Notification Service Backends

AppDynamics detects exit calls to the Amazon SNS messaging service using the Amazon SNS Client: `AmazonSNSClient.publish`.

Amazon SNS Naming Properties

You can enable or disable the use of the following properties for Amazon S3 backend identification:

Configurable Properties	Default Automatic Amazon SNS Backend Discovery/Naming
Vendor	Yes
Topic ARN	Yes

Amazon Simple Queue Service Backends

AppDynamics can detect the following types of Amazon Web Services SQS message queue actions:

- Basic send/receive
- Batched send/receive
- Asynchronous send/receive

Correlating SQS traffic requires you to configure the continuation entry point for the SQS message.

Amazon Simple Storage Service Backends

AppDynamics automatically detects the following exit points for Amazon Web Services S3 backends:

- `AmazonS3Client.deleteObject`
- `AmazonS3Client.getObject`
- `AmazonS3Client.getObjectMetadata`
- `AmazonS3Client.listObjects`
- `AmazonS3Client.putObject`

Amazon S3 Naming Properties

You can enable or disable the use of the following properties for Amazon S3 backend identification:

Configurable Properties	Default Automatic Amazon S3 Discovery/Naming
Bucket Name	Yes
Vendor	Yes
Object Key	No

Amazon Web Services Backends

AppDynamics detects the following exit calls to the following Amazon Web Services:

- DynamoDB requests using the `AmazonDynamoDBClient` create, read, update, and delete (CRUD) methods using the Amazon low-level DynamoDB API. The agent discovers the Document API calls as the underlying low-level API call.

Amazon Web Service Naming Properties

For Amazon Web Services, you can enable or disable the use of the following naming properties for backend identification:

Configurable Properties	Default Automatic Amazon Web Service Backend Discovery/Naming
Vendor	Yes
Service	Yes
Endpoint	Yes

Apache Cassandra CQL Backends

By default, AppDynamics automatically detects and identifies exit calls to Apache Cassandra using Thrift and DataStax drivers.

> Exceptions returned from Cassandra backends show up as separate exit calls. For the DataStax 1.0 driver, the Java Agent detects an "unknown" Cassandra backend in order to capture the error details.

Cassandra Backend Naming Properties

You can enable or disable the use of the properties below for Cassandra backend identification:

> When you configure Cassandra backend naming, consider the following:
>
> - For DataStax clients, AppDynamics does not recommend using the keyspace property. Due to driver limitations, the keyspace property is only available when `Statement.setKeyspace` is called explicitly in Cassandra. This can lead to the discovery of multiple backends.
> - Thrift clients don't support Cluster Name, Rack or Data Center properties, so those properties return null for Thrift.

Configurable Properties	Default Automatic Cassandra CQL Backend Discovery/Naming
Cluster Name	Yes
Keyspace	No
Host	Yes
Port	Yes
Data Center	No
Rack	No

Apache Kafka Backends

By default, the Java Agent detects publish activity to Apache Kafka as an exit point from a Java node. You can configure the Java Agent to detect Kafka consumer activity as an entry point, see Apache Kafka Consumer Backends.

Kafka Backend Naming Properties

You can enable or disable the use of the following properties for Kafka backend identification:

Configurable Properties	Default Automatic IBM MQ Backend Discovery/Naming
Vendor	Yes
Topic Name	Yes
Broker URL	No

Default Kafka JMX Metrics

There are lots of JMX metrics that are exposed by Kafka Producer and Consumer. We have selected a few of the metrics that we think could be useful for customers and created OOTB rules for them. PFB the list of metrics supported by Producer and Consumer respectively. In case you are interested in any of the other metric for which OOTB rule is not available you can go to MBean browser, select the metric and create a rule out of it.

Couchbase Backends

By default, AppDynamics automatically detects and identifies exit calls made to the Couchbase cluster node.

Couchbase Backend Naming Properties

You can enable or disable the use of the following properties for Couchbase exit backend identification:

Configurable Properties	Default Automatic Couchbase Backend Discovery/Naming
Server Pool	Yes
Vendor	Yes
Bucket Name	Yes

HTTP Backends

HTTP exit point activity includes all HTTP calls done outside of a web service call. Web service calls are not considered an HTTP exit point. By default, the Java Agent names the HTTP backend for the Host and Port properties. For instance, "myHTTPHost:5000".

HTTP Backend Naming Properties

You can enable or disable the use of the following properties for HTTP backend identification:

Configurable Properties	Default Automatic HTTP Backend Discovery/Naming
Host	Yes
Port	Yes
URL	No
Query String	No

For examples of common HTTP backend configurations, see HTTP Backend Detection.

For sample JDBC backend configurations, see Example JDBC Backend Configuration.

IBM Websphere Message Queue Backends

IBM MQ, also known as IBM WebSphere MQ and IBM MQSeries, is IBM's message-oriented middleware similar to JMS. Several additional properties are configurable, such as host and port. This is useful where you have lots of queues and you want to monitor them based on a subset of the properties.

IBM MQ Backend Naming Properties

You can enable or disable the use of the following properties for IBM MQ backend identification:

Configurable Properties	Default Automatic IBM MQ Backend Discovery/Naming
Destination	Yes
Destination Type	Yes
Host	Yes
Port	Yes
Major Version	Yes
Vendor	Yes

For examples of common message queue configuration scenarios, see Example Message Queue Backend Configuration.

Java Message Service Backends

JMS backend activity includes all JMS message send and publish activity. By default, AppDynamics identifies JMS back ends using the properties of the JMS server such as: vendor, destination name, and destination type. The default configuration uses all three properties of the JMS queue.

JMS Backend Naming Properties

You can enable or disable the use of the following properties for JMS backend identification:

Configurable Properties	Default Automatic JMS Backend Discovery/Naming
Destination	Yes
Destination Type	Yes
Vendor	Yes

Transaction names are derived from the enabled properties, for example, "ActiveMQ-OrderQueue".

For examples of common message queue configuration scenarios, see Example Message Queue Backend Configuration.

JDBC Backends

JDBC backend activity consists of all JDBC calls including inserts, queries, updates, getting connections from connection pools, and so on.

Class	Method
`oracle.jdbc.driver.PhysicalConnection`	• `prepareStatement` • `prepareCall` • `commit` • `rollback`
`oracle.jdbc.driver.OracleStatement`	• `executeQuery` • `executeUpdate` • `execute`
`oracle.jdbc.driver.OracleDataSource`	• `getConnection`
`oracle.jdbc.driver.OraclePreparedStatement`	• `executeQuery` • `addBatch` • `execute` • `executeUpdate`
`oracle.jdbc.driver.OraclePreparedStatementWrapper`	• `execute` • `executeQuery` • `executeUpdate`

The Java Agent measures the response time for a database call as the round trip time it takes for a JDBC call to return. The response time for a database call includes network round trip time + database connection time + SQL query time or any other time spent in the database.

By default, the agent identifies JDBC backends using the following logical properties of the database:

- URL
- Host name
- Port number
- Database schema
- Version
- Vendor

If a database backend has the same hostname, port number, and database type as a database server already configured in a database Collector, the database backend is automatically matched with the Collector, and drill-downs from the Application Flow Map, Tier Flow Map or Node Flow Map to Database Monitoring are enabled.

JDBC Backend Naming Properties

You can enable or disable the use of the following properties for JDBC backend identification:

Configurable Properties	Default Automatic JDBC Backend Discovery/Naming	Description
URL	Yes	JDBC URL provided to the driver
Host	Yes	Database host
Port	Yes	Database port
Database	Yes	Database schema
Version	Yes	Database version as reported by JDBC driver
Vendor	Yes	Database vendor as reported by JDBC driver

Jolt Backends

AppDynamics detects exit calls to Tuxedo services from the JoltRemoteService class in the Jolt Class Library.

Jolt Naming Properties

You can enable or disable the use of the following naming properties for Jolt backend identification.

Configurable Properties	Default Automatic Jolt Backend Discovery/Naming
Host	Yes
Port	Yes
Service	Yes

MongoDB Backends

By default, the Java Agent detects exit calls to MongoDB using the Mongo Java Driver over the MongoDB Wire Protocol. The agent detects the exit point on the Wire Protocol methods and excludes the methods between the external API and the Wire Protocol.

MongoDB Backend Naming Properties

Configurable Properties	Default Automatic MongoDB Backend Discovery/Naming
Host	Yes
Port	Yes
Database	Yes

RabbitMQ Backends

RabbitMQ is open source, commercially supported, messaging middleware that runs on many different operating systems. The Java Agent discovers exit points using the RabbitMQ Java API, amqp-client.jar in most distributions. By default, RabbitMQ backends are identified by Host, Port, Routing Key, and Exchange. For instance: "amqp://guest@127.0.0.1:5672/exchange/task_queue".

RabbitMQ Backend Naming Properties

You can enable or disable the use of the following properties for RabbitMQ backend identification:

Configurable Properties	Default Automatic RabbitMQ Backend Discovery/Naming
Host	Yes
Port	Yes
Routing Key	Yes
Exchange	Yes

For examples of common message queue configuration scenarios, see Example Message Queue Backend Configuration.

RMI Backends

The Java Agent automatically discovers backends called using the standard Java RMI API. For a list of supported RMI frameworks, see 'RPC/Web Services API/HTTP Client Support' on Java Supported Environments.

RMI Backend Naming Property

The JAVA Agent names the RMI backend for the URL. You can configure how the agent uses the URL to name the RMI backend.

Thrift Backends

By default, AppDynamics automatically detects and identifies Apache Thrift exit points (backends).
See Apache Thrift for details.

Thrift Backend Naming Properties

You can enable or disable the use of the following properties for Thrift exit backend identification:

Configurable Properties	Default Automatic Thrift Backend Discovery/Naming
Host	Yes
Port	Yes
transport	Yes

Web Services Backends

Web service backend activity includes all web service invocations. Web service backends are identified using the web service name.

Web Services Backend Naming Properties

Configurable Properties	Default Automatic Web Service Backend Discovery/Naming
Service	Yes
URL	No
Operation	No
Soap Action	No
Vendor	No

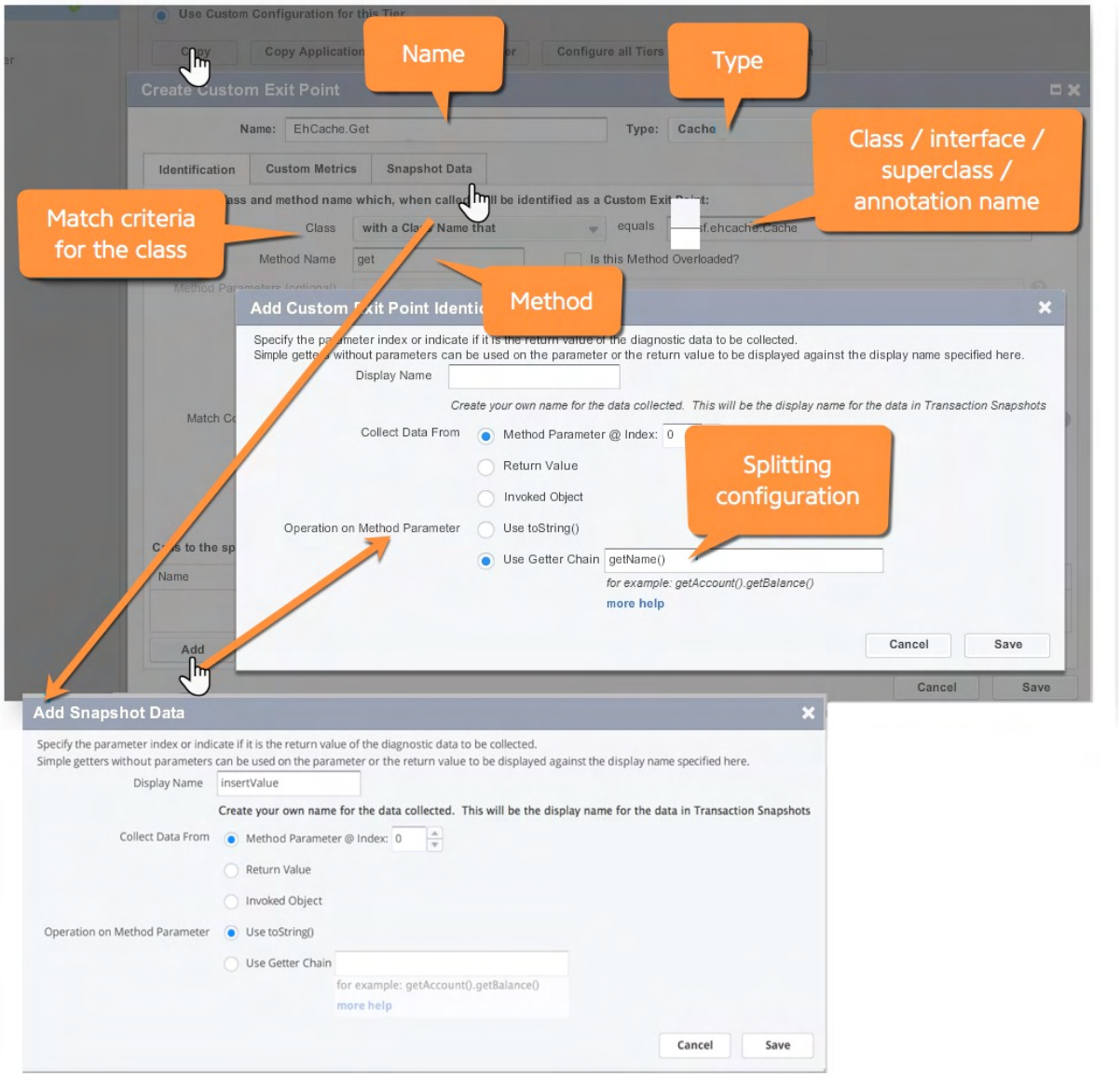
By default, the gRPC backend is identified as `ServiceName-gRPC`. This is also configurable under web services backend.

Custom Exit Points for Java

This page describes settings you can use to create custom exit point configurations for specific backends in Java environments.

In general, custom exit rules match calls to custom backends by class and method name. In some cases, you need to configure splitting as well. The following tables give you the class and method names to use for the various types.

Enter the configuration settings in the corresponding fields in the custom exit point UI. The following screenshot illustrates a sample configuration for an EhCache custom match rule:



For more about creating custom exit points, see create a custom exit point.

> Oracle Coherence, EhCache, and Memcached backends are detected automatically as high volume exit points. You can create custom exit points for the backends as described here for particular backend splitting and naming results.

Coherence Exit Points

Name of the Exit Point	Type	Method Name	Class Match Criteria	Class/Interface/Superclass /Annotation Name	Method Name	Splitting Configuration
Coherence.Put	Cache		that implements an interface which	com.tangosol.net.NamedCache	put	getCacheName()
Coherence.PutAll	Cache		that implements an interface which	com.tangosol.net.NamedCache	putAll	getCacheName()

Coherence.EntrySet	Cache		that implements an interface which	com.tangosol.net.NamedCache	entrySet	getCacheName()
Coherence.KeySet	Cache		that implements an interface which	com.tangosol.net.NamedCache	keySet	getCacheName()
Coherence.Get	Cache		that implements an interface which	com.tangosol.net.NamedCache	get	getCacheName()
Coherence.Remove	Cache		that implements an interface which	com.tangosol.net.NamedCache	remove	getCacheName()

EhCache Exit Points

Name of the Exit Point	Type	Class Match Criteria	Class/Interface/Superclass/Annotation Name	Method Name	Splitting Configuration
EhCache.Get	Cache	With a class name that	net.sf.ehcache.Cache	get	getName()
EhCache.Put	Cache	With a class name that	net.sf.ehcache.Cache	put	getName()
EhCache.PutIfAbsent	Cache	With a class name that	net.sf.ehcache.Cache	putIfAbsent	getName()
EhCache.PutQuiet	Cache	With a class name that	net.sf.ehcache.Cache	putQuiet	getName()
EhCache.Remove	Cache	With a class name that	net.sf.ehcache.Cache	remove	getName()
EhCache.RemoveAll	Cache	With a class name that	net.sf.ehcache.Cache	removeAll	getName()
EhCache.RemoveQuiet	Cache	With a class name that	net.sf.ehcache.Cache	removeQuiet	getName()
EhCache.Replace	Cache	With a class name that	net.sf.ehcache.Cache	replace	getName()

LDAP Exit Points

Name of the Exit Point	Type	Class Match Criteria	Class/Interface/Superclass/Annotation Name	Method Name
LDAPExitPoint.Bind	LDAP	With a class name that	javax.naming.directory.InitialDirContext	bind
LDAPExitPoint.Rebind	LDAP	With a class name that	javax.naming.directory.InitialDirContext	rebind
LDAPExitPoint.Search	LDAP	With a class name that	javax.naming.directory.InitialDirContext	search
LDAPExitPoint.ModifyAttributes	LDAP	With a class name that	javax.naming.directory.InitialDirContext	modifyAttributes
LDAPExitPoint.GetNextBatch	LDAP	With a class name that	com.sun.jndi.ldap.LdapNamingEnumeration	getNextBatch
LDAPExitPoint.NextAux	LDAP	With a class name that	com.sun.jndi.ldap.LdapNamingEnumeration	nextAux
LDAPExitPoint.CreatePooledConnection	LDAP	With a class name that	com.sun.jndi.ldap.LdapClientFactory	createPooledConnection
LDAPExitPoint.Search	LDAP	With a class name that	com.sun.jndi.ldap.LdapClientFactory	search
LDAPExitPoint.Modify	LDAP	With a class name that	com.sun.jndi.ldap.LdapClientFactory	modify

Mail Exit Points

Name of the Exit Point	Type	Class Match Criteria	Class/Interface/Superclass/Annotation Name	Method Name
MailExitPoint.Send	Mail Server	With a class name that	javax.mail.Transport	send
MailExitPoint.SendMessage	Mail Server	With a class name that	javax.mail.Transport	sendMessage

Memcached Exit Points

Name of the Exit Point	Type	Class Match Criteria	Class/Interface/Superclass/Annotation Name	Method Name
Memcached.Add	Cache	With a class name that	net.spy.memcached.MemcachedClient	add
Memcached.Set	Cache	With a class name that	net.spy.memcached.MemcachedClient	set
Memcached.Replace	Cache	With a class name that	net.spy.memcached.MemcachedClient	replace
Memcached.CompareAndSwap	Cache	With a class name that	net.spy.memcached.MemcachedClient	cas
Memcached.Get	Cache	With a class name that	net.spy.memcached.MemcachedClient	get

Memcached.Remove	Cache	With a class name that	net.spy.memcached.MemcachedClient	remove

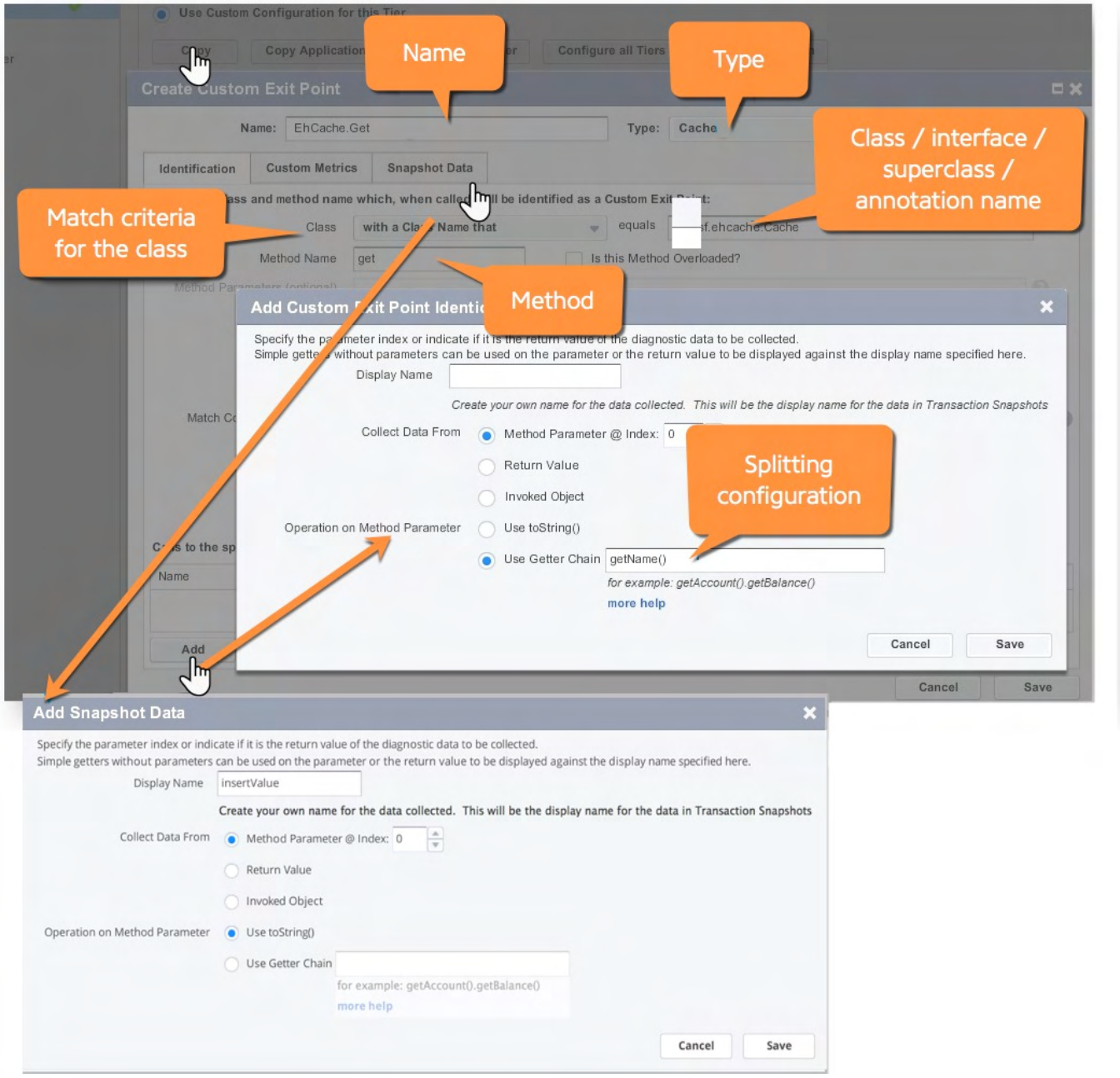
MongoDB Exit Points

Name of Exit Point	Type	Class Match Criteria	Class/Interface/Superclass/ Annotation Name	Method Name	Splitting Config/Custom Exit Point Identifier		Snapshot Data Operation on Invoked Object
					Collect Data From	Getter Chain Operation on Invoked Object	
MongoDB.Insert	JDBC	With a class name that	com.mongodb.DBCollection	insert	Invoked Object	getDB().getName()	Parameter_0.toString()
MongoDB.Find	JDBC	With a class name that	com.mongodb.DBCollection	find	Invoked Object	getDB().getName()	Parameter_0.toString()
MongoDB. Update	JDBC	With a class name that	com.mongodb.DBCollection	update	Invoked Object	getDB().getName()	Parameter_0.toString()
MongoDB. Remove	JDBC	With a class name that	com.mongodb.DBCollection	remove	Invoked Object	getDB().getName()	Parameter_0.toString()
MongoDB.Apply	JDBC	With a class name that	com.mongodb.DBCollection	apply	Invoked Object	getDB().getName()	Parameter_0.toString()

Avoiding high overhead

AppDynamics recommends that you avoid instrumenting highly-used methods, such as the find() method, because the activity around these methods can cause an undesired amount of overhead on the system.

Sample MongoDB Exit Point Configuration



SAP Exit Points

Name of the Exit Point	Type	Class Match Criteria	Class/Interface/Superclass/Annotation Name	Method Name
SAP.Execute	SAP	With a class name that	com.sap.mw.jco.rfc.MiddlewareRFC$Client	execute
SAP.Connect	SAP	With a class name that	com.sap.mw.jco.rfc.MiddlewareRFC$Client	connect
SAP.Disconnect	SAP	With a class name that	com.sap.mw.jco.rfc.MiddlewareRFC$Client	disconnect
SAP.Reset	SAP	With a class name that	com.sap.mw.jco.rfc.MiddlewareRFC$Client	reset
SAP.CreateTID	SAP	With a class name that	com.sap.mw.jco.rfc.MiddlewareRFC$Client	createTID
SAP.ConfirmTID	SAP	With a class name that	com.sap.mw.jco.rfc.MiddlewareRFC$Client	confirmTID

Example Message Queue Backend Configuration

Monitor the Server by Ignoring the Queue Name

In JMS example, the application is making calls to a message server that handles several queues. The sample destination names look like this:

- AccountQ
- AccountReplyQ
- AccountRecQ
- AccountDebitQ

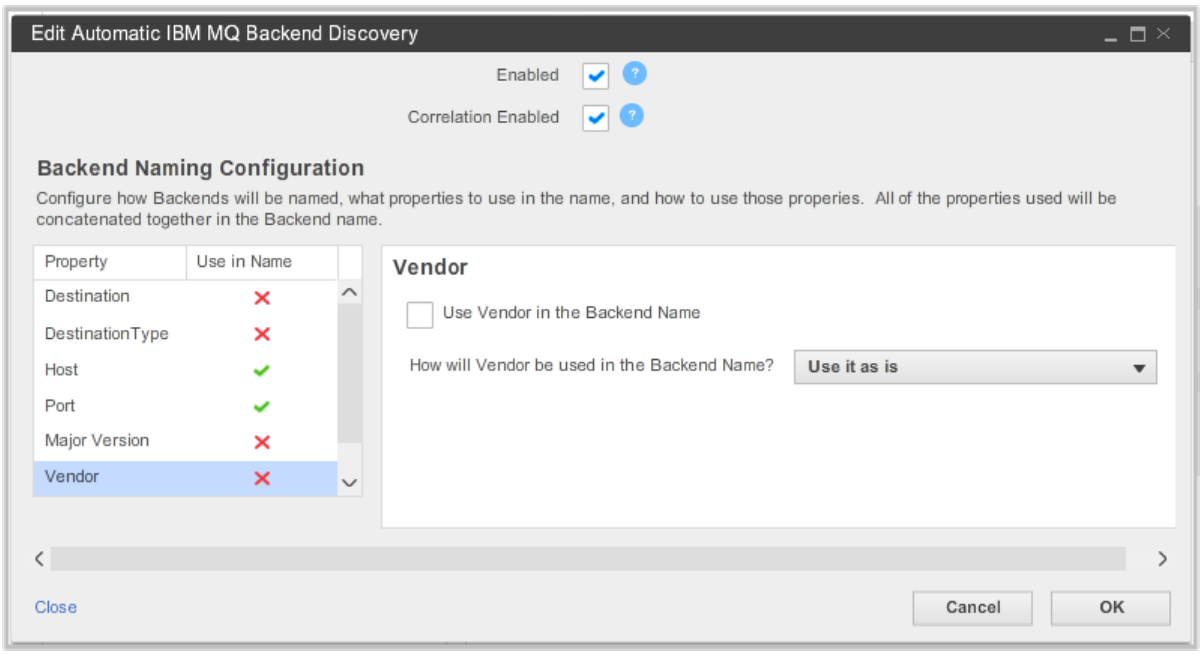
The default automatic discovery rule detects one backend for each unique destination and so the flow map shows one queue backend for each different queue name. In this example, each of the above would show as a separate backend on the application flow map. If you are interested in monitoring the performance of the server and not each queue name, you can modify the configuration to ignore the Destination property and use just the Type and Vendor.

To achieve this result, you create a new custom JMS Discovery Rule.

Taking IBM MQ for another example, the application is making calls to a message server that handles several queues. The sample destination names look like this:

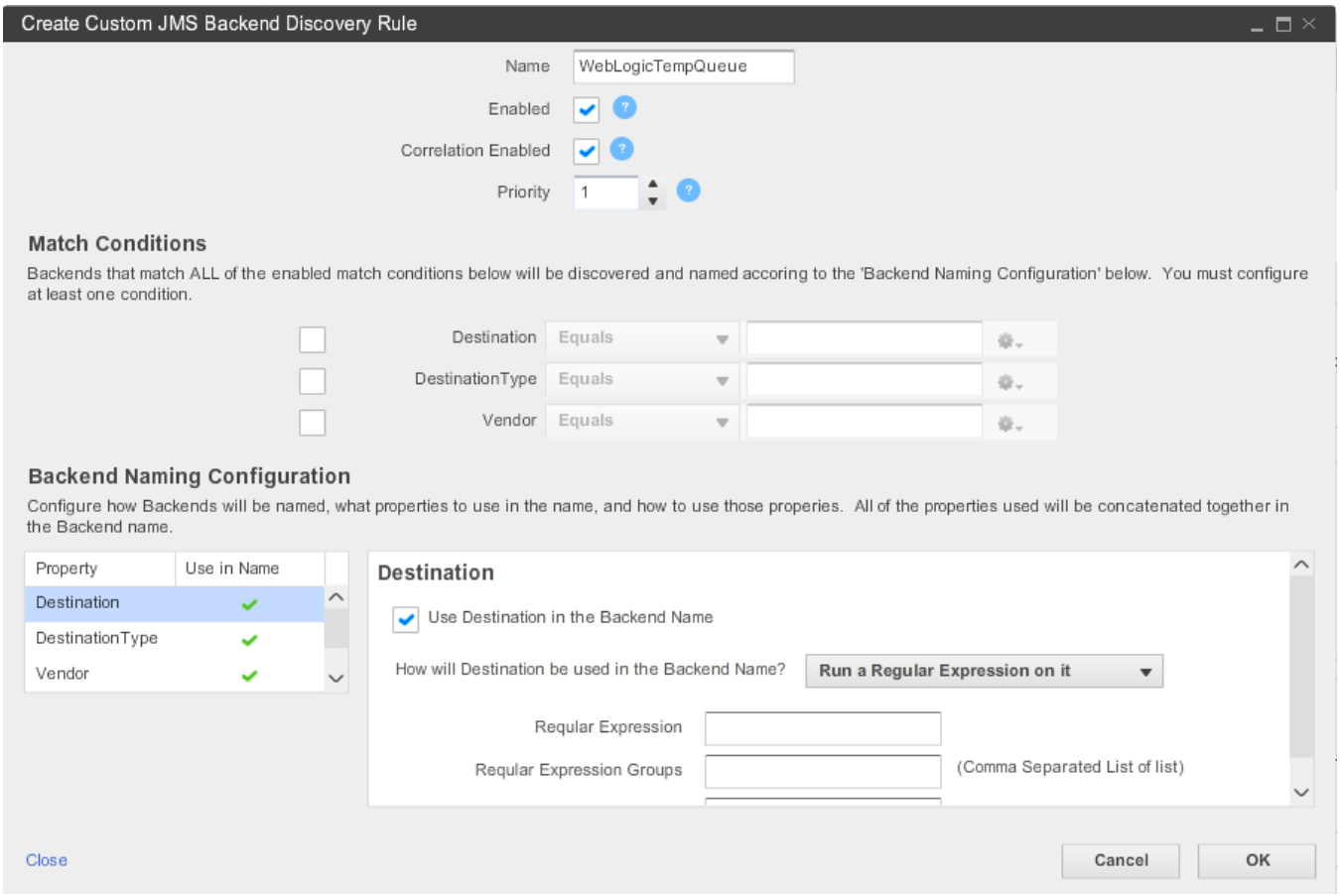
- MQhostwest-US:1521
- MQhosteast-US:1521
- MQhostsouth-US:1521

The default automatic discovery rule detects one backend for each unique destination and so the flow map shows one queue backend for each different queue name. In this example, each of the above would show as a separate backend on the application flow map. If you are interested in monitoring the performance of the server and not each queue name, you can create a discovery rule that just uses the Host and Port, as follows:



Temporary Queues

In this example an application creates many temporary JMS response queues that are deleted after the message is received. By default, AppDynamics discovers these queues separately and lists each one as a unique remote service. This default behavior probably does not enable effective monitoring. Instead, you can create a custom JMS discovery rule stating that if the destination name contains "TemporaryQueue", list it as "WeblogicTempQueue", or whatever name makes sense in your monitoring environment. In this way, you can monitor the performance that matters. The configuration to accomplish this is shown in the following screen shot:



Session ID in the Queue Name

If your JMS queues use the session ID in the destination, this causes each session to be identified as a separate backend. In this case, you might not be interested in seeing each queue separately, and instead want to aggregate everything for the same host and port to the same backend. You can generate the right name and the correct number of backends to monitor by editing the automatic discovery rule. Doing this enables you to monitor the key performance indicators (KPIs) that are of most interest to you.

Example JDBC Backend Configuration

Related pages:

- Java Backend Detection
- Using Regular Expressions

Depending on exactly what you need to monitor, you may want to change the default JDBC configuration. When you see the same physical database represented as multiple JDBC databases, you may need to revise the automatic discovery rule. Doing this enables you to more effectively monitor the key performance indicators (KPIs) that are of most value to you.

Multiple Databases from Same Vendor

JDBC connections to the same physical Oracle database (with the same URI) may appear as multiple backends. In some circumstances, the Vendor property captured for the database is different. This can happen when different drivers are used to access the database. For example, you might see JDBC backends with the following vendor names:

- Oracle DB
- Oracle

If the database driver package name matches the standard Oracle database driver, then the vendor name used is "Oracle DB". If it doesn't match, then the product name from the database metadata (using the java.sql.DatabaseMetaData class) is used as a vendor name. So database calls that use different drivers to reach the same physical database may be detected as separate databases. You can fix this by disabling the use of the Vendor property in the discovery rule.

JDBC with Complex URLs

In this example, the database URL is configured for high availability, so it is quite long and complex. Choosing the **Run a regular expression on it** URL option is the way to go. Disable the use of the Host and Port properties for JDBC automatic discovery. Instead enable the use of the URL that appears in the JDBC call, along with a regular expression to get the correct database naming and discovery.

For example, to extract all the hosts and all the ports from the following URL:

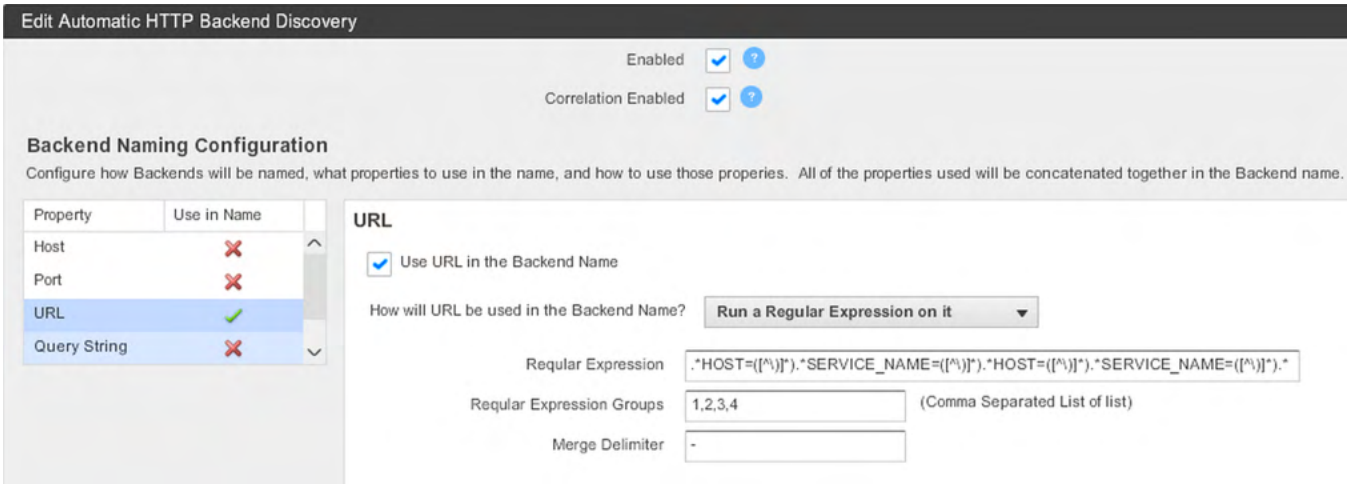
```
jdbc:oracle:thin:@(DESCRIPTION_LIST=(LOAD_BALANCE=OFF)(FAILOVER=ON)(DESCRIPTION=(ADDRESS_LIST=(LOAD_BALANCE=ON)
(ADDRESS=(PROTOCOL=TCP)(HOST=titanpfmcl01)(PORT=1521)))(CONNECT_DATA=(SERVICE_NAME=trafrefpfm01.global.
trafigura.com)))(DESCRIPTION=(ADDRESS_LIST=(LOAD_BALANCE=ON)(ADDRESS=(PROTOCOL=TCP)(HOST=titanpfmcl02)(PORT =
1521)))(CONNECT_DATA=(SERVICE_NAME=trafrefpfm01.global.trafigura.com))))
```

This sample is for a string that contains the host and service name twice. You can also use port in your regular expression if needed by your requirements.

The following regular expression applied to the previous high availability URL results in a backend name similar to this: titanpfmcl01-trafrefpfm01.global.trafigura.com-titanpfmcl02-trafrefpfm01.global.trafigura.com.

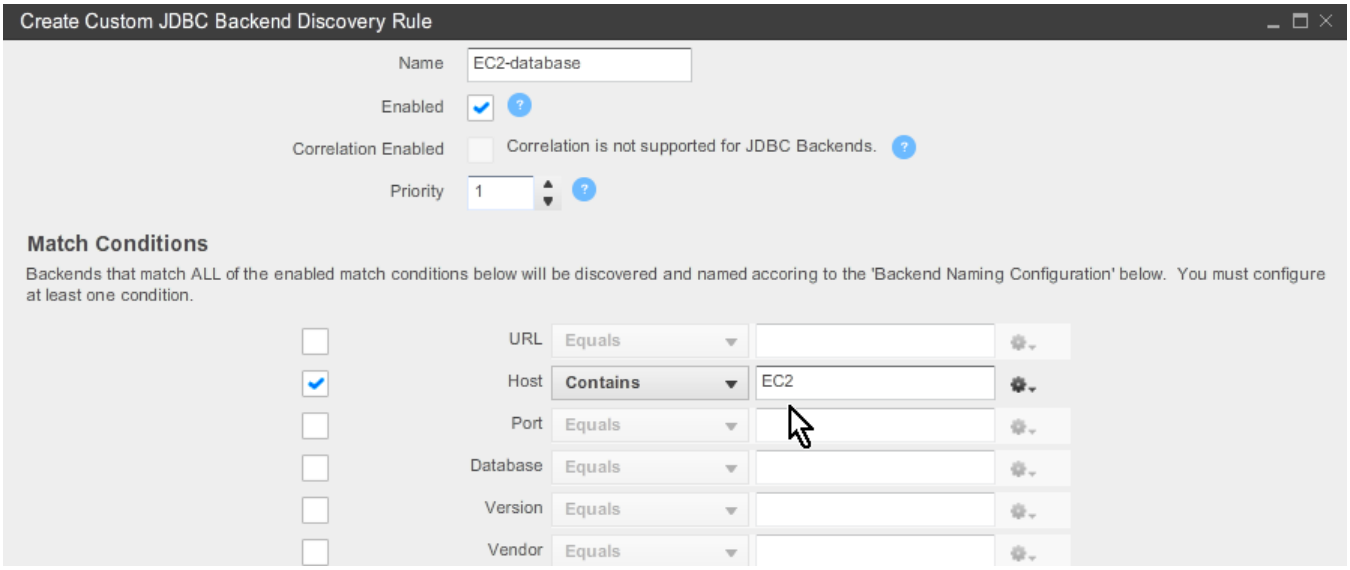
```
.*HOST=([^\)]*).*SERVICE_NAME=([^\)]*).*HOST=([^\)]*).*SERVICE_NAME=([^\)]*).*
```

Note: the expression starts and end with "." Set **Regular Expression Groups** to **1,2,3,4**.
Set the **Merge Delimiter** to "-".
This configuration looks like this in the UI:

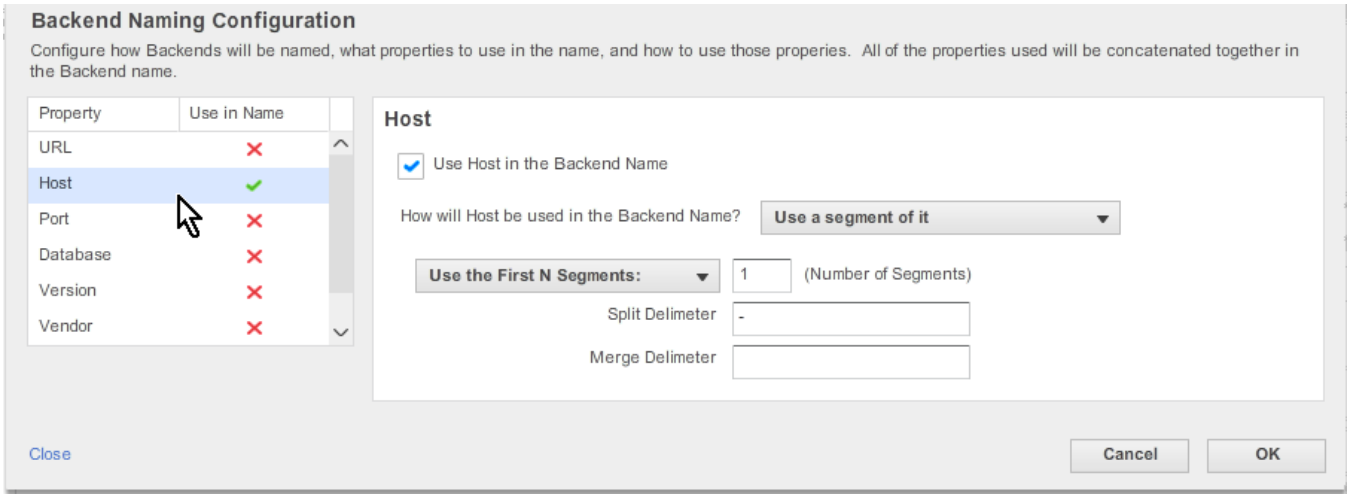


EC2 Hosted Databases

AppDynamics automatically discovers JDBC backends based on host, port, URL, database, version and vendor values. To monitor all JDBC databases that contain "EC2" in their host names as a single database, create a JDBC custom discovery rule and use the following match condition: Host Contains "EC2" as shown in the following screen shot.



Assuming host names of the format "EC2-segment2-segment3", use the following naming configuration:



This configuration results in a single database icon on the flow map named "EC2".

Apache Kafka Consumer Backends

You can configure the AppDynamics Java Agent to detect entry points for Apache Kafka consumer activity using `KafkaConsumer.poll()` (introduced in Kafka v0.10) and `Kafka SimpleConsumer.fetch()` (introduced in Kafka v0.9). Prior to version 0.11, the Kafka payload did not include a location to store correlation data, so end-to-end Business Transaction correlation is only possible with the Kafka client and broker >= 0.11.

KafkaConsumer.poll

To instrument Kafka consumer entry points using `KafkaConsumer.poll()`, identify the method in which the consumer reads messages in a loop with a custom interceptor definition. We instrument the iterator's next method to start and end the Business Transaction for each message. There could be many iterators used for iterating messages but we only support iterators of the following types:

- `kafka.consumer.ConsumerIterator`
- `org.apache.kafka.clients.consumer.ConsumerRecords$ConcatenatedIterable$1`

1. Identify the class and method of the loop that processes messages from Kafka. Consider, for example, a class `MyConsumer` that employs the following loop to poll and process messages from Kafka:

   ```
   ConsumerRecords<String, String> records = kafkaConsumer.poll(1000);
   pollMessages(records);
   private void pollMessages(ConsumerRecords<String, String>records) throws Exception {
      //AppDynamics instrumentation gets applied here
   for (ConsumerRecord<String, String> record : records) {
      //Processing of the records
      System.out.println(record.value());
      }
   }
   ```

   For this case, you want to intercept:

   - Class: `MyConsumer`
   - Method: `pollMessages`

   The interceptor can also be applied to a method that processes individual records, not just a loop. For example:

   ```
   ConsumerRecords<String, String> records = kafkaConsumer.poll(1000);
   pollMessages(records);
   private void pollMessages(ConsumerRecords<String, String>records) throws Exception {
      //AppDynamics instrumentation gets applied here
   for (ConsumerRecord<String, String> record : records) {
      processRecord(record)
      }
   }
   ```

2. Use your preferred text editor to create and edit a file named custom-interceptors.xml at the following path:
   `<agent_home>/<version_number>/conf`

   For example:
   `/usr/home/appdynamics/appagent/ver4.3.1.0/conf/custom-interceptors.xml`
3. Copy the following XML to `custom-interceptors.xml`:

   ```
   <custom-interceptors>
       <custom-interceptor>
           <interceptor-class-name>com.singularity.KafkaMarkerMethodInterceptor</interceptor-class-name>
           <match-class type="matches-class">
               <name filter-type="equals">my-fully-qualified-class-name</name>
           </match-class>
           <match-method>
               <name>my-method-name</name>
           </match-method>
       </custom-interceptor>
   </custom-interceptors>
   ```

4. Set the value of the class name to the name of your consumer class. For instance, to specify the `MyConsumer` class:

```
<match-class type="matches-class">
    <name filter-type="equals">com.mycompany.mypackage.MyConsumer</name>
</match-class>
```

5. Set the value of the method name to the name of your message processing loop method. For instance, to specify the `pollMessages` method:

```
<match-method>
    <name>pollMessages</name>
</match-method>
```

After the Java Agent reads the updated configuration, it detects consumer activity and upstream Kafka queue. The application flow map shows the tier receiving data from the Kafka queue.

Kafka SimpleConsumer Entry Points

To enable consumer entry points for Kafka clients that retrieve messages using `SimpleConsumer.fetch()`, register the `enable-kafka-consumer` node property with a value of "true."

Kafka consumer activity shows up as an exit call in this case.

.NET Backend Detection

The .NET Agent automatically detects many common backend types. Most backend types have a default discovery rule and a set of configurable properties. To revise backend discovery rules, see Backend Detection Rules and Exit Point Detection Rules.

For all other traffic transactions, see Business Transactions.

ADO.NET Backends

The .NET Agent automatically discovers ADO.NET data providers implementing standard Microsoft interfaces as database backends. See .NET Supported Environments.

Because the ADO.NET API is interface-based, AppDynamics instruments all ADO.NET database providers that implement these interfaces by default.

AppDynamics uses database identification information from the ADO.NET connection string. The connection string specifies the server address and schema, or the local file name. Most connection strings are formatted based on common rules that you can parse and distill to a database name. However because no standards exist for the connection string, the ADO.NET provider implementer determines the format.

For some providers, AppDynamics may fail to parse the connection string. In these cases, the .NET Agent uses the complete connection string minus any user password. The property is labeled ADO.NET connection string, and the value shows the connection string minus any user password.

For example, the .NET Agent names this database backend using the connection string pattern `<datasource name>-<database name>`

```
.\SQLEXPRESS-HowdyWorldDB
```

ADO.NET Configurable Properties

You can enable or disable the use of these properties for ADO.NET exit points:

Configurable Properties	Default Detection and Naming Property?	Description
`Host`	Yes	Data source or database server
`Database`	Yes	Database name
`Vendor`	No	Type of client-side ADO.NET library
`Connection String`	No	Full connection string with password filtered out
`Port`	No	Port number

Directory Service Backends

The .NET Agent automatically discovers exit calls to directory services that use the System.DirectoryServices.Protocols (S.DS.P) libraries.

The agent names the backend for the server name. If the agent cannot derive the server name from the request, it then constructs a name using Domain Component (DC) values.

For example: `activedirectory.example.com`

HTTP Backends

AppDynamics automatically detects HTTP exit points (backends). See HTTP Backend Detection.

.NET Agent for Windows

For the .NET Agent for Windows, the default HTTP automatic discovery rule uses the URL property. From the enabled properties, AppDynamics derives a display name using the URL.

For example: `http://api.example.com:8989/searchfares`

.NET Agent for Linux

For the .NET Agent for Linux, the default HTTP automatic discovery rule for backends uses the host name and port number.

For example: `http://api.example.com:8989/`

HTTP Configurable Properties

For both Windows and Linux versions of the .NET Agent, you can enable or disable the use of these properties for HTTP exit points:

Configurable Properties	Default Detection and Naming Property?		Description
	.NET Agent for Windows	.NET Agent for Linux	
`Host`	No	Yes	HTTP host
`Port`	No	Yes	HTTP port number
`URL`	Yes	No	Full URL
`Query String`	No	Yes	HTTP parameters/query string

As of 4.5.13, you can customize HTTP backend detection and naming for the .NET Agent for Linux through the Controller UI. Customizing HTTP backend detection and naming is a feature preview that is recommended for pre-production systems. See Enable Preview Features.

Message Queue Backends

By default, AppDynamics automatically detects and identifies many message queue exit points. For a list of the supported message-oriented middleware products, see Supported Remote Service Detection for .NET.

The default queue automatic discovery rule uses the destination property to name the message queue backend.

For example: `HowdyWorldQueue$\HWT_MQ_Server1_MsgQ`

Message Queue Configurable Properties

This table list the properties used for queue exit points. However, each message-oriented product is different and there may be variations in the properties, or their names.

Configurable Properties	Default Detection and Naming Property?	Description
`Host`	No	Queue server name
`Destination`	Yes	Name of topic or queue
`Destination Type`	No	Queue or topic
`Vendor`	No	Vendor from the client library

For specific queue types, see:

- MSMQ Backends for .NET
- NServiceBus Backends for .NET
- RabbitMQ Backends for .NET

MongoDB Backends

By default, the .NET Agent detects MongoDB exit calls for create, read, update, and delete (CRUD) operations using the C# and .NET MongoDB Driver versions: 1.10, 2.0, 2.2, and 2.4.

The default MongoDB automatic discovery rule uses the host, port, and database name to name the MongoDB backend.

For example: `mymongohost.27017.mymongodb`

MongoDB Configurable Properties

You can configure the use of these properties for MongoDB exit points:

Configurable Properties	Default Detection and Naming Property?	Description
`Host`	Yes	MongoDB host
`Port`	Yes	MongoDB port
`Database`	Yes	MongoDB database name

.NET Remoting

When an application uses .NET remoting, AppDynamics automatically detects and identifies remoting exit points (backends). See Enable Correlation for .NET Remoting to configure downstream correlation.

The default remoting automatic discovery rule uses the URL property.

For example: `tcp://remoting.example.com:8648/MovieTicketBooking`

.NET Remoting Configurable Properties

You can configure the use of this property for .NET Remoting exit points:

Configurable Properties	Default Detection and Naming Property?	Description
`URL`	Yes	Full URL

WCF Backends

AppDynamics automatically detects and identifies WCF exit points (backends) when an application uses the WCF client library. The default WCF automatic discovery rule uses the remote address property. The agent uses the enabled properties to derive a display name using the remote address.

For example: `http://wcf.example.com:8205/Services/Service1.svc`

Agent Compatibility

This table describes the WCF backend detection support for each variant of the .NET Agent:

Agent and Version	WCF Backend Detection Support
.NET Agent for Windows (.NET Framework)	Fully supported
.NET Agent for Linux >= 20.7.0	Partially supported • Works with .NET Core >= 3.1 • Compatible with asynchronous calls over HTTP • Does not support the `Operation Contract`, `Service Contract`, and `SOAP Action` properties.
.NET Agent for Windows (.NET Core) >= 21.3.0	

WCF Configurable Properties

You can enable or disable the use of these properties for WCF exit points:

Configurable Properties	Default Detection and Naming Property?	Description
`Remote Address`	Yes	URL minus the query, fragment, and user information (name and password)
`Operation Contract`	No	WCF operation contract name
`Service Contract`	No	WCF service contract name
`URL`	No	Full URL
`Host`	No	Host portion of URL
`Port`	No	Port number if present in the URL; otherwise, protocol default
`SOAP Action`	No	For web service calls, the SOAP action

.NET Web Services Backends

When an application uses the Microsoft Web Services client library, AppDynamics automatically detects and identifies web services exit points (backends) by default. The default web services automatic discovery rule uses the URL property. From the enabled properties, AppDynamics derives a display name using the URL.

For example: `http://webservice.example.com:8105/Services/Service1.asmx`

Web Services Configurable Properties

You can enable or disable the use of these properties for Web Services exit points:

Configurable Properties	Default Detection and Naming Property?	Description
`Service`	No	Web service name
`URL`	Yes	Full URL
`Operation`	No	Web service operation name
`Soap Action`	No	SOAP action

Custom Exit Points

To monitor a backend that is not included in .NET Supported Environments, you can configure a custom exit point.

By default, you have to restart the instrumented application for instrumentation changes to take effect. You can enable Runtime Reinstrumentation for the .NET Agent to avoid having to restart your application and the `AppDynamics.Agent.Coordinator` after instrumentation changes.

Asynchronous Exit Points for .NET

Related pages:

- .NET Backend Detection
- Using Asynchronous Methods in ASP.NET 4.5

Asynchronous programming patterns can enhance the scalability and performance of applications. Microsoft .NET lets you designate methods as asynchronous tasks.

The .NET runtime releases resources for asynchronous methods while tasks complete. When task processing finishes, the runtime calls back to the originating asynchronous method so the method may continue processing.

The .NET Agent detects certain asynchronous programming patterns as exit points. Because tasks may execute in parallel, AppDynamics sometimes represents asynchronous activity differently from synchronous activity in the Controller.

AppDynamics differentiates .NET async backend tracking from thread correlation. To configure thread correlation, see Thread Correlation for .NET.

Supported Asynchronous Exit Point Patterns

The agent discovers exit points for Microsoft .NET 4.5 *async* and *await* keywords. See Asynchronous Programming with Async and Await.

The agent detects asynchronous exit points for the following backend types:

- ADO.NET
- Azure Blob Storage, Azure File Storage, and Azure Table Storage
- CosmosDB
- HTTP
- MongoDB
- WCF
- Web Service

> Backend types are different for the .NET Core microservices agent. See .NET Core Microservices Agent Support.

Flow maps in the Controller UI represent asynchronous exit points as dotted lines labeled "async". Because calls in an asynchronous transaction may execute simultaneously, the Controller doesn't display end-to-end percentage times for individual asynchronous calls.

Troubleshoot Asynchronous Calls in Transaction Snapshots

Transaction snapshots include several features to help you discover problem areas in business transactions that use asynchronous methods. For an overview of transaction snapshots, see Transaction Snapshots.

Transaction Snapshot Flow Map

The Transaction Snapshot Flow Map graphically represents the business transaction. It displays the user experience, execution time, and timestamp of the transaction. The flow map also provides details of the overall time that is spent in a particular tier and in database and remote service calls. The *async* label indicates asynchronous calls.

Snapshot Execution Waterfall View

The transaction Snapshot Execution Waterfall View shows a timeline representation of the end-to-end execution of the business transaction. Synchronous and asynchronous processes appear on a bar diagram illustrating their relative execution time arranged in chronological order. Synchronous calls are represented by solid line arrows, and asynchronous calls are represented by dotted line arrows.

The waterfall view enables you to visually identify which processes are running the longest. Double-click a bar to drill down to a call graph and investigate problematic code.

Analyze Asynchronous Activity in the Metric Browser

The Metric Browser displays asynchronous activity in the following places:

- **Business Transaction Performance > Business Transactions > tier > business transaction > Thread Tasks > Asynchronous Operation > External Calls**
- **Overall Application Performance > *tier* > Thread Tasks > Asynchronous Operation > External Calls**
- **Overall Application Performance > *tier* > Individual Nodes > node name > Thread Tasks > Asynchronous Operation > External Calls**

MSMQ Backends for .NET

Related pages:

- .NET Backend Detection
- App Agent Node Properties

The AppDynamics .NET Agent (agent) automatically detects MSMQ backends when an instrumented tier makes a call to MSMQ. MSMQ exit points are methods that publish or push messages to a queue. If you are using NServiceBus over the MSMQ transport, see NServiceBus Backends for .NET.

MSMQ entry points are methods that listen or poll for new messages in the queue. Before the agent can detect entry points or perform downstream correlation for MSMQ you must define the correlation field and, for multithreaded architectures, specify threading architecture.

Exit Points and Backend Naming

The agent names the queue for the queue name returned from the system.

Entry Points

To enable downstream correlation for MSMQ, you must configure the agent.

Define the MSMQ Correlation Field

Register the *msmq-correlation-field* app agent node property on both the publishing tier and on the receiving tier. Specify the field where the agent will write and read correlation information.

The agent supports the following fields:

- Extension, the default field
- Label

By default, the .NET Agent writes to the Extension field. However, some frameworks built on MSMQ write data to the Extension field. In this case, the Extension field does not work because it is already in use, so you must configure the .NET Agent to write correlation data to the Label field.

> Choose a field not in use by your queue implementation. If your implementation uses both the "Label" and "Extension" fields, downstream correlation is not currently possible.

See "msmq-correlation-field" on App Agent Node Properties Reference. For instructions to register a node property, see App Agent Node Properties.

Specify the threading architecture

If your queue implementation uses a multithreaded architecture, you must register the *msmq-single-threaded* app agent node property to specify the threading architecture. Set msmq-single-threaded to "False" for multithreaded implementations of MSMQ. See "msmq-single-threaded" on App Agent Node Properties Reference. For instructions to register a node property, see App Agent Node Properties.

The threading architecture dictates how the agent calculates the call timing for the message queue:

- For single-threaded queues, the agent calculates the call time between receive requests. The call time begins at the end of the first receive method and ends when the next receive method starts.
  In the following example, the length of the `ProcessMessage()` method:

```
MessageQueue messageQueue;
for(;;)
{
        var message = messageQueue.Receive();
        ProcessMessage(message);
}
```

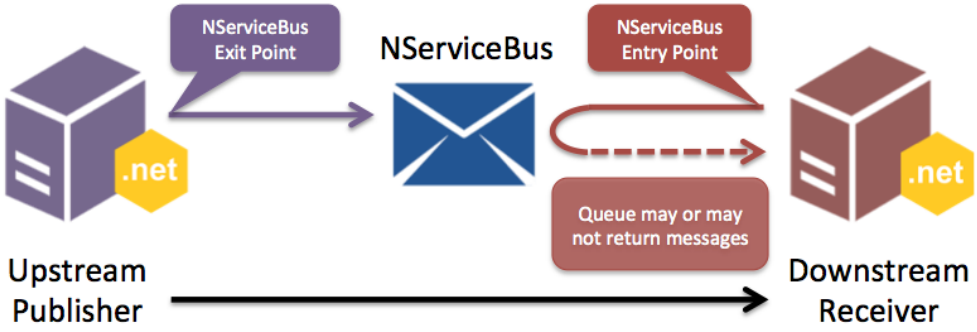
- For multithreaded queues, the agent does not capture timing.

NServiceBus Backends for .NET

Related pages:

- .NET Backend Detection

The AppDynamics .NET Agent automatically discovers exit points to NServiceBus backends when an instrumented tier makes a call to NServiceBus over MSMQ or RabbitMQ transports. The agent discovers NServiceBus entry points for downstream correlation from NServiceBus version 5.



Exit Points and Backend Naming

NServiceBus exit points are methods that publish or push messages to a queue from an upstream tier. The agent detects exit points regardless of the NServiceBus version or threading architecture. The agent names the queue for the queue name returned from the system.

Entry Points in a Single-threaded Architecture

NServiceBus entry points are methods that listen or poll for new messages in the queue. When the message receiver uses a single-threaded architecture, the .NET Agent automatically discovers entry points and correlates downstream activity without additional configuration.

The agent measures the call time from the end of the first receive method to the start of the subsequent receive method. In the following example loop, the length of the `ProcessMessage()` method:

```
MessageQueue messageQueue;
for(;;)
{
        // Call timing for previous iteration ends.
        var message = messageQueue.Receive();
        // Call timing begins.
        ProcessMessage(message);
}
```

Entry Points in a Multithreaded Architecture

For the .NET Agent to automatically detect NServiceBus entry points when the receiver is running a multithreaded architecture, set the *nservicebus-single-threaded* app agent node property to "false" on the receiver. See "nservicebus-single-threaded" on App Agent Node Properties Reference. For instructions to register a node property, see App Agent Node Properties.

For multi-threaded message receivers, the agent does not capture timing.

Resolve Issues with NServiceBus Backends

- If you have NServiceBus version 5 and are not seeing downstream correlation on the receiver, verify the threading architecture on the receiver. Set the *nservicebus-single-threaded* app agent node property to "false" on the multithreaded receivers.
- Only use the node properties for MSMQ, msmq-single-threaded and msmq-correlation-field, if you are using MSMQ independently of NServiceBus. The .NET Agent does not use MSMQ node properties when discovering and instrumenting NServiceBus over MSMQ transport.
- If you are using NServiceBus version 4, you can disable NServiceBus instrumentation on both the publisher and the receiver:
    - Copy the following Instrumentation element to a child of the Machine Agent element:

```
<!--Disable NServiceBus instrumentation-->
   <instrumentation>
     <instrumentor name="NServiceBusReceiveContextInstrumentor" enabled="false" />
     <instrumentor name="NServiceBusPublishExitInstrumentor" enabled="false" />
     <instrumentor name="NServiceBusEntryInstrumentor" enabled="false" />
     <instrumentor name="NServiceBusExitInstrumentor" enabled="false" />
   </instrumentation>
```

For more information, see "Instrumentation Element" on .NET Agent Configuration Properties.

- Configure the publisher and receiver as you would for MSMQ or RabbitMQ.

RabbitMQ Backends for .NET

Related pages:

- .NET Backend Detection
- RabbitMQ Monitoring Extension

The AppDynamics .NET Agent automatically detects RabbitMQ backends based upon calls from instrumented tiers. RabbitMQ exit points are methods that publish or push messages to a queue. RabbitMQ entry points are methods that listen or poll for new messages in the queue.

> .NET extends support for `BasicPublish` and `BasicConsume` in the RabbitMQ client >= 6.0.0.

If you are using NServiceBus over the RabbitMQ transport, see NServiceBus Backends for .NET.

Exit Points and Backend Naming

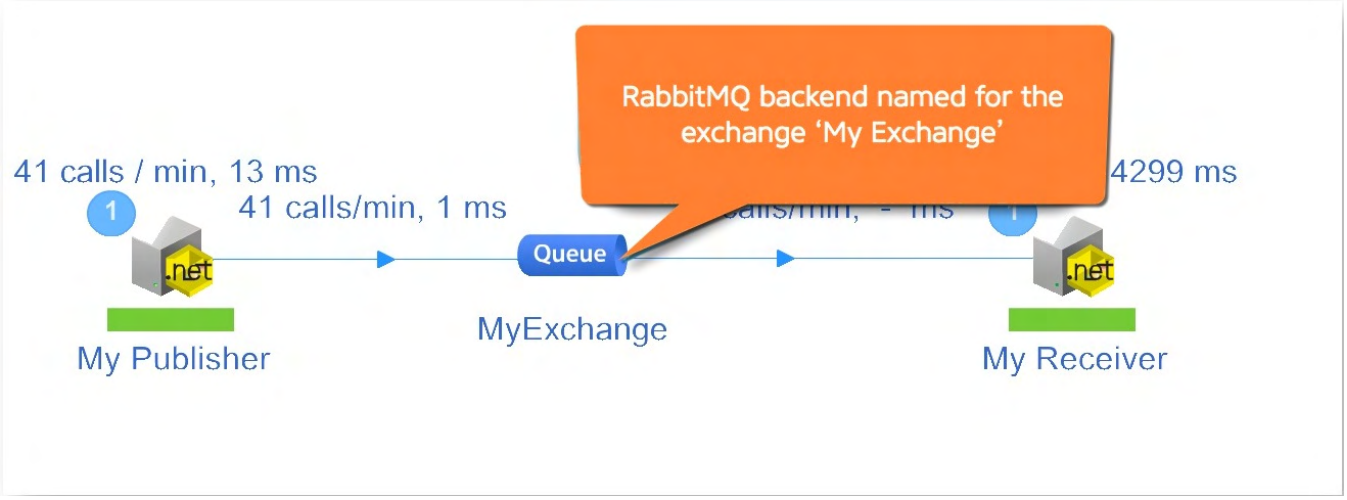
The agent discovers a RabbitMQ backend exit point when your application sends a message to the queue using the `BasicPublish()` method.

By default, the agent names the RabbitMQ backend for the exchange parameter of the `BasicPublish()` method.

For example:

```
model.BasicPublish("MyExchange", "", false, false,
    basicProperties, Encoding.UTF8.GetBytes(message));
```

In this case the agent names the queue **MyExchange**.



You can refine the backend name to include some or all segments of the routing key. To configure RabbitMQ naming you must be familiar with your implementation RabbitMQ exchanges and routing keys. See RabbitMQ Exchanges and Exchange Types.

Refine backend naming

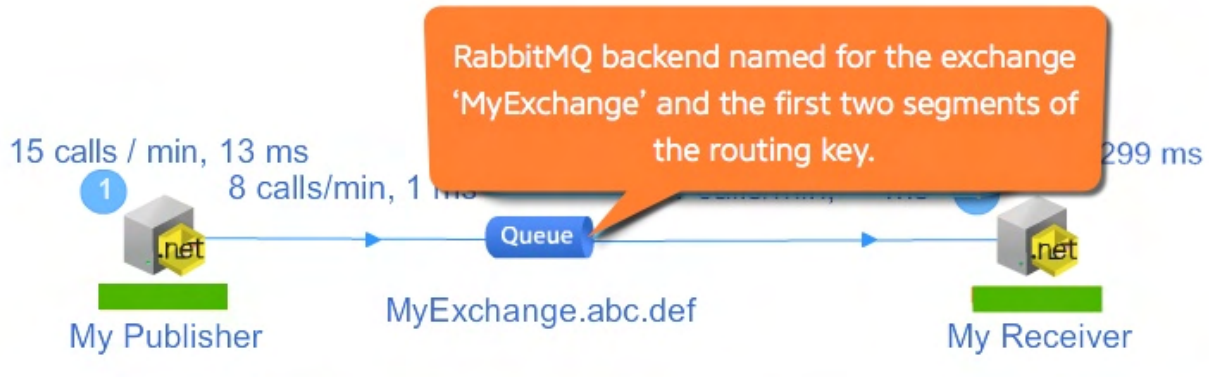
Register the *rmqsegments* node property. For instructions on how to set a node property, see App Agent Node Properties.

Name: rmqsegments
Description: "Configure RabbitMQ naming to include routing key segments."
Type: Integer
Value: <integer>
The routing key is a string. The agent treats dot-separated (".") substrings of the routing key as segments. Set the value to an integer that represents the number of routing key segments to include in the name.

In the following example the routing key is "abc.def.ghi". Set the rmqsegments value to "2" to name the queue "MyExchange.abc.def".

```
model.BasicPublish("MyExchange", "abc.def.ghi", false, false,
    basicProperties, Encoding.UTF8.GetBytes(message));
```

After you save the node property, the Controller sends the configuration to the agent. After some time the RabbitMQ backend shows up with the new name.



Entry Points

The agent discovers RabbitMQ backend entry point when your application polls the queue. AppDynamics auto-detects RabbitMQ based upon the following patterns:

- BasicGet
- HandleBasicDeliver

BasicGet Method

The agent detects the pattern below where the application periodically polls the message queue using the `BasicGet()` method. The call timing is limited to the time spent inside the `while` loop. The timer only starts when the `BasicGet` result returns a value at line 4. The timer ends when the next `BasicGet` executes. In this example, the application polls every five seconds, so the execution time equals the time in the `if` loop plus five seconds.

RabbitMQ Client before version 6.x:

```
while (true)
{
	var result = channel.BasicGet("MyExchange", true);
	if (result != null)
	{
		var body = result.Body;
		var message = Encoding.UTF8.GetString(body);
		Console.WriteLine("Received: {0}.", message);
	}

	Thread.Sleep(5000);
}
```

RabbitMQ Client version 6.x:

```
while (true)
{
	var result = channel.BasicGet("MyExchange", true);
	if (result != null)
	{
		var body = result.Body.ToArray()
		var message = Encoding.UTF8.GetString(body);
		Console.WriteLine("Received: {0}.", message);
	}

	Thread.Sleep(5000);
}
```

HandleBasicDeliver Method

The agent detects the `HandleBasicDeliver()` method for custom implementations of the `IBasicConsumer` interface. In this case, the call timing reflects the execution time for the `HandleBasicDeliver` method.

Correlation Over Microsoft BizTalk

When you install the .NET Agent on a Microsoft BizTalk Server, the agent automatically discovers BizTalk entry points and exit points so you can monitor your BizTalk integration.

Requirements

- Microsoft BizTalk 2010, 2013
- BizTalk must be a downstream tier from an originating tier for a business transaction. The agent doesn't discover BizTalk as the originating tier of a transaction.

Entry Points

The .NET Agent detects incoming BizTalk requests using the WCF Send Adapter or the SOAP Send Adapter.

Instrument the BizTalk Service

1. Install the .NET Agent on the BizTalk server. See Install the .NET Agent for Windows.
2. Register the "enable-soap-header-correlation" app agent node property with a value of "true" for the nodes upstream from the BizTalk tier. See App Agent Node Properties.
3. Instrument the BizTalk service executables: BTSNTSvc.exe and BTSNTSvc64.exe. See Configure the .NET Agent for Windows Services and Standalone Applications. For instance:

   ```
   <standalone-applications>
     <standalone-application executable="BTSNTSvc">
       <tier name="BizTalk Service"/>
     </standalone-application>
   </standalone-applications>
   ```

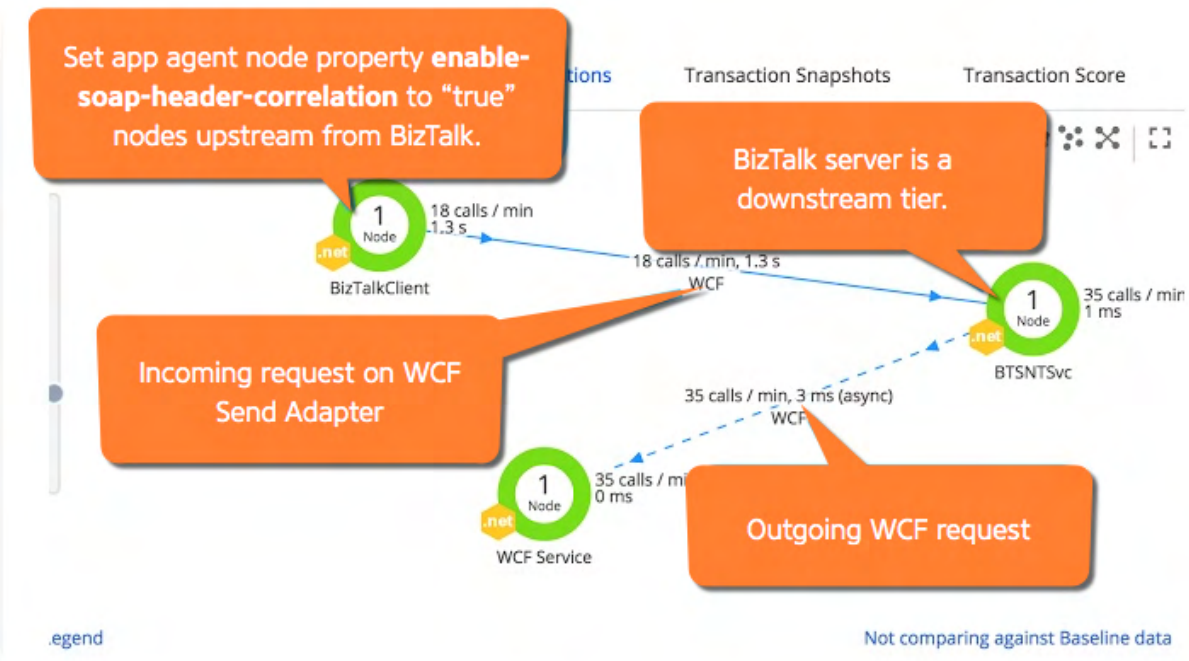
   If you run multiple instances of BizTalk on the same Windows server, you can assign the different instances to unique nodes using command line options. For example, to differentiate nodes within the same tier, specify the BizTalk Service command "name" parameter in the standalone application `command-line` attribute:

   ```
   <standalone-applications>
     <standalone-application executable="BTSNTSvc" command-line="-name &amp;quot;BizTalk1&amp;quot;">
       <tier name="BizTalk Service"/>
       <node name="BizTalk1"/>
     </standalone-application>
     <standalone-application executable="BTSNTSvc" command-line="-name &amp;quot;BizTalk2&amp;quot;">
       <tier name="BizTalk Service"/>
       <node name="BizTalk2"/>
     </standalone-application>
   </standalone-applications>
   ```

   Notice the use of quotes to enclose the name command-line parameter.
4. Optionally enable BizTalk performance counters. For more information, see Manage Windows Performance Metrics. Refer to the Microsoft BizTalk Server documentation for descriptions of individual performance counters.

```
<machine-agent>
  <!-- BizTalk Performance Counters -->
  <perf-counters>
    <perf-counter cat="BizTalk:Message Agent" name="High database session" instance="*"/>
        <perf-counter cat="BizTalk:Message Agent" name="High database size" instance="*"/>
        <perf-counter cat="BizTalk:Message Agent" name="High in-process message count" instance="*"/>
        <perf-counter cat="BizTalk:Message Agent" name="High message delivery rate" instance="*"/>
        <perf-counter cat="BizTalk:Message Agent" name="High message publishing rate" instance="*"/>
        <perf-counter cat="BizTalk:Message Agent" name="High process memory" instance="*"/>
        <perf-counter cat="BizTalk:Message Agent" name="High system memory" instance="*"/>
        <perf-counter cat="BizTalk:Message Agent" name="High thread count" instance="*"/>
        <perf-counter cat="BizTalk:Message Agent" name="Message delivery delay (ms)" instance="*"/>
        <perf-counter cat="BizTalk:Message Agent" name="Message delivery throttling state" instance="*"/>
        <perf-counter cat="BizTalk:Message Agent" name="Message publishing delay (ms)" instance="*"/>
        <perf-counter cat="BizTalk:Message Agent" name="Message publishing throttling state" instance="*"
/>
        <perf-counter cat="BizTalk:Message Agent" name="Message delivery incoming rate" instance="*"/>
        <perf-counter cat="BizTalk:Message Agent" name="Message delivery outgoing rate" instance="*"/>
        <perf-counter cat="BizTalk:Message Agent" name="Message publishing incoming rate" instance="*"/>
        <perf-counter cat="BizTalk:Message Agent" name="Message publishing outgoing rate" instance="*"/>
        <perf-counter cat="BizTalk:Message Agent" name="Message delivery throttling state duration"
instance="*"/>
        <perf-counter cat="BizTalk:Message Agent" name="Message delivery throttling user override"
instance="*"/>
        <perf-counter cat="BizTalk:FILE Receive Adapter" name="Bytes received/Sec" instance="*"/>
        <perf-counter cat="BizTalk:FILE Receive Adapter" name="Lock failures/sec" instance="*"/>
        <perf-counter cat="BizTalk:FILE Receive Adapter" name="Messages received/Sec" instance="*"/>
        <perf-counter cat="BizTalk:FILE Send Adapter" name="Bytes sent/Sec" instance="*"/>
        <perf-counter cat="BizTalk:FILE Send Adapter" name="Messages sent/Sec" instance="*"/>
        <perf-counter cat="BizTalk:SOAP Receive Adapter" name="Messages received/Sec" instance="*"/>
        <perf-counter cat="BizTalk:SOAP Send Adapter" name="Messages sent/Sec" instance="*"/>
  </perf-counters>
</machine-agent>
```

Once you've completed instrumentation, the .NET Agent correlates traffic through your BizTalk server:



If you enabled Performance Counter metrics, they appear under the Custom Metrics tree in the Metric Browser.

Apache Web Server Backend Detection

The Apache Agent discovers Apache Modules as backends. To review general information about monitoring backends, see Backend Detection Rules.

Apache Modules

The Apache Agent automatically detects loaded Apache modules as remote services:

- Modules must be in the handler stage.
- The agent excludes a list of common modules from discovery, see "Remote Service Detection" on Supported Apache Web Servers.

By default, the Apache Agent includes metrics for module backends with the downstream backend or tier. The Controller doesn't display the module on flow maps. For example, if Apache calls mod_jk.c, the module backend doesn't show on flow maps and metrics are included with the downstream Tomcat backend or tier.

To view Apache modules on the flow map and to see separate metrics for Apache modules, set *AppDynamicsResolveBackends* to "OFF" in the appdynamics_agent.conf file. See Install the Apache Agent.

The agent names the Apache modules backend for the module name and the server: <module>-<host>:<port> . For example, mod_jk.c-myapache.example.com:80.

Previously, the backend name for requests routed through the reverse proxy was `mod_proxy.c-<ip and port of server >`. It is now resolved to the actual URL where the request is transferred to.

To add end segments that are displayed in the backend naming, set `AppDynamicsBackendNameSegments` with the required end segments. See Install the Apache Agent.

HTTP Backend Detection

Related pages:

- Java Backend Detection
- .NET Backend Detection

AppDynamics automatically detects HTTP backends when outbound calls are made through supported HTTP clients, and by default names them according to their host and port. If the default configuration results in names that are not meaningful for your application (such as EC2 host names, file paths, and ports in the name) you can change the default discovery rule.

Additionally, if an HTTP call is destined for an http router which could route the request to one of several instrumented downstream tiers then the default naming rules will need to be modified such that the name includes enough segments of the target URL such that each destination tier is associated with a differently named backend.

HTTP Backend Naming

To ensure that the HTTP backends detected in your application have meaningful names, and that Business Transaction correlation functions correctly, your custom configuration will need to take into account the specific format used in your environment.

The format can vary even within a single environment. For example, some backend systems may have hostnames prefixed with ec2storage, which may not be meaningful in naming, while others may use hostnames such as salesforce.com, which may be meaningful.

To account for different formats, you should create a custom rule rather than changing the automatic discovery rule. This lets you apply different rules for different URL formats.

You can then apply a specialized approach for each case, as in the following examples:

- For format "ec2storage/servicename", you would use the URL
- For format "salesforce.com", use the host name
- For the other backends, you may use a query string
- For calls that could hit one of several downstream tiers via an HTTP router, you should use sufficient segments of the target URL in the name such that each downstream tier that could process the request is associated with a differently named backend

In some cases, your HTTP backend discovery configuration might consist of a combination of the default rule and custom rules. The following section walks you through a specific example:

Use the URL Path in a Detection Rule

For example, when all the HTTP backends for a tier or application have a similar format, such as a prefix like "ec2storage", you can generate the right name and the correct number of backends to monitor by editing the automatic discovery rule. This enables you to monitor the KPIs that are interested in.

Consider an application with the following HTTP URLs:

```
http://ec2-17:5400/service1
http://ec2-17:5450/service2
http://ec2-18:5400/service1
http://ec2-18:5450/service2
```

In this case, measuring performance based on host name would not be useful, since the IP addresses are transient and all performance numbers would be irrelevant after the IP addresses recycle. Instead, you can monitor by service name by avoiding the use of Host and Port properties and using only the URL property, as follows:

1. Edit the **Automatic Backend Discovery** rule for HTTP for your agent type. See Backend Detection Rules for details on accessing this pane.
2. First, select and disable the use of **Host** and **Port**.
3. Then select and enable the property you want to use to uniquely identify the backend. In this case, select **URL** and check **Use URL in the Backend Name**.
4. For the field **How will URL be used in the Backend name?**, select **Use a segment of it**.
5. From the segment options dropdown, select **Use the first N Segments**.

   The important URL segment is for the "create" service, so your configuration is the same as in the previous screenshot.
6. Enter "/" slash for the **Split Delimeter**.
   Use a similar technique to strip out some segment of the URL, such as a user name as in the following URLs:

   ```
   [http://host:34/create/username1]
   [http://host:34/create/username2]
   ```

7. Once you change the configuration, delete all HTTP backends. When the agent rediscovers the backends with the new configuration, the flow map shows only the service backends.

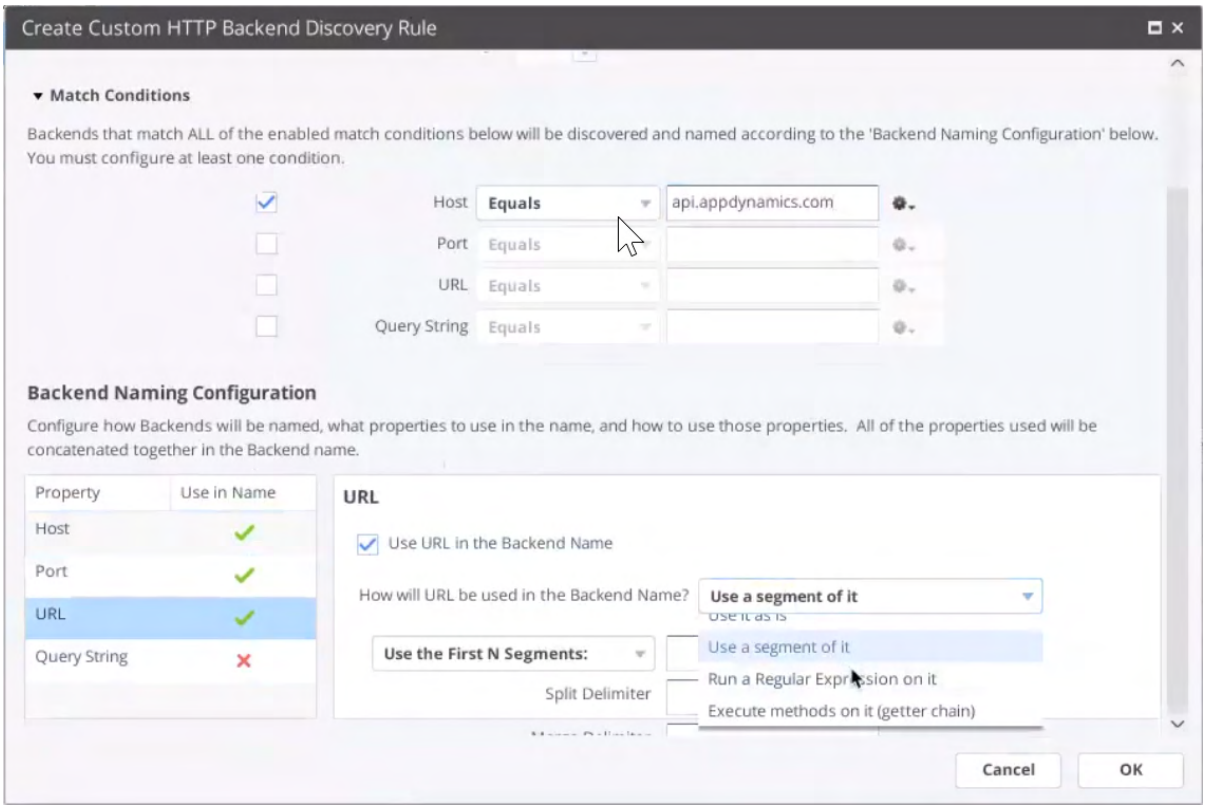
Custom HTTP Backend Detection and Naming Configuration

You can now customize HTTP backend detection and naming for the .NET Agent for Linux through the Controller UI.
Customizing HTTP backend detection and naming is a feature preview in the .NET Agent for Linux 4.5.13, recommended for pre-production systems. See Enable Preview Features.

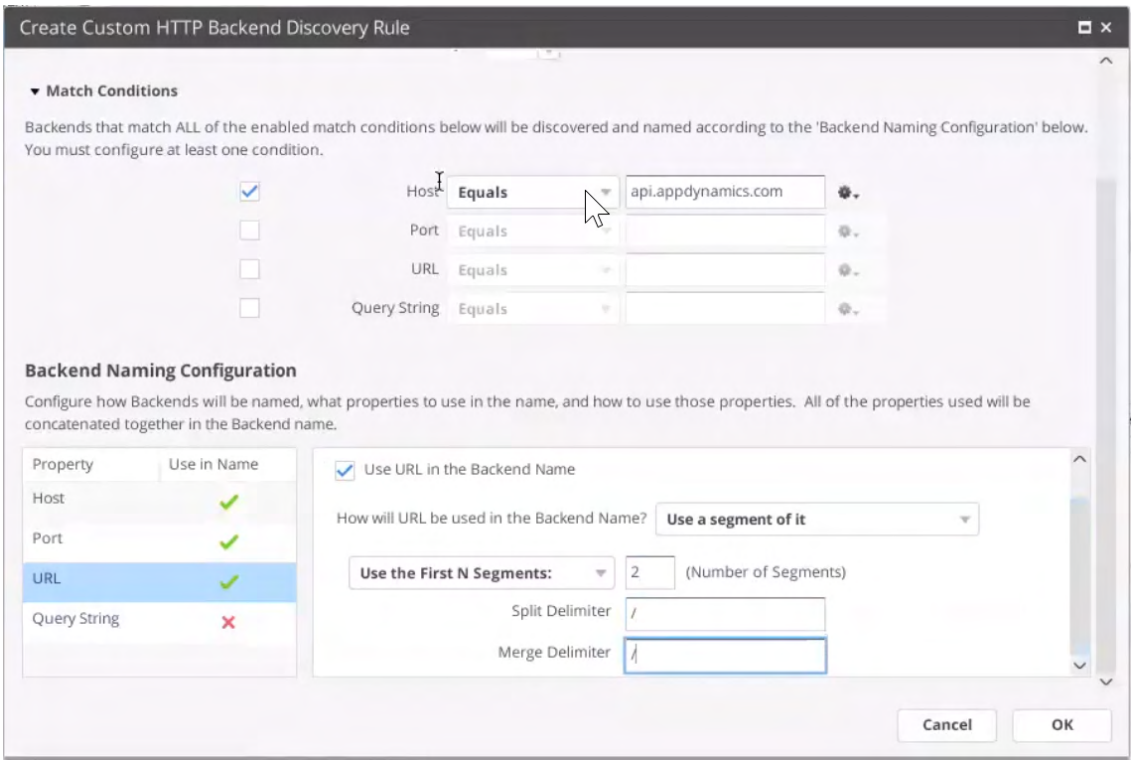
By default, AppDynamics uses the host name and port number to detect and name HTTP backends. This may be suitable in many cases, however, when your microservices, containers or serverlets are managed by an API gateway or portal, backend services communicating with a client or with each other cannot be correctly mapped to the proper tiers unless you edit automatic discovery to use a portion of the query string or use HTTP custom discovery rules to uniquely name them. AppDynamics recommends you create a custom HTTP backend discovery rule in this case.

The following screenshots provide an example configuration.

Here we choose Match Conditions to describe the API gateway host, in this case, api.appdynamics.com.



This screenshot shows where we choose to use the First 3 segments, using '/' as both the split and merge delimiters.



Configuring the HTTP Backend Discovery Rule as above detects backend metrics and reports them under a business transaction name that begins with api.appdynamics.com followed by the first three segments of the URL of the transaction, such as api.appdynamics.com/api/catalog or api.appdynamics/api/payment. Transactions matching the configured HTTP discovery rule are recognized as individual backends even when they share the same host name and port number. These transactions can be mapped to different tiers so you can analyze the metrics of the transactions that occur behind the API gateway.

There is a known issue in HTTP backend detection configuration in the current preview. When defining HTTP backend using URL segments, segment enumeration starts with number 2 instead of number 1, and segment 1 always returns empty.
For example, to define unique backends using the first two segments of the URL, you would need to configure HTTP backend detection to use the first 3 segments or segment 2 and 3.

Service Proxy Overview

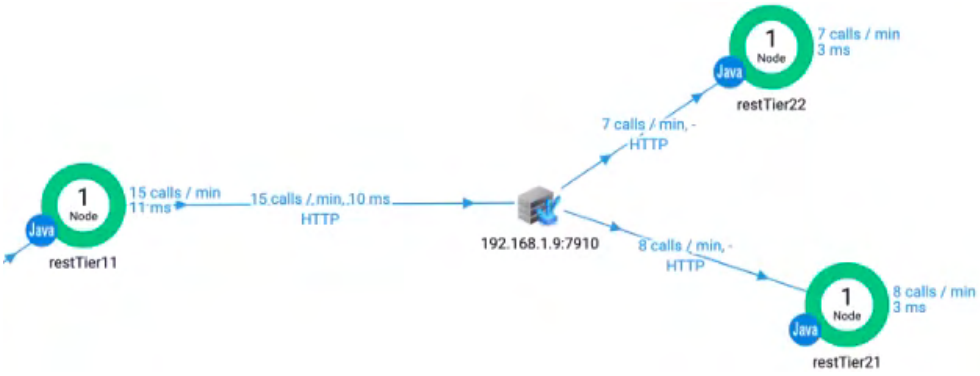
AppDynamics has identified an issue where Business Transactions have started reporting to all other traffic. Please see Support Advisory: Business Transactions Reporting to All Other Traffic before enabling Service Proxy Detection.

In Application Performance Management (APM), the service that is called in an application environment is usually identified by specific properties, such as the host or port properties. However in an environment with a gateway or service proxy, the host and port actually identify a *device* instead of the ultimate downstream tier that handles the request.

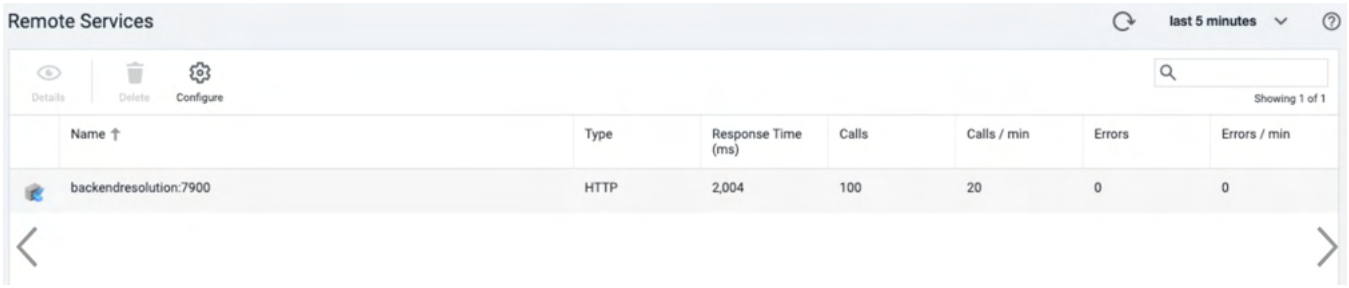
This inaccurate identification of the backend destination causes the system to periodically change the association of the single backend with one of the potential target downstream tiers to which the gateway could route. This results in the joining of incomplete flow maps and inconsistencies between tiers that are included within Business Transaction snapshots, and the corresponding Business Transaction flow map.

The service proxy identification feature enables the system to resolve and identify the correct backend and displays the correct component structure in the flow map. The service proxy detects these changes in the backend association and instructs the upstream agent to identify the affected backend(s) as a service proxy.

As a result, the flow map will show a service proxy (represented by this icon ) between the upstream and downstream tiers, for example:



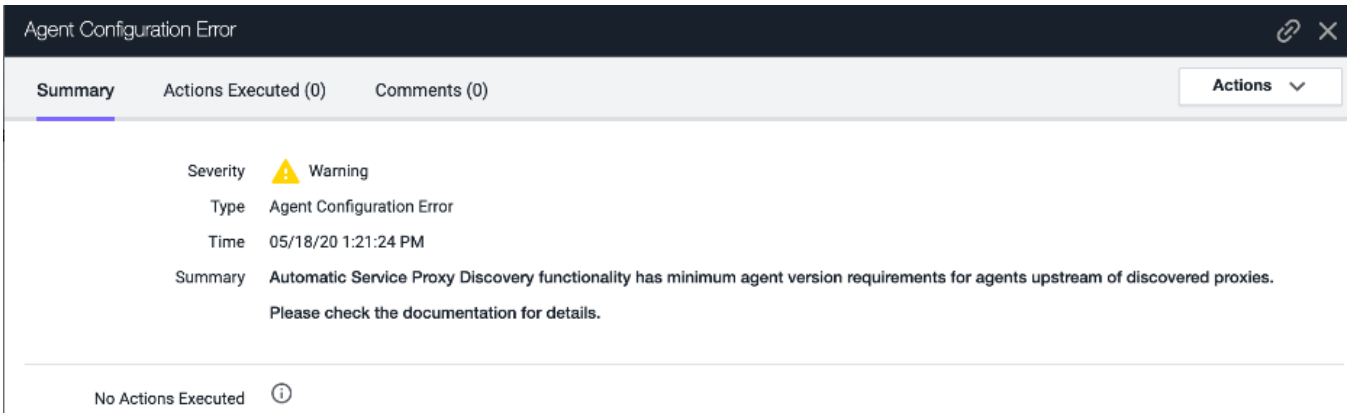
The service proxy is also visible in the **Remote Services** list:



If the upstream agent is an older version than what is currently supported, an event is generated with the following Agent Configuration Error message:

The service proxy feature is supported on these agents:

- Java Agent >= 20.5.0
- .NET Agent >= 21.3.0



The service proxy icon is also available on these flow maps:

- Application Dashboard
- Tiers and Nodes
- Drill Down Backend Node
- Transaction Snapshot (Waterfall View)

Enable Service Proxy Detection

The backend is identified as a service proxy in the Controller.

- For *SaaS customers*: To request that your Controller has the service proxy feature enabled, you should work closely with your AppDynamics account representative.
- For *on-premises customers*: To enable the service proxy feature:

  a. Access the Controller Administration Console.
  b. Open the admin.jsp page in the Controller from the url:
  <*controllerhost*>:<*controllerport*>/admin.jsp
  c. Access **Controller Settings**.
  d. Search for these two properties, and set each of them to `true`:
     - `backend.detect.loadbalancer.enabled = true`
     - `backend.detect.urimisconfig.enabled = true`
  e. Save your changes.

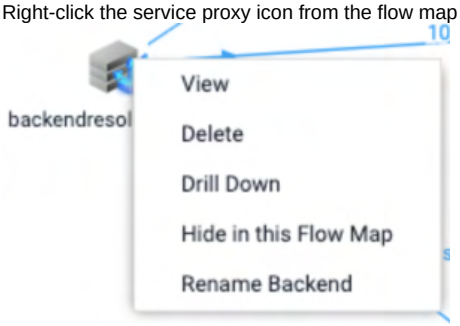
Once the Controller resolves the backend, it then sends the configuration information to the agent.

Delete a Service Proxy

If you remove the service proxy from the infrastructure but continue to use the same host and same port for the downstream nodes, then the service proxy will still show on the flow map. You must manually delete the service proxy to directly resolve the upstream and downstream tiers correctly.

To delete a service proxy for a backend node:

1. Right-click the service proxy icon from the flow map to open a popup menu.



2. Select **Delete**.

Error Detection

You can customize how AppDynamics associates code events with business transaction errors by adding or removing the methods, log messages, HTTP codes, or redirect pages that are configured to indicate business transaction errors.

Permissions

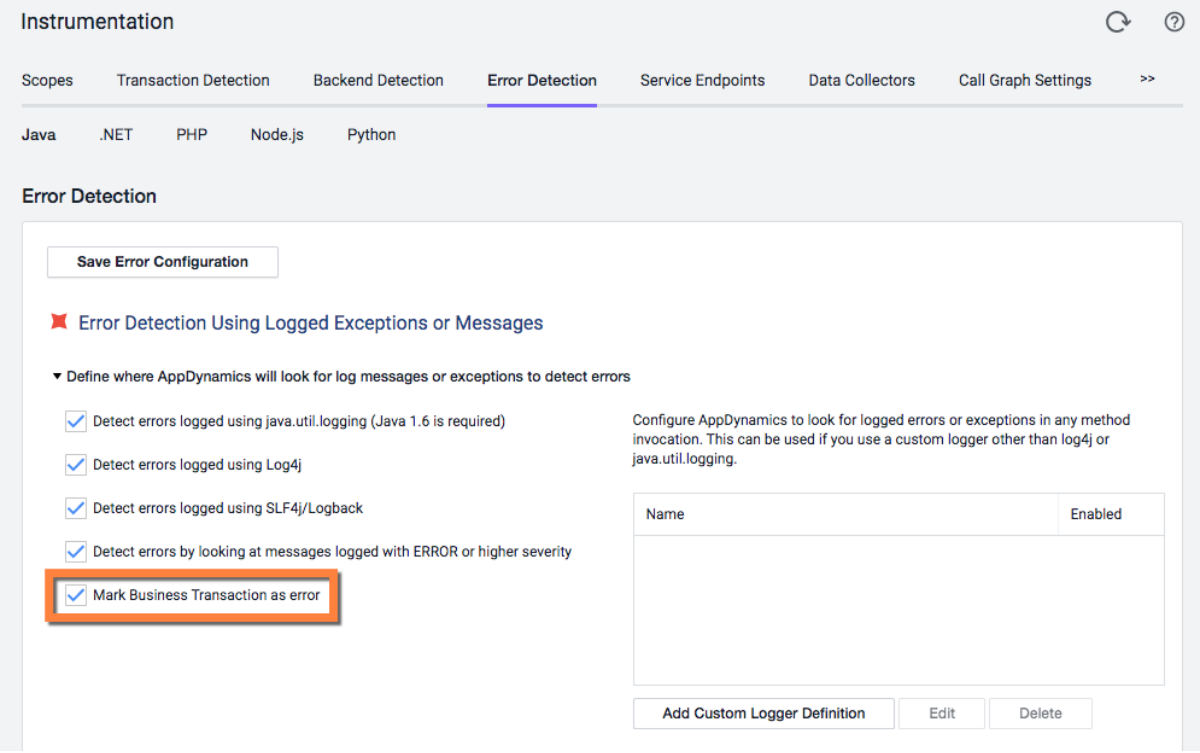
To configure error detection, you need the "Configure Error Detection" permission.

Error Detection Configuration

You can view or modify error detection configuration in the Error Detection tab. In the Controller UI, open an application and navigate to **Configuration** > **Instrumentation** > **Error Detection**.

The Error Detection tab takes you to the configurable options for each programming language or platform. You configure settings individually for each language or platform. Some settings are platform-specific. For example, custom loggers and error detection using redirect pages are available for Java and .NET only. Configuration changes made in the Error Detection tab apply to all business transactions for the selected application. You cannot make changes to individual transactions.

To prevent marking transactions as errors, uncheck the **Mark Business Transaction as error** for each applicable language or platform. If unchecked, the transactions are not added to the error count metric and are considered in other metrics, such as response time, despite the occurrence of the error.



Supplement Error Detection

For Java and .NET, you can supplement the methods that AppDynamics considers error methods from common error handling frameworks with your own, custom-defined error methods by defining a custom logger. When the application invokes the method in a business transaction, AppDynamics considers the transaction to be an error transaction for metric purposes.

In the custom logger configuration, you can specify the parameter that contains the exception or error information passed to the method. This information will appear as the error message in AppDynamics, which invokes `toString()` on the parameter to extract the message.

To configure a custom logger:

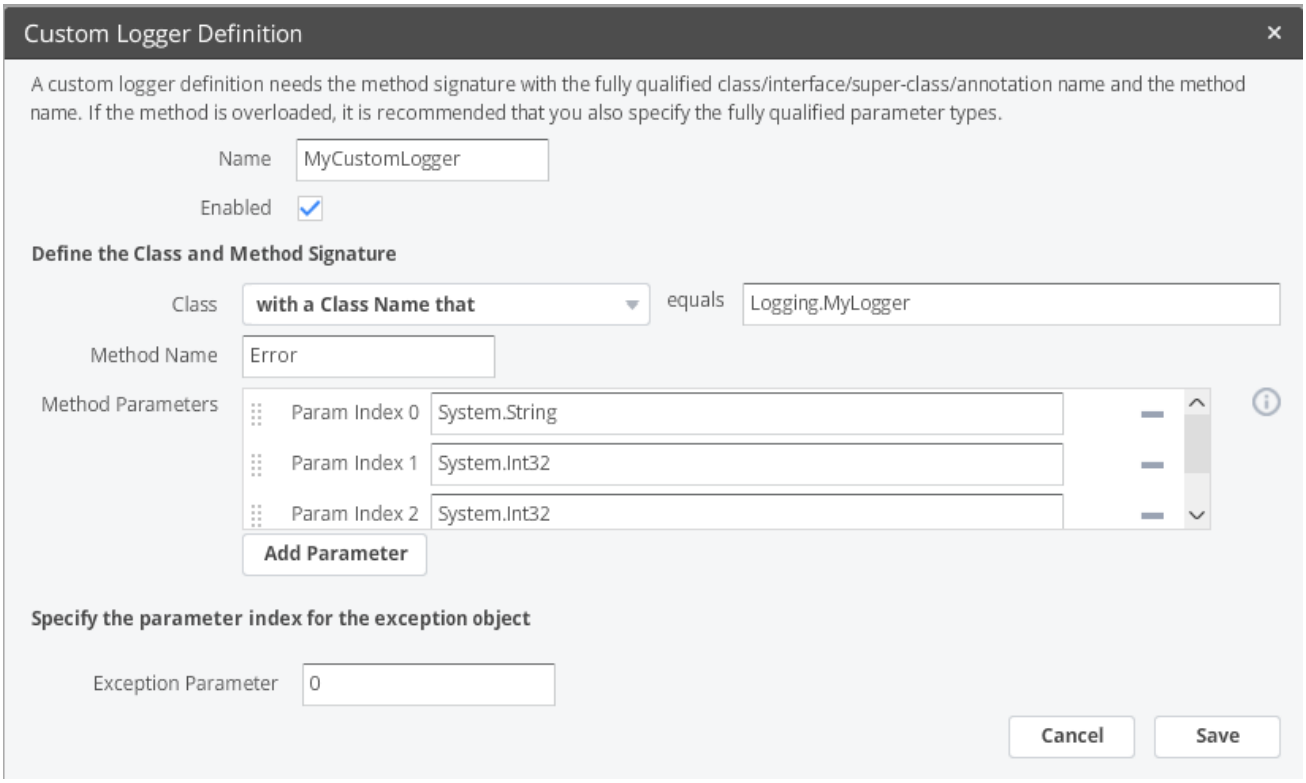
1. In the Controller UI, on the Error Detection configuration tab, click **Add Custom Logger Definition** in the error configuration panel.
2. Enter a descriptive name for the custom logger definition and use the settings to identify the class name and method for the custom logger. For more information on how to use the AppDynamics UI to identify classes and methods, see Data Collectors.
3. For the **Method Parameter** field, add a parameter definition for each parameter in the signature of the method.
   If the method is overloaded, create a logger definition for each form of the overloaded method where you want to detect errors. Your custom method must accept at least one parameter, which should be the parameter that conveys the logged error or exception information.
4. For the **Exception Parameter** field, identify the parameter in the method signature that contains the exception object by index number (0-based). This parameter, identified by an index, can be any type of object, including Arrays. If the object is not null, AppDynamics converts the object to a

string, the result of which constitutes the error details. A business transaction is considered to be an error only if a non-null exception object is passed to the logger method.

Consider the following custom logger class and methods in a .NET application:

- Namespace and class = Logging.MyLogger
- `logger.Error(string message, int param1)`
- `logger.Error(string message, int param1, int param2)`

Notice that the Error method is overloaded. To capture errors logged in the second form of the method using the first parameter as the error to log. This screenshot illustrates the configuration:



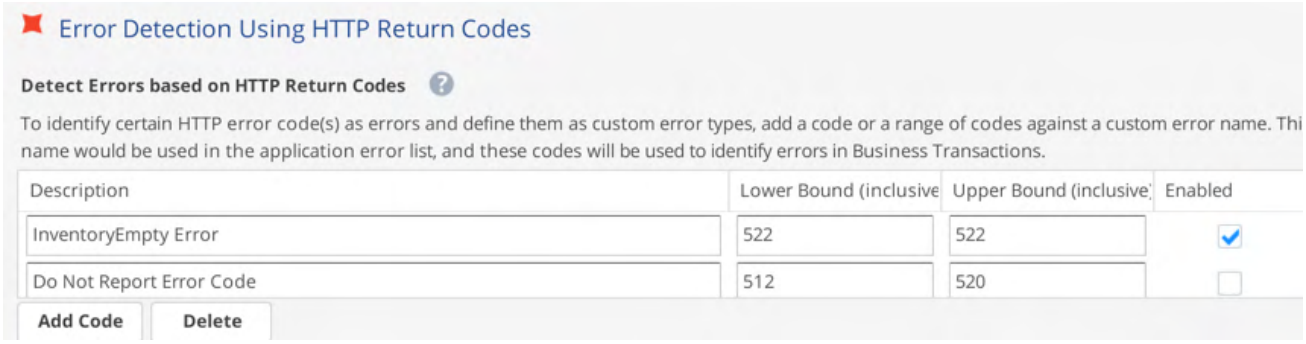
Notice that the exception parameter is set to 0, identifying the `string message` method parameter as the error message. When enabled, AppDynamics detects and reports the custom logger.

Detect HTTP Responses or Redirect Pages as Errors

In Java, .NET, Node.js, and Python you can configure errors based on HTTP response codes. If the error code is set as part of business transaction processing, the transaction is marked as an error. For PHP, it is possible to classify transactions as in error based on 4xx HTTP return codes using the PHP Agent Configuration Settings

By default, AppDynamics captures HTTP error codes from 400 to 505. HTTP response codes may convey errors that occur at the business level of an application. For example, in an e-commerce application, a 522 error might indicate that an item is out of stock. For this case, you may want to include the error as a default transaction error indicator.

To exclude a return code add them to the error detection list by creating a custom error code range and then disabling that error code by clearing its **Enabled** checkbox. This in effect excludes the error code as an error indicator.



For Java and .NET agents, you can specify custom redirect target error pages as error indicators. To specify a redirect page, click **Add Error Redirect Page** and add a name for the configuration and a regular expression to match the URL of the page, such as `AcmeErrorPage.jsp`.

Ignore Exceptions and Log Messages as Error Indicators

Certain types of exceptions, loggers or log messages as transaction error indicators may not reflect events that should be counted as transaction errors in your environment. They may include exceptions raised by application framework code or by user login failures.

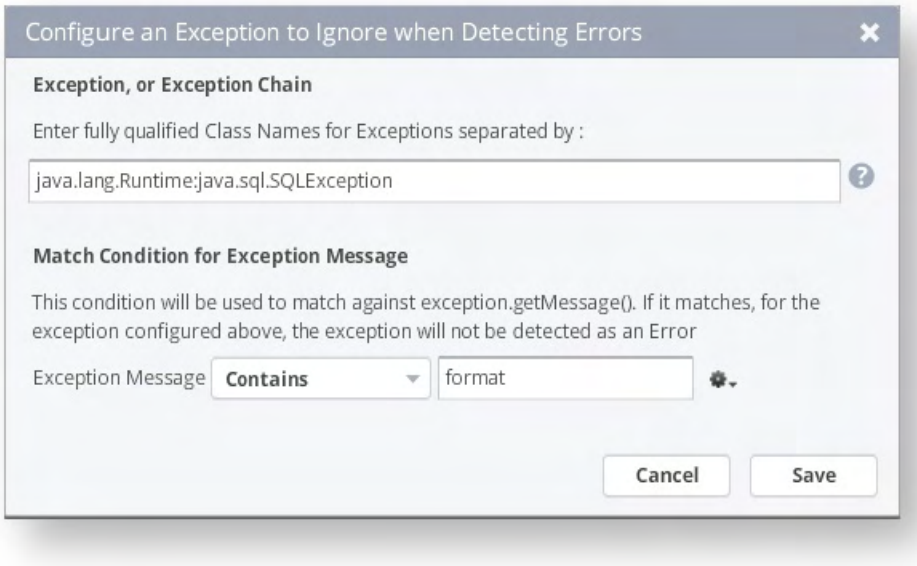
When you configure an exception to be ignored, the agent detects the exception, increments the exception count, and displays the exception in Exceptions lists in the UI, but the business transaction in which the error is thrown is not considered an "error transaction" for monitoring purposes. The transaction snapshot would not show the exception in the Summary or Error Details section and the user experience for the transaction instance would be unaffected by the exception.

To configure exceptions for the agent to ignore, click the **Add New Exception to Ignore** button in the error configuration window.

Enter the class name of the exception to be ignored and the match condition for the exception message. For the match condition, if you do not need to filter for specific messages in the exception, select "Is Not Empty". If you want to filter for Null, select "Is Not Empty" and use the gear icon to select NOT, which, in effect, tests for "is empty".

> The text field for **Match Condition for Exception Message** cannot contain the characters ", <, >, and &.

The following example directs the agent to ignore java.lang.RuntimeExceptions that wrap a javax.sql.SQLException only when the exception.getMessage() call in the root exception contains the string "format". (Other types of java.lang.Runtime exceptions will not be ignored.)



When you define the class of an exception to ignore by an exception chain, the exception message against which the match condition is applied must be in the root exception of the chain. The match is not applied to any nested exceptions.

In .NET and Java, you can specify that errors logged with certain loggers or logger categories be ignored. Click **Add New Category/Logger to Ignore**.

Error Detection Notes by Platform

The following sections list error detection configuration and error monitoring considerations, if any, by application platform.

Error Configuration for .NET

By default, the Windows .NET Agents can instrument calls using NLog and Log4Net. They can also gather information from System Trace and the Event Log. Beginning with Agents >= 4.5.19 on Linux, and >= 21.2.0 on Windows, Agents running on .NET Core can also instrument calls using loggers that implement the Microsoft.Extensions.Logging.ILogger API. To instrument calls using other loggers, add a custom logger definition. See Configuring a Custom Logger.

Messages logged as higher than ERROR include severe levels such as CRITICAL or FATAL.

If you do *not* want system trace or event log errors to be monitored, disable them by checking the appropriate box. System trace errors are anything written as an error message to the Listeners collection via TraceError.

Event log errors are anything written to the EventLog when the type is set to error. For example:

```
myEventLog.WriteEntry(myMessage, EventLogEntryType.Error, myID);
```

Error Configuration for PHP

The PHP Agent instruments the PHP reporting facility. PHP applications can use trigger_error to report errors through that facility. PHP extensions and PHP itself can also use the PHP facility for reporting errors.

By default, the agent can check for PHP errors using multiple thresholds:

- If you select Error, the agent reports only messages and exceptions marked Error.
- If you select Warning, the agent reports only messages and exceptions marked Error and Warning.
- If you select Notice, the agent reports messages and exceptions marked Error, Warning and Notice.

Error Configuration for Node.js

The Node.js Agent reports exceptions thrown by the Node.js application or by Node.js itself.

You can configure certain exceptions or logged messages not to cause transactions to be reported as errors using the ‘Ignored Exceptions’ and ‘Ignored Messages’ lists. See Ignoring Exceptions and Log Messages as Error Indicators.

Error Configuration for Python Notes

By default, the Python Agent reports error transactions for unhandled exceptions, HTTP status codes greater than or equal to 400, and messages logged at ERROR or higher by the Python logging module.

If you do not want any logged errors to cause transactions to be reported as errors, clear the "Mark Business Transaction as error" checkbox. If you do not want to capture logged messages at all, clear the "Detect Errors" checkbox.

You can configure certain exceptions or logged messages not to cause transactions to be reported as errors using the ‘Ignored Exceptions’ and ‘Ignored Messages’ lists. See Ignoring Exceptions and Log Messages as Error Indicators.

Service Endpoint Detection

AppDynamics automatically detects service endpoints for Java applications. You can configure custom service endpoints for Java or .NET to monitor the performance of services.

You can also configure exclude rules for service endpoints, which prevent agents from registering service endpoints for methods that match the exclude rules. The order of precedence for match, exclude and automatic rules are:

1. Exclude rules
2. Custom service endpoint rules
3. Automatic discovery

Permissions

To configure service endpoints your user account must have the Configure Service Endpoints permission for the business application. For information on AppDynamics Role Based Access Control, see Application Permissions.

Configure Custom Service Endpoints

You configure custom service endpoints like you configure custom match rules for business transactions:

1. From the **Service Endpoints** pane, click **Configure**.
2. Click the Custom Service Endpoint tab.
3. Choose the tier on which the service runs and click the plus icon to add a service endpoint configuration. The configuration settings are similar to business transaction entry point configuration settings. See Custom Match Rules.

As you choose a method for a service endpoint, avoid methods that occur within the following programming patterns:

- High-frequency loops, such as a for or while loop.
- Recursive functions.

Service endpoints inside such structures may cause performance overhead. Instead, consider adding a service endpoint at a point in the code upstream from a high-frequency loop or recursive function.

Configure service endpoint exclude rules in the same pane as the custom service endpoints. Click **Is Exclude Rule** on the **New Service Endpoint Definition** pane.

Automatic Discovery for Java

The Java Agent automatically discovers service endpoints. To configure service endpoint automatic detection settings, click to **Configure** on the **Service Endpoints** pane. You can configure the following settings:

- For existing service endpoint detection rules, you can enable or disable the detection rule.
- For servlet service endpoints, you can modify automatic naming as you would with servlet business transactions.

> The maximum limit for naming the service endpoints through automatic discovery is 99 characters.

POJO Service Endpoints

For the Java Agent to detect Plain Old Java Objects (POJO) service endpoints, you must do the following:

- Create at least one POJO custom service endpoint definition and enable it. The agent only detects POJO service endpoints for enabled custom service endpoint definitions.
- Verify automatic discovery for POJO service endpoints is enabled. It is enabled by default.

> When you disable automatic discovery for POJO service endpoints, the agent doesn't detect custom service endpoints.

Asynchronous Worker Thread Service Endpoints

The Java Agent automatically detects asynchronous worker threads spawned within a transaction as service endpoints. The agent names worker thread service endpoints for the class name, for example, "worker1". There is no end-to-end latency metric for worker thread service endpoints.

To disable worker thread service endpoints, register the enable-async-service-endpoints app agent node property with a value of "false". See App Agent Node Properties.

Configure Service Endpoints for .NET

The .NET Agent doesn't automatically discover service endpoints. To define a custom service endpoint for .NET, navigate to **Configuration > Instrumentation > Service Endpoints > Custom Service Endpoints**. Define the service endpoint as you would a custom match rule for a business transaction. See Service Endpoints and Transaction Detection Rules.

You can create service endpoints for the following entry point types:

- ASP.NET
- Message Queues
- POCO
- WCF
- Web Service

> The .NET Agent doesn't support service endpoints for MSMQ.

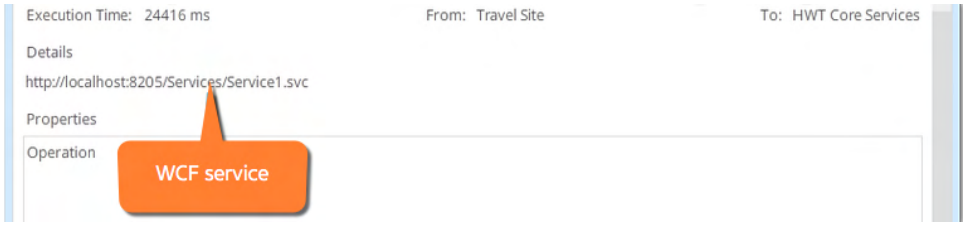
By default, you need to restart the instrumented application for instrumentation changes such as adding a POCO service endpoint to take effect. You can enable Runtime Reinstrumentation for the .NET Agent so that you don't need to restart your application and the AppDynamics.Agent.Coordinator after instrumentation changes.

After you configure a custom service endpoint, you can monitor it on the service endpoint dashboard that shows KPIs and transaction snapshots where the service endpoint executed. You can also use the metric browser to analyze service endpoint performance. See Service Endpoints.

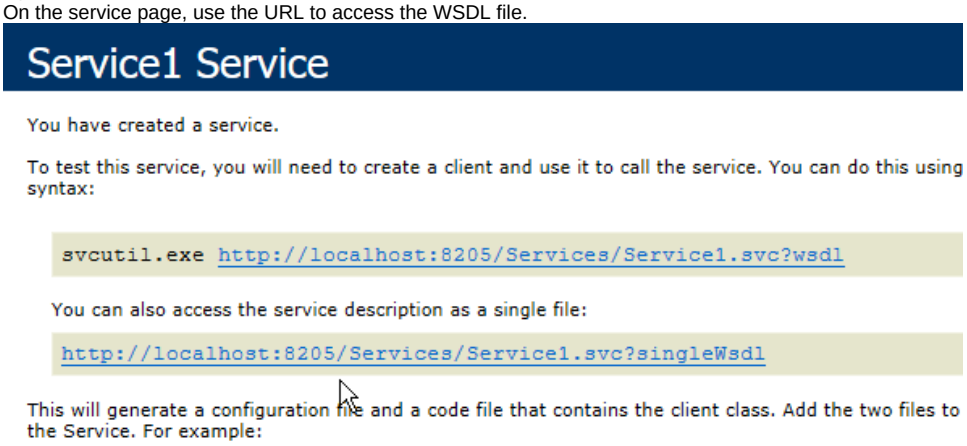
Tips for Defining .NET Service Endpoints

The methods that serve as the entry points for business transactions are often the same methods that you want to monitor as service endpoints as well. You can create service endpoints on the originating entry point for a business transaction.

- For ASP.NET service endpoints, you can use the URL to define the service endpoint. For example, in the Travel Site tier above, set a URL match to "`/Travel/Search`"
- For WCF and ASP.NET web service entry points, you can use an existing transaction snapshot to find the URL for the service.
    1. Open a full transaction snapshot for the business transaction.
    2. From the upstream tier, click the exit call link, either **WCF** or **Web Service**.
    The exit call window shows the URL for the web service.



    3. Open the service URL in a browser.
    4. On the service page, use the URL to access the WSDL file.



    5. The `service name element shows the service name.`

```
<wsdl:service name="Service1">
```

- Define POCO service endpoints exactly as you would a POCO custom match rule for a business transaction.

Data Collectors

Works with:

You can use data collectors to supplement business transaction and transaction analytics data with application data. The application data adds context to business transaction performance issues. For example, they may show the values of particular parameters or return values for business transactions affected by poor performance.

This data shows the business context affected by performance issues, such as the specific user, order, or product.

Types of Data Collectors

There are two types of data collectors:

- Method invocation data collectors - Capture code data, such as method arguments, variables, and return values.
- HTTP data collectors - Capture the URLs, parameter values, headers, and cookies of HTTP messages exchanged in a business transaction.

See Collect Transaction Analytics Data.

View Collected Data

When applied to business transactions, the data collectors supplement the information shown in transaction snapshots. The information captured by HTTP data collectors appears in the HTTP DATA and COOKIES sections, while method invocation data appears in the Business Data section.

To view collected data:

1. Navigate to the **Transaction Snapshots** pane.
2. Double-click the transaction for which you want to view data.
3. Click **Data Collector** to view collected data.

Configure a Data Collector

To configure data collectors, you need the Configure Diagnostic Data Collectors permission. Follow these steps to configure a data collector:

1. Identify the method on which to capture data. To do this, define the method signature and (optionally) filters based on the value of a code point in the method (such as, return value or argument).
2. Specify the actual code point that serves as the source of the data.
3. If the data collector applies to business transactions, then select the applicable business transactions.

When creating a data collector, you typically need to know the code on which you are setting up the collector, and whether it is based on access to the source code for the application, or its documentation. For the JVM 1.5 and .NET application environments, you must restart the JVM or application server if your method invocation data collector results in modifications to instrumentation configuration (class name, method name, method parameters, and so on).

These platform-specific considerations apply to data collectors:

- For C/C++ SDK: Create a method data collector using the `appd_bt_add_user_data function()`, not the Controller UI as described here. See C/C++ SDK Reference.
- For Node.js Agent: Create a method data collector using only the `addSnapshotData()` Node.js API, not the Controller UI as described here. See Node.js Agent API Reference.
- For Python Agent: Create a method data collector using the `add_snapshot_data()` Python Agent API, not the Controller UI. See Python Agent API Reference.
- For PHP method data collectors, only the **with a Class Name that** option is valid. You cannot add a method data collector to a standalone PHP function.

Configure a Data Collector Automatically

You can add a data collector directly from a transaction snapshot through the call graph. This automatically adds the class and method for you.

1. Double-click a call graph.
2. Drill down to the node you want.
3. Right-click the method you want.
4. Click **Configure Instrumentation for this Class/Method**.
5. Complete the wizard.

Configure a Data Collector Manually

Data collectors can be manually configured in at least two ways.

From the left pane:

1. Click **Configuration** in the left pane.
2. Click **Instrumentation**.
3. Click the Data Collectors tab, and click **Add** below the data collector box for the type of data collector you want to add. The data collector configuration panel displays.

From a call graph:

1. Double-click a call graph.
2. Click **Data Collectors** in the menu bar.
3. Click **Actions**.
4. Click **Configure Data Collectors**.
5. Click **Add** at the bottom of the data collector box for the type of data collector you want to add. The data collector configuration panel displays.

Note the following platform-specific considerations applicable to data collectors:

- C/C++ SDK - Create a method data collector using the `appd_bt_add_user_data function()`, not the Controller UI as described here. See C/C++ SDK Reference.
- Node.js Agent - Create a method data collector only using the `addSnapshotData()` Node.js API, not the Controller UI as described here. See Node.js Agent API Reference.
- Python Agent - Create a method data collector using the `add_snapshot_data()` Python Agent API, not the Controller UI. See Python Agent API Reference.
- PHP method data collectors - Only the "with a Class Name that" option is valid. Also, you cannot add a method data collector to a standalone PHP function.

The general steps for configuring a data collector are:

1. Identify the method on which to capture data. To do this, define the method signature and (optionally) filters based on the value of a code point in the method (such as return value or argument).
2. Specify the actual code point that serves as the source of the data.
3. If the data collector applies to business transactions, choose the applicable business transactions.

Typically, creating a data collector requires knowledge of the code on which you are setting up the collector, whether based on access to the source code for the application or its documentation. For some application environments, including JVM 1.5 and .NET, you will need to restart the JVM or application server if your method invocation data collector results in modifications to instrumentation configuration (class name, method name, method parameters, and so on).

Configuration Notes

- The **Apply to new Business Transactions** option applies the collector to business transactions created after you have configured the data collector. Otherwise, the data collector applies only to the business transactions you select in the subsequent data collector configuration pane.
- For **Class**, select the match condition that the data collector can use to identify the class, such as class name, implemented interface name, and so on. If matching by class name, then use the fully qualified name of the class, as appropriate for the application platform. For example, the form of the equals field value would be:
  - In Java: `com.appdynamics.model.Item`
  - In .NET: `Bookstore.Item`
  - In PHP: `book`
- **Is this Method Overloaded**: If this is an overloaded method, then add parameters that identify the signature. You must create a data collector definition for each form of the method for which you want to capture data. For example, given the overloaded method in this table, to capture data for only the second two forms, you would need to create two data collectors. The parameters to configure are shown:

Signature	Parameters to Configure
`getName()`	• None
`getName(int studentId)`	• Param Index 0: java.lang.int
`getName(int studentId, string name)`	• Param Index 0: java.lang.int • Param Index 1: java.lang.String

- You can refine the method selection to meet specific conditions. If you configure more than one match condition, then all match conditions must be satisfied by the request for the data collector to be applied to the request.
- Once you identify the method, specify the code point from which you want to capture data, such as the method return value, argument, or a value captured by the getter chain on the invoked object. Configure this code point in the **Specify the Data to Collect from this Method Invocation** section of the configuration settings.
- HTTP data collector can capture data from HTTP parameters, request attributes, cookies, and other data. Notice that the Content-Length HTTP header is already captured in AppDynamics as the Average Request Size metric. However, you may choose to configure a data collector for this header to have the value appear in transaction snapshots, providing you insight into, for example, whether message size corresponds to slow performance.
- You can configure multiple data collectors. The effect of multiple data collectors is cumulative. For example, if you add a custom data collector that does not include the collection of the URL HTTP request attribute, but keep the Default HTTP Request Data Collector configuration in which the URL is configured to be collected, then the URL is collected.

Method Invocation Data Collector Example

Method invocation data collectors are applicable to Java and .NET.

In this example of a manual setup (not using the wizard), we'll set up a method invocation data collector on a Java application. We want to create a data collector on the method `getCartTotal()`, which is shown in this code snippet with the method `getUser()` and which we will use later as a data source.

```
package com.appdynamicspilot.model;
...
public class Cart implements java.io.Serializable {
    ...
    private Double fakeAmount = 0.0;
    ...
    private User user;
    ...
    public User getUser() {
        return user;
    }
    ...
    public Double getCartTotal() {
        if (fakeAmount == 0.0) {
            double total = 0;
            if (items != null) {
                for (Item item : items) {
                    total += item.getPrice();
                }
            }
            return total;
        } return fakeAmount;
    }
   ...
}
```

To configure a data collector for an application, add a data collector and then configure it:

1. When you add your data collector, make your selections as appropriate for your configuration.
2. Add a class and a method, if not already filled in. Class and Method Name are used to identify the method.
3. At the **Enable Data Collector for** checkboxes, select **Transactions Snapshots** to add metadata for troubleshooting purposes and for APM snapshots, or select **Transaction Analytics** to collect metadata about every execution of the transaction to use later in the analytics platform. It is recommended that you start with ONLY Transactions Snapshots. Enable Transactions Analytics after the targeted data collection is confirmed.
4. Click **Add Parameter** or **Add Match Condition** to select the correct parameter from the popup.
    a. Method Parameters are added if the method is overloaded. Clicking **Add Parameter** adds the Param Index 0. Add the fully qualified class name for the parameter.
    b. Match Conditions allow you to pick specific data when a method or line of code is called multiple times. Match Conditions engage method parameters or return values. Selecting **Add Match Condition** opens the Create Match Condition popup. Make your selections, and then click **Save**.
    c. When setting up your data collector, note the checkbox beside **Apply to new Business Transactions**.
        i. Checking this box automatically applies the rule to new business transactions.
        ii. Not checking this box requires you to manually apply the rule to business transactions by navigating to **Configuration > Instrumentation > Data Collectors**.
    d. To add multiple collection types, click **Add** beneath Specify the Data to Collect from this Method Invocation to configure the source of data. For this section of code, use an invoked object and a getter chain of `user.getCustomerName`. This code snippet is an example of how to capture the user name on the invoked object. The `Cart` class instantiates a `User` object based on the following class in the same package as `Cart`. Note that the User class includes a method for returning the name of the user, `getCustomerName()`.

    ```
    package com.appdynamicspilot.model;
    ...
    public class User implements java.io.Serializable {
        ...
        private String customerName = null;
        ...

        public String getCustomerName() {
            return customerName;
        }
        public void setCustomerName(String customerName) {
            this.customerName = customerName;
        }
        ...
    }
    ```

    Using a getter chain, you can identify this method as another data source in the same data collector. For example, you could select **Invoked Object** as the source of the data. The getter chain `getUser().getCustomerName()` is the operation on the invoked object.
    For the PHP agent, if a method return value collected by MIDC is not stored in any variable, it is seen as null in both snapshot and Analytics data.
5. When finished, click **Save**.

After you configure the method invocation data collectors, you can click **Configure Transactions using this Data Collector** on the Instrumentation pane to drag and drop (or highlight and move) the transactions to a given data collector.

When complete, transaction snapshots for slow, very slow, and stalled transactions, the transaction snapshots will include the specified user data.

HTTP Data Collector Example

In the HTTP Data Collector configuration, specify the request data that you want to display in the snapshot.

To configure a data collector, add a data collector and then configure it:

1. Make your selections as appropriate for your configuration.
2. Determine if you are collecting parameters, cookies, session keys, or header data.
3. Click **+Add** beneath a data type to specify the request data that you want to display in the snapshot, and fill in as appropriate. When setting up your data collector, note the checkbox beside **Apply to new Business Transactions**.
    a. Checking this box will automatically apply the rule to new business transactions.
    b. Not checking this box requires you to manually apply the rule to business transactions by navigating to **Configuration > Instrumentation > Data Collectors**.
4. When finished, click **Save**.

Call Graph Settings

Related pages:

- Transaction Thresholds
- Transaction Snapshot Collection

Works with:

You can control the data captured in call graphs with the **Call Graph Settings** panel.

Permissions

To configure call graph settings, you need the **Configure Call Graph Settings** permission.

Call Graph Granularity

To control the granularity for call graphs, use the following settings:

- **Control granularity for Methods**: To ensure low-performance overhead, choose a threshold in milliseconds for method execution time. Methods taking less than the time specified here are filtered out of the call graphs.
- **Control granularity for SQL calls**: You can specify a threshold for SQL queries. SQL queries taking less than the specified time in milliseconds are filtered out of the call graphs. See Tune Java Agent Performance.

> Only SQL Capture Settings apply to call graph configuration for the Node.js agent.

To access call graph configuration, click **Configuration > Instrumentation** and choose the Call Graph Settings tab. There are subtabs for each application type.

Exclude Packages or Namespaces from Call Graphs

A call graph can potentially contain hundreds of methods. You can exclude packages (Java) or namespaces (.NET) with classes that you do not want to monitor.

For Java, some packages are excluded by default. These are visible in the Excluded Packages list. The packages that are excluded by default cannot be removed. However, you can include a particular sub-package from an excluded package.

You can customize call graph instrumentation from the call graph instrumentation page. From there, choose from these configuration options:

- Use the **Add Custom Package Exclude** (Java) or **Add Custom Namespace Exclude** (.NET) configuration options to exclude specific packages or namespaces from call graphs.
- Use the **Add Always Show Package/Class** (Java) or **Add Always Show Namespace/Class** (.NET) configuration options to have a package or namespace always shown in call graphs.

When the Controller constructs a call graph it uses excluded packages and included sub packages to determine which calls to include. However, the Controller includes some calls even if they are listed among the excluded packages. For example, web service calls.

SQL Capture Settings

The SQL capture settings control whether SQL statements are captured and presented in the Controller UI with dynamic parameters bound to their runtime values. For example, consider Java code that constructs a SQL call as follows:

```
stmt = new PreparedStatement("select * from user where ssn = ?")
stmt.bind(1, "123-123-1234")
stmt.execute()
```

With the capture raw SQL option enabled, AppDynamics captures and presents the SQL call in the following form:

```
select * from user where ssn = '123-123-1234'
```

If capture raw SQL is disabled, the SQL call appears in its original form, with question mark parameters not bound to values. Disabling capture-raw-sql and using question mark parameters in SQL prepared statements gives you a mechanism for preventing sensitive data from appearing in the Controller UI.

It is important to note that the sensitive values must be parameterized in the original, prepared statement form of the SQL statement, as shown above. The following statement results in the potentially sensitive information (social security number) appearing in the Controller UI whether capture raw SQL is enabled or disabled since the sensitive data is not parameterized.

```
stmt = new PreparedStatement("select * from user where ssn ='123-123-1234'")
```

To configure SQL capture settings, in the Call Graph Settings tab, scroll down to the **SQL Capture Settings** section and choose one of the following options:

- **Capture Raw SQL:** Select this option to have SQL statements that are composed as prepared statements captured with dynamic parameters bound to runtime values. By default, private SQL data and queries that take less than 10 ms are not captured.

  > **Important**
  >
  > You can select this option to capture raw query details for NoSQL databases such as MongoDB, DynamoDB, and CassandraDB and the relational queries will not be scrubbed.

  > When you enable Capture Raw SQL in .NET environments, the agent captures the parameters for ADO.NET stored procedure calls even though the parameters are not represented by question marks. It does not capture stored procedure local variables that are not available to the CLR.

- **Filter Parameter values:** Select this option to have SQL statements that are composed as prepared statements captured without dynamic parameters bound to runtime values.

Configure JMX Metrics from MBeans

Related pages:

- Exclude JMX Metrics

This page describes how to create persistent JMX metrics from MBean attributes. See Monitor JMX.

There are many helpful JMX topics on the AppDynamics community to help you with platform-specific troubleshooting tips.

JMX Metric Rules and Metrics

You can add persistent JMX-based metrics to AppDynamics using metric rules. Once you create a persistent JMX metric, you can:

- View it in the Metric Browser.
- Add it to a Custom Dashboard.
- Create a health rule for it so that you can receive alerts.

You can use the MBean Browser or JMX Metrics Rules Panel to create new metrics. MBean query expressions are supported.

You must have Configure JMX permissions for the application to configure new JMX Metrics your user account. See Create and Manage Custom Roles.

Access the MBean Browser

If the MBean is already monitored and you want to create a metric from one of its attributes, you can do so from the MBean Browser.

To create a metric from an existing MBean attribute:

1. Open the Node Dashboard of the node that hosts the MBean, and navigate to **MBean Browser** > **JMX**.
2. Expand the domains listed in the left panel and select the MBean that contains the attribute of interest.
3. Expand the **Attributes** section, select the attribute, and click **Create Metric**.

Configure JMX Metric Rules

1. Navigate to **Configuration > Instrumentation** > **JMX** to add an MBean and attributes, possibly based on complex matching patterns.
2. From the JMX Metric Configurations panel, click the Java platform to add the metric to a platform group.
3. Alternatively, create a new group and add your metric to the group.
   Groups allow you to organize metrics to meet your needs. For example, you may choose to keep custom metrics with their Java platform, or put all metrics you add into a Custom group to differentiate them from out-of-the-box metrics.

JMX Metric Settings

The general settings for adding instrumentation for an MBean are:

- The **Name** is the identifier you want to display in the UI for the MBean.
- An **Exclude Rule** excludes matched MBeans that would otherwise be included by other rules. See Exclude MBean Attributes.
- **Enabled** means that you want this rule to be applied.
- The **Metric Path** determines where the metric appears in the metric browser. For example, in the screenshot below, the metric path is "Web Container Runtime" for JMX metric "Tomcat_HttpThreadPools".

In the MBeans subpanel, add matching criteria to identify the MBeans that you want to monitor.

- The **Domain name** is the Java domain. This property must be the exact name; no wildcard characters are supported.
- The **Object Name Match Pattern** is the full object name pattern. The property may contain wildcard characters, such as the asterisk for matching all the name/value pairs. For example, specifying `jmx:type=Hello,*` matches a JMX MBean ObjectName, jmx:type=Hello,name=hello1, key1=value1.
- The **Instance Identifier** is the MBean ID.
  Instance Identifier is a field in the JMX Object pattern that can be used to categorize the runtime context of MBean category that has the same set of attributes. For example, the user category has two HTTP ports: 8090 and 8181. But attributes such as `currentThreadsBusy` and `maxThreads` are the same. If a JMX pattern has both HTTP and AJP, but the user is interested in monitoring only HTTP and not AJP thread pool, then the user uses an identifier, a name that matches HTTP so that only HTTP ports related MBean data is collected.
- **Advanced MBean Matching Criteria** is an optional control for matching against attribute values. Identify the attribute name and value to test, along with matching criteria, such as a substring match.

Without any instance identifier specified, a tree of resources or attributes is shown on the JMX Metric browser that matches the Object Name Match Pattern.

Example1

```
JMX -->
        ConnectionPoolModules
                        ConnectionPoolA

                                        PoolSize
                        ConnectionPoolB

                                        PoolSize
```

If you specify an `instanceIdentifier` like `instance identifier = path2`, it will show the second level attributes on the top level also.

The above tree will be shown as in the following example. The `PoolSize`, which is in the second level, also appears in the top level.

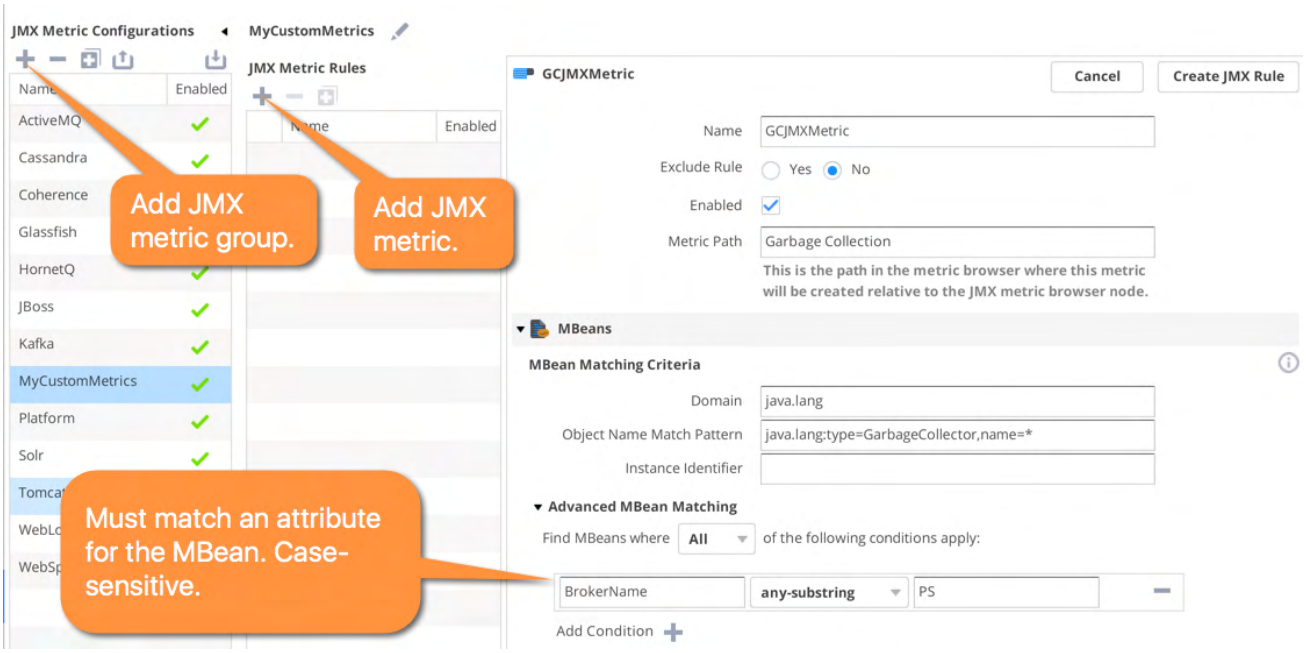
```
JMX -->
        ConnectionPoolModules
                        PoolSize

                        ConnectionPoolA

                                        PoolSize
                         ConnectionPoolB

                                        PoolSize
```

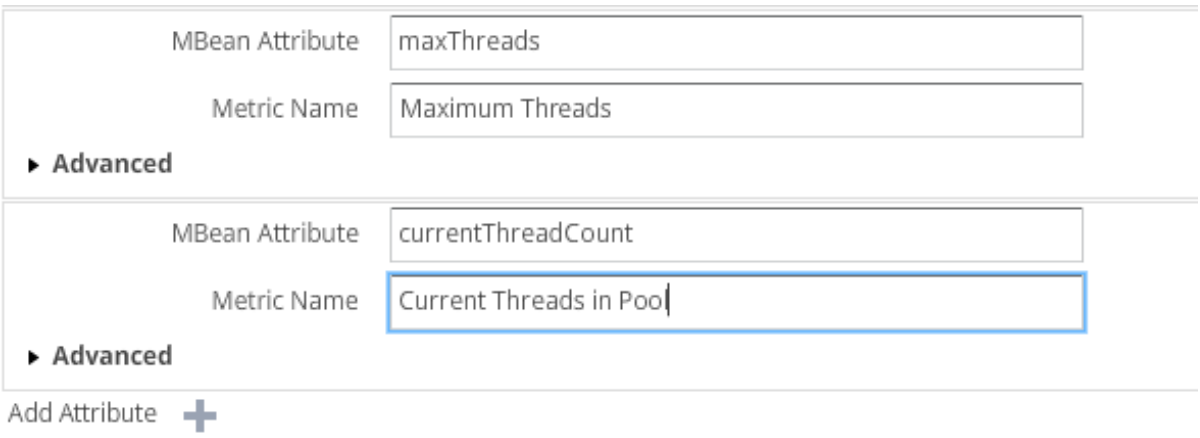
For example, this screenshot displays the MBean matching criteria for the `GCJMXMetric` rule.



For all matched MBeans, you can define one or more metrics for the attributes of those MBeans:

- **Metric Getter Chain**—Expressions can be executed against any value. In addition, getter chains for Strings and Booleans are supported using implicit conversion. See MBean Getter Chains.
- **Metric Time Rollup**—Determines how the metric will be aggregated over a period of time. You can choose to either average or sum the data points, or use the latest data point in the time interval.
- **Metric Cluster Rollup**—Defines how the metric will be aggregated for a tier, using the performance data for all the nodes in that tier. You can either average or sum the data.
- **Metric Aggregator Rollup**—Defines how the Agent rolls up multiple individual measurements (observations) into the observation that it reports once a one minute. For performance reasons, Agents report data to the Controller at one-minute intervals. Some metrics, such as Average Response Time, are measured (observed) many times in a minute. The Metric Aggregator Rollup setting determines how the Agent aggregates these metrics. You can average or sum observations on the data points or use the current observation. Alternatively, you can use the delta between the current and previous observation.

For example, the maxThreads MBean attribute is mapped to the Maximum Threads metric in the JMX metrics browser as shown here:



Example2

For instance, in a JBoss you get get the active count property from the following mbean:

jboss.as:subsystem=datasources,data-source=MyDatasource_01,statistics=pool

Define the metric rule:

Object name Match Pattern:jboss.as:subsystem=datasources,*,statistics=pool
Instance identifier: data-source
Attribute: ActiveCount

In the Object Name Match Patern, you can even use the wildcard to replace several branches, which is, if it were "jboss.as:subsystem=datasources, whatever=another_level,whatever=another_level,data-source=MyDatasource_01,statistics=pool" you can use "jboss.as:subsystem=datasources,*, statistics=pool".

The Instance Identifier is the instanced element that is separated:

jboss.as:subsystem=datasources,data-source=MyDatasource_01,statistics=pool

jboss.as:subsystem=datasources,data-source=MyDatasource_02,statistics=pool

...

jboss.as:subsystem=datasources,data-source=MyDatasource_07,statistics=pool

Therefore, you must use data-source as the instance identifier.

Export and Import JMX Configuration

After you modify the JMX configuration, you can backup or transfer the configuration using the JMX Configurations export and import icons at the top of the JMX Metric Configuration panel on the JMX instrumentation page. The configuration exports as an XML file.

Deactivate Transaction Monitoring for JMX Monitored Nodes

In some circumstances, such as for monitoring caches and message buses, you want to collect JMX metrics without the overhead of transaction monitoring.

You can do so by turning off transaction detection at the entry point. See Transaction Detection Rules.

MBean Getter Chains

Getter chains in custom MBean configuration specify the method that retrieves the value of interest.

Expressions can be executed against any value. In addition to getter chain support for numeric boxed primitives (Short, Integer, Long, and so on), Strings and Booleans are supported using implicit conversion.

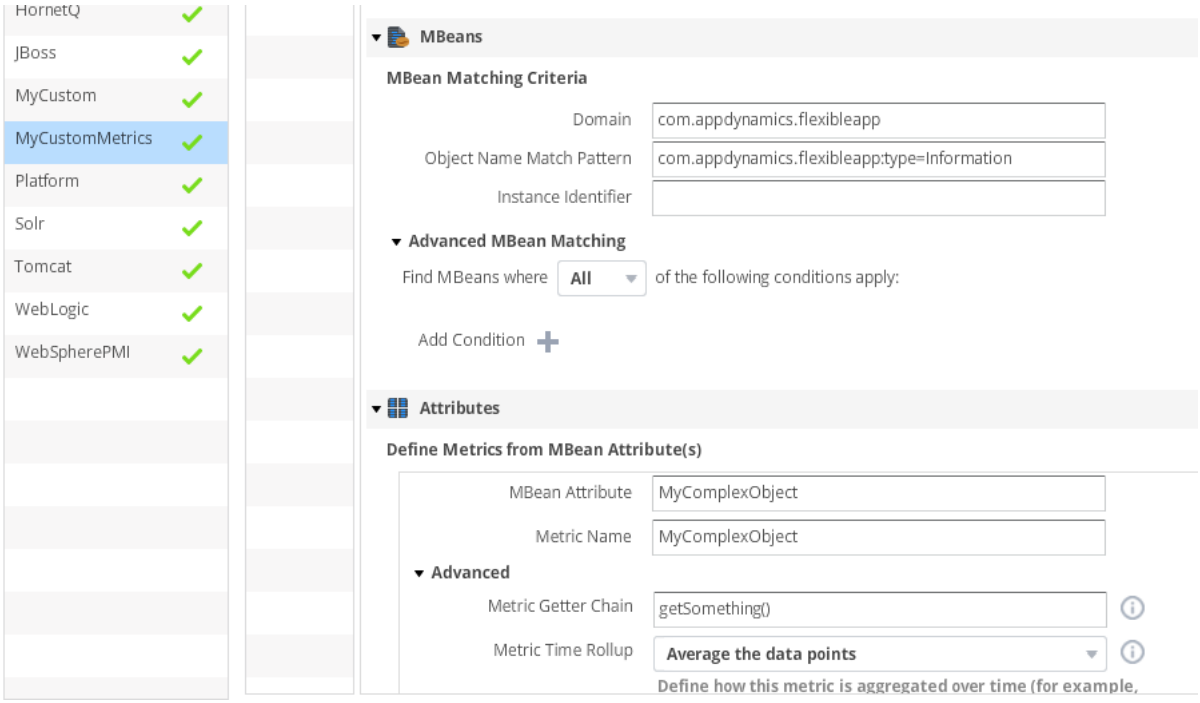
Boolean and String are implicitly converted to an Integer:

- Booleans are automatically converted to 0 (false) and 1 (true).
- Strings are converted to numeric values.

This example illustrates how to use a getter chain, given the class with a `getSomething()` method that returns the time:

```
1 package com.appdynamics.flexibleapp.mbean;
2
3 public class MyComplexObject
4 {
5     public long getSomething()
6     {
7         return System.currentTimeMillis();
8     }
9
```

In this example the metric getter chain configuration would be:



Exclude JMX Metrics

This page describes how to exclude MBean attributes from being monitored as JMX metrics. For background information about JMX metrics, see Monitor JMX.

Customize Metrics To Gather

AppDynamics provides a default configuration for certain JMX metrics. However, in situations where an environment has many resources, there may be too many metrics gathered. AppDynamics lets you exclude resources and particular operations on resources.

Exclude a Metric

For example, suppose you want to exclude monitoring for HTTP Thread Pools. Follow the procedure described in Create a new JMX Metrics Rule, using the following criteria:

1. Set the **Exclude Rule** option to **Yes**.
2. Provide the **Object Name Match Pattern**:

```
Catalina:type=ThreadPool,*
```

3. Provide the **Advanced MBean Matching** value:

```
http
```

This configuration directs AppDynamics to stop monitoring metrics for HTTP Thread Pools. You can clear the **Enabled** checkbox to disable the rule.

Exclude MBean Attributes

Some MBean attributes contain sensitive information that you do not want the Java Agent to report. You can configure the Java Agent to exclude these attributes using the `<exclude object-name>` setting in the `app-agent-config.xml` file.

To exclude an MBean attribute:

1. Open the `AppServerAgent/conf/app-agent-config.xml` file.
2. The new configuration takes effect immediately if the agent-overwrite property is set to true in the `app-agent-config.xml`. If `agent-overwrite` is `false`, which is the default, then the new configuration will be ignored and you have to restart the agent. Set the property to `true`.

```
<property name="agent-overwrite" value="true"/>
```

3. Locate the `JMXService` section. It looks like this:

```
<agent-service name="JMXService" enabled="true">
```

4. In the `JMXService <configuration>` section add the `<jmx-mbean-browser-excludes>` section and the `<exclude object-name>` property as per the instructions in the comment.

```
<configuration>
    <!--
       Use the below configuration sample to create rules to exclude MBean attributes from MBean Browser.
       <exclude object-name=<MBean name pattern> attributes=< * |comma separated list of attribute
names> >
       The example below will exclude all attributes of MBeans that match "Catalina:*".
       <jmx-mbean-browser-excludes>
          <exclude object-name="Catalina:*" attributes="*"/>
       </jmx-mbean-browser-excludes>
    -->
 </configuration>
```

5. Save the file.

JMX Logging

JMX logs contain information about JMX interactions between AppDynamics agents and the monitored server's JMX domains.

JMX log entries are useful for diagnosing problems associated with JMX metrics or with JVM metrics that rely on JMX. For example, they can help you determine why MBeans or metrics created from MBeans do not appear in the Controller UI.

The name of the log is JMX*Year_mon_day_hr_min*.#.log, where # is the log set. See Agent Log Files for information about the structure of the log files into sets.

The JMX log file can reach a maximum of 5MB within a set.

Collecting JMX log (sample):

```
<ADRRAFAppender name="JMXLogger" fileName="JMX.log">
            <PatternLayout pattern="[%t] %d{DATE} %5p - %m%n" />
            <SizeBasedTriggeringPolicy size="20 MB" />
            <ADRolloverStrategy max="5" />
        </ADRRAFAppender>
```

```
<AsyncLogger name="com.singularity.JMX" level="info" additivity="false">
            <AppenderRef ref="JMXLogger" />
        </AsyncLogger>
```

These configurations can be used in log4j2.xml file.

Asynchronous Transaction Demarcators

To monitor end-to-end transaction performance for an asynchronous transaction, you must identify the demarcator for the transaction's logical endpoint. There are two ways you can define a transaction demarcator:

- For Java applications, you can specify a tier on which the end to end transaction processing is completed. This lets you determine the logical transaction response time for transactions that perform asynchronous backend calls (such as JMS calls or web service calls). When the last of the threads spawned by the transaction terminates on that tier, the agent considers the transaction complete.

  > The Java Agent uses patent-pending heuristics to determine when the last thread associated with the Business Transaction has run. These heuristics are successful for many common cases. In complex environments with many thread handoffs unconstrained by frameworks, these heuristics may be inaccurate. For example Reactive Java, AKKA, or Scala environments.
  >
  > You should validate that the last thread on tier mechanism returns an end to end time consistent with the end to end latency shown within full transaction snapshots.

- For Java or .NET applications, you can identify the method that acts as the logical endpoint for the transaction processing sequence. For a response handler, this could be a method that watches for spawned threads to complete and when done, assembles the response and sends it back to the client. The end to end transaction time includes the time it takes for this configured method to finish processing, not when the method is invoked.

You may specify more than one endpoint demarcator for a particular business transaction. In this case, the first match ends the transaction for purposes of end-to-end latency monitoring. Be careful not to configure multiple demarcators that could be satisfied on different tiers for a given transaction.

For the last thread on tier demarcator type, the transaction endpoint is considered to be the time when the thread that receives the traced transaction terminates or the latest points at which any descendant threads terminate.

For the method-based transaction demarcator option, you can configure the demarcator to consider runtime state, such as the values of parameters passed to the method. This allows you to account for an application design in which the completion of a logical business transaction is signaled by the value of a method parameter or return value.

> For .NET POCO entry point on methods that return a `Task` class, AppDynamics tracks the return of the task object as an end to end latency transaction automatically. You can configure your own transactions as end to end transactions for Java and .NET applications as described here. See POCO Entry Points for information on `Task`-based POCO entry points.

To create the configuration:

1. Open the **Configuration** > **Instrumentation** page and choose **Asynchronous Transactions** from the top menu. You likely need to expand the menu list (**>>**) to view the **Asynchronous Transactions** item.
2. Click **Add** and choose the demarcator method:
   - Java only: **Transaction is complete when last thread on specified Tier is finished running**.
   - Java or .NET: **Transaction is complete when specified class/method is invoked**.
3. Follow the instructions that display for the option you choose. Note the following points:
   - You configure the class/method option using the standard method selector. See Configure Instrumentation for more information about identifying classes and methods in the AppDynamics configuration.
   - For the last thread on the tier approach, first, give a name for the transaction configuration and choose the tier.
4. Choose the business transaction where the configuration applies.

The threads in the logical transaction processing flow must be traceable by AppDynamics, including the thread that contains the end-point method. If needed, configure custom thread correlation to ensure that all threads are properly traced. For more information on thread correlation, see Configure the Thread Correlation in Java Agent and Thread Correlation for .NET.

Automatic Instrumentation of Specialist Packages and Frameworks

Related pages:

- Transaction Detection Rules
- Backend Detection Rules

One of the goals of the AppDynamics Application Performance Management (APM) Platform is to provide maximum visibility into Enterprise IT systems for the minimum of set-up effort.

To achieve this, AppDynamics configures its agents to place instrumentation on entry points, exit points, and thread hand-offs right out of the box. The user interface facilitates configuring naming rules for transactions and backends.

Within this documentation, you can find pages about Supported Environments, Transaction Detection, and Backend Detection for many common frameworks (including servlet, JAX-RS, and JDBC) and many HTTP clients.

In addition, the following pages describe the specialized out-of-the-box configuration for instrumenting off-the-shelf products, packages, or frameworks whose entry and exit points are too complex for the more general documentation:

- IBM-BPM Support
- Spring Batch Support
- OSB Support
- Open Tracing Support

IBM-BPM Support

IBM® Business Process Manager (IBM BPM) is a comprehensive business process management platform. It provides tools to enable authoring, testing, and deployment of business processes, as well as management capabilities. See IBM documentation.

IBM BPM 8.5.7 is used to define and execute Business Process Definitions (BPDs) which combine elements of human interaction Client-Side Human Services (CSHS) or 'user tasks' for example, manual loan approvals and so on with elements of system integration system tasks, for example, registering a loan approval in a CRM system and so on. These processes are critically important and it is, therefore, natural to use AppDynamics Business Transactions to monitor the system integration elements of the processes to ensure performance and reliability of these technical integrations. You can also use the AppDynamics Business Journeys capability to monitor the end to end progress of the processes, including the elements of human interaction.

AppDynamics Instrumentation Use Cases

To facilitate these monitoring use cases, the AppDynamics agent provides out-of-the-box configuration necessary to:

- Appropriately start and name Business Transactions originated within IBM BPM to allow management of system tasks with App iQ APM
- Correlate the processing within IBM BPM to facilitate end-to-end tracing of these Business Transactions
- Collect sufficient process metadata to allow these Business Transactions to be used within Business Journeys to provide visibility into the progress of processes including the human interactions

Agent Installation and Setup

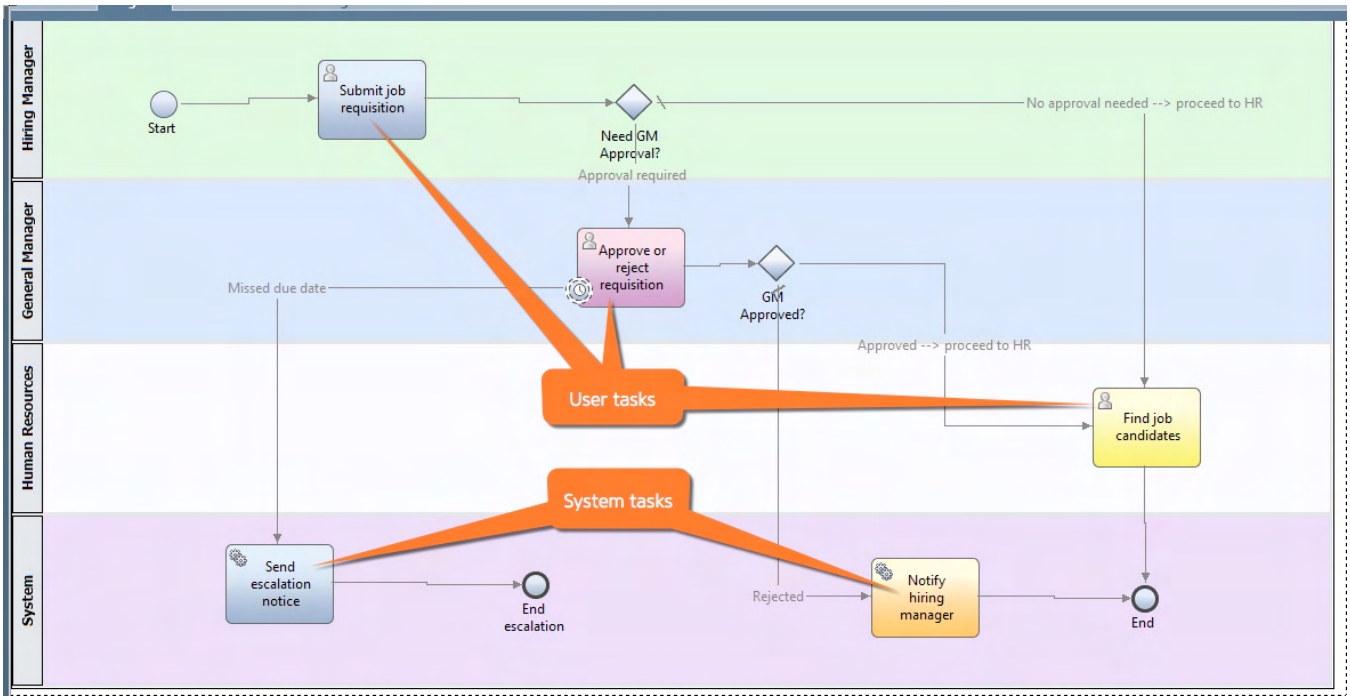
IBM BPM runs in WebSphere, see IBM WebSphere and InfoSphere Startup Settings for more details.

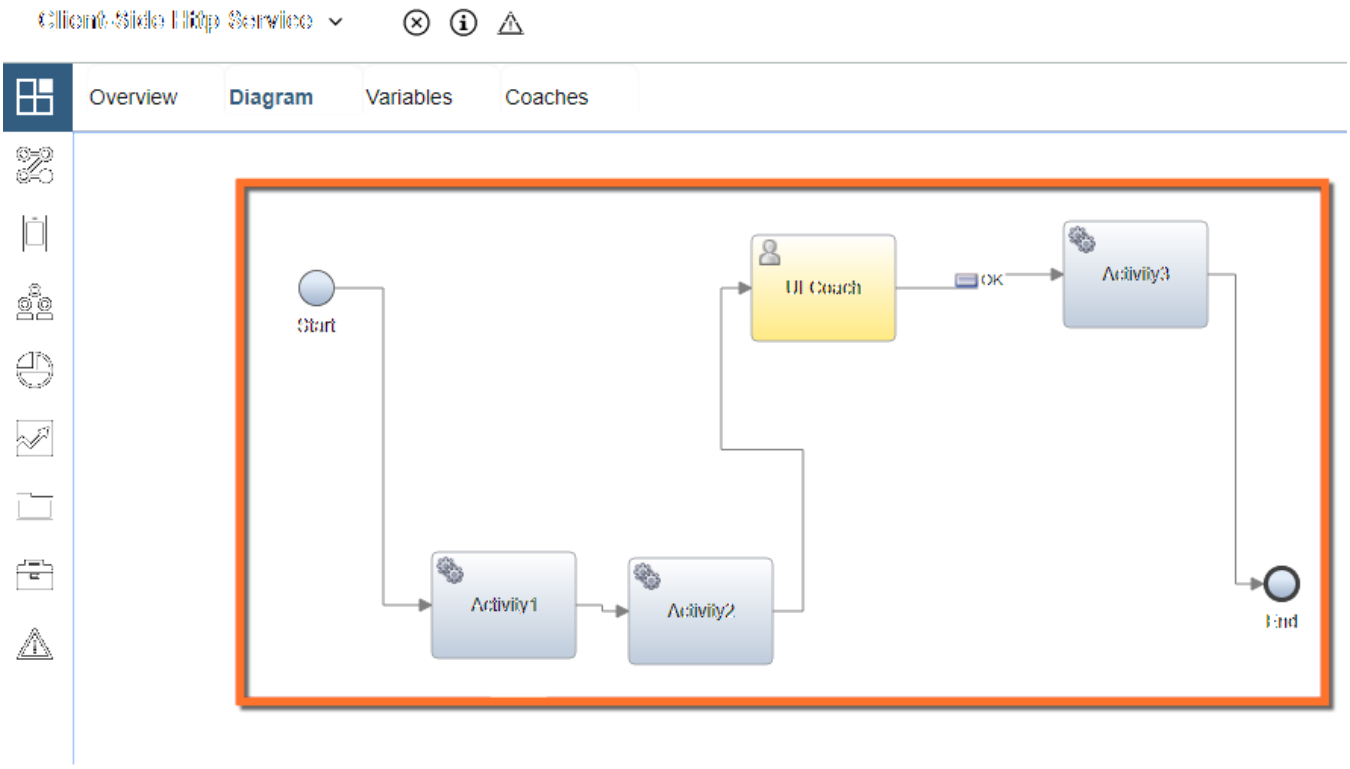
Business Process Definitions: System and User Tasks

A BPD process consists of the flow of a business process which may involve user interactions. It consists of User Tasks, System Tasks, and other logic.

The screenshot displays a sample process within the BPD designer:

User tasks, in turn, break down into one or more user interactions, such as submit a form and approve a decision. They can be implemented as a CSHS, one of which is shown below:





A CSHS always starts with a start event and consists of one or more activities each of which uses a system service as the underlying implementation, and one or more Coaches, which define the user interactions.

Business Transaction Support

Business Transaction Support for Client Side Human Services

CSHS Transaction naming

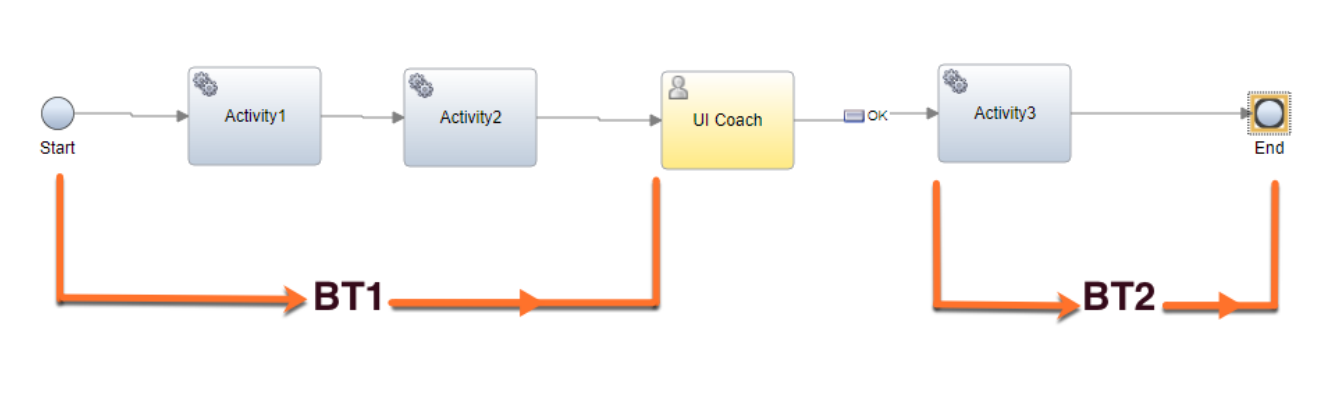
All Business Transactions detected within a CSHS are detected as servlet type transactions, and their naming can be configured using custom servlet naming rules. The default servlet naming rule uses only the first two segments of the URL for naming, hence, without custom naming configuration the Business Transactions will be detected as –

```
/<Project Name>/<BPD Name>
```

CSHS Transaction naming control

To disable the IBM BPM specific naming for CSHS Business Transactions use the `disable-ibmbpm-usertask-bt-naming` node property.

Since the Business Transaction is designed to monitor technical activity only, one CSHS may spawn multiple Business Transactions:



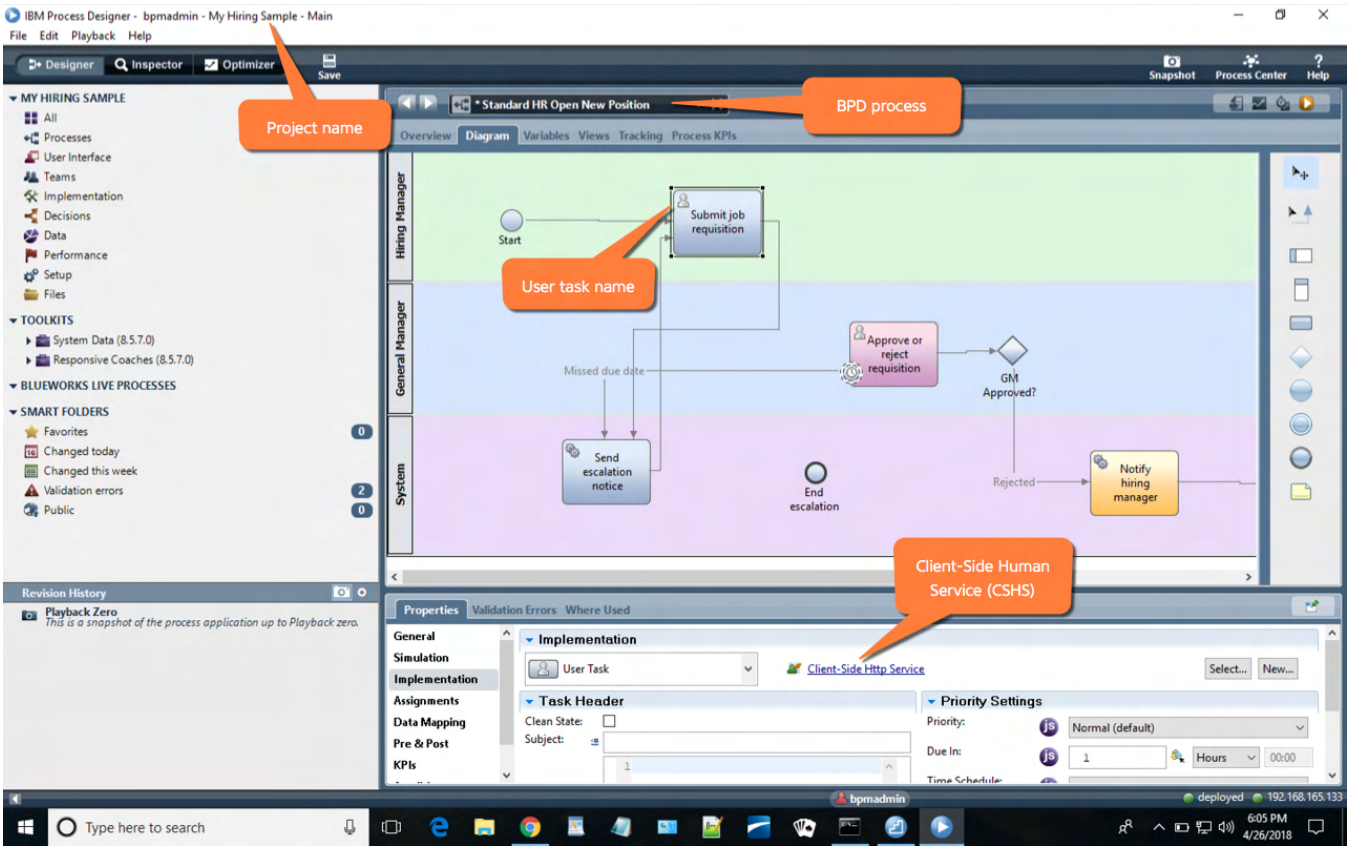
Any element inside a CSHS is associated with the following identifiers:

- Project Name
- BPD Process Name
- User Task Name
- CSHS Process Name
- Activity Name or Coach Name

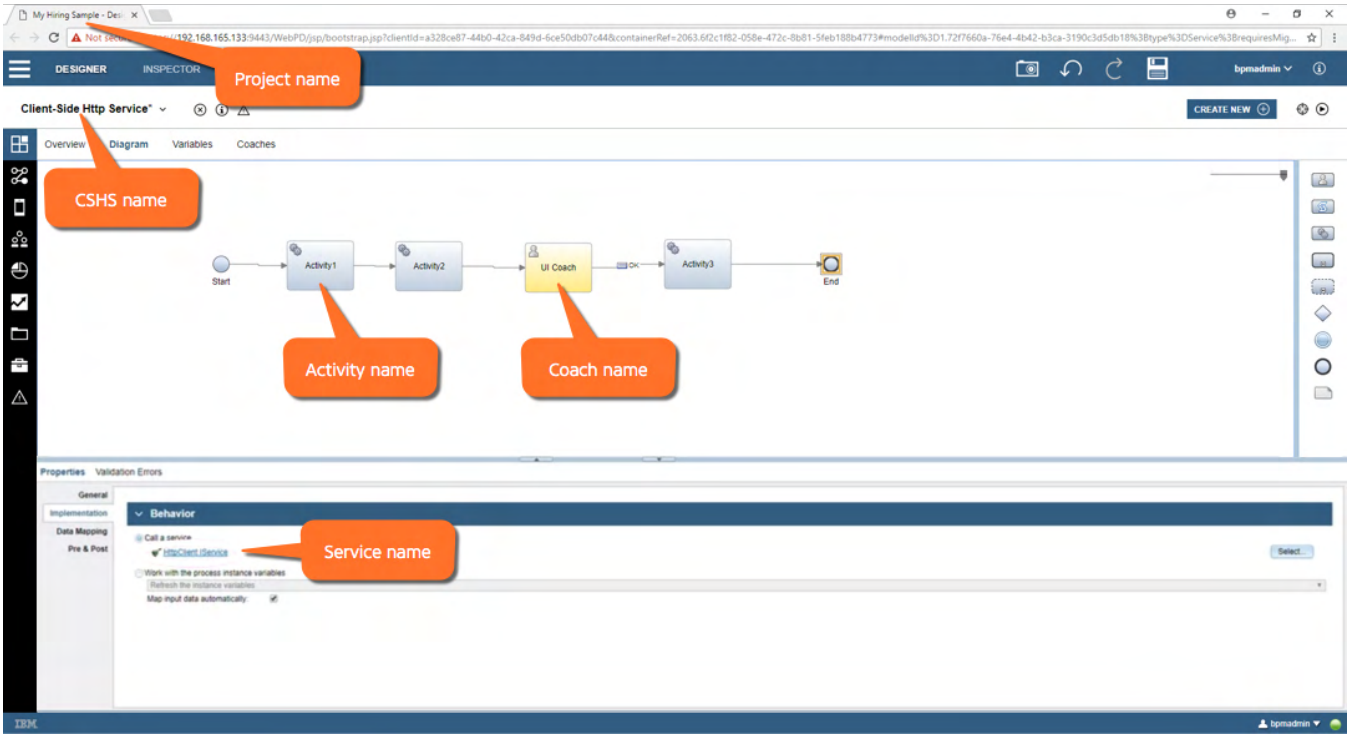
Additionally, for Activity (Service), the activity has a service as its implementation which has its own identifier (Service Name).

These are annotated on screenshots of the IBM design tools:

BPD Screen



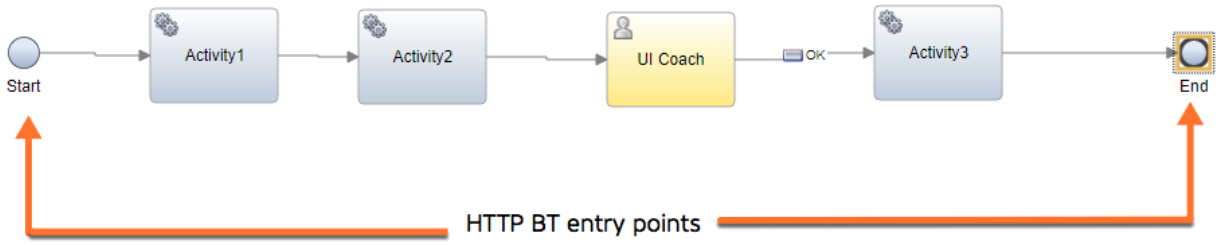
CSHS Screen



Business Transaction Detection and Naming at the Start and End Points within a CSHS

Within a CSHS, there can be one or more activity-services involved in the flow. Each of this hits a specific REST URL. Hence, an HTTP entry point is defined for each of these services.

- For example, the default URL for the **Start** BT – `/teamworks/process.lsw`
- For example, the default URL for the **End** BT – /teamworks/cfecontroller



.

Business Transactions originating from the start of a user task are detected as servlet Business Transactions and are named according to the following scheme:
`/<Project Name>/<BPD Name>/<Task Name>/<CSHS Name>/Start:Event`

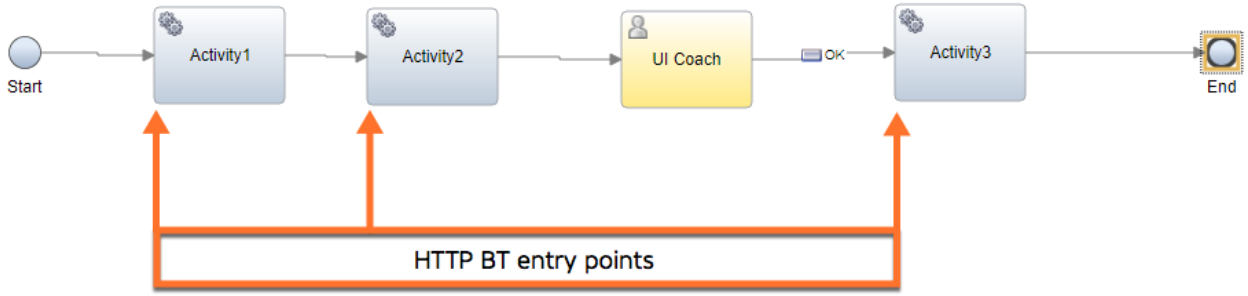
For example (corresponding to the images shown above, BPD image and CSHS image), the Business Transactions would be named –
`/My Hiring Sample/Standard HR Open New Position/Step: Submit job requisition/Client-Side Http Service/Start:Event`
`/My Hiring Sample/Standard HR Open New Position/Step: Submit job requisition/Client-Side Http Service/End:Event`

Business Transaction Detection and Naming at an Activity (Service) Node Within a CSHS

Each user task in the BPD process has a CSHS as its implementation. Within a CSHS, there can be one or more activity-services involved in the flow. Each of them hits a specific REST URL. Hence, HTTP entry point is defined for each of these services.

For example, the default URL is –

`/rest/bpm/wle/v1/coachflow/service/1.a22ce595-1573-4fdd-90f6-734884753610`



Business Transactions originating within a CSHS activity service are detected as servlet Business Transactions and are named according to the following scheme:
`/<Project Name>/<BPD Name>/<Task Name>/<CSHS Name>/<Activity Name>/<Implementation Service Name>:Service`

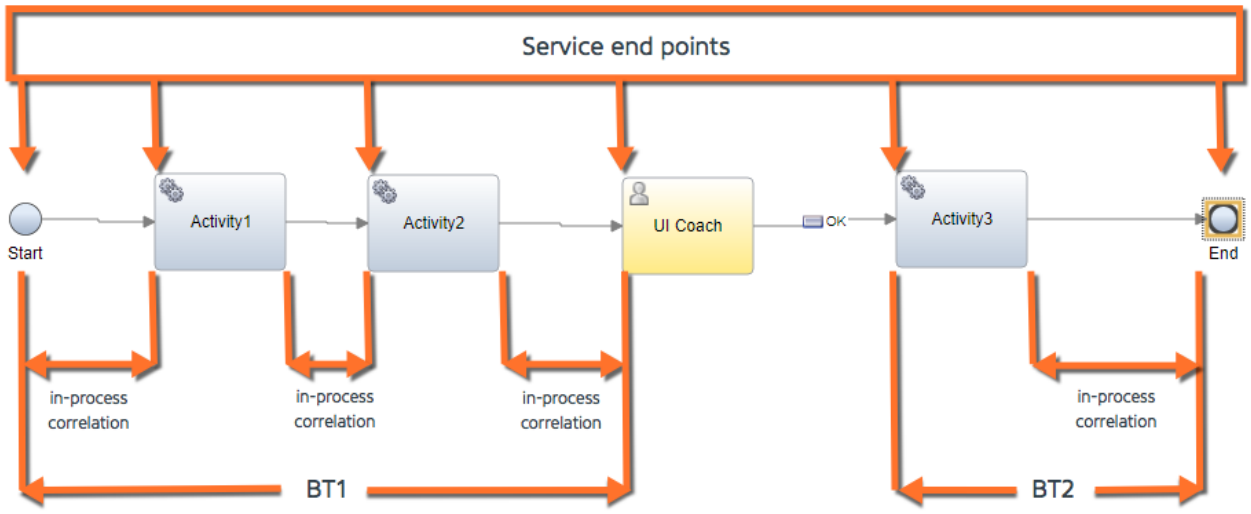
For example (corresponding to the images BPD image and CSHS image), the Business Transactions would be named:
`/My Hiring Sample/Standard HR Open New Position/Step: Submit job requisition/Client-Side Http Service/Activity1`
`/HttpClient IService:Service`

Business Transaction Detection at the CSHS Coach

A User Task in a BPD process has a CSHS as its implementation. Within a CSHS, there can be one or more UI Coach(es) involved in the flow. Each UI Coach represents the UI page(s) which opens up for the user to perform some actions (like completing a form, approving something, and so on). Each of these hits a specific REST–URL. Hence, the servlet entry point is used for the naming of the UI Coach.

For example, the default URL for this Business Transactions is –

`/teamworks/generatecoachng`



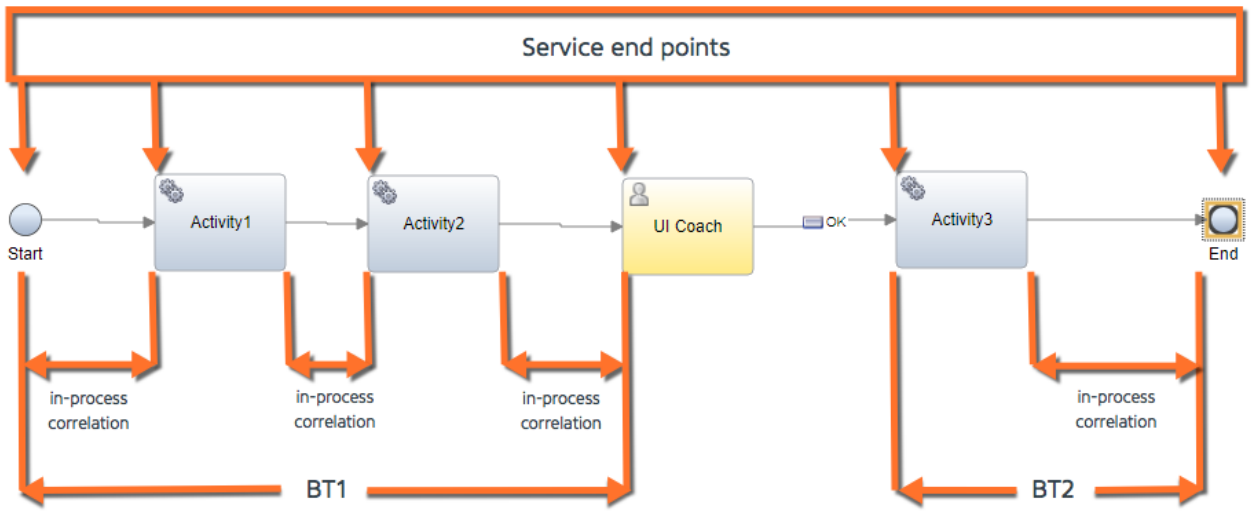
Business Transaction Naming Scheme at Coach of CSHS

The Business Transactions originating on entry to a UI coach are detected as Servlet Business Transactions –
`/<Project Name>/<BPD Name>/<Task Name>/<CSHS Name>/<Coach Name>:Coach`

For example (corresponding to the images BPD image and CSHS image), the business transaction would be named:
`/My Hiring Sample/Standard HR Open New Position/Step:Submit job requisition/Client-Side Http Service/UI Coach: Coach`

In-Process Correlation Between Elements within a CSHS

There can be more than one element involved within a User Task (CSHS) with no user interaction in between them. These form a part of the same business transaction. Service Endpoints give visibility of the individual steps within the resulting business transaction as illustrated below:



Business Transaction Support for System Tasks within a BPD Process

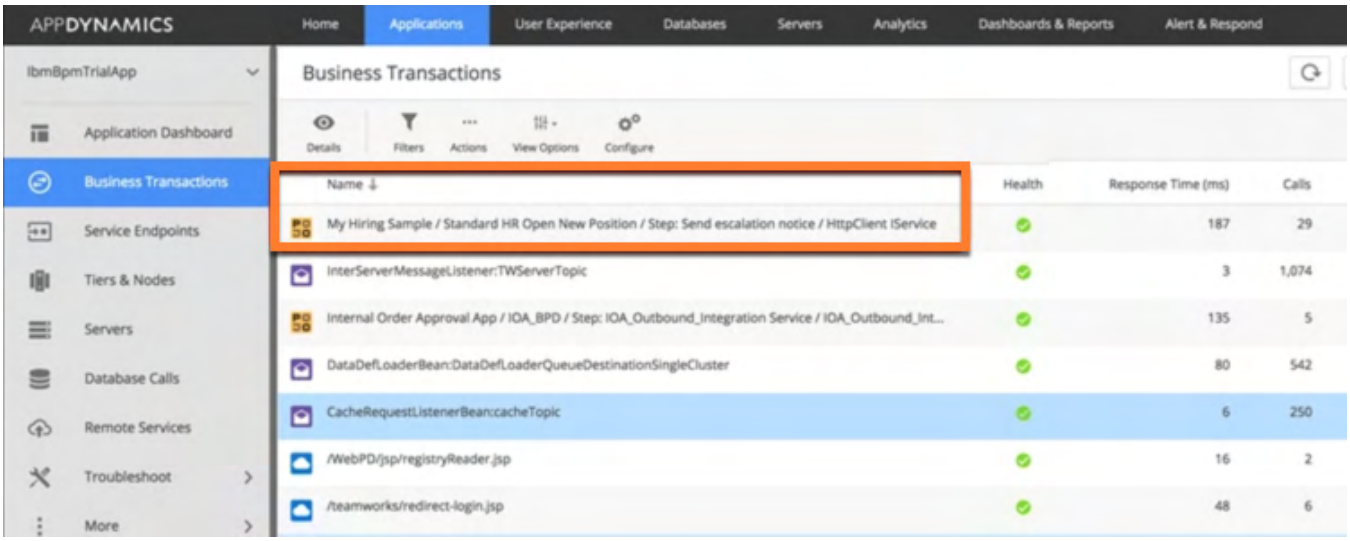
POJO business transactions are generated for System Tasks within a BPD process.

The POJO business transactions are named using four identifiers:

1. Project Name (*<project-name>*)
2. BPD Process Name (*<bpd-name>*)
3. System Task Name (*<task-name>*)
4. Implementation Process Name (*<implementation-name>*)

If any identifier's value is not available, it is replaced with <UNKNOWN> in the resulting transaction name.

For example, a business transaction detected for default configuration (*<project-name> / <bpd-name> / <task-name> / <implementation-name>*) is:



The default naming scheme can be customized using the ibmbpm-systemtask-bt-naming node property.

The property value is comma-separated identifiers chosen from project, bpd, task, implementation, and is order sensitive.
For example if,

value = "project, task" means POJO Business Transactions are named as: <project-name>/<task-name>

value = "task, project" means POJO Business Transactions are named as: <task-name>/<project-name>
value = "none" means POJO Business Transactions are not detected (disabled)
value = "default" means all the identifiers are used for Business Transactions naming in a default order, that is, they are named as *<project-name>/<bpd-name>/<task-name>/<implementation-name>*

Set ibmbpm-systemtask-bt-naming property to "none" to disable detection of Business Transactions for system tasks.

Built-in Data Collectors

> **Controlling the Built in Data Collectors**
>
> To disable the out of the box data collectors use the `disable-ibmbpm-data-collectors` node property.

For a System Task or a User Task, the following data collectors are available:

- PROCESS NAME
- TASK ID
- TASK NAME
- BPD INSTANCE ID
- BPD NAME
- PROJECT NAME

In addition to these, for a CSHS implementation of a User Task, if the IBM BPM specific naming Business Transactions naming scheme is enabled, an additional data collector, **ACTUAL URL**, will also be present, populated with the actual URL hit when the business transaction is initiated.

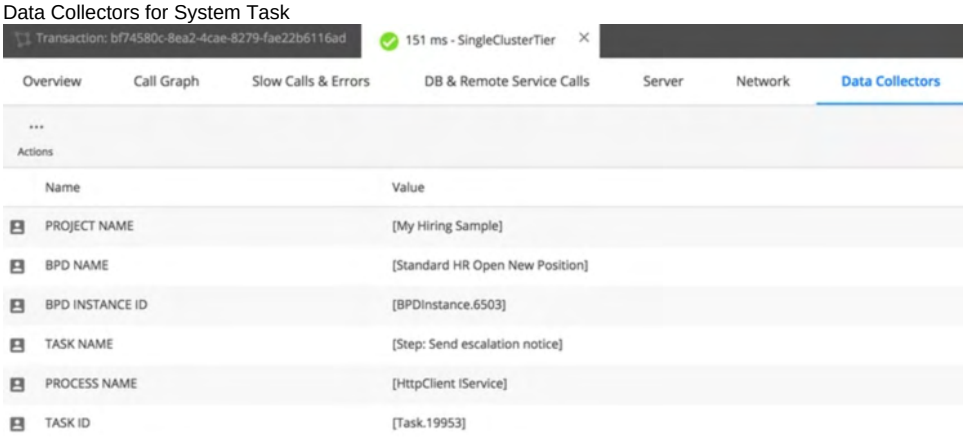
Additional data collectors defined for a User Task depending upon the element of the CSHS being called can be:

- COACH NAME for Coach
- EVENT TYPE for Start and End (value is either **Start** or **End** depending upon on the event)
- ACTIVITY NAME for Activity (Service)
- SERVICE NAME for Activity (Service – denoting the implementation service of the activity)

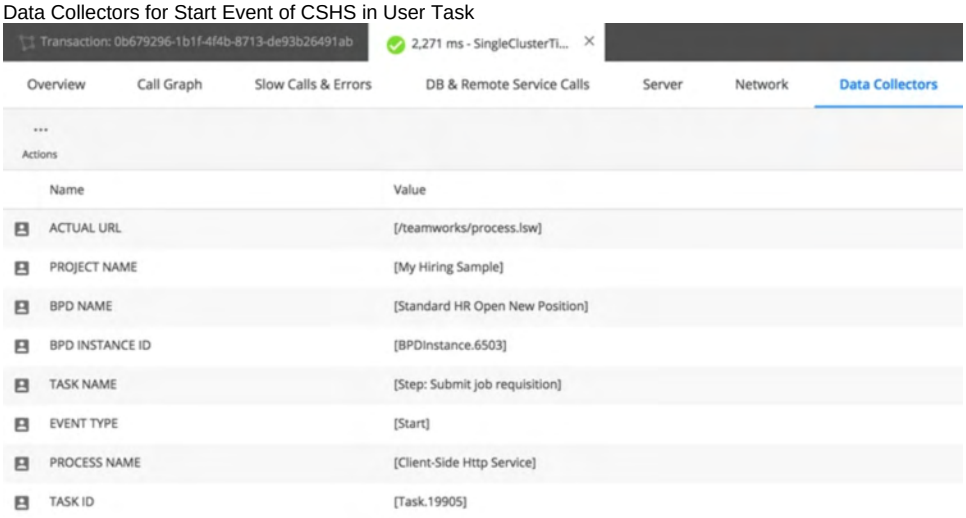
Sample Data Collectors

You can view the snapshot data collectors as:

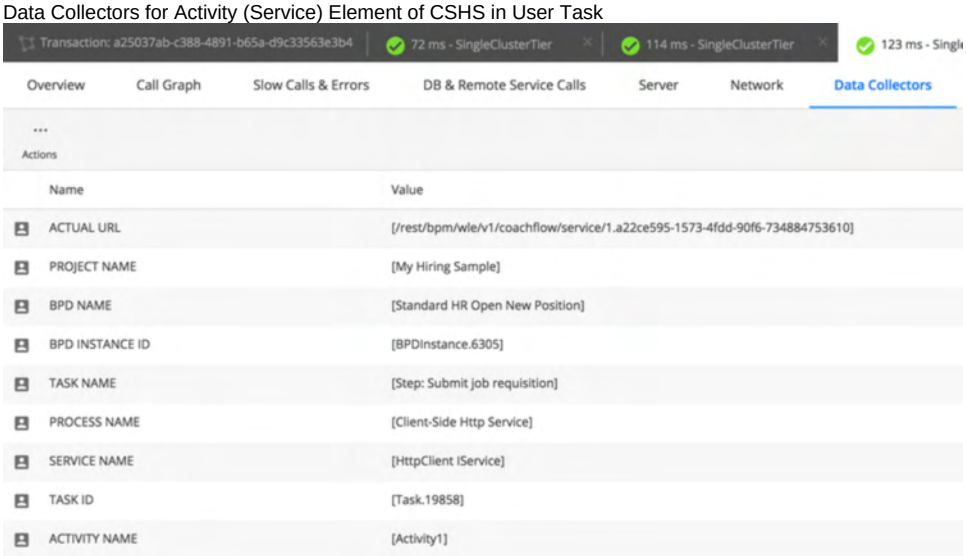
- Data Collectors for System Task



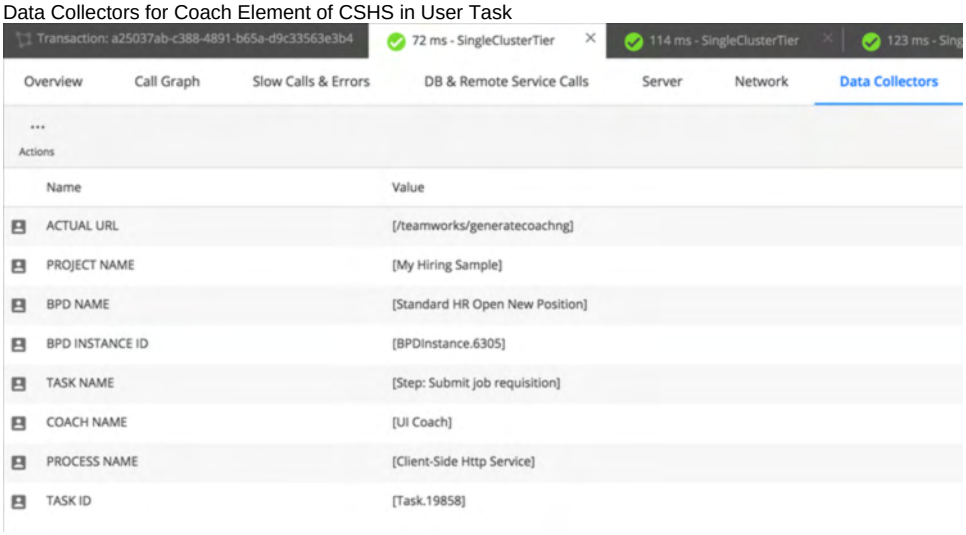
- Data Collectors for Start Event of CSHS in User Task



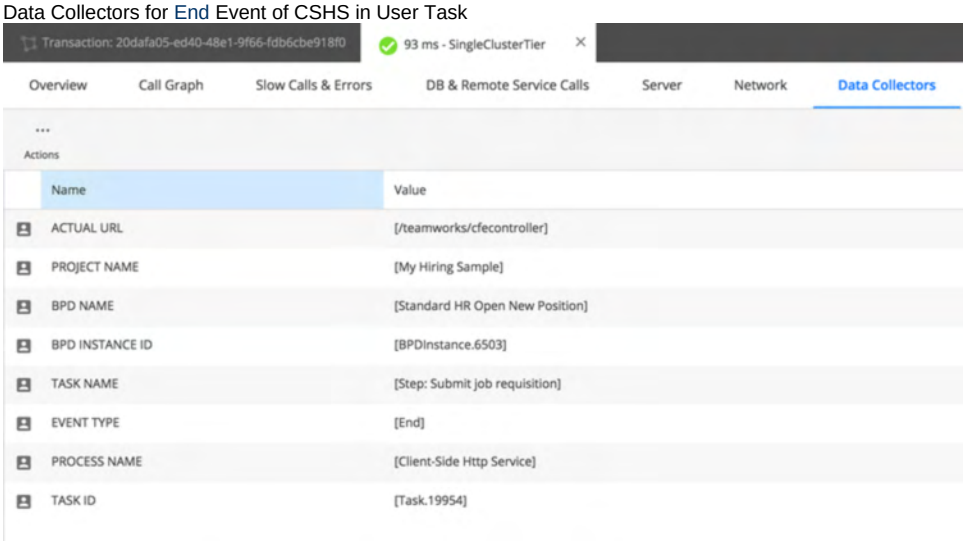
- Data Collectors for Activity (Service) Element of CSHS in User Task



- Data Collectors for Coach Element of CSHS in User Task



- Data Collectors for End Event of CSHS in User Task



Configuration Notes

There are some other configuration settings that may be helpful when implementing AppDynamics for IBM BPM monitoring:

- Increase the value of the node-property `max-business-transactions` if necessary, to accommodate the number of tasks involved in the BPD process (the default = 50).
- Add a node property `framework-support` with a value of "none" to disable CometD and GWT POJO Business Transactions a large number of which otherwise get detected. These correspond to activity within the IBM tooling and are not valuable for BPM monitoring.
- Exclude Servlet Business Transactions like `/portal/login`, `/webasset/..`, which are not valuable for BPM monitoring.

Node Properties That Control IBM BPM Instrumentation

Node Properties available when using IBM-BPM support:

- disable-ibmbpm-usertask-bt-naming: Sets whether Business Transactions naming scheme for IBM-BPM UserTask Business Transactions should be disabled (value = true) or enabled (value = false).
  When it is set to "true", the Business Transactions would be named as per the default URL and not the meaningful names.
- disable-ibmbpm-usertask-bt-in-process-correlation: Sets whether Business Transactions in-process correlation for IBM-BPM UserTask Business Transactions should be disabled (value = true) or enabled (value = false).
- ibmbpm-systemtask-bt-naming: Decides Business Transactions naming scheme for IBM-BPM System Task POJO Business Transactions.
- disable-ibmbpm-data-collectors: Sets whether data-collectors for IBM-BPM task Business Transactions should be disabled (value = true) or enabled (value = false).

Spring Batch Support

Spring Batch is an open-source project for batch processing (execution of a series of jobs).

A Batch Job is composed of one or more Steps. Each step consists of a Tasklet (explicitly or implicitly using ItemReader and ItemWriter). Each level (Job, Step and Tasklet) has an execute method.

Step execution can be parallelized using multiple threads. Remote Chunking (processing of part of Step, chunk on a remote JVM) can also be done with the help of JMS. Different steps in a job can be executed in separate threads use-case permitting.

From an APM perspective, tracking batch workloads at the Job level brings limited value, since batch Jobs are usually very long-running and an alert that the Job overran would come too late for remedial action to be taken. For this reason, the AppDynamics Business Transactions should be placed at the Tasklet level, meaning one batch Job will be composed of many Tasklet Business Transaction executions.

You can turn off spring batch instrumentation using the spring-batch-enabled node property.

Alerting on the response times of these Business transactions allows AppDynamics users to:

- Proactively detect that a Batch Job is progressing more slowly than usual, and thus predict if it will overrun any batch window
- Troubleshoot the root-cause of any unusual slow-downs in the progress of batch jobs in order that the issue can be remediated before the Job overruns

To achieve this, from Java Agent 4.5:

- Creates BTs at Tasklet.execute, named after the Job, Step, and Tasklet
- Correlates chunks across thread boundaries or process boundaries through JMS

OSB Support

Oracle Service Bus (OSB) is Oracle's Enterprise Service Bus (ESB) implementation. It is configuration-based and policy-driven and provides reliable service-oriented integration, service management, and traditional message brokering across heterogeneous IT environments.

This service-infrastructure software adheres to the SOA principles of building coarse-grained, loosely coupled, and standards-based services. Additionally, OSB acts as a message brokering, service monitoring, administration, dynamic routing, and message transformation layer to infrastructure.

OSB relies on Oracle WebLogic Server run-time facilities. OSB supports various service types and transports.

From agent release 4.5.8, AppDynamics supports Business Transaction entries and exits for the HTTP and JMS transports and correlates activity through the broker to give end-to-end transaction visibility. Outbound flows using other transports will be shown as uncorrelated custom backends on the flowmap.

If you want to use custom configuration to influence how the agent instruments non-http and jms outbound transportation, you can use the `ignore-exit-types` node property to disable the default behavior.

For Business Transactions originating in OSB itself, the transaction naming scheme that will be used is dependent on the OSB Proxy Service Type.

Service Type	Naming Scheme
REST, WSDL, SOAP	WebService Entry
Messaging, XML	Servlet Entry

For more details on proxy service types and transports in OSB, see Oracle documentation.

Open Tracing Support

OpenTracing is an open standard for Tracing in-process and out-of-process distributed transactions. At a high level, it performs shallow stitching of method executions involved in a business flow, and hence enables measuring it end to end.

AppDynamics Tracer is an implementation based on OpenTracing 0.31.0 standards.

Prerequisite

For the OpenTracer to work correctly, use the Executor strategy in the Java agent. You can specify it in `app-agent-config.xml` or as node property:

```
<!--As part of app-agent.config.xml-->
<property name="async-instrumentation-strategy" value="executor"/>

Or, just specify the following node property in the Controller:

async-instrumentation-strategy = "executor"
```

Obtain the AppDynamics OpenTracer

Install the Dependency

The OpenTracer jar can be accessed directly, or downloaded from Maven Central, or it can be downloaded from the AppDynamics portal. The library version changes with each new OpenTracer release, and is not tightly coupled to the version of the underlying agent, which must be at least = 4.5.13.

Enable or Disable the Tracer

By default when plugged in, OpenTracer is enabled. There are two ways to disable it:

- Programmatically you can enable and/or disable using:

  ```
  AppdynamicsTracerConfiguration.appdTracingConfiguration().setTracingEnabled();
  ```

  Optionally,

- At startup, you can specify the system property to enable and/or disable using:

  ```
  -Dappdynamics.opentracing.enabled=false
  ```

Plug the Tracer into Lightbend Telemetry

The first OpenTracing use case validated by AppDynamics is its use with Lightbend telemetry (also known as the Cinnamon agent), which allows transaction tracing through applications built on the Lightbend reactive platform, for example those built on Akka HTTP. For more details, refer to the Lightbend documentation, in particular as it relates to OpenTracing integration.

Full instructions as to how to set up Lightbend Telemetry are beyond the scope of the AppDynamics documentation. At a high level, the necessary steps for configuration are:

1. Configure the Lightbend telemetry agent. For Lightbend instructions, see Instructions.
2. Enable tracing for the Akka HTTP endpoints you want the AppDynamics agent to trace. For Lightbend instructions, see OpenTracing Configuration.

   > Ensure the active sampler is set to `const-sampler` in your `cinnamon.opentracing` file. (The low-overhead nature of the AppDynamics Java agent means that you can safely ignore the comment in the sample configuration file that states this should only be used for non-production).

3. Plug the AppDynamics OpenTracing implementation in to Lightbend Telemetry as a custom OpenTracing tracer using the AppdynamicsTracerFactory as described in the Lightbend documentation. A sample AppDynamics tracer factory is provided below:

```
import com.appdynamics.opentracing.core.AppdynamicsTracerFactory;
import com.lightbend.cinnamon.opentracing.TracerFactory;
import io.opentracing.Tracer;

public class AppdynamicsTracerPlugin implements TracerFactory {
    @Override
    public Tracer create() {
        return AppdynamicsTracerFactory.getTracer();
    }
}
```

Development Level Monitoring

Works with:

Development Level Monitoring and AppDynamics

AppDynamics monitors your production applications in a way that is designed to provide maximum visibility with the least amount of overhead. Development level monitoring is a mode in which certain default limits on the data that AppDynamics collects are turned off, giving you additional visibility on application activities.

This default mode is production-level monitoring and applies limits to the retention of certain types of information. In a testing environment or other environments where overhead is not an issue, you can enable development level monitoring. Development mode is intended for temporary use while setting up AppDynamics or for occasional application troubleshooting.

You apply development monitoring to a specific business transaction and originating node combination. An *originating node* is one or more of the nodes that serve as the entry points for the transaction.

Transactions originating on an enabled node are subject to development level monitoring on downstream nodes that participate in processing that business transaction. Note that this applies for that business application only, however. Processing for a continuing transaction in another business application is not monitored at the development level unless you have enabled development level monitoring at the continuing business application as well.

While development monitoring increases the retention of call graphs and SQL statement capture, certain limits still apply. The limits specify maximum thresholds for calls per minute, Java heap utilization, and snapshot segments generated across the Controller. If the thresholds are exceeded, development level monitoring is turned off. See Development Level Monitoring Limits.

Effects of Development Mode

Enabling development level monitoring affects the capture of the following information:

- **Exit Calls—**AppDynamics increases information collected for exit calls to backend systems. For database backends, the agent collects all SQL statements without per-transaction limits. The agent also collects all JDBC and ADO.NET calls attached to methods, even when the call duration is less than 10 ms.
- **Snapshots**—The agent attempts to take a snapshot for every transaction, ignoring the values for these agent node properties:
  - max-concurrent-snapshots
  - on-demand-snapshots

> Development mode does not guarantee snapshots for every transaction since it was not designed to capture data on every request.

- **Call Graphs**—The agent captures full call graphs in development mode.

Users must belong to a role with the Configure Monitoring Level (Production/Development) permission for the application to set development level m
and Manage Custom Roles.

Enable Development Level Monitoring

You enable development level monitoring in the context of a particular business transaction/originating node combination. To avoid exceeding development mode limits described in Development Level Monitoring Limits, it is recommended that you enable development monitoring at the smallest scope possible.

To enable development level monitoring:

1. In the business application for which you want to enable monitoring, click **Configuration** > **Development Level Monitoring**.
2. Enable the **Development Level Monitoring** switch at the top of the page.
   For agents <= 4.3, enabling development level monitoring applies globally, rather than for a particular business transaction. For agents >+ 4.3, you need to configure the business transaction and node to which the configuration applies, as described in the following step.
3. Select the Business Transaction for which you want to enable development monitoring.
4. Click the **Enabled** checkbox next to the originating nodes on which you want to enable development level monitoring.

Turn Off Development Level Monitoring

Even in pre-production environments, enabling development level monitoring would normally be a temporary measure done in select cases. When finished, turn off development level monitoring for the business application, as described here.

To turn off development level monitoring:

1. In the business application for which you want to turn off development level monitoring, click **Configuration** > **Development Level Monitoring**.
2. Use the **Development Level Monitoring** switch at the top of the page to turn off develop level monitoring. The switch turns off such monitoring for all business transactions in the business application.

Development mode may also be turned off automatically.

Development Level Monitoring Limits

The following agent-applied and Controller-applied limits prevent excessive resource consumption during development mode.

App Agent Limits

These limits apply for an App Agent:

- For each node, a maximum of 500 calls per minute. (See dev-mode-suspend-cpm.)
- For each JVM, the maximum heap utilization percentage of 90%. (See heap-storage-monitor-devmode-disable-trigger-pct.)

If a limit is exceeded, an agent log event is generated indicating that the development mode has been turned off because limits were exceeded

Controller Limits

The Controller monitors the number of snapshot segments collected during development monitoring across business transactions. A snapshot segment corresponds to the processing activities for a business transaction on a particular tier. A single business transaction that traverses many tiers can generate more data than several business transactions that only traverse a few tiers.

When the threshold of 5000 snapshot segments per minute is exceeded, the Controller turns off development mode for the top contributing business transactions. The Controller turns off development mode on as many business transactions as are needed to reduce the snapshot segment contribution by 20%.

For example, if a single business transaction is accountable for 25% of the snapshot segments when the threshold is exceeded, only it is turned off. As another example, given the following business transactions and the percentage of snapshot segments, they each contributed when the threshold was exceeded, the first two are turned off:

1. BT1: 17%
2. BT2: 16%
3. BT3: 15%
4. BT4: 15%
5. BT5: 13%
6. BT6: 12%
7. BT7: 12%

You cannot configure this Controller limit.

App Server Agents Supported Environments

This page provides an aggregated view of the supported environments for the App Server agents.

App Server agents have been tested successfully on AWS Outposts using these instructions: Install App Server Agents

Java Agent

This page lists the application environments and versions supported by the AppDynamics Java Agent.

Java Agent Supported Platforms

In the following tables, note that:

- A dash ("-") in a table cell indicates that this column is not relevant or not supported for that particular environment.
- In cases where no version is provided, assume that all versions are supported. Contact AppDynamics Sales for confirmation.
- For environments that require additional configuration, a separate table describing or linking to configuration information follows the support matrix.
- For environments supported by AppDynamics End User Monitoring, see Supported Environments and Versions - Web EUM.
- For environments supported by AppDynamics Server Visibility, Machine Agent Requirements and Supported Environments.

JVM Support

The AppDynamics Java Agent uses the standard JVM Tool Interface (JVMTI) mechanism allowing it to instrument any software running on a JVM supporting this mechanism.

AppDynamics certifies the successful operation of the basic mechanisms of instrumentation used by the agent on the following Java runtimes. These capabilities are supported on both JRE or full JDK installations.

Where the agent supports the following advanced memory monitoring features, they are listed for the JVM: Object Instance Tracking (OIT), Automatic Leak Detection (ALD), Content Inspection (CI), and Access Tracking (AT).

JVM	OS	Memory Monitoring Features
AdoptOpenJDK 8, 9, 10, 11, 12, 13, 14, 15, 16 (supported for both Hotspot and OpenJ9 JVMs)	Linux, Windows, MacOS	OIT (supported only for Hotspot JVM), ALD, CI, AT
Amazon Corretto 8, 11	Linux, Windows	OIT, ALD, CI, AT
Azul Zing 15.x.	Linux x64	OIT, ALD
Azul Zulu OpenJDK 1.6, 1.7, 1.8, 9, 10, 11, 13, 14, 15, 16 JDK11 is supported from 4.5.6 onwards JDK13 is supported from 4.5.15 onwards JDK14 is supported from 20.4.0 onwards JDK15 is supported from 20.10.0 onwards JDK16 is supported from 21.4.0 onwards	Linux, Windows	OIT, ALD, CI, AT
GraalVM 20.0.0, 20.2.0, 21.1.0	Linux, Windows, MacOS	OIT, ALD, CI, AT
HP OpenVMS		

IBM JVM 1.6.x, 1.7.x, 1.8.x	AIX, HP-UX, Linux, Solaris, Windows, z/OS	ALD, CI Object instance tracking, automatic leak detection, and custom memory structure monitoring are not supported with the AppDynamics IBM Java Agent. IBM JVMs can be instrumented with the AppDynamics Sun Java Agent to work around this limitation; however, this only enables automatic leak detection and custom memory structure monitoring. Object instance tracking is not available. Working around this limitation can result in negative performance impact and is not recommended. In such cases, the IBM JVM needs to be restarted to enable custom memory structure monitoring.
Oracle Rockit JVM 28.1+	Linux Intel 64, Windows	
Oracle/BEA JRockit 1.6		
Oracle/Sun JVM 1.6, 1.7, 1.8, 9, 10, 11, 12, 13, 14, 15, 16 JDK11 is supported from 4.5.6 onwards JDK12 is supported from 4.5.11 onwards JDK13 is supported from 4.5.15 onwards JDK14 is supported from 20.4.0 onwards JDK15 is supported from 20.10.0 onwards JDK16 is supported from 21.4.0 onwards	Solaris Sparc 64, Windows, Linux	OIT, ALD, CI, AT Content Inspection and Access Tracking require a JVM restart.
Open Source OpenJDK 1.7, 1.8, 9, 10, 11, 12, 13, 14, 15, 16 OpenJDK11 is supported from 4.5.6 onwards OpenJDK12 is supported from 4.5.11 onwards JDK13 is supported from 4.5.15 onwards JDK14 is supported from 20.4.0 onwards JDK15 is supported from 20.10.0 onwards JDK16 is supported from 21.4.0 onwards	Solaris Sparc 64, Windows, Linux	OIT, ALD
SAP JDK 6+	Windows, Solaris, Linux, HP-UX, i5/OS, AIX	

JVM Application Server and Framework Support

AppDynamics supports the use of the Java Agent to instrument any application component running on a supported JVM, irrespective of how that component is built. The power of the AppDynamics platform is that it can automatically discover the topology and behavior of complex enterprise applications without requiring deep technical knowledge of the application's underlying code.

Frequently, Java-based systems employ standard framework code to implement business logic. Automatic instrumentation of framework code relies on knowledge of the business logic and programming patterns employed by the framework. AppDynamics instrumentation targets processing hand-offs between components, called entry points and exit points, either within the JVM or between JVMs. This includes hand-offs between frameworks in cases where multiple frameworks are being used together. This section covers the capabilities for frameworks for which AppDynamics provides automatic detection rules.

Monitoring application components built using frameworks not listed here may require custom configuration. The custom configuration may involve, for example, custom POJO entry or exit points. If you understand how the application behaves internally, you can easily configure this type of instrumentation. For more complex configuration tasks, contact your account representative to discuss how to engage the AppDynamics customer success organization.

JVM Language Frameworks Support

No additional configuration is required for these frameworks.

Vendor	JVM Language Framework	Version	Correlation/ Entry Points	Exit Points	Transports	Notes
Open Source	Akka Actor	2.1 – 2.5.x	Yes	Yes	Netty	4.3.1 required for 2.4.x 2.5x support includes Persistence Remoting exit/entry supported
Open Source	Akka HTTP Name: `akka-http-stream-entry-enabled` Type: Boolean Default: False	Akka Actor 2.5.x Akka HTTP upto 10.2.4 Scala 2.11, 2.12	Yes	Yes	HTTP	EUM is supported Support for Non-Route DSL
Open Source	Http4s Blaze Client	Blaze versions: 0.21.1, 0.21.0, 0.20.23, 0.20.5 scala 2.11, 2.12	No	Yes	HTTP	
Open Source	Groovy	-	Yes	Yes		
Open Source	Ktor	1.0.x -1.5.x	Yes (Netty Engine)	-	HTTP	EUM is supported
Open Source	Play for Scala Play for Java	2.1 – 2.8 Scala 2.11, 2.12	Yes	-	HTTP over Netty server Akka HTTP server	Includes framework specific entry and exit points Play EUM-APM correlation supported
Open Source	Scala	2.11.6				
Open Source	Spray toolkit (Spray.io)	1.1.x 1.1.3	Yes	Yes	HTTP	Entry points are detected and configurable as servlet entry point and exit points as HTTP exits
Pivotal	Grails	-	-	-	-	

Java Frameworks Support

The Java Agent supports these Java frameworks. Some require additional configuration as indicated in the Configuration Notes column.

Vendor	Framework	Version	SOA protocol (WebServices)	Auto Naming	Entry Points	Exit Points	Detection	Configuration Notes
Adobe	BlazeDS	-	HTTP and JMS adaptor	-	Yes		-	Example Message Queue Backend Configuration
Adobe	ColdFusion	8.x, 9.x	-	-	Yes	-	Configuration required for transaction discovery	Configuration is required for transaction discovery. See: • Java Business Transaction Detection • Servlet Entry Points
Apache	Cassandra with Thrift framework	-	-	-	Yes	Yes	Apache Thrift Entry and Exit points are detected	
Apache	Struts	1.x, 2.x	-	-	Yes		Struts Actions are detected as entry points; struts invocation handler is instrumented	Struts Entry Points

Apache	Tapestry	5	-	-	Yes	-	Not by default	See: • Java Business Transaction Detection • Servlet Entry Points
Apache	Wicket	-	-	No	Yes	-	Not by default	See: • Java Business Transaction Detection • Servlet Entry Points
Apple	WebObjects	5.4.3	HTTP	Yes	Yes	-	Yes	Apple WebObjects Startup Settings
axonframework.org	Axon	2.x, 3.x	-	-	Commands on the Command Bus continue existing Business Transactions	Correlation for Distributed Command Bus on JGroups and for Spring Cloud Connector transport as an exit		
Open Source	CometD	2.6	HTTP	Yes	Yes	-	-	See also "HTTP Exit Points" on Java Backend Detection.
Open Source	Spring Batch	-						Spring Batch Support
Eclipse	RCP (Rich Client Platform)	-	-	-	-	-	-	
Google	Google Web Toolkit (GWT)	2.5.1	HTTP	Yes	Yes	-	-	
JBoss	JBossWS Native Stack	4.x, 5.x	Native Stack	-	-	-	-	
IBM	IBM-BPM	8.5.7, 8.6	-	Yes	Yes	Yes	Yes	IBM-BPM Support
Open Source	Direct Web Remoting (DWR)	-	-	-	-	-	-	
Open Source	Eclipse Vert.x Core	3.3.3-3.5.4, 3.6.x, 3.7.x, 3.8.x, 3.9.x, 4.0.x	HTTP	Yes	Yes	Yes	Yes	EUM Correlation is supported
Open Source	Enterprise Java Beans (EJB)	2.x, 3.x	-	-	Yes	-	-	EJB Entry Points
Open Source	Grails	-	-	-	Yes	-	Not by default	
Open Source	Hibernate JMS Listeners	1.x	-	-	-	-	-	
Open Source	Java Abstract Windowing Toolkit (AWT)	-	-	-	-	-	-	
Open Source	Java Server Faces (JSF)	1.x, 2.x	-	Yes	Yes	-	-	Java Business Transaction Detection and Servlet Entry Points
Open Source	Java Server Pages	2.x	-	Yes	-	-	Yes	Servlet Entry Points
Open Source	Java Servlet API	2.x, 3.0	-	-	-	-	-	
Open Source	Jersey	1.x, 2.x	REST, JAX-RS	Yes	Yes	No	Not by default	JAX-RS Support and node properties: • `rest-num-segments` • `rest-transaction` • `rest-uri-segment-scheme` See App Agent Node Properties Reference for information on the properties.
Open Source	JRuby HTTP	-	-	-	Yes	-	Not by default	See: • Java Business Transaction Detection • Servlet Entry Points
Open Source	Micronaut	1.1.0, 2.5.3	-	Yes	Yes	Yes	By default	-
Open Source	Netty	3.x, 4.x	HTTP	Yes	Yes	Yes	By default	• Node property to disable Netty Instrumentation: `netty-enabled`, by default it is true
Open Source	Spring Annotated Web Services	2.x+	HTTP	Yes	Yes	No	-	

Open Source	Spring WebFlux	5.0, 5.1, 5.2, 5.3	HTTP	Yes	Spring Boot (Netty, Jetty, Tomcat, Undertow)	WebClient (Reactor Netty, Reactive Jetty)	By default	The node property `enable-webclient`, disables the Netty instrumentation and enables WebClient configuration. This node property should not be enabled unless there is some issue or loss in visibility with the OOTB support. By default, the value of this property is false.
Open Source	Spring Cloud Gateway	2.0.x, 2.1.x, 2.2.x, 3.0.2	HTTP	Yes	Yes	Yes	By default	
Open Source	WebSocket	1.0 (Java EE 7, JSR-356)	-	Yes, BT Naming not configurable	Yes, correlation not supported	Yes	Detection is automatic	Node property: websocket-entry-calls-enabled
Oracle	Coherence with Spring Beans	2.x, 3.x	-	-	-	-	-	
Oracle	Swing (GUI)	-	-	-	-	-	-	
Oracle	WebCenter	10.0.2,10.3.0	-	-	-	-	-	
Spring	Spring MVC	5.3	-	-	Yes	-	Not by default	See App Agent Node Properties Reference.

Application Servers

The Java Agent supports the following application servers. Some require additional configuration. Click the link on the server or OSGi Runtime for information about additional requirements or related configuration topics. The agent usually discovers application servers as an entry point.

Vendor	Application Server / OSGi Runtime	Version	SOA Protocol	RMI Supported	JMX	Entry Points	Configuration Notes
Adobe	Cold Fusion	8.x, 9.x	-	No	-	Yes	Requires configuration for transaction discovery; see Servlet Entry Points
	Equinox	-	-	-	-	Yes	OSGi Infrastructure Configuration
Apache	Felix	-	-	-	-	Yes	OSGi Infrastructure Configuration
Apache	Sling	-	-	-	-	Yes	OSGi Infrastructure Configuration
Apache	Tomcat	5.x, 6.x, 7.x, 8.x, 9, 10	-	-	Yes	Yes	Apache Tomcat Startup Settings
Apache	Resin	1.x - 4.x	-	-	-	-	Resin Startup Settings
Eclipse	Jetty	6.x, 7.x, 8x, 9x	-	-	-	-	Jetty Startup Settings
IBM	InfoSphere	8.x	-	-	-	Yes	IBM WebSphere and InfoSphere Startup Settings
IBM	WebSphere	6.1, 7.x, 8.x, 9.x	JAX-WS	Yes, detect and correlate	Yes for WebSphere PMI	Yes	IBM WebSphere and InfoSphere Startup Settings
Open Source	Liferay Portal	-	-	-	-	-	
Open Source	JBoss EAP	7.1.5 and 7.2.0		Yes		Yes	JBoss and Wildfly Startup Settings
Open Source	JBoss Wildfly (formerly JBoss Application Server)	4.x to 23.x		Yes		Yes	JBoss and Wildfly Startup Settings
Sun/Oracle	GlassFish Enterprise Server	2.x	-	-	Yes	Yes	GlassFish Startup Settings
Oracle	GlassFish Server and GlassFish Server Open Source Edition	3.x, 4.x	-	-	Yes for AMX	Yes	GlassFish Startup Settings
Oracle and BEA	WebLogic Server	9.x+	JAX-WS	Yes, detect and correlate for 10.x To enable correlation using a header transported in the `SOAP:Envelope` set node property enable-soap-header-correlation=true	Yes	Yes	Oracle WebLogic Startup Settings
Software AG	webMethods	9.5, 9.6	-	-	-	Yes	webMethods Startup Settings
Tibco	ActiveMatrix BusinessWorks Service Engine	5.x, 6.x	-	To enable correlation using a header transported in the `SOAP:Envelope` set node property `enable-soap-header-correlation=true`	-	Yes	Tibco ActiveMatrix BusinessWorks Service Engine Settings
	Application Server (OC4J)	-	-	Yes, detect and correlate for 10.x	-	Yes	

-	Grails, with Tomcat 7.x, Glassfish v3, Weblogic 12.1.1 (12c)	-	-	-	-		

Servlet 3.x detection is not supported.

PaaS Providers

PaaS Provider	Buildpack
Pivotal Cloud Foundry	Java Buildpack 3.4 and higher See Using AppDynamics with Java Applications on Pivotal Cloud for more information.
Red Hat Openshift 3	JBoss EAP 6.4 and 7.x WildFly 8.1 Docker images For documentation and download information, see the AppDynamics Java APM Agent page on the Red Hat Customer Portal.

Message Oriented Middleware Support

The Java Agent supports the following message oriented middleware environments. Some require additional configuration as indicated in the Configuration Notes column. Message oriented middleware servers are usually found by the Java Agent as an entry point.

Vendor	Messaging Server	Version	Protocol	Correlation /Entry Points	Exit Points	JMX	Configuration Notes
Amazon	Simple Queue Service (SQS)	-	-	Yes (correlation only)	Yes	-	See "Amazon Simple Queue Service Backends" on Java Backend Detection
Amazon	Simple Notification Service (SNS)	-	-	No	Yes	-	See "Amazon Simple Notification Service Backends" on Java Backend Detection
Apache	ActiveMQ	5.x+	JMS 1.x	Yes	Yes	Yes	
Apache	ActiveMQ	5.x+	STOMP	No	-	Yes	
Apache	ActiveMQ	5.8.x+	AMQP 1.0	No	-	Yes	Example Message Queue Backend Configuration
Apache	Axis	1.x, 2.x	JAX-WS	Yes	Yes	-	Default exclude rules exist for Apache Axis, Axis2, and Axis Admin Servlets. See aslo "Web Service Entry Points" on Java Backend Detection.
Apache	Apache CXF	2.1	JAX-WS	Yes	Yes	-	To enable correlation, set node property `enable-soap-header-correlation=true`.
Apache	Kafka	0.9.0.0 to 2.8.0	-	Yes	Yes	Yes	Kafka consumer entry points are disabled by default. Correlation is supported. See Apache Kafka Consumer Backends.
Apache	Synapse	2.1	HTTP	Yes	Yes	-	To enable correlation, set node property `enable-soap-header-correlation=true`
Fiorano	Fiorano MQ		-	-	-	-	
IBM	IBM Web Application Server (WAS)	6.1+, 7.x	Embedded JMS	-	Yes	-	Example Message Queue Backend Configuration
IBM	IBM MQ (formerly IBM WebSphere MQ)	6+	JMS	Yes	Yes	-	Example Message Queue Backend Configuration
Mulesoft	Mule ESB	3.4, 3.6, 3.7, 3.8, 3.9, 4.1.x, 4.2.0, 4.2.1	HTTP, JMS	Yes	Yes	-	Mule ESB Startup Settings
Open Source	Eclipse Vert.x verticles	3.3.x, 3.4.x, 3.5.0, 3.6.0	-	Yes (correlation only)	Yes	-	The Java Agent detects messaging exit calls between verticles.
Open Source	Open MQ	-	-	-	-	-	
Oracle	Java Message Service	2.0	JMS	Correlation of the listener is disabled by default	Yes		
Oracle	Oracle AQ	-	JMS	-	Yes	-	
Oracle	OSB deployed on WebLogic	12.2.1	HTTP JMS	Yes	Yes		OSB Support
Oracle / BEA	WebLogic	9.x+	JMS 1.1	Yes	Yes	Yes	Oracle WebLogic Startup Settings
Progress	SonicMQ	-	-	-	-	-	
Pivotal	RabbitMQ	-	HTTP	-	Yes	-	See "RabbitMQ Backends" on Java Backend Detection

Rabbit	RabbitMQ Spring Client	-	-	Yes	Yes	-	See "RabbitMQ Backends" on Java Backend Detection
Red Hat	HornetQ (formerly JBoss Messaging and JBoss MQ)	-				Yes	
Red Hat	JBoss A-MQ	4.x+	-	-	-	Yes	
Spring	Spring Integration	2.2.0+, 4.0+	JMS	Yes	Yes	Yes	Spring Integration Support See also "Java Message Service Backends" on Java Backend Detection
WSO2	ESB	4.7.0	-	Yes	Yes	-	EUM Correlation is not supported
WSO2	API Microgateway	3.1.x, 3.2.0, 3.2.1	HTTP1	Yes	Yes	-	See WSO2 API Microgateway Startup Settings

JDBC Drivers and Database Servers Support

The Java Agent supports these JDBC driver and database server environments. AppDynamics can follow transactions using these drivers to the designated database.

JDBC Vendor	Driver Version	Driver Type	Database Server	Database Version
Apache	10.9.1.0	Embedded or client	Derby	-
Apache	-	-	Cassandra	-
Progress	DataDirect	data connectivity for ODBC and JBDC driver access, data integration, and SaaS and cloud computing solutions	-	-
IBM	JDBC 3.0 version 3.57.82 or JDBC 4.0 version 4.7.85	DB2 Universal JDBC driver	DB2	9.x
IBM	JDBC 3.0 version 3.66.46 or JDBC 4.0 version 4.16.53	DB2 Universal JDBC driver	DB2	10.1
IBM	-	Type IV	Informix	-
Maria				1.4.x - 2.6.x
Microsoft	4	Type II	MS SQL Server	2012
Oracle MySQL, MySQL Community	5.x, 6.x, 8.x	Type II, Type IV	MySQL	5.x
Oracle	RAC			
Oracle	9i, 10g 11g, 12c, 18c, 19c	Type II, Type IV	Oracle Database	8i+
Open Source PostgreSQL	42.2.5	Type IV	Postgres	8.x, 9.x, 11x
Sybase	jConnect	Type IV	Sybase	-
Teradata			Teradata	-

Notes:

- Type II is a C or OCI driver
- Type IV is a thin database client and is a pure Java driver

NoSQL/Data Grids/Cache Servers Support

The Java Agent supports these NoSQL, data grids and cache server environments. Some require additional configuration. Click the link on the database, data grid or cache name in the following support matrix for information about additional configuration required or related configuration topics.

Vendor	Database/Data Grid /Cache	Version	Correlation/Entry Points	JMX	Configuration Notes
Amazon	DynamoDB	-	Exit Points	-	See "Amazon Web Services" on Java Backend Detection.
Amazon	Simple Storage Service (S3)	-	-	-	"Amazon Simple Storage Service Backends" on Java Backend Detection.
Apache	Casandra • DataStax drivers • Thrift drivers	1.x, 2.x	Correlation for Thrift drivers only	Yes	• "Cassandra Backends" on Java Backend Detection. • For Cassandra server-side support, see Apache Cassandra Startup Settings
Apache	Lucene - Apache Solr	1.4.1	Entry Points	Yes	Apache Solr Startup Settings

Couchbase	Couchbase	3.x	Exit Points	-	See "Couchbase Backends" on Java Backend Detection
JBoss	Cache TreeCache	-	-	-	JBoss Startup Settings
JBoss	Infinispan	5.3.0+	Correlation	-	-
Open Source	Memcached	-	-	-	Memcached Exit Points
Open Source	MongoDB Async Driver	3.4-3.12	-	-	See "MongoDB Backends" on Java Backend Detection
Open Source	MongoDB Sync Driver	3.1-3.12, 4.0.x	-	-	See "MongoDB Backends" on Java Backend Detection
Open Source	MongoDB Reactive Streams Driver	1.3-1.13, 4.0.x	-	-	See "MongoDB Backends" on Java Backend Detection
Oracle	Coherence	3.7.1	Custom-Exit	Yes	Coherence Startup Settings
Red Hat	JBoss DataGrid	-	-	-	JBoss Startup Settings
	JBoss Cache TreeCache	-	-	-	
	JBoss Infinispan	5.3.0+	Correlation	-	
Terracotta	EhCache	-	-	-	EhCache Exit Points

RPC/Web Services API/HTTP Client Support

The Java Agent supports these RPC, web services or API framework types. Some require additional configuration as indicated in the Configuration Notes column.

Vendor	RPC/Web Services API Framework /HTTP Client Support	Version	SOA Protocol-WebServices	Auto Naming	Correlation/Entry Points	Exit Points	Configurable BT Naming Properties	Detection	Configuration Notes
Apache	Apache CXF	2.1	JAX-WS	Yes	Yes	Yes	Yes	Yes	
Apache	Apache HTTP Client	-	HTTPClient (now in Apache HTTP Components)	Yes	Yes (correlation only)	Yes	-	Yes	See "HTTP Backends" on Java Backend Detection
Apache	Apache Async HTTP Client	4.1.x	-	-	-	-	-	-	-
Apache	Ribbon HTTP Client	2.1.0	HTTP Client	Yes	Yes (correlation) Entry - NA	Yes	NA	Yes	
Apache	Apache Thrift	-	-	Yes	Yes	Yes	Yes	Yes	Binary Remoting Entry Points for Apache Thrift
Eclipse	Jetty	8.x, 9.x	HTTP Client	Yes	Yes (correlation only)	Yes (ART supported)	-	Yes	See "HTTP Backends" on Java Backend Detection
Google	gRPC	1.6.x to 1.37.x	RPC	Yes	Yes (Asynchronous)	Yes (Asynchronous)	ServiceName /MethodName	Yes	See Web Service Backend
IBM	WebSphere	6.x, 7.x, 8.x	JAX-RPC	-	-	-	-	-	IBM WebSphere and InfoSphere Startup Settings; also see Default configuration excludes WebSphere classes
IBM	Websphere	7.x, 8.x	IIOP	-	-	-	-	-	IBM WebSphere and InfoSphere Startup Settings; also see Default configuration excludes WebSphere classes
IBM	Websphere	6.1, 7.x, 8.x, 9.x	JAX-WS	Yes	Yes, detect and correlate. To enable correlation using a header transported in the `SOAP:Envelope` set node property `enable-soap-header-correlation=true`	-	Web Service naming	Yes	-

JBoss		7,8,11,16, and 18	JAX-WS	Yes	Yes, detect and correlate. To enable correlation using a header transported in the `SOAP:Envelope` set node property en `able-soap-header-correlation=true`		Web Service naming	Yes	To detect Web Service entry and to support correlation you must create a Servlet exclude rule. See Web Service Entry Points to exclude a rule for JBoss.
Open Source	`java.net.Http`	-	HTTP	Yes	-	Yes	Yes	Yes	See "HTTP Backends" on Java Backend Detection.
Open Source	HTTPClient	0.3-3	Oracle SOA (and potentially others that embed this library)	-	Correlation: Yes; Entry: No	Yes	-	Yes	Oracle WebLogic Startup Settings; also see Default configuration excludes WebLogic classes
Open Source	Grizzly	Grizzly Async HTTP Client (`com.ning.http-client` 1.8.x, 1.9.x, grizzly-http-client 1.1x) • NingAsyncClient v1 with NettyProvider, GrizzlyProvider • NingAsyncClient v2 with NettyProvider	HTTP	-	Correlation: Yes; Entry:No	Yes	-	Yes	
Oracle	GlassFish Metro	-	JAX-WS	-	-	-	-	-	
Oracle	GlassFish Metro with Grails	-	JAX-WS	-	Yes	-	-	Not by Default	
Oracle	JAX-WS RI	2.3.1	JAX-WS	-	To enable correlation using a header transported in the `SOAP:Envelope` set node property en `able-soap-header-correlation=true`	-	Web Service naming	Yes	-
Spring WS	Web Services	3.x, 4.x, and 5.x	HTTP, SOAP	-	To enable correlation using a header transported in the `SOAP:Envelope set node property enable-soap-header-correlation=true`	-	Web Service naming	Yes	-
Oracle	Oracle Application Server	ORMI	-	no	-	-	-	-	
Oracle	WebLogic	10.x	T3, IIOP	Yes	Correlation: Yes; Entry: No	Yes	-	Yes	
Oracle	WebLogic	9.x, 10.x	JAX-RPC	-	-	-	-	-	
Oracle/Sun	Java	11	-	-	-	Yes (ART supported)	-	Yes	
Oracle/Sun	Sun RMI	-	IIOP	-	Not by Default	-	-	-	
Oracle/Sun	Sun RMI	-	JRMP	-	No	Yes	host/port	Yes	
Red Hat	JBoss A-MQ	4.x+	RMI	Yes	Yes	Yes	Yes	Yes	JBoss and Wildfly Startup Settings
Square	OkHttp	2.x, 3.x, 4.x (upto 4.22)	HTTP	Yes	Correlation: Yes Entry: No	Yes	-	Synchronous (2.x, 3.x, and 4.x upto 4.22) and Asynchronous (3.x and 4.x upto 4.22)	
-	Web Services	-	SOAP over HTTP	-	Yes	Yes	-	-	Create Match Rules for Web Services "Web Service Entry Points" on Java Backend Detection
jersey.github.io	Reactive JAX-RS client API	2.25+	HTTP Client	Yes	Yes (correlation) Entry – NA	Yes	NA	Yes	"Web Service Entry Points" on Java Backend Detection

Business Transaction Error Detection

The Java Agent supports the following logging frameworks for business transaction error detection:

- Apache `Log4j` and `Log4j 2`
- `java.util.logging`
- Simple Logging Facade for Java (SLF4J)
  Support for the following method has been added: `public void error(String format, Object... argArray)`
- Logback

To instrument other types of loggers, see Error Detection.

.NET Agent

Supported Runtime Environments

This section lists the environments where the .NET Agent does some automatic discovery after little or no configuration. See Browser RUM Supported Environments for additional supported environments.

OS Versions

- Microsoft Windows Server 2012
- Microsoft Windows Server 2012 R2
- Microsoft Windows Server 2016
- Microsoft Windows Server 2019
- Microsoft Windows 8, 8.1, 10

Microsoft .NET Frameworks

Microsoft .NET Framework 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.2, 4.6, 4.7, and 4.8 on these runtime environments:

- Microsoft IIS 6.0, 7.0, 7.5, 8.0, 8.5, 10
- Managed Windows Services
- Managed Standalone Applications
- Microsoft SharePoint 2010, 2013 as services running inside IIS
- Microsoft ASP.NET Core 2.1 for Windows

Microsoft .NET Core and Microsoft .NET

Microsoft .NET Core 2.1, 3.1, and Microsoft .NET 5.0 with:

- Microsoft ASP.NET Core 2.1, 3.1, and Microsoft .NET 5.0 for Windows for .NET Agent >= 20.3

Microsoft Windows Azure

- Azure App Services for .NET 4.6 environments in the Azure Portal
  - Web Apps
  - Web Jobs
  - API Apps
  - Container Services

  For Azure App Services, the .NET Machine Agent disables certain .NET Machine Agent infrastructure monitoring features: CLR crash reporting, machine snapshots, and Windows performance counter monitoring.

- Azure Cloud Services
  - Web Roles
  - Worker Roles

Unsupported Frameworks

- Microsoft .NET 1.0 and 1.1
- Unmanaged native code

Automatically Discovered Business Transactions

The .NET Agent discovers business transactions for the following frameworks by default. The agent enables detection without additional configuration.

Type	Custom Configuration Options?	Downstream Correlation?

ASP.NET*	Yes	Yes
ASP.NET MVC 2 ASP.NET MVC 3 ASP.NET MVC 4 ASP.NET MVC 5	Yes	Yes
ASP.NET Core on the full framework	Yes	Yes
Open Web Interface for .NET (OWIN) web API	Yes	Yes
.NET Remoting	No	See Enable Correlation for .NET Remoting.
Windows Communication Foundation (WCF)	No	Yes
Web Services including SOAP	No	Yes
Message Queues		
Apache ActiveMQ NMS framework and related MQs	No	Yes
IBM WebSphere MQ	No	Yes
Microsoft Message Queuing (MSMQ)	No	Yes
Microsoft Service Bus / Windows Azure Service Bus	No	Yes
NServiceBus over MSMQ or RabbitMQ transport	No	Yes
RabbitMQ	Yes	Yes
TIBCO Enterprise Message Service	No	Yes
TIBCO Rendezvous	No	Yes
Windows Azure Queue	No	No

***** The .NET Agent automatically discovers entry points for ASP.NET web forms with the Async property set to "true" in the Page directive.

Supported Loggers for the .NET Agent

- Log4Net
- NLog
- System Trace
- Windows Event Log
- Loggers on .NET Core that implement the Microsoft.Extensions.Logging.ILogger API (Linux Agent >= 4.5.19 and Windows Agent >= 21.2.0)

If you are using a different logger, see Error Detection.

Remote Service Detection

The .NET Agent automatically detects these remote service types. The agent enables detection by default. You do not need to perform extra configuration.

Type	Custom Configuration Options?	Async Detection?*	Downstream Correlation?
CosmosDB: • v2.x (Microsoft.Azure.DocumentDB.Core) • v3.x (Microsoft.Azure.Cosmos) (Linux Agent >= 20.9.0, and Windows Agent >= 21.2.0)	No	See Asynchronous Exit Points for .NET.	N/A
Directory Services, including LDAP	No	No	N/A
HTTP	Yes	See Asynchronous Exit Points for .NET.	Yes
MongoDB: C# and .NET MongoDB Driver version 1.10, 2.0	No	See Asynchronous Exit Points for .NET.	N/A
.NET Remoting	Yes	No	See Enable Correlation for .NET Remoting.
WCF	Yes	See Asynchronous Exit Points for .NET.	Yes
WCF Data Services	Yes	No	No

Web Services, including SOAP	Yes	See Asynchronous Exit Points for .NET.	Yes
Azure Service Fabric Remoting v1 and v2—for the .NET Microservices Agent	-	-	-
Data Integration			
Microsoft BizTalk Server 2010, 2013	No	Yes	See Correlation Over Microsoft BizTalk.
Message Queues			
Apache ActiveMQ NMS framework and related MQs	Yes	No	Yes
IBM WebSphere MQ (IBM XMS)	Yes	No	Yes
Microsoft Message Queuing (MSMQ)	Yes	See MSMQ Backends for .NET	See MSMQ Backends for .NET
Microsoft Service Bus / Windows Azure Service Bus	No	Async exit points only	Yes
NServiceBus over MSMQ or RabbitMQ transport	No	See NServiceBus Backends for .NET	Yes
RabbitMQ	See RabbitMQ Backends for .NET	No	Yes
TIBCO Enterprise Message Service	Yes	No	Yes
TIBCO Rendezvous	Yes	No	Yes
Windows Azure Queue	No	No	No

* The agent discovers asynchronous transactions for the Microsoft .NET 4.5 framework. See Asynchronous Exit Points for .NET

Supported Windows Azure Remote Services

Type	Customizable Configuration?	Downstream Correlation?
Azure Blob	No	No
Azure Queue	No	No
Microsoft Service Bus	No	Yes

Cache Clients

Type	Customizable Configuration?	Async Detection?*	AppD for Databases?
StackExchange.Redis	No	Yes	No

Data Storage Detection

The .NET Agent automatically detects the following data storage types. The agent enables detection by default. You do not need to perform extra configuration.

Type	Customizable Configuration?	Async Detection?*	AppD for Databases?
ADO.NET (see supported clients below)	Yes	Yes	No
Windows Azure Blob Storage	No	Yes	No
Windows Azure File Storage	No	Yes	No
Windows Azure Table Storage	No	Yes	No

* The agent discovers asynchronous transactions for the Microsoft .NET 4.5 framework. See Asynchronous Exit Points for .NET

Supported ADO.NET Clients

AppDynamics can monitor any ADO.NET client version and type. Clients we've tested include the following:

Database Name	Database Version	Client Type

Oracle	10, 11, 12	ODP.NET
Oracle	10, 11, 12	Microsoft Provider for Oracle
MySQL	5.x	Connector/Net and ADO.NET
Microsoft SQL Server*	2005, 2008, 2012	ADO.NET

Node.js Agent

This page provides an overview of supported environments for the Node.js Agent.

Node.js Agent Versions

Agent Versions	Node.js Versions
>= 21.1.0	8.6+, 9, 10, 11, 12, 13, 14, 15
>= 20.5.0	8.6+, 9, 10, 11, 12, 13, 14
20.4.0	8.6+, 9, 10, 11, 12, 13
>= 4.5.21	8.6+, 9, 10, 11, 12
>= 4.5.12	6, 7, 8, 9, 10, 11
4.5.11	0.8, 0.10, 0.12, 4, 5, 6, 7, 8, 9, 10, 11

For agent >= 4.5.12, the `npm install` command will stop and print a message for the following scenarios.

- Node.js version less than 6.x:
  - This version of AppDynamics agent supports Node.js versions 6.0 and above.
  - For older versions of Node.js, use the AppDynamics agent 4.5.11 by installing with `npm install appdynamics@4.5.11`.
- Node.js version less than 8.x on Mac:
  - This version of AppDynamics agent on Mac OS supports Node.js >= 8.0.
  - For older versions of Node.js, use the AppDynamics agent 4.5.11 by installing with `npm install appdynamics@4.5.11`.

Operating Systems

- Alpine Linux Docker >= 3.7 on 64-bit platform
- Linux distribution based on glibc 2.5 and later and the x86/x86-64 architecture
- Mac OS X >= 10.8
- Windows Server 2012 R2 and later on 64-bit platform with Node.js >= 0.12.0

Transaction Naming

Entry Type	Default Transaction Naming
Node.js Web	URI

HTTP Exit Points

Supported HTTP Exit Points
Node.js HTTP client library

See Node.js Documentation for the Node.js HTTP client library.

Database Exit Points

Supported Database Exit Points

Cassandra
Couchbase
DynamoDB (using AWS SDK Driver)
Redis
Memcached
MongoDB
MSSQL
MySQL
PostgreSQL

PHP Agent

Related pages:

- PHP Business Transaction Detection

PHP Agent Support

PHP Versions

PHP Agent supports these versions of PHP:

- 5.6
- 7.0
- 7.1
- 7.2
- 7.3
- 7.4

PHP Web Servers

Apache 2.2 and 2.4 in these modes:

- `prefork` mode using `mod_php`
- worker MPM mode using `mod_fastcgi` with `php-fpm` or `mod_fcgid` with php-cgi

Any web server compatible with `php-fpm`.

Operating Systems

- Any Linux distribution based on `glibc 2.5`+ and the x86 32-bit or x86 64-bit architecture
- Mac OS X 10.9+

PHP Agent supports 32-bit operating system only on PHP 5.6.

PHP Frameworks and Protocols

Framework/Protocol	Version	Entry Point Type
Drupal	7	Drupal
Drupal	8	PHP MVC
WordPress	3.4+, 4.x, 5.x	Wordpress
Zend	1, 2, 3	PHP MVC
CodeIgniter	2.x, 3.x, 4.x	PHP MVC
FuelPHP	1.5x, 1.6x, 1.8x	PHP MVC

Magento	1.5, 1.6, 1.7, 2.3	PHP MVC
Symfony	1, 2, 3, 4	PHP MVC
CakePHP	2.x, 3.x, 4.x	PHP MVC
Laravel	5.7, 6, 8	PHP MVC
HTTP		PHP Web
CLI		PHP CLI

If your PHP framework is not listed here, the agent detects your entry points as PHP Web and names the business transactions based on the first two segments of the URI — the default naming convention for PHP Web transactions. So it is still possible to monitor applications on *unsupported* frameworks. Laravel BTs are detected as symfony, as laravel itself is built on top of symfony.

There are few limitations of the PHP Agent. The PHP Agent does not:

- Monitor PHP applications in Zend Thread Safety (ZTS) mode. If you are using ZTS, AppDynamics suggests that you review your dependencies on ZTS to confirm that you actually need it, and if you do not, to switch to the non-ZTS mode
- Support Zend Monitor
- Officially support plug-ins that encrypt and, or obfuscate PHP code such as Zend Guard or ionCUBE Loader
- Support compatibility with the Xdebug module

Transaction Naming

Framework/Environment	Default Transaction Naming
Drupal	page callback name
Wordpress	template name
PHP MVC Frameworks	controller:action
PHP Modular MVC Frameworks	module:controller:action
PHP Web	URI
PHP Web Service	service name.operation name
PHP CLI	last two segments of the script's directory path plus the name of the script

Virtual host prefixing is available for all supported entry point types except PHP CLI.

PaaS Providers

PaaS Provider	Buildpack
Pivotal Cloud Foundry	https://github.com/Appdynamics/php-buildpack See http://docs.pivotal.io/appdynamics/index.html for information about integration with PCF.

Exit Points

Supported HTTP Exit Points
`curl/curl-multi`*
`drupal_http_request()`
`fopen(), file_get_contents()`
`Zend_HTTP_Client::request()`

*The total time reported for a curl/multi_curl request in the Controller is the same as reported by the function `curl_getinfo`. Also, we report the the following execution metrics in the exit call details for the curl/multi_curl request which are included in the total time:

- `namelookup_time`
- `connect_time`

- `pretransfer_time`
- `redirect_time`

Supported Database Exit Points
MySQL old native driver (removed for PHP 7)
MySQLi Extension*
OCI8
PDO
PostgreSQL accessed via PDO and pgsql extensions

* `mysqli_multi_query` is not supported.

Supported Cache Exit Points
Memcache
Memcached
Predis 0.8.5 and 1.1.1, on PHP versions 5.6 and higher
Phpredis 4.1

Although Predis is a full PHP client library, the PHP Agent supports Predis as an exit point only, not as an entry point.

Supported Web Service Exit Points
PHP SOAPClient
NuSOAP 0.9.5

Supported Message Queue Exit Points
RabbitMQ

RabbitMQ support requires the amqp extension.

Opcode Cache Compatibility

Alternative PHP Cache (APC)	

Python Agent

Related pages:

- Python Agent

Python Agent Support

Python Versions

The Python agent supports Python 2.7, 3.4, 3.5, 3.6, 3.7, 3.8, and 3.9.

Operating Systems

The Python agent operates on any Linux distribution based on glibc 2.12+ and the x86 32-bit or x86 64-bit architecture.

- Support for Mac OS X will be deprecated from the 20.7.0 version of the Python Agent.
- Support for centos5 has been deprecated.

Python Frameworks and Protocols

Framework /Protocol	Version	Entry Point Type	Default Transaction Naming
WSGI	1.0	Python Web	First two segments of URI
Tornado	3.2 - 4.5 (all versions), Tornado 5.x without asyncio library, Tornado 6 (for Python 3.5.3+)	Python Web	First two segments of URI

AppDynamics has tested the Python Agent on Tornado, Django, Flask, CherryPy, Bottle, and Pyramid.

You can configure the agent to instrument any WSGI-based application or framework as Python Web, including but not limited to those listed below.

At present, the Python agent fully supports exception detection in Django, Flask, CherryPy, Bottle, Pyramid, and Tornado frameworks. Other WSGI frameworks and custom WSGI applications may install exception handlers that effectively hide some exceptions from the agent. In such cases, the agent will only detect exceptions during exit calls, missed exceptions that are propagated to the WSGI server, and exceptions reported via the custom business transaction API.

WSGI-Based Frameworks	Notes
Bottle	0.12.19
CherryPy	18.6.0
Django	3.1.5
Flask	1.1.2
PasteDeploy	2.1.0
Pyramid	1.10.5
mod_wsgi	4.7.1

Database Exit Points

Supported Database Exit Points	Version
cx_Oracle	5.1.x
MongoDB	3.1+
MySQL-Python	
mysqlclient	
MySQL Connector/Python	
Psycopg 2	
PyMySql	
TorMySql	

HTTP Exit Points

Supported HTTP Exit Points
httplib*
httplib2
requests
urllib
urllib2
urllib3
tornado.httpclient

* The agent detects calls to any external library built on top of `httplib`. Therefore, backend calls to such services, such as, boto, dropbox, python-twitter are detected and displayed as HTTP exit calls.

Cache Exit Points

Supported Cache Exit Points
Memcache
Redis-py

Apache Web Server Agent

Unable to render {include} The included page could not be found.

C++ Agent

C/C++ Supported Platforms

Operating Systems

- Any Linux distribution based on glibc >= 2.5 and the x86 32-bit or x86 64-bit architecture
- Windows Server >= 2012 R2
    - Visual Studio >= 2015

To develop with the C++ SDK, you need Visual Studio >= 2015 . However, the SAP ABAP Agent on Windows leverages the same SDK and does not require development work.

You must install the Visual Studio C++ Redistributable; the full development environment is not required.

Go Language Agent

Go Supported Platforms

Language Support

All Go language versions are currently supported.

Operating Systems

- Any Linux distribution based on `glibc` >= 2.5, and the x86 32-bit or x86 64-bit architecture
- Mac OS X >= 10.8

IIB Agent

IIB Agent Support

This page provides describes the supported versions, operating systems, and node types of the IIB agent. The AppDynamics IIB agent supports the Linux and AIX operating systems.

Operating Systems

The IIB agent supports these operating systems:

- Linux x86-64
- AIX v7.1 and v7.2

IIB Versions

The IIB Agent supports these versions of IIB for the Linux and AIX operating systems:

Operating System	IIB Version
Linux (Any Linux distribution based on glibc >= 2.5)	• IIB v9 • IIB v10 • ACE v11
AIX 7.1 (minimum patch level 7100-03-09-1717 or 7100-04-04-1717)	• IIB v9 • IIB v10
AIX 7.2	• IIB v9 • IIB v10 • ACE v11 (requires ACE Fix Pack >= v11.0.0.9)

IIB Node Types

The agent can continue business transactions detected upstream at these node types:

- SOAPInput
- HTTPInput
- JMSInput
- MQInput

The agent can detect and tag exit calls for downstream correlation at these node types:

- SOAPRequest
- HTTPRequest
- JMSOutput, JMSReply
- MQOutput, MQReply

> For MQ, we use the MQRFH2 message header to provide correlation. Any applications consuming MQ messages from IIB with the IIB agent must support the MQRFH2 header.

The agent can detect database backend calls for these node types:

- DatabaseRetrieve
- DatabaseRoute

All nodes are represented within AppDynamics Business Transactions as Threads. You can view the per node timings in the tree view of the Business Transaction dashboard and in the transaction snapshots.

AppDynamics Universal Agent

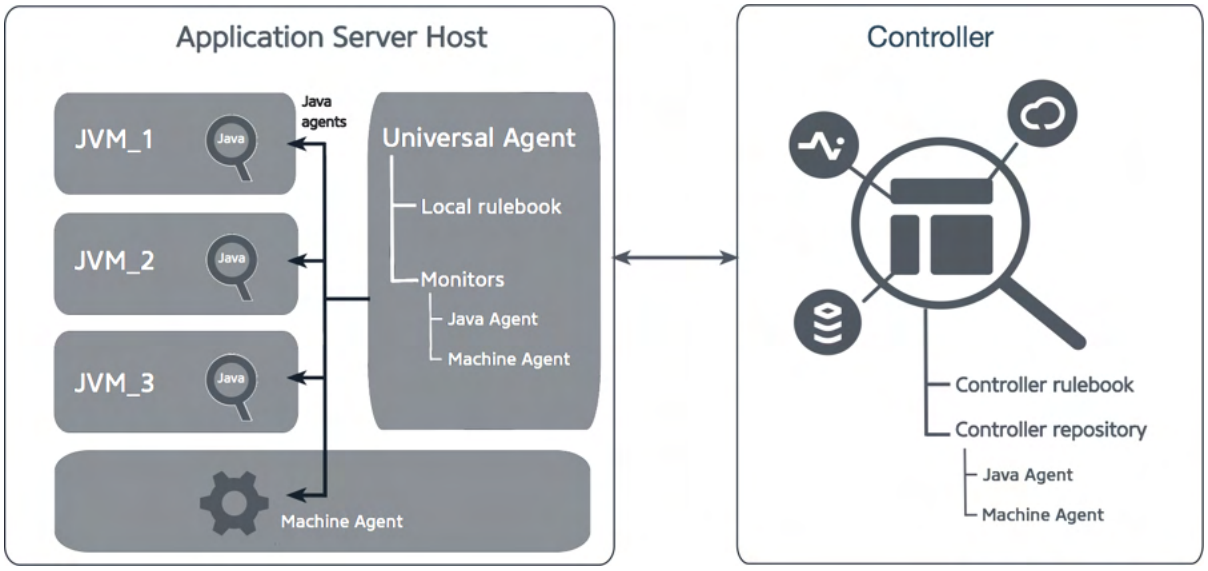
The Universal Agent provides a central management point for administering runtime agents for diverse application environments and eases the management of large environments. The Universal Agent automates the tasks of installing, running, and maintaining AppDynamics runtime agents, such as the Machine Agent and Java Agent.

After you setup the Universal Agent, you can update the versions of the runtime agents by putting new distributions in a central repository and updating the configuration with the latest version. To further ease agent management in large and dynamic environments, the Universal Agent configuration supports remote JVM attachment and dynamic node naming.

Universal Agent

The Universal Agent enables you to run and manage AppDynamics runtime agents by using a single configuration file, called a rulebook. The rulebook determines whether the Universal Agent installs, starts, and stops runtime agents. By default, the Universal Agent runs in controller mode and applies a shared rulebook stored and served by the Controller. Alternatively, you can run the Universal Agent in local mode, in which it uses a rulebook (stored in the Universal Agent home directory) that is specific to the instance of the agent. See Universal Agent Rulebooks.

When the Universal Agent finds a new rule for a runtime agent in the rulebook, it retrieves the runtime agent from a shared repository, installs the agent as a local monitor, and starts the agent. You can manage multiple runtime agents on a single host, which allows for having different versions of the runtime agents for each JVM or each host in your environment.



Set Up Tasks

To use the Universal Agent, you need to do the following tasks:

1. Set up the runtime agent source repository as either a Controller repository or file system repository. A file system repository would typically be on a shared repository accessible to the Universal Agent host machines.
2. Install the Universal Agent on the machines to be monitored.
3. Create the rulebooks to manage your runtime agent instances.

Getting Started

The following procedures walk you through the three set-up tasks for a simple scenario using a single Linux-based application server with an on-premises Controller. On Linux, these steps require `sudo` access to the application server machine, because the setup establishes the Universal Agent as a service.

When following these steps, adjust the values in the commands for your specific Controller, user, account, and so on. Before starting, review additional information in Install the Universal Agent.

Set Up the Runtime Agent Source Repository

To set up a Controller source repository:

1. Confirm that you have an on-premises Controller installed and running and you have access to the Controller machine.
2. On the controller, create the *<controller-home>*`/agent-binaries directory`. This directory serves as your Controller source repository for the runtime agent binaries.
3. Put the Universal Agent ZIP file in the directory.
4. Rename it to `universalagent-`*<version>*`-64bit-linux.zip`.
5. Download the Machine Agent ZIP file to the Controller source repository, *<controller-home>*`/agent_binaries` directory.

6. Rename the Machine Agent file to `machine-<version>-64bit-linux.zip.`
7. Download the Java Agent ZIP to the Controller source repository, <*controller-home*>`/agent_binaries` directory.
8. Rename the Java Agent file to `java-<version>.zip`. (Note there is no bitness in this name.)
9. Continue to the next section to install the Universal Agent.

Install the Universal Agent on the Machines to be Monitored

1. Get the account key for your Controller:
    a. From the Controller UI, open the **License** page and click the Accounts tab.
    b. Click the link to show the key.
2. From the application server machine, get the Universal Agent distribution. You can use the following CURL command to get the Universal Agent from the Controller repository. Replace the `<controller_user>`, <controller_account>, <controller_password>, and <controller_host> address and <primary_port> with values for your system. The Controller user should be an account in the Controller UI that has Universal Agent administration permissions. Replace <account_access_key> with your key and use the version of your specific Universal Agent.

```
curl -u '<controller-user>@<controller-account>:<controller-password>' -X POST -d
'action=downloadAgent&agentVersion=<version>&agentName=universalagent&classifier=64bit-linux' -o
ua<version>.zip 'http://<controller-host:primary-port>/controller/FileDownloadServlet'
mkdir ua_install
unzip ua<version>.zip -d ua_install/
sudo ./ua-install/ua<version>/bin/install.sh --account-access_key <account-access_key>
```

By default, the script installs the agent in the directory, `/opt/appdynamics/universal-agent/`. You can change that using the argument `--target`, but the steps (below) for creating your rulebooks assume the default installation location, `/opt`.

Create the Rulebook

1. From the application server machine, where you installed the Universal Agent, execute a REST API command against your Controller to add a single default rulebook.
Replace <controller-user>, <controller-account>, <controller-password>, <controller-host>, and <controller-port> with the values for your Controller.

```
curl -i -X PUT -su '<controller-user>@<controller-account>:<controller-password>' -H "Content-type:
application/json" -H "Accept:application/json"\
        "http://<controller-host>:<controller-port>/controller/universalagent/v1/user/rulebooks/byName
/default-controller" --data '{
        "name": "default-controller",
        "comments": "An example rule book to monitor Java Agent, Machine Agent and Universal Agent
itself.",
        "rules": [
        {
            "config": {
                "state": "started",
                "version": "<version>"
            },
            "monitor": "machine",
            "comments": "Rule to monitor Machine Agent",
            "condition": "True",
            "name": "Machine monitor"
        },
        {
            "name": "Universal Agent rule",
            "comments": "Rule to monitor Universal Agent",
            "monitor": "universal",
            "config": {
                "version": "<version>",
                "state": "started"
            },
            "condition": "True"
       },
       {
            "config": {
                "version": "<version>",
                "state": "started",
                "application_name": "YourAppName",
                "tier_name": "1stTier"
            },
            "condition": "True"
            "name": "Java Monitor",
            "comments": "Rule to monitor Java Agent",
            "monitor": "java"
       }
    ]
 }'
```

2. After a few moments, verify that the Universal Agent retrieves and installs the Machine Agent and the Java Agent. The corresponding agent files should now appear in the following directories:
    - Machine Agent: /opt/appdynamics/universal-agent/monitor/machine
    - Java Agent: /opt/appdynamics/universal-agent/monitor/java
    - or <target_dir>/appdynamics/universal-agent/monitor/... if you specified `--target <target_dir>` in the installation step
3. Check the status of the Universal Agent as reported to the Controller using the REST API, `universalagent/v1/user/agents/summary`. For instance, using cURL, enter the following command: Replace the username, password, and Controller address with values appropriate for your environment. The response should list the running agents with the version number and the applied rulebook for each.

```
curl -s -X GET -u '<controller-user>@<controller-account>:<controller-password>' -H 'Content-type:
application/json' http://<controller-host>:<controller-port>/controller/universalagent/v1/user/agents
/summary
```

Install the Universal Agent

This page provides an introduction to installing the Universal Agent in AppDynamics.

The AppDynamics Universal Agent runs on each app server machine where you want to deploy runtime agents. The first step in using the Universal Agent to manage your agent deployment, therefore, is installing the Universal Agent on the monitored machines.

When you are ready to install the Universal Agent, review the page specific to your operating system:

- Install the Universal Agent on Windows
- Install the Universal Agent on Linux

Supported Environments and Requirements

The Universal Agent occupies about 20 MB of disk space. When you start the installation, the script first checks the system for sufficient space. Using the Universal Agent to add more runtime agents to your target machine later will require additional space.

The installation process installs the Universal Agent as an automatically started system service. Therefore, you need to install on the system as a user with sufficient privileges for this type of installation. On Linux, for example, you typically need to run the script as a user with sudo privileges. See Permissions for Running the Universal Agent.

The Universal Agent can support the deployment and management of these runtime agents:

- Standalone Machine Agent
- Java Agent
- .NET Agent (Windows only)
- Analytics Agent
- Network Agent (Linux only)

Supported environments and requirements for the runtime agents still apply even when the Universal Agent is managing them. For example, in 4.4, the Network Agent is only available on Linux. Be sure to review the requirements for each runtime agent.

Some Universal Agent features (such as automatic JVM attachment and dynamic rulebook value propagation) have additional requirements and limitations, which are discussed in separate topics.

The Universal Agent is available on the following Linux and Windows versions:

Windows

Installing the Universal Agent on Windows requires the Universal C Runtime. If it the Universal C Runtime is not installed on your Windows server you should install the Update for Universal C Runtime in Windows.

AppDynamics supports the Universal Agent on:

- Windows Server >= 2008 SP2

Linux

Installing the Universal Agent on Linux requires the GNU C Library version 2.12, commonly known as glibc 2.12. The minimum versions of the common distributions that meet this requirement are:

- Ubuntu >= 11.04
- CentOS >= 6
- Red Hat Enterprise Linux >= 6

Controller Connection Settings

The initial configuration for the Universal Agent specifies the Controller host, port, account name, and account key, so you will need to know the values to use before starting. These settings are equivalent to those used by other types of agent. You can find out more in the Agent-to-Controller Connections topic . Note that, on Linux, if you acquire the Universal Agent from the Controller, the settings are preconfigured.

Maintaining Universal Agents

After initial installation, you can update new versions of the Universal Agent itself using the Universal Agent rulebook. See Universal Agent Rules.

Install the Universal Agent on Windows

This page describes how to install the AppDynamics Universal Agent on Windows systems.

Install the AppDynamics Universal Agent on every machine you want to manage runtime agents. When invoked, the installer checks your system for requirements (such as sufficient disk space). When complete, it leaves the agent running and installed it as a Windows service.

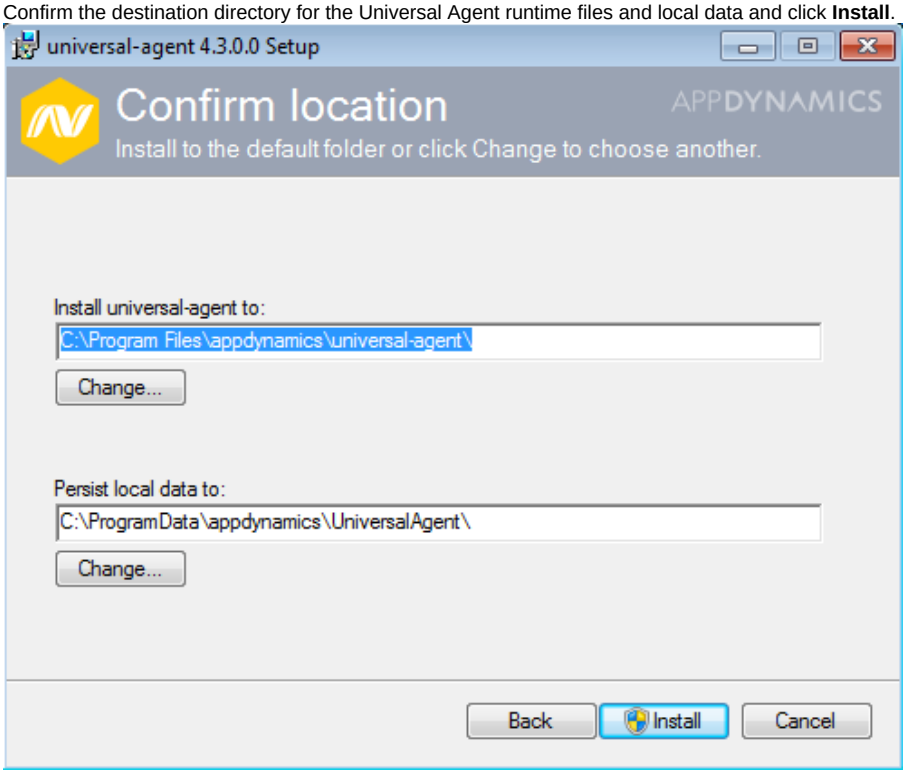
Install the Universal Agent on Microsoft Windows

The two files on the download site for Microsoft Windows installations are MSI and ZIP. The ZIP file is designed to be uploaded to the repository and used for Universal Agent self-upgrade via rulebook rules. To install the Universal Agent on Windows, use the MSI installer.

1. Prepare to install: Installing the Universal Agent on Windows requires the Universal C Runtime. If you have not installed the Universal C Runtime on your Windows server, you should install the Update for Universal C Runtime in Windows.
2. Download the MSI installation package from the AppDynamics download site.
3. Start the installation from the command line as follows. For your package, you must use the path of your MSI file and specify a location for the installation log.

   ```
   msiexec /i "C:\MyPackage\Example.msi" /L*V "C:\log\example.log"
   ```

   The installer starts and the End User Agreement appears.
4. As you review the license information, scroll to the end of the agreement.
5. Select the checkbox that indicates your acceptance of the agreement, and then click **Next** to continue the installation.
6. In the Configure pane, enter the following information and then click **Next**:
   a. **Controller host**: The hostname or IP address for the Controller.
   b. **Controller port**: The primary listening port for the Controller. If not specified, the Universal Agent uses port 80 or 443 (with SSL enabled), by default.
   c. **Account name**: The name and access key for the Controller account where the agents should report data. You can get the Account Name and Access Key value from the License Management page in the Controller UI.
   d. **Account access key**: The name and access key for the Controller account where the agents should report data. You can get the Account Name and Access Key value from the License Management page in the Controller UI.
7. Confirm the destination directory for the Universal Agent runtime files and local data and click **Install**.



   The local data directory will contain the configuration files for the agent. The installer installs and configures the agent.
8. When the installation is complete, click **Finish**.

When complete, the Universal Agent is running and installed as an automatically starting service.

Windows Installer Command Line Arguments

Instead of the GUI installer, you can pass Universal Agent configuration settings to the MSI program from the command line in the following format:

```
msiexec.exe /i <name_of_UA_package>.msi  /qn [ CONTROLLERHOST=<hostname> ] [ CONTROLLERPORT=<primaryport>] [
ACCOUNTNAME=<accountname> ] [ ACCOUNTACCESSKEY=<accountkey>] [ SSL_ENABLED= ]
```

The /qn switch suppresses the UI.

The command arguments are:

- CONTROLLERHOST=<value>
- CONTROLLERPORT=<value>
- ACCOUNTNAME=<value>
- ACCOUNTACCESSKEY=<value>
- CONTROLLERSSLENABLED=<value>

CONTROLLERHOST=<value>

Required. AppDynamics Controller hostname.

CONTROLLERPORT=<value>

Required. AppDynamics controller port number.

ACCOUNTNAME=<value>

Required. Name of the account under which the Universal Agent will report to the controller. If you are running a single tenant controller, use the name of the default, built-in account, `customer1`.

ACCOUNTACCESSKEY=<value>

Required. Account access key for the Universal Agent to authenticate with the controller.

CONTROLLERSSLENABLED=<value>

Optional. Specifies whether or not to use SSL for the connection. If included in the command line and set to any value, the Controller uses SSL. If excluded from the command line, SSL is not required.

- CONTROLLERHOST=<value>
- CONTROLLERPORT=<value>
- ACCOUNTNAME=<value>
- ACCOUNTACCESSKEY=<value>
- CONTROLLERSSLENABLED=<value>

Example

The command line is in the following format:

```
msiexec.exe /i universalagent-setup-4.3.0.0-64bit-windows /qn CONTROLLERHOST=controller.sample.host
CONTROLLERPORT=8090 ACCOUNTNAME=customer1 ACCOUNTACCESSKEY=ABCd-10123-XYZD-0123 CONTROLLERSSLENABLED=1
```

Start and Stop the Universal Agent Windows Service

You can stop and start the Universal Agent from the Windows services manager. Find the Universal Agent service and use the services manager controls to stop and start it.

Uninstall the Universal Agent

To uninstall the Universal Agent, use the Windows MSI program, as follows:

1. Disable auto-java (if you enable it) before uninstalling the Universal Agent. Use the --disable-auto-java command.
2. From the command terminal, enter the following command:

   ```
   msiexec.exe /x <name_of_UA_package>.msi
   ```

3. Follow the MSI tool prompts to complete the uninstallation.

4. Reboot the machine after the uninstall is complete.

Install the Universal Agent on Linux

This page describes how to install the AppDynamics Universal Agent on Linux systems.

About the Linux Installation

To install the Universal Agent on Linux, you use the installation script, `install.sh`. The script accepts configuration parameters in various forms:

- As settings in the `conf/universalagent.yaml` file. (If you download the Universal Agent from the Controller, this file may be preconfigured for your environment.)
- As command-line arguments you enter when invoking the install script.
- As settings in a parameter file named by the `-p` argument.

If the installer does not find the Controller host or port settings in one of these sources, it prompts you for the values at the command line.

The following steps illustrate a simple installation scenario. The procedure assumes you already have an AppDynamics Controller installed and running. When the installation is completed, the installation process leaves the Universal Agent running on the target machine.

Install the Universal Agent on Linux

1. Download the Universal Agent ZIP file to the machine where you want to deploy runtime agents.
    - To download from the controller source repository, use the following command. Substitute the placeholders with your username, account name and password, as indicated. Also, replace `<controller_host>` with your controller hostname or IP address and `<controller_port>` with the primary listening port for your Controller.

      ```
      curl -u <user_name>@<account_name>:<password> \
      -X POST \
      -d 'action=downloadAgent&agentVersion=4.4.0.0&agentName=universalagent&classifier=64bit-linux' \
      -o ua4.4.0.0.zip 'http://<controller_host>:<controller_port>/controller/FileDownloadServlet'
      ```

    - If you are not using the Controller repository, you can directly copy the Universal Agent binary onto the machine where you want to deploy runtime agents.
2. Extract the contents to the agent installation directory. This directory is referred to as *<universal_agent_home>* in these instructions.

   ```
   unzip ua4.4.0.0.zip
   ```

3. Specify the initial configuration settings for the Universal Agent. For example, to configure the settings in the Universal Agent configuration file, follow these steps:
    a. Navigate to the `conf` directory and open `universalagent.yaml` for editing. This file contains the configuration properties for the Universal Agent.
    b. Configure the settings with values that correspond to your Controller environment. The Universal Agent, and unless otherwise configured, the deployed runtime agents, use this information to connect to your Controller:
        - controller_host: The hostname or IP address for the Controller.
        - controller_port: The primary listening port for the Controller. If not specified, the Universal Agent uses port 80 or 443 (with SSL enabled), by default.
        - account_name and account_access_key: The name and access key for the Controller account where the agents should report data. You can get the Account Name and Access Key from the License Management page.
    c. If you want your agent to be registered with a given name, un-comment the `name` tag under the `agent` section and provide a value such as:

       ```
       name: your_agent_name
       ```

       If you do not provide an agent name, the name of the agent defaults to the hostname of the machine.

       > Note that `.yaml` files use a fixed indentation scheme. Therefore, be careful that all the sections are indented correctly. Make sure that the sections `controller` and `agent` are indented correctly with the right number of spaces (not tabs). For example, if you un-comment `account_name, account_access_key` under `controller` section, you need to add a space to make it align with the other tags.
       >
       > Do not use tabs in `universalagent.yaml`. The YAML loader used by the agent does not support tabs.

4. Specify the repository location. This is either the file location or network location where the Universal Agent gets the app agents to install. If using a local repository, see Runtime Agent Repository for instructions on setting up the repository.
5. Review and set any other properties in the configuration file applicable to your environment, as indicated by the inline comments.
6. Run the install script to install the Universal Agent into a Linux or Unix system.

```
sudo ./ua4.5.0.0/bin/install.sh
```

The Universal Agent starts immediately when installation is complete. It is registered as an automatically started service, so the Universal Agent starts automatically upon system reboots.

Linux Install Script Format

Run the install script.

```
install.sh [ -p <param file>] [ --controller_host <controller host>] [ --controller_port <controller port>] [ --account_name <account name>] [ --account_access_key <account access key>] [ --target <target directory>]
```

The following sections describe each setting:

-p <param_file>

Optional. This optional argument specifies the full or partial path name of a text file that contains all of the arguments supported by the script.

When specified, the file must contain the same keywords and values specified on the `install.sh` command. For readability, these can be contained on multiple lines. For example:

Example param file

```
--controller_host localhost
--controller_port 8080
```

A value specified on the command-line of `install.sh`, overrides a value specified in the file referenced by the `-p` argument.

--controller_host <value>

Required. AppDynamics Controller host name.

--controller_port <value>

Required. AppDynamics Controller port number.

--account_name <value>

Required. Name of the account under which the Universal Agent will report to the Controller. If you are running a single-tenant Controller, use the name of the default, built-in account, `customer1`.

--account_access_key <value>

Required. Account access key for the Universal Agent to authenticate with the Controller.

--target <value>

Optional. Specifies the target directory where the Universal Agent is installed. Default is `/opt.`

--no_service

Optional. Specifies that the `ua` daemon should not be defined as a service. In this case, you must manually start the `ua` daemon.

Example

In this example, the Controller host and account access key are found in `paramfile.txt`.

```
ua4.5.0.0/bin/install.sh -p paramfile.txt --controller_port 8081 --account_name customer1
```

Sample contents of the paramfile.txt are:

```
--controller_host localhost --controller_port 8080 --account_access_key 'abcdef$ghi'
```

The Controller port value specified in this file is ignored, because '-controller_port 8081' is specified on the command invocation.

Start and Stop the Universal Agent Linux Service

When the install script completes successfully, the Universal Agent is defined as a system service. You can start and stop it using Linux service commands.

In a `systemd` environment, use the following commands to start, stop, and restart the service:

- `systemctl start appdynamics-universal-agent`
- `systemctl stop appdynamics-universal-agent`
- `systemctl restart appdynamics-universal-agent`

In a non-systemd environment, use the following commands:

- `service appdynamics-universal-agent start`
- `service appdynamics-universal-agent stop`
- `service appdynamics-universal-agent restart`

Uninstall the Universal Agent

To uninstall the agent as a `sudo` user, run the `CLI` command passing the uninstall switch:

```
./ua --uninstall
```

Permissions for Running the Universal Agent

Related Pages:

- Install the Universal Agent

This page provides an overview of permissions needed to run the Universal Agent.

When the Universal Agent installs and starts other runtime agents, it starts them using the same user as the Universal Agent itself. During installation, the default user for running the Universal Agent is set to root. You can create a non-root user, for example, `<universal_agent_user>`, and assign the appropriate permissions to that user.

The installation process installs the Universal Agent as an automatically started system service. Therefore, you need to perform the installation on the system as a user with sufficient privileges for this type of installation. On Linux, for example, you typically need to run the script as a user with `sudo` privileges.

For all environments you can create a specific user with the necessary read/write/execute permissions for running the Universal Agent:

- All files in the `<universal-agent-home>` installation directory should be readable by the Universal Agent.
- The user that runs the Universal Agent must have write privileges to the logging output directory and to the `/conf` directory in the agent installation directory.
- The user that runs the Universal Agent must have write privileges to the `conf` and `logs` directories in the `<universal_agent_home>` directory.
- In addition, the user that runs the Universal Agent needs execute access as described below.

Linux

SystemD

- `systemctl stop`: Stops the Universal Agent service.
- `systemctl restart`: Restarts the Universal Agent after upgrade.
- `systemctl disable` - uninstalls the Universal Agent service.

Non-SystemD

- `service stop`: Stops the Universal Agent service.
- `chkconfig --del`: Uninstalls the Universal Agent service.
- `service restart`: Restarts Universal Agent after upgrade.

Other Commands

- `java` - to start and stop standalone Analytics JVM (usually only on Windows).
- `java -version` - to determine version of Java.
- `sudo -u <user-id> java .../javaagent.jar` - to remote attach to a JVM, if JVM is running with a different user id than the Universal Agent.
- `java .../javaagent.jar` - to remote attach to a JVM, if JVM running with same user id as UA.
- `machine-agent` - invokes machine-agent script to start machine agent.
- `/opt/appdynamics/universal-agent/ua --daemon` - to start the Universal Agent daemon, when it is not defined as a Linux service.

Setting up the Non-root User for Universal Agent

In most Linux installations, you can configure `sudo` ability for the Universal Agent by editing the `/etc/sudoers` file using `visudo`. The following steps provide an example of this configuration change:

1. Edit `/etc/sudoers` using the `visudo` command.
2. Find the line with "`Defaults requiretty`" and change it to "`Defaults !requiretty`".
3. Find the line with "`rootALL=(ALL) ALL`". After this line, add the line "`<user_name> ALL=(ALL) ALL`", where "<*user*>" is the user ID that the Universal Agent service is running under.
4. (For Java Agent Remote Attach) When deploying Java Agents into environments using remote attach, if the Universal Agent runs as root or as the same user that runs the JVMs to which you want to remotely attach, no additional user configuration is required. However, if the Universal Agent runs as a non-root user that is not the same user used to run the target JVM, then you need to authorize the Universal Agent user to use sudo privileges to enable the Universal Agent to retrieve environment variables used in dynamic variable binding.
   At the end of the /etc/sudoers file, add the following line:

   ```
   <ua_user> ALL = NOPASSWD: /opt/appdynamics/universal-agent/ua, /usr/bin/java
   ```

   The value of `<ua_user>` is the user id that the Universal Agent service is running under. Note that `/usr/bin/java` represents the fully-qualified path name for Java on this system. This value can be found by entering the `which java` command, and may be different from `/usr/bin/java`.
5. (For deploying the Network Agent) Installing the Network Agent using the Universal Agent requires elevated privileges for some commands. At the end of the `/etc/sudoers` file, add the following line:

```
<ua_user> ALL = NOPASSWD: /bin/chmod, /bin/chown, /sbin/setcap
```

Note that `/sbin/setcap` represents the fully-qualified path name for `setcap` binary on this system. This value can be found by entering the `which setcap` command, and may be different from `/sbin/setcap`.

Windows

Windows permissions for files and subfolders are inherited by default from the parent folder (<universal_agent_home>). It is good practice to restrict permissions to users authorized to start, stop, and configure the Universal Agent:

- Read and Write permissions to all files and subfolders under `<universal-agent-home>` .
- Permission to install and uninstall software.
- Start, Stop, and Restart permissions for the Universal Agent service. You need admin privileges to install and run the service.

Runtime Agent Repository

The Universal Agent can retrieve the runtime agent installation software from the Controller repository or from a local repository. See Download AppDynamics Software.

This page describes how to create each type of repository.

Configure the Repository Location

The repository types and locations are defined in the Universal Agent configuration file, `universalagent.yaml`, using the `repositories` keyword. You can specify a list of repository URLs (in order of preference) for retrieving AppDynamics monitor (agent) binaries and Universal Agent updates. The format is:

```
repositories:
     - file:///local/path
     - http://host:port/path
     - https://host:port/path
     - https://<username>:<password>@<host>:<port>/<path>
```

To configure a local repository, use the `file://` protocol prefix for the location. For example, if the local repository is in the `/users/appduser/repository` directory, configure the repository location as follows:

```
repositories:
    - file:///Users/appduser/repository
```

Notice that there are three slashes after `file:`. The first two slashes indicate the file protocol, and the third slash refers to the root directory of the file system. All slashes should be forward slashes, even on Windows machines.

To configure a local repository on an HTTP server. Define the URL in any of the formats below:

```
repositories:
    - https://host:port/path
    - http://host:port/path
```

You can also use basic authentication for an HTTP repository in the form:

```
repositories:
    - https://<username>:<password>@<host>:<port>/<path>
    - http://<username>:<password>@<host>:<port>/<path>
```

You can specify more than one repository location, in which case the Universal Agent uses the first agent file that matches a rule that it finds in the sequentially checked repositories. To use a combination of a local and Controller repository, include both, as follows:

```
repositories:
    - file:///Users/appduser/repository
    - https://<username>:<password>@<host>:<port>/<path>
    - controller
```

The Universal Agent first checks the local repository and then the Controller for the runtime agent downloads.

Use a Controller Repository

This type of Universal Agent Repository is not available for SaaS Controllers. Use a local repository instead.

Downloading the .NET Agent from a Controller repository is not supported. Place the .NET Agent distribution file in a local repository.

Using the Controller as the repository location gives the Universal Agents in your environment a single, central point for accessing the runtime agents. To use the Controller repository, you need to be able to access the Controller installation directory.

1. Create a subdirectory named `agent_binaries` in the Controller home directory.
2. Download the runtime agent distribution files to the Controller home subdirectory `agent_binaries` from the download site.

3. Rename the downloaded agents to the format expected by the Universal Agent. Use the following naming conventions where <*version*> is the agent version number:
   `java-<version>.zip` for non-IBM versions of the Java agent
   `java-<version>-ibm.zip` for IBM versions of the Java agent
   For example, `java-4.5.0.0.zip` or `java-4.5.0.0-ibm.zip`
   Your directory should look similar to the following:

```
<appdynamics_home>/
    Controller/
       agent_binaries/
          java-4.5.0.0.zip
          java-4.5.0.0-ibm.zip
          universalagent-4.45-64bit-linux.zip
```

4. Specify the repository location to the Universal Agent, as described in Configure the Repository Location.

Use a Local Repository

If putting the repository in the Controller directory structure is not feasible, you can serve the agent distribution files from a local repository. The likely location for the repository, in this case, would be a shared network directory. The directory would need to be mounted by the machines running the Universal Agent.

The Universal Agent depends upon a defined directory structure within the repository directory and conventionally named runtime agent distribution files, as described in the following steps.

To create your own local Universal Agent repository:

1. Create the `repository` directory with the structure shown in the following example:

```
repository/
   monitor/
      java/
         4.5.0.0
            java-4.5.0.0.zip
         4.5.0.0
            java-4.5.0.0.zip
      machine/
         4.5.0.0
            machine-4.5.0.0-64bit-linux.zip
```

   Your local directory should similarly have a directory named `monitor` that contains subdirectories for the runtime agent types, including `java` and `machine` and their respective versions, as illustrated by the example.
2. Download the agent files to the appropriate location in the directory based on the agent type and version.
3. Rename the downloaded agents to the format expected by the Universal Agent. For example, rename `AppServerAgent-<`*`version`*`>.zip` to `java-<`*`version`*`>.zip`. See Agent Naming Format.
4. Specify the repository location to the Universal Agent, as described in Configure the Repository Location.

Agent Naming Format

The Universal Agent requires the downloaded runtime agent ZIP file in the repository to have a specific name format. When placing runtime agent files in the repository, ensure that they are named using the following format:

Format	Examples	Description
`java-<version>.zip`	`java-4.5.0.0.zip`	Java App Server Agent
`machine-<` *`version`* `>- <` *`bitness`* `>bit- <` *`os`* `>.zip`	`machine-4.5.0.0-64bit-windows.zip`	Machine Agent for 64-bit Windows
	`machine-4.5.0.0-32bit-windows.zip`	Machine Agent for 32-bit Windows
	`machine-4.5.0.0-64bit-linux.zip`	Machine Agent for 64-bit Linux
	`machine-4.5.0.0-32bit-linux.zip`	Machine Agent for 32-bit Linux
`universalagent-<version>-<bitness>bit-<`*`os`*`>.zip`	`universalagent-4.5.0.0-64bit-windows.zip`	Universal Agent for 64-bit Windows

	`universalagent-4.5.0.0-32bit-windows.zip`	Universal Agent for 32-bit Windows
	`universalagent-4.5.0.0-64bit-linux.zip`	Universal Agent for 64-bit Linux
	`universalagent-4.5.0.0-32bit-linux.zip`	Universal Agent for 32-bit Windows
`analytics-agent-<version>.zip`	`analytics-agent-4.5.0.zip`	Analytics Agent
`analytics-agent-bundle-<`*`bitness`*`>bit-<`*`os`*`>-<`*`version`*`>.zip`	`analytics-agent-bundle-64bit-linux-4.5.0.0.zip`	Bundled Analytics Agent
`dotNetAgentSetup<`*`bitness`*`>-<`*`version`*`>.msi`	`dotNetAgentSetup64-4.5.0.0.msi`	Windows .NET Agent
`network-<`*`version`*`>-<`*`bitness`*`>bit-<`*`os`*`>.zip`	`network-4.5.0.0-64bit-linux.zip`	Network Agent

Notice that the Universal Agent and the Machine Agent include the bit number of the target operating system because the downloads bundle the JRE.

Universal Agent CLI

You can perform administrative tasks and configuration changes for a Universal Agent using the Universal Agent command-line interface (CLI), as described on this page.

Run the Universal Agent CLI Tool

You can start the Universal Agent, or administer and configure a running Universal Agent, from the command line by invoking the `/opt/appdynamics/universal-agent/ua` program.

Configuration changes you make with the Universal Agent command-line tool do not impact the content of the Controller rulebook. All rulebook changes affect only the local copy of the rulebook, `local.json`. Command-line arguments that cause rulebook changes also direct the background Universal Agent daemon task to switch to Local mode, so that it operates from the `local.json` rulebook rather than the Controller rulebook.

CLI Tool Command Syntax

The following listing shows the CLI tool syntax:

```
/opt/appdynamics/universal-agent/ua [ --daemon] [ --start-rule <rule_names> ]  [ --stop-rule <rule_names> ] [ --display-mode ] [ --set-mode [local | controller ] [ --modify-config <rule.attribute=value> ] [ --show-java-arguments <process_ids> ] [ --check-config ] [ --show-daemon-status ] [ --setup-win-service [ --data-dir <data-directory-path>] ] [ --restart-service ] [ --start-service ] [ --enable-auto-java ] [ --disable-auto-java ] [ --uninstall ]
```

You can pass more than one argument to the command. For example:

```
/opt/appdynamics/universal-agent/ua --start-rule rule1 rule2  --stop-rule rule3  --display-mode
```

Command Line Arguments

Quotes are necessary if the rule name contains spaces, otherwise, quotes are optional. This is true for all commands that reference rule names.

--daemon

Starts the Universal Agent as a daemon process. It is mutually exclusive with all other command-line arguments.

--start-rule <rule_names>

Changes the state of one or more rules in the rulebook to a value of `started`. The rules to be modified are identified by their rule names. The use of the **--start-rule** argument switches the operation mode of the Universal Agent to Local. You can change the state of multiple rules within a single invocation. For example, `/opt/appdynamics/universal-agent/ua --start-rule rule1 rule2` changes the state of both `rule1` and `rule2` to `started`.

The meaning of this state change varies from one monitoring agent to another.

- Machine Agent and Network Agent: A state value of `started` causes the Universal Agent background task to start the specified agent.
- Java Agent: A value of `started` activates the Java Agent within a matching JVM when that JVM starts. If the JVM has already started, it needs to be restarted for the Java Agent start to take effect.
- .NET Agent: Changing the state of the rule to `started` rules causes the specified rule to become `started` and installs and starts the .NET agent on the machine.

- Analytics Agent
    - For Linux platforms, the Analytics Agent runs as a Machine Agent extension. A value of `started` causes the Machine Agent to be configured to start the analytics extension the next time it is started.
    - For Windows platforms, the Analytics Agent runs in a standalone JVM. A value of `started` causes this JVM to be started if it is not already running.

--stop-rule <rule_names>

Changes the state of one or more rules in the rulebook to a value of `installed`. The rules to be modified are identified by their rule names. Using the `--stop-rule` argument switches the operation mode of the Universal Agent to Local. You can change the state of multiple rules within a single invocation. For example, `/opt/appdynamics/universal-agent/ua --stop-rule rule1 rule2` changes the state of both `rule1` and `rule2` to `installed`.

The meaning of this state change varies from one monitoring agent to another.

- Machine Agent and Network Agent: A state value of `installed` causes the Universal Agent background task to stop the specified agent if it is running.
- Java Agent: The Universal Agent cannot stop a Java Agent because it runs as part of an application JVM. A state value of `installed` prevents the auto-java feature (if enabled) from activating the Java Agent within a matching application JVM.
- .NET Agent: Changing the state of the rule to `installed` stops the .NET agent and uninstalls it.
- Analytics Agent:
    - For Linux platforms, the Analytics Agent runs as a Machine Agent extension. A value of `installed` causes the Machine Agent to be configured to not start the analytics extension the next time it is started.
    - For Windows platforms, Analytics Agent runs in a standalone JVM. A value of `installed` causes this JVM to be stopped, if it is currently running.

--display-mode

Displays the operation mode of the background Universal Agent daemon. It does not change anything.

--set-mode [local | controller]

Changes the operation mode of the background Universal Agent daemon to the specified mode.

If `local` is specified, and the daemon is currently in controller mode, then the current controller rulebook (as saved in the `controller-book.json` file) is renamed `local.json.`

If `controller` is specified, and the daemon is currently in local mode, then the `local.json` file is renamed `local.json.backup` (if possible).

--modify-config <rule_name> <attribute_name>=<new_value>

Changes an arbitrary attribute within the current rulebook to a new value. The use of the `--modify-config` argument switches the operation mode of the Universal Agent to Local.

You can specify the attribute in two ways:

- `<rule_name> <attribute_name>=<new_value>` this form changes a non-config attribute within the rule. `<rule_name>` identifies the name of the rule to be changed; `<attribute_name>` identifies the specific attribute; and `<new_value>` specifies the new value to be assigned to the attribute.
  For example: `/opt/appdynamics/universal-agent/ua --modify-config rule1 condition=False` changes the condition attribute of `rule1` to a value of `False`.
- `<rule_name> config.<attribute_name>=<new_value>` this form changes a config attribute within the rule. A config attribute is a monitor-specific attribute that is defined within the "config" attribute set for the rule. `<rule_name>` identifies the name of the rule to be changed; `<attribute_name>` identifies the specific config attribute; and `<new_value>` specifies the new value to be assigned to the attribute. For example: `/opt/appdynamics/universal-agent/ua --modify-config rule1 config.state=started` has the effect of changing the `state` config attribute of `rule1` to a value of `started`.

--show-java-arguments [<process_ids> | all]

Displays the changes that need to be made to a Java command line to correctly deploy the Java app agent. This command is helpful if you need to manually configure one or more Java startup scripts.

One or more arguments can be provided for this option. If a single `all` argument is provided, then this command displays the command line arguments that need to be defined for each JVM in the system that matches at least one of the Java rules. Otherwise, the arguments following `ua --show-java-arguments` should be the process ids for the processes that should be tested. If a process id represents a Java process, then the process is tested against the current Java monitoring rules. If the process matches at least one rule, the command line arguments that should be added to the process startup script are displayed.

Example:

Example of --show-java-arguments

```
ua --show-java-arguments 123

Java arguments for pid 123
    -javaagent:/opt/appdynamics/universal-agent/monitor/java/javaagent.jar
    -Dappdynamics.agent.applicationName=MyApp,-Dappdynamics.agent.tierName=MyTier,-Dappdynamics.agent.reuse.
nodeName=true,-Dappdynamics.ua.appagent.version=4.3.0.0
```

--check-config

Enables the Universal Agent to validity check the `conf/universalagent.yaml` configuration file for syntax errors. Reports if there are errors in the file that would cause the `ua` daemon process to fail initialization. The Universal Agent install script uses this argument to ensure a valid configuration file prior to installing the Universal Agent as a system service.

--show-daemon-status

Reports the status of the `ua` daemon process by scanning the processes running in the current OS and reporting the process ids of all the processes that are running the Universal Agent as a daemon. Run this command as the root user or by using `sudo`.

Linux only: When the Universal Agent daemon runs as a system service, it is normal for two process ids to be reported

--setup-win-service

Configures the Universal Agent as a service in the Windows environment. This command starts the Windows service (if it is not already running), and configures the registry so that the service starts automatically when the system is rebooted.
The `--data-dir` command-line option can be specified with `--setup-win-service`. The `<data-directory-path>` argument specifies the path name of the directory where the Universal Agent's data directories reside. These directories include `log`, `conf`, `download`, and `rulebook`. If the `--data-dir` option is not specified, then these directories must be located in the main installation directory of the Universal Agent.

--restart-service

Stops and starts the Windows Universal Agent service.

--start-service

Starts the Windows Universal Agent service, unless it is already started.

--enable-auto-java

Usage: Windows only.

Enables the "auto-java" feature. The auto-java feature enables automatic start of the Java app agent for all new JVMs. It directs the Universal Agent to inspect the command line arguments of each new process and add the "`-javaagent:...`" argument to all new Java processes. This option only modifies the command lines for processes that are recognized as JVMs. This option does not directly impact the background Universal Agent daemon.

This is an optional facility that is disabled by default. For Windows systems, this interface can potentially be used to deploy viruses or other types of potentially dangerous software. The intent of the feature is to ease the deployment of the AppDynamics Java agent when it is installed by the Universal Agent, by eliminating the requirement for users to modify Java startup scripts. The code has been thoroughly tested and peer-reviewed, as is the case of all of our product code.

This feature is not supported on Windows 2008 SP2 32-bit and 64-bit.

--disable-auto-java

Disables the auto-java feature that was previously enabled with the `--enable-auto-java` argument. This command-line option does not directly impact the background Universal Agent daemon.

--enable-ldpreload

Usage: Linux only. Exactly like `--enable-auto-java`, but on Linux.

--disable-ldpreload

Usage: Linux only. Exactly like `--disable-auto-java`. Disables the auto-java feature that was previously enabled with the `--enable-ldpreload` argument. This command-line option does not directly impact the background Universal Agent daemon.

--uninstall

Uninstalls the Universal Agent. The Universal Agent Daemon process is stopped and, if the Universal Agent is defined as a service, the service definition is deleted. On Windows, the Universal Agent definition is removed from the registry. Requires a reboot of the machine.

Universal Agent Rulebooks

The Universal Agent uses rulebooks to manage the deployment and maintenance of runtime agents. This page introduces rulebooks and strategies for applying them.

Work with Rulebooks

The Universal Agent operates according to rules that you define in rulebooks. A rulebook is a JSON-formatted configuration file that can direct the Universal Agent to install, stop, or start runtime agents.

The rulebook contains general properties, subject to condition evaluation logic, for the Universal Agent itself, along with rules for individual types of runtime agents.

So for instance, when you add a runtime agent rule to the rulebook, the Universal Agent to which the rule applies retrieves the runtime agent from an agent repository and installs the agent in the `monitor` directory in the Universal Agent home. The `monitor` directory contains the base install directories for each runtime agent.

Operation Mode

The Universal Agent may run in one of two modes: Controller mode or local mode. The mode it uses depends on the rulebooks that it finds in the rulebook directory, `<universal_agent_home>/rulebook`.

The following describes the rulebooks that apply in each mode:

- Local mode: If a rulebook file named `local.json` exists in the directory, the Universal Agent operates in local mode. It does not attempt to contact the Controller and `controller-book.json` is ignored if it exists. You can put the Universal Agent into local mode by using the Universal Agent CLI or by creating `local.json` manually.
- Controller mode: Otherwise, the Universal Agent operates in Controller mode. When in Controller mode, the Universal Agent tries to get the operative rulebook from the Controller. If successful, the Universal Agent writes the contents of the Controller rulebook to a file named `controlle r-book.json` in its own rulebook directory. If the `controller-book.json` file exists in the rulebook directory, from that read or a previous read, then it is used as the rulebook.
  Otherwise, it operates using the `default.json` rulebook .

For information on creating and managing controller rulebooks, see Universal Agent REST APIs.

Polling Interval

The Universal Agent reads the rulebook at regular intervals and applies changes in the rulebook as they occur, reporting the event to the Controller. By default, the Universal Agent checks the rulebook every 300 seconds (five minutes).

For testing and initial investigatory work, you may want to reduce this interval to induce more frequent polling. Use the `interval` property found in `univer salagent.yaml` to change the polling frequency.

Rulebook Structure

The basic parts of a rulebook are shown below in tabular format. The header section tab contains settings that apply globally to all runtime agents controlled by this rulebook. In this example, the values identify the Universal Agent version and the Controller connection settings. Global properties enable you to avoid repeating the property in each rule.

The rules section tab contains rules that define runtime agents, in this case, two Java Agents and one Machine Agent. The header section and each rule can contain a config object. Any value specified in the rule overrides the global value.

Rulebook Property Reference

Header Rulebook Properties

- `name`: Name for this rulebook.
- `comments`: An optional description for this rulebook.
- `config`: This object contains global properties that apply to all runtime agents in the rulebook. The `config` object notably contains connection properties for the Controller. See Controller Connection Properties.
- `rules`: Rules entries contain the specific rules for controlling one or more agents. There are common properties in the agent rules and properties that are specific to the agent type. See Runtime Agent Rule Properties.

The following sections provide more information on the `config` object and `rules` sections of the rulebook.

Controller Connection Properties

The rulebook example above shows properties that runtime agents use to connect to the Controller and how the agent instance is identified in the Controller UI. These properties correspond to values typically configurable for the runtime agents, particularly for the Java app agent configuration file, `con troller-info.xml`.

Using the Universal Agent, you can set any of the usual properties found in the runtime agent configuration file. In the Universal Agent rulebook, the properties have the same names that are in `controller-info.xml`, *except* that a hyphen in the `controller-info.xml` property name is replaced by an underscore in the rulebook version of the property name.

Specifically, the rulebook supports the following properties:

- `account_access_key`
- `account_name`
- `agent_runtime_dir`
- `application_name`
- `controller_host`
- `controller_port`
- `controller_ssl_enabled`
- `credential_store_filename`
- `credential_store_password`
- `enable_orchestration`
- `force_agent_registration`
- `machine_path`
- `node_name`
- `tier_name`
- `use_encrypted_credentials`
- `use_simple_hostname`

For more information on usage, see inline comments in the `controller-info.xml` file and the topic: Administer App Server Agents.

When you set the connection values in the rulebook, the Universal Agent updates the `controller-info.xml` on disk for the runtime agent. The following values are taken from `universalagent.yaml` if they are not specified in the rulebook:

- `controller_host`
- `controller_port`
- `controller_ssl_enabled`
- `account_name`
- `account_access_key`

Specifying the values in the rulebook, however, enables you to configure the Universal Agent and its configured runtime agents to talk to different Controllers, if necessary.

Runtime Agent Rule Properties

The rules section in the Universal Agent rulebook contains rules governing the presence and status of a runtime agent on the Universal Agent host. The default rulebook, `default.json`, is installed with the following predefined rule:

```
"rules": [
    {
        "name": "Universal Agent rule",
        "comments": "Universal Agent rule",
        "monitor": "universal",
        "config": {
            "version": "4.5.0.0",
            "state": "started"
        },
        "condition": "True"
   }
]
```

The properties for each rule are:

- `name`: Name for this rule.
- `comments`: An optional description for this rule.
- `monitor`: The agent type. Valid values are:
    - `java` for the Java Agent
    - `machine` for the Machine Agent
    - `universal` for the Universal Agent
    - `dotnet` for the .NET Agent
    - `analytics` for the Analytics Agent
    - `network` for the Network Agent
- `condition`: A statement that must be true to be applied by a particular Universal Agent. Typically a "condition" tests an environment variable or system property on the host on which the Universal Agent runs.
- `config`: A JSON Object containing agent-level configuration properties such as operating state, node name, and so on. These can be specified per runtime agent in the rule (as illustrated by the state and application_name properties in the sample) or globally (as illustrated by the version property in the header section). Values specified in the rule override the global value. The `config` properties include:
    - `version`: The AppDynamics version of the app agent to use
    - `state`: The action to be applied to the agent, such as `installed`, which installs the agent or `started`, which both installs and starts it. For agent specific state information, see Java Agent Rules and Machine Agent Rules.
    - `application_name`, `tier_name`, and `node_name`: Applicable to Java Agents, these properties are equivalent to application-name, tier-name, and node-name in `controller-info.xml` for the traditional agent configuration. They specify the business application, tier, and node by which the current monitored process is identified in the Controller UI. For additional connection related properties, see Controller Connection Properties. Other runtime agents may have agent-specific properties that can be defined in the rule-level `config` object.

Rulebook Strategies

Use Groups

Universal Agent groups are a way to manage multiple Universal Agents as a logical group. By default, Universal Agents are part of the default group and run the default rulebook, `default-controller`. As additional Universal Agents start and register with the controller, they join the default group.

You can create groups and add Universal Agents to them using the Universal Agent REST API.

A Universal Agent can be part of multiple groups. When the Universal Agent is added to the groups and those groups have different rulebooks, the Controller sends multiple rulebooks to the Universal Agent and the rulebooks are logically merged into a single rulebook.

The resulting merged rulebook is written to `controller-book.json`. You should not modify this rulebook directly.

Conditional Rules

You can use conditions to specify criteria for applying rules. In the simplest (and default) case, the value can simply be set to "true" to enable the rule as shown in the example:

```
"condition": "True"
```

You can also create test conditions that enable or disable the rule based on the environment in the Universal Agent host.

Condition property syntax in Universal Agent rulebooks is similar to conditions in Python—the condition should evaluate to a Boolean expression with identification data keys as the operands (which should match the regular expression "[a-zA-Z0-9_]+").

To install only on Linux machines and given the "platform_system" environment variable, you could specify the following condition:

```
"condition": "platform_system == '\''Linux'\''"
```

Notice that single quotes in the value are escaped using '\'.

In the following example, the condition checks for Linux and a version of the Universal Agent higher than 4.5.1:

```
"condition": "platform_system == '\''Linux'\'' and universalagent_version > '\''4.5.1'\''"
```

Operators you can use are:

- Logical operators: and, or, not
- Comparison operators: ==, !=, <, <=, >, >=
- Special operators:
    - Regular expression compare, where regular expression comes after the operator
    - ~ (AWK-like regular expression match)
    - !~ (AWK-like regular expression inverse match)
- Membership test: in

Instead of using dynamically evaluated conditions, you can put the static values of true or false in the condition property. When used in this way, the condition property gives you a convenient way to enable or disable individual rules.

Configuration Templates

Configuration templates enable you to define a set of default configuration values that apply across a set of rulebooks. See Rulebook Configuration Templates.

Uninstall a Runtime Agent

To uninstall a runtime agent, remove the rule for it in the rulebook and save the file.

The Universal Agent first stops the runtime agent, if it is running, and then uninstalls the runtime agent from the application or machine and removes the directory for it.

Universal Agent Rules

In addition to maintaining runtime agents, the Universal Agent can manage itself through the rulebook.

Define Universal Agent Rules

In a Universal Agent rulebook, a `monitor` value of `universal` identifies a Universal Agent rule.

```
...
"monitor": "universal",
...
```

The state property has the following values for the Universal Agent:

- `installed`: The Universal Agent binaries for the specified version should be downloaded and installed.
- `started`: The Universal Agent binaries for the specified version should be download and installed, and that this version should be running

When you install the Universal Agent, you get a default rulebook that includes a rule for the Universal Agent.

To upgrade the Universal Agent, you simply load the new version of the agent in the repository and increment the version number in the rule. When the Universal Agent detects that the rulebook specifies a different version than the version currently running, it automatically restarts itself with the new version.

Example Universal Agent Rule

The default rulebook includes a default rule for the Universal Agent, as follows:

```
       {
           "name": "Universal Agent rule",
           "comments": "Universal Agent rule",
           "monitor": "universal",
           "config": {
               "version": "4.5.0.0",
               "state": "started"
           },
           "condition": "True"
      }
```

Use Multiple Universal Agent Rules

You can have multiple Universal Agent rules in a rulebook. Each should specify a different name and a different version. Only one rule at a time can specify a state of `started`.

If multiple rules specify a state of `started`, only the first one is recognized, and the other `started` Universal Agent rules are ignored. As with other rules in the rulebook, Universal Agent rules are ignored if the `condition` expression yields a value of `False`.

Standalone Machine Agent Rules

The AppDynamics Universal Agent uses a rulebook to determine which versions of the runtime agents should be installed and deployed. You manage the deployment, versioning, and status of Standalone Machine Agents on a monitored machine by adding Machine Agent rules to the Universal Agent rulebook. This page describes the syntax and usage for Machine Agent rules.

Define Standalone Machine Agent Rules

A `monitor` value of `machine` identifies a Machine Agent rule.

The valid values for the `state` property for a Machine Agent rule are:

- `installed`
- `started`

The started state both installs the agent (if not installed yet) and starts it. The `sim_enabled` property enables Server Visibility for the machine agent. When a value is not specified in the rule, the value defaults to `true`. See Server Visibility for details on licensing and functionality.

The other properties shown in the example, such as `name` and `comments`, are common across the agent types. See Universal Agent Rulebooks.

Example Standalone Machine Agent Rule

The rule containing `"name": "Machine monitor"` specifies the configuration for a Machine Agent.

```
[
{
    "name": "default-controller",
    "comments": "An example rule book with a rule to start a machine monitor",
    "config": {
    },
    "rules": [
        {
            "config": {
                "state": "started",
                "version": "4.5.0.0"
                "sim_enabled": ["true"|"false"]
            },
            "monitor": "machine",
            "comments": "This is a Machine rule",
            "condition": "True",
            "name": "Machine monitor"
        },
        {
            "name": "Universal Agent rule",
            "comments": "Universal Agent rule",
            "monitor": "universal",
            "config": {
                "version": "4.5.0.0",
                "state": "started"
            },
            "condition": "True"
       }
    ]
}
]
```

Java Agent Rules

The AppDynamics Universal Agent uses a rulebook to determine which versions of the runtime agents should be installed and deployed. You manage the deployment, versioning, and status of Java Agents on a monitored machine by adding Java Agent rules to the Universal Agent rulebook. This page describes the syntax and usage for those Java Agent rules.

Define Java Agent Rules

The `monitor` value of `java` in the Universal Agent rule identifies it as a Java Agent rule.

A Java Agent rule contains properties common to all Universal Agent rules, such as the name, description and monitor type of the agent. It also includes properties that are unique to Java Agents.

One example of a Java Agent rule is the following:

```
{
    "name": "Java Rule 1",
    "comments": "Java agent rule",
    "monitor": "java",
    "config":
    {
        "version": "4.5.0.0",
        "state": "started",
        "application_name": "My Application",
        "tier_name": "Commerce Tier",
        "node_name_prefix": "$network_hostname.Commerce",
        "deploy_cmd": ".*-Dnode_type=commerce.*",
        "do_not_deploy_cmd": ".*-Dproduction=true.*"
    }
}
```

The following 'Deployment Options' section provides more information on using the state property. See the Java Agent Rule Syntax Reference for information about the other fields in the example.

Deployment Options

Using the `state` property, you can direct the Universal Agent to deploy the Java Agent using one of the following approaches:

- Remote attach: Universal Agent installs the Java Agent and if `auto-java` is enabled, automatically deploys the Java Agent into the JVM at startup time. If the target JVM is already running, then the Java Agent tries to remotely attach itself to the JVM without forcing a restart, regardless of whether auto-java is enabled.
- At JVM startup: Universal Agent installs the Java Agent and if `auto-java` is enabled, deploys the Java Agent into the JVM at JVM startup. If the JVM is already running, the agent does not try to remotely attach to it, so a restart is required to deploy the Java Agent.
- Install manually: Universal Agent installs the Java Agent, but does not deploy it to a JVM. In this approach, you deploy to a JVM by manually adding the Java Agent to the JVM startup routine.

Attach the Java Agent into Running JVMs

Remote attach enables the Java Agent to load itself into a running JVM, without a JVM restart or modification of the JVM startup commands.

When using this approach, take note of the following points:

- The same requirements apply to using remote attach with the Universal Agent as apply when using it directly with the Java Agent. These requirements include the use of Oracle HotSpot JVM, for example. See Install the Java Agent for more information on these requirements.
- The user under which the Universal Agent runs, must adhere to the following conditions:
    - On Windows, the Universal Agent must run under the Local System account.
    - On Linux, the Universal Agent must run as one of the following users:
        - root
        - the same user that runs the JVMs
        - a user authorized to use sudo. See more information at Permissions for Running the Universal Agent.

Remote attach can affect the performance of the application during the attachment initialization process. Be sure to use this feature only after careful testing in a staging environment.

To attach the Java Agent into running JVMs:

1. Ensure that `tools.jar` is in the `JAVA_HOME/lib/ext` directory on the monitored hosts.
2. In the Java Agent rule of the Universal Agent rulebook, set the value of the `state` property to `attached`.

3. If the Universal Agent is not running as `root` or as the same user that runs the JVM, authorize the user for `sudo` privileges, as described in Permissions for Running the Universal Agent.
4. Specify conditions for attaching the Java Agent to your JVMs by using the `deploy_cmd` and `deploy_env_vars` properties in the Java Agent rule. For example, you can set conditions on installation based on startup commands that match Tomcat startup commands.
5. Save the rulebook. The Universal Agent applies the rule change the next time it reads the active configuration.

Install the Java Agent Automatically at JVM Startup

When you set the `state` to `started`, the Universal Agent examines each new process started on the host. If the process is a new JVM, the Universal Agent injects the javaagent argument into the startup arguments for the process. With this approach, you do not need to modify the Java startup script and it is supported on all JVMs types.

Since this affects the JVM startup process, JVMs that are already running will not be affected. Also, the Java Agent version must be 4.2.3 or later.

> AppDynamics Java Agent versions 4.2.6.x and 4.2.7.x include an issue that prevents the Universal Agent from being able to install those versions directly. The issue was addressed in Java Agent version 4.2.8.
>
> If you are using version 4.2.6.x or 4.2.7.x, you can work around this incompatibility by having the Universal Agent first install Java Agent 4.2.8, or a later version. You do not need to deploy the later version to your application; simply create a rule for it with a `state` value of `installed`, while specifying `attached` or `started` as the state for the 4.2.6.x or 4.2.7.x agent rules.

If you are testing this feature on Linux, keep in mind, this feature relies on the Linux `LD_PRELOAD` facility, along with environment variables set in the JVM startup environment. Command shells that are open before you have enabled `auto-java` may not have the environment variables properly set.

On Windows, the auto-java feature is analogous to the LD_PRELOAD feature on Linux. When the auto-java feature is enabled, the Universal Agent gains visibility to the command line arguments for any new JVM process being started and adds two new arguments to the command line as follows:

- `-javaagent:`<FQPN_`javaagent.jar`>, where <FQPN_javaagent.jar> is the fully-qualified path name of the `javaagent.jar` file, as installed by the Universal Agent. The addition of this command line argument causes the Java agent to be loaded into the new JVM
- `-Dappdynamics.ua.rule.dir=`<FQPN_UA_ `rule_ dir`>, where <FQPN_UA_ rule_ dir> is the fully-qualified path name of the directory containing the Universal Agent rulebook.

If the Java command line arguments already contain the `-javaagent` argument, referring to `javaagent.jar`, or `javaagent.jar` cannot be located, then the Universal Agent does not modify the Java command line.

To insert the Java Agent into the JVM at JVM start up:

1. Specify `started` as the value of the `state` property for the matching Java agent rule in the rulebook:
    a. Set the `state` property for the matching Java agent rule in the rulebook to started.
    b. When using the "started" (or auto-java) mode, it is likely that you will want to specify conditions for attaching the Java Agent to JVMs. For example, you can set conditions on installation based on startup commands that match Tomcat startup commands. Use the `deploy_cmd` and `deploy_env_vars` configuration attributes in the Java Agent rule for this type of control.
2. Enable the `auto-java` feature by executing the following command from a command prompt:

    ```
    ua --enable-auto-java
    ```

3. When you save the rulebook, the Universal Agent applies the rule change the next time it reads the active configuration.

See Universal Agent CLI for more details on the auto-java command line arguments.

Install the Java Agent Manually

Instead of relying on the Universal Agent to install the Java Agent into JVMs, you can modify the startup scripts for the JVM to inject the Java Agent. This is the traditional method for installing Java Agents, as described in Install the Java Agent. However, by using the Universal Agent, you can rely on it to manage the download and maintenance of the agent on the host.

The Universal Agent CLI tool lets you inspect the arguments for a given Java process. You can use this to determine any other command line options that need to be added in addition to the `-javaagent` option. Use the CLI command `ua --show-java-arguments` <`pid`> argument—where <`pid`> is the process ID of the target application JVM—to see what command line options are required.

To install the Java Agent but inject into JVMs manually:

1. Set the value of the `state` property for the matching Java agent rule in the rulebook to `installed`.
2. Add the Java Agent JAR file to the startup routine of the JVM, as described for your Java framework type in Install the Java Agent. The agent JAR should be the Java Agent located in the <`ua_home`>`/monitor/java directory`.

Java Agent Rule Matching

A rulebook can have multiple Java Agent rules. When applying rules, the Universal Agent applies the first rule that matches the JVM, ignoring other possibly matching rules.

The `condition` property also affects rule selection. If the evaluation of `condition` yields a value of `False`, the rule is not applied for a JVM.

Because a single rule can match multiple JVMs, you should use the auto-node naming feature to ensure that each matching JVM is given a unique node name. To do this, do not include the `node_name` property in the rule. Instead, either provide a `node_name_prefix` property to provide a node name prefix, or omit the `node_name_prefix` property to enable the Controller to generate a node name using the default prefix. In either case, a unique node name is generated for the JVM.

Java Agent Rule Syntax Reference

The syntax of the Java Agent rule is illustrated in the following rulebook entry:

```
{
    "name": "<name of rule>",
    "comments": "<comments>",
    "monitor": "java",
    "config":
    {
        "version": "<Java agent version>",
        "state": "installed | started | attached",
        "controller_host": "<name of host running controller>",
        "controller_port": "<controller primary port>",
        "account_name": "<controller account name>",
        "account_access_key": "<controller account access key>",
        "application_name": "<business application>",
        "tier_name": "<tier name>",
        "node_name": "<node name>",
        "node_name_prefix": "<node name prefix for auto-naming>",
        "deploy_cmd": "<regular expression matching Java command line>",
        "deploy_env_vars": "<regular expression matching JVM environmental variables>",
        "do_not_deploy_cmd": "<regular expression matching Java command line to be excluded>",
        "crash_age_threshold_days": "<prevents installation to JVMs with recent crash logs>",
                "runtime_directory": "...identifies the runtime directory for the Java agent...",
        "additional_props": "...additional properties to be passed to the JVM...",
        "kind": "...indicates whether or not IBM version of Java agent is to be installed..."
    },
    "condition": "<rule status>" "...boolean expression indicating status of this rule..."
}
```

The name, comments, monitor, and condition properties are common for all Universal Agent monitor rules, and have no special meaning for the Java Agent. See Universal Agent Rulebooks for information on these values.

The `config` object contains properties specific to the Java Agent. The following table describes the config properties.

Keyword	Description	Example
`version`	The version of the Java Agent to run.	`"version" : "4.5.0.0"`
`state`	Indicates how the Java Agent deploys itself into the target JVM. Also related to the auto-java feature. Valid values are installed, started and attached. For more information, see Deployment Options.	`"state": "started"`
`controller_host`	The name of the controller host where the Java Agent should connect. Optional. Can be inherited from the `controller_host` property defined in the header section of the rulebook or from the Controller identified in the `universalagent.yaml` file.	`"controller_host": "localhost"`
`controller_port`	The Contoller port number where the Java Agent should connect. Optional. Can be inherited from the `controller_host` property defined in the header section of the rulebook or from the Controller identified in the `universalagent.yaml` file.	`"controller_port": "8080"`
`account_name`	The account name that is passed to the Controller when the Java Agent attempts to connect. Optional. Can be inherited from the `controller_host` property defined in the header section of the rulebook or from the Controller identified in the `universalagent.yaml` file.	`"account_name": "customer1"`
`account_access_key`	The account access key that is passed to the Controller when the Java Agent attempts to connect. Optional. Can be inherited from the `controller_host` property defined in the header section of the rulebook or from the Controller identified in the `universalagent.yaml` file.	`"account_access_key": "APJC234bcd$123"`
`application_name`	The application name to be provided by the Java Agent when it connects to the controller.	`"application_name"`

	Optional. Can be inherited from the header section of the rulebook defined in a global `config` object or from the Controller identified in the `universalagent.yaml` file.	`: "ACME Book Store Application"`
`tier_name`	The tier name to be provided by the Java Agent when it connects to the controller. Optional. Can be inherited from the header section of the rulebook defined in a global `config` object or from the Controller identified in the `universalagent.yaml` file.	`"tier_name": "ECommerce Server"`
`node_name`	The node name to be provided by the Java Agent when it connects to the controller. Optional. If this property is missing and the `node_name_prefix` property is provided, then the `node_name_prefix` is used to name the node of the JVM. If both the `node_name` and the `node_name_prefix` properties are missing, then the auto-node name feature is used to name the node. The controller provides a unique name for the node, using the tier name as a prefix for the node name.	`"node_name": "Node_8000"`
`node_name_prefix`	Indicates that the Java Agent should use the "auto node name" feature to define a unique name dynamically for the JVM. The value of the "node_name_prefix" property specifies a prefix for the generated node name; the controller supplies a suffix to make the node name unique. If the "node_name" property is specified, then the "node_name_prefix" property is ignored, and the "node_name" property explicitly names the node. In the example Java Agent rule, "$network_hostname.Commerce" serves as a prefix. Notice that "$network_hostname" refers to a Universal Agent environment value, which the Universal Agent binds to a value at runtime. (Environment variable references always begin with '$'). If both the "node_name" or the "node_name_prefix" properties are missing, then the "auto node name" feature is used to name the node. The controller provides a unique name for the node, using the tier name as a prefix for the node name.	`"node_name_prefix": "Commerce"`
`unique_host_id`	The value of the "appdynamics.agent.uniqueHostId" property. Optional. Can be inherited from the header section of the rulebook (defined in a global "config" object) or from the Controller identified in the `universalagent.yaml` file.	`"unique_host_id": "cart-machine"`
`deploy_cmd`	A regular expression that identifies the JVMs that the rule applies to. This regular expression is matched against the command line arguments that are used to start the JVM. If the expression matches the command line, then this rule is considered a candidate for deploying the Java Agent within the JVM. If the `deploy_env_vars` property is also provided, then the value of that property must also match the set of environmental variables associated with the JVM. If the `do_not_deploy_cmd` property is also provided, then JVMs whose command line arguments match the `deploy_cmd` are not considered candidates for deployment if the command line also matches the value of the `do_not_deploy_cmd` property. In the example Java Agent rule, the `deploy-cmd` regular expression is looking for a Java property "`node_type=commerce`". If the command line for a starting JVM matches the regular expression, it is considered a candidate for the rule. (However, because the `do_not_deploy_cmd` property is also specified in this example, the rule is not applied to a JVM if it's command line matches the value of `deploy_cmd` but also matches the value of `do_not_deploy_cmd`.	`"deploy_cmd": ".*/TIER1TOMCAT/.*"`
`deploy_env_vars`	A regular expression that identifies the JVMs that the rule applies to. The set of environmental variables associated with the JVM is formed into a string, consisting of "key=value" pairs, delimited by blanks, and sorted alphabetically based on the key name. If the resulting string matches the regular expression defined by the "deploy_env_vars" property, then this rule is considered a candidate for deploying the Java Agent within the JVM. If this property is not provided, then the contents of the environmental variables are not used to select JVM(s) for deployment. If the "deploy_cmd" property is also provided, then the value of that property must also match the command line used to start the JVM. If the "do_not_deploy_cmd" property is also provided, then JVMs whose command line arguments match the "deploy_cmd" property are not considered candidates for deployment if the command line also matches the value of the "do_not_deploy_cmd" property.	`"deploy_env_vars": ".*SHOULD_APPLY=true.*"`
`do_not_deploy_cmd`	A regular expression that is used to exclude JVMs that the rule should not apply to. This regular expression is matched against the command line arguments being used to start the JVM. If the expression matches the command line, then this rule is explicitly excluded from being considered a candidate for deploying the Java Agent within the JVM. In the example Java Agent rule, the regular expression will match those Java command lines that contain "-Dproduction=true". JVMs whose command lines match this expression are excluded from deployment for this rule.	`"do_not_deploy_cmd": ".*-Ddo_not_apply=true.*"`
`crash_age_threshold_days`	Indicates that the Universal Agent should not remotely attach itself to a JVM where there are one or more recent Java crash log files. The value of this property represents the number of days since the crash log file was created in order to be considered 'recent'. For example, if a value of 10 is specified for this property, then only crash log files created in the last 10 days prevent the Universal Agent from attempting remote attach. Crash logs older than 10 days do not prevent the attempted attach.	

	If this property is not specified, then the Universal Agent does not look for recent Java crash log files, and does not use their presence to prevent remote attach attempts.	
`additional_props`	A comma-separated list of key=value pairs providing additional property definitions to be passed to the JVM. Passed to the JVM if the value of the **state** keyword is specified as either "started" or "attached".	`"additional_props": "appdynamic.agent.uniqueHostId=host"`
`kind`	Use this property when the IBM version of the Java agent is being deployed. The value should be "ibm". If the property is missing, then the non-IBM version of the agent is deployed.	`"kind": "ibm"`

Dynamic Configuration Values

Configuration settings can be set to static or dynamic values. Dynamic values may be bound to environment variables or system properties from the target application JVM itself. This lets you name nodes, tiers, or applications based on JVM-specific variables. This is particularly useful for elastic environments, in which node or tier identity cannot be known in advance.

To reference dynamic elements in rules, use the following format:

- For Java environmental variables, prefix the variable name with `$javaenv_`;
- For system properties, use the syntax `${property-name}`, where `property-name` is the name of the Java system property.

When determining whether or not a Java rule applies to a particular JVM, the Universal Agent extracts the values of each environment variable, and perform symbolic substitution into the config values in the rule for each `$javaenv_` reference. The Universal Agent does not resolve Java system properties within the configuration rule. Instead, the references are resolved by the Java Agent itself when it starts running within the JVM.

Example

The following shows a rule with sample dynamic values:

```
{
    "name": "Java rule",
    "comments": "...comments...",
    "monitor": "java",
    "config":
    {
        "version": "4.5.0.0",
        "state": "attached",
        "application_name": "$javaenv_APP_NAME",
        "tier_name": "${tierName}",
        "node_name": "$javaenv_APP_NAME-$javaenv_NODE_NAME",
        "deploy_cmd": "...regular expression matching Java command line..."
    },
    "condition": "...boolean expression indicating status of rule..."
}
```

In this example, several of the properties have values referencing either JVM environment variables or system properties:

- `application_name: $javaenv_APP_NAME` indicates that the value of the `application_name` config property is the value of the `APP_NAME` environmental variable within a JVM for which the rule applies.
- `tier_name: ${tierName}` indicates that the value of the `tier_name` config property is the value of the `tierName` system property within a JVM for which the rule applies.
- `node_name: $javaenv_NODE_NAME` indicates that the value of the `node_name` config property is the value of the `APP_NAME` environmental variable within a JVM for which the rule applies, concatenated with a '-', followed by the value of the `NODE_NAME` environmental variable within the JVM.

Suppose a JVM is started with the following commands:

```
export APP_NAME=MyApplication
export NODE_NAME=MyNode


java -DtierName=MyTier  -jar app.jar
```

If matched to the sample rule, the result would the following configuration bindings for the JVM:

- `"application_name": "MyApplication"`

- `"tier_name": "MyTier"`
- `"node_name": "MyApplication-MyNode"`

Notes on Dynamic Configuration

- References to Java system properties are not resolved by the Universal Agent when evaluating rule conditions, since the property values from the JVM are not available to the Universal Agent.
- For environment variable substitution, if a Java Agent is introduced into a JVM at startup time (as opposed to remote attach), the `javaagent.jar` file must be version 4.3 or later. Note that the Java Agent must be installed by the Universal Agent, but it does not need to be deployed—installing the Java Agent causes the supporting `javaagent.jar` file to be loaded into each JVM.

.NET Agent Rules

You can install, maintain, and run .NET Agents with the AppDynamics Universal Agent. The Universal Agent uses a rulebook to determine which versions of the .NET Agent should be installed and deployed. You need to define and configure the .NET Agent entry in the Universal Agent rulebook. See Universal Agent Rulebooks.

Define .NET Agent Rules

A `monitor` value of `dotnet` identifies a .NET Agent rule.

The valid values for the `state` property for a .NET Agent rule are:

- `installed`
- `started`

.NET Agent Rule Syntax

```
{
   "condition":"platform_family == 'windows'",
   "comments":"...comments...",
   "name":"...name of rule...",
   "monitor":"dotnet",
   "config":{
      "version":"....NET agent version...",
      "state":"installed | started",
      "controller_host":"...name of host running controller...",
      "controller_port":"...port number controller is listening on...",
      "account_name":"...controller account name...",
      "account_access_key":"controller account access key...",
      "controller_ssl_enabled":"true",
      "controller_tls12_enabled":"false",
      "controller_secure_enabled":"false",
      "config_xml":[
         "list of repositories where a paired config.xml can be found"
      ]
   }
}
```

The following properties are defined for the Universal Agent and common to all agent types. The values have no special meaning for the .NET agent:

- `name`: A string for quickly identifying the rule.
- `comments`: A long string describing the rule's purpose.
- `monitor`: Name of the targeting monitoring agent, for .NET Agent, must be dotnet.
- `condition`: The conditions for applying the rule. This consists of a Boolean expression where the operands are the values collected by the Environment Modules.

Keyword	Description	Example
`version`	The version of the .NET Agent to run.	`"version": "4.5.0.0"`
`state`	Indicates how the .NET Agent should deploy on the target machine. • "installed": downloads the specified version of the .NET Agent • "started": a version of the .NET Agent is installed and is active	`"state": "started"`
`controller_host`	Identifies the name of the Controller host where the agent should connect. This property is optional in the rule; it can be inherited from the controller_host property defined in the header section of the rulebook or from the Controller identified in the `universalagent.yaml` file.	"controller_host": "localhost"
`controller_port`	Identifies the port number of the Controller where the agent should connect. This property is optional in the rule; it can be inherited from the controller_host property defined in the header section of the rulebook or from the Controller identified in the `universalagent.yaml` file.	"controller_port": "8080"
`account_name`	Identifies the account name that is passed to the Controller when the agent attempts to connect. This property is optional in the rule; it can be inherited from the controller_host property defined in the header section of the rulebook or from the Controller identified in the `universalagent.yaml` file.	"account_name": "customer1"

`account_access_key`	Identifies the account access key that is passed to the Controller when the agent attempt to connect. This property is optional in the rule; it can be inherited from the controller_host property defined in the header section of the rulebook or from the Controller identified in the `universalagent.yaml` file.	"account_access_key": `"APJC234bcd$123"`
`controller_ssl_enabled`	SSL is enabled	"controller_ssl_enabled": "true"
`controller_enable_tls12`	tls12 is enabled	"controller_enable_tls12": "false"
`controller_secure`	Secure is enabled	"controller_secure": "false"
`config_xml`	Specifies, in JSON format, a list of repository URLs where `config.xml`, related scripts, and binaries are located. See Configuring the .NET Agent.	"config_xml": ["`repository`,repository,`...`"] where `repository=server URL`, such as `http://server/path` or `file://local/path`

Example .NET Agent Rule

In this .NET Agent rule example, the Controller and account settings are not shown because they are inherited from the global Universal Agent `config` or from the `universalagent.yaml` file.

```
{
   "condition":"platform_family == 'windows'",
   "comments":"This is an .NET Agent rule",
   "name":".NET Agent rule for Windows",
   "monitor":"dotnet",
   "config":{
      "state":"started",
      "version":"4.5.0.0",
      "config_xml":[
         "file:///C:/Tools/Repo"
      ]
   }
}
```

Configure the .NET Agent

The Universal Agent installs the .NET Agent and deploys the configuration file (`config.xml`) specified in the rulebook. The .NET Agent uses the config.xml file to determine which processes to instrument upon restart of the process. It is up to you to build the `config.xml` file and specify its location using the `config_xml` property in the .NET rule. A sample setup configuration file can be viewed in Unattended Installation for .NET.

For each .NET Agent version, you can specify a corresponding value for the `config.xml property`. This config file is retrieved from the repository specified by the `config_xml property`. The location can be local to the machine in a shared director or can be external.

Analytics Agent Rules

The AppDynamics Universal Agent uses a rulebook to determine which versions of the runtime agents should be installed and deployed. You can manage the deployment, versioning, and status of Analytics Agents collecting Transaction Analytics on a monitored machine by adding Analytics Agent rules to the Universal Agent rulebook. This page describes the syntax and usage for the Analytics Agent rules.

The supported functionality includes:

- Running the Analytics Agent as either an extension to the Standalone Machine Agent or in a standalone JVM
- Configuring the Analytics Agent property files, to run as either an extension to the Standalone Machine Agent or in a standalone JVM as indicated with the `mode` property
- Upgrading the standalone Analytics Agents to new versions
- Downgrading standalone Analytics Agents to prior versions
- Using the Analytics Agent to collect Transaction Analytics

You typically run the Analytics Agent in standalone mode if you are collecting only analytics data from the target machine. If you also need to collect hardware metrics or need to run other Machine Agent extensions, you would use the machine agent mode.

When deploying the Analytics Agent on Linux using the Universal Agent, Standalone mode is not supported, you must use machine-agent mode.

If you have an existing Machine Agent running outside of Universal Agent control and you want to install the Analytics Agent, you should either manage both with the Universal Agent, or manage each separately (not using the Universal Agent).

Define Analytics Agent Rules

A `monitor` value of `analytics` identifies an Analytics Agent rule.

The valid values for the `state` property for an Analytics Agent rule are:

- `installed`
- `started`

Analytics Agent Rules Syntax

The syntax for an Analytics Agent rulebook entry is as follows:

```
{
    "name": "...name of rule...",
    "comments": "...comments...",
    "monitor": "analytics",
    "config":
    {
        "mode": "machine-agent" | "standalone",
        "version": "...Analytics agent version...",
        "state": "installed" | "started",
        "controller_host": "...name of host running controller...",
        "controller_port": "...port number controller is listening on...",
        "account_name": "...controller account name...",
        "global_account_name": "...global account name...",
        "account_access_key": "controller account access key...",
        "deploy_in": [ "<machine-agent-rule-name>", ... ],
        "props": { "<prop-name-1>": "<prop-value-1>", "<prop-name-2>": "<prop-value-2>", ..., "<prop-name-n>":
"<prop-value-n>" },
        "vmoptions: [ "<vmoption-1>", "<vmoption-2>",..., "<vmoption-n>" ],
        "force_recycle": "true" | "false"
    },
    "condition": "...boolean expression indicating status of rule..."
}
```

The name, comments, monitor, and condition properties are common to all agent rules and have no special meaning for the Analytics Agent:

- `name`: A short string to identify the rule.
- `comments`: A long string describing the rule's purpose.
- `monitor`: Name of the target runtime agent (for Analytics Agent, must be "analytics").
- `condition`: Conditions for applying this rule. A condition consists of a Boolean expression where the operands are the values collected by the environment modules.

The properties within the `config` object have specific meanings for the Analytics Agent. They identify the following:

- Analytics Agent execution mode
- Property values for the Universal Agent to use to override the default values in the analytics property files
- Network location of the Controller that the agent connects with
- Whether or not changes to the Analytics Agent configuration should cause the hosting Machine Agent to be recycled.

Keyword	Description	Example
`mode`	Indicates whether the analytics agent is to run as a machine agent (machine-agent) or as a standalone JVM (standalone)	`"mode": "machine-agent"`
`version`	Identifies the version of the analytics agent to run. This is applicable only if the mode is "standalone". When running as a machine agent extension, the analytics agent uses the same version as the machine agent	`"version": "4.5.0.0"`
`state`	Indicates the state of the analytics agent: • `installed`: The analytics agent should be installed, but should not be started • `started`: The analytics agent should be installed and started	`"state": "started"`
`controller_host`	Identifies the name of the host that the controller the agent should connect to is running on. This property is optional in the rule; it can be inherited from the controller_host property defined in the header section of the rulebook or from the Controller identified in the `universalagent.yaml` file.	`"controller_host": "localhost"`
`controller_port`	Identifies the port number that the controller the agent should connect to is listening on. This property is optional in the rule; it can be inherited from the controller_host property defined in the header section of the rulebook or from the Controller identified in the `universalagent.yaml` file.	`"controller_port": "8080"`
`account_name`	Identifies the account name that is passed to the controller when the agent attempt to connect. This property is optional in the rule; it can be inherited from the controller_host property defined in the header section of the rulebook or from the Controller identified in the `universalagent.yaml` file.	`"account_name": "customer1"`
`global_account_name`	Identifies the global account name that is used to populate the `http.event.accountName` property in the `analytics-agent.properties` file. This property is optional in the rule; it can be inherited from the controller_host property defined in the header section of the rulebook or from the Controller identified in the `universalagent.yaml` file.	`"global_account_name": "customer1"`
`account_access_key`	`Identifies the account access key that is passed to the Controller when the agent attempts to connect. This property is optional in the rule. The account access key can be inherited from the controller_host` property defined in the header section of the rulebook `or from the Controller identified in the universalagent.yaml file.`	`"account_access_key": "AB1a2b3c4$123"`
`events_services_host`	Identifies the network name or address of the Events Service host. This value is used to populate the `http.event.endpoint` property in the `analytics-agent.properties` file. If this property is missing, then the value of the `"controller_host"` property is used.	`"event_services_host": "1.2.3.4"`
`events_services_port`	Identifies the port that the Events Service is listening on. This value is used to populate the `http.event.endpoint` property in the `analytics-agent.properties` file. If this value is missing, then the default value `9080` is used.	`"event_services_port": "444"`
`deploy_in`	Applies only when the mode is `machine-agent`. This property identifies the Machine Agent where the Analytics Agent runs as an extension. The value is a list of one or more strings, each of which contains the name of a Machine Agent rule within the Universal Agent rulebook. This argument is required if `"mode": "machine-agent"` is specified.	`"deploy_in" : [ "machine-agent-1" ]`
`props`	Contains one or more property definitions. Each property definition is added (or modified, if the property already exists) in the `conf/analytics-agent.properties` file for the analytics agent.	`"props": { "http.event.error.retryAttempts": "500" }`
`vmoptions`	Contains one or more JVM options definitions. Each JVM options is added to the `conf/analytics-agent.vmoptions` file	`"vmoptions" : [ "-verbose:class" ]`
`force_recycle`	If `true` is specified, then the Universal Agent restarts any Machine Agents hosting this Analytics Agent if any of the configuration arguments change. The default is `false`. Applicable only if `"mode": "machine-agent"` is specified.	`"force_recycle": "true"`

Example for Machine Agent Mode

This example specifies that the Analytics Agent should run as a Machine Agent extension. In this example:

- `mode: machine-agent` indicates machine-agent mode

- `state: "started"` indicates that the Analytics Agent should be started within the Machine Agent.
- `deploy_in:` Identifies the Machine Agents where the Analytics Agent should run.
- `props:` Defines a property that will be added to, or modified within, the `analytics-agent.properties` file of the Analytics Agent.
- `force_recycle:` A value of `true` indicates that the hosting Machine Agent should be recycled if the any of the config properties of the Analytic Agent change.

The following example illustrates these properties.

```
{
      "mode": "machine-agent",
      "state": "started",
      "deploy_in": [ "machine-agent-1" ],
      "props": { "http.event.error.retryAttempts": "500" },
      "force_recycle": true
}
```

Example for Standalone Mode

This example specifies that the Analytics Agent should run in a standalone JVM. In this example:

- `mode: standalone` indicates standalone mode.
- `state: started` indicates that the Analytics Agent should be started
- `props:` Defines a property that will be added to, or modified within, the `analytics-agent.properties` file of the Analytics Agent.

The following is an example of the Analytics Agent rules:

```
{
      "mode": "standalone",
      "version": "4.5.0.0",
      "state": "started",
      "props": { "http.event.error.retryAttempts": "500" }
}
```

Special Considerations for Standalone Mode

When you specify `"mode": "standalone"` for your Analytics Agent rule, there are some special considerations:

- Although a rulebook can contain multiple entries defining a standalone Analytics Agent, only one Analytics Agent runs at a time. When multiple rules are present that specify `"state": "started"`, the agent defined by the first rule is started. Analytics Agents identified by subsequent rules are not started. If other agents are currently running, they are stopped.
- If any configuration properties change for an Analytics Agent that is currently running, that agent is restarted.

Network Agent Rules

The Universal Agent can download, install, configure, and deploy the Network Agent. The Universal Agent uses a rulebook to determine which version of the Network Agent should be installed and deployed. You need to define and configure the Network Agent rule in the Universal Agent rulebook. See Universal Agent Rulebooks.

Define Network Agent Rules

The Network Agent is currently supported only on Linux. Therefore, Universal Agent support only works when running on a Linux machine. Installing the Network Agent using the Universal Agent requires elevated privileges for some commands. See Permissions for Running the Universal Agent.

A monitor value of `network` identifies a Network Agent rule.

Valid values for the Network Agent rule's `state` property are:

- `"installed":` Downloads and installs the Network Agent.
- `"started":` Starts the Network Agent. (Also installs the Network Agent if its not already installed.)
- `"stopped":` Stops the Network Agent.

Add to the Source Repository

When you place the Network Agent in the source repository, the installation distribution must have the following format:

```
network-<version>-<word_size>-<os>.zip
```

- `version`: Agent version. For example, `4.5.0`
- `word_size`: `32bit` or `64bit`
- `os`: `linux`

Network Agent Syntax

Network Agent rulebook is similar to the following.

```
{
    "name": "network",
    "comments": "Example Network Agent rulebook",
    "config": {
    },
    "rules": [
        {
            "name": "Network Agent rule",
            "comments": "Install and start the Network Agent",
            "monitor": "network",
            "condition": "True",
            "config": {
                "version": "4.5.0.0",
                "state": "started"
            }
       }
    ]
}
```

Dynamic Configuration Values

In rulebooks, you can set configurations with either static or dynamic values. Static values are set with literals, such as a literal string, whereas, dynamic values are based on environment variables. For most rulebook configurations, using static values is sufficient. For an elastic environment where the identity of nodes or tiers cannot be known in advance, however, you may want to use dynamic configuration values to name nodes, tiers, or applications.

This page describes the types of environment variables, how they are referenced in rulebooks, which agents have access to them, and how they are set. The last section provides a rulebook example using dynamic values for the configurations for rules for both the Java and Machine Agents.

Types of Environment Variables

The Universal Agent can access the three types of environment variables shown in this table to set dynamic configuration values.

Type	Syntax	Example	Agents with Access to Environment Variable
Java environment variables	`$javaenv_<variable_name>`	`$javaenv_APP_NAME`	Java Agent
Java system properties	`"${property-name}"`, where `property-name` is the name of the Java system property.	`${tierName}`	Java Agent
system environment variables	`$env_<variable_name>`	`$env_TIER_NAME`	Java Agent, Machine Agent

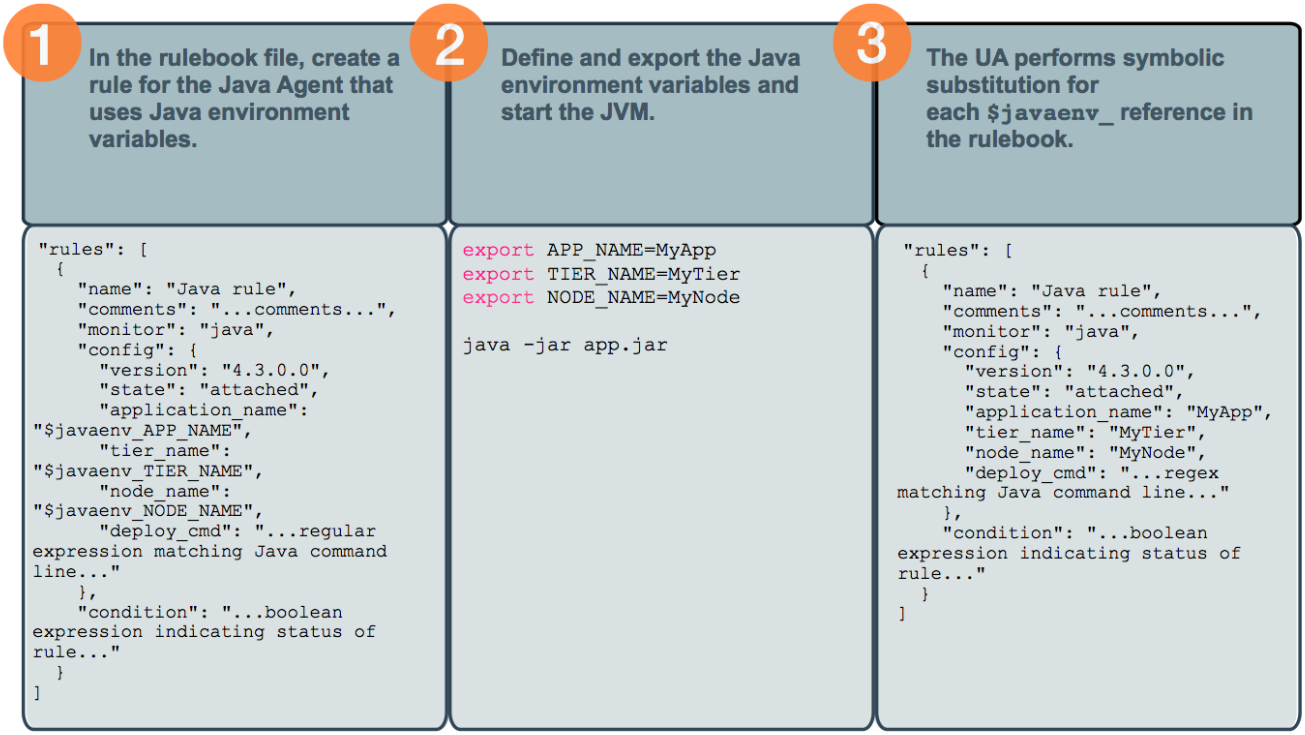
Define Environment Variables

The following sections describe the process for defining, setting, and applying different types of environment variables.

Java Environment Variables

To set Java environment variables, you define export environment variables and then start the JVM with the environment variables as command-line arguments. The JVM will then set the Java environment variables so that they are accessible to the Universal Agent. If the Java rule applies to a particular JVM, the Universal Agent then extracts the values of each environment variable and performs symbolic substitution into the config values in the rule for each `$javaenv_` reference.

The flow for setting and applying Java environment variables is illustrated in the diagram:

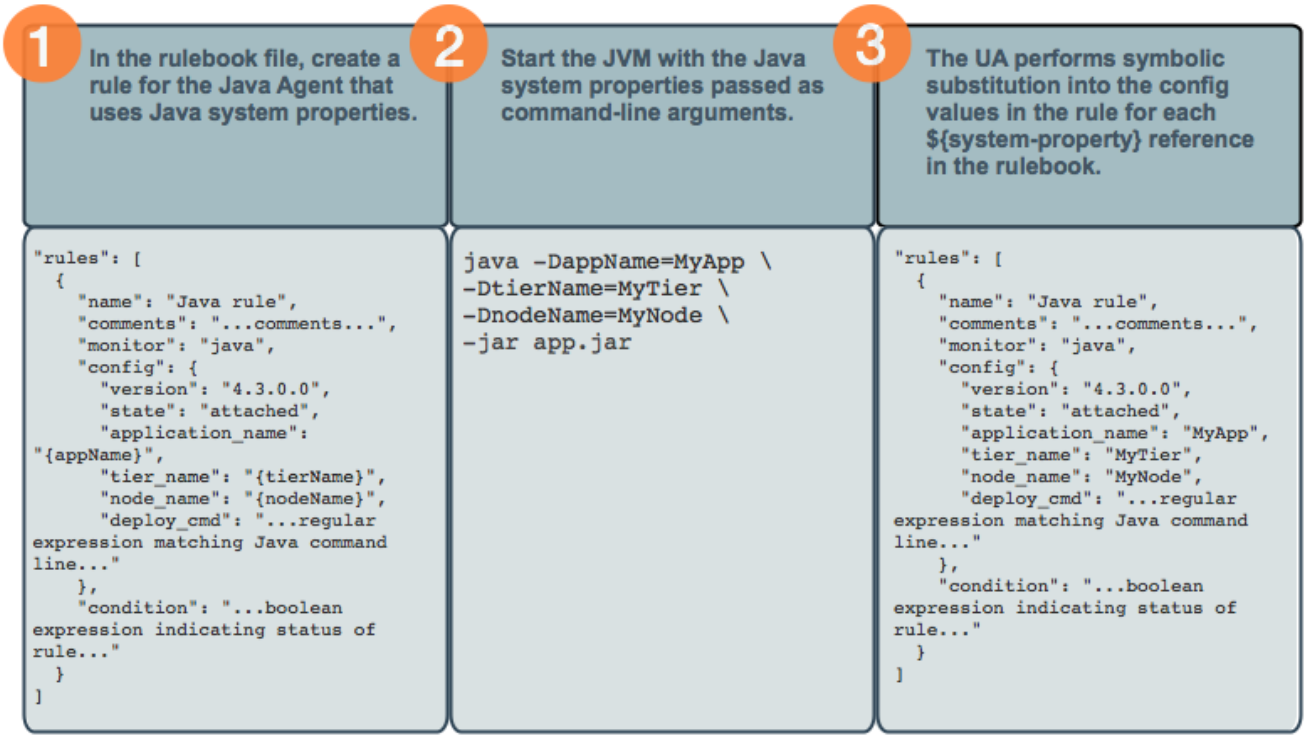


For environment variable substitution, if a Java Agent is introduced into a JVM at startup (as opposed to being remote attached), the `javaagent.jar` file must be version 4.3 or later. The 4.3 Java Agent must be installed by the Universal Agent, but it does not need to be deployed—simply installing the 4.3 Java Agent causes the supporting `javaagent.jar` file to be loaded into each JVM.

Java System Properties

As with Java environmental variables, you can also set Java system properties by passing command-line arguments when you start the JVM. In addition, you can programmatically set Java system properties. If the Java rule applies to a particular JVM, the Universal Agent extracts the values of each environment variable and performs symbolic substitution into the config values in the rule for each `${system-property}` reference.

The flow for setting and applying Java system properties is illustrated in the diagram below:



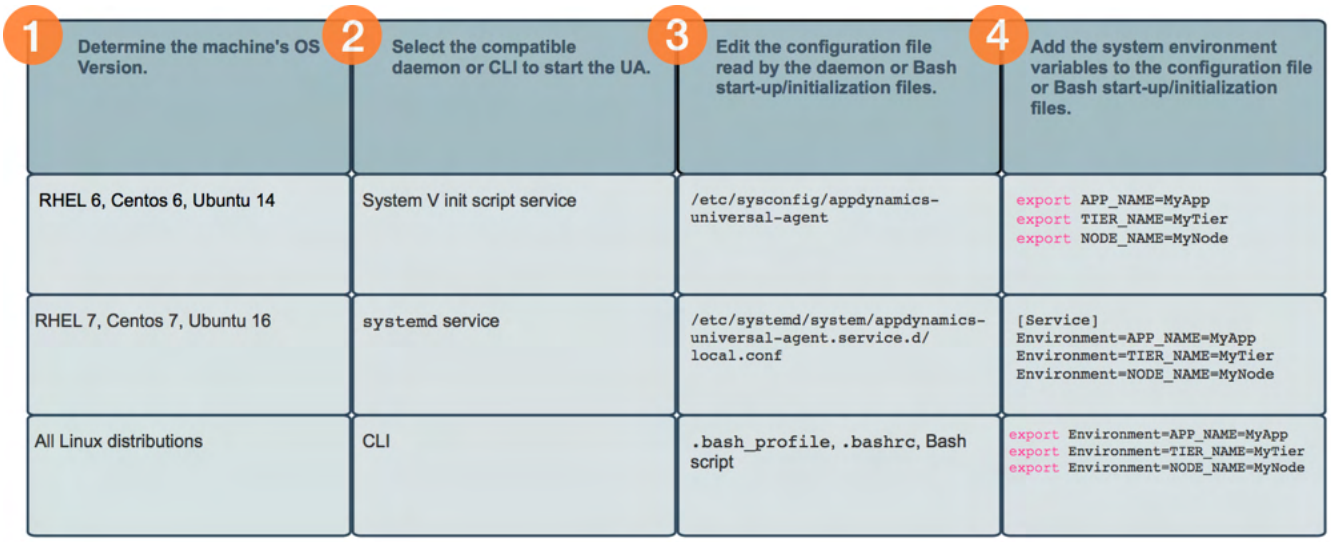
As mentioned above, you can also programmatically set a Java system property through the `System` class as shown below.

```
Properties props = System.getProperties();
props.setProperty("tierName", "MyTier");
```

System Environment Variables

The method of setting system environment variables and starting the Unified Agent depends on the version of the machine OS. The general process is to set system environment variables in a configuration file and then use the system-specific daemon or the CLI to start the Unified Agent. If a rule applies to either a particular JVM or system, the Universal Agent extracts the values of each system environment variable and performs symbolic substitution into the config values in the rule for each `$env_` reference.

The flow for setting and applying system environment variables is illustrated in the diagram:



1 Determine the machine's OS Version.	2 Select the compatible daemon or CLI to start the UA.	3 Edit the configuration file read by the daemon or Bash start-up/initialization files.	4 Add the system environment variables to the configuration file or Bash start-up/initialization files.

RHEL 6, Centos 6, Ubuntu 14	System V init script service	`/etc/sysconfig/appdynamics-universal-agent`	`export APP_NAME=MyApp`
			`export TIER_NAME=MyTier`
			`export NODE_NAME=MyNode`

RHEL 7, Centos 7, Ubuntu 16	`systemd` service	`/etc/systemd/system/appdynamics-universal-agent.service.d/local.conf`	`[Service]`
			`Environment=APP_NAME=MyApp`
			`Environment=TIER_NAME=MyTier`
			`Environment=NODE_NAME=MyNode`

All Linux distributions	CLI	`.bash_profile`, `.bashrc`, Bash script	`export Environment=APP_NAME=MyApp`
			`export Environment=TIER_NAME=MyTier`
			`export Environment=NODE_NAME=MyNode`

Example Rulebook with Dynamic Configuration Values

The following example rulebook has rules containing dynamic configuration values for both Java and Machine Agents. For more information about the dynamic configuration values, see the explanation following the example.

> The `env` and `javaenv` variables can be used in condition fields. In condition fields you should leave off the $ sign from the `env` and javaenv variables. Review the java rule in the rulebook that follows for an example.

```
{
  "name": "AD-Capital",
  "comments": "Agents for AD-Capital",
  "config": {},
  "rules": [
    {
      "name": "Java rule",
      "comments": "...comments...",
      "monitor": "java",
      "config":
      {
        "version": "4.5.0.0",
        "state": "attached",
        "application_name": "$env_APP_NAME",
        "tier_name": "${tierName}",
        "node_name": "$javaenv_APP_NAME-$javaver_NODE_NAME",
      },
      "condition": "env_DEPLOYMENT==\"PROD\""
    },
    {
      "name": "machine-4.5.0.0",
      "monitor": "machine",
      "comments": "Update Machine Agent to 4.5.0.0",
      "config": {
        "state": "started",
        "version": "4.5.0.0",
        "application_name": "$env_APP_NAME",
        "tier_name": "$env_TIER_NAME",
        "node_name": "$env_NODE_NAME"
      },
      "condition": "platform_system == '\''Linux'\'' and universalagent_version > '\''4.5.0'\''"
    },

    {
      "name": "Universal Agent rule",
      "monitor": "universal",
      "comments": "Update Universal Agent to 4.5.0.0",
      "config": {
        "state": "started",
        "version": "4.5.0.0"
      },
      "condition": "True"
    }
  ]
}
```

In the previous rulebook example, the rule for the Java Agent contains the following attributes having values referencing JVM environment variables, Java system properties, and system environment variables:

- `"application_name": "$env_APP_NAME":` Indicates that the value of the `application_name` config attribute is the value of the `APP_NAME` environment variable set by the system (daemon/CLI) for which the rule applies.
- `"tier_name": "${tierName}":` Indicates that the value of the `tier_name` config attribute is the value of the `tierName` system property within a JVM for which the rule applies.
- `"node_name": "$javaenv_APP_NAME-$javaver_NODE_NAME":` Indicates that the value of the `node_name` config attribute is the value of the `APP_NAME` environment variable within a JVM for which the rule applies, concatenated with a `'-'`, followed by the value of the `NODE_NAME` environment variable within the JVM.
- "`env_DEPLOYMENT`": This variable is used in the `condition` field. This checks whether there is an environment variable `DEPLOYMENT` set with a value equal to `PROD`. Note that although the `env` and javaenv variables can be used in condition fields, the format is different. You should not specify the $ sign.

The rule for the Machine Agent contains the following attributes having values referencing system environment variables:

- `"application_name":"$env_APP_NAME":` Indicates that the value of the `application_name` config attribute is the value of the `APP_NAME` environment variable set by the system (daemon/CLI) when the Universal Agent is started, for which the rule applies.
- `"tier_name": "$env_TIER_NAME":` Indicates that the value of the `tier_name` config attribute is the value of the `TIER_NAME` environment variable set by the system (daemon/CLI) when the Universal Agent is started, for which the rule applies.
- `"node_name": "$env_NODE_NAME":` Indicates that the value of the `node_name` config attribute is the value of the `NODE_NAME` environment variable set by the system (daemon/CLI) when the Universal Agent is started, for which the rule applies.

Rulebook Configuration Templates

Configuration templates enable you to define a set of default configuration values that apply across a set of rulebooks. The configuration templates are stored on the AppDynamics Controller and used to update the rulebooks on machines where the Universal Agent manages the runtime agents. The use of configuration templates is optional, however, using them can reduce the size of your rulebooks and simplify rulebook management across a set of machines.

Rulebook Background

The rulebook for each Universal Agent instance uses a JSON object called `config` to define a set of key-value pairs containing configuration values that are specific for the agent that is being monitored. These `config` objects can be large and in many cases may contain redundant fields across numerous runtime agents that are being managed by a single Universal Agent instance.

Configuration templates implement a JSON object that defines a basic, default configuration for your runtime agents. The template object is a set of key-value pairs that can be referenced in your rulebooks at both the header level (applying globally to all runtime agents in the rulebook) and in agent-specific rules.

The configuration template is applied to the rulebook first, and properties in the `config` object in the rulebooks can override the default values if needed for specific use cases.

Users with the `Configure Universal Agent` permission can create and update configuration templates using the REST APIs. See Universal Agent REST APIs.

- Configuration templates can be used only when running the Universal Agent in controller mode.
- The rulebooks reference applicable configuration templates using the `configEntity` property.
- Individual rules in your rulebook can override values defined in the templates using the rule-level `config` object.
- When the Universal Agent processes a rule that contains a `configEntity` property, a request is sent to the Controller for the content of the specified template. The template values are then pushed to the Universal Agent, where they are applied to the rulebook.
- The Universal Agent reports the version ID of each rulebook and template that was sent in the prior cycle back to its Controller. If necessary, the Controller then sends more recent versions of the rulebooks and templates back to the Universal Agent.

Workflows for Using Configuration Templates

Related pages:

- Rulebook Configuration Templates
- Universal Agent REST APIs

Knowledge and understanding of the Universal Agent rulebook structure is a prerequisite to understanding and using configuration templates. See Universal Agent Rulebooks before using configuration templates.

Specify Global Configuration Values in a Template

When you have common configuration values across different rulebooks, instead of repeating all the fields in a header config section in each separate rulebook, you can create a configuration template that contains the common values and then add a reference to the template in the relevant rulebooks.

This scenario uses the Configuration Template to specify configuration values that apply across multiple rulebooks. The configuration values in the template will be incorporated into the referring rulebook.

1. Create the configuration template using the Configuration Template `PUT` API. The following tabs show a sample command and the response format.
2. Create a rulebook that refers to the configuration template using the rulebook `PUT` API and the `configEntity` keyword. The following tabs show a sample command and the response format.

This scenario is the equivalent of having the following config in the rulebook:

```
"config": {
            "controller_host":"host.controller.com",
            "controller_port":"8080",
            "account_name":"customer1",
            "account_access_key":"Acces$$Key"
           }
```

If both `config` and `configEntity` are provided in a rulebook, the resulting configuration is computed by over-riding `configEntity` fields with `config` fields.

For example, if the rulebook request shown in step 2 above, contains the config shown here in Config section and also specifies the "`configEntity`": "`controllerConfig`" property, the result is equivalent to what is shown in Config block:

Placeholder for Rule Configurations in Rulebooks

Configuration templates can also be used as placeholders for individual rule configurations. This means a "`configEntity`":*<configuration_template_name>* can be used to replace the "`config`" key-value pairs.

1. Create the configuration template. The tabs show a sample command and the response format.
2. Create or update a rulebook that refers to the template. This example updates a previously created rulebook. You can use the same procedure to create a new rulebook with `configEntity` reference. The tabs show a sample command and the response format.

Reuse Configuration Templates

The primary use of templates is for reuse across multiple rulebooks (or even rules). In this scenario, we create a rulebook with different controller settings but the same rule configuration as the Java rule in the previous example.

1. Create the Controller configuration template.
2. Create a rulebook referring to `OtherControllerConfig` and sharing `javamonitorconfiguration.`
3. View the shared configuration `referredRuleBooks` field by doing a `GET controller/universalagent/v1/user/configurations/javamonitorconfiguration.`

Universal Agent REST APIs

The Universal Agent API is a Controller REST API that enables you to monitor and manage a Universal Agent deployment programmatically.

The Universal Agent API enables you to act on Universal Agent groups, individual Universal Agents, rulebooks, and Universal Agent events.

Permissions

The following RBAC permissions and roles authorize the use of the Universal Agent REST APIs.

- Permissions
  - Configure Universal Agent: Allows all `PUT` and `DELETE` APIs
  - View Universal Agent: Allows all `GET` APIs
- Roles
  - Universal Agent Administrator with both view and configure permissions
  - Universal Agent User with view permission

Universal Agent Administrator and Universal Agent User roles are added to all new and existing accounts. Also, Universal Agent Administrator and Universal Agent User roles are added to users who have *Account Administrator* role. Configure Universal Agent and View Universal Agent permissions are added to *Account Administrator* role of all accounts.

See Roles and Permissions for AppDynamics permissions and roles.

Using Environment Data

Using the API calls described in Get Complete Information for Universal Agents, you can get information about the running environment for each Universal Agent and the runtime agents. You can then use the values to populate rulebook properties, such as tier and application names. See Dynamic Configuration Values.

To read environment variable data from JVM environments, the Universal Agent needs to have sufficient permissions. See more information about how to provision the Universal Agent with sudo access if necessary in Permissions for Running the Universal Agent.

As an API response, the environment data is made up of the following name/value pairs:

- Information about the underlying platform:
  - `platform_machine`: Machine type. For example, `i386`
  - `platform_processor`: Processor name. For example, `amdk6`
  - platform_release: System/OS release. For example, `2.2.0` or `NT`
  - platform_system: System/OS name. For example, `Linux` or `Windows`
  - platform_version: System release version. For example, `Darwin Kernel Version 14.4.0`
- Network-related information:
  - `hostname`: Fully qualified domain name
  - `ip_addresses`: List of IP addresses for machines where the Universal Agent is running
  - `mac_addresses`: List of MAC addresses for machines where the Universal Agent is running
- Names and values of each environment variable. The keys are the name of each environment variable, prefixed with env_. For example, if the environment contains the variables HOME and SHELL, then this section may look like:
  - `env_HOME: /Users/john.smith`
  - `env_SHELL: /bin/bash`
- Time information:
  - `time_utc`: Current UTC date and time in ISO 8601 format
- Information about the Universal Agent
  - `universalagent_version`: Current version. For example, 4.5.0.0.
  - `universalagent_build`: Build number
- Information about the Java Agent:
  - `java_targets`: List of running Java processes on the machine. This is organized as a map of process IDs (PID) to process command line
  - `java_installs`: Absolute paths of the Java runtime agents installed by the Universal Agent. The path is organized as nested maps, using the app/tier/node name as the key. For example:

```
"java_installs": {
    "MyApp": {
        "MyTier": {
            "MyNode1": "/path/to/java/app/monitor/1",
            "MyNode2": "/path/to/java/app/monitor/2",
        }
    },
    "MyApp2": {
        "MyTier2a": {
            "Node": “/path/to/java/app/monitor/3"
        },
        "MyTier2b": {
            "Node": “/path/to/java/app/monitor/4"
        },
        "null": {
            "null": "/path/to/java/app/monitor/that/only/had/app/specified/in/controller/info/xml"
        }
    }
}
```

- Information about the Machine Agent
    - `machine_installs`: List of absolute paths to Machine Agents installed by the Universal Agent
    - `machine_starts`: List of absolute paths to the Machine Agents started by the Universal Agent
- Information about .NET Agent
    - `dotnet_frameworks`: List of the installed .NET frameworks
    - `dotnet_version`: .NET Agent version, if installed
- Information about the Network Agent
    - none
- Information about Analytics Agent
    - none

API Rate and Total Count Limits

The following rate limits apply by default:

- Maximum 5000 agent registrations per minute per account
- Maximum 5000 event registrations per minute per account
- Maximum 500 user requests per minute per account

The following total count limits apply by default:

- Maximum 5000 agents per account
- Maximum 500 rulebooks per account
- Maximum 500 groups per account

For on-premises Controllers, you can configure rate and total count limits from the Controller Administration Console.

Get Complete Information for Universal Agents

Returns information about running and previously running Universal Agents. When used with the `groups` query parameter, this API returns information only for the agents under the specified groups.

```
GET controller/universalagent/v1/user/agents?agents=<value1, value2>
GET /controller/universalagent/v1/user/agents?agents=<value1, value2>&groups=<value>
GET /controller/universalagent/v1/user/agents?groups=<value1, value2>
```

Query Parameters

Parameter Name	Value	Required
agents	A comma-delimited string of Universal Agent names. When present, returns the relevant matching subset of agents.	No
groups	A comma-delimited string of group names. When present, returns only agents under the specified groups.	No

Example Command

```
curl -i -X GET -u '<userName>@<accountName>:<password>'
-H 'Content-type: application/json'
http://<controller-host>:8080/controller/universalagent/v1/user/agents
```

Response Format

```
{
    "agent1": {
        "version": <string>,
        "ruleBookName": <string>,
        "environment": {
            "key1": <object>,
            ...
        }
    },
    ...
}
```

Response Fields

- `agent1`: Name of the Universal Agent
- `version`: Property specifying the version of the Universal Agent
- `ruleBookName`: Property specifying the name of the rulebook of the Universal Agent
- `environment`: Object holding the environment data collected by the Environment Modules

Get Summary Information for Universal Agents

View summary information about all running Universal Agents. The environment data is suppressed. When used with the `groups query` parameter, this API returns information only for the agents under the specified groups.

```
GET /controller/universalagent/v1/user/agents/summary?agents=<value>
GET /controller/universalagent/v1/user/agents/summary?agents=<value1, value2>&groups=<value>
GET /controller/universalagent/v1/user/agents/summary?groups=<value1, value2>
```

Query Parameters

Parameter Name	Value	Required
`agents`	A comma-delimited string of Universal Agent names. When present, the relevant matching subset of agents is returned.	No
`groups`	A comma-delimited string of group names. When present, returns only agents under the specified groups.	No

Example Command

```
curl -i -X GET -u '<userName>@<accountName>:<password>'
-H 'Content-type: application/json'
http://<controller-host>:8080/controller/universalagent/v1/user/agents/summary
```

Response Format

```
{
    "agent1": {
        "version": <string>,
        "ruleBookName": <string>
    },
    ...
}
```

Response Fields

- `agent1`: Name of the Universal Agent
- `version`: Version of the Universal Agent
- `ruleBookName`: Name of the rulebook of the Universal Agent

Get Universal Agent By Name

View details for the Universal Agent with a specified name.

```
GET /controller/universalagent/v1/user/agents/byName/<name>
```

Path Parameters

Parameter Name	Value	Required
`name`	Name of the Universal Agent. For example, `agent1.`	Yes

Example Command

```
curl -i -X GET -u '<userName>@<accountName>:<password>'
-H 'Content-type: application/json'
http://<controller-host>:8080/controller/universalagent/v1/user/agents/byName/agent1
```

Response Format

```
{
    "version": <string>,
    "ruleBookName": <string>,
    "environment": {
        "key1": <object>,
        ...
    }
}
```

Response Fields

- `agent1`: Name of the Universal Agent
- `version`: Version of the Universal Agent
- `ruleBookName`: Name of the current rulebook of the Universal Agent

Delete Universal Agent by Name

Delete an agent by name.

```
DELETE /controller/universalagent/v1/user/agents/byName/<name>
```

Deleting an agent also deletes events associated with it.

Path Parameters

Parameter Name	Value	Required
`name`	Name of the Universal Agent	Yes

Example Command

```
curl -i -X DELETE -u '<userName>@<accountName>:<password>'
-H 'Content-type: application/json'
http://<controller-host>:8080/controller/universalagent/v1/user/agents/byName/agent1
```

Create or Update a Group

Universal Agent groups are a logical way of grouping different Universal Agents that are running on different machines. You can add different Universal Agents to a group and then add a common rulebook to that group. All Universal Agents in the group execute the same set of rules (that apply to their environment). For details on groups: see the section 'Rulebook Strategies' in Universal Agent Rulebooks

Create the group. If the group exists, it is updated.

```
PUT /controller/universalagent/v1/user/groups/byName/<name>
```

Path Parameters

Parameter Name	Description	Required
`name`	Name of the group. In the example that follows, `"name": ""` is overridden by the path parameter `name`	Yes

Example Command

```
curl -i -X PUT -u '<userName>@<accountName>:<password>'
-H 'Content-type: application/json'
-H 'Accept:application/json'
http://<controller-host>:8080/controller/universalagent/v1/user/groups/byName/group1
-d '{"name":"","comments":"This is group1"}'
```

Response Format

```
{
    “name": <string>,
    “comments": <string>
}
```

Add Universal Agent to a Group

Add the specified Universal Agent to the specified group.

```
PUT /controller/universalagent/v1/user/agents/membership/<agentName>/groups/<groupName>
```

Path Parameters

Parameter Name	Value	Required
`agentName`	Name of the Universal Agent	Yes
`groupName`	Group name. For example: `group1`	Yes

Example Command

```
curl -i -X PUT -u '<userName>@<accountName>:<password>'
-H 'Content-type: application/json'
-H 'Accept:application/json'
http://<controller-host>:8080/controller/universalagent/v1/user/agents/membership/agent1/groups/group1
```

Response Format

```
{
    "agent1": {
        "version": <string>,
        "ruleBookName": <string>,
        "environment": {
            "key1": <object>,
            ...
        }
    },
    ...
}
```

Delete Universal Agent From a Group

Delete the specified Universal Agent from the specified group.

```
DELETE /controller/universalagent/v1/user/agents/membership/<agentName>/groups/<groupName>
```

Path Parameters

Parameter Name	Value	Required
`agentName`	Name of the Universal Agent. For example, `agent1`	Yes
`groupName`	Group name. For example, `group1`	Yes

Example Command

```
curl -i -X DELETE -su '<userName>@<accountName>:<password>'
-H 'Content-type: application/json'
-H 'Accept:application/json'
http://<controller-host>:8080/controller/universalagent/v1/user/agents/membership/agent1/groups/group1
```

Move Universal Agent to Multiple Groups

Universal Agent is added to multiple groups and removed from previous groups.

```
PUT /controller/universalagent/v1/user/agents/groupsMembership/<agentName>
```

Path Parameters

Parameter Name	Value	Required
`agentName`	Name of the Universal Agent. For example, `agent1`	Yes
`groupName`	A list of group names formatted as JSON data. For example: `-d '[{"groupName":"group1"},{"groupName":"group2"}]'`	

Example Command

```
curl -i -X PUT -su '<userName>@<accountName>:<password>'
-H 'Content-type: application/json'
-H 'Accept:application/json'
http://<controller-host>:8080/controller/universalagent/v1/user/agents/groupsMembership/agent1
-d '[{"groupName":"group1"},{"groupName":"group2"}]'
```

Response Format

```
{
    "agent1": {
        "version": <string>,
        "ruleBookName": <string>,
        "environment": {
            "key1": <object>,
            ...
        }
    },
    ...
}
```

Get a Group By Name

Return information for the named group.

```
GET /controller/universalagent/v1/user/groups/byName/<name>
```

Path Parameters

Parameter Name	Description	Required
`name`	Name of the group. For example, `group1`	Yes

```
curl -i -X GET -u '<userName>@<accountName>:<password>'
-H 'Content-type: application/json'
http://<controller-host>:8080/controller/universalagent/v1/user/groups/byName/group1
```

Response Format

```
{
    "name": <string>,
    "comments": <string>
}
```

Get Multiple Groups

Get all groups or specific groups, groups having specific agents or groups having specific rulebooks.

```
GET /controller/universalagent/v1/user/groups?groups=group1,group2&agents=agent1,agent2&rulebooks=rulebook1
```

Query Parameters

Parameter Name	Description	Required
`groups`	Use to get specific groups, such as /controller/universalagent/v1/user/groups?groups=group1,group2	No
`agents`	Use to get groups having specific agents.	No
`rulebooks`	Use to get groups specific rulebooks.	No

Example Command

```
curl -i -X GET -u '<userName>@<accountName>:<password>'
-H 'Content-type: application/json'
http://<controller-host>:8080/controller/universalagent/v1/user/groups
```

Response Format

```
{
    "name": <string>,
    "comments": <string>
},
{
    "name": <string>,
    "comments": <string>
}
```

Delete a Group

Deletes the group. Deletion of the default group named `default` is not allowed.

Format

DELETE /controller/universalagent/v1/user/groups/byName/<name>

Path Parameters

Parameter Name	Description	Required
`name`	Name of the group, for example, `group1`.	Yes

Example Command

```
curl -i -X DELETE -u '<userName>@<accountName>:<password>'
-H 'Content-type: application/json'
http://<controller-host>:8080/controller/universalagent/v1/user/groups/byName/group1
```

Create or Update a Rulebook

Create a rulebook. If a rulebook with "`name`" already exists, it is updated.

Format

PUT /controller/universalagent/v1/user/rulebooks/byName/<name>

Path Parameters

Parameter Name	Value	Required
`name`	Name of the group. This path parameter overrides `"name": "rulebook1"`	Yes

Example Command

```
curl -i -X PUT -u '<userName>@<accountName>:<password>'
-H 'Content-type: application/json'
-H 'Accept:application/json'
http://<controller-host>:8080/controller/universalagent/v1/user/rulebooks/byName/rulebook1
-d ' {
 "name": "rulebook1",
 "comments": "An example rulebook with a rule to start a machine agent",
 "config": {
     "controller_host": "localhost",
     "controller_port": "8080",
     "account_name": "",
     "account_access_key": ""
 },
 "rules": [
     {
         "name": "Example rule to start a 4.5.0.0 machine agent",
         "comments": "Enable by changing 'condition' to 'True'. Controller information is inherited from the
'config' section above. The machine agent will be installed in <universal_agent_dir>/monitor/machine/4.5/...",
         "monitor": "machine",
         "config": {
             "version": "4.5.0.0",
             "state": "started"
         },
         "condition": "True"
     }
 ]
}'
```

Response Format

```
{
    "name": <string>
    "comments": <string>,
    "config": {
        "key1": <object>
        ...
    },
    "rules": [
        {
            "name": <string>,
            "comments": <string>,
            "monitor": <string>,
            "config": {
                "key1": <object>
                ...
            },
            "condition": <string>
        },
        ...
    ]
}
```

Response Fields

- name: Name of the rulebook
- comments: Comments describing the rulebook
- config: Optional set of name-value configuration settings inherited by each rule or specified in a specific rule
- rules: Object with a list of rules in the rulebook

Get Rulebooks

View specific rulebooks by name or group or view a list of all rulebooks.

```
GET /controller/universalagent/v1/user/rulebooks
GET /controller/universalagent/v1/user/rulebooks/byName/<name>
GET /controller/universalagent/v1/user/rulebooks/current/<groupName>
```

Returns the name, comments, and the configuration rules.

Path Parameters

Parameter Name	Value	Required
name	Name of the rulebook	Yes
group_name	Name of the group	Yes

Example Command for a List of All Rulebooks

```
curl -i -X GET -u '<userName>@<accountName>:<password>'
-H 'Content-type: application/json'
http://<controller-host>:8080/controller/universalagent/v1/user/rulebooks
```

Response Format

This version returns a list of rulebooks similar to the following:

```
[
    {
        "name": <string>,
        "comments": <string>,
        "config": {
            "key1": <object>
            ...
        },
        "rules": [
            {
                "name": <string>,
                "comments": <string>,
                "monitor": <string>,
                "config": {
                    "key1": <object>,
                    ...
                },
                "condition": <string>
            },
            ...
        ]
    },
    ...
]
```

Example Command for Specific Rulebook by Name

```
curl -i -X GET -u '<userName>@<accountName>:<password>'
-H 'Content-type: application/json'
http://<controller-host>:8080/controller/universalagent/v1/user/rulebooks/byName/rulebook1
```

Example Command for Rulebooks in a Group

```
curl -i -X GET -u '<userName>@<accountName>:<password>'
-H 'Content-type: application/json'
http://<controller-host>:8080/controller/universalagent/v1/user/rulebooks/current/group1
```

Response Format

The versions using `name` and `groupName` return a single rulebook similar to the following:

```
{
        "name": <string>,
        "comments": <string>,
        "config": {
            "key1": <object>
            ...
        },
        "rules": [
            {
                "name": <string>,
                "comments": <string>,
                "monitor": <string>,
                "config": {
                    "key1": <object>,
                    ...
                },
                "condition": <string>
            },
            ...
        ]
    }
```

Response Fields

- `name`: Name of the rulebook
- `comments`: Comments describing the rulebook
- `config`: Optional set of name-value configuration settings inherited by each rule or specified in a specific rule
- `rules`: Object with a list of rules in the rulebook

Add a Rulebook to a Group

Indicate which rulebook should be used by the Universal Agents in the group indicated by `groupName.`

```
PUT /controller/universalagent/v1/user/rulebooks/current/<groupName>
```

Path Parameters

Parameter Name	Value	Required
`groupName`	name of the group	Yes

Example Command

```
curl -i -X PUT -u '<userName>@<accountName>:<password>'
-H 'Content-type: application/json'
-H 'Accept:application/json'
http://<controller-host>:8080/controller/universalagent/v1/user/rulebooks/current/group1
-d '{"ruleBookName":"rulebook1"}'
```

Response Format

```
{
    "name": <string>
    "comments": <string>,
    "config": {
        "key1": <object>
        ...
    },
    "rules": [
        {
            "name": <string>,
            "comments": <string>,
            "monitor": <string>,
            "config": {
                "key1": <object>
                ...
            },
            "condition": <string>
        },
        ...
    ]
}
```

Response Fields

- `name`: The name of the rulebook
- `comments`: Comments describing the rulebook
- `config`: Optional set of name-value configuration settings inherited by each rule or specified in a specific rule
- `rules`: Object with a list of rules for each agent in the rulebook

Get a Rulebook Associated with a Group

View the name of the current rulebook for the specified group of Universal Agents.

```
GET /universalagent/v1/user/rulebooks/current/<groupName>
```

Path Parameters

Parameter Name	Value	Required
`groupName`	The name of the group.	Yes

Example Command

```
 curl -i -X GET -u '<userName>@<accountName>:<password>'
-H 'Content-type: application/json'
http://<controller-host>:8080/controller/universalagent/v1/user/rulebooks/current/group1
```

Response Format

```
{
    "name": <string>
    "comments": <string>,
    "config": {
        "key1": <object>
        ...
    },
    "rules": [
        {
            "name": <string>,
            "comments": <string>,
            "monitor": <string>,
            "config": {
                "key1": <object>
                ...
            },
            "condition": <string>
        },
        ...
    ]
}
```

Response Fields

- `name`: Name of the rulebook
- `comments`: Comments describing the rulebook
- `config`: Optional set of name-value configuration settings inherited by each rule or specified in a specific rule
- `rules`: Section for a list of rules in the rulebook

Delete a Rulebook By Name

Remove the rulebook with the specified name.

```
DELETE /controller/universalagent/v1/user/rulebooks/byName/<name>
```

Deleting a rulebook also removes it from any group association. Deletion of the default rulebook named "default-controller" is not allowed.

Path Parameters

Parameter Name	Value	Required
`name`	name of the rulebook	Yes

Example Command

```
curl -i -X DELETE -u '<user_name>@<account_name>:<password>'
-H 'Content-type: application/json'
http://<controller_host>:8080/controller/universalagent/v1/user/rulebooks/byName/rulebook1
```

Delete a Rulebook From a Group

Remove a rulebook for Universal Agents by group.

```
DELETE /controller/universalagent/v1/user/rulebooks/current/<groupName>
```

Universal Agents without a rulebook execute local rulebooks (thus allowing manual rulebook creation). Deletion of the default rulebook named "default-controller" is not allowed.

Path Parameters

Parameter Name	Value	Required

groupName	name of the group	Yes

Example Command

```
curl -i -X DELETE -u '<userName>@<accountName>:<password>'
-H 'Content-type: application/json'
http://<controller-host>:8080/controller/universalagent/v1/user/rulebooks/current/group1
```

Get All Events

View events that have occurred across all the Universal Agents. When the query parameters are present, the relevant matching subset of events is returned. By default, all events are shown up to the maximum limit of 600 events.

```
GET /controller/universalagent/v1/user/events?agentIds=<value1,value2>&eventTypes=<value1,
value2>&subTypes=<value>&timeRange=<value>
```

Query Parameters

Parameter Name	Value	Required
agentIds	A comma-delimited strings of Universal Agent names. When present, events for the relevant matching subset of agents is returned.	No
eventTypes	A comma-delimited string of event types. When present, returns only relevant events. The valid event types are: • UNIVERSAL_AGENT_STARTUP • UNIVERSAL_AGENT_SHUTDOWN • UNIVERSAL_AGENT_EXECUTE_RULE	No
subTypes	The success or error code. The valid subtypes are: • Success • Error	No
timeRange	Time range specifier string as used in other parts of the Controller. For example: • last_1_hour.BEFORE_NOW.-1.1424287621109.60 • last_5_minutes.BEFORE_NOW.-1.-1.5	No

Example Command

```
curl -i -X GET -u '<userName>@<accountName>:<password>'
-H 'Content-type: application/json'
http://<controller-host>:8080/controller/universalagent/v1/user/events
```

Response Format

```
[
 {
"id":<long>,
"version":<long>,
"eventType":<string>,
"eventTime":<Date>,
"severity":<string>,
"eventSummary":<string>,
"affectedEntities":[
   {
     "id":<long>,
     "version":<long>,
 "entityType":<string>,
     "entityId":<long>,
     "prettyToString":<string>
   }...,
],

"correlationKeys":[],
"properties":<string>,
"markedAsRead":<boolean>,
"markedAsResolved":<boolean>,
"archived":<boolean>,
"endTime":<long>,
"guid":<string>,
"subType":<string>,
"triggeredEntity":{
    "id":<long>,
    "version":<long>,
 "entityType":<string>,
    "entityId":<long>,
   "prettyToString":<string>
   }
},...
]
```

Response Fields

- `id`: ID of the event
- `version`: Version of the event
- `eventType`: Type of the event
- `eventTime`: Time when the event was generated
- `severity`: Can be INFO or ERROR
- eventSummary: Brief description of the event
  affectedEntities:
    - `entityType`: A string representing universal agent, UNIVERSAL_AGENT
    - `entityId`: ID of the Universal Agent
- `correlationKeys`: Not applicable for Universal Agent
- `properties`: Not applicable for Universal Agent
- markedAsRead: Not applicable for Universal Agent
- `markedAsResolved`: Not applicable for Universal Agent
- `archived`: Not applicable for Universal Agent
- `endTime`: Not applicable for Universal Agent
- guid: Not applicable for Universal Agent
- `subtype`: Success or error code
- triggeredEntity:
    - `entityType`: A string representing universal agent: UNIVERSAL_AGENT
    - `entityId`: ID of the Universal Agent

Get Event Details

Get event details for all events. The number of events that can be returned is limited to 600.

```
GET /controller/universalagent/v1/user/events/details&timeRange=<value>
```

Query Parameters

Parameter Name	Value	Required
timeRange	Time range specifier string as used in other parts of the Controller. For example: • last_1_hour.BEFORE_NOW.-1.1424287621109.60 • last_5_minutes.BEFORE_NOW.-1.-1.5	

Example Command

```
curl -i -X GET -u '<userName>@<accountName>:<password>'
-H 'Content-type: application/json'
http://<controller-host>:8080/controller/universalagent/v1/user/events/details?timeRange=last_5_minutes.
BEFORE_NOW.-1.-1.5
```

Response Format

```
[
  {
   "eventDetails": [
       {
        "id":<long>,
        "version":<long>,
        "name":<string>,
        "value":<string>
       }
    ]
    eventProperties: [],
    userComments: [ ],
    eventId: <long>
  },...
]
```

Response Fields

- `eventId`: ID of the event
- `version`: Version of the event
- `name`: Details of the event
- `value`: Not applicable for Universal Agent

Get Event Details By Event ID

Get details for a specific event by event id.

```
GET /controller/universalagent/v1/user/events/detailsByEventId/<eventId>
```

Path Parameters

Parameter Name	Value	Required
eventId	ID of the event	Yes

Example Command

```
curl -i -X GET -u '<userName>@<accountName>:<password>'
-H 'Content-type: application/json'
http://<controller-host>:8080/controller/universalagent/v1/user/events/detailsByEventId/1
```

Response Format

```
{
   "eventDetails": [
       {
        "id":<long>,
        "version":<long>,
        "name":<string>,
        "value":<string>
       }
    ]
    eventProperties: [],
    userComments: [ ],
    eventId: <long>
 }
```

Response Fields

- `eventDetails`: Not applicable for Universal Agent
    - `id`: ID of the event
    - version: Version of the event
    - `name`: Details of the event
    - `value`: Not applicable for Universal Agent
- eventProperties: Not applicable for Universal Agent
- userComments: Not applicable for Universal Agent
- `eventId`: Not applicable for Universal Agent

Create a Configuration Template

Create a configuration template to reference in your rulebooks using:

```
PUT /controller/universalagent/v1/user/configurations/{name}
```

The template is stored on the Controller and can be referenced from multiple rulebooks. See Rulebook Configuration Templates.

Path Parameters

Parameter Name	Value	Required
`name`	Name of the configuration template.	Yes

Example Create Template

```
curl -i -X PUT -u '<userName>@<accountName>:<password>' -H 'Content-type: application/json' -H 'Accept:
application/json' http://localhost:8080/controller/universalagent/v1/user/configurations/440startedconfig -d ' {
"name": "450startedconfig",
 "config": {
 "state":"started",
 "version": "4.5.0"
 },
 "model": "NameValue"
}'
```

Response Format

Response Payload: Created template

```
 {
    "name": "450startedconfig",
    "model": "NameValue",
    "config": {
        "state": "started",
        "version": "4.5.0"
    },
    "referredRuleBooks": []
}
```

HTTP Response Status Code: 200 OK

Response Fields:

- `name`: Name of the configuration template
- `model`: Model of the template. NameValue is the only supported model.
- `config`: Configuration properties, a list of name-value pairs.
- `referredRuleBooks`: A list of the rulebooks that refer to this template.

Update a Configuration Template

Update an existing configuration template using:

```
PUT /controller/universalagent/v1/user/configurations/{name}
```

Parameter Name	Value	Required
`name`	Name of the configuration template.	Yes

Example Update Template Command

```
curl -i -X PUT -u '<userName>@<accountName>:<password>' -H 'Content-type: application/json' -H 'Accept:
application/json' http://localhost:8080/controller/universalagent/v1/user/configurations/450startedconfig -d ' {
"name": "450startedconfig",
 "config": {
 "state":"started",
 "version": "4.5.0" # changed
 },
 "model": "NameValue"
}
```

Response Format

Response Payload: Changed template

HTTP Response Status Code: 200 OK

```
{
    "name": "450startedconfig",
    "model": "NameValue",
    "config": {
        "state": "started",
        "version": "4.5.0"
    },
    "referredRuleBooks": []
}
```

Response Fields

- `name`: Name of the configuration template
- `model`: Model of the template. NameValue is the only supported model.
- `config`: Configuration properties, a list of name-value pairs.
- `referredRuleBooks`: A list of the rulebooks that refer to this template.

Delete a Configuration Template

Delete a configuration template using:

```
DELETE /controller/universalagent/v1/user/configurations/{name}
```

Path Parameters

Parameter Name	Value	Required
name	Name of the configuration template.	Yes

Example Delete Template Command

```
curl -i -X DELETE -su '<controller-user>@<controller-account>:<controller-password>' -H "Content-type:
application/json" -H "Accept:application/json" \
        "http://<controller-host>:<controller-port>/controller/universalagent/v1/user/configurations
/450startedconfig"
```

Response

Response Payload: None

HTTP Response Status Code: 200 OK, if the template is present, 404 Not Found, for non-existent templates.

Fields: Not Applicable

View Configuration Templates

View specific templates by name or view a list of all templates using:

```
GET /controller/universalagent/v1/user/configurations
GET /controller/universalagent/v1/user/configurations/{name}
```

Path Parameters

Parameter Name	Value	Required
name	Name of the configuration template.	Yes

Example Command: View Specific Template

```
curl -i -X GET -su '<controller-user>@<controller-account>:<controller-password>' -H "Content-type: application
/json" -H "Accept:application/json" \
        "http://<controller-host>:<controller-port>/controller/universalagent/v1/user/configurations
/450startedconfig"
```

Example Command: View All Templates

```
curl -i -X GET -su '<controller-user>@<controller-account>:<controller-password>' -H "Content-type: application
/json" -H "Accept:application/json" \
        "http://<controller-host>:controller-port>/controller/universalagent/v1/user/configurations"
```

Response Payload for Specific Template

```
 {
    "name": "450startedconfig",
    "model": "NameValue",
    "config": {
        "state": "started",
        "version": "4.5.0"
    },
    "referredRuleBooks": []
}
```

Response Payload: All Templates

```
[
    {
        "name": "450startedconfig",
        "model": "NameValue",
        "config": {
            "state": "started",
            "version": "4.5.0"
        },
        "referredRuleBooks": []
    },
    {
        "name": "controllerConfig",
        "model": "NameValue",
        "config": {
            "controller_key": "some",
            "controller-port": "9000",
            "controller-host": "localhost"
        },
        "referredRuleBooks": []
    }
]
```

HTTP Response Status Codes:

- `200` OK
- `404 Not Found`: when template is not present.

Response Fields

- `name`: Name of the configuration template
- `model`: Model of the template. NameValue is the only supported model
- `config`: Configuration properties, a list of name-value pairs
- `referredRuleBooks`: A list of the rulebooks that refer to this template
- `for view all templates`: List of configuration templates
- `[]` if no configuration templates are present

Troubleshoot Universal Agent Setup

Universal Agent Log Files

The Universal Agent log files can help you troubleshoot your Universal Agent configuration. These logs provide detailed information on the activities of the Universal Agent itself, but for a complete picture of the setup, they may not be the only resource you rely upon. Since the JVM has its own log context, you may need to check its logs to analyze unexpected behavior or results.

Troubleshooting Workflow

The general recommended workflow for troubleshooting Universal Agent setup issues is:

1. Missing or incorrect connection settings are a common agent misconfiguration issue. Check the rulebooks applied by the Universal Agent to make sure that the connection settings are correct. Verify the account name and key, along with the controller host and port.
2. Check the Universal Agent log file. The log files are located in the `log` directory for the monitor. This log file may be in one of the following locations:

    a. If the runtime_directory attribute is not specified in the Java Agent rule:
    `<universal_agent_home>/monitor/java/ua<version>/logs`
    b. If the runtime_directory attribute is specified in the Java Agent rule:
    `<runtime_directory>/logs`
    Where the exact location of runtime_directory is defined by the rule attribute.
3. Enrich the JVM log with Universal Agent logging by setting the following environmental variable in the JVM environment. Note that this requires a restart of the JVM:

    ```
    export APPDYNAMICS_UA_DEBUG=true
    ```

Application Security Monitoring

> Cisco Secure Application is available for the SaaS environment only.

AppDynamics with Cisco Secure Application reduces the risk of security exposure without compromising the delivery speed for an APM-managed application. Normally, the traditional vulnerability scanning occurs before the application is launched to production, and then continues on a monthly, or quarterly cadence. As soon as the app is deployed to production, new security gaps, and zero-day exploits make the application vulnerable despite pre-production testing. Cisco Secure Application enables continuous vulnerability assessment and protection by scanning code execution to prevent possible exploits.

Cisco Secure Application enables:

- The IT Operations team responsible for performance monitoring to gain real-time access to all security events.
- Application security (AppSec) developers and application developers to gain insights into violations of best practices and to collaborate on a solution without friction.
- AppSec and DevOps to add security into the existing automation, which benefits the DevSecOps environment.
- Businesses to operate at a faster pace with a lower risk profile due to constant run-time protection, real-time remediation, and security automation.

The Cisco Secure Application features are built into AppDynamics Java Agent. To monitor the application security, you must enable the security for the application using the Cisco Secure Application dashboard. Use the **Security Events** widget on the AppDynamics Application dashboard to navigate to the Cisco Secure Application dashboard. To view the **Security Events** widget within AppDynamics Performance Monitoring (APM), enable your SaaS account with the subscription license for Secure Application. See License Entitlements and Restrictions.

Cisco Secure Application Components

Cisco Secure Application uses the combination of the APM Agent, Controller, and Cisco Secure Application dashboard to monitor the security of the applications.

- **Java APM Agent**: Cisco Secure Application library is bundled with the Java Agent. The agent communicates with the Cisco Secure Application se rvice within the Controller, which is maintained in the cloud.

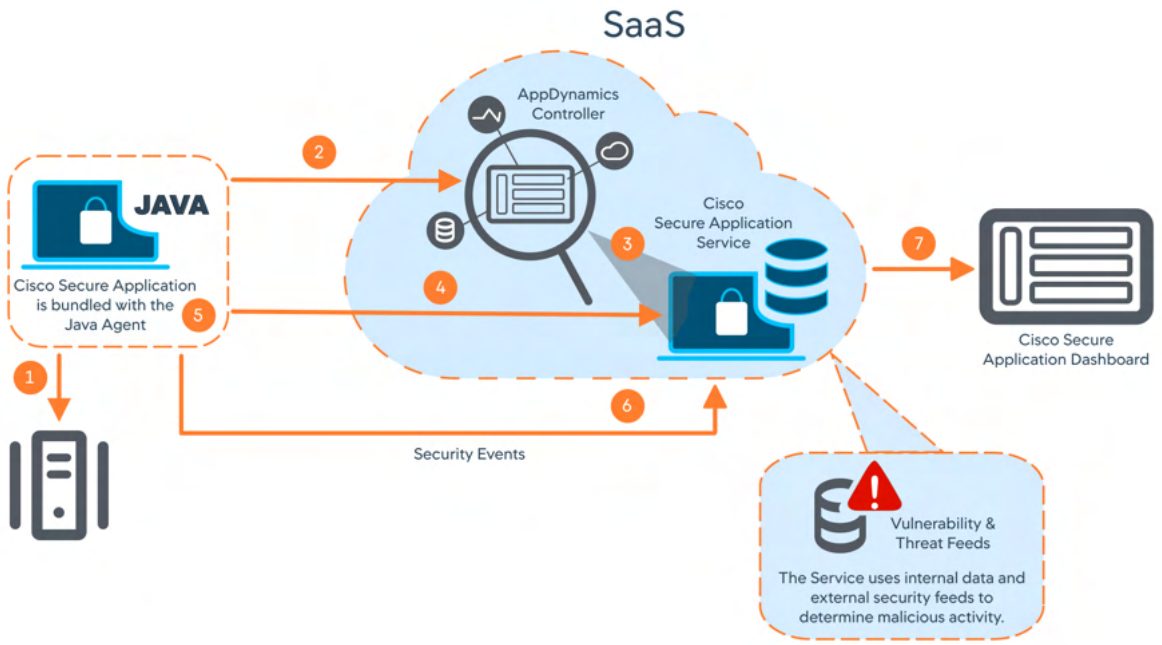
- **AppDynamics Controller**: The Cisco Secure Application service is maintained in the cloud by AppDynamics.
  The APM Agent sends data to the service within the Controller. The service analyzes the data to protect against different types of attacks and vulnerabilities and then the service provides the analysis to the dashboard. For information about the attacks and vulnerabilities that Cisco Secure Application detects, see Cisco Secure Application Policies.
  It uses external feeds along with internal data to analyze the behavior of the application. It analyzes the CVEs (Common Vulnerabilities and Exposures) against a curated vulnerability feed.
  The service can detect:
  - An attack when it is enabled in the policy and abnormal behavior is detected.
  - A vulnerability when it is enabled in the policy and when the associated behavior and the library used are considered vulnerable.

- **Cisco Secure Application Dashboard**: A graphical representation of all the analyzed data. You can view this dashboard based on the role defined in the AppDynamics Controller. The data is updated on the dashboard when the service within the Controller sends the analyzed data to the dashboard.

Cisco Secure Application Architecture

This is a high-level architecture of Cisco Secure Application.

> The APM Agent (Java Agent) communicates to the Cisco Secure Application service through the AppDynamics Controller.



1. You install the APM Agent and then add the Cisco Secure Application license.
2. The APM-managed application runs and the Java Agent retrieves the data to send to the Controller.
3. The Cisco Secure Application service retrieves the application, tiers, and nodes data from the Controller.
4. The APM Agent communicates with the Cisco Secure Application service to check if the security is enabled for the application.
5. If the security is enabled, then the agent downloads the configuration along with the policies from the Cisco Secure Application service.
6. Based on the configured policies, the agent sends the security events to the Cisco Secure Application service.
7. The service collects all the data, analyzes the application behavior, and then provides the analyzed data to the Cisco Secure Application dashboard.

Cisco Secure Application Requirements

This page describes the requirements, supported environments, and versions supported by the Cisco Secure Application.

Software Requirements

The Cisco Secure Application capabilities are integrated with the Java APM Agent, and therefore works on the following platforms:

- Operating Systems - AIX, Linux, and Windows
- Containers - All major container systems
- Languages - Java versions 7 to 14, inclusive, Oracle, OpenJDK, Azul, and BM
- Application Framework Support - All major frameworks such as Spring (see Java Supported Environments)
- Application Server Platform Support - All major Application Servers (see Java Supported Environments)

Review Install App Server Agents for instructions and guidance. AppDynamics provides an Agent Installer that simplifies the agent installation process and streamlines the deployment of Java and Machine Agents.

Resource Utilization and Performance Impact

Within the Java Agent, the Cisco Secure Application capabilities require:

- Disk - 4 MB (install) and <15 mb (daily usage)
- Memory - consistently heap/mem usage should be 4-6 MB
- CPU - consistently < 1% - spikes < 5%
- Latency - consistently < 4-6 ms per transaction (average is lower) - spikes < 10ms
- Classes Instrumented: rules (12) classes (generally < 30 - based on implementations of some of the interface rules)

Browser Requirements

Currently, Google Chrome is the supported browser for accessing Cisco Secure Application dashboard.

Before You Begin

To use Cisco Secure Application within the Java APM Agent, ensure:

- That you have met the preceding Cisco Secure Application Requirements.
- You have enough Cisco Secure Application licenses to cover your Java APM agent usage within the applications you plan on securing. To get a Cisco Secure Application license, contact the AppDynamics sales representative, or email salesops@appdynamics.com.
- A Java APM Agent >= 21.3.0 is installed and licensed.
- Controller >= 21.4.2.
- That you have successfully configured the Java Agent in your application environment.

Getting Started with Cisco Secure Application

This page provides a step-by-step procedure on how to get started with Cisco Secure Application.

Step	Task	Reference
Step 1: Ensure: • All requirements are met • The licensed version of APM Agent is installed.	Check the required version of the Controller and APM Agent. Cisco Secure Application is supported on a SaaS Controller only.	• Install the Java Agent • Cisco Secure Application Requirements
Step 2: Get the Cisco Secure Application license	Contact the AppDynamics sales representative or email salesops@appdynamics.com.	Cisco Secure Application Requirements
Step 3: Assign roles using the AppDynamics Administration Console.	• Assign the **Configure Cisco Secure Application** account permission to the users who are required to modify configurable fields on the Cisco Secure Application dashboard. • Assign **View Cisco Secure Application** account permissions to users who are required to only monitor the dashboard.	Account Permissions
Step 4: Click the **Security Events** widget on an application flowmap.	Launch the required AppDynamics Application dashboard using your account, and then click **Security Events** on the right pane. This redirects you to the Cisco Secure Application dashboard.	Monitor Application Security Using Cisco Secure Application
Step 5: From the Cisco Secure Application Dashboard navigate to the **Applications** page, and then set **Security Setting** as **Enabled** for the target application. This is applicable for all users with Configure permission for Cisco Secure Application.	The **Security Setting** value is set to **Inherit** by default for all applications that inherit the non-configurable tenant setting of **Disabled**. To enable security for an application you must set **Security Setting** to **Enabled**.	Monitor Security Status of Applications
Step 6: From the **Applications** page, verify that the application nodes are registered and active.	From the **Applications** page, check the **Active Nodes** and **Registered Nodes** fields for the specific application. Ensure that the application nodes are active. If the nodes are not active, then the application security data is not displayed on the dashboard.	Monitor Security Status of Applications
Step 7: From the **Policies** page, create **Vulnerability** policies for all applications. This is applicable for all users with Configure permission for Cisco Secure Application.	Create required **Vulnerability** policies for all applications to detect the **Library** vulnerabilities.	
Step 8: From the **Libraries** page view the risk-sorted libraries of secured applications.	The **Libraries** page displays all the existing libraries of application(s) based on the selected application scope. You can use the risk score to prioritize the remediation task.	Monitor Libraries

Based on your analysis from the dashboard you can create or modify policies and then view the different pages on the dashboard to monitor the security of APM-managed applications. See Cisco Secure Application Policies.

Monitor Application Security Using Cisco Secure Application

Monitor the Home Page of Cisco Secure Application

Monitor Security Status of Applications

Cisco Secure Application Policies

Cisco Secure Application **Policies** specify actions when an attack or vulnerability matches your defined criteria. They also determine what types of attacks and vulnerabilities are addressed. You can create and configure policies to specify an action to mitigate the attacks and vulnerabilities.

To monitor the security of the application, you must create policies. To create policies, you require the Configure permission for Cisco Secure Application. By default, Cisco Secure Application includes a policy that provides the best detection of all the attacks and vulnerabilities, reducing the false positives.

Cisco Secure Application scans these types of attacks and vulnerabilities:

Attack Type	Description
Local File Inclusion (LFI)	Software imports, requires, or includes executable functionality (such as a library) from a source that is outside the intended control sphere.
Path Traversal (PATH)	Uses external input to construct a pathname that is intended to identify a file or directory that is under a restricted parent directory. However, the software does not properly neutralize special elements within the pathname. This may cause the pathname to resolve to a location outside of the restricted directory.
Remote Code Execution (RCE)	Triggers arbitrary code execution over a network. This occurs through the framework's backdoor, using content such as XML, JSON, and Java serialization.
Untrusted Deserialization (DESERIAL)	The application deserializes untrusted data without sufficiently verifying that the resulting data is valid.

Vulnerability Type	Description
Cookie: HttpOnly (C_HTTP)	Cookie: HttpOnly: A cookie with the HttpOnly attribute is inaccessible to the JavaScript Document.cookie API; it is sent only to the server. For example, cookies that persist server-side sessions do not need to be available to JavaScript, and should have the HttpOnly attribute. This precaution helps mitigate cross-site scripting (XSS) attacks.
Cookie: SameSite (C_SITE)	Cookie: SameSite: The SameSite attribute enables servers to prevent cookies from being sent with cross-origin requests (where Site is defined by the registrable domain). This provides protection against cross-site request forgery attacks (CSRF).
Cookie: Secure (C_SECURE)	Cookie: Secure: A cookie with the Secure attribute is sent to the server only with an encrypted request over the HTTPS protocol. It is never sent with unsecured HTTP, and therefore cannot be accessed by a man-in-the-middle attacker.
HTTP Header: X-Content-Type-Options (H_CONTENT)	The X-Content-Type-Options response HTTP header is a marker used by the server to indicate that the MIME types advertised in the Content-Type headers should not be changed. This provides a way to opt-out of MIME type sniffing, or, in other words, to say that the MIME types are deliberately configured.
HTTP Header: Content-Security-Policy (H_CSP)	HTTP Header: Content-Security-Policy: The HTTP Content-Security-Policy response header allows web site administrators to control resources the user agent is allowed to load for a given page. With a few exceptions, policies mostly involve specifying server origins and script endpoints.
HTTP Header: X-Frame-Options (H_FRAME)	HTTP Header: X-Frame-Options: The X-Frame-Options HTTP response header can be used to indicate whether or not a browser should be allowed to render a page in a <frame>, <iframe>, <embed> or <object>. Sites can use this to avoid click-jacking attacks, by ensuring that their content is not embedded into other sites.
HTTP Header: Strict-Transport-Security (H_STS)	The HTTP Strict-Transport-Security response header (HSTS) is used in a website to send information to the browsers that the website should only be accessed using HTTPS, instead of using HTTP. This header is used to prevent man-in-the-middle attacks or sensitive data leakage.
HTTP Header: X-XSS-Protection (H_XSS)	HTTP Header: X-XSS-Protection: The HTTP X-XSS-Protection response header is a feature of Internet Explorer, Chrome and Safari that stops pages from loading when they detect reflected cross-site scripting (XSS) attacks.
Uncaught Exception (EXC)	Uncaught Exception: this is a very common issue - the runtime generally has built in security detection mechanisms in the form of Exceptions that are often times not "caught" and/or "not logged" - or if they are - it's buried in a file that is not reviewed until days or weeks after a security breach has occurred. This reviews ALL security related Exceptions across the entire runtime and using the security policy can report and/or take other additional actions on the information.
Vulnerable Libraries (LIB)	Detect all applicable vulnerabilities to the application's services and tiers as reported in the National Vulnerability Database (NVD) .

Create a Security Policy

To create a policy for an attack or vulnerability, perform the following steps:

1. Click **Policies** > **Create New Policy**.
2. From the **Add Policy** dialog, select the required criteria for the attacks in these fields:
    - **Policy Type**: Select **Attack Policy** to create a policy for a specific attack. Select **Vulnerability Policy** to create a policy for a specific vulnerability.
    - **Name**: Select the type of attack or vulnerability for which you require the policy.
    - **Application**: Select the application that includes the tiers or services on which you require to apply the policy.
      Select **All** to apply the policy on specified tiers or services of all the applications.
    - **Tier**: Select the required application-specific tier or service to apply the policy.
      Select **All** to apply the policy on all the tiers or services.

      > AppDynamics recommends that you review the default policy and if needed create a policy for specific applications and tiers.

    - **Action**: Select the action for this policy.
        - For Attacks: Select **None** for no notifications on the **Attacks** page; select **Detect** to detect the attack and display the details on the **Attacks** page; or select **Block** to block a specific attack and to display it as **Blocked** on the **Attacks** page.
        - For Vulnerabilities: Select **None** for no notifications on the **Vulnerabilities** page; select **Detect** to detect the vulnerability and display the details on the **Vulnerabilities** page; select **Patch** to fix the security issues.

      > **Block** is unavailable for some of the supported attacks and **Patch** is unavailable for some of the supported vulnerabilities.

3. Click **Save**.

Modify a Security Policy

To view and modify a policy for an attack or vulnerability, perform the following steps:

> You can use the **Search** filter to search based on the values of the **Name** or **Application Name** fields.

1. Click **Policies**.
2. Select the application and the tier or service whose policy you want to modify.
3. Click **Attacks** to view the list of attack policies. Click **Vulnerabilities** to view the list of vulnerability policies.
   At the bottom right corner of the page, you can view 5, 10, 20, or 50 policies in the list by selecting the **Show<*number of policies*>** dropdown.
4. Click the **Modify** icon.
5. Modify the required fields.
6. Click **Update** or click **Delete Policy** based on the requirement.

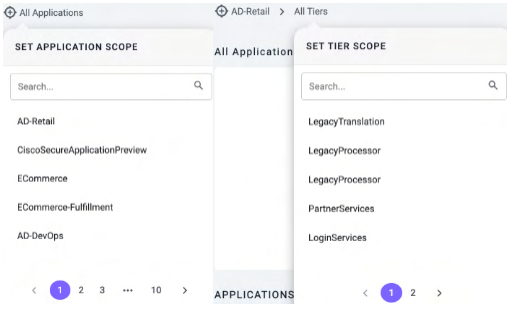
Monitor Application Security Using Cisco Secure Application

Cisco Secure Application provides a real-time dashboard that provides visibility on the security health of your applications. This dashboard is available when an application is registered with an APM agent and has the appropriate licensing. The agent sends the security events to Cisco Secure Application through the Controller.

The **Security Events** widget on the AppDynamics Application Dashboard provides high-level information about the security of a registered application. This widget displays the number of critical, warning, and normal security events. To view the details of security for the selected application on Cisco Secure Application, click the **Security Events** widget.

Select Scope for the Dashboard

Cisco Secure Application dashboard provides an option to select the required application and tier scope that will be applied across all views within this dashboard except **Policies**. By default, the application scope is the selected application on the AppDynamics dashboard prior to navigating to Cisco Secure Application.

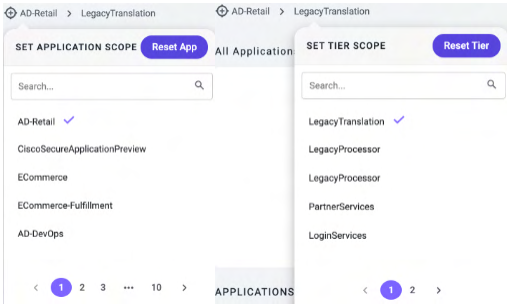


Application Scope

Perform the following to view the data on the dashboard for a specific application:

1. Click the **Application Scope Selector icon** ( ) on the top left corner of the dashboard.
2. Search for the specific application.
3. Select the application.

   To return to the default view of all applications, click the **Application Scope Selector icon** ( ) > **Reset App**



Tier Scope

Perform the following to view the data on the dashboard for a specific application tier:

1. Click **All Tiers** next to the **Application Scope Selector icon** ( ).
2. Search for the specific tier.
3. Select the tier.
   To return to the default tier view, click ***<service name>*** > **Reset Tier**.

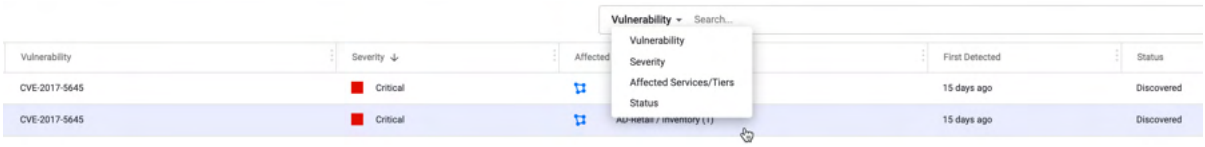
View Data Using Search Filter

The Cisco Secure Application provides a **Search** filter on various pages in the dashboard. This filter helps in getting the required data quickly.

The search filter allows you to search based on the selected category.

For example, in the following image these are the categories:

- Vulnerability
- Severity
- Affected Tiers
- Status



You can select the required search category from the dropdown list, then click the **Search** field to view the list of values corresponding to the category. You can also find the required value as you type.

If you do not require the exact match, you can enter the required generic value in the **Search** field.

For example, consider that you want to search for all applications that start with a specific prefix, AD. You can select the search category as **Application** an d enter AD in the **Search** field.

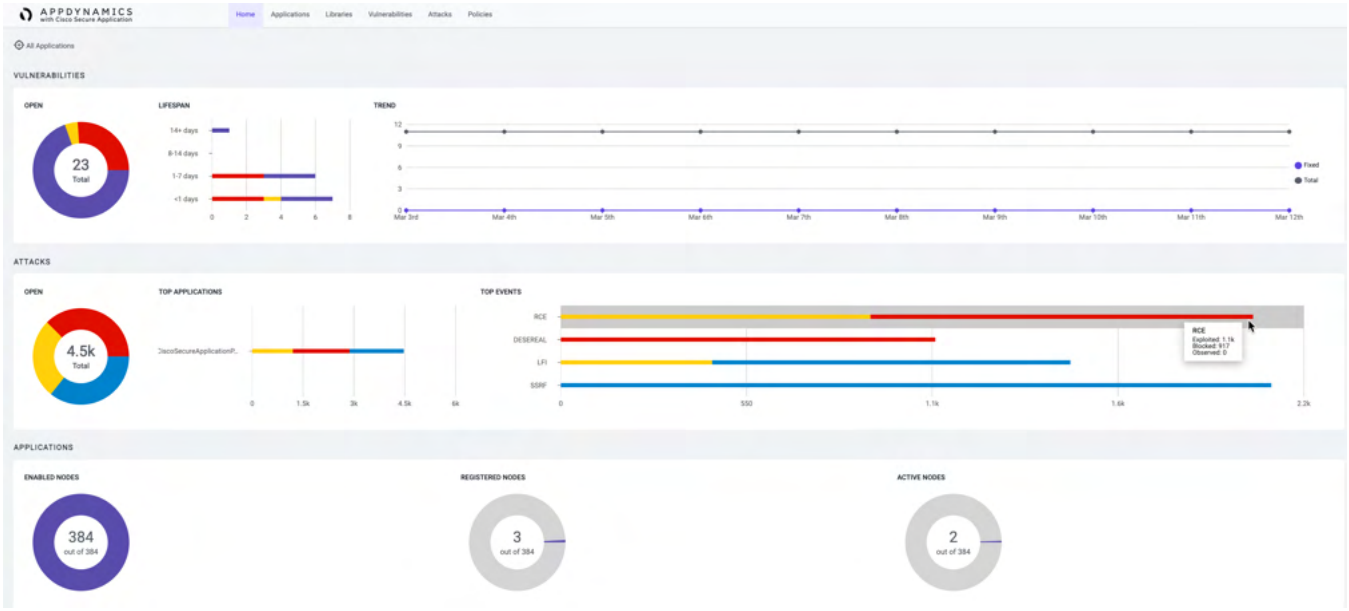
> You can search using one or all the categories, but each category can have a single search value. A category is disabled when you specify a search value for that category, but you can continue to select another available category and specify its search value. These search values act as filters. You can remove the search values to remove the search filter.

Cisco Secure Application provides a real-time dashboard that displays these pages:

- **Home**: This page provides an overall overview of attacks and vulnerabilities of monitored applications.
  See Monitor the Home Page of Cisco Secure Application.
- **Applications**: This page provides the details of monitored nodes that are registered with Cisco Secure Application for the managed applications.
  See Monitor Security Status of Applications.
- **Libraries**: This page provides details of the existing libraries that require remediation.
  See Monitor Libraries.
- **Vulnerabilities**: This page provides information about all discovered vulnerabilities.
  See Monitor Vulnerabilities.
- **Attacks**: This page provides information about all detected attacks.
  See Monitor Attacks.
- **Policies**: This page allows you to create or customize the policies for vulnerabilities and attacks.
  See Cisco Secure Application Policies.

Monitor the Home Page of Cisco Secure Application

The **Home** page is the landing page for Cisco Secure Application. This page provides a quick view of the security of the selected application. The following image is an example of the **Home** page:



The Home page includes the following panes:

- Vulnerabilities
- Attacks
- Applications

> By default, this page displays an overview of the selected application. For information about selecting a specific application or service, see Select Application Scope at Monitor Application Security Using Cisco Secure Application.

Vulnerabilities

The Vulnerabilities pane includes a real-time trend graph that shows the number of both fixed and open vulnerabilities.

Chart	Description
OPEN	The pie chart represents the total number of open vulnerabilities. For example, the preceding image shows 23 open Vulnerabilities. This chart displays visualization for the number of vulnerabilities based on these severity levels: • Critical = Red • High = Orange • Medium = Yellow • Low = Purple Hover on each severity to view the number of related open vulnerabilities. To display all charts in a pane based on a specific severity, click the severity on the pie chart. To return to the complete chart, click the same severity again.
LIFESPAN	This chart displays the number of days the vulnerability is open versus the severity of the vulnerability (critical, high, medium, or low)
TREND	This chart displays the number of open tickets versus the number of fixed tickets. This shows the trend of fixing the open vulnerabilities.

Attacks

The **Attacks** pane includes the visualization for the number of open attacks.

Chart	Description

OPEN	This pie chart represents the total number of open attacks. For example, the preceding image shows 4.5K open attacks. This chart displays the number of attacks based on these states: • Observed = Blue • Exploited = Red • Blocked = Yellow Hover on the required state to view the number of open attacks in that state. To display all charts of the data based on a specific state, click the state on the pie chart. To return back to the complete chart, click the same state again.
TOP APPLICATIONS	This chart displays the top 10 applications based on open attacks per application. If you select a specific application scope, then only that application is displayed. To view all the applications, you require to reset the application scope. For more information about changing the scope of the application, see Monitor Application Security Using Cisco Secure Application. These applications are in either an **exploited**, **blocked**, or **observed** state versus the total number of open attacks on the application. Hover on each state to view the number of blocked, exploited, and observed open attacks.
TOP EVENTS	This chart displays the top 10 attack events that are in an **exploited**, **blocked**, or **observed** state versus the total number of open attacks on the events. Hover on each state to view the number of blocked, exploited, and observed open attacks.

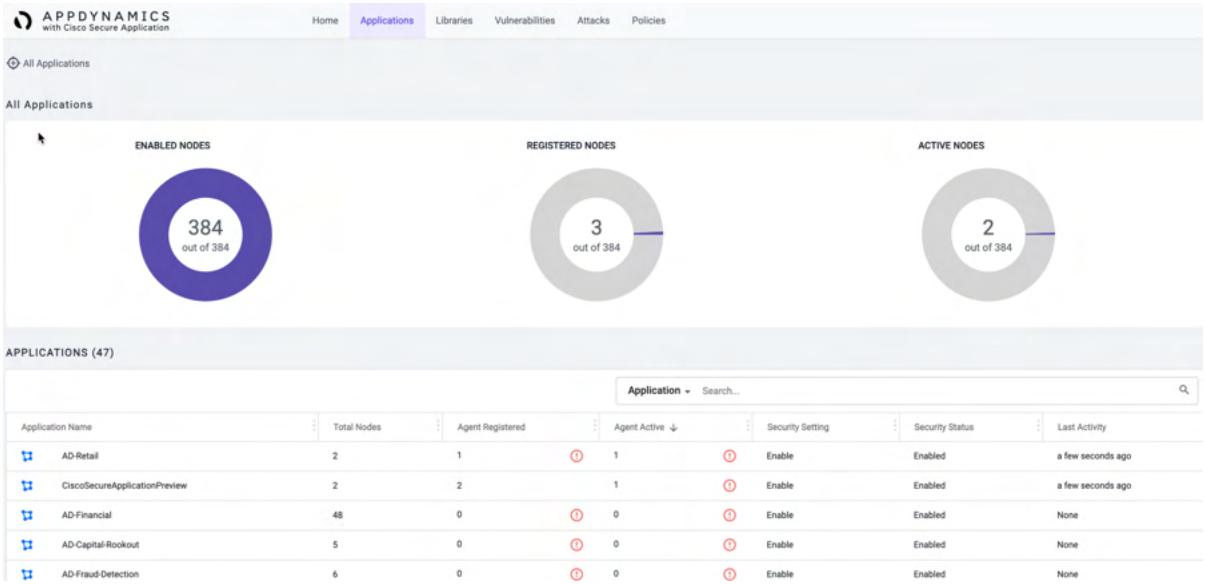
Applications

The Applications pane includes the visualization for the nodes of the managed applications.

Chart	Description
ENABLED NODES	The pie chart highlights the total number of nodes that have security enabled for the applications. The purple color represents the number of nodes enabled for security.
REGISTERED NODES	The pie chart highlights the number of nodes registered with the Cisco Secure Application. The purple color represents the number of nodes that are registered for the security setting.
ACTIVE NODES	The pie chart highlights the number of nodes that are actively communicating with the Controller. The purple color represents the number of active nodes.

Monitor Security Status of Applications

This page focuses on the current state of security capabilities across applications, tiers, and nodes. You can control the security setting (with the configure permission) and identify the issues with the agents that are not registered or are inactive. This image is an example of the **Applications** page:



The top pane includes the visualization of the number of enabled, registered, and active nodes for all applications.

- ENABLED NODES - nodes with security enabled.
- REGISTERED NODES - nodes with APM agents that are registered with Cisco Secure Application.
- ACTIVE NODES - nodes with currently active registered APM agents.

- Use the **Application Scope Selector icon** ( ) to view the data for a specific application, then select the required tier to view the data for a specific service of the selected application. For information about selecting the application and tier scope, see Monitor Application Security Using Cisco Secure Application.
- In the **Applications** table, you can click the application row to drill down to all the tiers for that application. Also, you can click a tier row to drill down to the corresponding nodes.
- You can use the **Search** filter for the following categories**:**
  - **(Applications) Application**, and **Security Status**
  - **(Tiers) Tier**, and **Security Status**
  - **(Nodes) Node**, **Security Status**, and **Active Node**

  For more information about the **Search** filter, see **View Data Using Search Filter** in Monitor Application Security Using Cisco Secure Application.

The bottom pane provides the following details:

The total number of applications is mentioned in parenthesis.

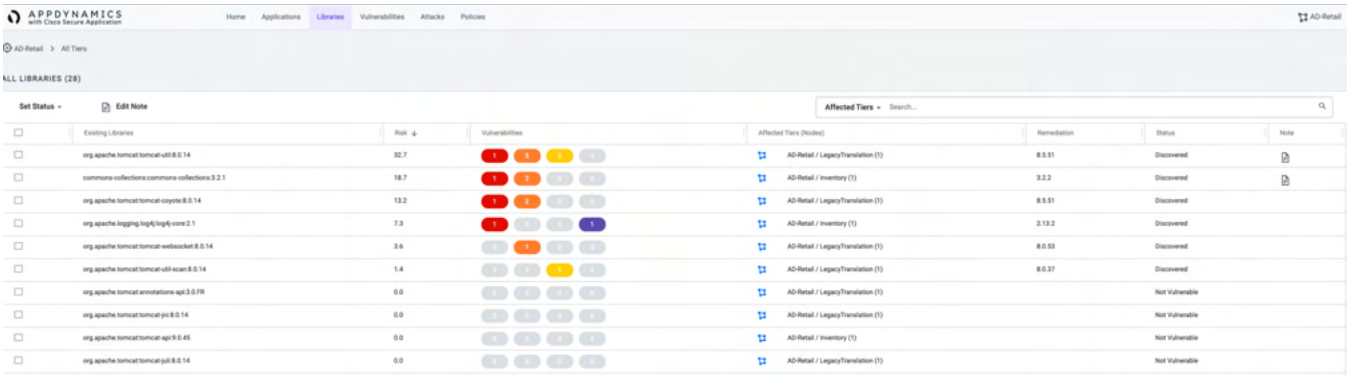
Field Name	Description
Application Name	This field displays the list of all applications. Click the flowmap icon ( ) next to an application to view the application flow map in the AppDynamics dashboard. Click this field to sort in alphabetical order. This field changes when you click to drill down to the tier list, then the node list. You can use the respective **Search** filter for each list view to quickly search the application, tier, or node.
Total Nodes	This field displays the total number of nodes that exist in the corresponding application. You can drill down to tiers or nodes list to view the total number of nodes in a tier, or a node respectively. Click this field to sort the number of nodes in increasing or decreasing order.

Registered Nodes	This field displays the number of nodes that are registered with the Cisco Secure Application for the corresponding application. You can drill down to tiers or nodes list to view the number of registered nodes in a tier, or a node respectively. The info icon () indicates that there are security-enabled nodes that are not registered with the agent. Click this field to sort the number of registered nodes in increasing or decreasing order.
Active Nodes	This field displays the number of nodes that are active for the corresponding application. You can drill down to tiers to view the number of active nodes in a tier. When you drill down to the nodes list, this field indicates if the node is active or not. The info icon () indicates a difference in the number of enabled nodes compared to active nodes. (For application and tiers) Click this field to sort the number of active nodes in increasing or decreasing order. (For nodes) Click this field to sort based on the values **Yes** or **No**.
Security Setting	The security setting is disabled by default. If you require to enable the security, select **Enable**. This field displays these security settings for the corresponding application: • **Enable**: When the security setting is enabled for the nodes. • **Disable**: When the security setting is disabled for the nodes. • **Inherit**: When the security setting is inherited from the Controller. By default, the security setting is **Disabled**. Therefore, **Security Status** is **Disabled** for the application. For tiers and nodes, the security setting is inherited from the parent application and tier respectively. You can drill down to tiers or nodes list to view the security settings of a tier, or a node respectively. You can update these settings if you have the Configure Cisco Secure Application permission. Be aware that changing security settings at one level affect other levels. When you update **Security Setting** for any object (application, tier, or node), it can affect all the nodes in the object hierarchy (**Application**>**Tier**>**Node**). This happens when **Security Setting** is set to **Inherit** for either the object or its parent object. Click this field to sort the security setting in alphabetical order.
Security Status	This field displays the security status for the corresponding application based on the security setting. This field displays if the security status is enabled or disabled for the corresponding application based on the security setting. You can drill down to tiers or nodes list to view the security status of a tier, or a node respectively. Click this field to sort the status based on whether the status is enabled or disabled. If you have the Configure permission, you can change the security status using the **Set Status** option.
Last Activity	This field provides the timeframe when the last activity is detected.
Node Version	This field is displayed only when you drill down to the nodes list. This displays the version of the APM Agent used by the tier.

Monitor Libraries

The **Libraries** page provides a list of all libraries that are in use by the corresponding applications. The page highlights the vulnerabilities and associated risks introduced by the use of those libraries.

This image is an example of the **Libraries** page:



- You can use the **Search** filter for the **Affected Tiers**, **Status**, and **Existing Libraries** categories**.** For more information about the **Search** filter, see **View Data Using Search Filter** in Monitor Application Security Using Cisco Secure Application.
- The **Set Status** and **Edit Note** bulk edit options are available if you have the **Configure Cisco Secure Application** permission.

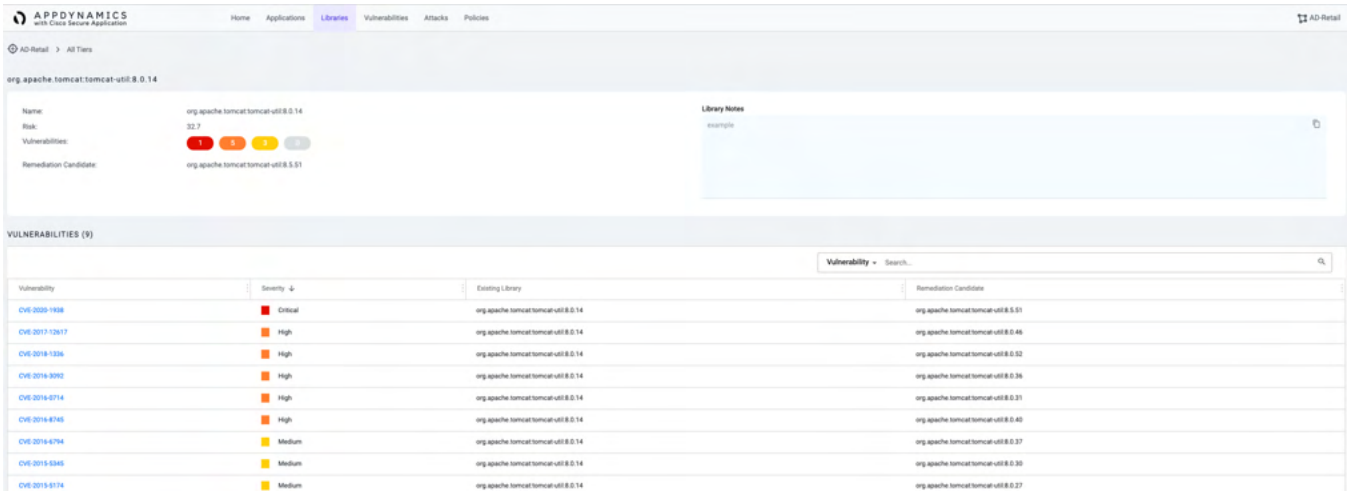
The libraries page includes these details:

Field Name	Description
Existing Libraries	Existing libraries in the selected application. You can click on the row to view the details of the library. See View Vulnerabilities for a Library. Click this field to sort the libraries in alphabetical order.
Risk	The risk score given to the library. This helps to identify which libraries require immediate remediation. The higher the risk score, the higher the impact of the collection of vulnerabilities within the library. Click this field to sort the risk score from high to low or low to high.
Vulnerabilities	The number of vulnerabilities based on severity. Severity is represented with the following colors: Red = Critical Orange = High Yellow = Medium Purple = Low If any of the preceding colors are replaced with the grey color, it indicates that there are no vulnerabilities with that specific severity. For example, **Vulnerabilities** display the grey, orange, yellow, and purple colors instead of the red, orange, yellow, and purple colors. This indicates that there are no critical vulnerabilities. Click this field to sort in increasing or decreasing order based on the number of vulnerabilities.
Affected Tiers	The application tier that is vulnerable because of its relationship to the corresponding library. Click the flow map icon ( ) next to an application to view the application flow map in the AppDynamics dashboard.
Remediation	The recommended version of the library that can be used for remediation.

Status	The status of the vulnerable libraries. By default, when a vulnerability is detected the value is **Discovered**. The status value can be: • **Discovered** (at least one vulnerability is discovered in the library) • **Confirmed** (Library is reviewed) • **Fixed** (Library is fixed) • **Ignored** (the library does not require to be considered when remediating vulnerability libraries because of some mitigations or exception) • **Not Vulnerable** (no vulnerabilities are found in the library) The **Discovered** and **Fixed** status are auto-populated. If you have the **Configure Cisco Secure Application** permission, you can change the **Status** by selecting the required libraries and using the **Set Status** option. ⓘ You must have the configure permission to view and use the **Set Status** option.
Note	Notes can be used to share information with other users or document findings during the review of a vulnerability. If you have the Configure permission, you can add notes by selecting a library and using the **Edit Note** option. Without the Configure permission, the **Edit Note** option is unavailable.

View Vulnerabilities for a Library

To view all the vulnerabilities within a specific library, click the row on the **Libraries** page. This directs you to the library details page with the following information:



The upper pane displays the details about the vulnerable library including the risk involved, any remediation to be taken, and the **Library Notes**. You can copy the notes if required.

If you have the Configure Cisco Secure Application permission, then you can add or edit the **Library Notes**.

The bottom pane provides the following details:

ⓘ You can use the **Search** filter to view the vulnerability details list based on the **Severity** or the **Vulnerability** value. For more information about the **Search** filter, see **View Data Using Search Filter** in Monitor Application Security Using Cisco Secure Application.

Field Name	Description
Vulnerability	The vulnerabilities scanned for the selected library. Click the field to sort in increasing or decreasing order. Click the CVEs to view the vulnerability details. For information about Vulnerabilities, see Monitor Vulnerabilities.
Severity	The severity of the vulnerability. Click the field to sort the severity in alphabetical order.

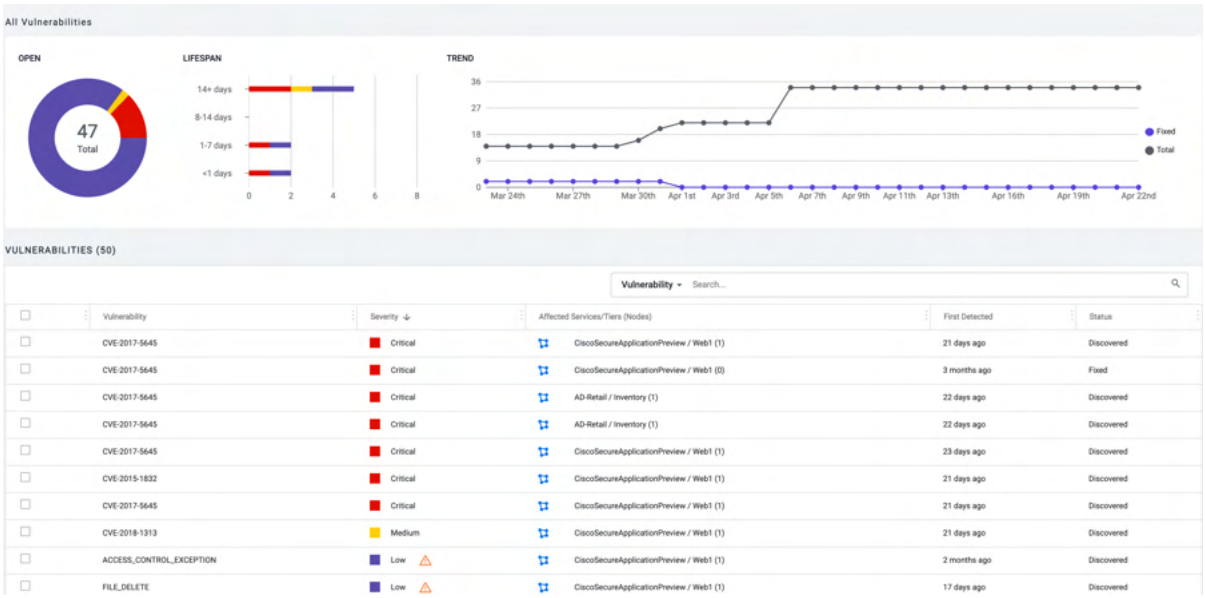
Existing Library	The library that introduced the vulnerability to the application.
Remediation Candidate	The version of the library that should be used to remediate the vulnerability.

Monitor Vulnerabilities

An application registered with Cisco Secure Application is scanned and continuously monitored for vulnerabilities. The **Vulnerabilities** page displays the list of all the scanned vulnerabilities.

When vulnerabilities are detected, a user with Configure permission can prioritize the vulnerabilities and change the status based on the details on this page.

This image shows you the scanned vulnerabilities on the **Vulnerabilities** page:



> By default, this page displays an overview of the selected application. For information about selecting a specific application or service, see Select Application Scope at Monitor Application Security Using Cisco Secure Application.

The top pane includes these charts:

Chart	Description
OPEN	This pie chart represents the total number of open vulnerabilities. Open vulnerabilities are vulnerabilities that currently exist in the runtime application, which are not patched or fixed yet and are not explicitly marked **Ignored** by the user. The chart displays the number of vulnerabilities based on the following severity: • Critical = Red • High = Orange • Medium = Yellow • Low = Purple Hover on the required severity to view the number of open vulnerabilities with that severity. If you require all the charts in the pane to display based on a specific severity, click the severity on the pie chart. To return back to the complete chart, click the same severity again.
LIFESPAN	This chart displays the number of days the vulnerability is open versus the severity of the vulnerability (critical, high, medium, or low)
TREND	This chart displays the number of open tickets versus the number of fixed tickets. This shows the trend of fixing the open vulnerabilities.

The bottom pane includes different fields and corresponding columns that provide details of the vulnerabilities:

> • Use the **Search** filter to search based on the **Vulnerability**, **Severity**, **Affected tiers**, and **Status** values. For more information about the **Search** filter, see **View Data Using Search Filter** in Monitor Application Security Using Cisco Secure Application.
> • There are few options such as **Set Status** and **Set Severity** that are available only if you have the Configure permissions for Cisco Secure Application.

Name	Description

Vulnerability	The vulnerability name and Common Vulnerabilities and Exposure (CVE) identifier.
Severity	The severity level of the corresponding vulnerability. The warning icon () next to a severity indicates that the exploit is detected in your application. You can click this icon to view the attack details page. The red icon () next to a severity indicates that this exploit is detected somewhere else in the Secure Application network. If you have Configure permission, you can change the severity by selecting the checkbox next to required vulnerabilities, and then click **Set Severity** to choose the appropriate severity. You can sort this column alphabetically.
Affected Tiers (Nodes)	The services or the tiers affected because of the corresponding vulnerability. The number in parenthesis indicates the number of nodes. Click the flow map icon() to view the AppDynamics flow map for that tier.
First Detected	The time elapsed after the vulnerability was first detected.
Status	The status of the corresponding vulnerability. The status value can be: • **Discovered** (at least one vulnerability is discovered in the library) • **Confirmed** (Library is reviewed) • **Fixed** (Library is fixed) • **Ignored** (not a library) • **Not Vulnerable** (no vulnerabilities are found in the library) The status Ignored can be updated by the developer with Configure permission for Cisco Secure Application. If you have Configure permissions, you can select the vulnerabilities using the checkbox, and then set the status by using the **Set Status** option. Without Configure permission, the **Set Status** option is unavailable.

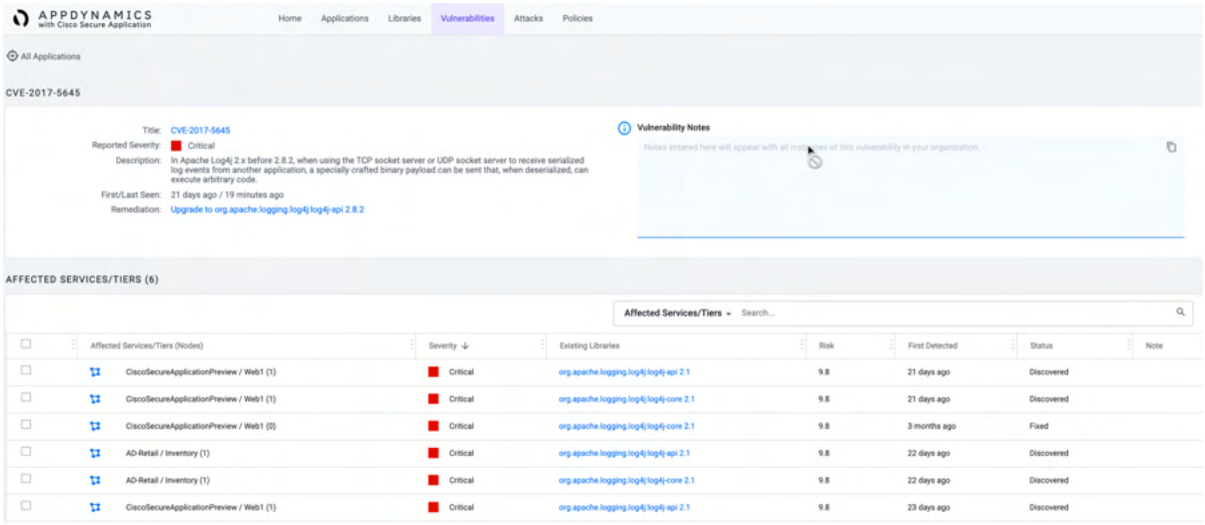
View Vulnerability Details

To prioritize vulnerabilities, you may require additional information. Click a vulnerability row to view detailed information about a vulnerability. The vulnerability details view is displayed.

The top pane displays the following details:

Field Name	Description
Title	The name of the vulnerability.
Reported Severity	The severity of the vulnerability, which can be critical, high, medium, or low.
Description	Details of the vulnerability.
First/Last seen	The timeframe when the vulnerability was first detected and when the vulnerability was last detected.
Remediation	The recommended remediation action. In the case of a vulnerable library, the version to upgrade the library for remediation.
Vulnerability Notes	If you have the Configure permission, you can add the notes under **Vulnerability Notes**. You can use the copy icon to copy the notes, if required.

The bottom pane displays the following details:

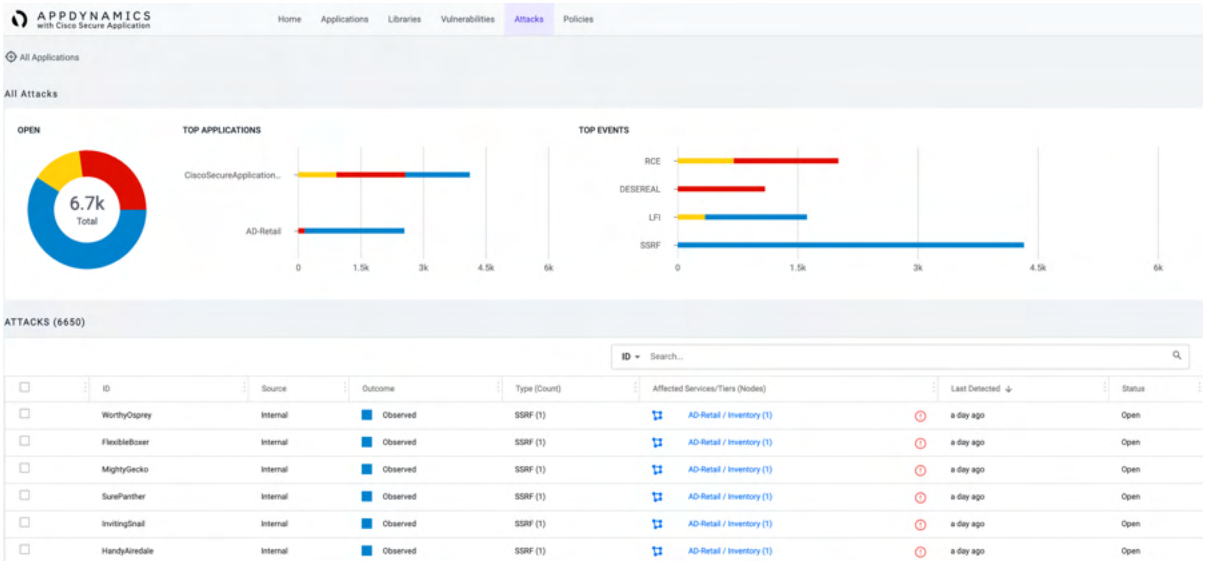


Field Name	Description
Affected Tiers (Nodes)	The services or the tiers that are affected because of the selected vulnerability. The number indicates the number of affected nodes. The flow map icon( ) directs to the AppDynamics flow map for that tier.
Severity	The severity of the vulnerability. You can edit the severity if you have Configure permission.
Existing Library	The library affected because of the vulnerability. You can click the library to view the details of the library. See Monitor Libraries.
Risk	The risk score of the vulnerability. This helps in prioritizing the affected services. A higher risk score indicates that the corresponding library on the corresponding service is at risk.
First Detected	The time elapsed since the vulnerability is detected.
Status	The status of the selected vulnerability. The status value can be: • **Discovered** (at least one vulnerability is discovered in the library) • **Confirmed** (vulnerability is reviewed) • **Fixed** (vulnerability is fixed) • **Ignored** (not a vulnerability) • **Not Vulnerable** (no vulnerabilities are found in the library) If you have Configure permissions, you can select the rows using the checkbox, and then set the status by using the **Set Status** option. Without Configure permission, the **Set Status** option is unavailable. ⓘ The **Discovered** and **Fixed** status are automatically detected based on the libraries used in the application.
Note	Under **Note**, if you have the Configure permission, you can select required checkboxes and then use the **Note** option to add notes.

Monitor Attacks

The **Attacks** page includes details of all the open and closed attacks on the managed applications.

The following image is an example of the **Attacks** page:



> By default, this page displays an overview of the selected application. For information about selecting a specific application or service, see **Select Scope for the Dashboard** at Monitor Application Security Using Cisco Secure Application.

The top pane includes these details:

Chart	Description
OPEN	This pie chart represents the total number of open attacks and displays the number of attacks based on the following state: • Observed = Blue • Exploited = Red • Blocked = Yellow Hover on the required state to view the number of open attacks in that state. If you require all the charts to display the data based on a specific state, click the state on the pie chart. To return to the complete chart, click the same state again.
TOP APPLICATIONS	This chart displays the top 10 applications based on open attacks per application. If you select a specific application scope, then only that application is displayed. To view all the applications, reset the application scope. For more information about changing the scope of the application, see Monitor Application Security Using Cisco Secure Application. These applications are in either an **exploited**, **blocked**, or **observed** state versus the total number of open attacks on the application. Hover on each state to view the number of blocked, exploited, and observed open attacks.
TOP EVENTS	This chart displays the top 10 attack events that are in an **exploited**, **blocked**, or **observed** state versus the total number of open attacks on the events. Hover on each state to view the number of blocked, exploited, and observed open attacks.

The bottom pane displays these details:

You can use the Search filter for the following categories:

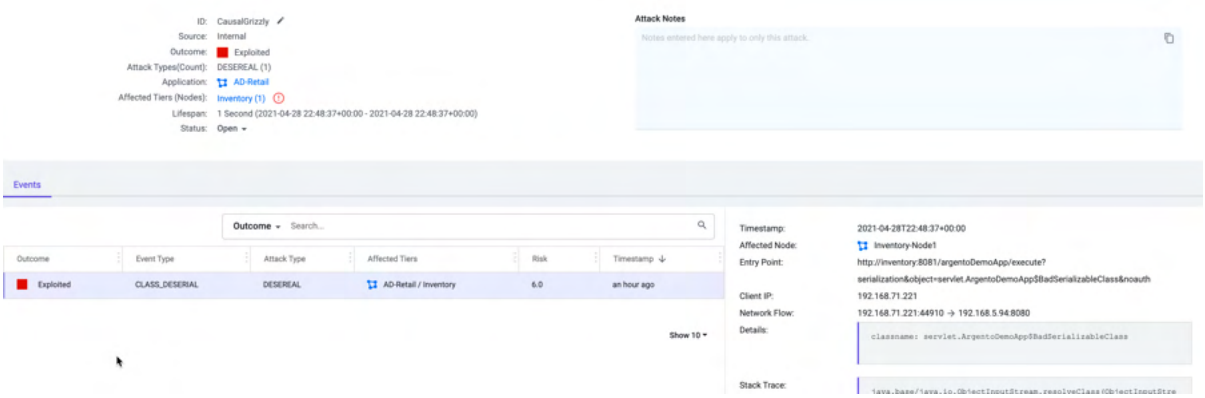
- **ID**
- **Source**
- **Outcome**
- **Affected Services/Tiers**
- **Attack Type**
- **Attack Status**

For more information about the **Search** filter, see **View Data Using Search Filter** in Monitor Application Security Using Cisco Secure Application.

Name	Description
ID	The ID of the corresponding Attack. Cisco Secure Application generates this ID. You can modify this ID on the attacks details page. To view the attack details page, click the desired row. Click this field to sort the ID alphabetically.
Source	The source of the corresponding attack. The value can be: • **Unknown** (Source is not identified) • **Internal** (Source of the attack is an internal asset) • **External** (Source of the attack is an external asset) For the attack events that are triggered by a web transaction, Cisco Secure Application uses these criteria to identify the source of attack: • **Internal:** The client is on a localhost or a local network of the server. • **External:** The client is outside the local network. Click the row for detailed information about the source of an attack. Click this field to sort the values alphabetically.
Outcome	The outcome of the corresponding attack. This provides information on these state of the attack: • **Observed**: When the events may impact the security, but any malicious intent is not determined. For example, an application opening a file outside the application directory causes **Observed** state. • **Blocked**: When the events are blocked based on the attack policy. • **Exploited**: When malicious activity is performed to impact the application's security. Click this field to sort the values alphabetically.
Type (Count)	The type of the attack and count of that attack type.
Affected Tiers (Nodes)	The application affected by the attack along with the tier name and the number of nodes. You can click to launch the application flowmap in the Appdynamics Dashboard. The info icon ( ) next to an affected tier indicates that the attacked nodes in the tier include critical or medium vulnerability.
Last Detected	The time that is elapsed since the last event within the attack. Click this field to sort the values in ascending or descending order.
Status	The status of the attack is defined as either **open** or **closed**. If you have Configure permissions, click the checkboxes for the required rows and then click the **Set Status** option to set the appropriate status. Click this field to sort based on the **Open** or **Closed** state.

View Attack Details

The attack details page provides more details of the attack. Click any attack to view the attack details page.



The top pane provides a summary of the attack.

A user (with Configure permission) can add notes under **Attack Notes** if desired. This note is visible to all users when monitoring attack details.

The bottom pane is split into left pane (a list of events correlated to the attack automatically) and right pane (the details of a selected event).

You can use the **Search** filter to filter based on the following categories:

- **Outcome**
- **Event Type**
- **Attack Type**
- **Affected Tiers**

For more information about the **Search** filter, see **View Data Using Search Filter** in Monitor Application Security Using Cisco Secure Application.

The left pane displays these details:

Field Name	Description
Outcome	The outcome of the event. This provides information on whether the selected event is **Observed**, **Blocked**, or **Exploited**.
Event Type	The type of the attack event or the vulnerability name.
Attack Type	The type of the attack such as RCE and so on.
Affected Services/Tiers	The affected application and the tier.
Risk	The risk score given for the specific event within the attack.
Timestamp	The time the event is detected.

The right pane displays the following details based on the selected event:

These fields are displayed when the events are triggered during a web transaction.

Field Name	Description
Timestamp	The date and time when the event is detected.
Affected Node	The name of the affected node. You can click the flowmap icon ( ) to view the **Tiers and Nodes** flowmap on the AppDynamics dashboard.
Vulnerabilities	The type of vulnerability used for the attack. Based on the event type, this field may not be displayed. If the value is displayed, click the value to view the vulnerability details. For information about Vulnerabilities, see Monitor Vulnerabilities.
Entry Point	The webserver URL accessed by the client in the transaction that triggered the event. Based on the event type, this field may not be displayed.

Client IP	The IP address of the remote endpoint of the connection in the transaction. This IP address can be the IP address of client machine, load balancer or proxy in a client network.
Network Flow	The network flow as observed from the node that includes the source and the destination IP address.
Details	The details about the resulting behavior of the node triggered by an inbound request. The details may change based on the event and attack type. Click **Show More** to view the **Details** dialog box. You can copy the details as per your requirement.
Stack Trace	Details of the stack trace for the corresponding event. Click **Show More** to view the **Stack Trace** dialog box. You can use this information to guide developers to the lines of code that were used to achieve the result of the event. You can copy the details as per your requirement.
Policy	The action that is used for this event based on the existing policy when the event is detected. If you have the Configure permission, you can change the policy by clicking the modify icon next to the policy. See Cisco Secure Application Policies.

Troubleshooting Cisco Secure Application Issues

This page provides common troubleshooting actions that you can take to solve Cisco Secure Application issues.

Cisco Secure Application Permissions Are Unavailable in the Role Configuration

- Check if the Controller version meets the requirements. See Cisco Secure Application Requirements.
- If the permissions are unavailable even when the Controller version is supported, ensure that you have activated the Cisco Secure Application license for the Controller. To get a Cisco Secure Application license, contact the AppDynamics sales representative, or email salesops@appdynamics.com.

Security Events Widget Is Not Displayed in an Application Flow Map

- You can view the **Security Events** widget after the license activation. Ensure that you have the Cisco Secure Application license. To get a Cisco Secure Application license, contact the AppDynamics sales representative, or email salesops@appdynamics.com.
- If the widget is not displayed even when the license is activated, then confirm with the administrator that the account has the necessary permissions to view or configure Cisco Secure Application. See Account Permissions.

Number of Registered Nodes and Active Nodes Does Not Match

On the **Applications** page, drilldown in the application to identify the inactive node. Check the APM agent logs on the inactive node(s). See Troubleshooting Java Agent Issues.

Vulnerabilities or Attacks Are Not Displayed on the Home, Vulnerabilities, or Attacks Page

- Ensure that the **Security Setting** is set to **Enabled** on at least one of the applications, tiers, or nodes within the **Applications** page. Also, that the agents are registered and active. See Monitor Security Status of Applications.
- If the vulnerabilities and attacks are not displayed even when the **Security Setting** is enabled, review the following troubleshooting scenarios to identify other potential issues.

Number of Enabled Nodes and Registered Nodes Does Not Match

- Check if the nodes that have **Security Status** set to **Enabled** use the APM Agent version that meets the requirements. See Cisco Secure Application Requirements.
- If the agent version is supported, then drilldown in the **Application** view to identify the unregistered node(s). Check the APM agent logs on the unregistered node(s). See Troubleshooting Java Agent Issues.

Security Setting is Enabled and No Libraries Are Listed

- Confirm that there are active nodes in the **Application** view. See Monitor Security Status of Applications.
- It is also possible that there are no third-party libraries used in the monitored applications.

Security Setting is Enabled and No Vulnerabilities Are Listed

- Confirm that there are active nodes in the **Application** view. See Monitor Security Status of Applications.
- Verify that there is a vulnerability policy enabled, and that it has an action of **Detect** or **Patch** for a monitored application with active nodes. See Cisco Secure Application Policies.
- It is also possible that there are no vulnerabilities within the third-party libraries used within the monitored applications or observed in the application behavior.

Security Setting is Enabled and No Attacks Are Listed

- Confirm that there are active nodes in the **Application** view. See Monitor Security Status of Applications.
- Verify that there is an attack policy enabled, and that it has an action of **Detect** or **Block** for a monitored application with active nodes. See Cisco Secure Application Policies.
- It is also possible that there are no attacks detected in the monitored applications.

End User Monitoring

AppDynamics End User Monitoring (EUM) provides end-to-end visibility on the performance of your web and mobile applications. EUM helps you troubleshoot problems such as slow web responses, Ajax errors, mobile network requests, and IoT application errors. EUM provides metrics on application performance and user activity, such as:

- How server performance impacts your web, mobile, and device performance
- How third-party APIs impact your web, mobile, and device performance
- Where your heaviest loads originate
- How your users connect to and navigate your application

Installation

- Inject the JavaScript Agent
- Instrument iOS Applications
- Instrument Android Applications
- Instrument Applications with the IoT C/C++ SDK
- Instrument Applications with the IoT Java SDK

Configuration

- Configure the Controller UI for Browser RUM
- Configure the Controller UI for Mobile RUM
- Configure IoT Application Monitoring
- Configure the JavaScript Agent

Using EUM

- Browser Snapshots
- Pages & Ajax Requests View
- Synthetic Scripts
- Mobile App Dashboard
- Monitor Applications with the IoT Dashboards

Extensibility and Reference

- Customize the iOS Instrumentation
- Customize the Android Instrumentation
- Browser RUM Metrics
- Browser Synthetic Metrics
- Mobile RUM Metrics

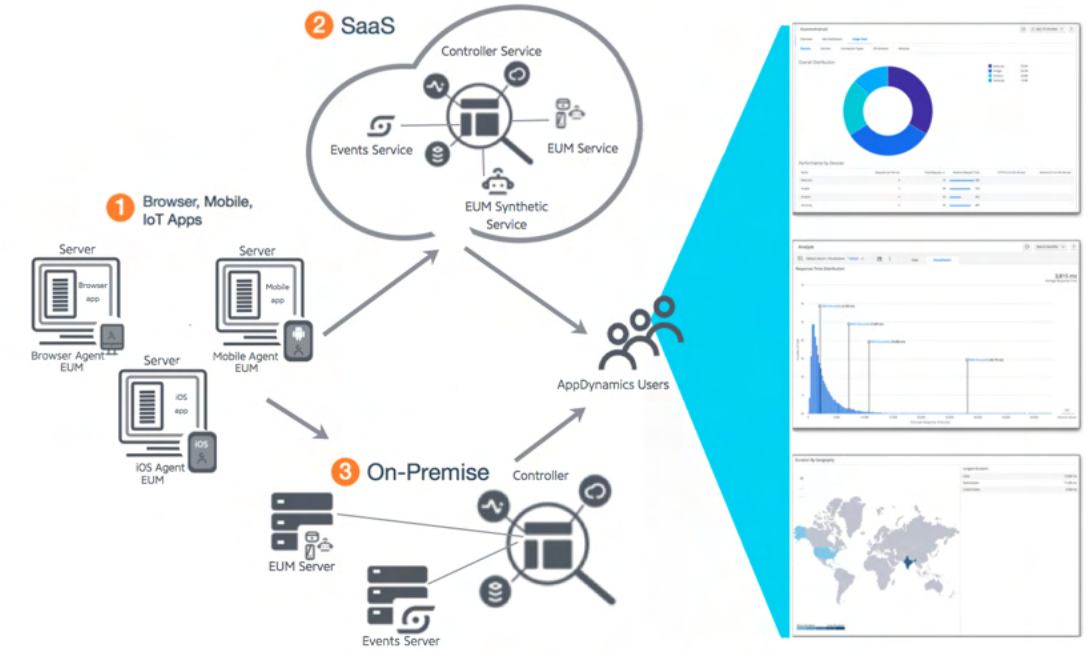
Overview of End User Monitoring

AppDynamics End User Monitoring (EUM) gives you visibility on the performance of your application from the viewpoint of the end user.

While Application Performance Monitoring (APM) measures user interaction starting at the web server or application server entry point, EUM extends that visibility all the way to the web browser, mobile, or IoT application. As a result, EUM reveals the impact the network and browser rendering time have on the user experience of your application.

The diagram below provides an overview of the different components, deployment models (SaaS/on-premises), and the Controller UI as seen by AppDynamics end users. The SaaS deployment uses services (Controller Service, Events Service, EUM Service, EUM Synthetic Monitoring Service) to collect, store, and process data. The on-premises deployment requires DevOps to install discrete components such as the Controller and servers (Events Server, EUM Server) to collect, store, and process data.

How EUM Works



Step	Description	Component(s)
1	The Browser, Mobile, and IoT Agents run in end user applications, collect metrics, and then transmit that data to either a SaaS or an on-premises deployment of AppDynamics.	• Browser/Mobile/IoT Apps • EUM Agents
2	The SaaS Cloud stores, processes, and analyzes data, and then delivers EUM metrics to the Controller UI.	• **Controller Service**: stores data and metadata, makes calls to the EUM Server for raw data and to the Events Service for analytics data • **Events Service**: stores short-term EUM data (sessions, network requests, snapshots, etc.) for heavier analysis • **EUM Service**: verifies, aggregates, and packages raw app metrics • **EUM Synthetic Monitoring Service**: schedules and executes Browser Synthetic jobs and returns session data to the Controller

3	The on-premises deployment has most of the same components and data as the SaaS model. DevOps installs and administers their own Controller, Events Service, and EUM Server. The EUM Synthetic Monitoring Service and sessions data, however, are not available in the on-premises deployment.	• Controller • Events Server • EUM Server See the descriptions for each component above in step 2. The EUM Server performs the same functions as the EUM Service.
4	Users can go to the Controller UI to view and analyze EUM metrics as snapshots, pages, Ajax requests, sessions, network requests, or in the form of charts and graphs.	• Controller UI

Understand End User Activity

Using EUM, you can determine:

- Where geographically your heaviest application load is originated
- Where geographically your slowest End User response times occur
- How performance varies by
  - location
  - client type, device, browser and browser version, and network connection for web requests
  - application and application version, operating system version, device, and carrier for mobile requests
- What your slowest web/Ajax requests are, and where the problem may lie
- What your slowest mobile and IoT network requests are, and where the problem may lie
- How application server performance impacts the performance of your web and mobile traffic
- Whether your mobile or IoT applications are experiencing errors or crashes and the root cause of the issues. For example, for mobile applications, EUM provides stack traces and event trails for the crash or error, helping you troubleshoot and optimize mobile applications.

View EUM Data

The performance information generated by EUM is distinct from the application monitoring data generated by app server agents.

EUM data appears in various locations in the Controller UI, including in the **User Experience** dashboard, **Metric Browser**, and AppDynamics **Analytics** pa ges.

When linked to application business transactions, EUM data gives you a complete view of your end users' experience from the client request, through the application environment, and back to the client as the user response.

You can view EUM performance data in the Controller UI in the **User Experience** tab. From there, you can access information specific to browser applications, mobile applications, or connected devices (IoT applications).

On-Premises EUM Deployments

By default, EUM is configured to use an AppDynamics-hosted component called the EUM Cloud. For a fully on-premises installation, the EUM Server provides the functionality of the EUM Cloud. For information, see EUM Server Deployment.

Some functionality for EUM depends on the AppDynamics Platform Events Service. In a SaaS environment, this is managed by AppDynamics, but it is also possible to use this functionality in an on-premises form.

To host the Events Service on premises, see:

- Custom Install
- Events Service Deployment

If you are adding EUM to an existing on-premises Controller installation, you should evaluate your current configuration's ability to handle the additional load imposed by EUM. For more information, see Additional Sizing Considerations.

Access the SaaS EUM Server

The SaaS EUM Server consists of the components listed below. Each component may have different endpoints depending on the region of your Controller.

- EUM Services - The Mobile Agents, JavaScript Agent, and IoT SDKs send data to the EUM Services. The Controller fetches data from the EUM Server.
- Events Service - The EUM Server sends analytics data to the Events Service. The Controller also queries the Events Service.
- Synthetic Services - The Synthetic Private Agent and Synthetic Hosted Agent send data to the Synthetic Services.

If your SaaS or on-prem deployment requires access to any of these components on the Internet, make sure the URLs given in SaaS Domains and IP Ranges are accessible from your network.

> For on-prem deployments, however, the EUM Server can either be located on the Internet or hosted inside your own data center/network. On-premises access points are configured at installation or through the UI. See EUM Server Deployment and Events Service Deployment for more information.

How EUM Works with other AppDynamics Products

This section describes how other App iQ Platform products work with EUM to provide complete, full visibility on application health and user experience.

EUM and Application Performance Monitoring

Using APM with EUM provides you with greater insight into how the performance of your business application affects the end-user experience. To integrate APM with EUM, you correlate business transactions with browser snapshots. This enables you to trace bad user experiences to issues with your backends such as an unresponsive web service, bad database query or slow server response. To learn how to integrate APM with EUM, see Correlate Business Transactions for EUM.

You can also use the server app agents running on business applications that serve your browser applications to inject JavaScript agent into the code that runs on the browser. This obviates the need to manually inject the JavaScript agent. For more information, see Automatic Injection and Assisted Injection.

> You must assign unique names to EUM applications and business applications. For example, if you created the business application "E-Commerce", you cannot create a browser, mobile, or IoT application with that same name or vice versa.

EUM and Application Analytics

AppDynamics Application Analytics enables you to use the powerful AppDynamics Query Language (ADQL) to analyze different types of EUM data through complex queries. The Analytics components are based on the Events Service, which is also the source of data for Browser Analyze, Crash Analyze , Network Requests Analyze, and all IoT data. Analytics requires a license separate from the EUM licenses except for IoT Monitoring.

Learn More

For more information on the type of user monitoring you interested in, see:

- Browser Real User Monitoring
- Browser Synthetic Monitoring
- Mobile Real User Monitoring
- IoT Monitoring

Experience Journey Map

Experience Journey Map provides real-time insights into business and application performance, visualizing key user journeys and the correlation between performance and traffic. This perspective unifies all application stakeholders: application owners, developers, and IT operations.

Experience Journey Map visualizes:

- Performance metrics for each step in a user journey
- Top incoming and outgoing traffic data for each step
- Drop-off rates

Learn about use cases for Experience Journey Map.

Requirements

To use Experience Journey Map, the following requirements must be met:

- SaaS: Controller >=20.6.0
- On-premise: Controller >=20.7.0
- EUM Peak license (RUM Peak, Browser RUM Peak, or Mobile RUM Peak)
- Instrumented browser or mobile application

Access Experience Journey Map

To access Experience Journey Map:

1. Under the **User Experience** tab, go to a browser or mobile app.
2. In the left application panel, click **Experience Journey Map**.

Experience Journey Map UI

The sections below provide an overview of the Experience Journey Map UI. In the Controller, click on the **Legend** for key terms.

Experience Journey Map Dashboard

The Experience Journey Map dashboard displays the top user journeys, or the most trafficked parts of an app. The default time frame is set to one hour, but you can adjust the time and the dashboard automatically updates the user journeys and data for that time frame.

End User Events

Each step in a user journey is visualized with an end user event. An end user event is a browser page or mobile view/activity. Experience Journey Map displays the most trafficked end user events.

Click an end user event to see:

- Total user visits from all sources
- Incoming and outgoing traffic sources
- Performance breakdowns for each traffic source
- Drop-off rate

Traffic Segments

A traffic segment connects two end user events and contains data about users who journeyed from one end user event to the next. Traffic segments display a health status icon for errors and exceeded performance thresholds. To edit performance thresholds, see Configure Experience Journey Map.

Click a traffic segment to see:

- Number of users who came from the previously mapped end user event
- Performance metrics for those users
- Option to analyze individual browser or mobile sessions for that end user event

Refresh Loops

A refresh loop is a type of traffic segment and contains data for users who refresh an end user event.

Click a refresh loop to see:

- Number of users who refreshed the end user event
- Performance metrics for those users
- Option to analyze browser or mobile sessions for that end user event

Experience Journey Map for Browser vs. Mobile Apps

The table below describes the differences in **Experience Journey Map** data for browser vs. mobile apps.

Component	Browser App	Mobile App
End User Event	An end user event shows data for one browser page. To customize end user event names, see Configure Page Identification and Naming.	An end user event shows data for one iOS view or Android activity. To customize how the mobile agent reports the view/activity names via the `SessionFrame` API, see the following: • Customize the iOS Instrumentation • Customize the Android Instrumentation • Customize the Xamarin Instrumentation • Customize the Cordova Instrumentation Hybrid agents (Cordova, React Native, and Xamarin) do not support individual screen tracking, which affects how Experience Journey Map displays events in a user journey. To ensure each screen is displayed in a user journey, we recommend using the `SessionFrame` API to define when a screen starts and ends. See documentation for each hybrid agent: • Customize the Xamarin Instrumentation • Customize the Cordova Instrumentation • Customize the React Native Instrumentation
Default performance threshold metric	Performance thresholds (Slow, Very Slow, Stall, and Normal) are set to End User Response Time (EURT), or page load time.	Performance thresholds (Slow, Very Slow, Stall, and Normal) are set to the average time of mobile network requests.
Types of errors captured	JavaScript and AJAX errors.	Crashes and Application Not Responding (ANRs).
Analyze functionality	When you click **Analyze**, you are redirected to Browser RUM Analyze with filters applied for that mapped browser page.	When you click **Analyze**, you are redirected to Mobile Sessions with filters applied for that mapped mobile view/activity.
Misc.	Iframes are captured and included as a node with performance metrics, such as for SPA2 base pages.	You may notice that sometimes the metric breakdowns of performance thresholds do not add up to 100%. See Analyze Traffic for an example. This is because for mobile apps, performance thresholds are set to an average of only the network request(s) triggered from a view/activity. If the number of triggered network requests is less than the number of total network requests, the performance threshold breakdown does not add up to 100%.

Using Experience Journey Map

- Make a Custom Map
- Filter Experience Journey Map
- Analyze Traffic

Learn More

- Use Cases for Experience Journey Map
- Configure Experience Journey Map
- Browser RUM Analyze
- Mobile Sessions

How to Use Experience Journey Map

This page covers how to use Experience Journey Map to view select user journeys.

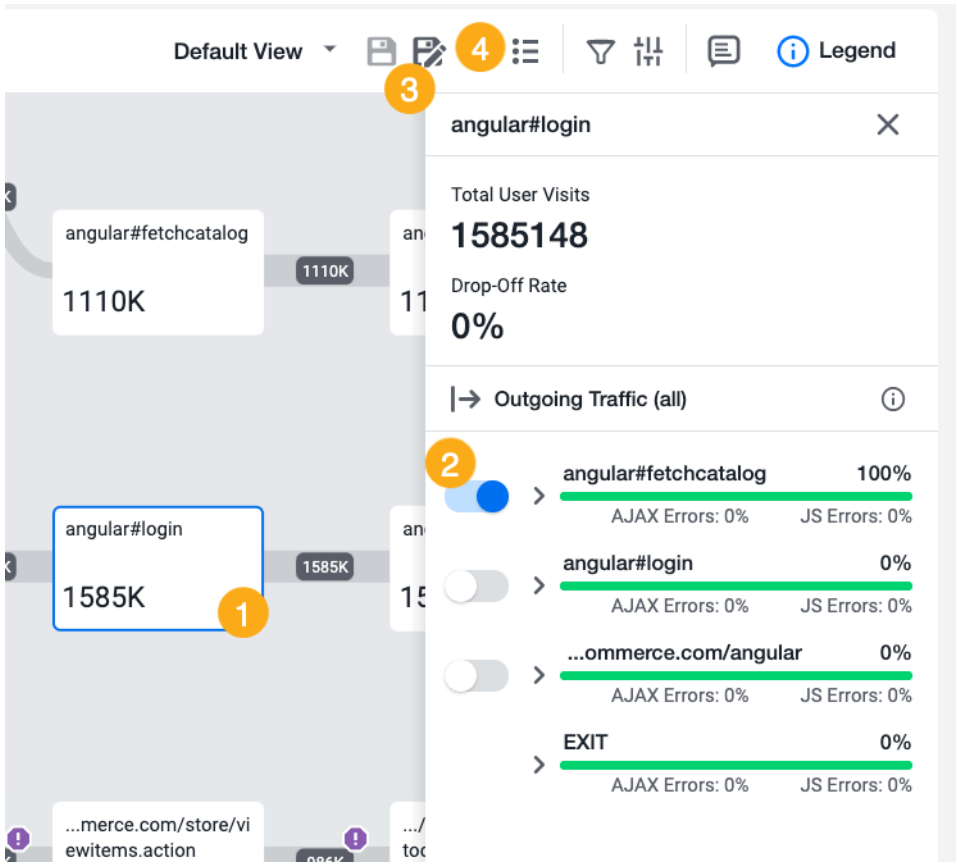
Create a Custom View

The default Experience Journey Map view shows top user journeys for the entire application. If you want to see the top journeys for specific flows and events, you can create a custom view. For example, if you designed for different flows, such as a travel booking app for booking a hotel, a flight, and a rental car, you can create three custom views for each flow.

> Custom views are available with SaaS Controller >= 21.4.0.

To create a custom view:

1. Click on any event of a user flow.
2. Toggle the next event you want to see.
3. Click and name the custom view.
4. Click to view, edit, and delete your custom views.



Filter Experience Journey Map

You can filter user journeys by the fields listed below to see user journeys through a helpful context, such as troubleshooting a crash by mobile device version. When you apply a filter, such as a mobile device version iOS 10, Experience Journey Map shows the top journeys for iOS 10 users.

To use the filters:

1. In the upper right-hand corner, click on the **Filter Panel** .
2. Select a filter(s).
3. Click **Apply**.

Available Filters

The table below lists available filters for Experience Journey Map.

Application Type	Fields

Browser	• Browser • Country • Device • Device OS • Internet Service Provider (ISP) • Custom User Data
Mobile	• Application Version • Carrier • Connection Type • Country • Device Model • Device Version • OS Version • Custom User Data

Using the Custom User Data Filter

The custom user data filter allows you to filter user journeys by user data strings you previously configured (strings with a maximum of 5000 values).

> Custom user data you configure is automatically included in the Experience Journey Map data collection.

To filter by custom user data:

1. Click on the **Filter Panel** .
2. In the panel, click on > **Select Visible Filters**.
3. In the **Filter Panel** , check the "Custom User Data" box.
4. Start typing a data string. The screen will auto-populate with data available.
5. In the pop-up, check data you want to filter.
6. Click **Apply**.

> To configure custom user data, see:
>
> - Add Custom User Data to a Page Browser Snapshot
> - Customize the iOS Instrumentation
> - Customize the Android Instrumentation

Analyze Traffic

A traffic segment connects two end user events and contains data about users who journeyed from one end user event to the next. The **Analyze** functionali ty allows you to drill down into individual browser or mobile sessions for that traffic segment. When you click **Analyze**, you are redirected to Browser RUM Analyze or Mobile Sessions with applied filters for the users who navigated from the previous end user event to the next.

Analyze User Journeys with Applied Filters

If you apply a filter and click **Analyze** on a traffic segment or end user event, the applied filters will carry over to Browser RUM Analyze or Mobile Sessions.

Use Cases for Experience Journey Map

Related pages:

- Experience Journey Map
- Configure Experience Journey Map
- Browser RUM Analyze
- Mobile Sessions

This page describes top use cases for Experience Journey Map and is categorized by application stakeholder: IT operations, developers, and application owners.

IT Operations

Goal	Use Case
Prioritize performance-related incidents	Investigate why users are experiencing Ajax errors on a browser application: 1. In Experience Journey Map, you see that a correlation of Ajax errors, a high drop-off rate, and the majority of Ajax errors occurs on Internet Explorer browsers. 2. Filter user journeys by browser and notice that the trend maintains. 3. Click on **Analyze** to see Browser Sessions with the criteria you filtered in Experience Journey Map.
Create real-world, data-backed testing	Create a data-backed Synthetic script for browser application testing: 1. In Experience Journey Map, look at top user journeys within the default time frame. 2. Adjust the time frame to one day, one week, and one month. 3. Identify the top user journeys in each time frame and note if there are consistencies throughout. 4. Write a synthetic script to test those top user journeys.
	Investigate if crashes correlate with a specific mobile device version: 1. In Experience Journey Map, click on a few iOS views containing crashes. You see that most users experiencing those crashes are using iPhone 8. 2. Filter user journeys by mobile device version to diagnose if one version is causing those crashes. 3. Fix the issue related to the device version. 4. Filter Experience Journey Map by that device version again and validate the fix.

Developers

Goal	Use Case
Prioritize and resolve errors	Identify the source of iOS application crashes in the last 24 hours: 1. In Experience Journey Map, click the traffic segment to identify the top user device and OS. 2. Use this data to reproduce the crashes. 3. Click **Analyze** on the traffic segment to view each mobile session crash and identify the mobile crash stack trace. 4. Once you fix the issue, use Experience Journey Map to validate the fix.

Application Owners

Goal	Use Case
Manage and prioritize resource investments	Evaluate user interactions with a new banking application feature: 1. In Experience Journey Map, filter user journeys to see traffic metrics for the new feature. 2. Depending on whether the new feature is gaining more or less traffic than expected, collaborate with IT operations and developers to increase or decrease resources around the feature.

Configure Experience Journey Map

Related pages:

- Configure Mobile Network Request Naming

This page describes how to configure Experience Journey Map. Though Experience Journey Map does not require configuration, you can adjust performance thresholds and change end user event names to better suit your business needs.

Configure User Experience Thresholds

You can configure the timing and standard deviations for performance thresholds such as Slow, Very Slow, and Stall. When you configure these deviations, the changes apply to all Controller views of that application, not just to Experience Journey Map. To configure from Experience Journey Map, go to **Info** > **Configure thresholds**.

Configure Performance Thresholds and Drop-off Rates

To configure performance thresholds, you must have Controller >=20.8.0 and a "Configure EUM" user permission. To configure user and role permissions in the Controller UI, go to **Settings** > **Administration**.

Toggle On or Off

You can toggle performance thresholds and drop-off rates on or off. When a threshold or drop-off rate is toggled off, the marker does not appear on Experience Journey Map. When you toggle a threshold on, the threshold resets to the previous value (and not necessarily the default value).

Configure Values

For performance thresholds and drop-off rates, the percentage is set to the minimum traffic for a marker to appear on a user journey event. For example, if Ajax Errors is set to 30%, Experience Journey Map displays an error icon when 30% and more of user traffic experiences Ajax Errors. You can configure the values for performance thresholds and drop-off rates. The table below lists default performance thresholds values for existing and new applications.

Change End User Event Names

In Experience Journey Map, a user journey is a collection of end user events. End user events represent browser pages or mobile views/activities.

Browser Applications

For browser apps, you can rename and group end user events that are meaningful to your business. For example, if you have a collection of ecommerce URLs of user steps to place an order, you can rename those URLs with pages names like "Order info," "Order review," and "Order confirmation." To rename end user events, see Configure Page Identification and Naming.

Mobile Applications

For mobile apps, you cannot change mobile view/activity names in the Controller, but you can customize how the mobile agent reports the event names via the `SessionFrame` API. See these pages for instructions:

- Customize the iOS Instrumentation
- Customize the Android Instrumentation

About Hybrid Applications

Hybrid agents (Cordova, React Native, and Xamarin) do not support individual screen tracking, which affects how Experience Journey Map displays events in a user journey. To ensure each screen is displayed in a user journey, we recommend using the `SessionFrame` API to define when a screen starts and ends. See documentation for each hybrid agent:

- Customize the Xamarin Instrumentation
- Customize the Cordova Instrumentation
- Customize the React Native Instrumentation

Correlate Business Transactions for EUM

This page describes how to correlate business transactions for End User Monitoring (EUM). You can correlate the network request events of EUM applications with the business transactions of Business Applications. The EUM application can be a browser, mobile, or IoT application. The correlation is made between EUM application beacons containing network request event information and instances of business transactions (transaction snapshots).

For detailed information, including how and what is correlated for each type of EUM application, see these pages:

- Correlate Business Transactions for Browser RUM
- Correlate Business Transactions for Mobile RUM
- Correlate Business Transactions for IoT Monitoring

Benefits of Correlating Business Transactions

By correlating business transactions with network request snapshots, you can identify potential issues with the backend application that are causing bad user experiences. For example, you might find that a server error or a database query is causing a slow user experience.

Before You Begin

To correlate business transactions:

1. Purchase licenses for Application Performance Monitoring (APM) and End User Monitoring (EUM).
2. Instrument a business application with a supported app server agent.
3. Instrument an EUM application (browser, mobile, or IoT application).

> If automatic business transaction correlation is not supported for your app server agent, you can manually enable business transaction correlation in the Controller UI.

Business Transaction Correlation Support for App Server Agents

This table shows the app server agents that support business transaction correlation.

- Automatic correlation support means that business transactions are automatically correlated.
- For manual correlation, you must manually enable business transaction correlation.

App Server Agent	Type of Correlation Support
Java Agent	Automatic (Java Agent >= 20.3.0, and Controller >= 20.4.0)
.NET Agent	.NET Framework: Automatic (.NET Agent >= 20.9.0, and Controller >= 20.10.0)
	.NET Core: Automatic (.NET Agent >= 21.1.0, and Controller >= 20.10.0), Windows only
	Linux: Automatic (.NET Agent >= 21.5)
Node.js Agent	Manual
PHP Agent	Manual
Python Agent	Manual

Access Business Transaction Correlation

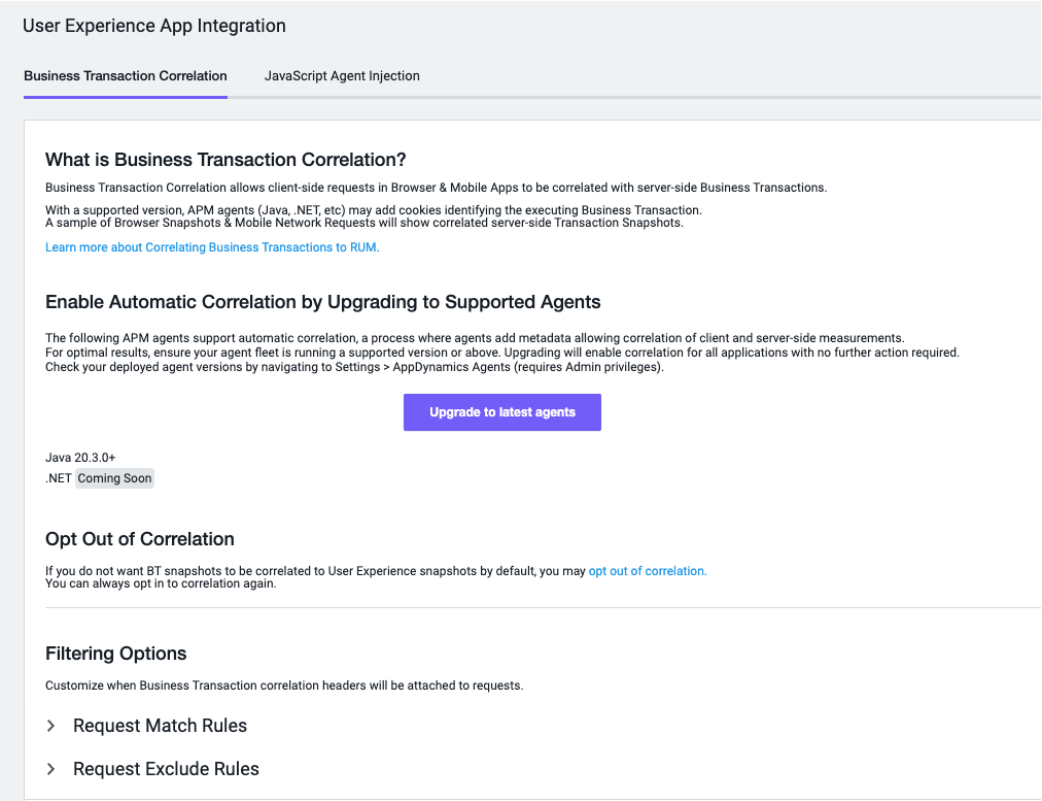
To configure business transaction correlation:

1. In the **Application Dashboard**, click **Configuration** > **User Experience App Integration**.
2. Under the **Business Transaction Correlation** tab, there are two options depending on the correlation support for your App Server Agents:
    - For agents with support for automatic correlation, see Automatic Correlation of Business Transactions.
    - For agents without support for automatic correlation, see Manually Enable Business Transaction Correlation.

Automatic Correlation of Business Transactions

By default, the correlation of business transactions of business applications instrumented with Java Agent >= 20.3.0 is enabled.

Under the **Business Transaction Correlation** tab, you can choose to opt out or filter requests with match or exclude rules.



Opt In to Automatic Correlation

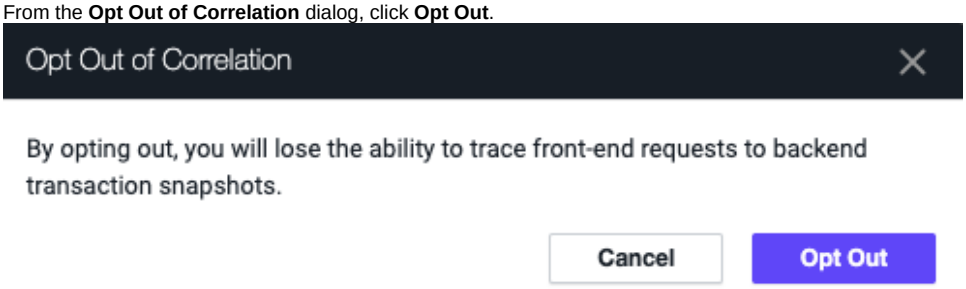
To opt in to have business transactions automatically correlated, you must use Java Agent >= 20.3.0. If you are using an older version of the Java Agent, you must upgrade the agent.

For unsupported agents, you need to manually enable business transaction correlation.

Opt Out of Automatic Correlation

If business transaction correlation is enabled by default, you can opt out by:

1. In the **Application Dashboard**, click **Configuration** > **User Experience App Integration**.
2. Under the **Business Transaction Correlation** > **Opt Out of Correlation**, click **opt out of correlation**.
3. From the **Opt Out of Correlation** dialog, click **Opt Out**.

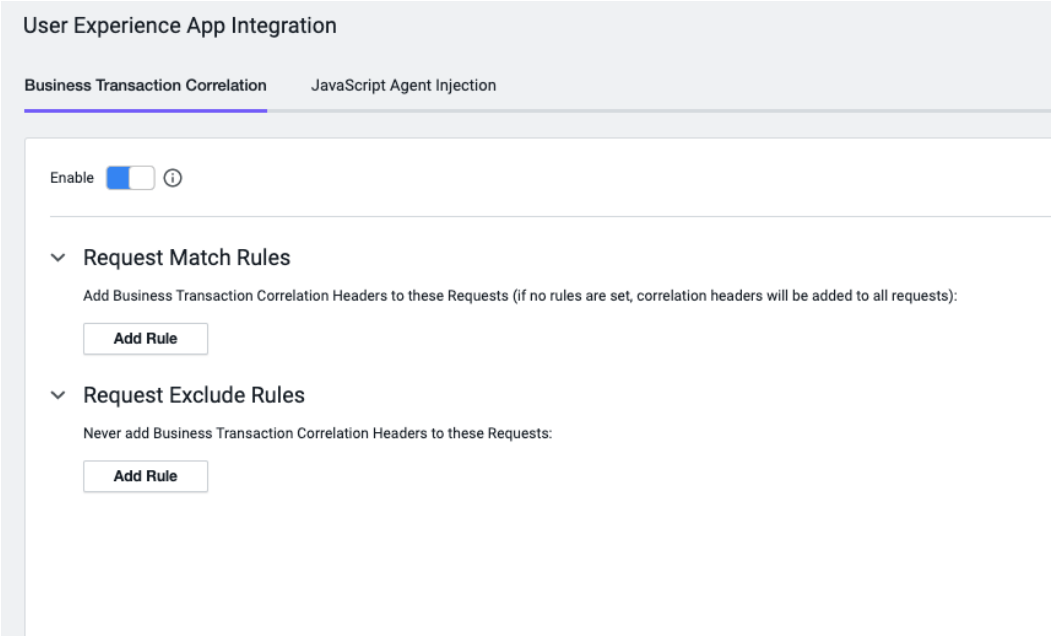


Manually Enable Business Transaction Correlation

For app agents that do not have automatic correlation support, you need to manually enable business transaction correlation.

To manually enable business transaction correlation:

1. In the **Application Dashboard**, click **Configuration** > **User Experience App Integration**.
2. Under the **Business Transaction Correlation** tab, check the **Enable Business Transaction Correlation** checkbox.



Disable Business Transaction Correlation

To disable business transaction correlation:

1. In the **Application Dashboard**, click **Configuration** > **User Experience App Integration**.
2. Under the **Business Transaction Correlation** tab, toggle the **Enable** button.

Filter Business Transactions for Correlation

You can also specify which business transactions includes or excludes correlation headers. If you do not add request match rules or request exclude rules, all requests have correlation headers by default.

To add a request match or exclude rule:

1. From **Filtering Options**, expand **Request Match Rules** or **Request Exclude Rules**.
2. Click **Add Rule**.
3. Check the **Method** to use the default `HTTP` method `GET` or select a different `HTTP` method.
4. Check the **URI** checkbox and select one of the matching methods from the dropdown:
    - **Starts With**
    - **Ends With**
    - **Contains**
    - **Equals**
    - **Matches RegEx**
    - **Is in List**
5. (Optional) Click  and check the **NOT** checkbox. This condition reverses the results by excluding the requests that match the specified parameters.
6. Click **Add Rule**.

EUM Data

Related pages:

- Using the Controller APIs

With the iOS 14 release (supported by iOS Agent 20.10.0), Apple introduced a new data collection policy. See iOS Data Collection Disclosure for details.

This page describes the types of EUM data, how data is collected, where data is store, and where data is displayed in the Controller. For data retention and license consumption details, see License Entitlements and Restrictions.

Types of EUM Data

Metric Data

Metrics are data that reflect your application's performance. Browser RUM captures metrics from browser applications, such as timing and Ajax metrics. Mobile RUM captures metrics from mobile applications, such as crash metrics and network request metrics. You can view and analyze EUM metrics in the Metric Browser.

To learn more about EUM metrics, see:

- Browser RUM Metrics
- Browser Synthetic Metrics
- Mobile RUM Metrics
- Extensions and Custom Metrics

Custom Data

Browser RUM

You can add specific user information a browser snapshot of your application. The information is expressed as key-value pairs, appears in the **User Data** s ection of the snapshot, and is available for page snapshots, Ajax requests, and virtual pages. To learn how to set custom user data, see Add Custom User Data to a Page Browser Snapshot.

Mobile RUM

Mobile RUM custom data such as Info Points, Custom Timers, and Custom Metrics are considered metrics and stored as mobile request event data. User data is another type of Mobile RUM custom data. Custom user data can be reported through network request events because user data is attached to each network request.

To learn how to set user data, see User Data (iOS SDK) or User Data (Android SDK).

EUM Analytics Data

You can view EUM data in Analytics if you have a Real User Monitoring Peak license. The EUM Analytics data consists of event data and is stored in the Events Service and visualized in widgets. EUM Analytics provides data for these event types:

- Browser Requests Event Data
- Browser Sessions Event Data
- Mobile Requests Event Data
- Mobile Sessions Event Data
- Mobile Crash Report Event Data
- Synthetic Sessions
- Connected Device Data

Cookies for Browser Applications

Browser RUM uses two different kinds of short-lived cookies for the JavaScript Agent to collect data and correlate events. Both type of cookie do not contain any personally identifiable information (PII) and are immediately deleted after being read.

- The `ADRUM` cookie: Written by the JavaScript Agent, this cookie contains the referral page URL and some timing information to assist gathering First Byte Time for some browser types. When the agent loads on the subsequent page, it reads the information and then deletes the cookie. If there is no agent on that page, the cookie is deleted when the browser is closed. For privacy purposes, the URL of the referral page is hashed.
- The `ADRUM_BT` cookies: Written by the server-side agent when the page is served from an instrumented server. These cookies help correlate browser data with related server-side performance data.
    - `ADRUM_BTa`: Contains the backend transaction ID as well as timing info and is used to correlates end-user experience with the health of the backend app.
    - `ADRUM_BTg`: Contains the backend transaction ID and is used as an alternative method to correlate end-user experience with the health of the backend app.

- `ADRUM_BT[1-5]`: Contains the business transaction numbers as well as timing and error info for the first five business transactions, such as `ADRUM_BT1`, `ADRUM_BT2`, etc.
- `ADRUM_BTs`: Contains a link from a browser snapshot to a server snapshot.
- `ADRUM_BTh`: Written only if there was a server-side error.

If Browser RUM detects that the page is HTTPS, the `Security` attribute is set for cookies. The `Security` attribute is a flag that forbids a cookie from being transmitted via an unencrypted HTTP connection.

Web Storage

Browser RUM stores key-value pairs in web storage to associate page views with a particular session and browser. The value for each key is a randomly generated ID.

The following lists the keys and the expiration time for each key-value pair:

- `ADRUM_AGENT_INFO` (never)
- `ADRUM_CLIENTINFO` (never)
- `ADRUM_XD_AGENT_ID` (never)
- `ADRUM_XD_AGENT_INFO` (1 week)

Mobile Local Storage

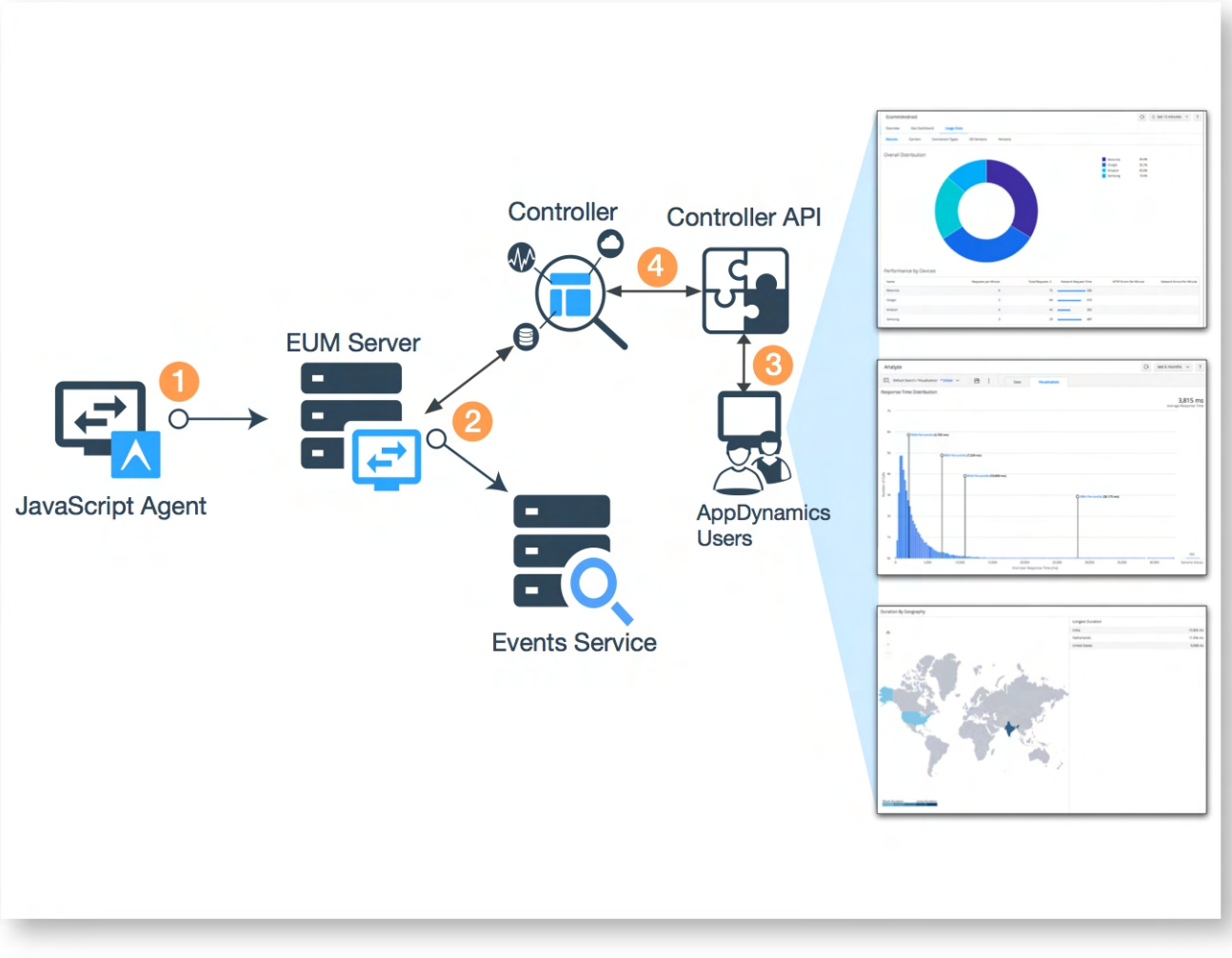
Mobile agents use beacons to transmit metrics, app metadata, network requests, crashes, and custom data. When a beacon cannot be transmitted, the data is persisted in permanent storage within the container of the application and subject to the security configuration of the device and application. No encryption is currently being utilized. Once the network connection is restored, the beacons resume transmitting data. Because some of the data is provided by the developer's instrumentation of the app, such as breadcrumbs, user info, and the app (URLs, crash reports), some information is not explicitly collected by AppDynamics.

Mobile agents also locally stores a randomly generated ID for tracking sessions and license usage. The ID is stored in the Events Service.

How Data Is Stored and Retrieved

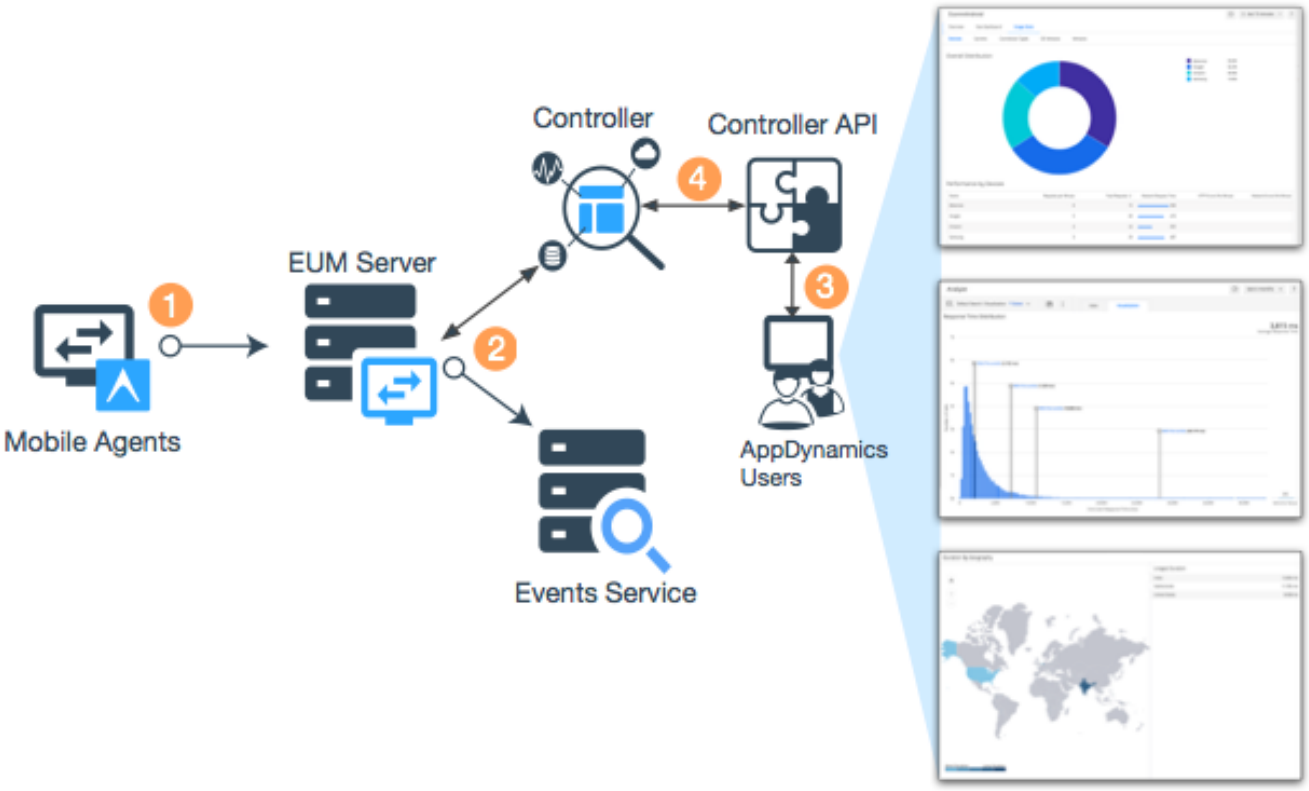
Browser RUM

Browser RUM data is stored in the Controller, the Events Service, and the EUM Server. 1 The JavaScript Agent sends raw data to the EUM Server, where the data is verified, aggregated, and packaged every minute. 2 The EUM Server then sends data to the Events Service. 3 The Controller UI makes requests to the 4 Controller API, which fetches data from one of the three data stores (EUM Server, Events Service, Controller). Because the Controller UI only interacts with the Controller API, it does not need to know where the data is stored.



Mobile RUM

Mobile RUM data is stored in the Controller, the Events Service, and the EUM Server. 1 The Mobile agents send raw data to the EUM Server, where the data is verified, aggregated, and packaged every minute. 2 The EUM Server then sends the data to the Events Service. 3 The Controller UI makes requests to the 4 Controller API, which fetches data from one of the three data stores (EUM Server, Events Service, Controller). Because the Controller UI only interacts with the Controller API, the Controller UI does not need to know where the data is stored.



Data Storage Details

Browser RUM

This table shows where different Browser RUM data is stored. To see how long the data is retained, see License Entitlements and Restrictions. Resource details are only available for those browser snapshots with resource timing.

	Controller	Events Service	EUM Server (SaaS/On-Premises)
Browser Metrics			
Browser Snapshots			
Resource Details			(filesystem)
EUM Page Configuration			(filesystem)
Page View Events			
Ajax Events			
Session Events			
Metadata			(MySQL)
Licenses (SaaS)			(MySQL)
Licenses (On-Premises)			(MySQL)

Browser RUM Default Data Limits per App Key

- Metrics: 100k
- Pages: 500
- Ajax Requests: 500

- Max Event Size: 1 MB

Mobile RUM

This table shows where different Mobile RUM data is stored. To see how long the data is retained, see License Entitlements and Restrictions.

	Controller	Events Service	EUM Server (SaaS/On-Premises)
Mobile RUM Metrics			
Network Request Snapshots			
Custom Data			
Crash Analyze			(filesystem)
Events			
Session Events			
Metadata			(MySQL)
ProGuard/dSYM Files			(filesystem)
Screenshot Files			(S3 for SaaS, filesystem for on-prem)
Licenses (SaaS)			(MySQL)
Licenses (On-Premises)			(MySQL)

Mobile RUM Default Data Limits per Mobile App and App Key

Unit	Metric Limit	Network Requests	Max Event Size
Mobile App	N/A	500	1 MB
App Key	100,000	2000	

Controller Mapping of Data

Browser RUM

This table shows the relationship between the Controller UI components and their data sources.

Controller Component	Storage Mechanism
Overview	Controller
Geo Dashboard	Controller
Browser Snapshots	Controller (no resource details) / EUM Server (SaaS/On-Premises) (resource details)
Usage Stats	Controller
Sessions	EUM Server (SaaS/On-Premises), Events Service
Pages & Ajax Requests	Controller / Events Service (limited)
Analyze	Events Service

Mobile RUM

This table shows the relationship between the Controller UI components and their data sources.

Controller Component	Storage Mechanism
Overview	Controller
Geo Dashboard	Controller
Usage Stats	Controller

Sessions	Events Service
Network Requests	Controller, Events Service (limited)
Network Requests Snapshots	Controller
Crashes	Events Service, EUM Server (SaaS/On-Premises)
Crashes Analyze	Events Service
Custom Data	Controller
Events	Controller

How to Access EUM Data

In addition to accessing EUM data through the Controller UI, you can also access event data through Analytics and the AppDynamics APIs. The AppDynamics APIs include the Analytics Events API and the Metric and Snapshot API that you can use to access EUM metric data.

If you have enabled Analytics and Browser RUM, you can use Browser Analytics to view data for these event types:

- Browser Requests
- Browser Sessions

If you have enabled Analytics and Browser Synthetic Monitoring, you can use Synthetic Analytics to view data for this event type:

- Synthetic Sessions

If you have enabled Analytics and Mobile RUM, you can use Mobile Analytics to view data for these event types:

- Mobile Requests
- Mobile Sessions
- Mobile Crash Reports

If you have enabled IoT Monitoring, you can use Connected Devices Analytics to view data for the event types below. You *do not* need to enable Analytics to use IoT Analytics.

- Connected Device Data

Data and Deployment Models

AppDynamics offers SaaS and on premises deployment models. You can access most Browser and Mobile RUM data from either deployment model.

EUM Accounts, Licenses, and App Keys

This page covers account and licensing information for End User Monitoring (EUM).

View License Usage

To view your EUM license consumption in the Controller, go to  > **License**.

Overages

If your license does not allow overages, AppDynamics stops reporting metrics after your license limit has been reached. If your license does allow overages and your usage exceeds the limit, AppDynamics continues reporting EUM metrics and bills you for the overages at the unit rate stipulated by your license agreement.

License Key

If you have at least one EUM license type, you have an EUM Account, which has a name and a license key associated with it. This is the unique identifier that AppDynamics uses to associate EUM data to your account. You only need to know this information for troubleshooting purposes. The same key applies to all four EUM services. However, each product has its own types and metrics for allowed usage.

EUM App Key

This is the unique identifier that AppDynamics uses to associate end user data to specific EUM applications. Each EUM application will be associated with one EUM App Key. For your applications to be monitored, they will need to include the EUM App Key when reporting data. The EUM App Keys are also associated with an EUM account name and license key.

Changing the license key does not affect existing EUM App Keys.

License Optimization and Usage Periods

For EUM license metric definitions and data retention, see License Entitlements and Restrictions.

EUM License	License Optimization	Usage Period
Real User Monitoring	See Browser and Mobile RUM rows below.	See the Browser and Mobile RUM rows below.
Browser RUM	If the value of Pageviews is greater than **Pageviews Allocated** and your license allows overages, you are incurring overage charges. If you need to stop incurring overage charges, you can disable Browser RUM to stop end user monitoring and stop Pageviews from being charged (once disabled, expect a delay of approximately one minute).	The Browser RUM usage period is per year, even if you have a multi-year license. The meter resets every year based on the expiration of your license agreement. For example, if your license expires at 12:00pm PST on May 1, 2020, your usage period resets every year starting 12:00pm PST on May 1, 2021.

Mobile RUM	License consumption is affected when you have one instrumented application installed on multiple devices, multiple applications on the same device, or uninstalls followed by reinstalls: • If you install one instrumented application on two devices within a month, each device uses one Monthly Active Agent. • If you uninstall and then reinstall an instrumented application multiple times within a month, each reinstall uses one Monthly Active Agent. • If you install two different instrumented applications on the same device within a month, each instrumented application consumes one Monthly Active Agents, so. Through SDKs, you can customize application instrumentation to choose which agents send data. For further instructions, see the following: • Customize the Android Instrumentation • Customize the iOS Instrumentation • Customize the Xamarin Instrumentation • Customize the Cordova Instrumentation	The Mobile RUM usage period begins and the meter resets on the first day of each month at 12:00pm PST. When your license consumption reaches 90% of your Monthly Active Agent allocation, you will see a warning in the Mobile RUM dashboard.
Synthetic	To help you optimize the use of your Synthetic Agent license, keep the following in mind: • Synthetic jobs are measured in terms of the time it takes to actually run the test in AppDynamics systems. Because it is not possible to know in advance how long a given test may take, you can set a timeout value when you create the job. If the job times out, you receive partial results. • Once you have used up your monthly allotment, all synthetic jobs are paused until the following month. • If there is a problem accessing a site, such as receiving a 5xx error, the system automatically runs the job again. You are charged for these retests. • If you see a session with the status INTERNAL_ERROR, this means there is a problem with AppDynamics infrastructure, and you will not be charged, even if the session contains partial results. • Adjusting synthetic job configurations, such as the number of browser types, frequency, and duration, may have an impact on license consumption. To gauge estimated consumption in the Controller, create a new job and see the section **Set a Timeout** (step 6). • Locations are user-defined. You can add multiple servers (agents) to a location to increase capacity, and this does not impact your license.	The usage period begins and the meter resets on the first day of each month at 12:00pm PST.
IoT	License consumption is based on Application Instances. We recommend that you register each Application Instance and each device so that you can correlate data with each data source. To reduce license consumption, you can register just one device with multiple Application Instances, or one Application Instance on multiple devices. However, in doing so, AppDynamics cannot differentiate data, such as errors, for each application instance or device unless each one is registered.	The usage period begins and the meter resets on the first day of each month at 12:00pm PST. The usage period for IoT is always the current month, even if you have a multi-year license.

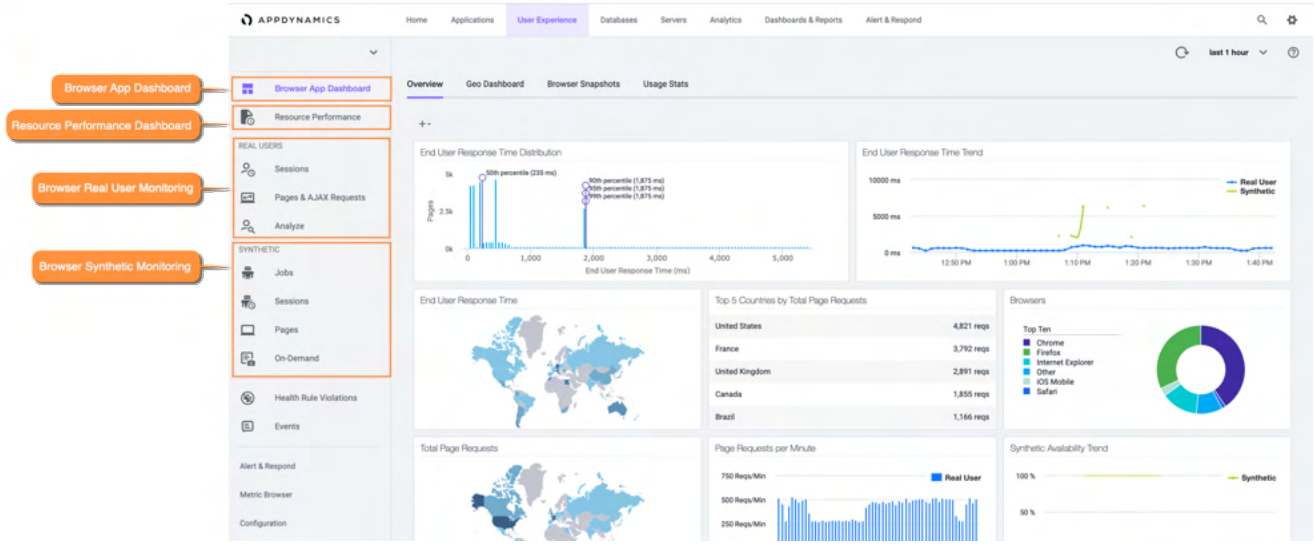
Browser Monitoring

AppDynamics offers two products to monitor browser applications:

- Browser Real User Monitoring (Browser RUM) - Monitors how your web application is performing, using real user data to analyze application performance and user experience.
- Browser Synthetic Monitoring - Analyzes application availability and performance, using scheduled testing to analyze website availability.

Overview of the Controller UI for Browser Monitoring

Browser RUM and Browser Synthetic Monitoring share two dashboards: Browser App Dashboard and Resource Performance Dashboard.



Browser App Dashboard

The Browser App Dashboard provides a high-level understanding of how your application's overall performance. When you first navigate to a browser application, you are defaulted to the **Browser App Dashboard** > **Overview** tab. The **Overview** tab contains widgets for both Browser and Synthetic data.

Resource Performance Dashboard

The Resource Performance Dashboard provides a high-level understanding of how your resources affect the performance of your browser application. You can use this dashboard to pinpoint resource-related performance issues affecting the user experience. You can toggle between viewing Browser and Synthetic data.

Real Users

Browser RUM contains these UI tabs for more detailed real-user data analysis:

- Browser RUM Sessions
- Pages & AJAX Requests
- Browser RUM Analyze

Synthetic

Browser Synthetic Monitoring contains these UI tabs for more detailed synthetic user data analysis:

- Synthetic Jobs
- Synthetic Sessions
- Synthetic Pages
- Synthetic On-Demand

Differences Between Browser RUM and Browser Synthetic Monitoring

Although Browser Synthetic Monitoring and Browser RUM report similar metrics, there are differences in:

- Hardware
- Network connections
- Different browsers
- Browser caching is not present in synthetic sessions

If you see a sudden change in any of those metrics, you should compare the results of Browser Synthetic Monitoring and Browser RUM to determine if there is an existing problem.

Performance Versus Workflows

Browser RUM excels at capturing the full breadth of performance and your real users' experience. Browser Synthetic Monitoring gives you confidence that your key workflows are always working and performing.

Synthetic Metrics and Screenshots

Browser Synthetic Monitoring can collect certain data that Browser RUM cannot. For example, Browser Synthetic Monitoring provides screenshots, which help you see per-page testing. You can also use the Visually Complete Time metric and other related metrics to understand how users experience page load time.

How to Use Browser RUM and Browser Synthetic Monitoring Together

Identify Issues

Although Browser RUM can detect certain kinds of problems (like JavaScript exceptions), it cannot comprehensively test for functional correctness. For example, you may want to verify that your online store has reasonable prices a list of items. If your site is down entirely, then the JavaScript Agent will never be loaded, so errors or verifications will not be reported. Fortunately, Browser Synthetic Monitoring will keep running, discover the error, alert you, and provide detailed information about the problem.

Control Environmental Factors

Hardware, browsers, operating systems, and networks complicate performance analysis in Browser RUM. Browser Synthetic Monitoring uses consistent hardware, software, and network configurations you set when you create a synthetic job, so if you see deviations in performance, you can be fairly certain a problem exists.

Browser App Dashboard

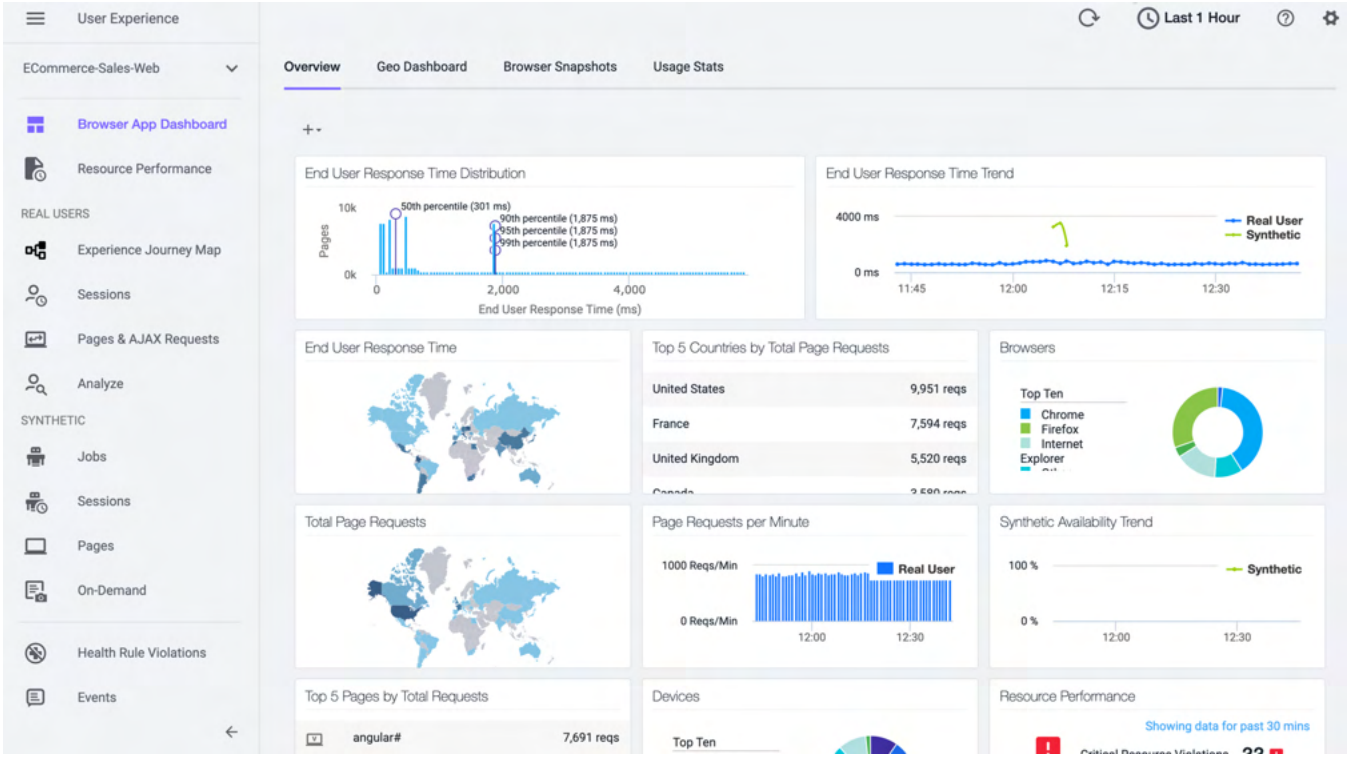
The **Browser App Dashboard** gives you a high-level overview of your browser application performance. To access the **Browser App Dashboard**:

1. In the Controller, go to **User Experience.**
2. Under **Browser Apps**, select an application.
3. In the left navigation bar, click **Browser App Dashboard**.

Overview

The **Overview** tab displays a set of configurable widgets. The default widgets contain multiple graphs and lists featuring common high-level indicators of application performance. You can remove a widget by clicking the X icon and add a widget using the + icon.

> For SPA2 applications, the **Overview** tab only displays data for base pages and virtual page events. Data from Iframe and Ajax requests is not included. If you have an application that only has Iframe and Ajax requests, then the **Overview** tab will not display any data.

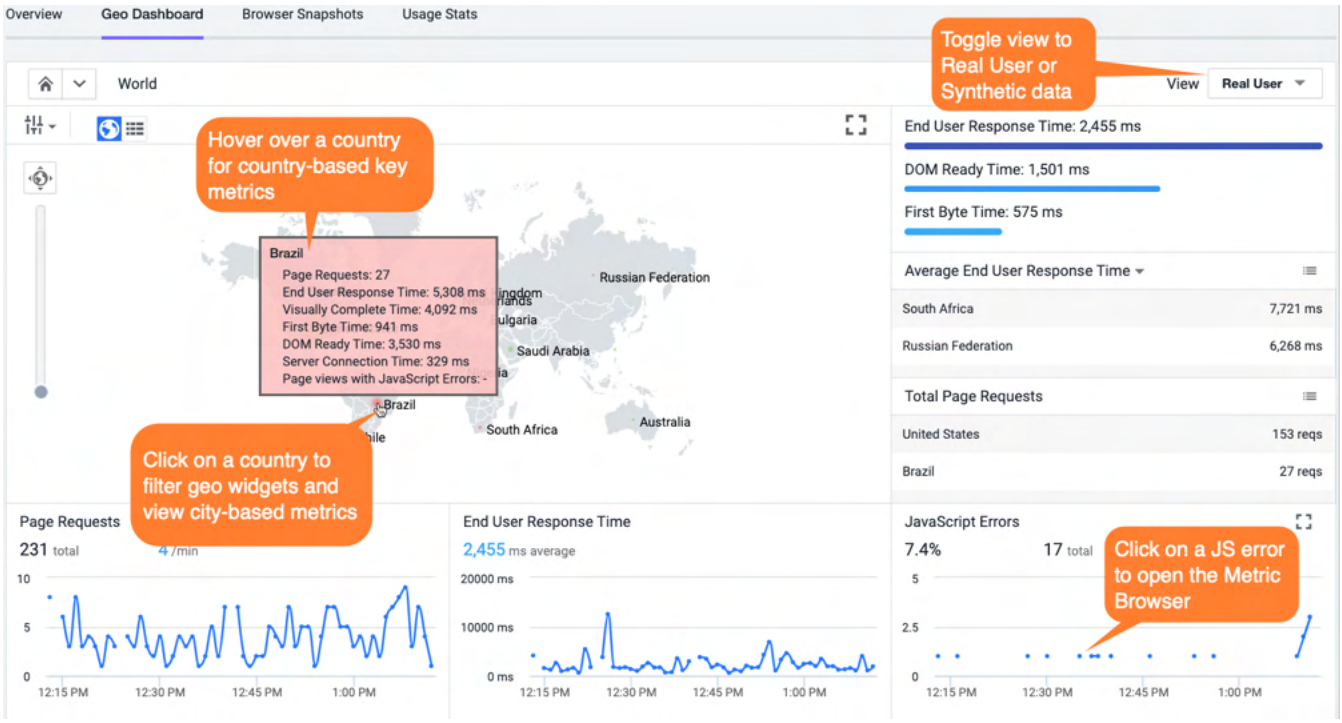


Geo Tab

The **Geo** tab displays key performance metrics by geographic location based on page loads. If you are using Browser Synthetic Monitoring for an application, you can view either Real User or Synthetic data using the **View** dropdown. See Browser Synthetic Monitoring Versus Browser Real User Monitoring.

The metrics displayed throughout the dashboard are for the region currently selected on the Map or in the Grid. For example, on the map if click on France, the widgets and trend graphs update to display data for France.

See Browser RUM Metrics for definitions of the metrics.



Map View Labels

The Map view displays load circles with labels for countries that are in the key timing metrics given in the right panel. Some countries and regions, however, are only displayed in the Grid view.

Regions and Countries

Regions are subdivisions of a country, such as a state, province, or city. In the default Map view, key performance metrics are displayed by country. You can check the **Show regions in Global View** box to view the key performance metrics by region. The default Grid and Map views display key performance metrics by country, but you can view metrics based on region. Because the Map view displays fewer regions than the Grid view, if you do not see a region displayed in the Map view, switch to the Grid view.

See Browser RUM Countries and Regions by Geo Dashboard for a list of the countries and regions available in the Map and Grid views.

Unknown Locations

An unknown location means the agent cannot determine the country from which the request originated.

You may also see metrics reported for a location named "Anonymous Proxy". This means that the agent cannot identify one or more private IP addresses, and the "Anonymous Proxy" represents aggregated metrics from those IP addresses.

One of the effects of Unknown regions is that it possible for a country to display as slow (red circles) on the global map, but when you drill down to the country all its regions appear normal (green circles). Or a country may display as normal on the global map, but some sub-regions may display as slow when you drill down.

Browser Snapshots

The **Browser Snapshots** tab captures and displays a broad set of metrics associated with a single request. You can drill down into errors and view any server-side transaction snapshots associated with that request (if your app server is instrumented with server-side app agents). See Browser Snapshots for more information.

Usage Stats

The **Usage Stats** tab presents aggregated page load usage data based on your users' browser type, device, and platform.

Usage Stats data helps you discover:

- The slowest browsers in terms of total end-user response time.
- The slowest browsers to render the response page.
- The browsers that most of your end users use.
- The browsers that most of your end users use in a particular country or region.

Devices data shows mobile access only via browsers on the device. **Devices** helps you discover:

- The slowest devices in terms of total end-user response time
- The slowest devices to connect to the server
- The devices that most of your end users use
- The devices that most of your end users use in a particular country or region

Browser Snapshots

Related pages:

- Transaction Snapshots
- Configure Browser Monitoring Snapshot Thresholds
- Add Custom User Data to a Page Browser Snapshot

This page describes browser snapshots. When Browser RUM is enabled, the JavaScript Agent collects browser snapshots for:

- Every base page (including virtual pages), iframe, and Ajax request; these serve as a heartbeat snapshot
- The slowest page for every region, every device, and every browser
- Unique JavaScript errors; identified by script name and line number
- Unique Ajax errors; identified by the HTTP error code in the Ajax response

For more information about browser snapshot collection, see Configure Browser Snapshot Collection.

Browser Snapshots Tab

A list of available browser snapshots appears. You can change the time frame of your search by changing the **Time Frame** dropdown.

List Key

 Normal user experience

 Slow user experience

 Very slow user experience

 JavaScript errors (Hover over the icon for a popup with summary information)

 Correlated server-side transaction snapshot exists

 Snapshot includes resource timing information

Browser Snapshot Types

There are three browser snapshot types, depending on whether the original object was a page, an iframe, or an Ajax request.

- Page Browser Snapshots
- Ajax Request Snapshots
- Iframe Snapshots

Page Snapshots

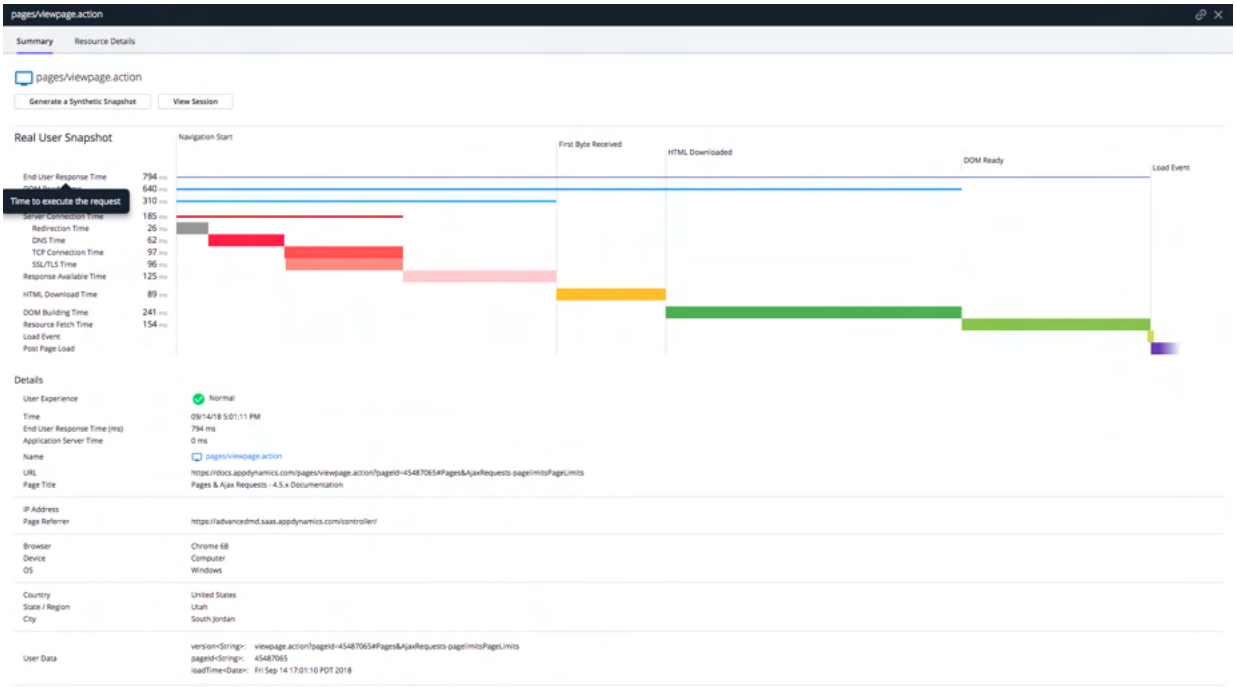
Related pages:

- Ajax Request Snapshots
- Iframe Snapshots

Page snapshots give you a detailed look at individual browser page requests. The **Summary** tab covers the general characteristics of the page.

Waterfall Graph

The top of the page snapshot **Summary** tab displays a waterfall graph of the overall transaction timing for the page. You can hover over each of the metrics to see a popup definition for that metric.



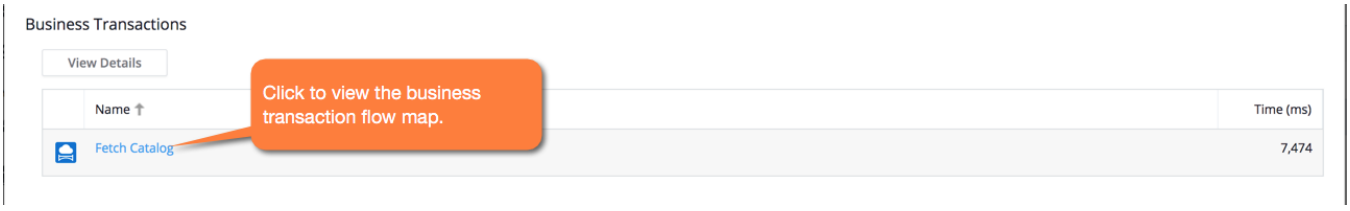
For more information about metrics, see Browser RUM Metrics. Additional details associated with the snapshot, including any related snapshots and any custom user data, are displayed as a table below the graph.

Generate a Synthetic Snapshot - SaaS Only

Synthetic snapshots use geographically distributed agents to create snapshots of web page download performance without the idiosyncratic and potentially skewing effects of real-user, last-mile performance. Click **Generate a Synthetic Snapshot** to create an on-demand synthetic snapshot to compare with a problematic real user monitoring snapshot.

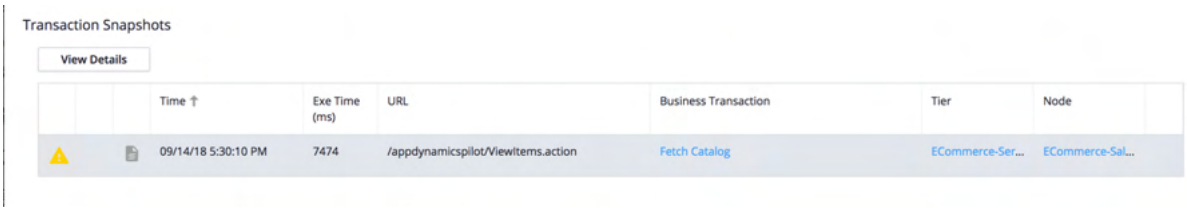
Correlated Transaction Snapshots and Business Transactions

If server-side correlation has been set up, a link to any related business transaction is shown:

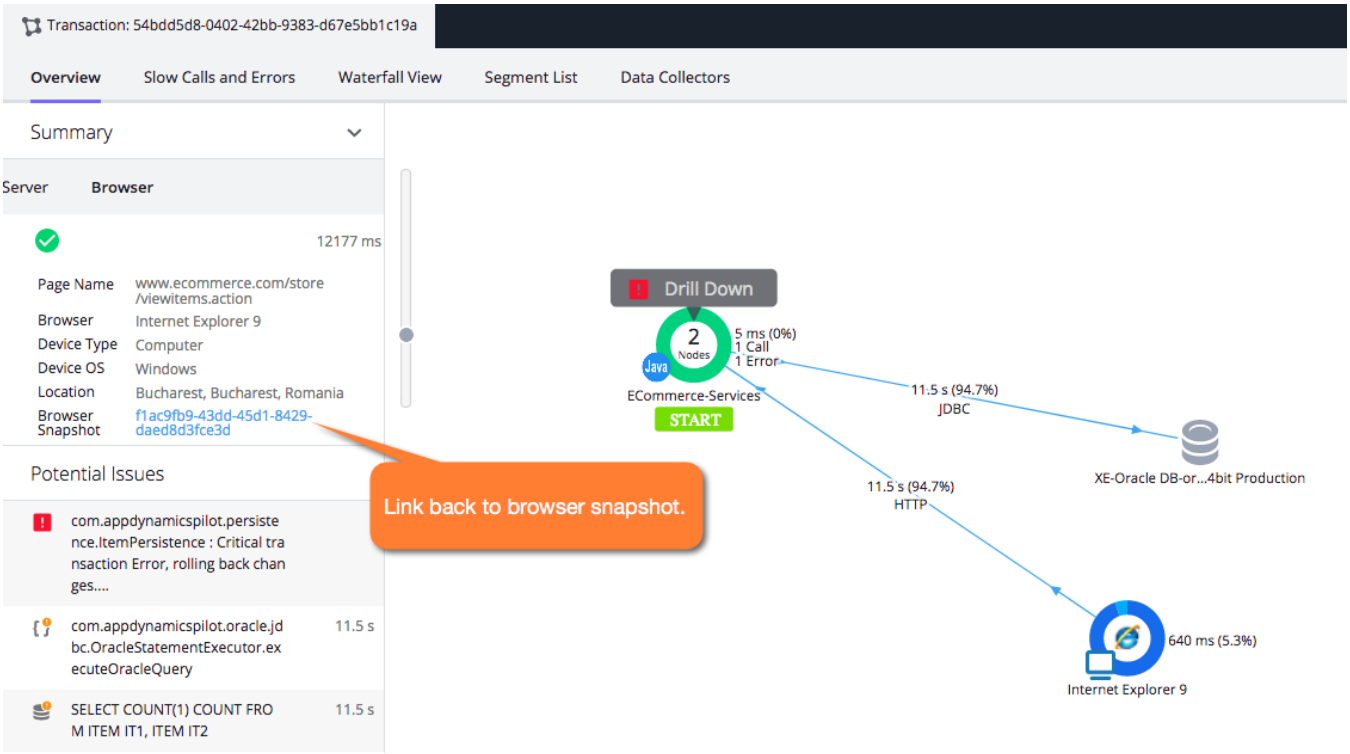


Clicking the link takes you to the flow map for that business transaction on the server side.

If the business transaction created a transaction snapshot, links to the business transaction, tier, and node flow maps are displayed, along with a direct link to the transaction snapshot flow map. Cross-app correlation is supported.

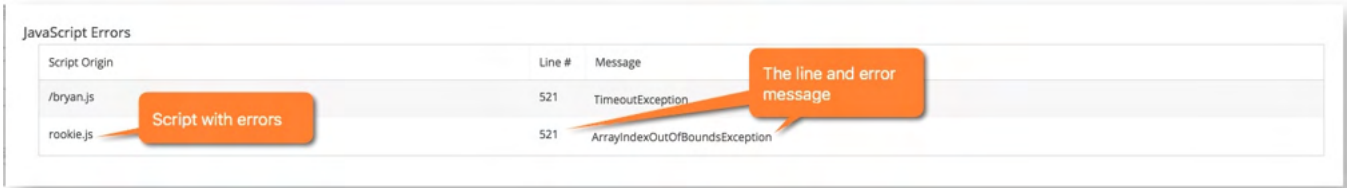


In this case, the transaction snapshot also has a link back to the browser snapshot.



JavaScript Errors

If there are JavaScript errors on the page, they are displayed below the waterfall graph.



You can configure to ignore errors if you are seeing too many irrelevant or excessive errors. See Configure JavaScript and Ajax Error Detection for instructions.

> When an error occurs in a script that is hosted on a domain other than the domain of the current page, most browsers prevent the JavaScript Agent from recording any details of the error. In such cases, the string `CROSSDOMAIN` is displayed as the **Script Origin**.

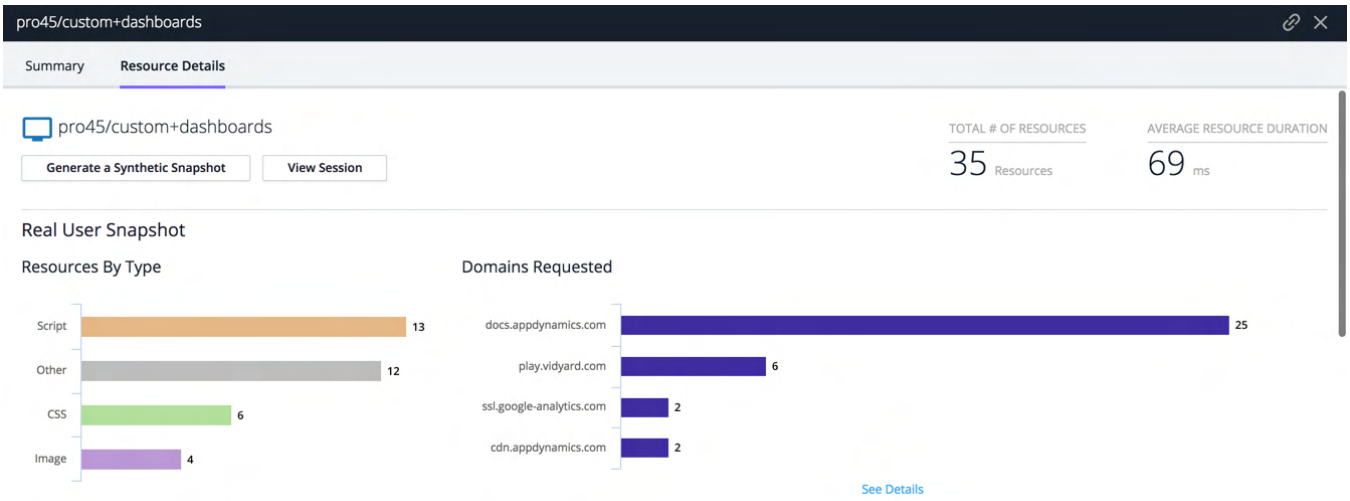
User Data

If you have custom user data set up, the data appears in the **Details** section:

User Data	algo<String>:	LOG(N)
	Random Number<String>:	1938299132
	hotel<String>:	AppDynamics Resort

Resource Details

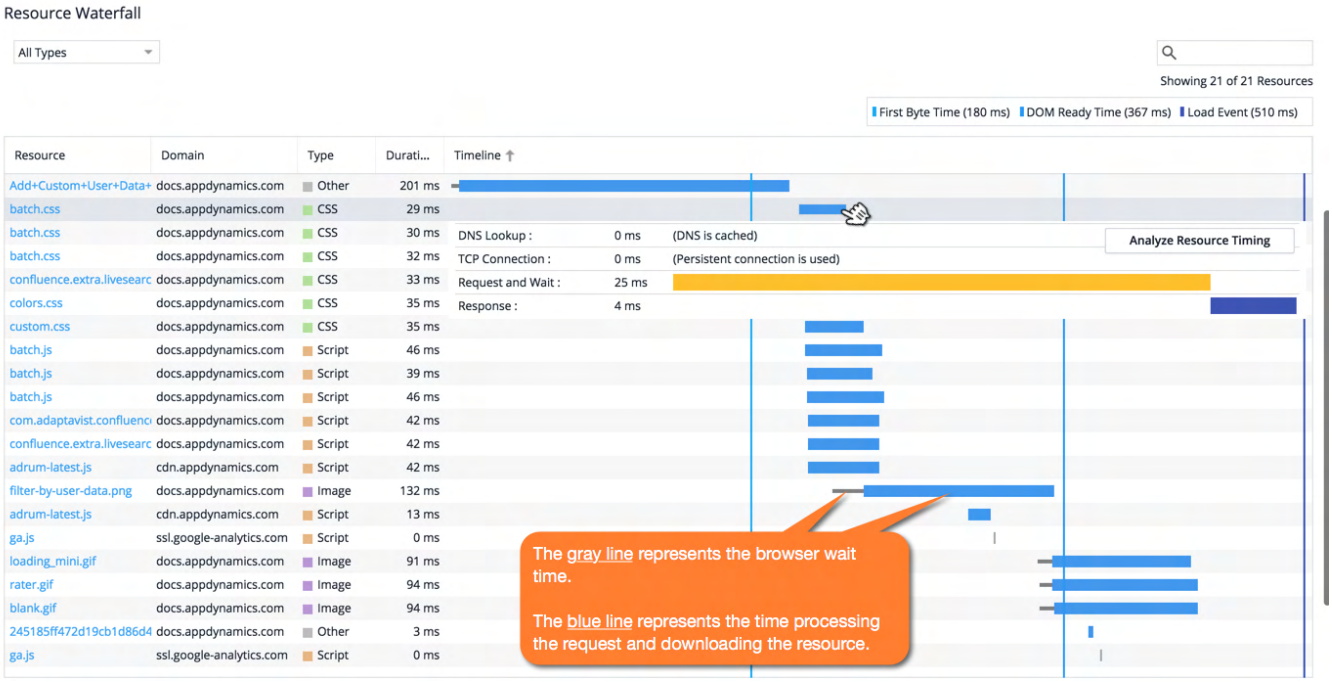
If the browser supports the Resource Timing API, the Resource Details tab gives you a detailed breakdown of the performance of resources—scripts, CSS files, fonts, SVGs, and images—as they are loaded into the page.



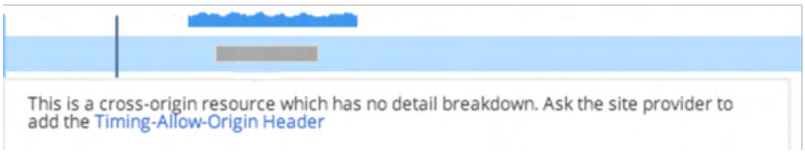
Resource Waterfall

The **Resource Waterfall** provides a snapshot of when each resource was loaded in the page load process (before First Byte Time, before DOM Ready Time, before Onload), including the name of the resource, the domain from which it was fetched, the type of resource, and how long it took. You can filter the items by type using the upper-right dropdown and search using the search box.

To see more details, click any of the blue timelines. These are from the primary domain or cross-domains that have set the `Timing-Allow-Origin` HTTP header. See Resource Timing Metrics for descriptions and explanations about how the metrics are calculated.



Grey timelines indicate cross-domains that have not set the `Time-Allow-Origin` HTTP header. Working with your CDN provider to add this header can mean you get better information on shared and CDN-served content.



Unknown Metrics in Page Snapshots

Browser RUM captures metrics using your end-users' web browsers. Occasionally, you may see unknown data reported for one or metrics in a browser snapshot. This occurs on older or less sophisticated browsers that do not support the collection of a given metric.

See Browser Monitoring Metrics for details about which metrics may not be captured based on browser capabilities.

Ajax Request Browser Snapshots

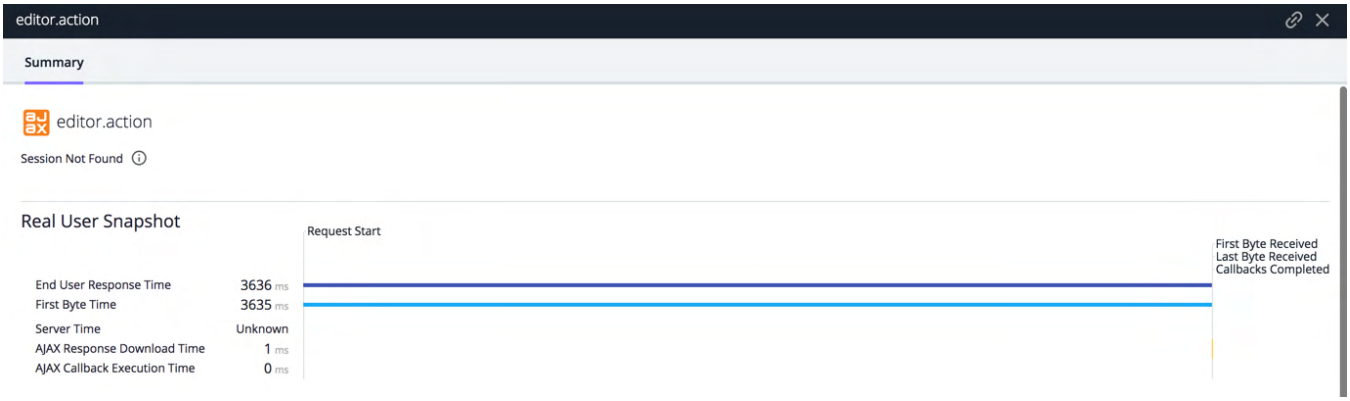
Related pages:

- Page Snapshots
- Iframe Snapshots

Ajax request snapshots give you a detailed look at an individual Ajax request. The **Summary** tab of the snapshot describes the general characteristics of the page.

Waterfall Graph

The top of the Ajax snapshot displays a waterfall graph of the overall transaction timing for the Ajax request. The snapshot is labeled either a **Real User Snapshot** or a **Synthetic Snapshot** based on the origin of the request. You can hover over each of the metrics to see a popup definition for that metric.



For a detailed description of what these metrics mean, see Browser RUM Metrics.

Details

Much of the information is the same as you see for page snapshots. The main differences are:

- If there is an Ajax error, the error code returned with it is listed.
- The parent page from which the Ajax call originates is listed.

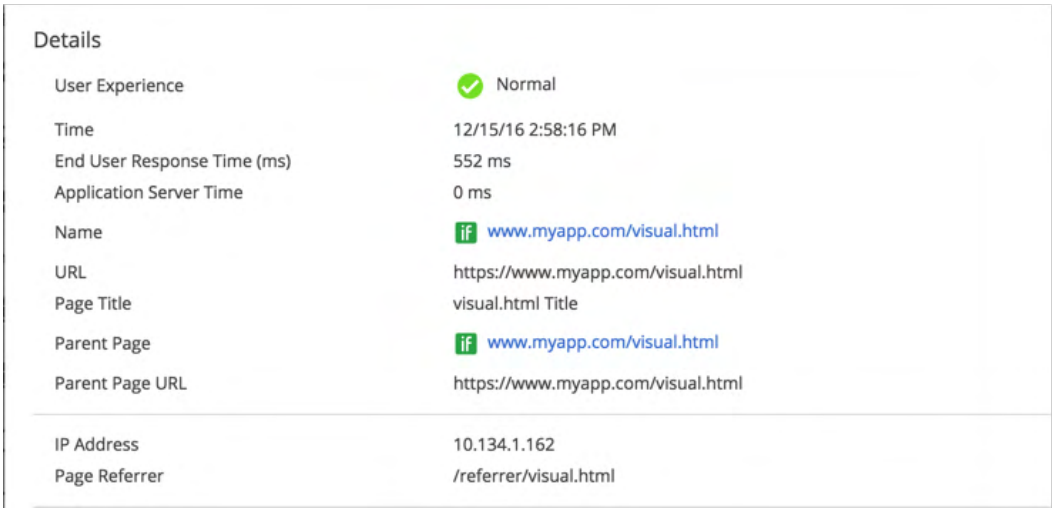
You can configure errors to ignore if you are seeing too many errors that are not of interest. See Configure JavaScript and Ajax Error Detection.

Iframe Browser Snapshots

Related pages:

- Page Snapshots
- Ajax Request Snapshots

Iframe-based snapshots give you a detailed look at an individual iframe request. The UI display is identical to Page Snapshots except in the **Details** section , where the parent page may also be displayed.



Resource Performance Dashboard

Deployment Support

Related pages:

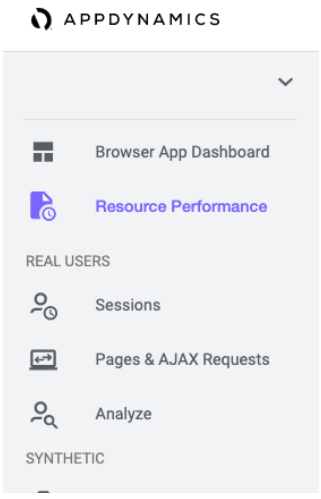
- Use Cases for the Resource Performance Dashboard
- Browser RUM Metrics
- Configure Resource Violation Rules

The **Resource Performance Dashboard** provides a high-level overview of how your resources affect the performance of your browser application. You can use this dashboard to pinpoint resource-related performance issues affecting the user experience, such as the following:

- A prioritized list of resource performance issues by comparing their performance against thresholds
- Changes in the number of resources
- Large resources (images, JavaScript, CSS, etc.)
- Size increase of resources impacting performance. For example, a page banner might be replaced with an uncompressed image, slowing down the page load.
- Slow CDNs
- Resources that haven't been compressed
- Comparison of real user and synthetic resource performance

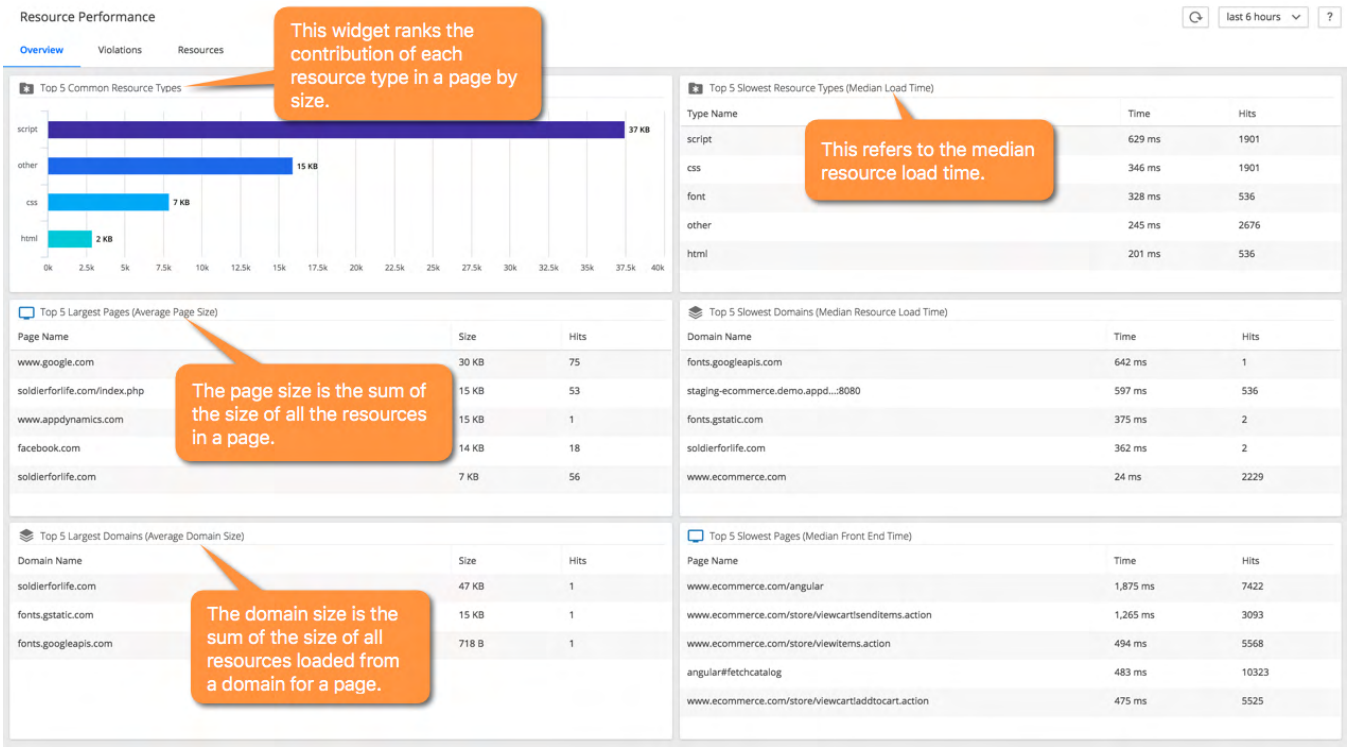
Overview of the Resource Performance Dashboard UI

Once you navigate to a browser application, the **Resource Performance Dashboard** is located on the left-side panel.



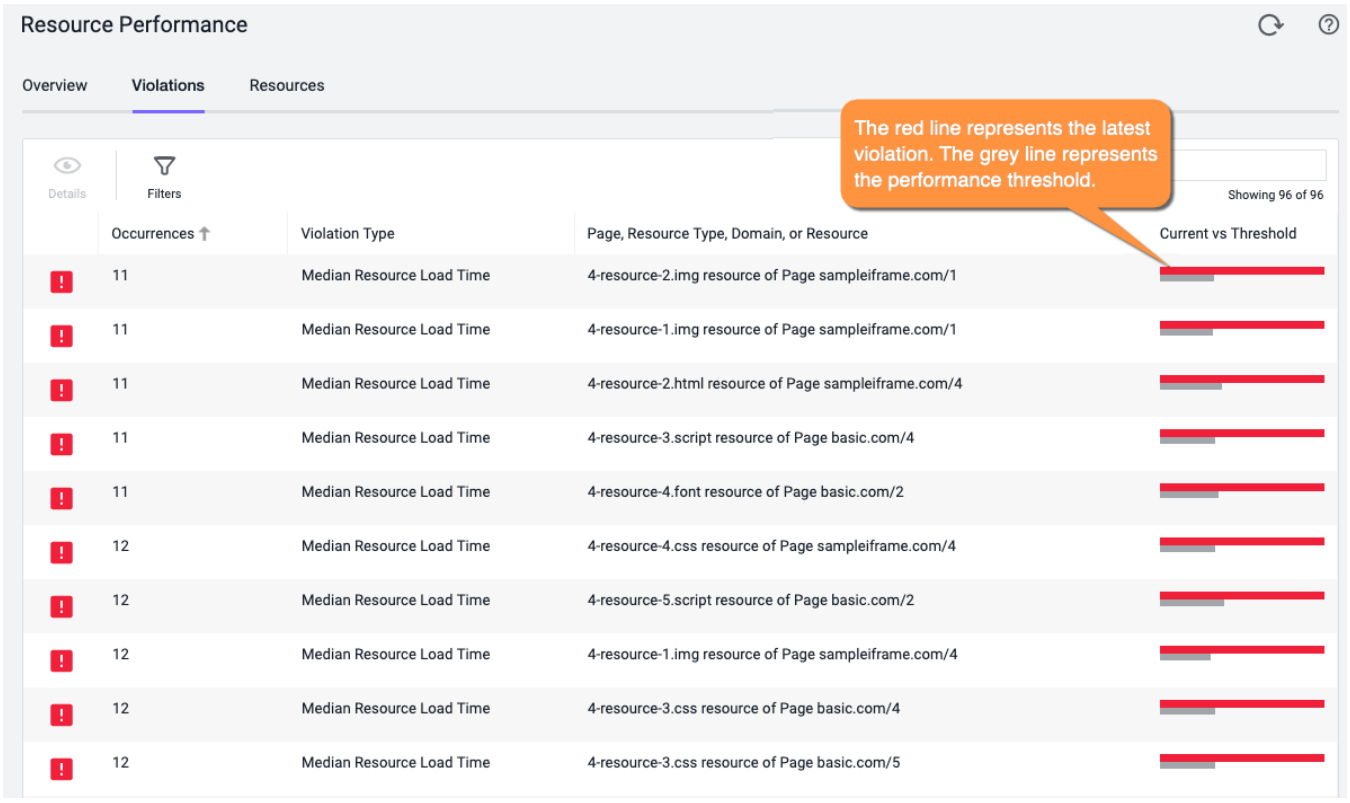
Overview Tab

The **Overview** tab displays widgets providing high-level indicators of resource performance over a specified time period. The dashboard can be filtered to real user or synthetic data. The widgets only show a small number of resources, but you can click **See More** to view up to 100 resources per widget.



Violations Tab

The **Violations** tab shows a list of pages, resource types, and domain or resource violations that have exceeded performance thresholds. You can use the **Violations** tab to not only find problematic resources, but also to become aware of sudden changes that negatively impacted the performance of a resource. Clicking a specific violation leads to the **Resources** tab, and the data is filtered with that violation for further diagnostics. The configured violation rules are evaluated every 10 minutes for the last 30 minutes.



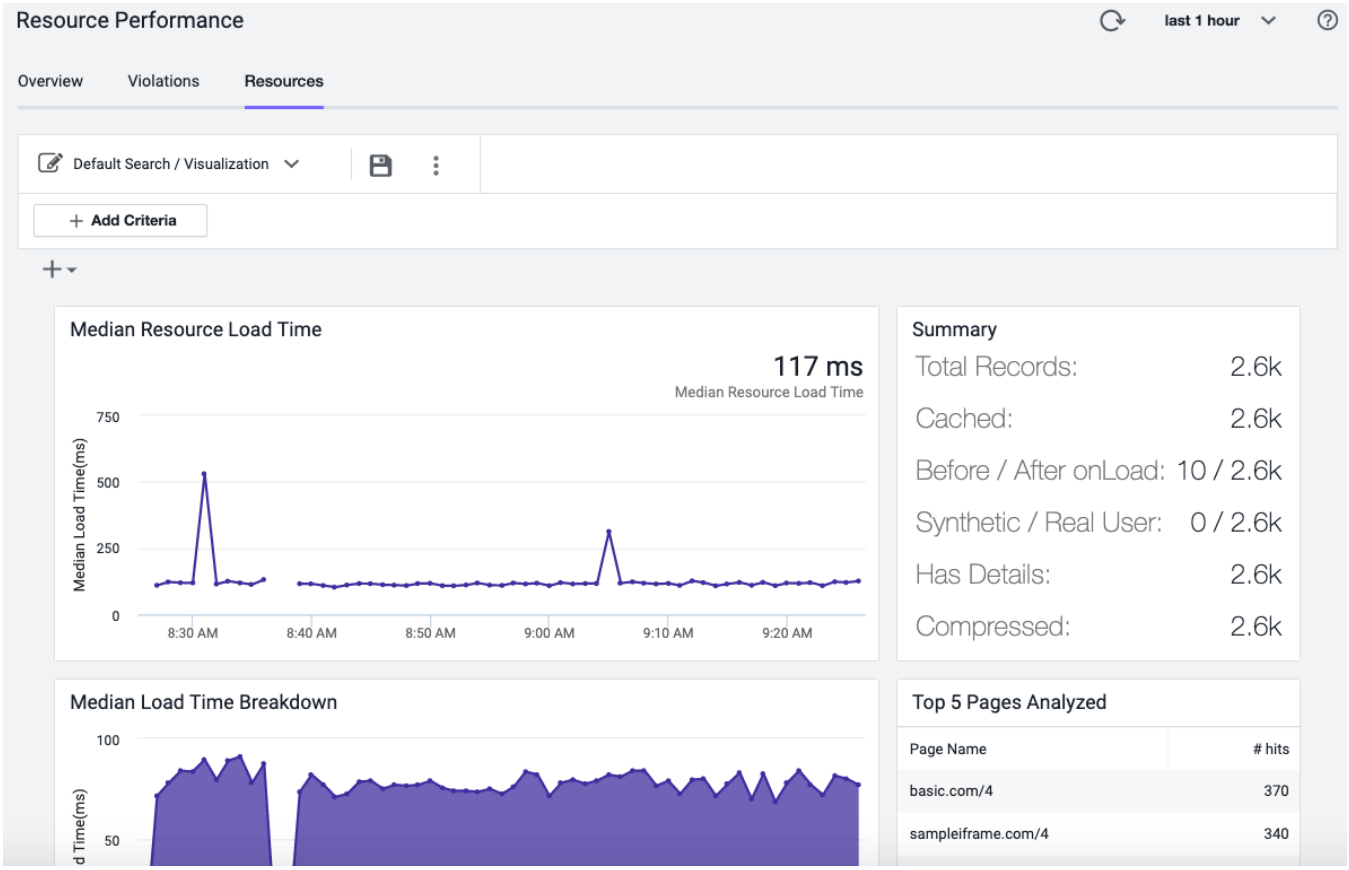
Violation Types

The supported violation types include:

- Median Domain Load Time
- Average Domain Size
- Average Page Size
- Median Resource Load Time
- Average Resource Size
- Median Resource Type Load Time
- Average Resource Type Size

Resources Tab

You can use the **Resources** tab to diagnose a problematic resource. You can also add criteria as a filter to the widgets. All use cases to troubleshoot a resource leads to the **Resources** tab where you can learn more about an individual resource's impact on an application. See Use Cases for examples.



Additional Resource Performance Data Information

For the **Resource Performance Dashboard** to be effective, it is highly recommended that you set the `Timing-Allow-Origin` HTTP header in all of your CORS domains to enable access to resource timing information. Without this header, the JavaScript Agent cannot capture the resource size, and, of the supported Resource Timing Metrics, only the resource load time can be calculated.

For information about resource performance data retention, see License Entitlements and Restrictions.

Use Cases for the Resource Performance Dashboard

The Resource Performance Dashboard provides a high-level overview of how your resources affect the performance of your browser application.

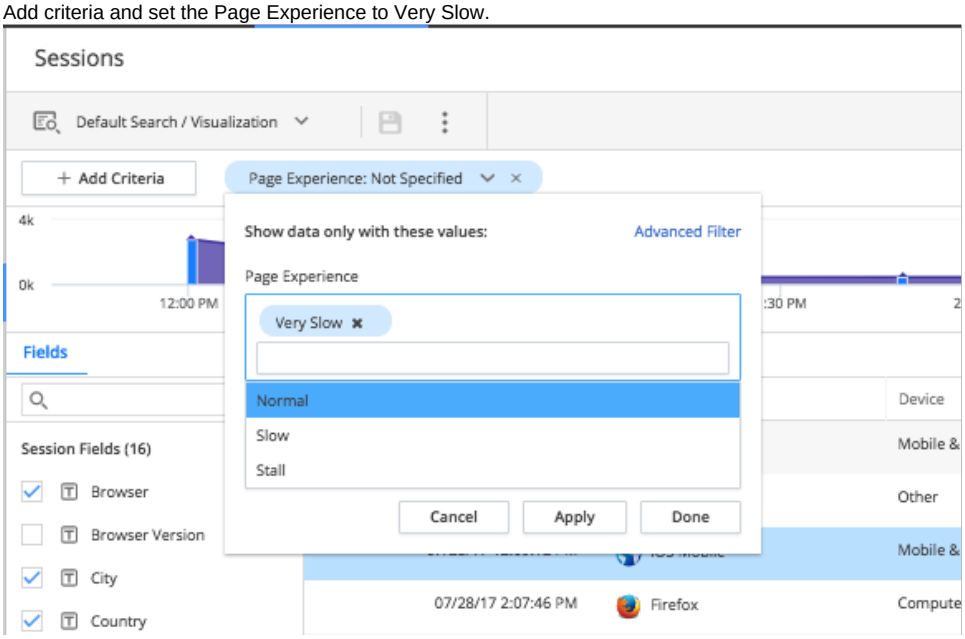
The table presents common troubleshooting and analysis use cases for resource performance.

Goal	Use Case
Detect and resolve poor performance issues	Troubleshoot from Sessions
Track and monitor resource violations	Discover Top Resource Violations
View a resource's performance from a particular region	Analyze Resources by Region
Diagnose cause of lengthy load time for image resources	Diagnose Problematic Resources

Troubleshoot a Resource from Sessions

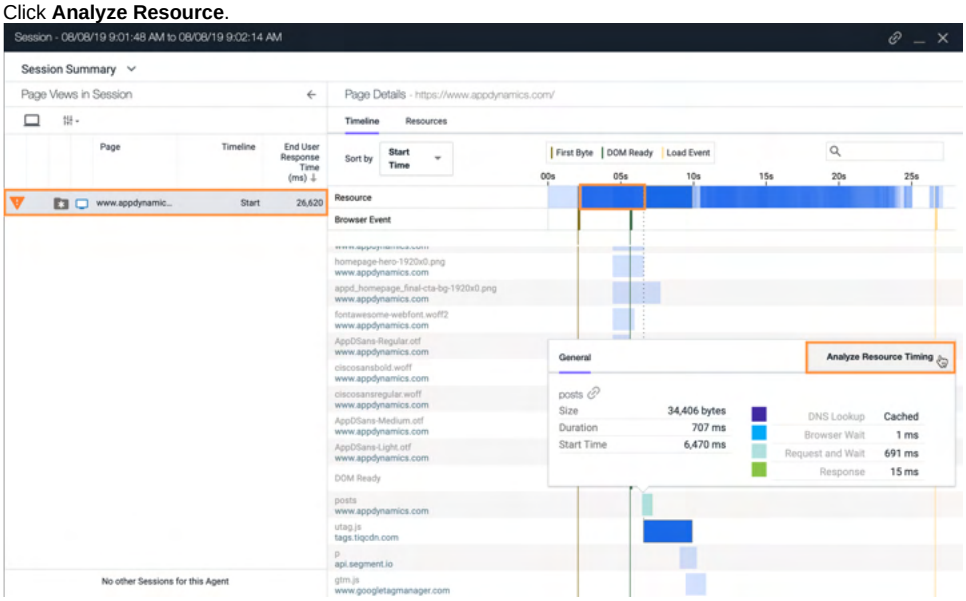
Follow these steps to troubleshoot a resource from **Sessions**:

1. Go to **Sessions**.
2. Add criteria and set the Page Experience to Very Slow.

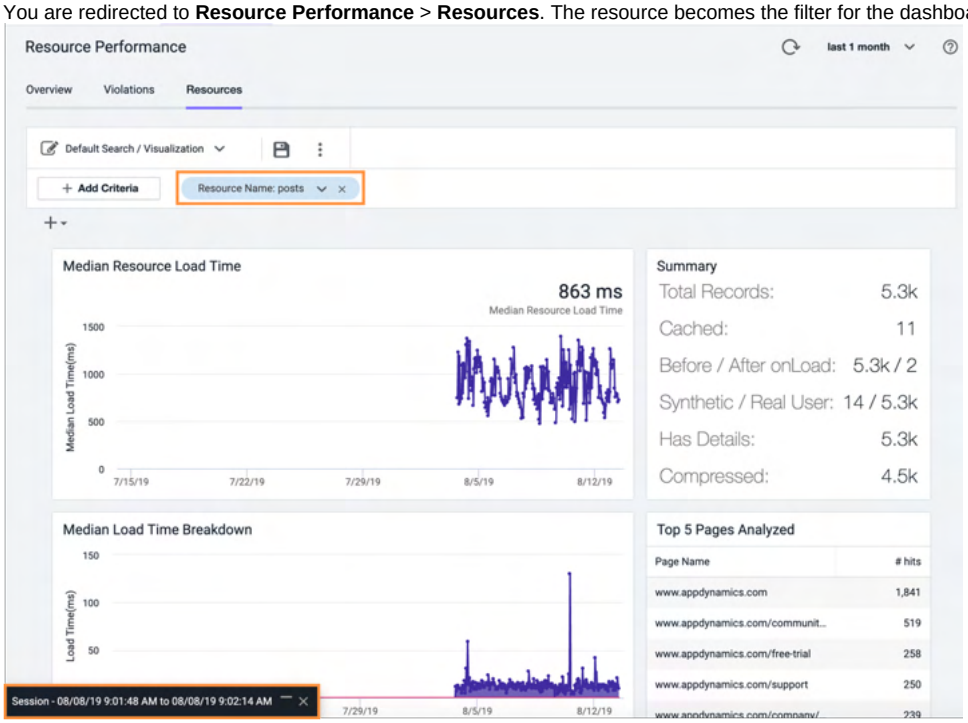


3. Double-click on a session record.
4. In the **Session Summary** select the page with the longest **End User Response Time**.
5. In the **Waterfall** tab, click a problematic resource to open the **Resource Details** dialog.

6. Click **Analyze Resource**.



7. You are redirected to **Resource Performance** > **Resources**. The resource becomes the filter for the dashboard data.



8. In the widgets, look for patterns or possible factors such as geographic location, domains, and so on. See Diagnose Problematic Resources to learn some strategies for diagnosing resource issues.

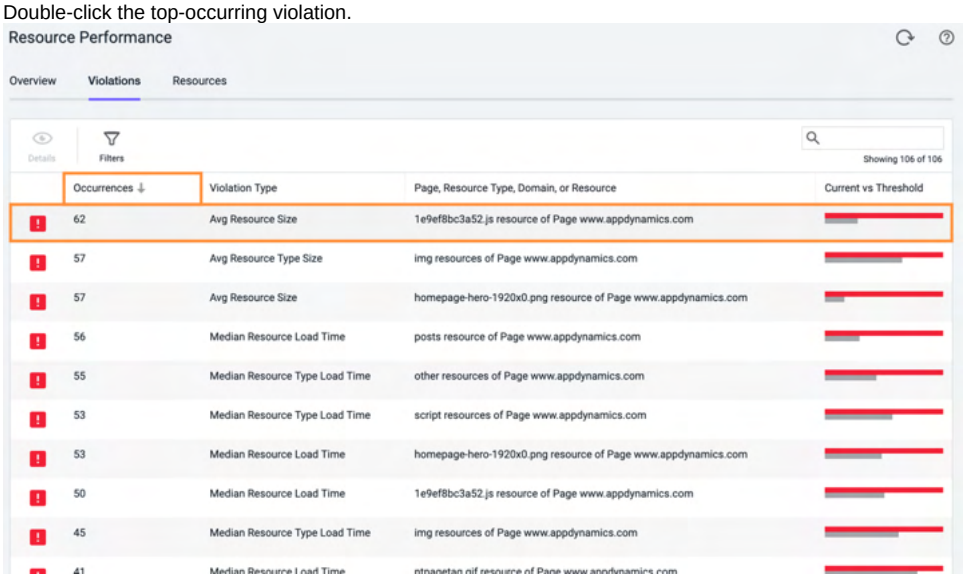
> To rule out environmental factors such as network connections, browser, and so on, create a synthetic job that calls a page with the resource to confirm the resource is the problem.

Discover Top Resource Violations

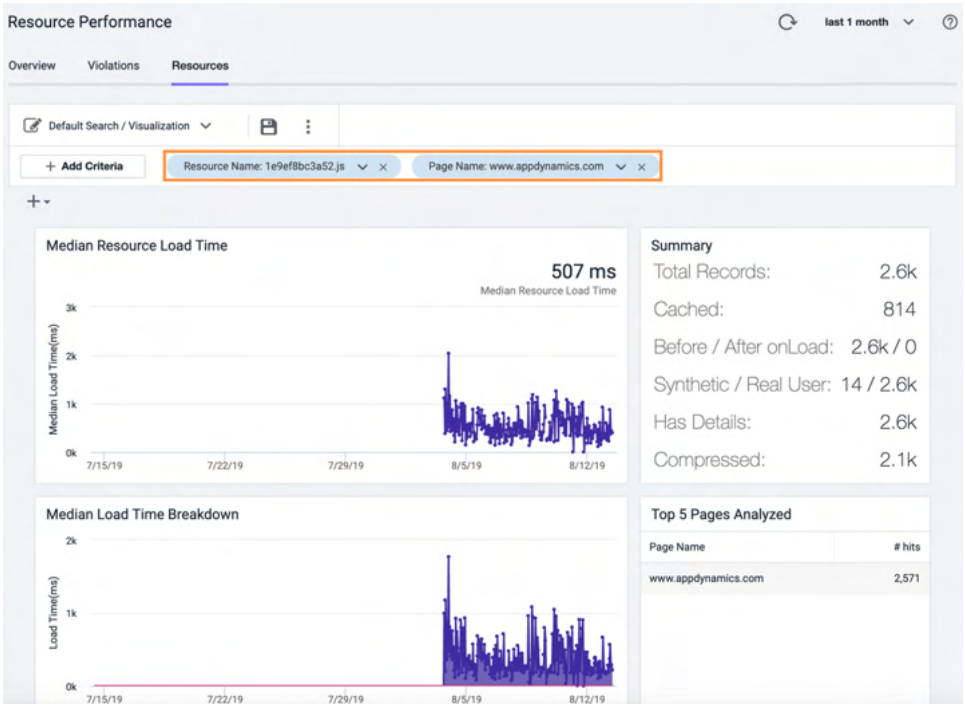
Follow these steps to determine which resource is causing the most performance violations:

1. Go to the **Resource Performance** > **Violations**.
2. Select the **Occurrences** column so that number of occurrences is in descending order.

3. Double-click the top-occurring violation.



4. You are redirected to the **Resources** tab with information for the resource in violation.

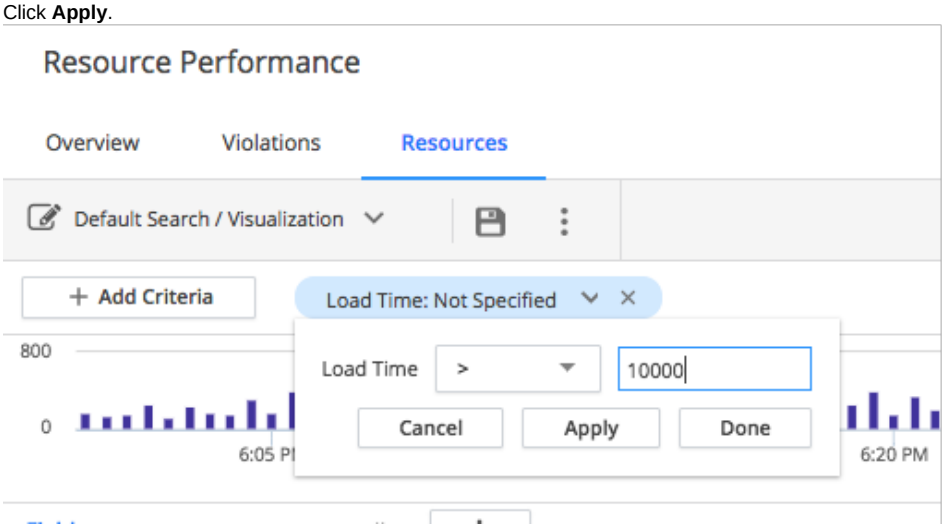


Analyze Resources by Region

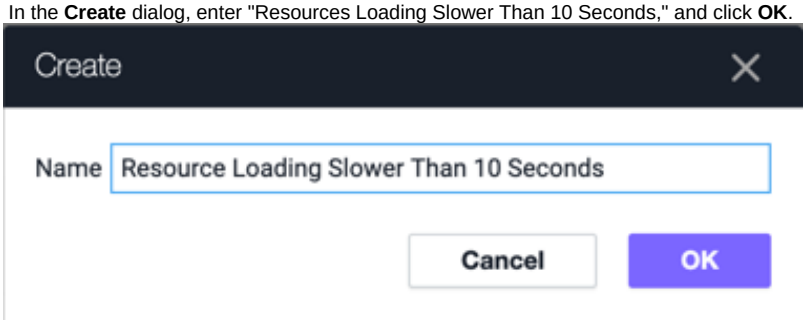
Follow these steps to analyze and monitor a resource based on a region:

1. Go to the **Resources** tab.
2. From the **Add Criteria** drop-down, select **Load Time** and specify the value to be greater than 10000 (10 seconds).

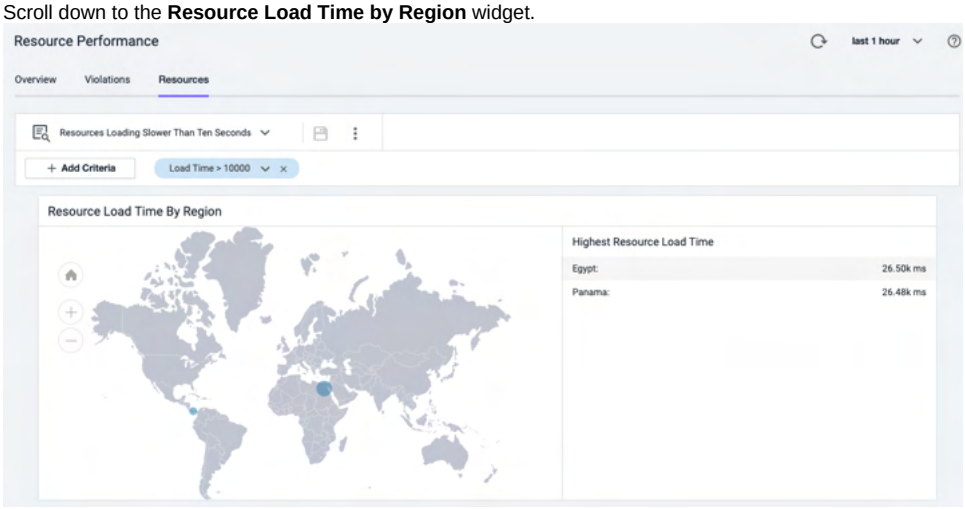
3. Click **Apply**.



4. Click  .
5. In the **Create** dialog, enter "Resources Loading Slower Than 10 Seconds," and click **OK**.



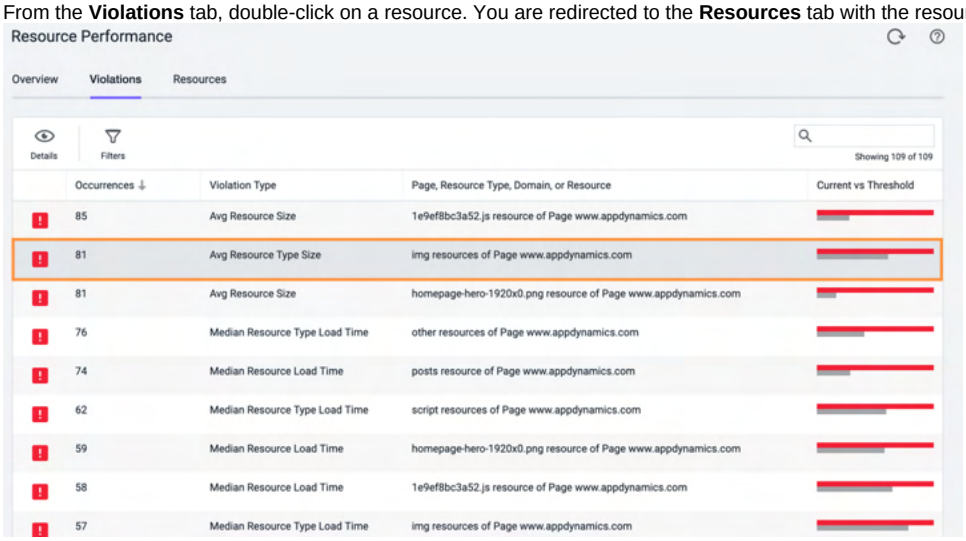
6. Scroll down to the **Resource Load Time by Region** widget.



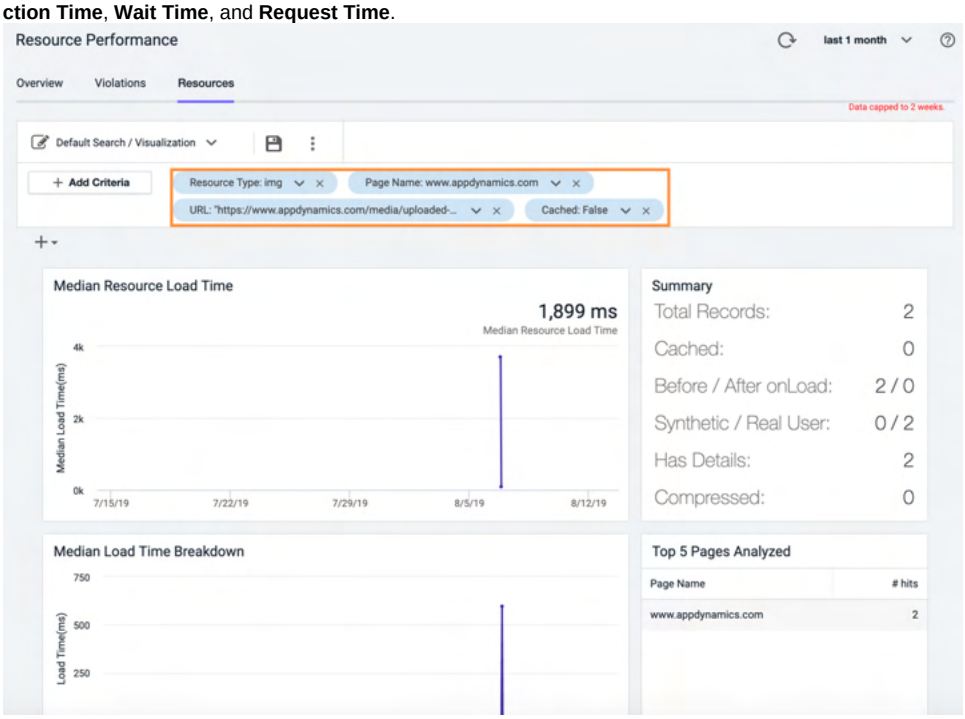
Diagnose Problematic Resources

Follow these steps to diagnose why certain image resources are taking a long time to load:

1. From the **Violations** tab, double-click on a resource. You are redirected to the **Resources** tab with the resource applied as a data filter.



2. Select criteria fields to determine possible causes, such as **Region**, **Cached**, **Compressed**, **Load Time**, **Domain**, and so on.
    - Example: Confirm that resources are cached or compressed.
    - Example: Add criteria that may indicate issues with the server based on the location of the resource, such as **DNS Lookup Time**, **Connection Time**, **Wait Time**, and **Request Time**.



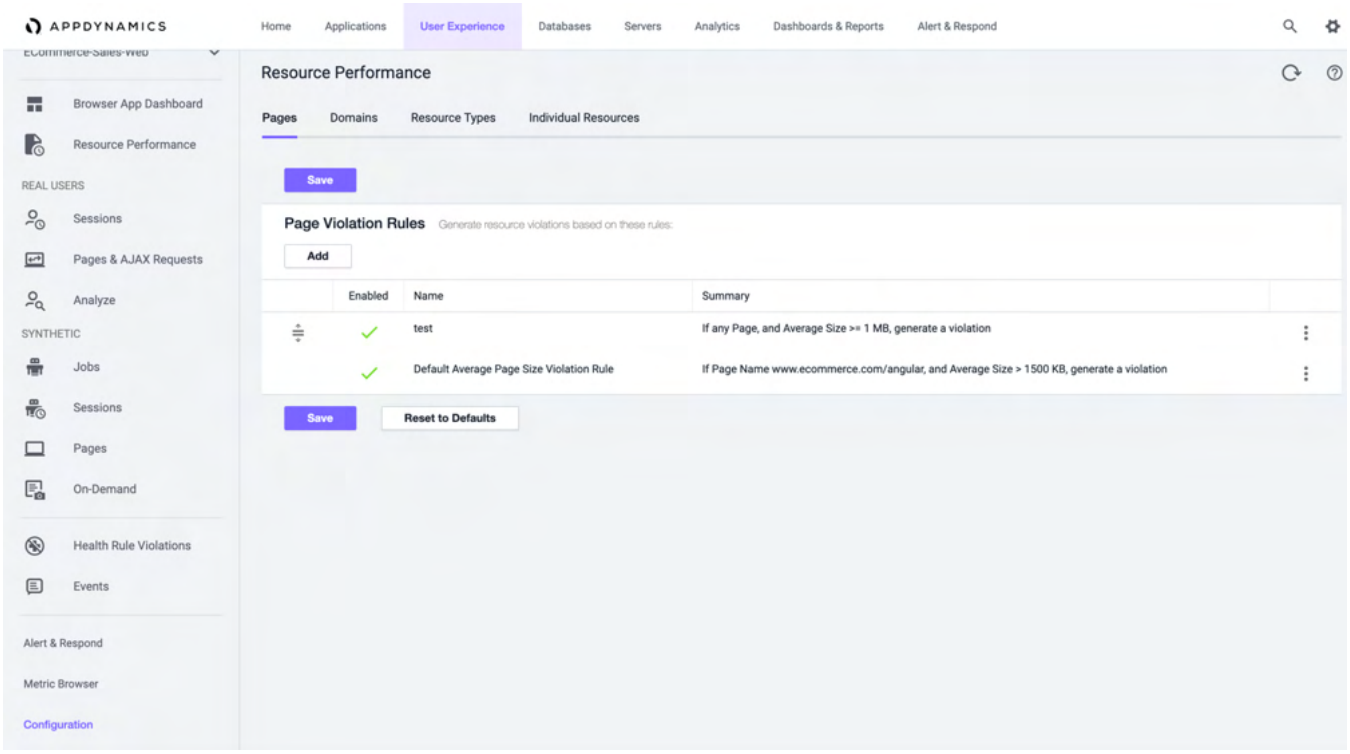
3. Search for patterns that indicate the cause within the widgets, providing insights such as:
    - Resource load time speed in certain regions
    - Top pages using the resource
    - Median load time breakdown showing that load time slows down at certain intervals
    - Resource size increases

To rule out environmental factors such as network connections, browser, and so on, create a synthetic job that calls a page with the resource to confirm the resource is the problem.

Configure Resource Violation Rules

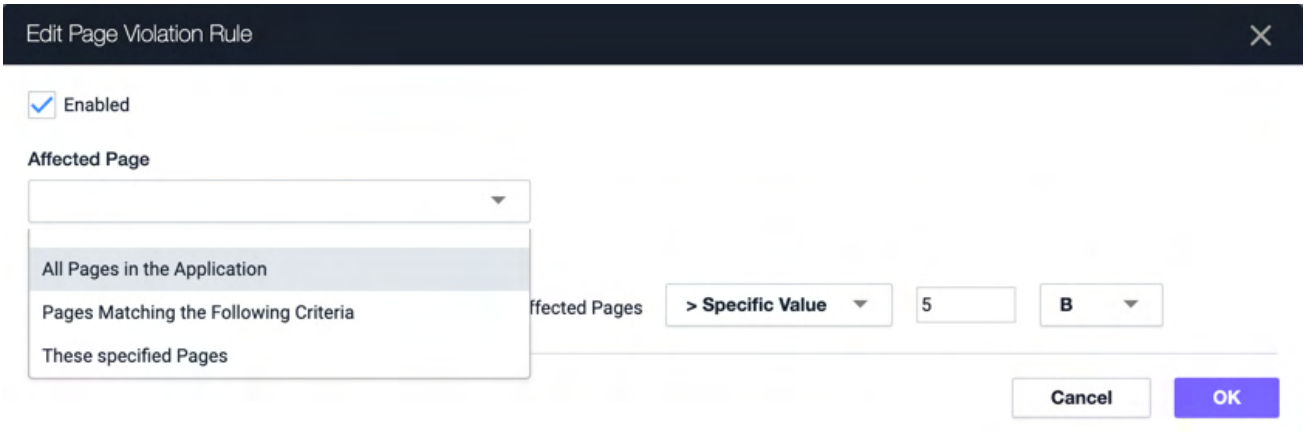
This page describes how to configure resource violation rules for the Resource Performance Dashboard. You can configure violation rules for pages, domains, resource types, and individual resources. You can use or edit the default rules per category or create your own rules. Resource Performance configuration is only available to admin-level users.

To configure resource violation rules, navigate to **Configuration** > **Resource Performance**.



Pages Violation Rules

Page violations help monitor your most important pages and ensure they are light and efficient for end users. You can set violation rules for all pages, individual pages, or by pages matching query criteria within an application. The average page size can be set to megabytes, kilobytes, or bytes. This helps you troubleshoot application performance based on pages.

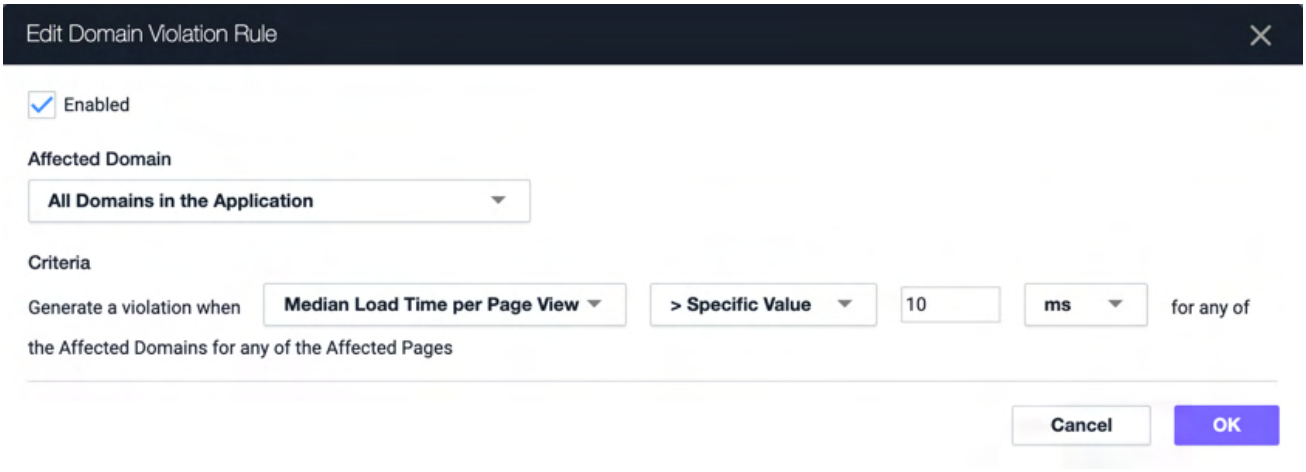


Use Case

In this scenario, your home page size average is 1500 KB, and you want to be notified when it surpasses that threshold. You can set your home page at a maximum size of 1500 KB. If the violation rule is triggered, you will see the violation in the Violations tab as well as in the "Resource Performance" widget in the Browser App Dashboard.

Domain Violation Rules

Domain violations help monitor the effects of domain performance on your application. You can set violation rules for all affected domains, individual domains, or by domains matching query criteria within an application. The median load time per page view can be set to milliseconds or seconds, and the average domain size can be set to megabytes, kilobytes, or bytes. This helps you troubleshoot poor application performance based on domains.

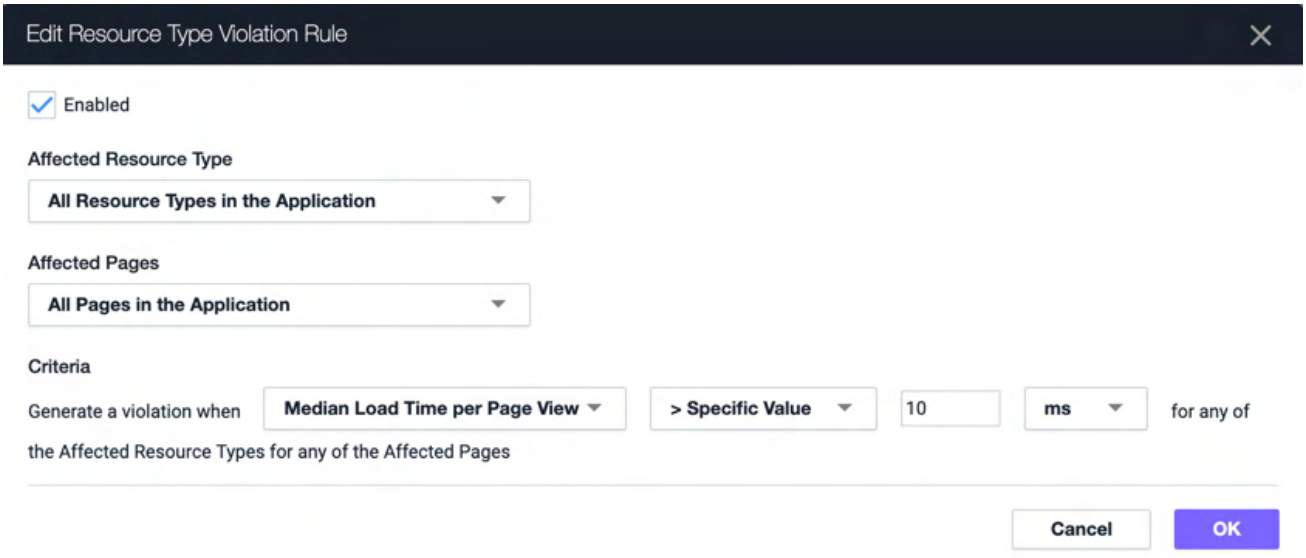


Use Case

In this scenario, you want to monitor the performance of your CDN. You can set your CDN domains to a maximum median load time of 1000 ms for all domains in the application. If the violation rule is triggered, you will see the violation in the Violations tab as well as in the "Resource Performance" widget in the Browser App Dashboard.

Resource Type Violation Rules

Resource type violations help monitor the effects of resource types, such as images, html, css, and so on, within your application. You can set violation rules for all affected resource types or by individual resource type within an application. You can also set the rule to one page, pages matching query criteria, or all pages in the application. The median load time per page view can be set to milliseconds or seconds, and the average domain size can be set to megabytes, kilobytes, or bytes. This helps you troubleshoot poor application performance based on resource types.

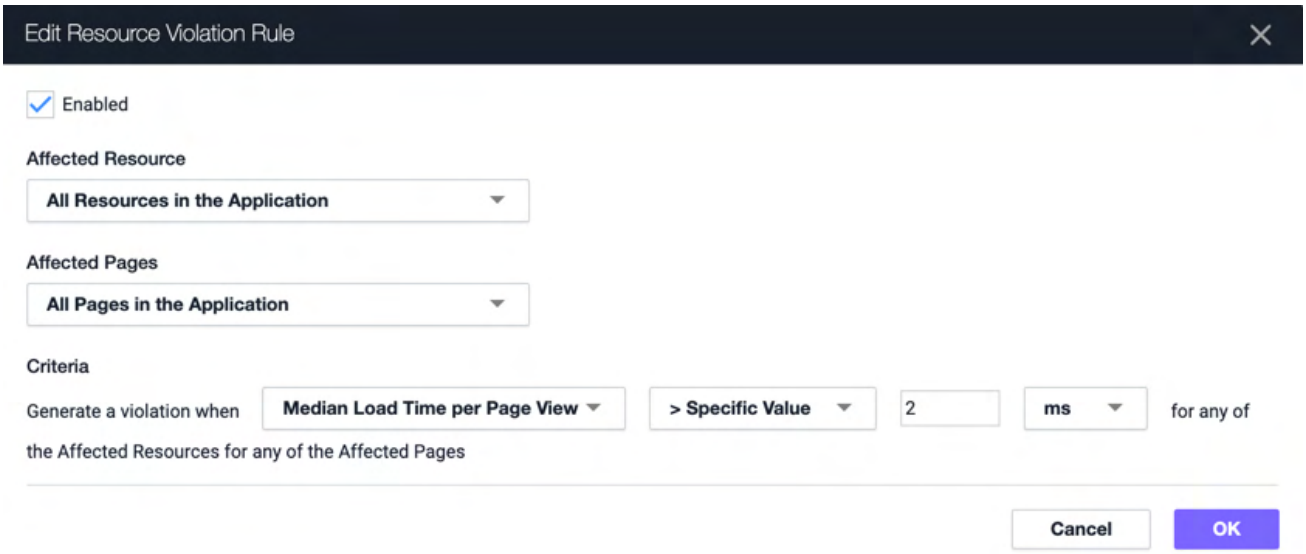


Use Case

In this scenario, you want to monitor all the images in all pages across your application. You can set all images to maximum average size of 500 KB for all pages in the application. If the violation rule is triggered, you will see the violation in the Violations tab as well as in the "Resource Performance" widget in the Browser App Dashboard.

Individual Resource Violation Rules

Individual resource violations help monitor the effects of one resource across your application. You can set violation rules for all affected resources or by resources matching query criteria within an application. The median load time per page view can be set to milliseconds or seconds, and the average domain size can be set to megabytes, kilobytes, or bytes. This helps you troubleshoot poor application performance based on individual resources.



Use Case

In this scenario, you want to monitor all resources on your home page. You can set all resources to a maximum load time per page view at 2 ms within the scope of the home page. If the violation rule is triggered, you will see the violation in the Violations tab as well as in the "Resource Performance" widget in the Browser App Dashboard.

Browser Real User Monitoring

Related pages:

- End User Monitoring

This page provides an overview of AppDynamics Browser Real User Monitoring (Browser RUM).

Browser Real-User Monitoring (Browser RUM) allows you to see how your web application is performing from the point of view of a real end user. You can drill into the data to explore how users experience your application in their web browsers and answer questions like:

- Which 1st or 3rd-party Ajax or iframe calls are slowing down page load time?
- How does server performance impact end-user experience in aggregate or in individual cases?

Monitor Your Application

Browser RUM offers multiple ways to look at your data in real time. You can:

- Understand and improve your web page's performance
  - Know how your pages, Ajax requests (XHR, Fetch API calls), and IFrames are performing over time. See The Pages & Ajax Requests View.
  - Gain insight into individual requests, with detailed charts on how your pages, Ajax requests, and IFrames load and build in your end user's browsers, with links, if enabled, to reports on server-side performance. See Browser Snapshots.
  - Find your worst performing pages by multiple common metrics. See Top Pages.
- Reduce errors
  - Learn which pages are loading with JavaScript errors, including the script file and line number that are creating the problem. See Browser Snapshots.
- Learn about your users
  - See how your web users are connecting to your application by device/platform and browser. See Browser App Dashboard.
  - Find out where geographically your web users are and how your application is performing across countries and regions. See Browser App Dashboard.
  - Understand how users are navigating through your website and what actions they are taking. See Browser RUM Sessions.

For more information on using Browser RUM, see Overview of the Controller UI for Browser RUM.

How Browser RUM Works

Browser RUM works in the following way:

1. An end user requests the first page from your web application.
2. Your web application executes whatever business logic that particular page requires.
3. Your web application creates the response page to return to the end user. The response page includes:
   a. application-specific information.
   b. a copy of a small JavaScript script that knows how to collect relevant performance information about that page. This script is called the JavaScript Agent.
4. The page, with the JavaScript Agent included, is returned to the end user.
5. As the page is being constructed in the browser, the script collects relevant information about the page's performance.
6. At approximately the same time as the `onload` event for the page fires, a copy of a somewhat larger JavaScript file, the JavaScript Agent extension, is downloaded asynchronously by the injected agent.
7. This second script packages the collected performance information and sends it via a web beacon to the EUM Server collector for processing.
8. The two scripts work together to collect and send performance information as the end user navigates through the instrumented pages of your application.

Set Up and Configure Browser RUM

Browser RUM is easy to set up. It is also highly configurable. You can:

- Set up and enable Browser RUM. See Set Up and Access Browser RUM.
- Instrument your application to work with Browser RUM. For more information, see Configure the JavaScript Agent.
- Set up how your information appears in the Controller UI. For more information, see Configure the Controller UI for Browser RUM.

License and Enable Browser RUM

- Browser RUM requires a separate license and must be enabled before it is available for use. For information about licensing, including a description of the types of licenses, see EUM Accounts, Licenses, and App Keys.
- For information on enabling or disabling Browser RUM, see Enable and Disable Browser RUM.

Set Up and Access Browser RUM

Related pages:

- EUM Accounts, Licenses, and App Keys.
- Inject the JavaScript Agent

This page explains how to set up Browser Real User Monitoring (Browser RUM). Browser RUM collects metrics on your end user's web browsers via the AppDynamics JavaScript Agent. Your web application must be configured to insert this agent into the web pages. This process is called injection. See Inject the JavaScript Agent on how to inject the JavaScript Agent into your web pages.

To access Browser RUM, you need an EUM Account and a Browser RUM license. EUM must be enabled through the Controller UI. You can Enable and Disable EUM if you decide not to use the feature.

EUM Account

A special EUM account is required. An EUM account is created when you have a license for Browser RUM, Browser Synthetic, or Mobile RUM. There is an account license key associated with this account.

For on-premises customers, the account license key is set up within the Controller license file. If you added EUM after your initial installation, you may need to upload and install the new Controller license file. If you are a SaaS customer, the EUM license is set up in the SaaS Controller for you.

Each Browser RUM license provides a certain number of page views per year. For information on how to examine your current page view usage, see EUM Accounts, Licenses, and App Keys.

Access Browser Monitoring Configuration

Your data is displayed in the Controller UI in the context of an EUM application. You must assign unique names to EUM applications and business applications. For example, if you created the business application "E-Commerce", you cannot create a browser, mobile, or IoT application with that same name or vice versa. An EUM application name is associated with an EUM App Key, which identifies data coming into the Controller.

1. If you do not yet have an EUM application, click the **User Experience** tab, the **Browser Apps** tab, and the **+Add** button. You can create an application name either by using the **Getting Started Wizard** or by simply choosing an application name.
2. Open the EUM application you are interested in by double-clicking on the application name in the **Browser Apps** list.
3. In the left navigation bar of the application UI, click **Configuration.**

Browser RUM Prerequisites

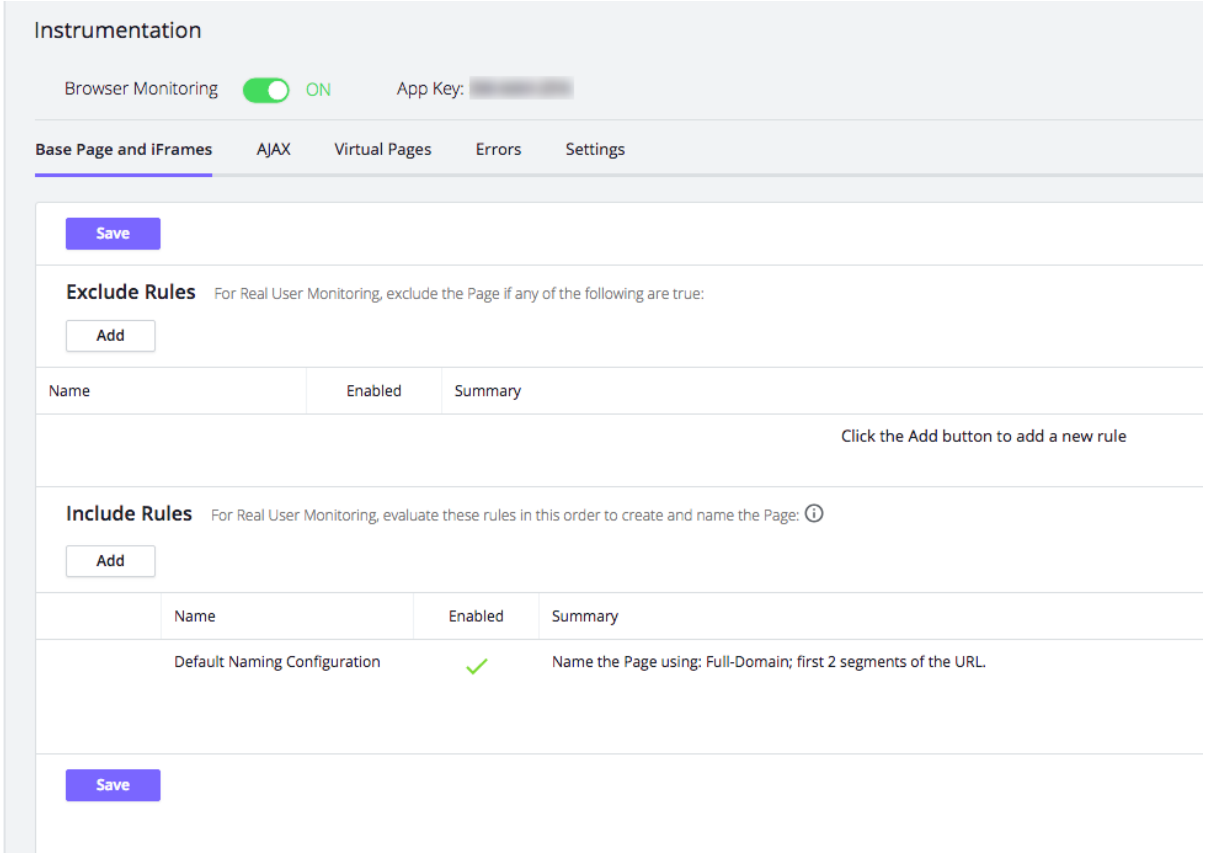
To turn on Browser RUM, you need to:

- Enable Browser RUM
- Inject the JavaScript Agent into your application pages

If you are an on-prem customer, your JavaScript Agent version should be equal to or less than your EUM Server version.

Enable and Disable Browser RUM

At the top of the **Configuration** > **Instrumentation** page, toggle the **Browser Monitoring**:



To disable Browser RUM, toggle the switch to **OFF**.

Inject the JavaScript Agent for Browser RUM into Your Application Pages

The JavaScript Agent for Browser RUM collects Browser RUM metrics. See Browser RUM Metrics for more detail.

The JavaScript Agent for Browser RUM must be inserted into the headers of the pages for which you want to see these metrics. This process is called *injection*. There are several ways to accomplish this. See Inject the JavaScript Agent.

External Access Locations

Browser RUM is made up of several components, which, in various configurations, can either be located on the Internet or hosted inside your own data center/network. On-premises access points are configured at installation or through the UI. If your installation requires access to any of these components on the Internet, see Access the SaaS EUM Server for more information.

If you are using the AppDynamics CDN to fetch the JavaScript Agent (`adrum.js`) and JavaScript Agent extension (`adrum-ext.js`), you also need to make sure that `cdn.appdynamics.com` is accessible from your network.

Additional Browser RUM Configurations

You can also configure:

- Page Identification and Naming
- JavaScript and Ajax Error Detection
- Browser Snapshot Thresholds
- Browser Snapshot Collection Rules

Correlate Business Transactions for Browser RUM

Related pages:

- Correlate Business Transactions for EUM
- Business Transactions

You can correlate page and Ajax requests with business transactions. The correlation is made between instances of page/Ajax requests (browser snapshots) and instances of business transactions (transaction snapshots).

How Browser Application Network Requests Are Correlated

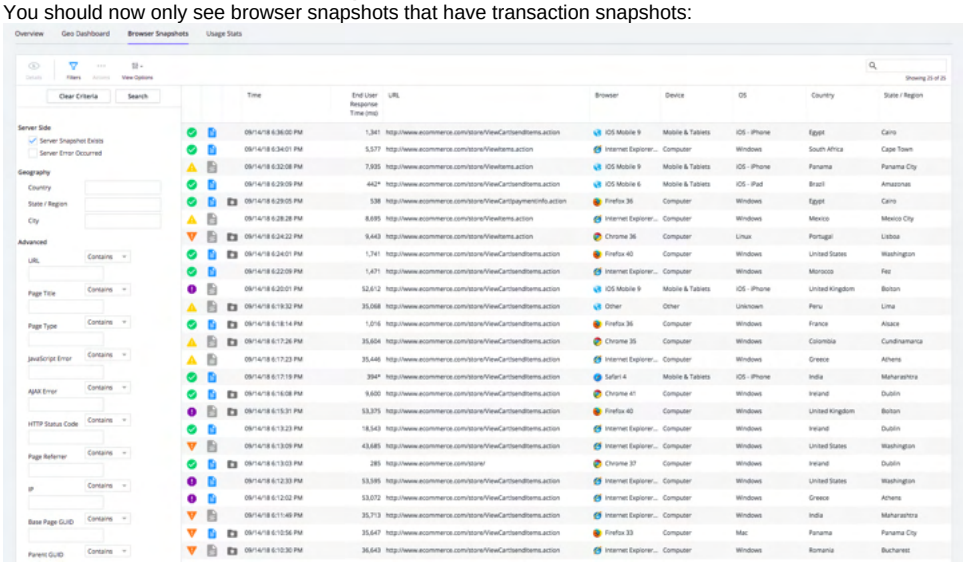
When an end user requests a page from your browser app:

1. The App Agent:
    - Sends HTTP headers identifying the business transaction and the HTML with the injected JavaScript Agent to the end user's browser.
    - Aggregates backend metrics and sends them along with the business transaction identifiers to the Controller. This serves as the content for the transaction snapshot.
2. The JavaScript Agent sends browser metrics and the business transaction identifiers (from the HTTP header) to the EUM Server. This serves as the content for the browser snapshot.
3. The Controller fetches the metrics and business transaction identifiers from the EUM Server. These business transaction identifiers are then used to correlate the browser snapshot with the transaction snapshots.

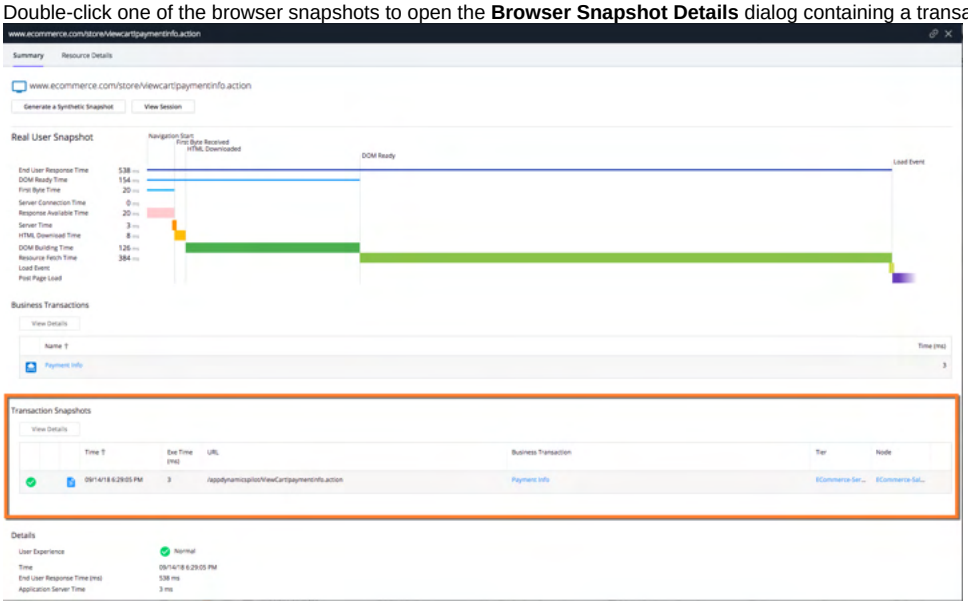
View Business Transactions Correlated with Browser Applications

There are several ways to navigate from a browser snapshot to its correlated business transaction. The following steps show you one possible way.

1. From the **Browser App Dashboard**, click **Browser Snapshots**.
2. Click **Filters** and check the **Server Snapshot Exists** checkbox.
3. You should now only see browser snapshots that have transaction snapshots:



4. Double-click one of the browser snapshots to open the **Browser Snapshot Details** dialog containing a transaction snapshot.



5. You can then click links in the transaction snapshot or transaction snapshot itself to view corresponding pages in APM.

Get Complete Timing Data for Correlated Business Transactions

> This functionality is currently not supported for .NET Core.

To get the full real execution time for correlated business transactions, your injection method may need to write the `JS_FOOTER` variable to your page. Manual injection gives the server-side agent the ability to write data only to the header of the page as it is being constructed by your web application.

It is possible that complete business-transaction timing information is not available at the moment that the header data is written. Using the footer allows the server-side agent to write timing data at the footer of the page, by which time a fuller picture of business transaction timing may be available.

You can write the `JS_FOOTER` data variable into the footer of a web page using the following techniques:

- If you use automatic injection for the injecting into the `head` element, you automatically get an injection into the footer as well.
- If you use manual injection for the `head` element, for applications built on Java platforms you can use assisted injection to inject into the footer. Or for applications built on Java servlet or ASP.NET platforms, you can use assisted injection using attribute injection.

If you cannot add the `JS_FOOTER` variable to your page, the timing shown for correlated business transactions may be the average response time for that transaction rather than the real execution time for that specific page.

Avoid Tagging Cookies with the HttpOnly Flag

Servers set the `HttpOnly` flag on cookies to prevent their contents from being accessed by JavaScript. This is often done for session cookies to hide the session identifier as a security measure. The JavaScript Agent, however, needs to be able to read special cookies set by the AppDynamics server-side agent (all prefixed with `ADRUM`) to collect correlation information. If `HttpOnly` is set on these cookies, no server-side correlation information can be transmitted. Thus, make sure that your server does *not* set the `HttpOnly` flag on any cookies prefixed with `ADRUM`.

If you want to securely transmit cookies, use HTTPS. The app agent sets the `secure` flag if the originating base page is loaded over HTTPS.

These can be turned off, but that would prevent correlation from working for base pages unless the customer changes the page source code to use footer injection.

Configure the JavaScript Agent

Related pages:

- Injection Problems
- Inject the JavaScript Agent

This page describes configuration and hosting options for the JavaScript Agent.

Configure the JavaScript Agent

To configure the JavaScript Agent in the Controller:

1. Go to a browser application.
2. In the left-hand panel, go to **Configuration** > **Configure the JavaScript Agent**.
3. Select a hosting option.
4. Customize the code snippet in **Step 2: Advanced settings (Optional)**.
5. Click **Save**.

Advanced Configurations

This table describes advanced configurations for the JavaScript Agent snippet. You can add these advanced configurations manually or using Controller >=21.7.0. When you check a box in the UI, such as **Use HTTPS by default**, the JavaScript snippet automatically updates to reflect the selection, such as c `onfig.useHTTPSAlways = true` added to the UI snippet.

Available in SaaS Controller >=21.7.0	Manual Configuration Instructions	Summary
✓	Use HTTPS by default	Use HTTPS as the default transport protocol for the JavaScript Agent. If unchecked, the agent chooses the transport protocol used to load the base page.
✓	Set cookie to strict domain	Limit the cookie to the full strict domain name. If unchecked, the cookie is set to the broadest version of the originating domain (such as, `*.domain.com`).
✓	Set GeoServer URL	If your application uses internet IP addresses or users connect through a VPN, enter your custom GeoServer URL to map internal IP addresses to physical locations. If unchecked, end-user locations are resolved using the default GeoServer.
✓	Configure URL length	Set the maximum number of segments and characters in resource URLs to view more or less of a URL in the Controller UI.
✓	Hide URL query string	By default, URL query strings may be long or contain sensitive information. Check this box to hide the full string from the Controller UI.
✓	Cross-domain session correlation	Enable sessions to continue across subdomains. If unchecked, sessions are restricted to one domain and navigating from one domain to another will end the session.
✓	Configure request origin location	Set a specific IP address or location as the origin of the request. You can specify only the IP address, or the complete location (IP, country, region, and city).
✓	Configure resource options	Configure which resources are monitored based on how they are ordered, the maximum number of resources that will be evaluated, and whether the resource timing buffer should be cleared when it's full.
✓	Modify resource sampling options	Modify how the resources are ordered (using the sampling algorithm) and the maximum number of resources to be evaluate.
✓	Capture page title	Capture the page title as part of the beacon, displayed in **Browser Snapshots**. Uncheck this setting to hide the page title for security or privacy reasons. Check this setting to set a custom page title with an arbitrary URL string.
✓	Enable SPA2 monitoring	Monitor single-page applications (SPAs) to detect virtual pages and correlate resources, JavaScript errors, and AJAX requests.
✓	Enable resource timing buffer clearing for SPAs	For SPAs, the JavaScript Agent by default clears the resource timing buffer after it is full and saves the data in a local buffer. You can configure the JavaScript Agent so that the resource timing buffer is not cleared to capture resource timing data.
✓	Set the custom page name	The page name is used to identify and group pages in **Pages & AJAX Requests**. If unchecked, the default page name consists of the hostname, port, and path.
✓	Filter virtual pages	Configure rules to exclude specific virtual pages from being monitored.

✓	Monitor Fetch API calls	By default, the JavaScript Agent monitors Fetch API calls for all SPAs except applications using Zone libraries, including Angular applications. Only consider disabling this setting for specific use cases.
✓	Set AJAX request limit	To prevent an overload of AJAX requests, you can limit the number of requests sent per base and virtual pageview. The limit is 250 requests for single-page applications (SPAs) and 50 requests for non-SPAs.
✓	Filter XHR calls by URL	Include or exclude specific XHR calls to be monitored.
	Report events with the JavaScript API	The JavaScript API enables you to manually report events to the agent so that it can time the various parts of your virtual page loads and correlate Ajax calls to those page loads. You can also capture and report errors with this API.
	Limit beacon types	By default, sending image beacons is disabled and beacons are only sent with Cross-Origin Resource Sharing (CORS). If you have an older browser that does not support CORS beacons, you can enable sending image beacons.
	Disable browser monitoring programmatically	For pages in which the JavaScript Agent was injected manually, you can disable the agent programmatically by adding a script to the header.
	Handle the window.onerror event	If any script on your monitored web pages or library code sets the JavaScript `window.onerror` event, add the method `ADRUM.listenForErrors()`to the page immediately after setting `window.onerror`.
	Set Ajax request names based on captured POST parameters	You can configure the JavaScript Agent to capture `POST` parameters and then use the parameter(s) to name the Ajax request in Pages & Ajax Requests. This configuration enables you to identify and sort Ajax requests from the same page based on `POST` parameter(s).
	Set custom virtual page names	You can configure the JavaScript Agent to use any arbitrary string (that is not necessarily a part of the URL) to name a virtual page.
	Add custom user data to a page browser snapshot	You can add user information that is specific to your application to a browser snapshot. The information is expressed as key-value pairs that are attached the JavaScript Agent configuration and later included in the beacons sent to the EUM Server.

Configure Shared-Hosting

If you are using shared-hosting and do not want to use the default CDN, follow these instructions to provide your own CDN for only hosting the main JavaScript Agent file (`adrum-[version].js`):

1. From **Controller Admin,** go to **Controller Settings**.
2. For the values of settings `eum.jsagent.cdn.host.http` and `eum.jsagent.cdn.host.https`, enter the URLs where the JavaScript Agent files are being hosted.
3. Confirm these URLs match the URLs in the JavaScript snippet you want to inject.

Configure Self-Hosting

If you are self-hosting and do not want to use the default CDN, follow these instructions to provide your own CDN for hosting all of the JavaScript Agent files:

1. From the **Controller Admin,** go to **Controller Settings**.
2. For the values of settings `eum.jsagent.cdn.host.http` and `eum.jsagent.cdn.host.https`, enter the URLs where the JavaScript Agent files are being hosted.
3. Confirm these URLs match the URLs in the JavaScript snippet you want to inject.

Next Steps

After you configure the JavaScript Agent, inject the JavaScript snippet in your page HTML with one of these methods:

- Manual Injection of the JavaScript Agent
- Automatic Injection of the JavaScript Agent
- Assisted Injection

See Overview of Injection Types for differences between the injection methods.

Add Custom User Data to a Page Browser Snapshot

Related pages:

- Page Browser Snapshots

You can add user information that is specific to your application to a browser snapshot. The information is expressed as key-value pairs that are attached the JavaScript Agent configuration and later included in the beacons sent to the EUM Server.

The JavaScript Agent initializes user data differently depending on the page type (base, Ajax, virtual) and the method used (immediately invoked function expressions, function pointers, and literals). See Methods for Setting Custom User Data to learn when the JavaScript Agent initializes the user data for each page type and method.

The maximum size allowed for user data in a page is 2048 characters (CORS beacon) or 100 bytes (image beacon). The maximum size includes the keys-value pairs, syntax characters such as braces, and quotation marks.

View Custom User Data

Custom user data appears in **Browser Analyze**, **Browser Snapshots**, and **Sessions**. View the following tabs to learn more about how user data is viewed and used in the Controller UI.

Methods for Setting Custom User Data

You can use several methods to set custom data for each page type (base, Ajax, virtual). Each method has its own syntax, execution time, and use case. This table outlines the execution time and potential use case for each method and page type.

Method Type	Page Type	Execution Time	Potential Use Cases
Immediately Invoked Function Expressions (IIFE)	Base	These are JavaScript functions that run as soon as they are defined. They're also known as self-executing anonymous functions.	If your base/virtual page or Ajax request depends on information from different resources, you can set static information for custom user data with IIFE. For example, if two different server scripts generate content for the base page, you could use the IIFE on the client to set the static user data.
	Ajax		
	Virtual		
Function pointers	Base	The JavaScript Agent executes function pointers when the `onload` event is triggered.	Data extracted from cookies, the page, and meta data.
	Ajax	This event is triggered when an Ajax call is made. Custom user data for the Ajax events are attached to AJAX requests.	Meta data regarding the Ajax request such as the URL, HTTP method, or the request payload.
	Virtual	This event is triggered when the virtual page is created. The virtual page is a dynamically recreated version of the base page, and custom user data set for `VPageView` events are attached to the virtual page records.	Use if the information is derived or found somewhere on the page because of the creation of the virtual page. User-specific fields or user data set based on the URL and DOM changes.
Literals	Base	You can simply use literal values to set custom data. The values, as with IIFE, are set as soon as the values are defined.	Constants, static data extracted and set on the server.
	Ajax		
	Virtual		

User Data Types

For each event callback that is triggered, you can add user data by returning objects containing key-value pairs of different data types. Each user data type is an object containing properties of a specific data type.

⚠ Failing to send the wrong type of data as values to the keys inside the `userData` object will throw a generic error, so use with caution.

Data Type	Description	Example

`userData`	This data type is for setting one or more properties of type `string`. The value must be in a string, even if the value is set to a numeric value.	`{`
		`  user: "john_thompson",`
		`  cart: "100.00"`
		`};`

`userDataLong`	This data type is for setting one or more properties of type `long` and must be a numeric value without decimals.	`{ numberOfProducts: 17, numberOfSales: 1213 };`
`userDataDouble`	This data type is for setting one or more properties of type `double` and must be a numeric value with decimals.	`{ monthlyVisitFrequency: 0.13333333, avgCustomerConversion: 0.0361922 };`
`userDataBoolean`	This data type is for setting one or more properties of type boolean.	`{ returnCustomer: true, subscriber: false };`
`userDataDate`	This data type is for setting one or more properties of type `date`. The value must be the converted Epoch value of the date.	`{ epoch1: 1448675019877, epoch2: 1418475982737 };`

Syntax of User Data Objects

You add user data as objects for each page event to the JavaScript Agent configuration. The base page, Ajax, and virtual pages have the following corresponding user data objects:

- `PageView`
- `Ajax`
- `VPageView`

The `PageView`, `Ajax`, and `VPageView` objects have slightly different syntaxes. `UserPageName` is only available for `PageView` and `VPageView`, and not for `Ajax` and `Error`.

Capture Ajax Data for Setting User Data

When setting user data for the Ajax event, the JavaScript Agent configuration provides a `context` object that has properties for the HTTP method, the request URL, and the request payload of the Ajax request. You can use this information to set values for the user data configuration object.

The JavaScript Agent does not intercept or expose the response from the Ajax request. Thus, you can only include the request information (HTTP method, request URL, and request payload) in the user data that you report.

Ajax Context Object

To access the URL and HTTP method of the Ajax request, you can simply access the `method` and `url` properties of the `context` object. For the request payload of the Ajax request, you need to first match the payload parameters (HTTP method or Ajax URL) to access the `data` property. See Match the Ajax Payload Parameters to learn how to define the filters to match the payload parameters.

The `context` object has the following properties:

Property	Description	Data Type	Requirement to Access Property
`method`	The HTTP method used to make the Ajax request.	string	XHR call

`url`	The URL used to make the Ajax request.	string	XHR call
`data`	The request payload attached to the Ajax request.	Any data type that can be passed as the body to xhr.send.	• XHR call • XHR filter for the payload parameter

Match the Ajax Payload Parameters

To access the request payload, you need to use `xhr.payloadParams` array to match the HTTP method and/or the Ajax URL. To match the Ajax URL, you can specify one or more patterns. To match the HTTP method, you include one or more objects specifying HTTP methods.

In the "Match HTTP Methods" example, the `payloadParams` array is used to match the HTTP methods "POST" and "GET". In the "Matching URLs" example, the `payloadParams` array is used to match URLs with the following string: `'.*/xhr/uptime'`.

Ajax Payload Filter Examples

Conventional Ajax Requests

In the following examples, the data from a conventional Ajax request payload is used to set custom user data. For capturing the request payload for Fetch API calls, see Ajax Requests Using the Fetch API.

Ajax Requests Using the Fetch API

The example below demonstrates how to use the JavaScript Agent configuration to match the HTTP method, the request URL, and the request body for Ajax calls using the Fetch API. The Fetch API call is also provided to add context. The JavaScript Agent **cannot access** the request body, however, if you supply your own Request object to the `fetch` method.

Custom User Data Examples

The following examples show you how to set custom data for different page types and using the different methods:

- Setting User Data with Function Pointers
- Setting User Data with an Anonymous Function
- Setting User Data with Multiple Methods

Setting User Data with Function Pointers

The example below assigns a value to the `userPageName` property of the `PageView` object by executing the function `extractUserData`, which extracts data from the cookies.

```
< script type = 'text/javascript'
charset = 'UTF-8' >
        (function (config) {
                (function (info) {
                        info.PageView = extractUserData;
                })(config.userEventInfo || (config.userEventInfo = {}))
        })(window['adrum-config'] || (window['adrum-config'] = {}));

function extractUserData() {
        var cookies = document.cookie.split(';');
        for (var i = 0; i < cookies.length; i++) {
                var cookie = cookies[i].trim();
                if (cookie.indexOf("role=") === 0) {
                        var role = cookie.substring(5);
                }
        }
        return {
                userPageName: role
        };
} <
/script> <
script src = '//cdn.appdynamics.com/adrum/adrum-latest.js'
type = 'text/javascript'
charset = 'UTF-8' > < /script>
```

Setting User Data with an Anonymous Function

You can also use anonymous functions that return an object as shown in this example for setting user data for virtual pages.

```
< script type = 'text/javascript'
charset = 'UTF-8' >
        (function (config) {
                (function (info) {
                        info.VPageView = function () {
                                return {
                                        userData: {
                                                version: window.location.href.split("/")[4],
                                                space: space(),
                                                email: getEmail()
                                        },
                                        userPageName: $('title').get(0).text,
                                        userDataDate: {
                                                currentTime: ((new Date()).getTime())
                                        },
                                        userDataBoolean: {
                                                watchingPage: watchingPage()
                                        }
                                }
                        }
                })(config.userEventInfo || (config.userEventInfo = {}))
        })(window['adrum-config'] || (window['adrum-config'] = {})); <
/script> <
script src = '//cdn.appdynamics.com/adrum/adrum-latest.js'
type = 'text/javascript'
charset = 'UTF-8' > < /script>
```

Setting User Data with Multiple Methods

You might also want to use a combination of literal values, named and anonymous functions as this example does for setting user data for the `Ajax` event. Note, the `context` object is only available for `Ajax` events, and this object has the properties `data` (stores the request payload), `method` (HTTP method used to make Ajax request), and `url` (the Ajax request URL).

```
<script type='text/javascript' charset='UTF-8'>
    (function(config){
        (function(info) {
            info.Ajax = function(context) {
                return {
                    userPageName: $('title').get(0).text,
                    userData: extractUserData(context)
                    userDataBoolean: {
                                    "visited": true
                    }
                }
            };
        })(config.userEventInfo || (config.userEventInfo = {}))
   })(window['adrum-config'] || (window['adrum-config'] = {}));

   function extractUserData(context) {
        var cookies = document.cookie.split(';');
        for (var i = 0; i < cookies.length; i++) {
            var cookie = cookies[i].trim();
            if (cookie.indexOf("email=") === 0) {
                var role = cookie.substring(5);
            }
        }
        return {
            role: role,
            url: context.url,
            method: context.method
        };
    }
</script>
<script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
```

Set Custom Page Names

Related pages:

- Configure Naming for Ajax Requests
- Configure Virtual Page Naming

This page describes how to set custom page names. In the **Configuration > Instrumentation** page, you can configure rules that name pages, iframes and Ajax requests based on various parts of the page URL. See Configure Page Identification and Naming.

You can also configure the JavaScript Agent to use any arbitrary string, not necessarily a part of the URL, to name a page or an iframe, but not an Ajax request. To do so, you assign a string of 760 characters or less to the `userPageName` property of the `PageView` object. If the string length exceeds 760 characters, the page name will not be set.

For example, this configuration shown below would set the page name to "My Custom Page". That page name would then be used to identify and group pages in the **Pages & AJAX Requests** page in the Controller UI.

```
<head>
    <script type='text/javascript' charset='UTF-8'>
        (function (config) {
            (function (info) {
                info.PageView = {
                    userPageName: "My Custom Page"
                }
            })(config.userEventInfo || (config.userEventInfo = {}));
        })(window['adrum-config'] || (window['adrum-config'] = {}));
    </script>
    <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
    ....
</head>
```

The default page name consists of the hostname, port, and path. For example, if the page URL is `http://example.com:8080/yourpath`, the default page name would be `example.com:8080/yourpath`.

Set Custom Virtual Page Names

Related pages:

- Configure Page Identification and Naming
- Configure Naming for Ajax Requests
- Configure Virtual Page Naming

This page explains how to create custom virtual page names.

To name virtual pages with the JavaScript Agent, you need to enable SPA2 monitoring. You can configure the JavaScript Agent to use any arbitrary string, not necessarily a part of the URL, to name a virtual page. The virtual page name must consist of a string of 760 or fewer alphanumeric characters. If the string length exceeds 760 characters, the page name will not be set.

If you are using SPA1 monitoring, the method called to set the name the virtual page will be ignored.

To name a virtual page, you execute the method `setVirtualPageName` with `ADRUM.command` as shown below. The `ADRUM` object is globally accessible after you load the JavaScript Agent (`adrum-latest.js`).

```
<head>
    <script charset='UTF-8' type='text/javascript'>
        window['adrum-start-time'] = new Date().getTime();
        (function(config){
            config.appKey = '<EUM_APP_KEY>';
            ...
        })(window['adrum-config'] || (window['adrum-config'] = {}));
    </script>
    <script charset='UTF-8' src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' ></script>
    <script type='text/javascript' charset='UTF-8'>
        ...
        ADRUM.command("setVirtualPageName", "myCustomVPName");
    </script>
</head>
```

When you name a virtual page with `setVirtualPageName`, the custom name will be applied to the next virtual page you manually mark with `markVirtualPageBegin`. Thus, the virtual page currently being monitored will not get the custom name.

The code example below shows how to call the methods to manually mark the beginning and end of the virtual page as well as execute `setVirtualPageName` to name the next virtual page that is monitored. See Report Virtual Pages to learn how to manually mark the beginning and end of virtual pages with the JavaScript API.

```
.controller('VPNaming', ['$scope', '$http', function ($scope, $http) {
            $scope.startVirtualPageWithCustomUrl = function () {
                console.log("Marking the beginning of the virtual page and waiting for the end to be manually
marked.");
                ADRUM.markVirtualPageBegin("homepage", true);
            }
            $scope.startVirtualPageNotWaitingForMarkVirtualPageEnd = function () {
                console.log("Marking the beginning of the virtual page and allowing the JS Agent to mark the
end of the virtual page.");
                ADRUM.markVirtualPageBegin("homepage", false);
            }
            $scope.endVirtualPage = function () {
                console.log("Manually marking the end of the virtual page.");
                ADRUM.markVirtualPageEnd();
            }
            $scope.setVirtualPageName = function () {
                console.log("Setting a custom name for the next virtual page to be monitored: have it's
beginning and end marked.");
                ADRUM.command("setVirtualPageName", "myCustomVPName");
            }
        }
    ]
);
```

Set Ajax Request Names Based on Captured POST Parameters

Related pages:

- Configure Virtual Page Naming
- Page, Ajax, and Iframe Dashboards
- Set Custom Page Names

You can configure the JavaScript Agent to capture POST parameters and then use the parameter(s) to name the Ajax request in the **Pages & Ajax Requests** page. This enables you to identify and sort Ajax requests from the same page based on POST parameter(s).

For example, customers on your home page of your website can either register an account or log in. The Ajax request may pass the parameter `action` to reflect one of these user actions. By capturing the `action` parameter, you can differentiate the Ajax requests to monitor performance and debug issues.

Configure the JavaScript Agent to Capture Parameters

You use the `xhr` object to configure the JavaScript Agent to capture POST parameters. The `xhr` object has the property `parameter` that is used to match resource URLs and define a callback for parsing the request body and return the desired results.

In the example below, the configuration sets a pattern to match the URL `http://www.mystore.com`, parses the request body, and returns an object containing the parameter `action`. As mentioned earlier, you could use a configuration like this to differentiate Ajax requests that are sending requests to register or log in users.

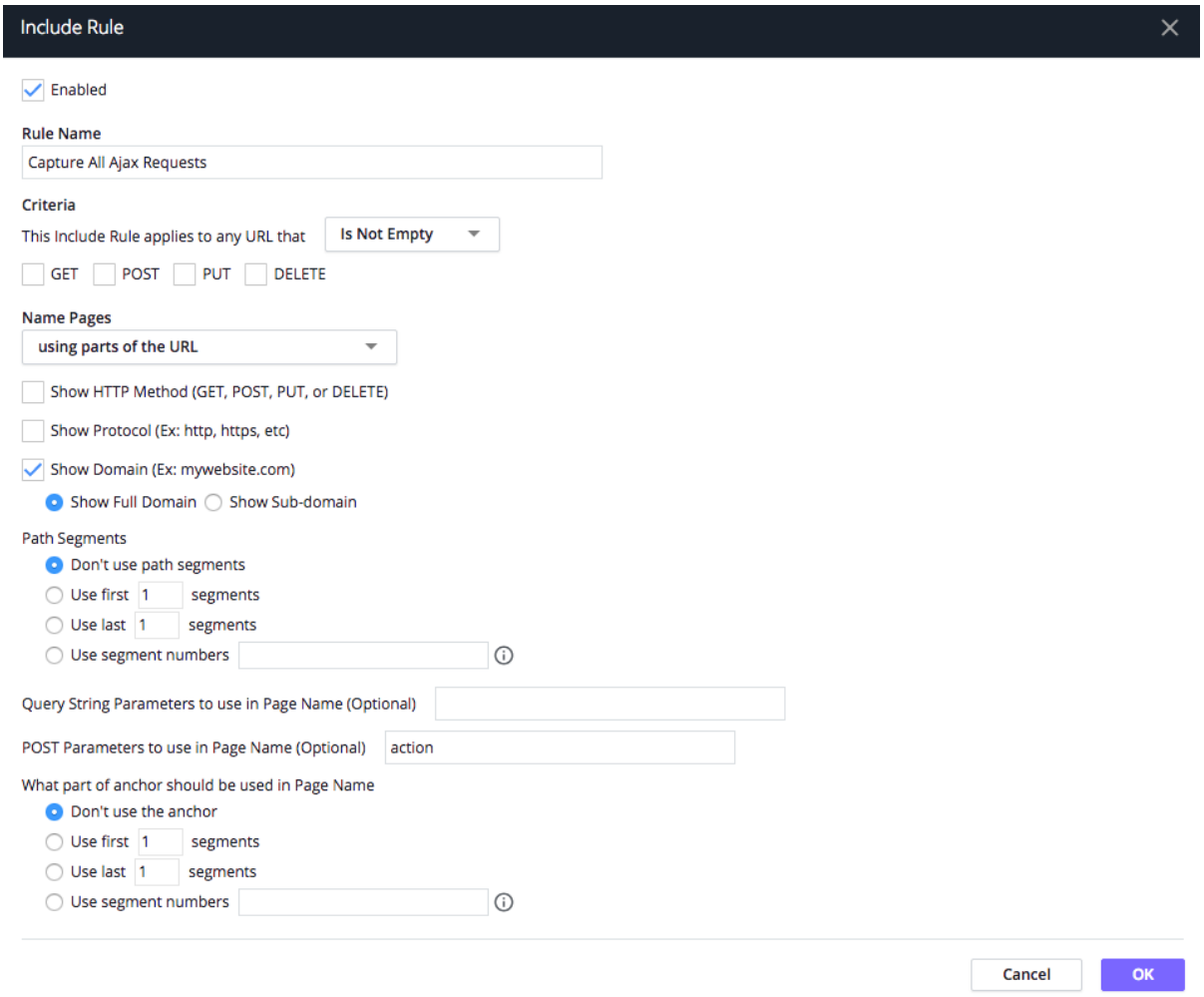
```
<script type='text/javascript' charset='UTF-8'>
    window['adrum-start-time'] = new Date().getTime();
    window['adrum-config'] =  {
        xhr: {
            parameter : {
                urls : [{pattern : '^https?:\\/\\/www\\.mystore\\.com'}],
                getFromBody: function(data) {
                    if (typeof data === 'string') {
                        var fields = data.split("&");
                        for (var i = 0; i < fields.length; i++) {
                            var keyAndValue = fields[i].split('=');
                            if (keyAndValue.length > 1) {
                                var key = keyAndValue[0],
                                value = keyAndValue[1];
                            if (key === 'action')
                                return {action: value};
                            }
                        }
                    }
                }
            }
        }
    }
</script>
<script charset='UTF-8'>
    (function(config){
        config.appKey = '<EUM_APP_KEY>';
        config.adrumExtUrlHttp = 'http://cdn.appdynamics.com';
        config.adrumExtUrlHttps = 'https://cdn.appdynamics.com';
        config.beaconUrlHttp = 'http://col.eum-appdynamics.com';
        config.beaconUrlHttps = 'https://col.eum-appdynamics.com';
        config.xd = {enable : true};
    })(window['adrum-config'] || (window['adrum-config'] = {}));

    (function (cfg) {
        if (cfg.beacon) cfg.beacon.neverSendImageBeacon = true;
        else cfg.beacon = { neverSendImageBeacon: true };
    })(window['adrum-config'] || (window['adrum-config'] = {}));
</script>
<script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
```

Create an Ajax Include Rule

To name an Ajax request based on a captured request parameter, you need to define an include rule that specifies the captured POST parameter. See Configure Naming for Ajax Requests to learn how to create include rules for Ajax requests.

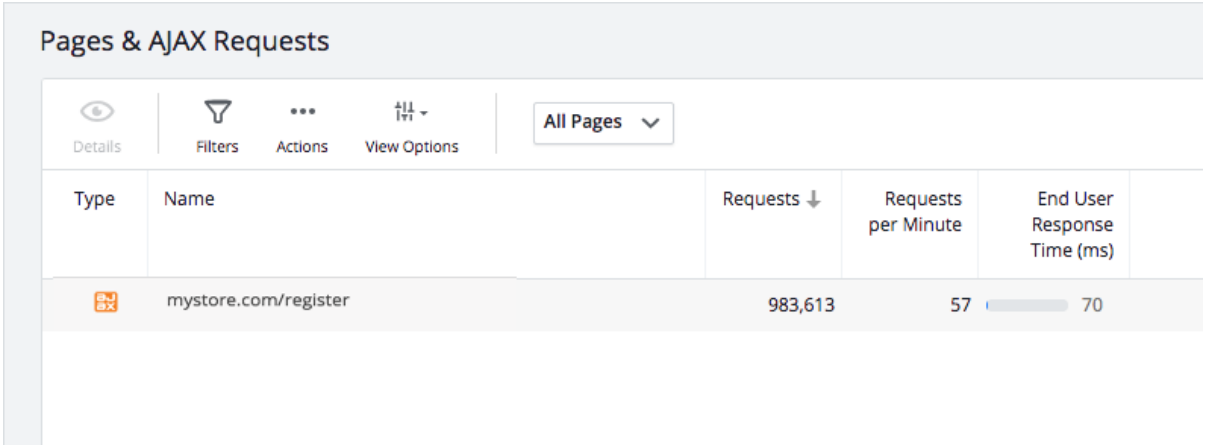
The example include rule below uses the full domain and the captured POST parameter to name the Ajax request in the **Pages & Ajax Requests** page. For example, if an Ajax request is made to `www.mystore.com` and the value of the `action` parameter is `register`, the Ajax request will be named `www.mystore.com/register`.



View Results in Pages & Ajax Requests

From the **Pages & Ajax Requests** page, you can view the Ajax requests that are named based on your include rule.

Using the configuration and include rule shown above, you might see the following Ajax request with the name **mystore.com/register.**



Handle the window.onerror Event

Related pages:

- JavaScript Errors
- Configure JavaScript and Ajax Error Detection
- Visualize JavaScript Errors

If any script on your monitored web pages, including library code, sets the JavaScript `window.onerror` event, add the following method to the page immediately after setting `window.onerror`:

```
<script>
ADRUM.listenForErrors()
</script>
```

The JavaScript Agent (`adrum.js`) sets `window.onerror` to listen for uncaught JavaScript errors. If this listener is overwritten, errors are not reported.

The agent invokes your original `onerror` handler.

Disable Browser Monitoring Programmatically

Related pages:

- Enable and Disable Browser RUM

For pages in which the JavaScript Agent was injected manually, you can disable the agent programmatically by adding a script to the header.

To disable Browser Monitoring add the snippet below before `adrum.js` agent is injected.

```
window["adrum-disable"] = true
```

For example:

```
<head>
    <script type='text/javascript' charset='UTF-8'>
        // before adrum.js
        window["adrum-disable"] = true
    </script>

    <script>
        window["adrum-start-time"] = new Date().getTime();
    </script>
     <!-- adrum.js injection -->
    <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
    ...
</head>
```

Old Browser Monitoring data is preserved, but no new monitoring data is collected while the agent is disabled.

You can re-enable monitoring by removing the disable script statement.

Set the Exact Current Domain in the JavaScript Agent Cookie

This page describes how to set the current domain in the JavaScript Agent cookie.

The JavaScript Agent itself writes a session cookie to the page, for timing purposes. This cookie is set when the user clicks a link and the `unload` event is fired. By default, the cookie is set to the broadest possible version of the originating domain (such as, `*.domain.com`) to increase the likelihood that the next page opened in the same domain can access that cookie. See Cookies and Browser Monitoring Data.

In some cases, however, it may be necessary to limit the cookie to the full exact domain name. To do this, add a flag in the `"adrum-start-time"` script to the header of each page right *after* the `<head>` tag and *before* the entry that includes the location of the agent (`adrum.js`). The flag should read: `windo w["adrum-use-strict-domain-cookies"] = true`.

After you include the start-time line, the strict domain flag, and the agent, the `<head>` section in your monitored web pages should look something like this:

```
<head>
    <script type='text/javascript' charset='UTF-8'>>
        window["adrum-start-time"] = new Date().getTime();
        window["adrum-use-strict-domain-cookies"] = true;
    </script>
    <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
    ...
</head>
```

Limit Beacon Types

Related pages:

- Alter or Eliminate the Page Title Captured in the Beacon

The JavaScript Agent collects and sends information to the EUM Server via a beacon, either through Cross-Origin Resource Sharing (CORS) or an image beacon which returns a small, transparent `.gif` file.

If you want the agent to only use CORS to transport the beacon, you can turn off the image beacon request mechanism. To turn off the image beacon request, add the following snippet to your HTML preceding the injected `adrum.js` script:

```
<head>
    <script type='text/javascript' charset='UTF-8'>
       (function (cfg) {
           if (cfg.beacon) cfg.beacon.neverSendImageBeacon = true;
           else cfg.beacon = { neverSendImageBeacon: true };
        })(window['adrum-config'] || (window['adrum-config'] = {}));
    </script>
    <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
    ...
</head>
```

Alter or Eliminate the Page Title Captured in the Beacon

Related pages:

- Limit Beacon Types
- Set Custom Page Names

In a standard Browser RUM beacon, the value for `document.title` is collected as part of the data to be sent in the beacon. You may want to alter or eliminate the page title for security or privacy reasons.

You can choose to:

- Not collect a page title at all
- Use a title that is created by a function with 0 arguments
- Use a title that is an arbitrary string

The page title is different than the page name. The page title is just additional information included in browser snapshots, whereas, the page name is used to identify and group records in the **Pages & AJAX Requests** page.

Remove a Page Title

To remove the page title entirely, add the following snippet to your page before you inject the `adrum.js` script:

```
<head>
    <script type='text/javascript' charset='UTF-8'>
        (function (config) {
           (function (page) {
               page.captureTitle = false;
           })(config.page || (config.page = {}));
        })(window['adrum-config'] || (window['adrum-config'] = {}));
   </script>
   <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
   ...
</head>
```

Use a Page Title Created by a Function

To use a function with no arguments to create a page title, before you inject the `adrum.js` script, define your function and call it, as in this example:

```
<head>
    <script type='text/javascript' charset='UTF-8'>
        function title() { return document.title.split('-').slice(1,3).join('-'); } // define a function
        (function (config) {
            (function (page) {
                page.title = title; // call your function
            })(config.page || (config.page = {}));
        })(window['adrum-config'] || (window['adrum-config'] = {}));
    </script>
    <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'/>
    ...
</head>
```

Something like this could be used, for example, to remove sensitive data from the page title.

Use an Arbitrary String as a Page Title

To use any arbitrary string as the page title, before you inject the `adrum.js` script, set `page.title`.

```
<head>
    <script type='text/javascript' charset='UTF-8'>
        (function (config) {
           (function (page) {
               page.title = "My Special Page Title";
           })(config.page || (config.page = {}));
        })(window['adrum-config'] || (window['adrum-config'] = {}));
    </script>
    <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
    ...
</head>
```

This would set the page title in the beacon to "My Special Page Title".

Modify Resource Sampling Options

Related pages:

- Page Browser Snapshots
- Capture Resource Timing Data Without Loss

Some pages use a very large number of resource files, more than can usefully be monitored. By default, the JavaScript Agent decides which resource files to monitor based on two factors:

- How the resources are ordered (the sampling algorithm used)
- The maximum number of resources to be evaluated

Both of these factors can be modified.

To modify the sampling algorithm, use `resTiming.sampler`. Possible values are:

- Top N: the resources that take the most time to load, up to the maximum number
- First N: the resources that load first, up to the maximum number
- Relevant N: the resources that are deemed "most relevant" by an algorithm that takes into account both overall load time and when in the sequence the resource loads. So, for example, something that takes a long time to load and is early in the load sequence is "more relevant" than something that takes exactly the same time to load, but does so later in the sequence, on the assumption that the former would have more impact on the overall user experience. This is the default.

To modify the maximum number of resources to be evaluated, use `resTiming.maxNum.`

For example, to sample based on Top N and to set the max number at 100, you could add the following snippet to your page, *before* you inject the `adrum.js` script.

```
<head>
    <script type='text/javascript' charset='UTF-8'>
        (function (config) {
            config.resTiming = {
                sampler: "TopN",
                maxNum: 100
            };
        })(window['adrum-config'] || (window['adrum-config'] = {}));
    </script>
    <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
    ...
</head>
```

Set the Origin Location of the Request

Related pages:

- Geo Dashboard

There are some cases where it would be useful to be able to set a specific IP address/location as the origin of the request by modifying the JavaScript Agent itself. Add the following snippet to your page, before you inject the `adrum.js` script:

```
<head>
    <script type='text/javascript' charset='UTF-8'>
        window['adrum-config'] = (function(config) {
            config.geo = {
                localIP: "192.168.200.255",
                city: "San Francisco",
                region: "California",
                country: "USA"
            };
            return config;
        })(window['adrum-config'] || {});
    </script>
   <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
</head>
```

You can use this snippet in two ways:

- Specify only the IP address, in which case this IP address has the highest precedence for use by the geo server, local or cloud-based, in resolving the request location.
- Specify all of IP, `country`, `region`, and `city`, in which case the request location is set by the script, and is not resolved by the Geo Server.

If you specify only some of the location parameters, the entire configuration is ignored. For example, if you only specified country, but not region, the entire configuration is ignored.

The value for `city` is only displayed in snapshots and can be any string. Using this value it is possible to specify location more precisely, as desired, for example, `San Francisco, Sutter Street.`

Hide All or Parts of the URL Query String

Related pages:

- Configure the Number and Length of URL Segments

By default, we capture URLs with their query strings. Because query strings may result in very long URLs or contain information that you don't want to expose, you may want to prevent all or parts of the query string from being displayed in the Controller UI.

You can do this by configuring the JavaScript Agent to remove query strings from URLs for the following:

- pages
- virtual pages
- XHR calls
- referrers
- scripts
- resources

Filter for URL Query Strings

The filter `filterURLQuery` is used to remove query strings from URLs. You can use it to remove all query strings or specific key-value pairs from query strings. The table below specifies the supported values and describes the result of the filter.

Filter	Value	Default	Description
`filterURLQuery`	`true`	No	Removes query strings from all URLs.
`filterURLQuery`	`false`	Yes	Retains the query strings for all URLs.
`filterURLQuery`	array of strings	No	Each string represents a key in the query string. The given keys are removed from the query string. Strings that do not match keys in the query string will have no effect on the filtering.

Unsupported values will result in the default behavior of retaining the entire query string for all URLs.

Remove All Query Strings

The configuration below will remove all URL query strings, so you'll only see the domain name and the URL path in the Controller UI.

```
<head>
    <script type='text/javascript' charset='UTF-8'>
       (function (config) {
           config.urlCapture = {
               filterURLQuery: true
           }
       })(window['adrum-config'] || (window['adrum-config'] = {}));
    </script>
    <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
    ....
</head>
```

Remove Specific Keys From Query Strings

This configuration will remove the key-value pairs from the query string where the keys match the strings `"name"`, `"page"`, and `"id"` given in the array.

```
<head>
    <script type='text/javascript' charset='UTF-8'>
        (function (config) {
            config.urlCapture = {
                filterURLQuery: ["name", "page", "id"]
            }
        })(window['adrum-config'] || (window['adrum-config'] = {}));
    </script>
</head>
```

Configure the JavaScript Agent to Use HTTPS

Related pages:

- Limit Beacon Types
- Enable the Content Security Policy (CSP)

By default, the JavaScript Agent chooses the same transport (protocol) used to load the base page, but you can configure the JavaScript Agent to use HTTPS for a more secure network connection.

When configured to use HTTPS, the JavaScript Agent will use HTTPS to do the following:

- fetch `adrum-ext.js`
- fetch geolocation data
- send CORS beacons
- send image beacons

Set the useHTTPSAlways Flag

Following the code example below, you can set the boolean `useHTTPSAlways` to `true` to configure the JavaScript Agent to always use HTTPS.

```
<head>
    <script type='text/javascript' charset='UTF-8'>
        (function(cfg) {
            cfg.useHTTPSAlways = true;
        })(window['adrum-config'] || (window['adrum-config'] = {}));
    </script>
    <script src='https://cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'><
/script>
    ...
</head>
```

Configure the Number and Length of URL Segments

Related pages:

- Hide All or Parts of the URL Query String

When the number of segments or the length of a segment exceeds the default maximum, the URL will appear truncated in the Controller UI. For example, the resources shown in the Resource Details tab or the **Session Details** dialog may display truncated because of the number of URL segments or the length of the segments in the URL.

You can set configurations for the JavaScript Agent that set the maximum number of segments and the maximum number of characters for each segment. This allows you to view more or even less of the resource URLs in the Controller UI.

Segment Definition

A segment is a fragment of a URL. The following constitute one segment:

- `<protocol>://<domain-name>:<port>`
- `?key=value&key=value (query string)`
- `#someAnchor (anchor)`
- `/file-path/` (file path specified between two forward slashes)

For example, this URL consists of five segments: `http://example.com:8090/a/b.html#someAnchor?test=true&segments=5`

Truncation Rules and Examples

The maximum number of segments determines how many segments will be displayed in the Controller UI. The last segment will always be shown, and then the first *n* number of segments until the maximum number is reached.

For example, if the original URL is `http://example.com:8090/a/b.html#someAnchor?test=true&segment=5` and the configured maximum number of segments is 3, then the Controller UI would display the following: `http://example.com:8090/a/...?test=true&segment=5`

The maximum length of segments determines how many characters of a segment will be displayed in the Controller UI. The truncation rules are slightly more complicated as they depend on the type of segment:

- For the segment `<protocol>://<domain-name>:<port>`, the `<protocol>` and `<port>` are displayed, and if the remaining amount of the maximum length is greater than the length of `<domain-name>`, then the entire domain name is displayed. If the remaining amount of the maximum length is less than the length of `<domain-name>`, then the Controller UI would only display `http...8090`.
- For query strings, anchors, and file paths, the first character (?, #, /) is displayed as well as the last *n* number of characters until the maximum length is reached. For example, for the segment `?test=true&segment=5`, if the maximum length is 10, then the segment will be displayed as the following: `?...segment=5`

Default Values for Segments

The variable for setting the maximum number of segments to display is `maxResUrlSegmentNumber`. The variable for setting the maximum length of segments is `maxResUrlSegmentLength.`

The following are the default values for the two variables:

- `maxResUrlSegmentNumber=10`
- `maxResUrlSegmentLength=64`

Set the Maximum Number of Segments

The code snippet below sets the maximum number of segments to display at 15.

```
<head>
    <script type='text/javascript' charset='UTF-8'>
        (function(config){
            config.maxResUrlSegmentNumber = 15;
        })(window['adrum-config'] || (window['adrum-config'] = {}));
    </script>
    <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
    ...
</head>
```

Set the Maximum Length of Segments

The code snippet below sets the maximum number of characters of segments to display at 100.

```
<head>
    <script type='text/javascript' charset='UTF-8'>
        (function(config){
            config.maxResUrlSegmentLength = 100;
        })(window['adrum-config'] || (window['adrum-config'] = {}));
    </script>
    <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
    ...
</head>
```

Exclude Virtual Pages from Being Monitored

You can exclude virtual pages from being monitored based on the virtual pages' URLs. You define `exclude` filters to match URLs of virtual pages that you don't want to be monitored. You cannot, however, exclude virtual pages that you are manually reporting with the SPA2 JavaScript API.

Virtual Page Object

The `vp` object contains the `exclude` filter that specifies an array of patterns to match the URLs of virtual pages to exclude. By defining an `exclude` filter, you automatically enable SPA2 monitoring.

```
"spa": {
    "spa2": {
        "vp": {
            "exclude": {
                 "urls": [{"pattern": 'api'}, {"pattern": "resources"}]
            }
        }
 }
```

Virtual Page Filter Example

To use the virtual page filter, you set the JavaScript Agent with the `exclude` filter before you inject the `adrum.js` script.

For example, in the code snippet below, the pattern would match and exclude all virtual pages with URLs that have the strings "contact", "api", or "api" followed by one or more characters.

```
<head>
    <script type='text/javascript' charset='UTF-8'>
        (function(config){
            config.spa = {
                spa2: {
                    vp: {
                        exclude: {
                            "urls": [{"pattern": "contact"}, {"pattern": "api*"}]
                        }
                    }
                }
                        };
        })(window["adrum-config"] || (window["adrum-config"] = {}));
    </script>
    <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
    ...
</head>
```

Limit the Number of Ajax Requests

Related pages:

- Ajax Dashboard
- Ajax Request Snapshots
- Configure Naming for Ajax Requests
- Configure JavaScript and Ajax Error Detection

This page covers how to set the number of Ajax Requests.

By default, the JavaScript Agent limits the Ajax requests (using XHR or the Fetch API) sent for base or virtual pages. The limit is 250 requests for single-page applications (SPAs) and 50 for non-SPAs. Although base or virtual pages normally do not send large amounts of Ajax requests, in some cases, a page may send redundant Ajax requests, especially those containing error reports.

For example, a series of redundant Ajax requests containing a dead loop of errors and exceptions with no new information could overload the EUM Server. To prevent an overload of Ajax requests, you can configure the JavaScript Agent to limit the number of Ajax requests sent for base and virtual pages.

Set the Number of Ajax Requests

The code example below limits the number of Ajax request per base or virtual page to 7.

```
<head>
   <script type='text/javascript' charset='UTF-8'>
      window["adrum-start-time"] = new Date().getTime();
      window['adrum-config'] = {
         xhr: {
            maxPerPageView: 7
         }
      };
      ...
   </script>
   <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
   ...
</head>
```

Set an Unlimited Number of Ajax Requests

For use cases where all Ajax requests for a page need to be captured, you can configure the JavaScript Agent to report an unlimited number of Ajax requests per page with the configuration `xhr.maxPerPageView`.

> To prevent an overload of Ajax requests, it is highly recommended to only use this configuration for use cases when you absolutely need to capture all Ajax requests.

The following code example configures the JavaScript Agent to report an unlimited number of Ajax requests per page:

```
<head>
   <script type='text/javascript' charset='UTF-8'>
      window["adrum-start-time"] = new Date().getTime();
      window['adrum-config'] = {
         xhr: {
            maxPerPageView: "UNLIMITED"
         }
      };
      ...
   </script>
   <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
   ...
</head>
```

Capture Resource Timing Data Without Loss

Most browsers stop capturing resource timing data in a page when the number of resources reaches 150 because of the limit set by the Resource Timing API. You can configure the JavaScript Agent for conventional web pages (non-single page applications) and single-page applications (SPAs), however, to overcome this limitation.

> For Internet Explorer, you currently *cannot* use the JavaScript Agent configuration above to override the Resource Timing API's default limit of resources returned by the browser.

Because the JavaScript Agent configuration is different for non-SPA and SPA, see these sections for details and instructions:

Set and Clear the Resource Timing Buffer for Non-SPAs

For non-SPAs, you can configure the JavaScript Agent to use a buffer to capture resource timing data and then set a flag to clear the buffer once the beacon has transmitted the resource timing data to the EUM Server. Clearing the buffer clears the browser's resource buffer array and ensures that new resources will be sent in the next beacon.

> Overriding the Resource Timing API and clearing the buffer may affect JavaScript code on your page that is using the Resource Timing API. Thus, the JavaScript Agent uses the default set by the Resource Timing API unless you override the default by configuring the JavaScript Agent to set and clear the resource timing buffer.

JavaScript Configuration Example

The `resTiming` object is used for configuring the buffer size (maximum number of resources to return) and whether the buffer is reset once the beacon is transmitted to the EUM Server. The code snippet below shows you how to specify the buffer size and set the flag for clearing the buffer with the properties `bufSize` and `clearResTimingOnBeaconSend`.

```
<head>
    <script type='text/javascript' charset='UTF-8'>
        window['adrum-config'] = {
            resTiming: {
                bufSize: 200,
                clearResTimingOnBeaconSend: true
            }
        };
    </script>
    <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
</head>
```

Disable the Clearing of the Resource Timing Buffer for SPAs

For SPAs, the JavaScript Agent by default clears the resource timing buffer after it's full and saves the data in a local buffer. You can configure the JavaScript Agent so that the resource timing buffer is not cleared to capture resource timing data.

> If the resource timing buffer is not cleared, the resource timing correlation may miss some resources due to the buffer limit of the browser.

JavaScript Configuration Example

The default for `clearResTiming` is `true`. Setting `clearResTiming` to `false`, as shown below, disables the automatic clearing of the resource timing buffer. The resource timing buffer for virtual pages will also not be cleared.

```
 window['adrum-config'] = {
      ...
      "spa": {
           "spa2": {"clearResTiming": false}
      }
}
```

Filter XHR Calls by URLs

Related pages:

- Set the Number of Ajax Requests
- Configure Naming for Ajax Requests
- Configure Which Ajax Requests Are Sent to the Events Service

If you have many XHR calls (using XMLHttpRequest or the Fetch API) from your page that you do not need to monitor, you can use XHR filters so that the Agent only monitors a regex-defined list of specified calls.

When XHR calls are monitored, the absolute path of the XHR calls is provided to the JavaScript Agent. In the Controller UI, those monitored XHR calls will be displayed as configured by naming rules. If there are no naming rules, the absolute path of the XHR call is displayed.

XHR Filters

You can use the XHR filters below to include or exclude XHR calls. The filters can be in the form of an XHR filter object or an array of XHR filter objects.

- `xhr.include`
- `xhr.exclude`

Filter Object Structure

The XHR filter object has two properties: `urls` and `method`. The `urls` property is an array of objects containing a `pattern` property that specifies a regular expression for matching URLs. The `method` property is a string that specifies an HTTP method. You can use one or both of the properties. If you only use the `urls` array, the matched URLs will be either included or excluded for all HTTP methods. If you only specify `method`, all calls using the specified HTTP method will be either included or excluded.

The following is the general structure for the filter object:

```
{
    urls:
      [
        {
          pattern: ".*foo.*"
        },
        {
          pattern: ".*bar.*"
        }
      ],
    method: 'GET'
}
```

XHR Filter Example

To use XHR filters, you must assign XHR filters to `xhr.include` and `xhr.exclude` before you inject the `adrum.js` script. It's important to note that the exclude patterns override the include patterns, so those URLs that are matched by both the include and exclude patterns will ultimately be excluded.

For example, in the code snippet below, the include pattern would match all HTTP requests to `http://somedomain/app_status/user-profile.jsp`, but the exclude pattern would exclude POST calls to that URL.

```
<head>
    <script type='text/javascript' charset='UTF-8'>
        (function(config){
            (function(xhr) {
                xhr.include = {
                    urls: [
                        {
                            pattern: ".*ajax_info.txt"
                        },
                        {
                            pattern: ".*app_status.*"
                        }
                    ]
                };
                xhr.exclude = {
                    urls = [
                        {
                            pattern: ".*user-profile.*"
                        }
                   ],
                   method: "POST"
                };
            })(config.xhr || (config.xhr = {}));
        })(window["adrum-config"] || (window["adrum-config"] = {}));
    </script>
    <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
    ...
</head>
```

Report Events with the JavaScript API

The JavaScript API enables you to manually report events to the agent so that that it can time the various parts of your virtual page loads and correlate Ajax calls to those page loads. You can also capture and report errors using this API.

Notify the Agent of Events

Events are reported to the JavaScript Agent (ADRUM) by calling the `ADRUM.report` method and passing in an event tracker object.

API	Description
`ADRUM.report(tracker: eventTracker);`	Notifies the agent of an event.

Report Virtual Pages

SPA1 Monitoring

For SPA1 monitoring, use the event tracker VPageView to manually report virtual pages.

SPA2 Monitoring

You are required to enable SPA2 monitoring to use the API below to manually report virtual pages for SPAs. These APIs will also work in non-SPAs if you enable SPA2 monitoring.

To report virtual pages, you mark the beginning and end of virtual pages with the methods below. Both methods are called from the `ADRUM` object. See also Set Custom Virtual Page Names.

API	Parameter (s)	Default Value	Descriptions
`markVirtualPageBegin(VPName: string, manuallyMarkVPEnd?: boolean)`	`VPName`	N/A	Used to set the label for the virtual page. This label will be displayed in the Controller UI. The virtual page name must consist of a string of 760 or fewer alphanumeric characters. If the string length exceeds 760 characters, the page name will not be set.
	`manuallyMarkVPEnd`	`true`	A flag that indicates whether you or the JavaScript Agent mark the end of the virtual page. When set to true, you need to call `markVirtualPageEnd` to report the virtual page. When set to `false`, the JavaScript Agent will automatically mark the end of the virtual page.
`markVirtualPageEnd()`	N/A	N/A	Calling this method marks the end of the virtual page and triggers the JavaScript Agent to send a beacon with the virtual page information.

How the API Works

The steps below describe the process of manually reporting virtual pages with the API:

1. Start monitoring a virtual page by manually marking the beginning of the virtual page with `ADRUM.markVirtualPageBegin(VPName, manuallyMarkVPEnd)`.
2. A beacon with the set virtual page name is sent to the EUM Server. If `manuallyMarkVPEnd` is set to `true`, the JavaScript Agent will wait for you to call `ADRUM.markVirtualPageEnd` to report the virtual page. If `manuallyMarkVPEnd` is set to `false`, the JavaScript Agent will automatically mark the end of the virtual page.
3. You either call `ADRUM.markVirtualPageEnd()` to mark the end of the virtual page or the JavaScript Agent automatically marks the end of the virtual page.
4. The JavaScript Agent reports the virtual page metrics to the EUM Server.

Example of Reporting Virtual Pages

The Angular example below shows both ways to mark the beginning of a virtual page. The function `manualMarkVPEnd` calls `ADRUM.markVirtualPageBegin` that uses the default requiring you to manually mark the end of the virtual page. The function `allowJSAgentMarkVPEnd` passes the value `false` as the second parameter, so that the JavaScript Agent will automatically mark the end of the virtual page for you.

```
angular.module('myApp.controllers', [])
    .controller('VPCtrl', ['$scope', '$http', function ($scope, $http) {
            $scope.manualMarkVPEnd = function () {
                console.log("Mark the beginning of the virtual page and wait for markVirtualPageEnd() to be
called.");
                ADRUM.markVirtualPageBegin("VPExample-ManuallyMarkEnd");
            }
            $scope.allowJSAgentMarkVPEnd = function () {
                console.log("Mark the beginning of the virtual page and allow the JS Agent to mark the virtual
page end.");
                ADRUM.markVirtualPageBegin("virtualPageExample-JSAgentMarksEnd", false);
            }
            $scope.endVirtualPage = function () {
                console.log("Mark the end of the virtual page.");
                ADRUM.markVirtualPageEnd();
            }
            ...
        }
    ]
);
```

Report Events

Events are reported to the agent using event trackers. There are three different kinds of event trackers:

Event Tracker	Enabled for SPA2 Monitoring?	Description
VPageView	No	Used to track the stages of a virtual page view.
Ajax	Yes	Used to track Ajax requests.
Error	Yes	Used to track errors.

Common Properties

There are also two properties that are common to all tracker types:

- Gets or sets a URL

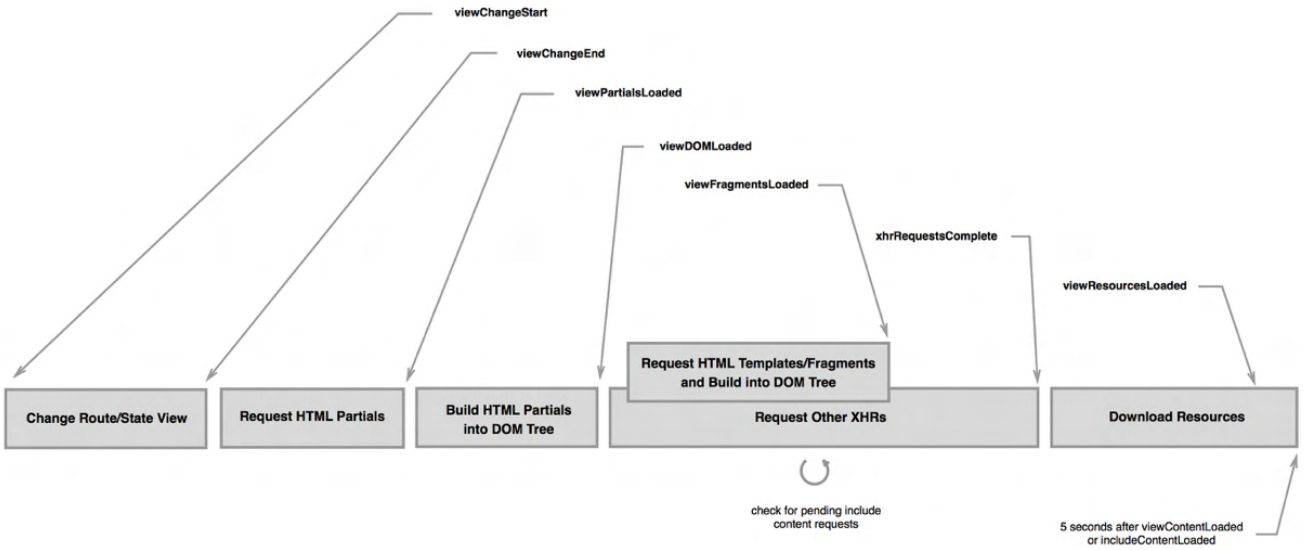
API	Description
`url(url?: string)`	Gets or sets a URL.

- Gets or sets the parent event identifier

API	Description
`parent(parent?: object)`	Gets or sets the parent event identifier.

VPageView

The following is the page view load flow in SPA1 monitoring. You'll use the SPA1 monitoring API below to set timing marks to match the below workflow as closely as possible in your own single page app framework. For SPA2 monitoring, see Report Virtual Pages: SPA2 Monitoring to learn about the SPA2 APIs for manually reporting virtual pages.



Based on the marks you set, AppDynamics derives the following key timing metrics. Marks should be called in the order in which they occur in the flow. The following table describes which marks used to calculate each metric.

Full Metric Name	Short Metric Name	How Calculated
End User Response Time (not used for waterfall UI)	PLT	`virtualPageStart` to `virtualPageEnd`
HTML Download Time	DDT	`viewChangeStart` to `viewChangeEnd`
HTML Download and DOM Building Time	DRT	`viewChangeStart` to `viewDOMLoaded`
DOM Building Time	DPT	`viewChangeEnd` to `viewDOMLoaded`
DOM Ready Time (used instead of PLT for waterfall UI)	DOM	`viewChangeStart` to `viewDOMLoaded`

Instantiate using `ADRUM.events.VPageView()`.

API	Description
`start()`	Indicates when a virtual page starts. It automatically calls: • `startCorrelatingXhrs()` • `markVirtualPageStart()`
`end()`	Indicates when a virtual page ends. It automatically calls: • `stopCorrelatingXhrs()` • `markVirtualPageEnd()`
`startCorrelatingXhrs()`	Correlates the Ajax requests sent after this call with the virtual page view event. The last tracker calling this method wins. This method is called automatically in the VPageView constructor. When a VPageView is created, the AJAX requests made after that call are automatically correlated to that VPageView. Use this separate call only when you want to set up manual correlation.
`stopCorrelatingXhrs()`	Stops correlating Ajax requests to the virtual page view event. Use this separate call only when you wish to set up manual correlation.
Setters	The default value for these is the time when the API is called.

`markViewChangeStart()`	Sets the view change start time.
`markViewChangeEnd()`	Sets the view change end time.
`markViewDOMLoaded()`	Sets the view DOM loaded time.
`markXhrRequestsCompleted()`	Sets the XHR requests completed time.
`markViewResourcesLoaded()`	Sets the view resources loaded time. This includes images, CSS files, and scripts.
`markVirtualPageStart()`	Sets the virtual page start time.
`markVirtualPageEnd()`	Sets the virtual page end time.
Getters	
`getViewChangeStart()`	Gets the view change start time.
`getViewChangeEnd()`	Gets the view change end time.
`getViewDOMLoaded()`	Gets the view DOM loaded time.
`getXhrRequestsCompleted()`	Gets the XHR requests completed time.
`getViewResourcesLoaded()`	Gets the view resources loaded time.
`getVirtualPageStart()`	Gets the virtual page start time.
`getVirtualPageEnd()`	Gets the virtual page end time.

Ajax

Instantiate using `ADRUM.events.Ajax()`.

API	Description
Property Setters/Getters	Call this without a parameter to get the value and with a parameter to set the value.
`method(method?: string)`	Gets or sets the method ("GET" or "POST") of the Ajax.
Timing Setters	The default value for these is the time when the API is called.
`markSendTime(sendTime?: number)`	Sets the time the request is sent.
`markFirstByteTime(firstByteTime?: number)`	Sets First Byte Time.
`markRespAvailTime(respAvailTime?: number)`	Sets Response Available Time.
`markRespProcTime(RespProcTime?: number)`	Sets the time the response is completely processed.
Timing Getters	
`getSendTime()`	Gets the time the request was sent.
`getFirstByteTime()`	Gets First Byte Time.
`getRespAvailTime()`	Gets Response Available Time
`getRespProcTime()`	Gets the time the response was completely processed.

Errors

Instantiate using `ADRUM.events.Error()`.

API	Description
Property Setters/Getters	Call these without a parameter to get the value and with a parameter to set the value.
`msg(msg?: string)`	Gets or sets the error message.
`line(line?: number)`	Gets or sets the line number of source code where the error happened.

Correlate Ajax Requests

Ajax requests can be correlated to virtual page views automatically or manually. When you create a vPageView tracker, `startCorrelatingXhrs()` is called automatically in the constructor, correlating any subsequent Ajax calls with that VPageView event. To set up manual correlation, call `stopCorrelatingXhrs()` to stop the automatic process and then call `startCorrelatingXhrs()` where you wish correlation to re-commence.

Sample Code

Report a custom Error event by passing properties via setters

```
var errorT = new ADRUM.events.Error();
errorT.msg('I am a custom error at line 100');
errorT.line(100);
ADRUM.report(errorT);
```

Report a custom Error event by passing properties via the constructor

```
var errorT = new ADRUM.events.Error({
msg: 'I am a custom error at line 100',
line: 100
});
ADRUM.report(errorT);
```

Report a custom Ajax event passing properties via setters

```
var ajaxT = new ADRUM.events.Ajax();

// set url
ajaxT.url('your xhr Url');

// mark timings
ajaxT.markSendTime(100);
ajaxT.markFirstByteTime(200);
ajaxT.markRespAvailTime(300);
ajaxT.markRespProcTime(400);
ADRUM.report(ajaxT);
```

Set up backbone SPA monitoring

```
var AppRouter = Backbone.Router.extend({
    routes: {
        "wines/:id": "wineDetails"
    },
    wineDetails: function (id) {
        var vpView = new ADRUM.events.VPageView();
        vpView.markVirtualPageStart();
        // vpView.markViewChangeStart();
        var wine = new Wine({id: id});
        wine.fetch({success: function(){
            vpView.markXhrRequestsCompleted();
            $("#content").html(new WineView({model: wine}).el);
            vpView.markViewDOMLoaded();
            vpView.markVirtualPageEnd();
            ADRUM.report(vpView);
        }});
        this.headerView.selectMenuItem();
    }
});
```

Correlate Ajax requests wih VPageView Events

```
var vPageView = new ADRUM.events.VPageView({
    url: 'http://localhost/#virtualpage1',
});

vPageView.start();

// SPA view routing and HTML partials fetching
vPageView.markViewChangeStart()
// AJAX requests for the HTML partials are automatically correlated with the VPageView
...
vPageView.markViewChangeEnd();

// HTML partials inserted into Browser DOM tree
...
vPageView.markViewDOMLoaded();

// SPA HTML AJAX data fetching
// Data AJAX requests are automatically correlated with the VPageView
...

vPageView.markXhrRequestsCompleted();

// call this when ending a new virtual page
vPageView.end();

ADRUM.report(vPageView);
```

You can exclude certain Ajax calls from being monitored by configuring ADRUM itself Before you invoke the `adrum.js` script at the top of your page, add lines similar to the following:

Exclude Ajax from VPageView using ADRUM configuration

```
window['adrum-config'] = {
  "spa": {
    "angular": {
      "vp": {
        "xhr": {
          "exclude": {
            "urls": [{
              "pattern": 'heartBeatAjax'
            }]
          }
        }
      }
    }
  }
}
```

Or you can exclude certain Ajax calls using the `vPageView.stopCorrelatingXhrs()` call, and then turn correlation back on with `vPageView.startCorrelatingXhrs()`, as in the following:

Exclude Ajax from VPageView event manually

```
var vPageView = new ADRUM.events.VPageView();
vPageView.stopCorrelatingXhrs();

var xhr = new XMLHttpRequest();
xhr.open('GET', '/heartBeatAjax');
xhr.send();

vPageView.startCorrelatingXhrs();
```

Disable Monitoring of Fetch API Calls

By default, the JavaScript Agent reports Ajax requests made with XMLHttpRequest object (XHR) and the Fetch API.

You should only consider disabling monitoring support for the Fetch API for these use cases:

- You just started using the JavaScript Agent 4.5.6 and start to see JavaScript errors caused by Ajax requests.
- Your browser app uses one of these libraries:
  - Bluebird
  - shim.js
  - Zone.js

To disable monitoring for the Fetch API, you set `config.fetch` to `false` as shown below.

```
<script type='text/javascript' charset='UTF-8'>
    (function(config){
       config.fetch = false;
    })(window['adrum-config'] || (window['adrum-config'] = {}));
</script>
<script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
```

To disable monitoring for the Fetch API for single-page applications:

```
<script type='text/javascript' charset='UTF-8'>
    (function(config){
       config.fetch = false;
       config.spa = {
            "spa2": true
        };
    })(window['adrum-config'] || (window['adrum-config'] = {}));
</script>
<script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
```

Inject the JavaScript Agent

Related pages:

- Overview of Injection Types
- Undo Injection

This page describes how JavaScript Agent injection works, including the different injection types.

To instrument your application for Browser RUM, you must configure your JavaScript Agent and insert it into the page that is returned to the end user as part of the normal process it follows. The act of inserting the agent is called *injection*. There are several ways to inject the JavaScript Agent for Browser RUM into your web pages. You also have several JavaScript hosting options to choose from that offer differing degrees of control, simplicity, and reliability.

To configure the JavaScript Agent, see Configure the JavaScript Agent.

How the JavaScript Agent Works

The JavaScript Agent consists of two files: `adrum.js` and `adrum-ext.js.`

1. The file `adrum.js` is injected into each instrumented page, as close to the top as possible, as the page is served. The file loads synchronously at around 90 KB (30 KB with compression).
2. This first JavaScript file starts a timer and does some quick configuration and setup.
3. At the `onload` event, `adrum-ext.js` is fetched *asynchronously* to prevent blocking the page load. This is the code that does most of the heavy lifting. Once it has been fetched, it is cached for 24 hours on the browser.
4. When cross-domain session correlation is enabled, the first time a user visits a domain, the browser additionally loads the file `adrum-xd.html` to generate or load a piece of common information that enables cross-domain sessions. Future visits will not load `adrum-xd.html` again.
5. When the page has completed loading, the collected data is bundled into a beacon and sent to the EUM Server by `adrum-ext.js`.
6. The data is processed by the EUM Server and then made available for pickup by the Controller.

Not all types of injection are supported on all frameworks and platforms. See the Script Injection columns in the Supported Platform Matrix for Browser Monitoring matrices to find out what types are supported for your application.

Injection Methods

There are three methods for injecting the JavaScript Agent. See the following for each injection method:

- Manual Injection of the JavaScript Agent
- Automatic Injection of the JavaScript Agent
- Assisted Injection

JavaScript Agent Hosting Options

The `adrum.js` file is inserted into the page when it is downloaded from your web application. The `adrum-ext.js` (and `adrum-xd.html` when cross-domain session correlation is enabled) file is loaded asynchronously by the `adrum.js` file. By default, `adrum-ext.js` and `adrum-xd.html` are fetched by the highly available Amazon CloudFront CDN infrastructure.

There are three hosting options for the JavaScript Agent:

- AppDynamics CDN - All the JavaScript Agent files are from the host `cdn.appdynamics.com`. This is the simplest hosting option and ideal for testing.
- Self-Hosting - You host all of the JavaScript Agent files. This option gives you the most control and is recommended for production.
- Shared Hosting - You host the main file `adrum.js`, but the other files that are loaded asynchronously are from the AppDynamics CDN. This choice offers control of the most important file of the JavaScript Agent and is ideal for small to medium-sized businesses that don't have the resources or prefer not to host the entire JavaScript Agent.

See Manual Injection of the JavaScript Agent for step-by-step instructions.

Cross-domain session correlation

By default, Browser RUM sessions are restricted to one domain. Thus, when an end user navigates to a page in another domain or even subdomain, the session for that user is ended. You can, however, configure Browser RUM to enable sessions to continue across subdomains. Thus, when an end user navigates from `http://example1.com/` to `http://example2.com/`, the session will continue as long as the configured session inactivity time has not expired.

How Sessions Are Correlated Across Multiple Domains

For sessions to be correlated across domains, each page is required to use HTTPS to load the file `adrum-xd.html` from the same URL. If your pages are fetching the files `adrum-ext.js` and `adrum-xd.html` files from the AppDynamics CDN (this includes the shared hosting use case), sessions will automatically be correlated across domains because the pages will be fetching the file `adrum-xd.html` from the AppDynamics CDN.

For those *self-hosting* the JavaScript Agent files, you will need to configure the JavaScript Agent to use HTTPS to load the `adrum-ext.js` file from the same URL. The `adrum-xd.html` file is served from the location of the `adrum-ext.js` file. See the sections below for configuration instructions for the two use cases.

Configuration for Session Correlation Across Multiple Domains

AppDynamics CDN / Shared Hosting

When using the AppDynamics CDN or shared hosting, your JavaScript Agent configurations must enable cross-domain session correlation and use HTTPS to call the `adrum-ext.js` file. Thus, in your JavaScript Agent configuration make sure you have the following lines:

```
...
config.adrumExtUrlHttps = 'https://cdn.appdynamics.com';;
config.xd = {enable : true};
...
```

Self-Hosting

When using self-hosting, your JavaScript Agent configurations must enable cross-domain session correlation and use HTTPS to call the `adrum-ext.js` file from the same URL. Thus, in your JavaScript Agent configuration make sure you have the following lines and that the value for `adrumExtUrlHttps` is the same for all pages requiring cross-domain session correlation.

```
...
// HTTPS is needed to fetch adrum-xd.html.
config.adrumExtUrlHttps = 'https://<your-adrum.ext-host>';
// This config enables cross-domain session correlation.
 config.xd = {enable : true};
...
```

Set the Geo Server URL

By default, end-users' locations are resolved using public geographic databases. You can host an alternate geo server for your countries, regions, and cities instead of using the default geo server hosted by AppDynamics. See Install and Host a Custom Geo Server for Browser RUM.

Custom Configuration

You can add configuration for `adrum.js` to customize and extend the functionality of the JavaScript Agent. See Configure the JavaScript Agent for instructions and examples on adding custom configuration.

JavaScript Agent Limitations

The JavaScript Agent cannot monitor and report the activity of service workers or web workers. For example, if you use a service worker to make an XHR request, the JavaScript Agent will not be able to report it.

Overview of Injection Types

You need to inject the JavaScript Agent into your web pages to use Browser RUM. There are several injection types that you can use depending on your use case. This page describes the different injection types, how they work, and when to use each type.

Choosing an Injection Method

If you are uncertain which procedure to use to inject the agent into your web pages, follow these guidelines in the given order:

- If you want to use Browser RUM and do not have any app agents on the server side, use manual injection. See Manual Injection of the JavaScript Agent.
- If automatic injection is available and works for your framework, use automatic injection. Automatic injection requires the least amount of effort because you do not have to manually instrument every page. Check the matrices at Browser RUM Supported Environments to see if automatic injection has been tested in your environment. Also, see Automatic Injection of the JavaScript Agent.
- If you cannot use automatic injection, and you can edit the source code of *your web pages*, use manual injection. See Manual Injection of the JavaScript Agent.
- If you cannot use automatic injection, and you can edit the source code of *your web application*, use one of the kinds of assisted injection. See Using Injection Rules (Java Only) or Assisted Injection-Using Attribute Injection.

Manual Injection

Manual injection is supported on all platforms and frameworks. To set up a manually injected page:

- Choose a JavaScript Agent hosting option. See JavaScript Agent Hosting Options to learn what hosting option best suits your use case.
- Manually configure each page to find the location of the JavaScript Agent, so that it can be executed as the page is constructed by the browser.

For detailed instructions on using manual injection in your application, see Manual Injection of the JavaScript Agent.

Automatic Injection

If you are using a Java or .NET app agent on the server-side, and your application is running in an environment that supports any of the following:

- Apache Jasper JSP compiler (for Java) or
- ASP.NET or ASPX (for .NET) or
- ASP.NET Core (for .NET; Windows support only)

You *may* be able to use automatic injection, where the server-side app agent completely manages injecting the code at runtime. For instructions on using automatic injection in your application, see Automatic Injection of the JavaScript Agent.

Assisted Injection

Assisted injection is available in two variants. In both cases, some of the work is done manually by you and some of the work is done by the server-side Java or .NET app agent.

> Assisted injection is currently not supported for .NET Core.

Assisted Injection Using Injection Rules (Java Only)

In this type of assisted injection, you configure rules that define which app server Java classes and methods write to the output stream of your application and the writer object that is used to do that writing. AppDynamics intercepts the method and injects the JavaScript Agent into the output stream. You also specify which server-side business transactions you wish to have instrumented in this way.

For detailed information on using this form of assisted injection, see Assisted Injection.

Assisted Injection Using Attribute Injection

In this type of assisted injection, you copy small code snippets appropriate to your framework into your page templates or other code that creates your pages. This snippet contains the two variables `JS_HEADER` and `JS_FOOTER`, which the app agent replaces with the appropriate information in the response object at runtime.

For detailed information on using this form of assisted injection, see Using Attribute Injection.

Container-Based Injection

If you are using Nginx or Apache as a web container or as a reverse proxy in front of your web container, you can use directives to inject the agent into the response object. See Injection Using Nginx or Injection Using Apache.

Manual Injection of the JavaScript Agent

This page describes how to instrument a browser application by manually injecting the JavaScript Agent. To configure the JavaScript Agent before injection, see Configure the JavaScript Agent.

Manually Inject the JavaScript Agent

The Controller UI allows you to configure the JavaScript Agent, generate an HTML snippet, and add that snippet to the header of the web pages.

1. In the Controller UI, go to a browser application.
2. In the left navigation menu, select **Configuration** > **Configure JavaScript Agent**.
3. Configure the JavaScript Agent.
4. Save and copy the HTML snippet.
5. Add the HTML snippet to the header of the web pages.

JavaScript Agent Placement

The recommended location for the JavaScript Agent is at the top of the `<head>` section. The `adrum.js` file captures the time as the page begins to load and measures that amount of time against the time that other timings are calculated for all browsers. Many modern browsers, however, support the Navigation Timing API (see this list), and for these browsers, timings can be acquired via the API. In this case, it is possible to place the JavaScript Agent somewhere else on the page, but useful timing information is only available for those `NavTime`-capable browsers.

Synchronous Versus Asynchronous Script Load

AppDynamics recommends that the `adrum.js` file loads synchronously because once downloaded, the JavaScript Agent begins monitoring for page data such as Ajax events, resources, and errors. If `adrum.js` loads asynchronously for the first time (through the `async` attribute), the JavaScript Agent might not report all page data.

Use Previous JavaScript Agent Versions

Previous versions of the agent can be found at the following location, where `VERSION` is the version number you want to access (for example, 4.5.0.).

```
http://cdn.appdynamics.com/adrum/adrum-VERSION.js
```

- If you are using a SaaS EUM Cloud and want a hosted version of `adrum.js`, AppDynamics recommends you use the `adrum-latest.js` version. However, you can use any version of the JavaScript Agent as it is compatible with any SaaS EUM Cloud version currently being deployed.
- If you are using an on-premises EUM Server and want to use a hosted version of `adrum.js`, you need to match the `adrum.js` version to your EUM Server version.
- For Controller versions, a JavaScript Agent version is compatible with Controller versions with the same or older versions. For example, JavaScript Agent 20.6.0 is compatible with Controller <= 20.6.0.

Verify Manual Injection

Once the agent is injected, it can take the AppDynamics Controller a few minutes to discover and recognize the page, which must happen before data will begin to appear. If, however, you have configured your page using manual injection and are not seeing Browser RUM metrics after running load for a while, check the web page to confirm that the JavaScript Agent for Browser RUM is present in the page. If not, try injecting the script again.

If, after two attempts, you still do not see Browser RUM metrics, try one of the other injection schemes if they are available for your platform. See Troubleshoot Browser RUM.

Automatic Injection of the JavaScript Agent

This page describes how to instrument a browser application by automatically injecting the JavaScript Agent. Automatic injection uses AppDynamics server-side agents to automatically add the `adrum` header and footer to each of your web pages.

Automatic Injection Support for Server-side Agents

The Java Agent and .NET Agent support automatic injection. See .NET Supported Environments for version specifics.

Automatic injection is available only for server-side applications built on these environments:

- Jasper-supported JSP (Java)
- ASP.NET
- ASP.NET Core
- ASPX (.NET) Framework

- Although the Apache Agent does *not* support automatic injection, you can configure Apache or Nginx to inject the JavaScript Agent.
- If the server-side application does not return well-formed HTML, the JavaScript Agent may not be able to instrument the page. For example, the JavaScript Agent may not be able to instrument HTML pages that are missing elements or have unclosed tags.

JavaScript Agent and Controller Versions

The table below lists the Javascript Agent version deployed for auto-injection in each Controller version.

The JavaScript Agent version does not match the Controller version because the Controller and agents have different release cycles.

Controller Version	JavaScript Agent Version Supported for Auto-Injection
21.4.0	20.12.0

Access the User Experience App Integration Panel

1. From the **Applications** page, open the business application that you want to automatically inject the JavaScript Agent into your browser application.
2. From the left navigation bar, select **Configuration**.
3. Click **User Experience App Integration**.

Enable Business Transaction Correlation

From the **Business Transaction Correlation**, check the **Enable Business Transaction Correlation** check box to correlate Pages and Ajax requests with server-side business transactions.

Specify Business Transactions to Include Correlation Headers

You can also specify which business transactions will include or exclude correlation headers. If you do not add request match rules or request exclude rules, correlation headers will be added to all requests.

To add a request rule:

1. Click the **plus** icon. The **Create HTTP Request Match Rule** will display.
2. Check the **Method** checkbox and select an HTTP method that you want to match.
3. Check the **URI** checkbox and enter your criteria.
4. Click **Save**.

Enable Automatic Injection

1. Select the JavaScript Agent Injection tab.
2. From the **Inject the JavaScript Agent configured for this Browser App** dropdown, select the browser application you want to use automatic injection.
3. Check the **Enable Automatic Injection of JavaScript** check box to enable automatic injection.
4. If you enabled automatic injection, click **OK** in the **Confirm Enabling Automatic Injection** dialog.
5. Click **Save** in the outer configuration pane.

Configure Automatic Injection

After you have enabled automatic injection:

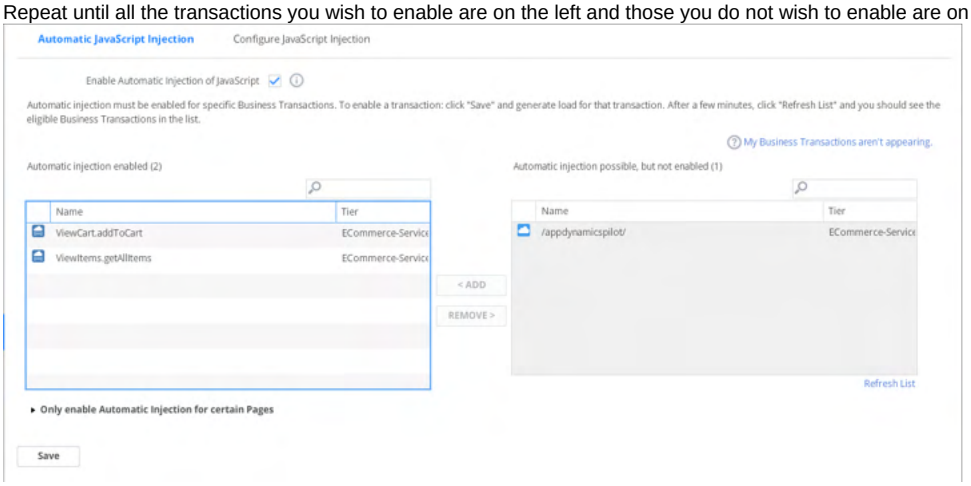
- You must specify the server-side business transactions for which automatic JavaScript injection is enabled.
- You can limit which pages to inject by creating custom match and exclude rules for automatic injection. If you do not configure these rules, by default AppDynamics injects all pages visited by the enabled business transactions.

Use these rules to fine-tune which business transactions to include or exclude from injection based on match criteria. For example, you can exclude all business transactions that have a certain string in their URLs or set a certain cookie. The configurations for include rules and exclude rules are similar. It depends on your situation whether it is more convenient to restrict transactions based on inclusion or exclusion.

Specify Business Transactions for Automatic Injection

You must select at least one business transaction for automatic injection.

1. From the list on the right shown in the screenshot, select one or more business transactions. (If you don't see any business transactions, click **Refresh List**.) Not all your business transactions may appear here—the list includes only those transactions that AppDynamics can parse for automatic injection, those based on Jasper-compiled JSPs or ASP.NET, ASP.NET Core, or ASPX.NET pages.
2. Click **< Add** to move the business transaction to the list on the left.
3. Repeat until all the transactions you wish to enable are on the left and those you do not wish to enable are on the right.



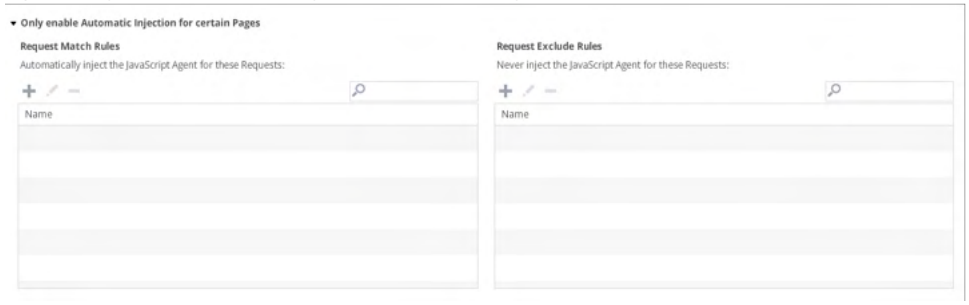
4. Click **Save**.

Create Match Rules for Automatic Injection

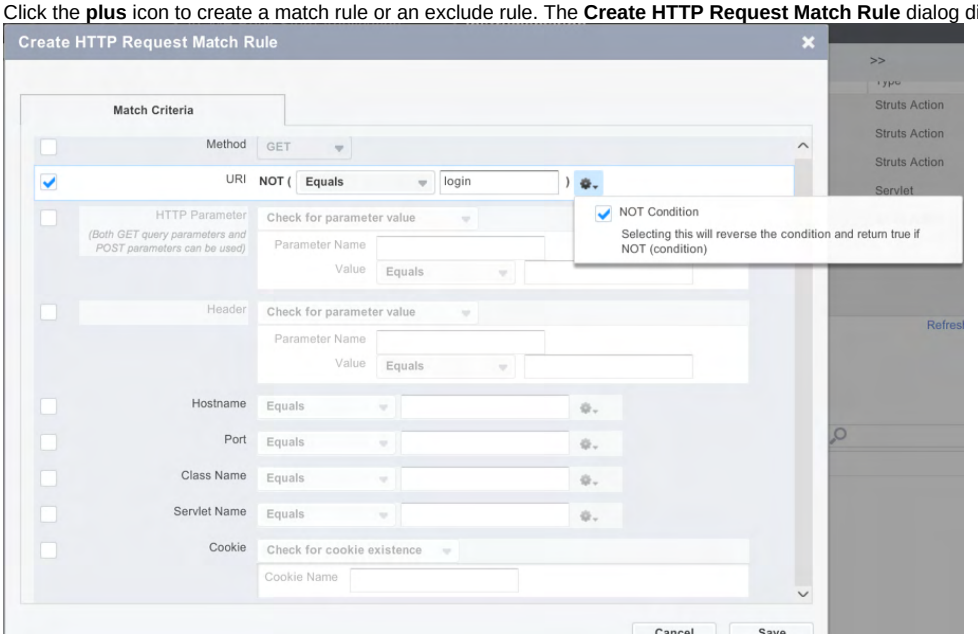
You may not wish to instrument every page in your application. For example, if your application has a very large number of pages, you might want to instrument only the key ones for your business, to avoid hitting licensing limits. Or, when you are in the setup and test phase, you might only want to instrument a few pages to keep your initial sample manageable.

Use match rules to include or exclude certain pages:

1. Expand **Only enable Automatic Injection for certain Pages** if it is closed.



2. Click the **plus** icon to create a match rule or an exclude rule. The **Create HTTP Request Match Rule** dialog displays.



3. Select one or more criteria to match. If you select multiple criteria, **all** must match for the rule to come into effect. Use the **gear** icon to set a NOT condition. See Using Regular Expressions for general information about match rules.
4. Click **Save**.
5. Click **Save** in the outer configuration pane.

You can later edit or remove a match rule by selecting it in the list and clicking the edit or delete icon.

Injection Using Nginx

Related pages:

- Manual Injection of the JavaScript Agent
- Automatic Injection of the JavaScript Agent
- Using Attribute Injection

If you are using Nginx as your web container, or use Nginx as a reverse proxy, you can use a container substitution module to automatically inject the JavaScript Agent into your pages. The module intercepts the response object as it is being returned and makes a string substitution.

Download the Agent

You must first download the JavaScript Agent from the **Configuration** screen.

1. Open the browser application in which you are interested.
2. From the left navigation menu, select **Configuration**.
3. Click the **Configure and download JavaScript Agent**.
4. For the JavaScript hosting option, select **I will host all the JavaScript agent files.**
5. Click **Download** to download the JavaScript Agent.
6. Place the file somewhere accessible to the Nginx instance. The name of the saved file should be `adrum.js`.

Configure Nginx with `ngx_http_sub_module`

The `ngx_http_sub_module` module is a filter that modifies a response by replacing one specified string by another. You can use this feature to have the server automatically inject the header portion of the JavaScript Agent into a served page. For more information on the process, see the Nginx documentation, Module ngx_http_sub_module.

For example, modify the location context to replace the <head> tag with the <head> tag and the JavaScript Agent scripts.

Sample Nginx Configuration

```
location / {
    sub_filter      <head>
        '<head><script>window["adrum-app-key"]="<EUM_APP_KEY>";window["adrum-start-time"]=new Date().getTime();<
/script><script type="text/javascript" src="//cdn.appdynamics.com/adrum/adrum-latest.js"></script>';
    sub_filter_once on;
}
```

In the sample above, note that `/adrum-latest.js` is the path to a copy of the `adrum` file that is accessible to the server. Also, note the timer initialization `<script>window['adrum-start-time'] = new Date().getTime();</script>`. Keep these as close as possible to the top of the document, preferably right after the <head> tag, ensures the best possible timings.

If you use <meta> tags, you should place them right after your <head> tag, and then place the JavaScript Agent directly after the last <meta> tag. This can avoid issues with some versions of IE.

You may need to escape some characters, depending on your platform. For example, on Mac OS:

```
location / {
    sub_filter      <head>
        '<head><script>window[\'adrum-start-time\'] = new Date().getTime();</script><script src="/adrum.js"><
/script>';
    sub_filter_once on;
}
```

This Nginx module is often used for adding the Google Analytics script. Be careful not to overwrite any existing GA script when you do this.

Possible variations on the script string can be found in Configure the JavaScript Agent.

Configure Ngnix without `ngx_http_sub_module`

You can configure Ngnix without the `ngx_http_sub_module` module if you prefer. Insert the required script into the `conf.d/default.conf` location.

Sample Nginx Configuration

```
location / {
    sub_filter                 <head>
                '<head><script>window["adrum-app-key"]="<EUM_APP_KEY>";window["adrum-start-time"]=new Date().
getTime();</script><script type="text/javascript" src="//cdn.appdynamics.com/adrum/adrum-latest.js"></script>';
    sub_filter_once on;
}
```

Injection Using Apache

Related pages:

- Manual Injection of the JavaScript Agent
- Automatic Injection of the JavaScript Agent
- Using Attribute Injection
- Using Injection Rules

This page describes how to instrument a browser application using Apache to automatically inject the JavaScript Agent. If you are using Apache as your web container, or you are currently using, or willing to use, Apache as a reverse proxy, you can use a container substitution module to automatically inject the JavaScript Agent into your pages. The module intercepts the response object as it is being returned and makes a string substitution.

This method uses the two Apache modules `mod_substitute` and `mod_filter`.

Download the Agent

You must first download the JavaScript Agent from the Browser Monitoring configuration pane.

1. Open the browser application in which you are interested.
2. From the left navigation menu, select **Configuration**.
3. Click the **Configure and download JavaScript Agent**.
4. For the JavaScript hosting option, select **I will host all the JavaScript agent files.**
5. Click **Download** to download the JavaScript Agent.
6. Place the file somewhere accessible to the Apache instance. The name of the saved file should be `adrum.js`.

Configure Apache

The basic configuration procedure includes:

1. Make Sure the Modules are Loaded
2. Create an Adrum Configuration File
3. Add the Location of the Adrum Configuration File to httpd.conf
4. Restart the Web Server

Make Sure the Modules are Loaded

Check your global Apache `httpd.conf` file and make sure that the following two `LoadModule` commands are in the file:

```
LoadModule substitute_module modules/mod_substitute.so
LoadModule filter_module modules/mod_filter.so
```

Create an Adrum Configuration File

Create a file named `adrum.conf` with contents similar to the example below, based on your Apache version. (See this Stack Overflow post for details.) In this case, the substitution rule covers the location of the entire site (`Location /`) but you can also recursively select a specific directory and its subdirectories by using `/somedirectory`.

Sample adrum.conf Apache 2.2

Replace the sample code below with your own JavaScript Agent script.

```
<Location "/">
    SetOutputFilter INFLATE;SUBSTITUTE;DEFLATE
    AddOutputFilterByType SUBSTITUTE text/html
    Substitute "s#<head>#<head><script>window['adrum-start-time'] = new Date().getTime();</script><script>
(function(config){config.appKey='<EUM_APP_KEY>';})(window['adrum-config'] || (window['adrum-config'] = {}));<
/script><script src='./adrum.js'></script>#inq"
</Location>
```

If you use `<meta>` tags, you should place them right after your `<head>` tag, and then place the JavaScript Agent directly after the last `<meta>` tag and before other `<script>` tags. This can avoid issues with some versions of IE and improve the accuracy of the resource timing.

Sample adrum.conf Apache 2.4

Replace the sample code below with your own JavaScript Agent script.

```
<Location "/">
    AddOutputFilterByType SUBSTITUTE text/html
    Substitute "s#<head>#<head><script>window['adrum-start-time'] = new Date().getTime();</script><script>
(function(config){config.appKey='<EUM_APP_KEY>';})(window['adrum-config'] || (window['adrum-config'] = {}));<
/script><script src='./adrum.js'></script>#inq"
</Location>
```

Where `./adrum.js` is the path to a copy of the adrum file that is accessible to the server. The flags after the `#` are:

- `i` - matching is case-insensitive
- `n` - pattern is treated as a fixed string (removing the n means the pattern is treated as a regular expression)
- `q` - module does not flatten the buckets after each substitution - this can speed performance.

For more information, see the Apache module docs here.

If your <head> tag has an attribute like `<head lang="en">`, you can use a regex in the substitution string and omit the n flag. Replace the sample code below with your own JavaScript Agent script.

```
Substitute "s#(<head[^>]*>)#$1<script>window['adrum-start-time'] = new Date().getTime();</script><script>
(function(config){config.appKey='<EUM_APP_KEY>';})(window['adrum-config'] || (window['adrum-config'] = {}));<
/script><script src='./adrum.js'></script>#iq"
```

Note the timer initialization: `<script>window['adrum-start-time'] = new Date().getTime();</script>`. Injecting these scripts as close as possible to the top of the document, preferably right after the <head> tag, ensures the best possible timings.

Also, if you use <meta> tags, you should place them right after your <head> tag, and then place the JavaScript Agent directly after the last <meta> tag and before other `<script>` tags. This can avoid issues with some versions of IE and again improve the accuracy of the resource timing.

(Optional) Adjust for gzipped resources

If your page is compressed, the substitution won't work unless the content is INFLATEd, the substitution is made, and then the content is DEFLATEd. There are multiple ways to do this. For example, in the `FilterProvider` line:

```
FilterProvider AdrumFilter INFLATE;SUBSTITUTE;DEFLATE resp=Content-Type $text/html
```

See this Stack Overflow post for more information. If you are using Apache as a proxy, you can also instruct it not to accept gzip-encoded content.

Add the Location of the Adrum Configuration File to httpd.conf

Add the following line to your global Apache `httpd.conf` file:

```
Include [absolutePathTo]/adrum.conf
```

Alternatively, you can add the directives to the `httpd.conf` file directly instead of creating a separate adrum file.

Restart the Web Server

To pick up the new configuration, restart:

```
sudo apchectl -k restart
```

If you get a warning "Useless use of AllowOverride in line 2 of [absolutePathTo]/adrum.conf", it can be ignored. It simply means `AllowOverride` is redundant. You can remove it if you wish.

Possible variations on the script string can be found in Configure the JavaScript Agent.

Other Alternatives

If you are setting up your automatic injection using an Apache instance that is configured as a reverse proxy, you *must* use the `Location` directive method described above, with the `ProxyPass` and `ProxyPassReverse` directives also in the `Location` directive. If you are using an Apache instance that is your primary web container you have two additional options for describing the actual substitution step:

- Using the Directory Directive
- Using .htaccess

Use the Directory Directive

You can use the `Directory` directive instead of the `Location` directive.

Use .htaccess

Add lines similar to this to an `.htaccess` file in the base document directory for your site, replacing your filter name for `<MyFilter>`. If you don't have an `.htaccess` file, create one:

Apache 2.2

Replace the sample code below with your own JavaScript Agent script.

```
Substitute "s#<head>#<head><script>window['adrum-start-time'] = new Date().getTime();</script><script>(function
(config){config.appKey='<EUM_APP_KEY>';})(window['adrum-config'] || (window['adrum-config'] = {}));<
/script><script src='./adrum.js'></script>#inq"
```

```
AllowOverride Options
FilterDeclare <MyFilter>
FilterProvider <MyFilter> SUBSTITUTE resp=Content-Type $text/html
FilterChain <MyFilter>
Substitute "s#<head>#<head><script>window['adrum-start-time'] = new Date().getTime();</script><script>(function
(config){config.appKey='<EUM_APP_KEY>';})(window['adrum-config'] || (window['adrum-config'] = {}));<
/script><script src='./adrum.js'></script>#inq"
```

Apache 2.4

Replace the sample code below with your own JavaScript Agent script.

```
Substitute "s#<head>#<head><script>window['adrum-start-time'] = new Date().getTime();</script><script>(function
(config){config.appKey='<EUM_APP_KEY>';})(window['adrum-config'] || (window['adrum-config'] = {}));<
/script><script src='./adrum.js'></script>#inq"
```

```
AddOutputFilterByType SUBSTITUTE text/html
Substitute "s#<head>#<head><script>window['adrum-start-time'] = new Date().getTime();</script><script>(function
(config){config.appKey='<EUM_APP_KEY>';})(window['adrum-config'] || (window['adrum-config'] = {}));<
/script><script src='./adrum.js'></script>#inq"
```

Where `./adrum.js` is the path to a copy of the adrum file that is accessible to the server. Make sure `.htaccess` is world-readable.

Assisted Injection

Related pages:

- Automatic Injection of the JavaScript Agent
- Injection Using Nginx
- Injection Using Apache

This page describes how to instrument a browser application through assisted injection of the JavaScript Agent. Assisted injection is when your server-side application injects the JavaScript Agent into your browser application. You can configure your business applications to inject the JavaScript Agent into your browser applications.

Types of Assisted Injection

You can perform assisted injection with rules or through attributes. This table summarizes the platforms supported for each type of assisted injection and the process for performing the assisted injection.

Type of Assisted Injection	Supported Platforms	Description
Injection Rules	Java	Type of assisted injection uses rules to configure which Java classes should be intercepted. You create the injection rules in the **User Experience App Integration Panel** of the Controller UI for the business application that will inject the JavaScript Agent.
Attribute Injection	Java, ASP.NET	Type of assisted injection relies on templates that tell the app agent where to inject information. You enable attribute injection in the **User Experience App Integration Panel** of the Controller UI for the business application that will inject the JavaScript Agent. Additionally, you add code snippets in the page templates that determine where the JavaScript Agent is injected.

Injection Rules (Java Only)

To have your server-side application use assisted injection of the JavaScript Agent using injection rules, you define rules to configure:

- The Java classes and methods that should be intercepted
- The Java writer object and method to use to add the agent to the response object

> Assisted injection using rules is available for Java frameworks only.

Access the User Experience App Integration Panel

1. From the **Applications** page, open the business application that you want to automatically inject the JavaScript Agent into your browser application.
2. From the left navigation bar, select **Configuration**.
3. Click **User Experience App Integration**.

Create JavaScript Injection Rules

1. In the JavaScript Agent Injection tab, select a browser application from the **Inject the JavaScript Agent configured for this Browser App** dropdown.
2. From the Configure JavaScript Injection tab, expand **Create Injection Rules.**
3. Click the **+** icon to open the **Create Manual Injection Rule** dialog.
4. From the Where to Inject JavaScript tab:
    a. In the **Name** field, enter a name for the rule.
    b. Check **Enable**.
    c. In the **Class and Method to intercept** section, define match conditions for the class and method that write to the output stream in your application. This is the class that server-side agent intercepts for injection.
    d. If the write method is overloaded:
        i. Check the **Is this Method Overloaded?** checkbox.
        ii. Click **Add Parameter.**
        iii. Add the parameters that define the method.
    b. In the **Pointer to the writer** section:
        i. Select how to obtain a reference to the writer object using either the selected method with a configured number of parameters, return value, or invoked object.
        ii. Specify a getter chain.

c. In the **Injection options** section, specify:
   - the output stream write method the server-side agent should use to inject the JavaScript Agent
   - when the injection should occur: when the method begins or when the method ends
   - which part of the script should be injected: the header or the footer
   - optional prefix to output before writing the header or footer, such as `<DOCTYPE. . . >`
5. Click **Create Injection Rule**.

Attribute Injection

To have your server-side application use assisted injection of the JavaScript Agent using attribute injection, you:

- Enable attribute injection
- Copy code snippets into your page template

> Only Servlet containers supported assisted injection.

Access the User Experience App Integration Panel

1. From the **Applications** page, open the business application that you want to automatically inject the JavaScript Agent into your browser application.
2. From the left navigation bar, select **Configuration**.
3. Click **User Experience App Integration.**

Access the JavaScript Injection Configuration Panel

1. From the **User Experience App Integration** page, click the JavaScript Agent Injection tab.
2. In the JavaScript Agent Injection tab, select a browser application from the **Inject the JavaScript Agent configured for this Browser App** dropdown.

Copy Code Snippets into Your Page Template

These examples show code snippets that you can copy directly into your page templates or into other pages. These code snippets direct the app agent where to inject information. The header value must be injected at the very top of the `<head>` section and the footer value must be added at the very end of the code creating the page.

If you have already injected the header portion of the agent using manual injection, you can use these code snippets to automatically inject the footer data portion only. In this case, add only the `JS_FOOTER` values shown in the sections below.

JSF

```
<h:outputText rendered="#{AppDynamics_JS_HEADER != null}" value='#{request.getAttribute
("AppDynamics_JS_HEADER")}' escape="false"/>
<h:outputText rendered="#{AppDynamics_JS_FOOTER != null}" value='#{request.getAttribute
("AppDynamics_JS_FOOTER")}' escape="false"/>
```

JSP

```
<% if (request.getAttribute("AppDynamics_JS_HEADER") != null) { %> <%=request.getAttribute
("AppDynamics_JS_HEADER")%> <% } %>
<% if (request.getAttribute("AppDynamics_JS_FOOTER") != null) { %> <%=request.getAttribute
("AppDynamics_JS_FOOTER")%> <% } %>
```

Servlet

```
if (request.getAttribute("AppDynamics_JS_HEADER") != null)
{
    out.write(request.getAttribute("AppDynamics_JS_HEADER".toString());
}
if (request.getAttribute("AppDynamics_JS_FOOTER") != null)
{
    out.write(request.getAttribute("AppDynamics_JS_FOOTER").toString());
}
```

Groovy

```
<g:if test="${AppDynamics_JS_HEADER}">
    ${AppDynamics_JS_HEADER}
</g:if>

<g:if test="${AppDynamics_JS_FOOTER}">
    ${AppDynamics_JS_FOOTER}
</g:if>
```

Velocity Template

```
#if ($AppDynamics_JS_HEADER)
    $AppDynamics_JS_HEADER
#end
#if ($AppDynamics_JS_FOOTER)
    $AppDynamics_JS_FOOTER
#end
```

ASP.NET C#

```
<% if (Context.Items.Contains("AppDynamics_JS_HEADER"))
    Response.Write(Context.Items["AppDynamics_JS_HEADER"]); %>
<% if (Context.Items.Contains("AppDynamics_JS_FOOTER"))
    Response.Write(Context.Items["AppDynamics_JS_FOOTER"]); %>
```

MVC Razor

```
@if(HttpContext.Current.Items.Contains("AppDynamics_JS_HEADER"))
{ @Html.Raw((string)HttpContext.Current.Items["AppDynamics_JS_HEADER"]) }
@if(HttpContext.Current.Items.Contains("AppDynamics_JS_FOOTER") )
{ @Html.Raw(HttpContext.Current.Items["AppDynamics_JS_FOOTER"].ToString()) }
```

Undo Injection

Related pages:

- Manual Injection of the JavaScript Agent
- Automatic Injection of the JavaScript Agent
- Assisted Injection

If you try one way to inject and it does not work, AppDynamics recommends that you undo the current injection configuration before implementing another one.

Reverse Manual Injection

To undo manual injection, simply delete the JavaScript Agent from your web pages.

Undo Automatic Injection

To undo automatic injection:

1. Navigate to the **Application Dashboard** of the application that is injecting the JavaScript Agent.
2. Click **Configuration > User Experience App Integration**.
3. From the **Inject the JavaScript Agent configured for this Browser App** dropdown, select your browser application.
4. From the Automatic JavaScript Injection tab, clear the **Enable Automatic Injection of JavaScript** check box.

Undo Assisted Injection Using Attribute Injection

To undo assisted injection using attribute injection:

1. Navigate to the **Application Dashboard** of the application that is injecting the JavaScript Agent.
2. Click **Configuration > User Experience App Integration**.
3. From the **Inject the JavaScript Agent configured for this Browser App** dropdown, select your browser application.
4. From the Configure JavaScript Injection tab, clear the **Request Attribute Injection** check box.

Undo Assisted Injection Using Injection Rules

To undo assisted injection using injection rules:

1. Navigate to the **Application Dashboard** of the application that is injecting the JavaScript Agent.
2. Click **Configuration > User Experience App Integration**.
3. From the **Inject the JavaScript Agent configured for this Browser App** dropdown, select your browser application.
4. From the Configure JavaScript Injection tab, double-click the rule that you'd like to disable.
5. Clear the **Enabled** check box.

You can also simply delete rules by selecting the rule and then clicking the **Delete** icon.

Upgrade the JavaScript Agent

Related pages:

- Manual Injection
- Automatic Injection
- Assisted Injection
- Undo Injection

This page covers how to upgrade the JavaScript Agent version. Upgrading to the latest version of the JavaScript Agent will allow you to use the latest features and get the latest bug fixes.

The following sections will discuss possible compatibility issues, show you a new way to include the JavaScript Agent, and provide you with manual injection code examples for the different JavaScript Agent hosting options.

Compatibility Issues

For on-premises deployments, your JavaScript Agent version should the same or older than your EUM Server version. For example, if you have deployed the EUM Server 20.3.0, the latest version of the JavaScript Agent you can use is 20.3.0. We recommend to upgrade your EUM Server to the latest version so you can take advantage of the latest JavaScript Agent features.

- If you are using a SaaS EUM Cloud and want a hosted version of `adrum.js`, AppDynamics recommends you use the `adrum-latest.js` version. However, you can use any version of the JavaScript Agent as it is compatible with any SaaS EUM Cloud version currently being deployed.
- If you are using an on-premises EUM Server and want to use a hosted version of `adrum.js`, you need to match the `adrum.js` version to your EUM Server version.
- For Controller versions, a JavaScript Agent version is compatible with Controller versions with the same or older versions. For example, JavaScript Agent 20.6.0 is compatible with Controller <= 20.6.0.

Injection Code Changes

If the URL to the server hosting your JavaScript Agent is the same for both HTTP and HTTPS, you should no longer use `document.write` to inject the `<script>` tag. Instead, just hardcode the `<script>` tag with the following syntax:

```
<script src="//cdn.appdynamics.com/adrum/adrum.js"></script>
```

Injection Code for Different Hosting Options

The injection code will vary slightly depending on your deployment (SaaS/on-premises). The following provides you with code snippets for each type of deployment. Be sure to place the code snippets below right after the `<head>` tag.

Overview of the Controller UI for Browser RUM

Browser RUM presents information in these views:

- An **Overview** dashboard, with sections for:
  - widgets for graphic display of common metrics
  - map-based performance display
  - snapshots of individual requests
  - usage statistics by browser and device/platform
- Unified data for entire sessions, following an individual user's path as they navigate through your site
- Detailed information on the performance of individual pages, as
  - aggregated lists of page, Ajax, iframe, and virtual page request types
  - multi-faceted data from a complete data store

Access a Browser RUM Application

To see the data for your app in the Controller UI, you must first open your application:

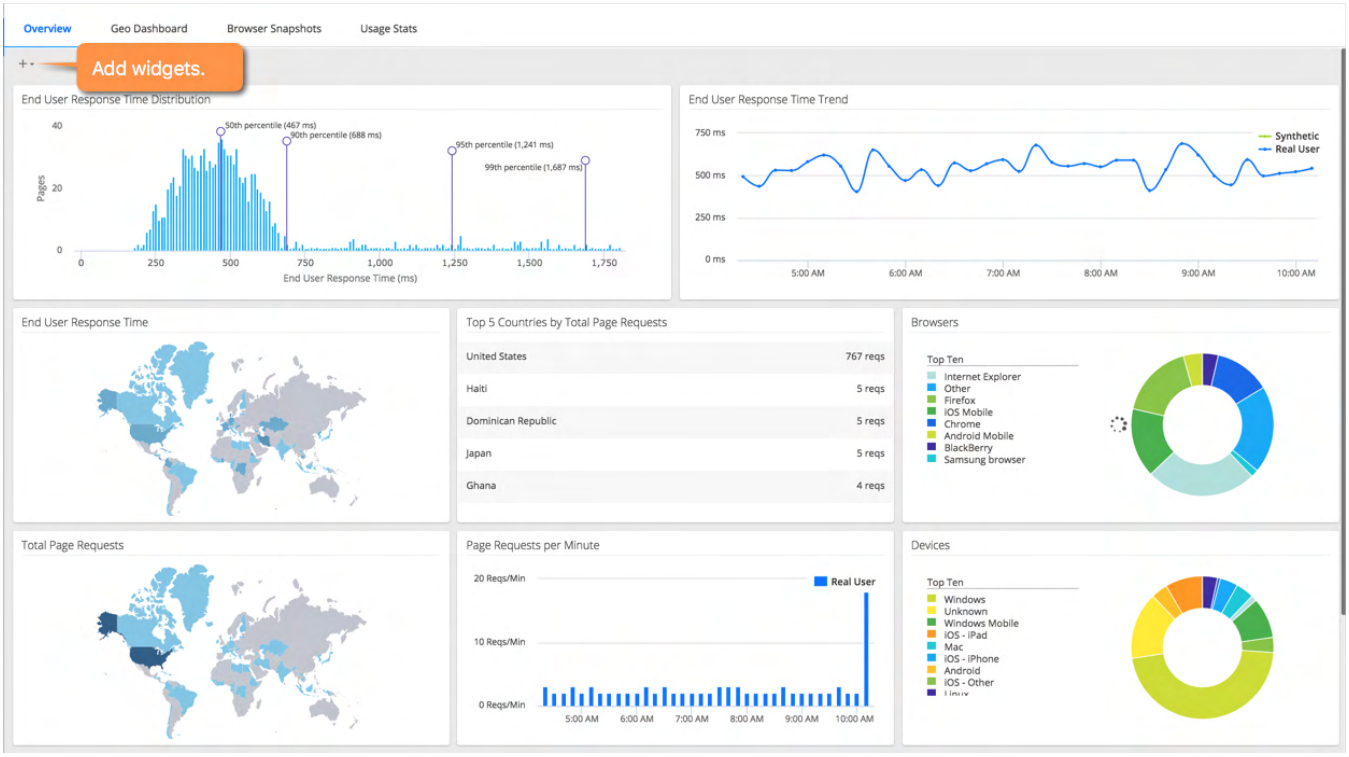
1. In the top tab bar, select **User Experience**. The list of instrumented Browser and Mobile Apps is displayed.
2. Make sure the **Browser App** tab is selected.
3. Double-click on the app you are interested in. The monitoring UI opens.

Browser App Dashboard

This view is good for getting a high-level understanding of how your app is performing overall.

Overview

The **Overview** tab is made up of a set of widgets showing common usage metrics. You can add, delete, move, and resize widgets as you wish. If you see a metric that interests you, click through to the main view. If you are using Synthetic, data from both RUM and Synthetic (separated by color) show up in some of the widgets.

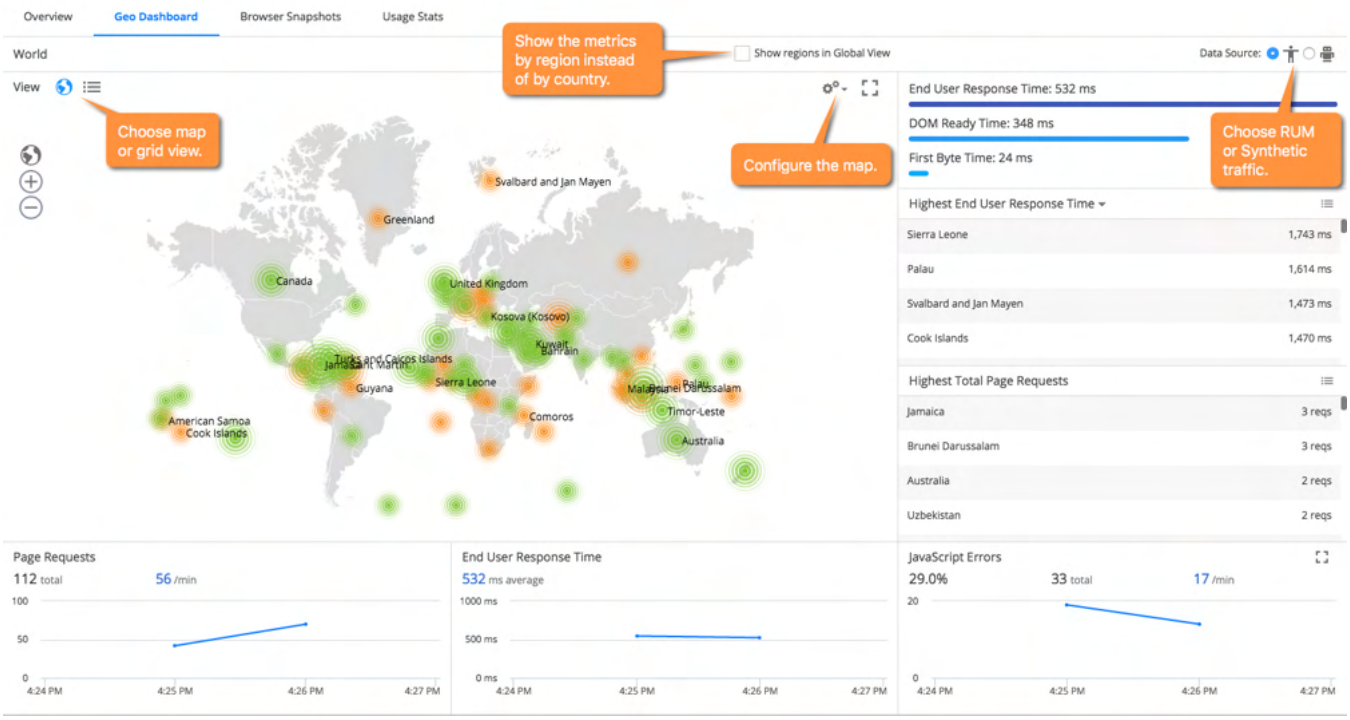


For more information, see Browser App Dashboard.

Browser App Geo Dashboard

The **Geo Dashboard** view provides high-level insight into how your application is performing across the world for your users. The size of a dot indicates the number of page views from that region, and the color indicates the average page load End User Response time in that region (red is slow). You can click to drill down to areas of specific interest. You can also see the same information presented in tabular form by clicking the grid icon in the upper left of the panel.

If you are using Synthetic, you can also select to see either RUM or Synthetic data displayed here. By default, the **Geo Dashboard** displays key performance metrics by country, but you can view performance metrics by region by checking the **Show regions in Global View**.

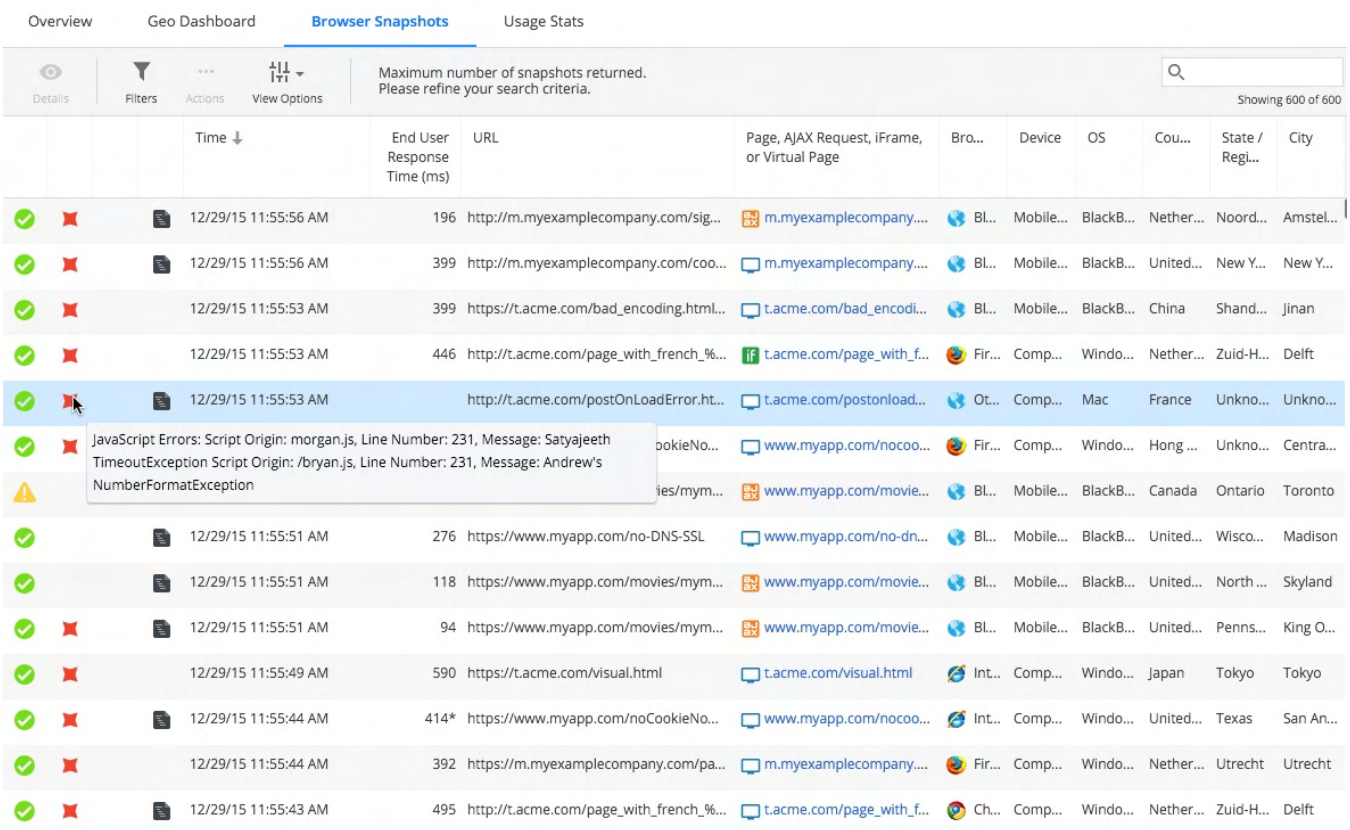


For more information, see Browser App Dashboard.

> The option **View Dark Mode** has been removed from the **Geo Dashboard**.

Browser Snapshots

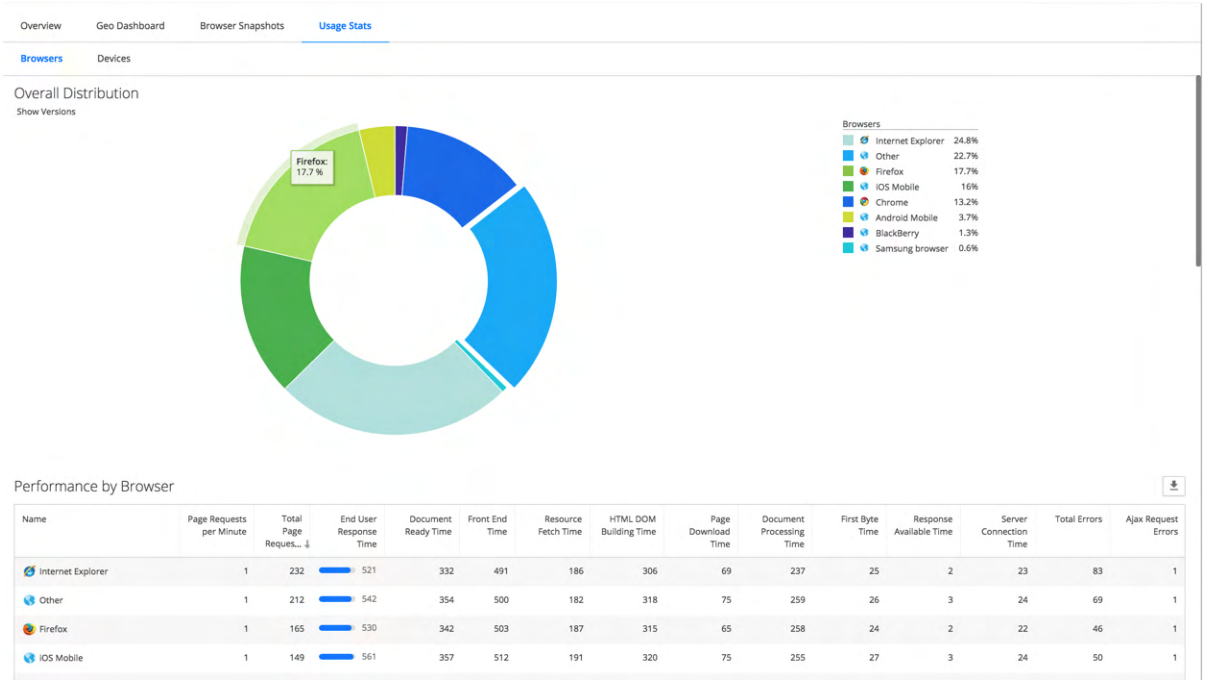
The **Browser Snapshot** view provides access to detailed information for individual requests. The list includes both periodic snapshots of requests operating within normal boundaries and problem snapshots of requests that have exceeded one or more configurable performance criteria. Double-clicking a specific item takes you to a detailed graphical representation of the execution flow of that request and other data associated with it.



For more information, see Browser Snapshots.

Usage Stats

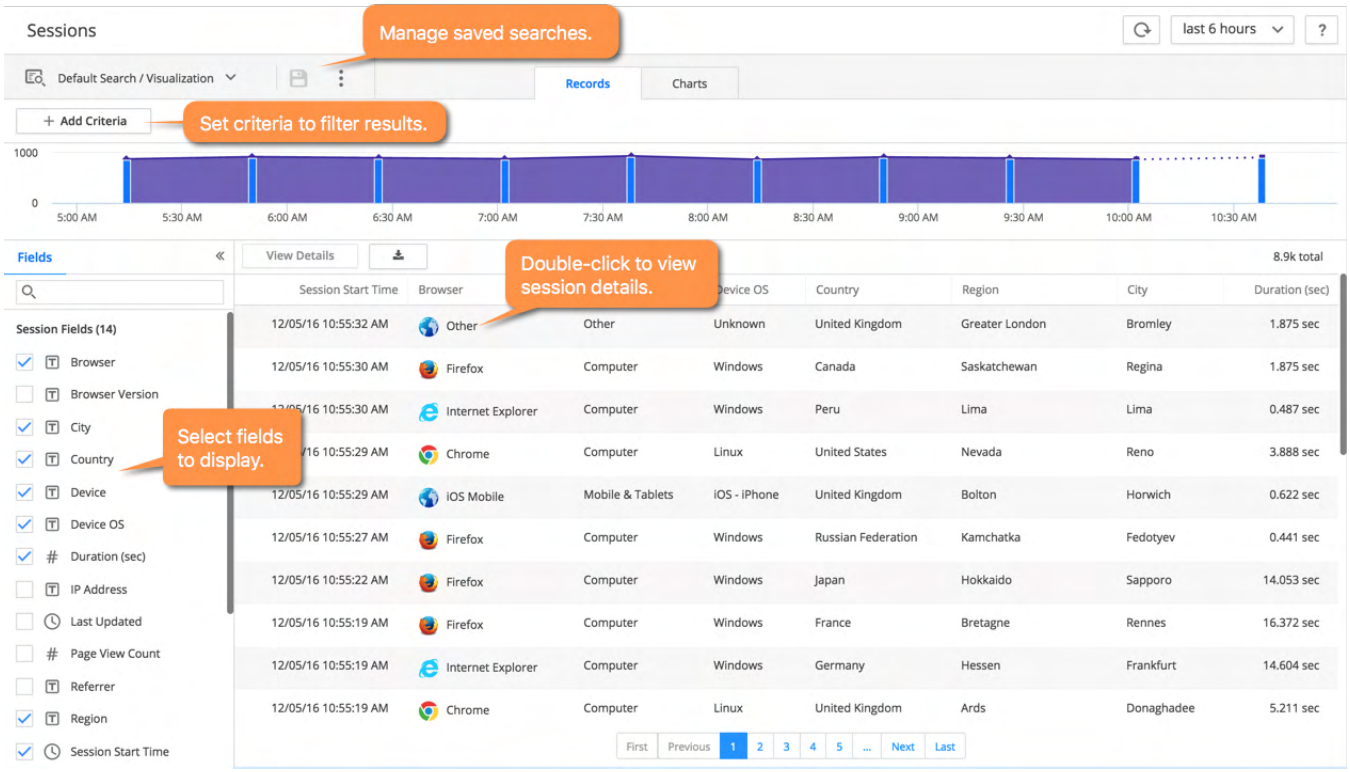
The **Usage Stats** view presents aggregated page load usage data based on the browser type and device/platform employed by your users. The view also breaks down performance by type and usage by country.



For more information, see Browser App Dashboard.

Sessions

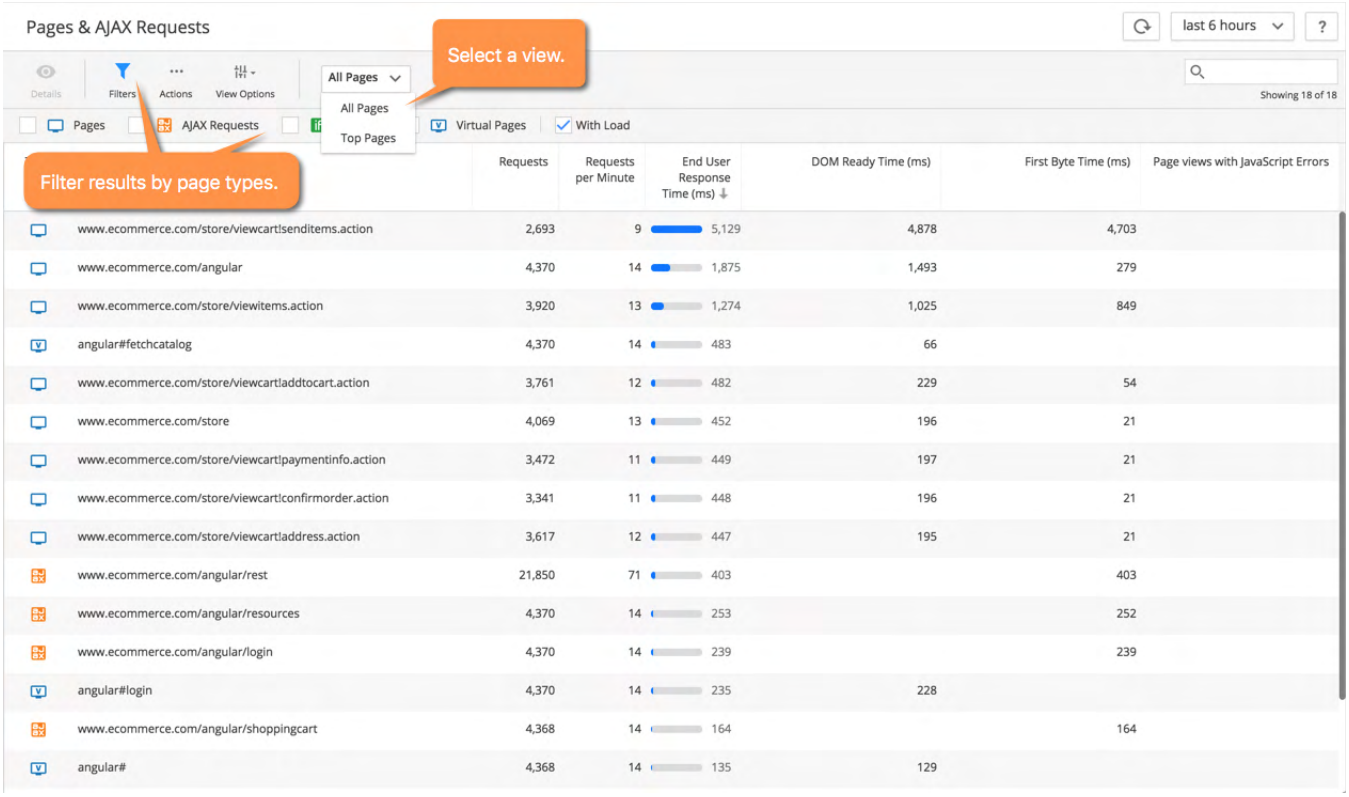
In most cases, users interact with your web application over a series of real (or virtual) pages. Sessions allow you to track your users' interactions across time, as they navigate through an entire engagement period with your app. Using Browser RUM Sessions, you can analyze session results from all requests, as stored in the AppDynamics Events Service. Use the **Records** tab to focus on pertinent data and the **Charts** tab to view a dashboard of useful graphics widgets. Or follow a session from page to page using the **Session Detail** dialog.



For more information, see Browser RUM Sessions.

Pages & Ajax Requests

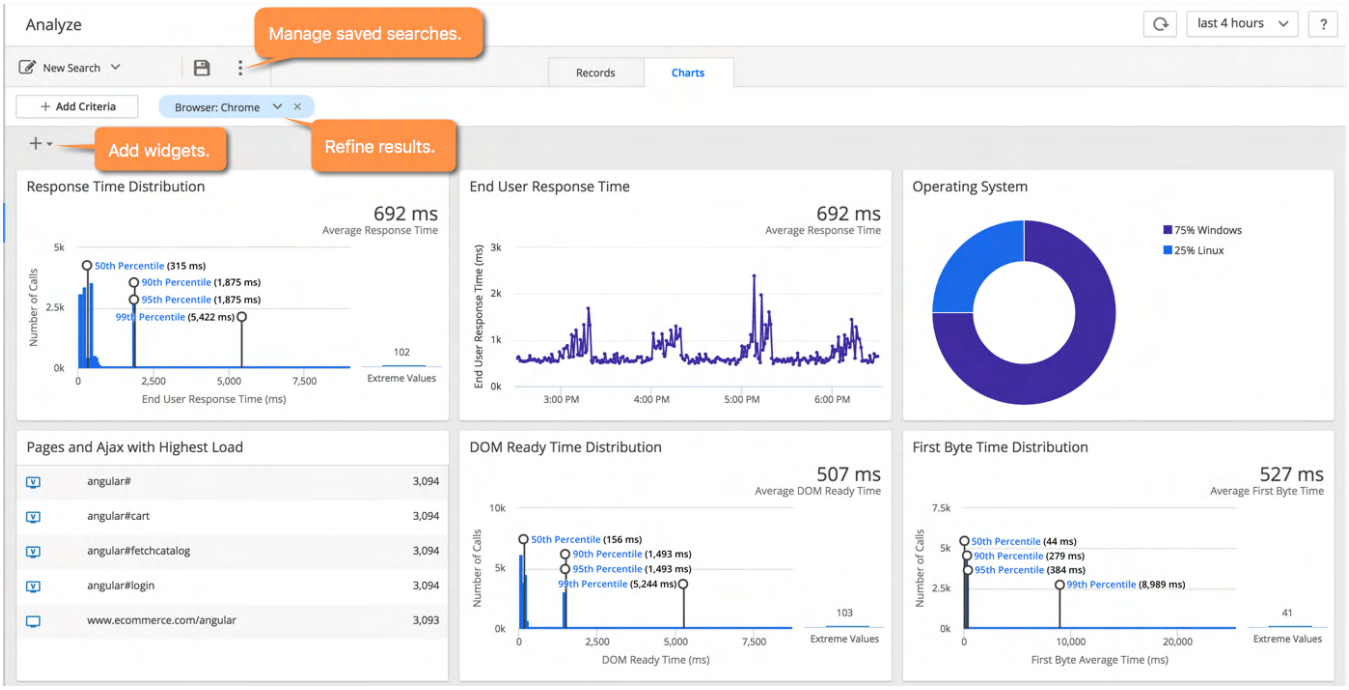
This view is good for tracking the performance of individual pages and components as well as understanding any issues that may be emerging. The **Pages & Ajax Requests** view shows you an aggregated view of how each of your page, Ajax request, iframe, and virtual page types are performing over time. You can look at **All Pages** for an overall sense or select a **Top Pages** view to see the worst performing pages sorted by common metrics like Page views with JavaScript Errors and First Byte Time. And you can drill down to a graphical dashboard showing a wide range of charted performances characteristics for any specific request type.



For more information, see The Pages & Ajax Requests View.

Analyze

The **Analyze** view allows you to query a complete repository of all traffic data that Browser RUM has collected in the past two weeks and display that data in multiple ways.



For more information, see Browser RUM Analyze

Enable Browser RUM

Browser RUM requires a separate license and must be enabled before it is available for use.

For information about licensing, see EUM Accounts, Licenses, and App Keys.

For information on enabling or disabling Web User Experience, see Set Up and Access Browser RUM.

Browser RUM Sessions

This page describes Browser RUM sessions, and the **Session UI** you use to to view session data.

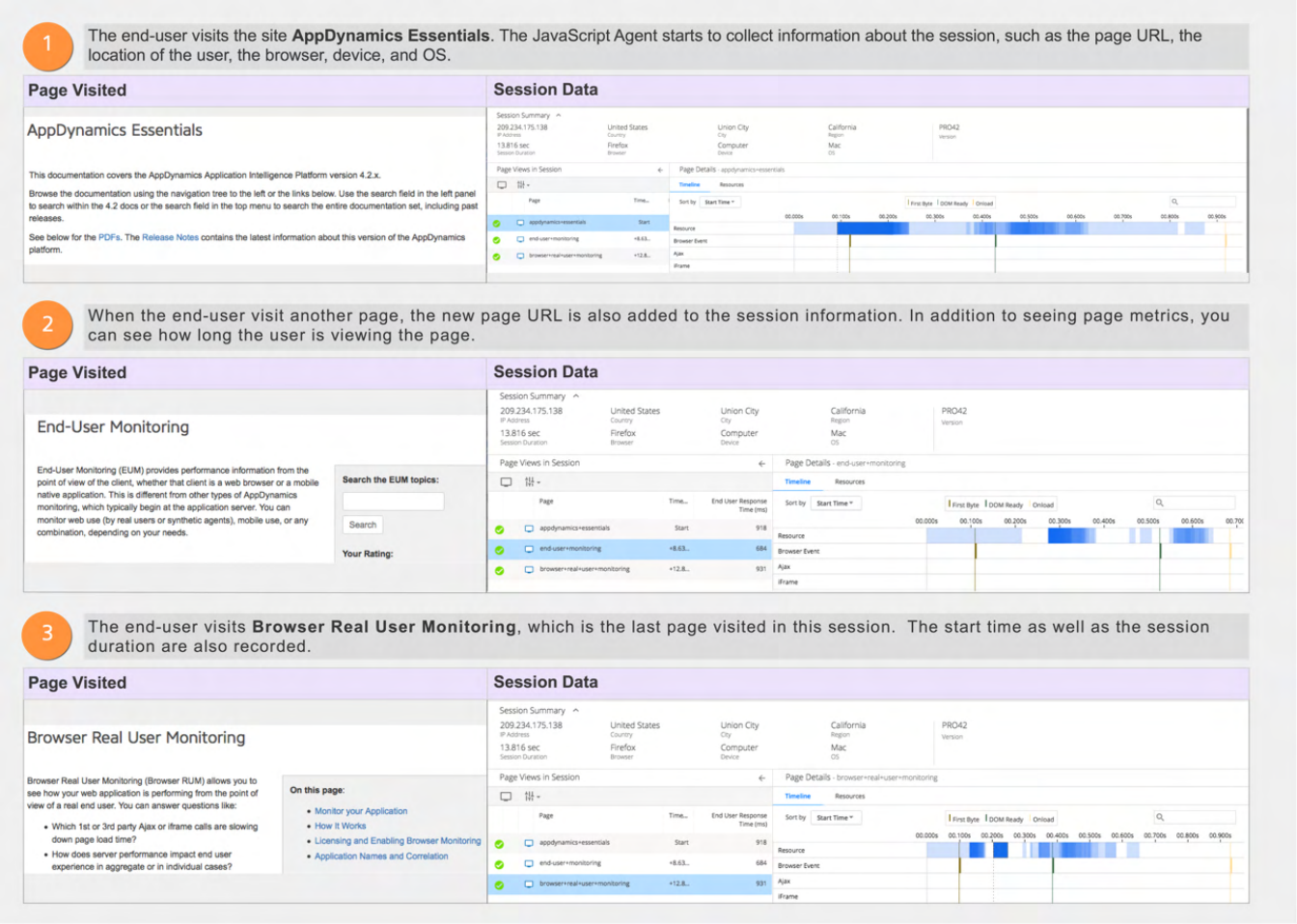
Browser RUM Session

A Browser RUM session is a collection of chronological events for a particular end user from the start of a session to a configurable period of inactivity. An event represents an activity occurring at the moment in time that can be recorded, aggregated, and correlated at certain levels. In Browser RUM, events are limited to page loads, virtual page loads, iFrames, and Ajax requests.

> Only Ajax requests configured to be published to the Events Service are included in sessions.

For example, when a user purchases product from an e-commerce site, the session might capture the page/virtual page loads, iFrames, or Ajax requests for the user logging in (start), selecting and paying for products, and logging out (end).

This figure describes how the user's actions cause events that are tracked in the session.



RUM Sessions Importance

You can think of sessions as a time-based context to analyze a user's experience interacting with an application. By examining RUM sessions, you can understand how your applications are performing and how users are interacting with them. This enables you to better manage and improve your application, whether that means modifying the UI or optimizing performance.

Use Cases

In some use cases, you can use RUM sessions to analyze a user's or a group's behavior. You analyze a user's behavior based on the user's unique ID across multiple sessions in what is called *sniping*. In addition, you can query for a segment of users with similar behavior, such as visiting a certain specific page or using a particular device. This technique of grouping users with similar behavior is called *bucketing*. Browser RUM sessions allow you to query users on a wide array of criteria such as the type of browser, device, location, landing page, IP address, and more.

How EUM Captures RUM Sessions

The JavaScript Agent loads a temporary GUID into the browser local storage and then transmits beacons to the EUM Cloud or EUM Server that include this GUID. We can then use the GUID to track the user's device over the duration of one or more sessions. The EUM Cloud then publishes these events into a session record stored in the Events Service.

Session Persistence

Sessions persist until a period of user inactivity that you configure in the controller. Sessions can also end when the GUID is removed from local storage, which can happen for a number of reasons. For example, the user could clear local storage or the browser is configured to clear the local storage when the browser is closed.

Maximum Session Time

We don't technically limit the time of a session. Instead, we restrict the maximum number of events to 1000. You can, however, configure the maximum time for each event by configuring session monitoring timeouts and then derive the maximum time for a session by multiplying the configured session inactivity timeout by 1000. For example, if you configure the session inactivity timeout for each event to be five minutes, the maximum session time would be 5000 minutes. Of course, in this case, most sessions would not last 5000 minutes because end users would trigger events well before the maximum time of five minutes.

Single Page / Multiple Page Sessions

When the browser cannot write the GUID to local storage, Browser RUM cannot track the user's device over one or more sessions. Instead, Browser RUM uses the temporary GUID in memory to create one-page or multiple *virtual* page sessions. Virtual pages are simply those sections of the DOM that are dynamically created and loaded into the current page.

Privacy/Security of Session Data

We do not perform browser or device fingerprinting. By default, we do not store user IP addresses. If you configure the Controller to store IP addresses, then the IP addresses are stored for up to 14 days.

Browser RUM Support for Sessions

JavaScript Agent Version Requirement for Sessions

You are required to use the JavaScript Agent >= 4.2. Older versions of the JavaScript Agent do not support sessions.

Browser Requirements for Sessions

To use Browser RUM sessions, your browser is required to support for the following:

- cross-origin resource sharing (CORS) for beacons
- local storage for multiple-page sessions (single-page / multiple virtual page sessions don't need local storage)

Browser RUM sessions do not support beacons implemented with GIFs.

Session Limits

The table below lists the limits for session operations per account.

Session Operations	Maximum Per Account
Stored session resource snapshots	10k per minute
Pending uploads of resource timing snapshots	100k
Published session beacons	10k per minute
Tracked sessions	5k per minute

After the pending uploads of resource timing snapshots reach 100k, no further resource-timing snapshots are queued.

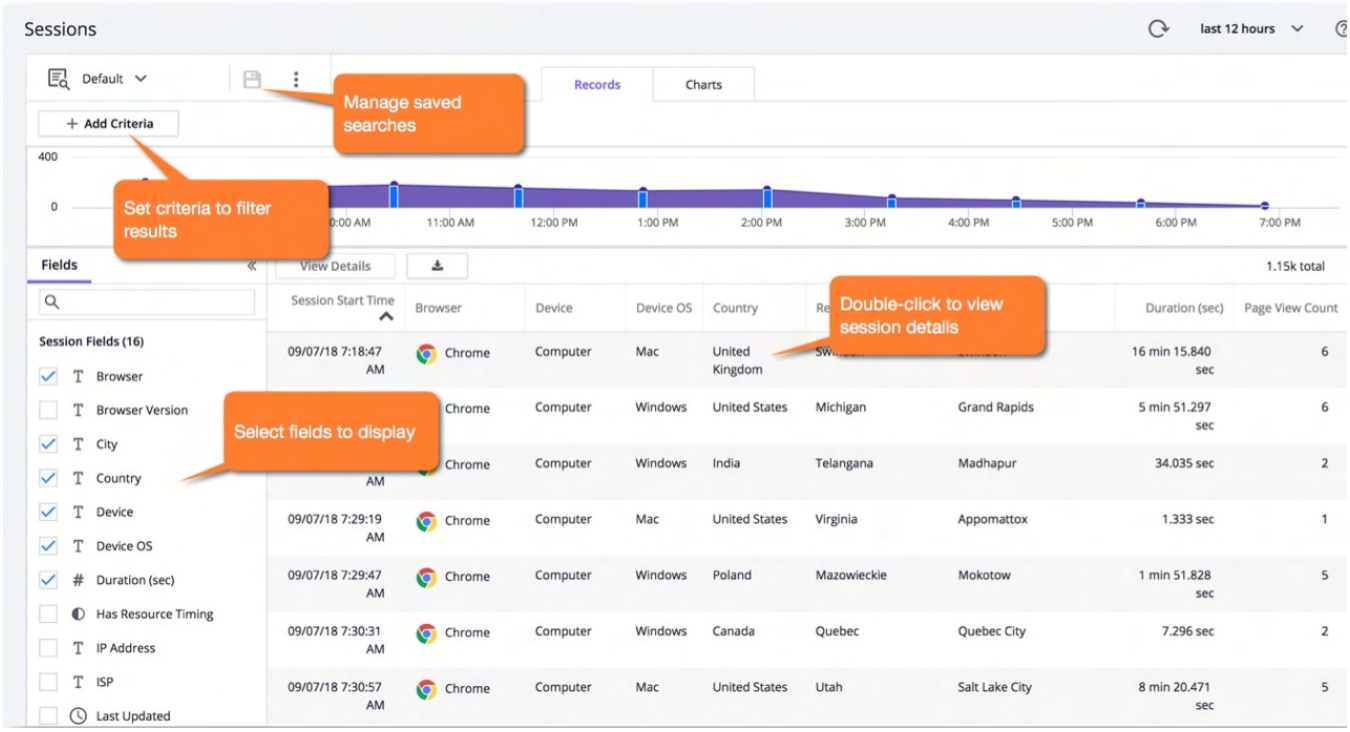
Sessions UI

As in Browser RUM Analyze, the main **Sessions** page is made up of two tabs:

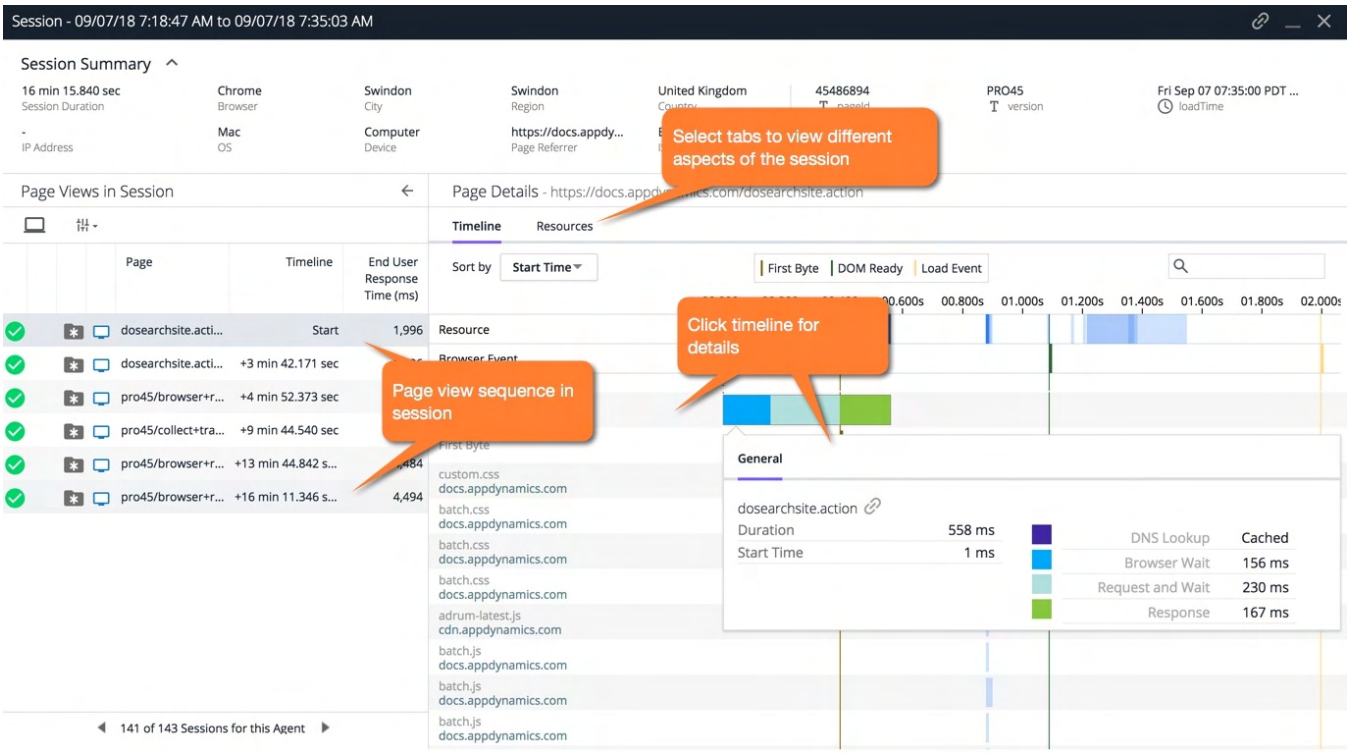
- Records
- Charts

Records

The Records tab lets you scan individual sessions and allows you to filter and sort to get exactly the data in which you are interested.

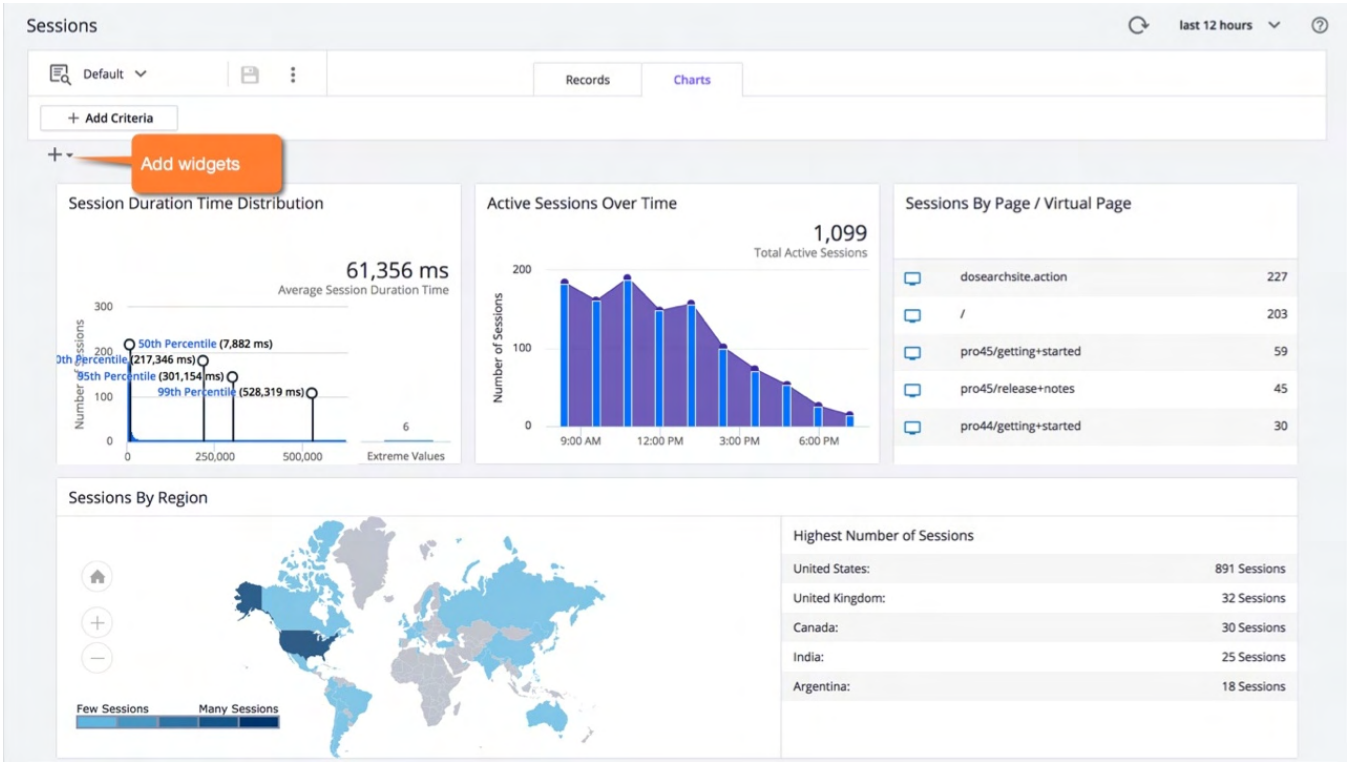


Click **View Details** or double-click an item to see the information for a specific session. The sequence of page views for the session is shown on the left side of the screen. Select a specific page view to see detailed information, including a page load waterfall, tabular details for resources, and, if your backend is instrumented with AppDynamics, correlated server-side business transactions.



Charts

The Charts tab provides you with a set of predefined widgets of the data set you have created plus a custom widget builder. As with the Charts tab of the **A nalyze** UI, you can delete, re-add, resize, and drag-and-drop to move the widgets.



Pages and Ajax Requests

The **Pages & Ajax Requests** page provides detailed information on how your pages, Ajax requests, iframes, and virtual pages perform over time.

Access Pages & Ajax Requests

1. Open the browser application in which you are interested.
2. On the left navigation bar, select **Pages & Ajax Requests**.

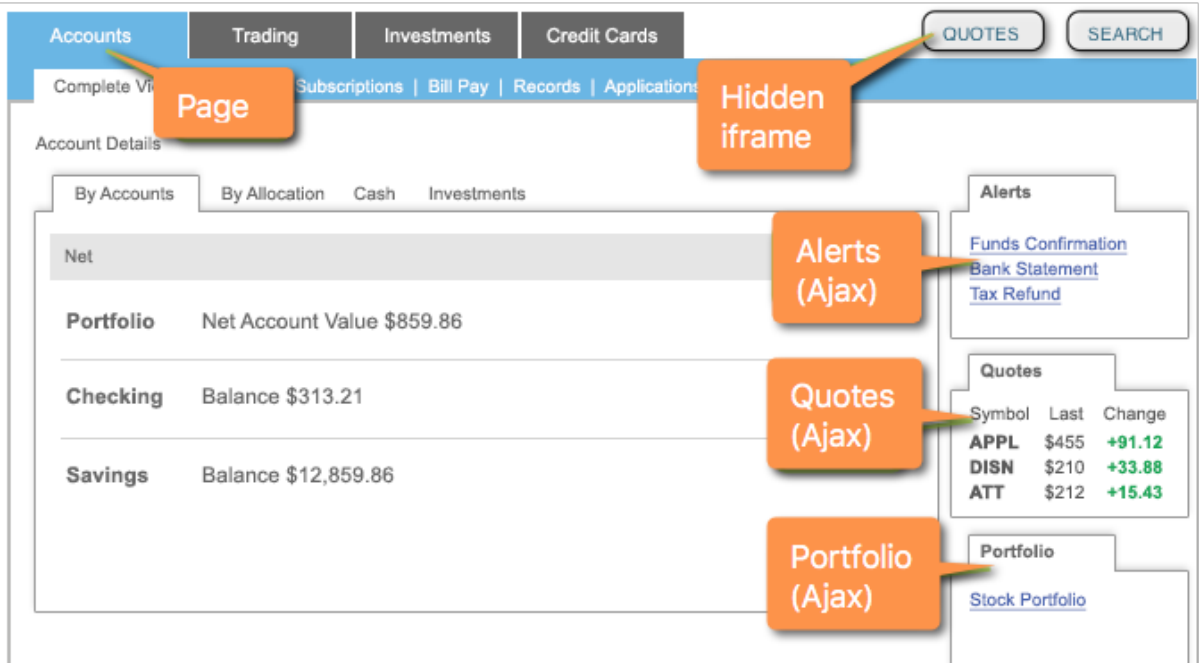
What is a Page?

In Browser RUM, a page represents an individual source for the information that an end-user sees in one browser window.

Types of Pages

A base page is the core HTML page, and may also include one or more iframes, which you can nest.

A base page or an iframe can also make one or more Ajax requests (using `XMLHttpRequest` or the Fetch API) to fetch data to display on the Web page.



You can collect Browser RUM metrics for base pages, iframes, Ajax requests, and virtual pages.

Each base page, iframe, Ajax request, and virtual page type is assigned a unique name, which you can configure. See Configure Page Identification and Naming.

Pages & Ajax Requests

The **Pages & Ajax Requests** page has two options: **All Pages** and **Top Pages**, selected using the dropdown in the upper right. **All Pages** displays a list showing a high-level summary of all the monitored types, along with their key performance indicators. **Top Pages** displays the ten worst performing items grouped by common metrics: Requests per Minute, Page Render Time, and so forth. Use this option for a quick start to troubleshooting.

How the All Pages List is Organized

The **All Pages** list displays each monitored base page, iframe, Ajax request, and virtual page type.

Click a column header to sort the pages based on the column's metric.

To view a dashboard for a specific page item in the list, select it and click **View Dashboard** or just double-click the page. See Page, Ajax, and Iframe Dashboards.

To filter the types of pages displayed in the list, select the type at the top of the list. Check **With Load** to see only pages currently reporting load.

More Actions Menu

Using the **More Actions** menu, you can select a page in the list and make changes to it, including directing the agent to ignore the page and stop reporting metrics for it.

Access Top Pages

Click the **View** dropdown on the upper-right side of the page and select **Top Pages** as a shortcut to troubleshooting the ten worst performing pages regarding various metrics.

Page Limits

There is a limit of 500 total base pages (including virtual pages) plus iframes and 500 Ajax calls that can be individually tracked per application. If your usage exceeds these limits, the Controller begins to drop metrics. If your installation is approaching these limits, you can modify how your metrics are collected by:

- Limiting the number of pages you instrument. If you are using manual injection, remove the JavaScript Agent from pages that are less important. See Inject the JavaScript Agent for more on injection types. If you are using automatic injection, create request match rules and request exclude rules to restrict injection to pages that meet certain criteria. See To Create Match Rules for Automatic Injection.
- Using custom naming rules to group similar pages together. See Configure Page Identification and Naming.

If you reached your page limit, you need to delete the affected old page names before any new page names can be added to the database. Doing so also deletes any accumulated metrics for those page names.

Page, Ajax, and Iframe Dashboards

Dashboards provide quick access to graphic representations of metrics for pages (including virtual pages), iframes, and Ajax request types. Each page, iframe, and Ajax request type has a dashboard. See Page and IFrame Dashboards and Ajax Dashboard.

Access These Dashboards

To view a dashboard for a page, iframe or Ajax request type:

1. Open the browser application in which you are interested.
2. On the left navigation bar of your application, select **Pages & AJAX Requests**.
3. From the list, select the page, iframe or Ajax request type in which you are interested.
4. Either double-click the item, or select the item and then click **Details**.

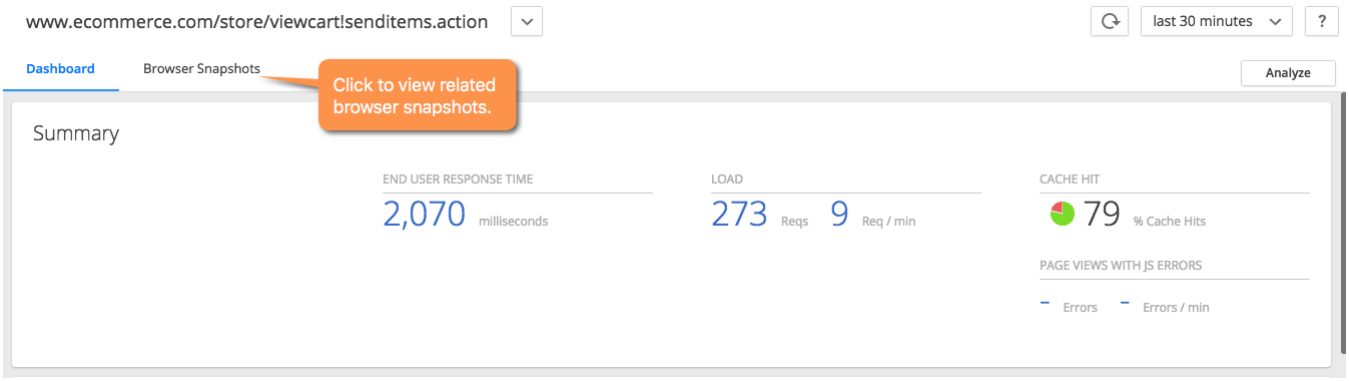
Page and IFrame Dashboards

Page and **iframe** dashboards are divided into six areas:

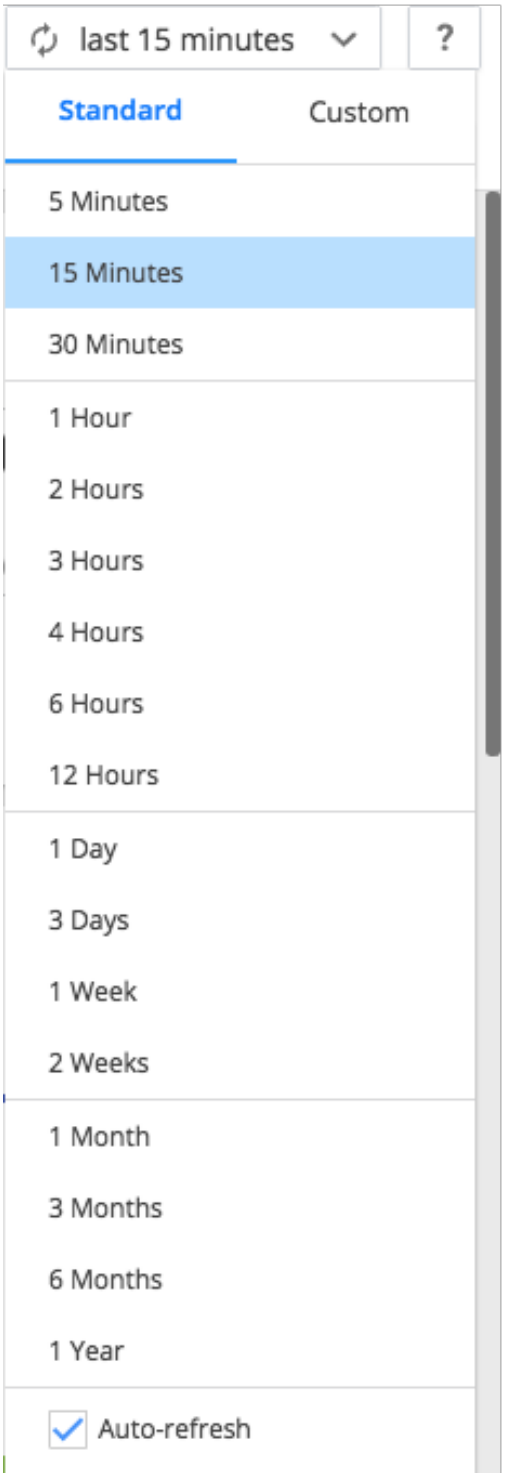
- A summary of key performance indicators. To see browser snapshots associated with these instances of this page type, open the Browser Snapshots tab. To move to the **Analyze** view, click **View Analyze**.
- A waterfall graph of the entire load sequence. To see details for each set of data, use the **Trends/Details** checkboxes.
- Four sections of **Trends/Details** across time for the main performance categories:
    - Overall performance
    - Time between the request and the first byte of the response
    - Time taken by the server to process the request through the completion of the HTML download for the item
    - Time taken to process and render the item, including any external resources, in the browser
- Detailed information on the performance of Ajax requests and iframes for this item.

Page Dashboard Summary Section

This section provides a quick overview of the item's performance over time.

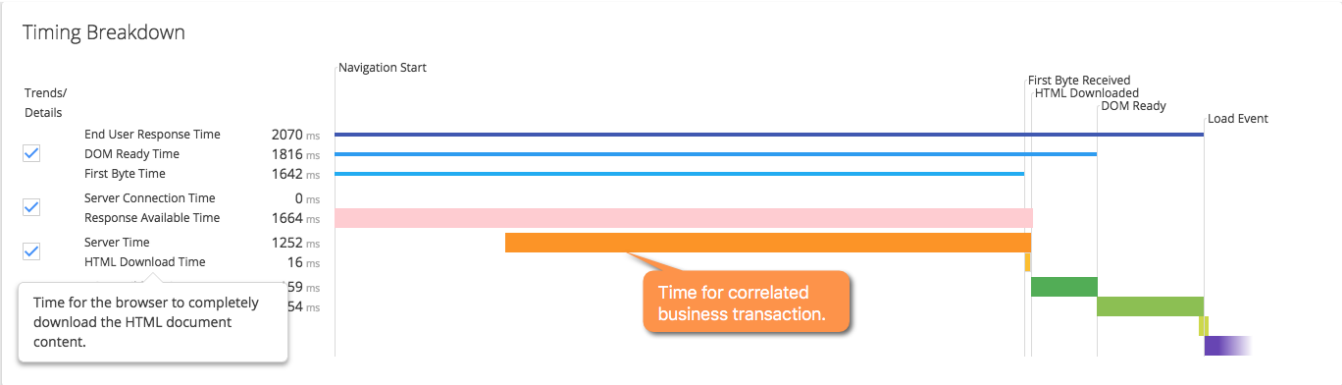


Key performance indicators—End User Response Time, Load, Cache Hits, and Page Views with JS errors—across the period selected in the timeframe dropdown from the upper-right side of the Controller UI are displayed across the top of the summary area. Cache Hits indicates a resource fetched from a cache, such as a CDN, rather than from the source. This metric is only reported when there is a correlated AppDynamics agent on the server-side source.



Timing Breakdown

The waterfall graph shown below displays the average times needed for each aspect of the page load process. For more information on what each of the metrics measures, hover over its name on the left. A popup appears with a definition. See Browser RUM Metrics.



To see detailed breakouts of the data behind the graph, check the **Trend/Detail** checkbox by the data group in which you are interested. To turn the details off, uncheck the checkbox.

> Some metrics—for example, TCP Connection Time—only appear if they have a non-zero value.

Overall Performance

This section displays detailed trend graphs of key performance metrics measured across the selected timeframe. To see detailed information for a particular moment, hover over the graph and a popup with that information appears. To see any of the listed total metrics in the context of the metric browser, click the desired value (shown in link blue) on the left side of the panel. The Metric Browser appears, with that metric displayed. You can then use the **Metric Browser** to compare other related values in one display.

Key Performance Times / Load

This section displays detailed trend graphs of key performance indicators and the request load across the selected timeframe. They measure:

- Time between the user's request and the completion of the page load of the response.
- Time between the user's request and the DOM being loaded.
- Time between the user's request and the browser receiving the first response byte.
- Number of requests per minute.

Server Connect Time Breakdown / Response Available Time

This section displays detailed trend graphs of initial server connection metrics measured across the selected timeframe. They measure:

- Time the user's request takes in negotiating its initial connection with the server, which may include the DNS, TCP Connect, and SSL/TLS time. The **Total Server Connect** value is always displayed.
- Time between that initial connection and the time the first byte of information begins to arrive at the browser.

Server Time / HTML Download Time / Related Business Transactions

This section displays detailed graphs for the time spent acquiring the page data:

- Time the request takes on the server.
- Time it takes to complete the download of the HTML to the page.
- If the request is correlated with a server-side app agent, related business transactions on the server are displayed. Cross-app correlation is supported.

DOM Building Time / Resource Fetch Time / Page Resources Requested

This section displays detailed graphs for the time spent creating the page and getting resources. They measure:

- Time required by the browser to create the DOM from the end of the HTML download.
- Time required to fetch any external resources. For example, the results of a third-party Ajax request.

The **Page Resources Requested** section displays detailed graphs of when in the page load cycle individual external—first and third-party—resources are fetched, and how much time is taken to fetch them, based on the selected timeframe. They measure:

- Average time and load associated with that resource
- If the request is blocking or non-blocking

- Request and response time per resource request
- Type—iframe or Ajax—of the resource

To see the dashboard for any of the listed resources, click the name.

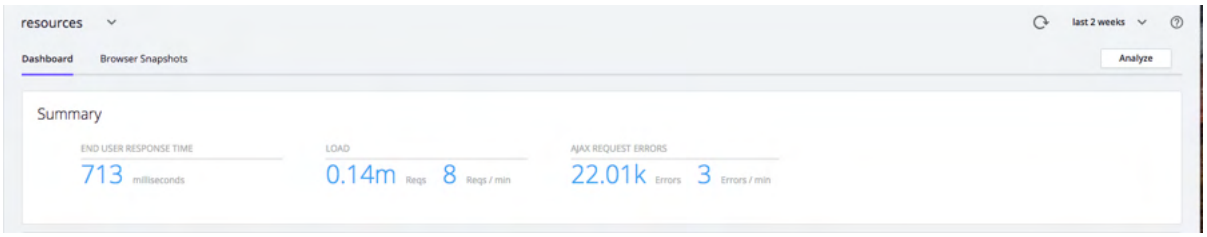
Ajax Dashboard

The **Ajax Dashboard** is divided into three areas:

- A summary, with a waterfall graph of the entire load sequence. To see details for each set of data, use the **Trends/Details** checkboxes.
- Two sections of **Trends/Details** across time for the main performance categories:
  - Overall performance
  - Time used by the server to process the request through the browser's incorporation of the data into the HTML document

Ajax Dashboard Summary Section

This section provides a quick overview of the item's performance over time.



Key performance indicators—End User Response Time, Load, Cache Hits and Views with Errors—across the period selected in the timeframe dropdown from the upper-right side of the GUI—are displayed across the top of the summary area.

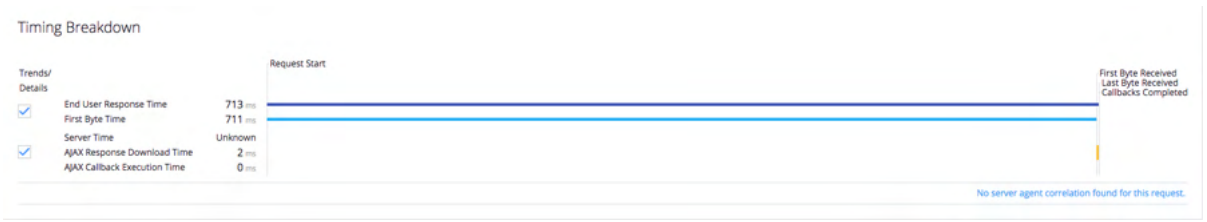
A waterfall graph displays the average times needed for each aspect of the Ajax request load process.

For more information on what each of the metrics measures, hover over its name on the left side of the graph. A popup appears with a definition. See the graphic above for an illustration. See Browser RUM Metrics.

To see detailed breakouts of the data behind the graph, check the **Trend/Detail** checkbox by the data group in which you are interested.

Timing Breakdown

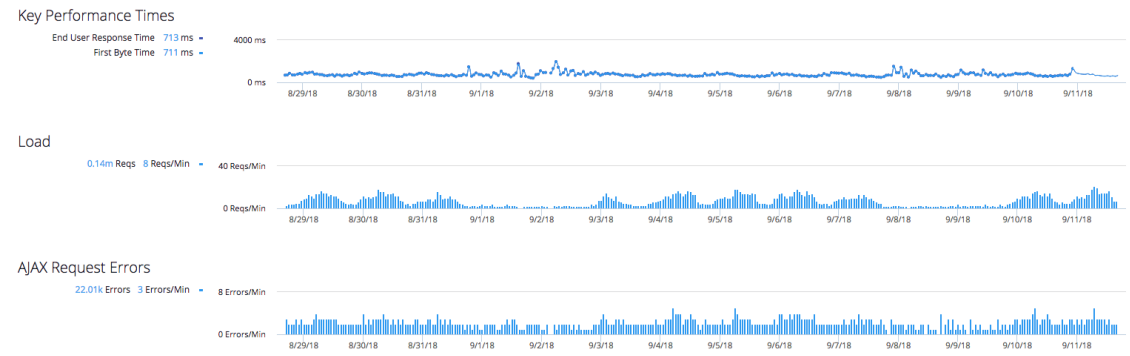
A waterfall graph shown below displays the average times needed for each aspect of the Ajax request. For more information on what each of the metrics measures, hover over its name on the left. A popup appears with a definition. See Browser RUM Metrics.



To see detailed breakouts of the data behind the graph, check the **Trend/Detail** checkbox by the data group in which you are interested. To turn off details, uncheck the checkbox.

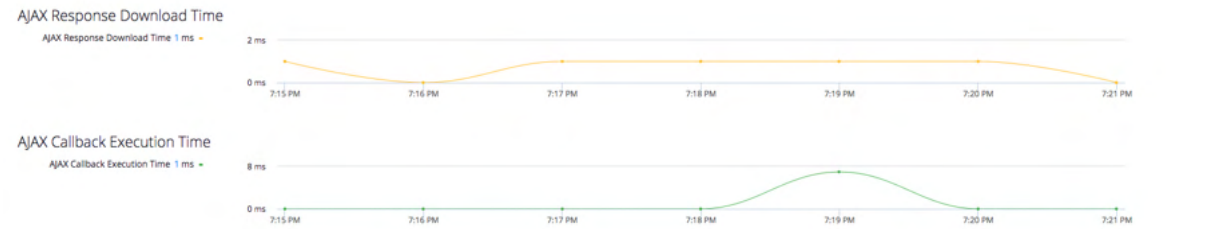
Overall Performance

This section displays detailed trend graphs of key performance metrics, load requests, and Ajax request errors measured across the selected timeframe. To see detailed information for a particular moment, hover over the graph and a popup with that information appears. To see any of the listed total metrics in the context of the metric browser, click the desired value (shown in link-blue) on the left side of the panel. The metric browser appears, with that metric displayed. You can then use the metric browser to compare other related values in one display.



Ajax Response Download and Ajax Callback Execution Time

This section displays detailed trend graphs of the Ajax response download time and the Ajax callback execution time. To see detailed information for a particular moment, hover over the graph to view a popup.



These metrics measure the total time for processing all server-side business transactions for this item.

- Total time for the browser to completely download all of the Ajax responses
- Time for the browser to complete any Ajax callbacks

Browser RUM Analyze

Related pages:

- Analytics Browser Requests Data
- Analytics Browser Sessions Data

Browser RUM collects data on every load event that you have instrumented, and it also takes detailed snapshots periodically and when performance issues are detected. Over time, the load event metrics are rolled up based on averages and percentiles. Sometimes you want to see results based on *all* of the data.

With **Browser Analyze**, every event is collected and stored for a license specified period in the AppDynamics Platform Events Service. Using the Analyze tab you can see results based on this cumulative data.

The **Analyze** view is comprised of two tabs:

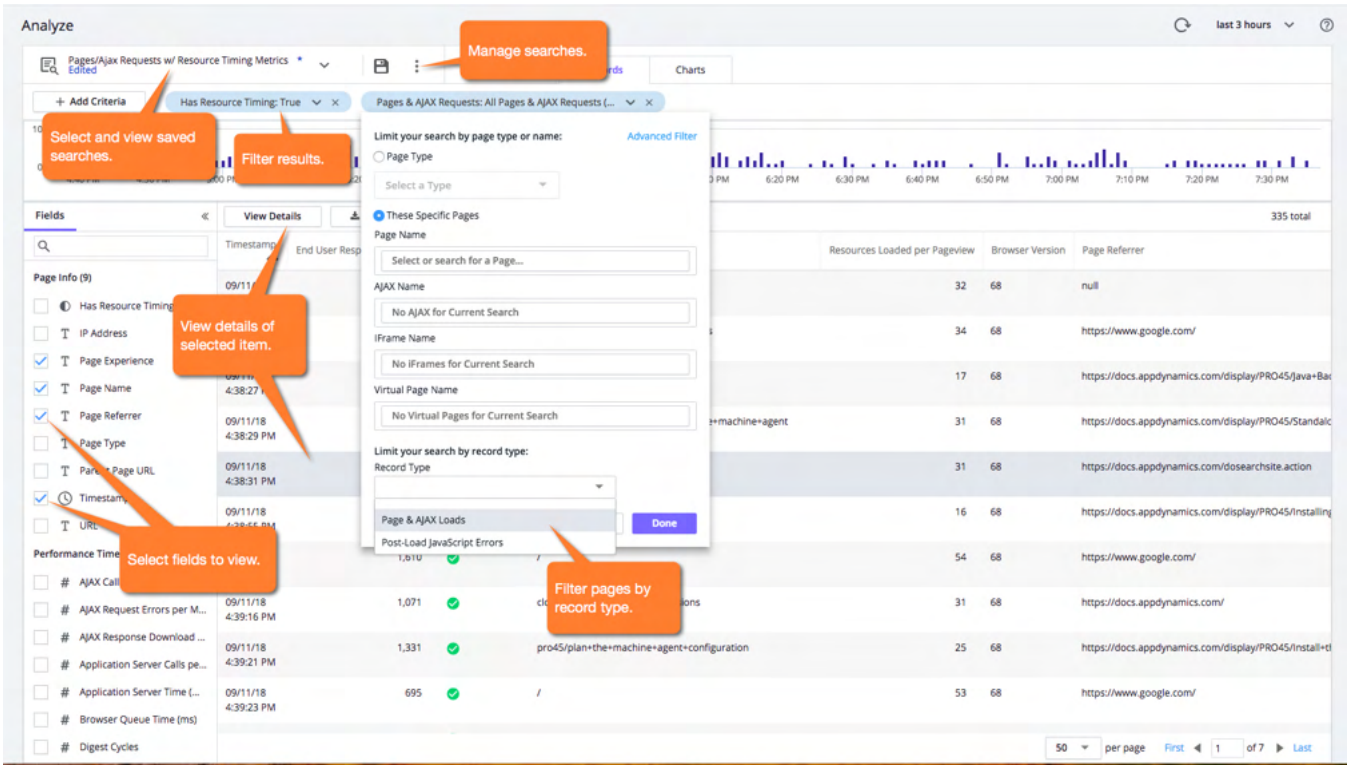
- Records
- Charts

Access the Browser Analyze View

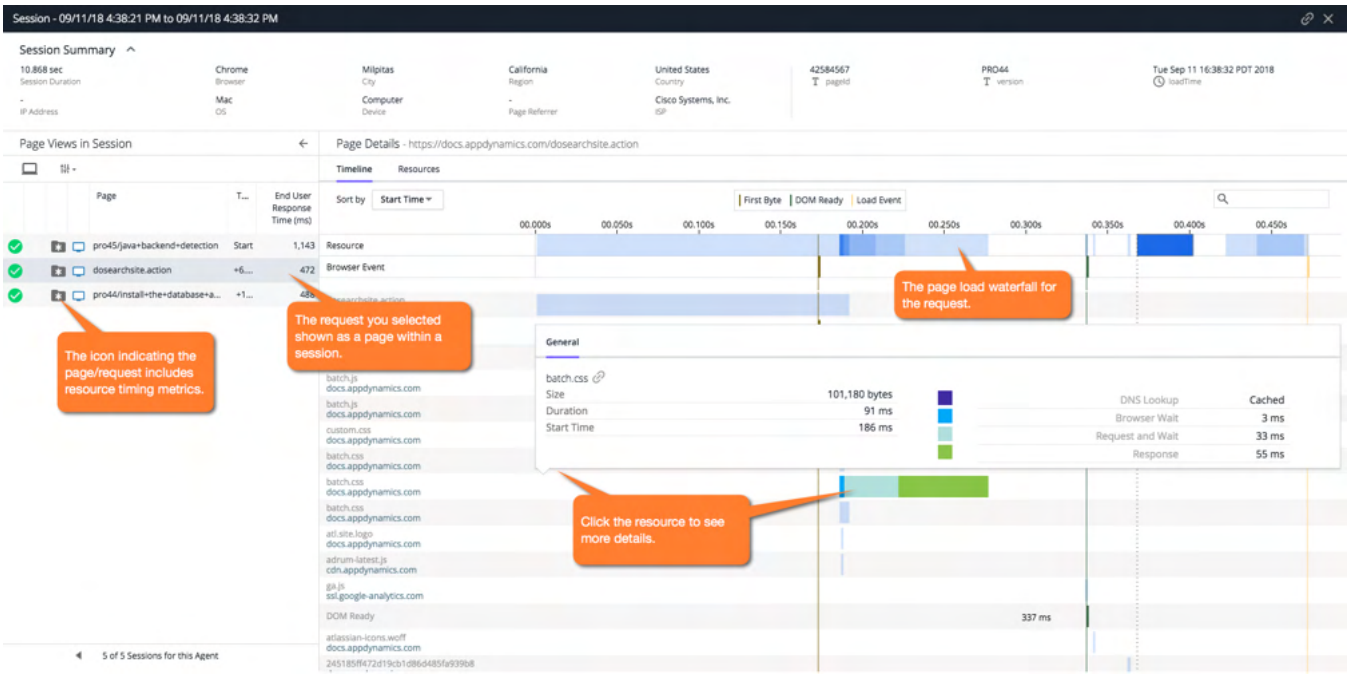
1. Open the application in which you are interested.
2. On the left navigation bar, select **Analyze**.

Records

The Records tab lets you scan individual events and filter and sort to get exactly the data set in which you are interested.



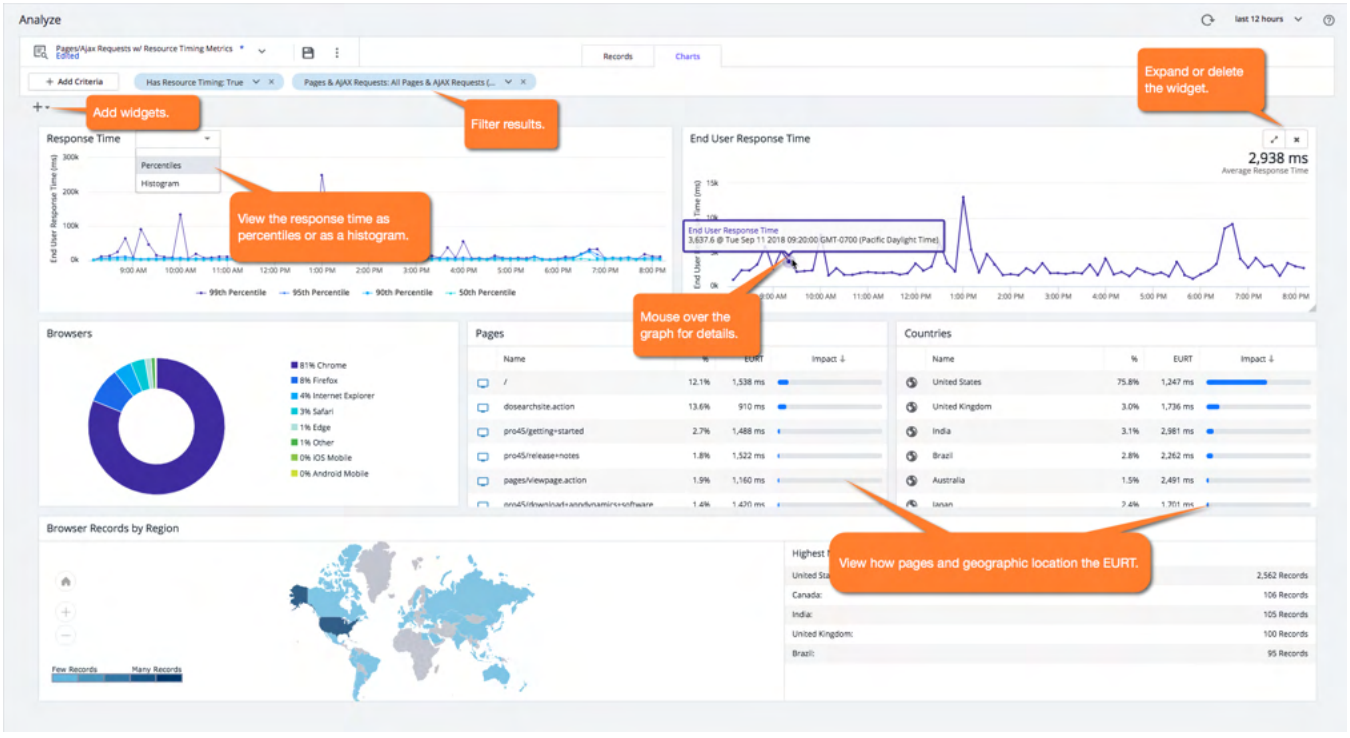
After you click **View Details**, you see the details of a particular request as a page with a browser session in the **Session Details** dialog.



Charts

The Charts tab provides you with a set of predefined widgets that offer visualizations of the data set you have created.

You can quickly see how pages, browsers, and geographic location (such as, dimensions) affect the end user response time (EURT). You can also delete, re-add, resize, and drag-and-drop to move all of the widgets.



The time series widgets such as **Response Time** and **End User Response Time** use median values, not averages.

Request Number Differences in Browser Analyze and Pages & AJAX Requests

The number of requests for a page over a period of time should be the same in **Pages & AJAX Requests** and **Browser Analyze**, but they may vary for these reasons:

- If there are fewer requests given in **Browser Analyze**, the Events Service, which is responsible for the data in **Browser Analyze**, has probably not received all of the records yet. To verify this, compare the time of the last request for the page in **Browser Analyze** and **Pages & AJAX Requests**. The latter will most likely have a later timestamp.
- If there are more requests given in **Browser Analyze**, it's possible that the reason is because error records are included in the total number of requests in **Pages & AJAX Requests**, but listed separately in **Browser Analyze**. The errors are listed separately in **Browser Analyze** to enable you to investigate and understand the cause of the error.

Browser Analyze versus Browser Request Analytics

The data shown on the **Analyze** page is processed and stored by the AppDynamics Platform Events Service. The **Charts** tab displays a set number of widely used chart types to let you explore your application's performance. A separate product, AppDynamics Application Analytics, has a component called Browser Analytics. This component is based on the same Events Service and uses the same data, but it offers additional capabilities, including:

- Additional predefined widgets, such as the funnel widget
- ADQL for searching the data
- Creating custom widgets
- Manipulating multiple dashboard types
- Longer retention time for data storage

Browser Analytics requires a license separate from the Browser RUM license.

To reduce noise in data stored in the Event Service, Ajax calls from the following tracker domains are no longer published to the Events Service:

- `.mixpanel.com`
- `.google-analytics.com`
- `.altocloud.com`
- `.optimizely.com`
- `inspectlet.com`

Configure the Controller UI for Browser RUM

You can enable or disable Browser RUM with the **Browser Monitoring** toggle.

You can manage how Browser RUM information is displayed in the Controller UI, including:

- The display names for your pages, Ajax requests, and iframes
- The display names for your virtual pages
- The errors that should be shown in the UI, and the ones that should not be shown

You can also configure:

- The thresholds for slow, very slow, and stalled transactions
- When browser snapshots should be taken
- Percentile levels you would like to display, if any
- Which Ajax requests should be sent to the Event Service
- The session timeout period
- Whether to store request IP addresses or not

In addition, you can do the following:

- Choose a hosting option for the JavaScript Agent

To configure Browser RUM from the Controller UI, your user account must belong to a role that has the **Configure EUM** permission. See End User Monitoring Permissions for more information.

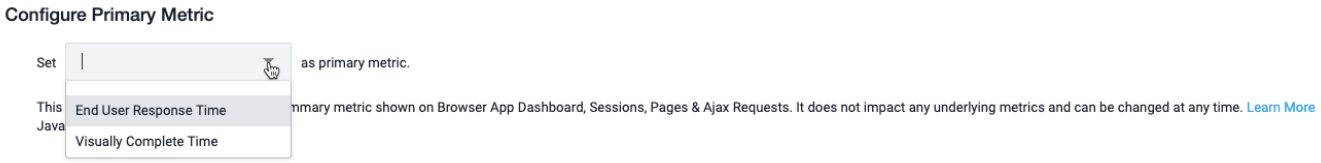
Access the Browser RUM Instrumentation Configuration

To access the instrumentation configuration for a browser app:

1. Open the browser application in which you are interested.
2. From the left-hand navigation menu, click **Configuration**.
3. From the **Configuration** page, click **Instrumentation >**.

Configure the Primary Metric

The primary metric is a way to customize your browser application widgets. The default primary metric is set to End User Response Time (EURT), but you can choose between EURT and Visually Complete Time (VCT). Once the primary metric is configured, the various dashboards, summaries, and charts will automatically update to show trending data for that configured metric. This allows you to view the EUM event timing that is most relevant to your business needs.



To configure the primary metric, go to **Configuration** > **Instrumentation** > **Settings**.

Available Metrics

You can set the primary metric to EURT or VCT. The default primary metric is EURT. EURT tracks the total amount of time for all content on a web page to be loaded. VCT tracks the total amount of time for all visual content to be loaded in the viewport. To see how EURT and VCT are calculated for SPA pages, see SPA2 Metrics.

Primary Metric Visibility

The primary metric is visible across several dashboards, charts, and summaries. The following screenshots are examples of primary metric visibility when configured to VCT.

Configure Page Identification and Naming

Related pages:

- Configure Virtual Page Naming
- Page, Ajax, and Iframe Dashboards
- Set Custom Page Names

You can configure the display names by which various pages and iframes are referred to and sorted in Controller lists and dashboards.

You can:

- Use the AppDynamics default naming rule, which you can leave as is or modify.
- Create custom include rules to override the default convention.
- Create custom exclude rules to exclude from monitoring pages that meet certain criteria.
- Disable the default naming configuration and use only your custom include rule(s).

On this page, the term "pages" includes iframes and base pages.

No matter how the page is named, AppDynamics always reports the page name in lowercase.

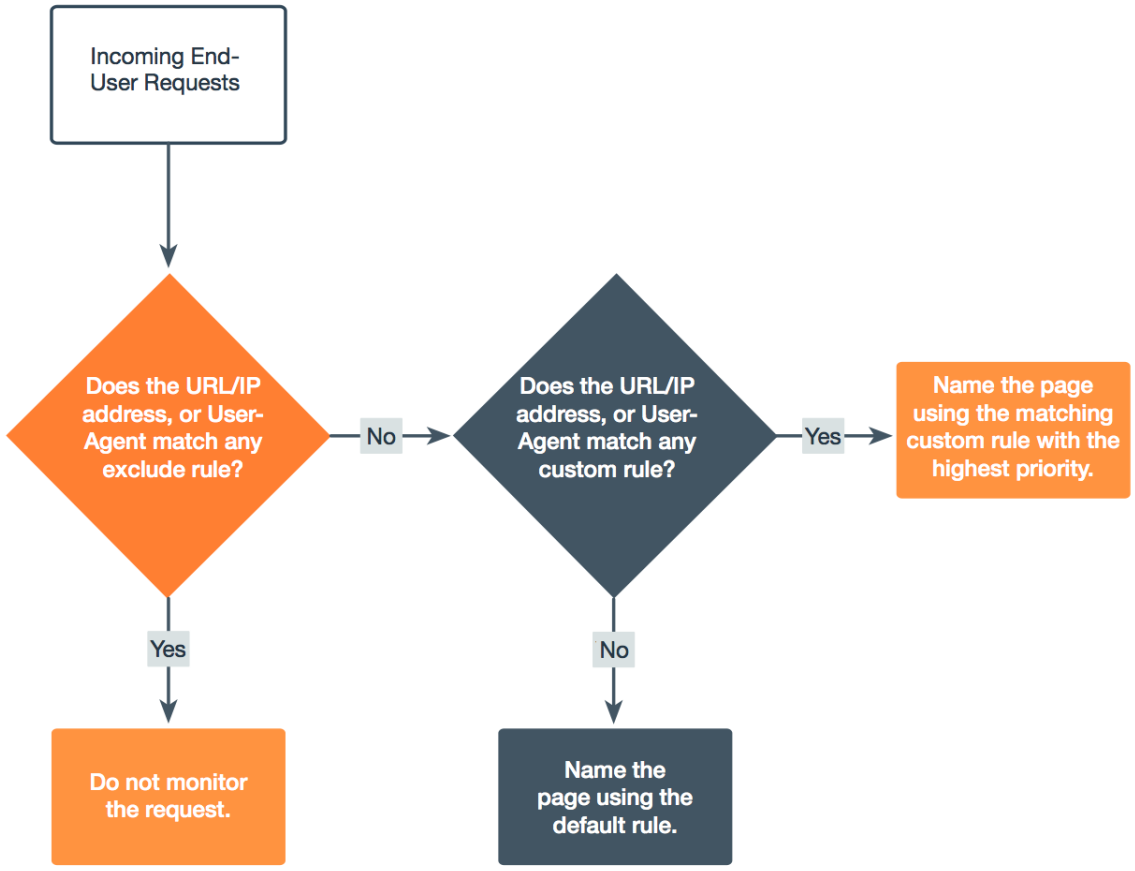
Access Exclude/Include Rules

1. Open the browser application in which you are interested.
2. On the left navigation bar, click **Configuration.**
3. Click **Instrumentation >**.
4. From the Base Page and iFrames tab, you can define include/exclude rules or modify the **Default Naming Configuration**.

To save any configuration changes, click **Save**.

Evaluation Logic for Exclude/Include Rules

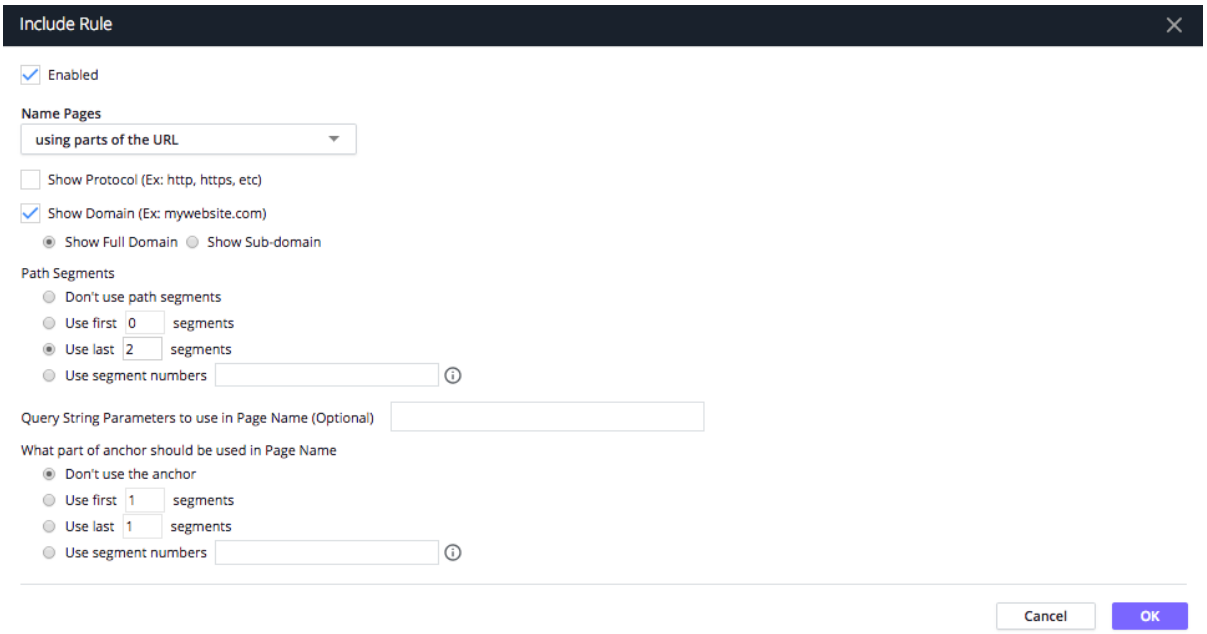
This is the order in which AppDynamics evaluates the page naming rules.



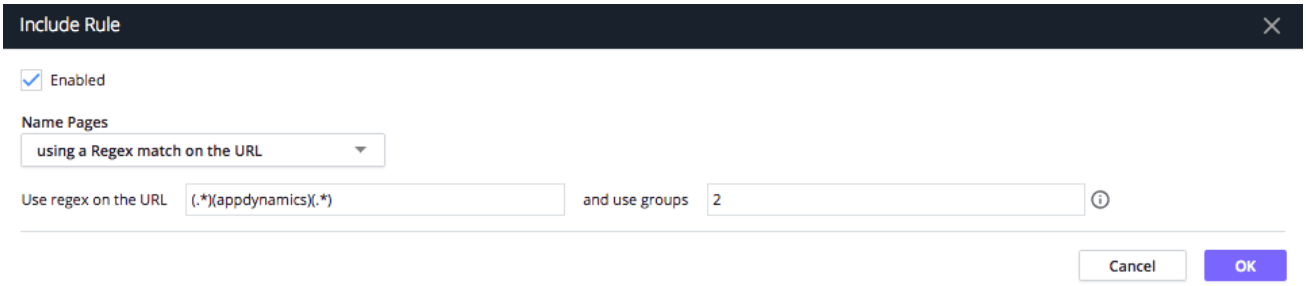
Default Page Naming Rules

If you enable the default naming configuration and do not modify it, AppDynamics identifies and names your pages using the first two segments of the page URL.

You can modify the default configuration by double-clicking **Default Naming Configuration** in the **Include Rules** section. This opens the **Include Rule** popup, where you can select the dropdown list **using parts of the URL** to include the protocol or domain in the name, use different segments of the URL, or include query parameters or anchors in the name.



You can also choose the dropdown **using a Regex match on the URL** to match page names by running a regular expression.



If you do not want to use the default convention at all, disable it by clearing the **Enabled** checkbox. In this case, you must configure at least one custom page naming rule so that AppDynamics can identify and name pages.

Custom Include Rules

You can create custom include rules for identifying and naming pages.

To create a custom include naming rule, from the **Include Rules** section, click **Add**. Then you use the same **Include Rule** popup to configure the custom include rule to identify and name the page.

The order of the include rules in the list determines the priority of the rules: The rules at the top are evaluated first, and the rules at the bottom are evaluated last. You cannot change the priority of **Default Naming Configuration**, but you can modify or disable it.

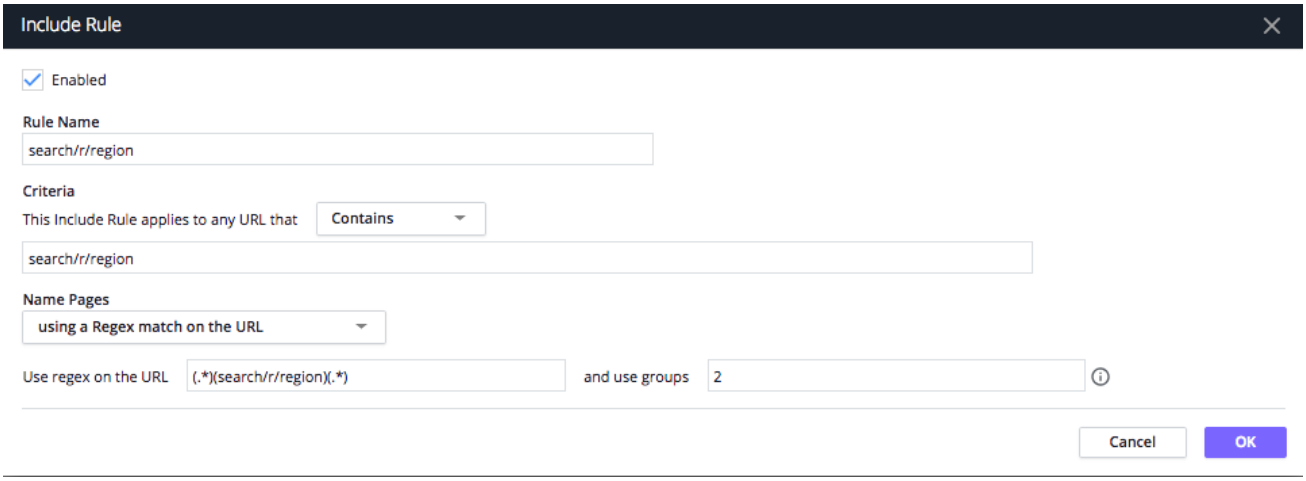
Guidelines for Using Regular Expressions in Include Rules

When using regular expressions to match URLs in your include rules, you should note the following:

- URL strings are case-sensitive. So, although the page names displayed in Pages & Ajax Requests are converted to lowercase, your regular expressions still need to match the case used in URLs that your include rule are trying to capture.
- Your regular expression should match the entire URL, from beginning to end, not just a section; otherwise, the rule will not match. This differs from using regular expressions in custom match rules for naming transactions, which only need to match sections of the URL.
- Beacons for older browsers (primarily Internet Explorer 6-8) are sent using image requests, and therefore, have an inherent length limitation. To manage this limitation, Browser RUM does not support URLs longer than 180 characters, page names longer than 50 characters, and user data longer than 128 characters for these browsers.

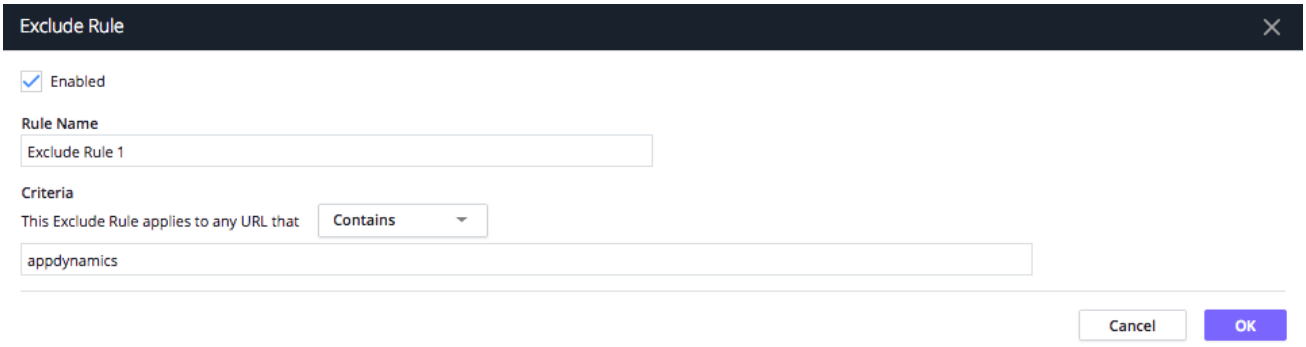
Example Include Rule

Suppose you have multiple pages that include the string `search/r/region` in their URLs, such as `search/r/region01`, `search/r/region23`, etc., and you want to name all the pages from that set as a single-page named `search/r/region`. By selecting the dropdown **using a Regex match on the URL**, you can enter a regular expression to remove the domain name and the number at the end of the URL, grouping all your `/search/r/region` URLs into one set. Because all the URLs contain `search/r/region`, AppDynamics now collects information for them all under the single-page name `search /r/region`. Otherwise, AppDynamics uses the default page naming rule or a rule with higher priority.



Custom Page Exclude Rules

You can configure custom exclude rules for pages in the same way you configure include rules. Any page with a URL matching the configuration is excluded from monitoring.



Configure Naming for Ajax Requests

Related pages:

- The Pages & Ajax Requests View
- Configure Page Identification and Naming
- Page, Ajax, and Iframe Dashboards
- Set Custom Page Names

You can configure the display names by which Ajax requests are referred to and sorted in controller lists and dashboards.

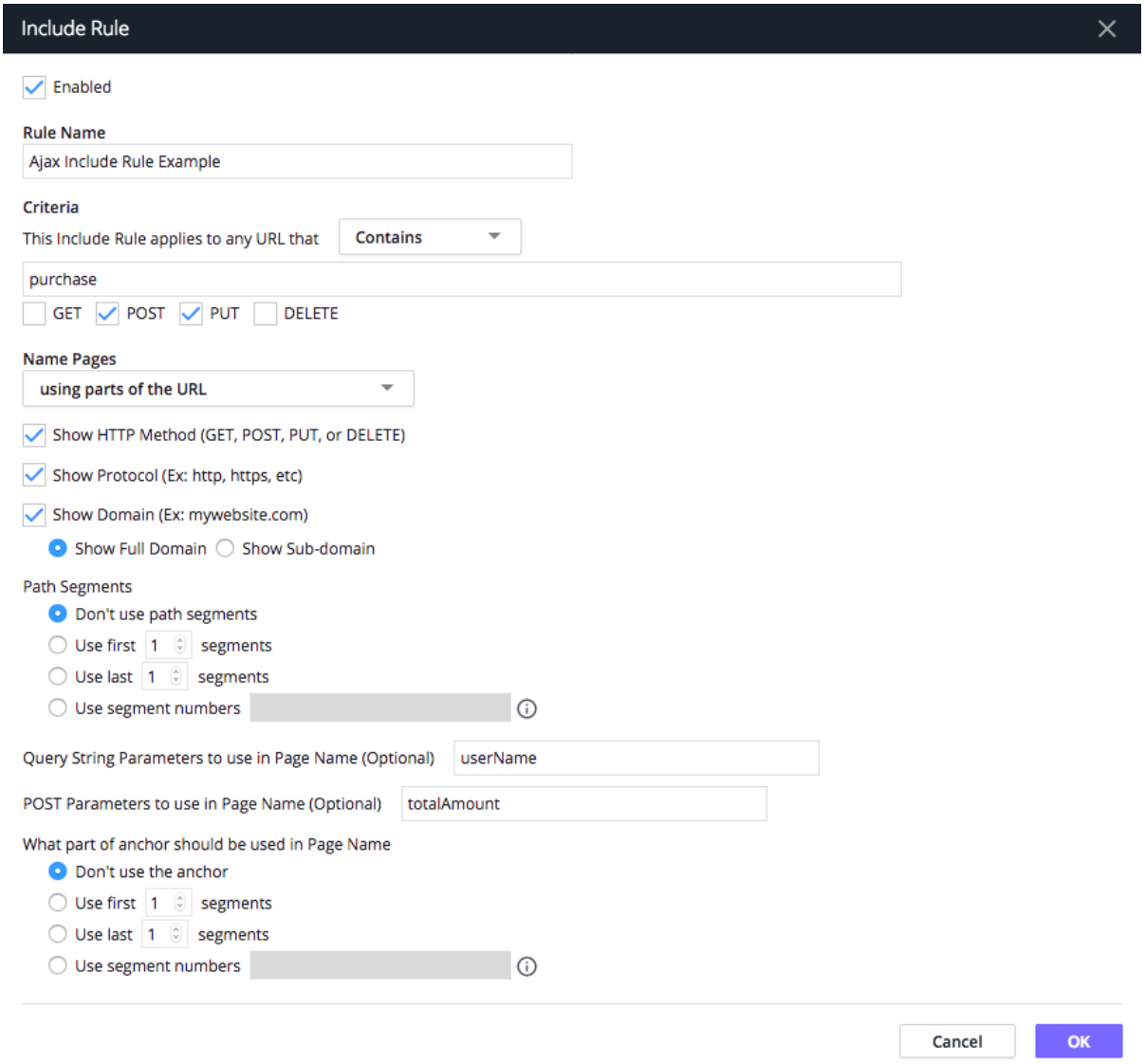
Access Naming Rules for Ajax Requests

1. Open the browser application in which you are interested.
2. On the left navigation bar, go to **Configuration** > **Instrumentation** > **AJAX** > **Monitor.**

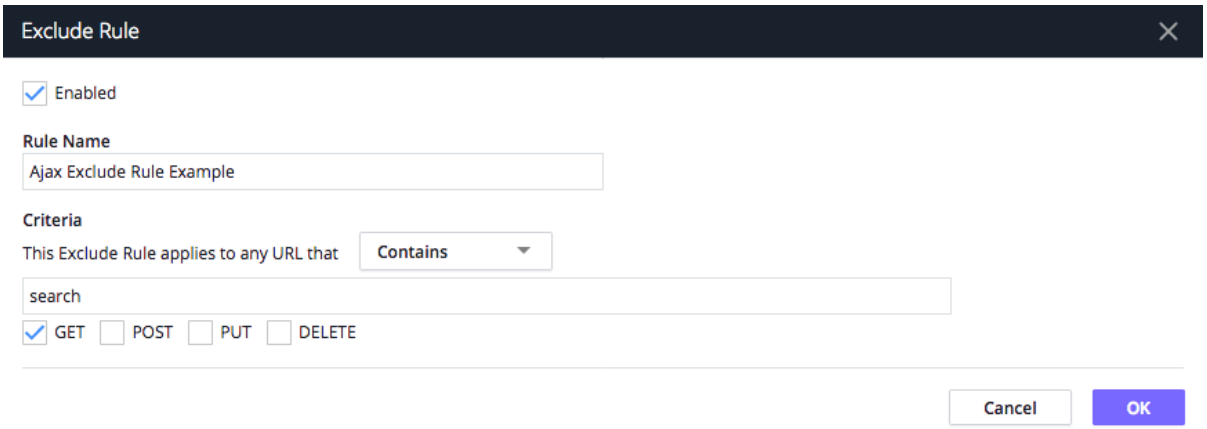
Naming Rules for Ajax Requests

The logic for naming Ajax requests is nearly identical to that for naming virtual pages in that they can use anchors (the part of the URL after the #) to distinguish among Ajax requests. In these cases, using the **What part of anchor should be used in page name** section allows you to specify which Ajax request is being accessed correctly. The difference in naming rules for Ajax requests is that you can also specify the HTTP method(s) to filter in the rule.

For example, the Ajax naming include rule below specifies the HTTP methods POST and PUT and is configured to show the HTTP method in the results displayed in the Controller UI:



You can also create exclude rules that specify which Ajax requests to filter based on the specified HTTP method(s) as shown here:


Exclude Rule
Enabled
Rule Name
Ajax Exclude Rule Example
Criteria
This Exclude Rule applies to any URL that
Contains
search
GET
POST
PUT
DELETE
Cancel
OK

Configure Which Ajax Requests Are Sent to the Events Service

Modern web pages often include a large number of Ajax requests. Not all of these Ajax requests may be equally important to monitor, but they all equally affect license entitlements and restrictions. Thus, you may want to configure which Ajax requests are sent to the Events Service to manage the impact on your overall Events Service usage.

To do this, you can create rules to specify the requests to be sent to the Events Service, either by excluding a request entirely, including a particular request or a sample of that request types by percentage or by just allowing the request to be sent. In general, the behavior follows this pattern:

- If no rules are specified, no data on Ajax requests is sent.
- If exclude rules are specified, and an Ajax request satisfies a rule, that data is *not* sent.
- If include rules are specified, any Ajax request that satisfies a rule is sent, based on sampling defined by the percentage indicated in the rule.
- If both include and exclude rules are specified, an Ajax request that satisfies an include rule but does *not* satisfy an exclude rule is sent.
- If you *only* configure exclude rules and the sample percentage, the non-excluded calls would be sampled at the specified rate.

See Configure Page Identification and Naming for more information about default naming rules and how rules are evaluated.

Access Ajax Requests Rules

1. Open the browser application in which you are interested.
2. On the left navigation bar, select **Configuration**.
3. Click **Instrumentation** >.
4. Select the **Ajax** > **Events Service** tab.

Configure Exclude Rules

1. Click **Add** to create a new rule. The **Exclude Rule** popup appears.
2. Specify a display name for the rule.
3. Check **Enabled** to enable the rule.
4. Enter a string and select one of the options in the dropdown to match the URLs that you want to exclude from the Ajax requests.
5. Click **OK** to save the rule.

Configure Include Rules

1. From the **Events Service Include Rules** section, click **Add** to create a new rule. The **Include Rule** popup appears.
2. Give your rule a display name.
3. Check **Enabled** to enable the rule.
4. For **Criteria**, enter a string and select one of the options in the dropdown to match URLs for including Ajax requests.
5. Configure how to name pages:
    - Select one of the options from the dropdown to determine the part of the URL.
    - Select the path segment.
    - Enter the query string to use in the page name.
    - Select the part of the anchor (or none) to use in the page name.
6. Click **Save**.

Configure Virtual Page Naming

Related pages:

- The Pages & Ajax Requests View
- Page, Ajax, and Iframe Dashboards
- Set Custom Page Names

Web applications built using Single Page Application (SPA) principles minimize network traffic by transferring computing work for creating what the user sees to the browser. The initial page request downloads necessary data for constructing the application, with some possible exceptions. For example, HTML partials and fragments may be fetched dynamically from the backend in response to user interaction.

The individual views that the user sees are known as virtual pages. Browser RUM supports virtual pages created using the AngularJS framework, sometimes known simply as Angular.

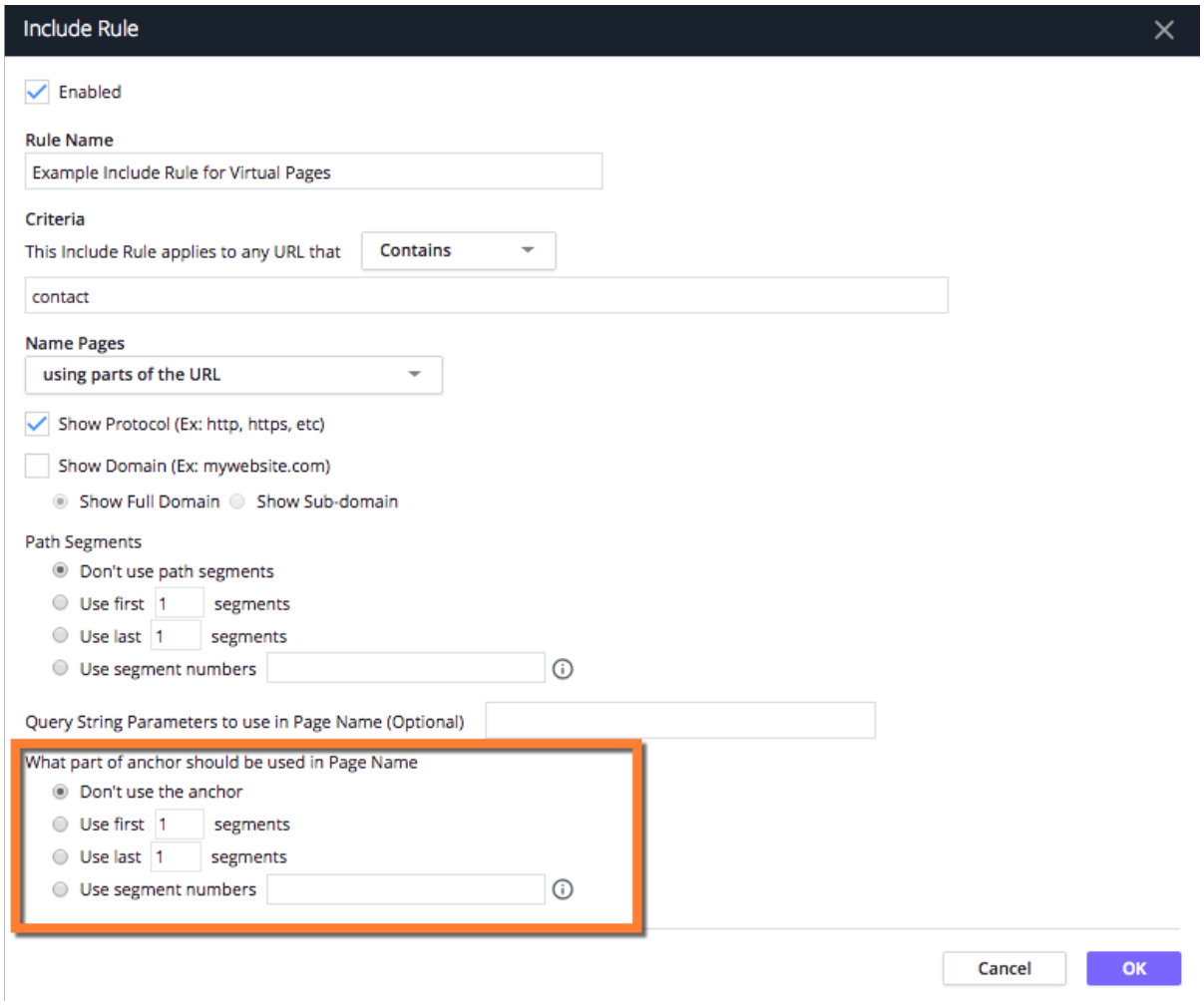
In AngularJS, a virtual page is an individual view, comprised of the rendered template of the current route in the context of the main layout file. You can configure how AngularJS virtual pages are referred to and sorted in UI lists and dashboards. For more information on using Browser RUM with virtual pages, see AngularJS Support.

Access Virtual Page Naming Rules

1. Open the browser application in which you are interested.
2. On the left navigation bar, click **Configuration**.
3. Click **Instrumentation >**.
4. Select the **Virtual Pages** tab.

Virtual Page Naming Rules

The logic for naming virtual pages is identical to that for naming regular pages, with one exception. Because AngularJS pages can use anchors (the part of the URL after the #) to distinguish among virtual pages, in these cases, using the **What part of anchor should be used in page name** section allows you to specify which virtual page is being accessed correctly.


Include Rule
Enabled
Rule Name
Example Include Rule for Virtual Pages
Criteria
This Include Rule applies to any URL that
Contains
contact
Name Pages
using parts of the URL
Show Protocol (Ex: http, https, etc)
Show Domain (Ex: mywebsite.com)
Show Full Domain
Show Sub-domain
Path Segments
Don't use path segments
Use first 1 segments
Use last 1 segments
Use segment numbers
Query String Parameters to use in Page Name (Optional)
What part of anchor should be used in Page Name
Don't use the anchor
Use first 1 segments
Use last 1 segments
Use segment numbers
Cancel
OK

Configure JavaScript and Ajax Error Detection

Related pages:

- Set Up and Access Browser RUM
- Browser Snapshots

You can enable and disable reporting of JavaScript and Ajax request errors. When enabled, the Browser Monitoring UI reports Ajax request errors in the following places:

- **Geo Dashboard**
- **User Stats** page and device dashboards
- Page list
- Browser snapshots

Also, you can configure which errors are included in the error count by specifying the errors to "ignore". You can specify errors to ignore by:

- The script and/or error message
- Page
- URL

"Ignored errors" are not actually ignored. They are still tracked, but the error count in the places where error totals are reported on the user interface is not incremented.

Access Error Detection Rules

1. From the browser application you are interested in, click **Configuration**.
2. Click **Instrumentation >**.
3. Select the Errors tab.

Enable and Disable Browser RUM Error Detection

In the Errors tab:

- Check/clear the **Enable JavaScript Error Capture** checkbox to enable/disable JavaScript error display.
- Check/clear the **Enable AJAX Request Error Capture** check box to enable/disable Ajax error display.

When both checkboxes are clear, no JavaScript or Ajax request errors are displayed.

Even if capture is enabled globally, you can configure certain errors to be ignored so that they are not counted in the error totals.

Configure Rules to Ignore Errors based on Script or Error Message

You can configure the agent to ignore specific JavaScript errors that are identified by:

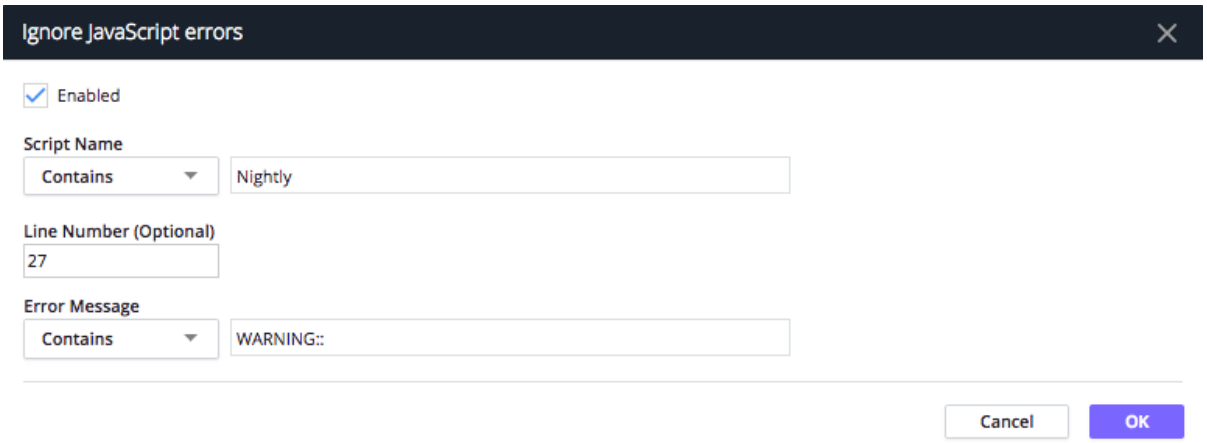
- a matching string pattern in the name of the script that generated the error
- the line number in the script
- a matching string pattern in the error message

You can specify one, two, or all three of these criteria. Configure more criteria to increase the granularity of which errors you ignore.

Add Rules to Ignore Errors

From the **Ignore JavaScript Error Rules** section, click **Add** to open the **Ignore JavaScript Errors** popup. From there, you can enable the rule, specify the script name, line number, and optionally, the error message.

For example, the following configuration in the **Ignore JavaScript errors** section where all three fields are specified means "Ignore all errors generated by line 27 of a script whose name starts with "Nightly" and whose error message contains the string "WARNING::".



If the line number were not specified (e.g., set to 0), the configuration would mean "Ignore all errors generated any line of a script whose name starts with "Nightly" and whose error message contains the string "WARNING::".

If neither the line number nor the error message field was specified, the configuration would mean "Ignore all errors generated by any line of a script whose name starts with "Nightly".

If the error message were the only field specified, the configuration would mean "Ignore all errors generated by any script when the error message contains the string "WARNING::".

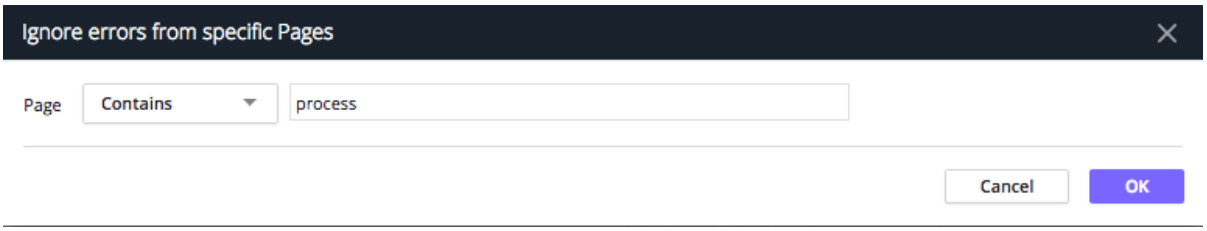
Modify Rules

To modify an existing ignore rule, you can either double-click the rule in the list or select the rule and then click **Edit** from the dropdown. To remove an ignore rule, select the rule in the list and then click **Delete** from the dropdown.

Configure Rules to Ignore Errors by Page

You can also ignore all errors generated by a specific page, iframe, or Ajax request.

From the **Ignore Errors from these Pages** section, click **Add** to open the **Ignore errors from specific Pages** popup, where you can create a rule for every page for which you want to ignore all errors.

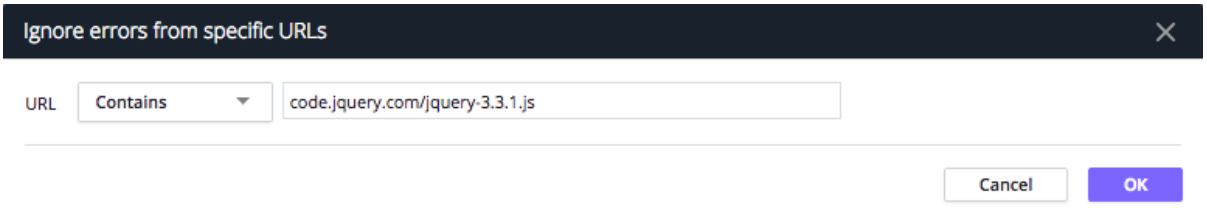


To remove an ignore rule, select it in the list and click **Delete** from the dropdown.

Configure Rules to Ignore Errors by URL

You can ignore all errors generated by a specific URL.

From the **Ignore Errors from these URLs** section, click **Add** to open the **Ignore errors from specific URLs** popup, where you can create a rule for every URL for which you want to ignore all errors.



To remove an ignore rule, select it in the list and click **Delete** from the dropdown.

Configure Browser RUM Performance Thresholds

You can configure the thresholds that define slow, very slow, and stalled end-user requests. These thresholds are used to trigger browser snapshots. See Transaction Thresholds for a detailed discussion about static, dynamic, percentage, and standard deviation thresholds.

You can define Browser Monitoring thresholds as one of the following:

- a multiple of the standard deviation. The default standard deviation threshold duration is two hours. For example: "Experience is slow if end-user response time is slower than 3 X the standard deviation."
- a static value; For example: "Experience is stalled if end-user response time is slower than 30000 ms."

The default thresholds are:

- Slow = 3 x standard deviation
- Very Slow = 4 x standard deviation
- Stalled = 45000 ms

Access Browser RUM Threshold Rules

1. Open the browser application in which you are interested.
2. On the left navigation bar, click **Configuration**.
3. Click **Instrumentation >**.
4. From the Settings tab, find the **Thresholds for Slow End User Experience** section.

Configure Browser RUM Threshold Rules

1. Select the relevant radio button to indicate whether the threshold is based on standard deviations or static values.
2. Type the values in the fields or select them using the scrollbars for one or more of the following:
    a. **Slow greater than**
    b. **Very slow greater than**
    c. **Stall greater than**
3. Click **Save**.

Configure Browser Snapshot Collection

Related pages:

- Browser Snapshots
- Set Up and Access Browser RUM

Every page view and AJAX request sends a beacon to the EUM server. Every minute, the EUM server evaluates the collection of beacon samples and sends what are called "snapshots" to the Controller.

With browser snapshots, you can enable or disable:

- Slow snapshot collection, that is snapshots of requests where the End User Response Time is higher than the configured threshold.
- Periodic snapshot collection.
- Error snapshots, that is snapshots of requests for which a JavaScript error is reported or an Ajax request receives an HTTP error response. An error response is any HTTP code equal to or greater than 400.

If all three kinds of browser snapshot types—periodic, error, and slow response time—are disabled, the agent does not collect any browser snapshots.

Access Snapshot Collection Rules

1. Open the browser application in which you are interested
2. On the left navigation bar, select **Configuration**.
3. Click **Instrumentation >**.
4. Select the Settings tab.
5. Find the **Event Policy Configuration** section.

Configure Snapshot Collection Rules

1. Check or clear the checkbox for one or more of the following:

- **Enable Slow Snapshot Collection** to enable/disable slow response time snapshot collection.
- **Enable Periodic Snapshot Collection** to enable/disable periodic snapshot collection.
- **Enable Error Snapshot Collection** to enable/disable error snapshot collection.

2. Click **Save**.

Configure Web Percentile Metrics

Related pages:

- Browser App Dashboard
- EUM Data

Parts of the Controller UI for Browser RUM rely on the processing done by the Events Service, including some of the widgets in the **Browser App Dashboard** and the **Analyze** page. For these metrics, you can choose to display either averages or percentiles.

A percentile is a measure that indicates a value below which a given percentage of values in a set falls: for example, the 99th percentile means that 99% of all values are below this level. Using percentiles can be a good way to reduce the impact of extreme outliers in performance metrics, which can be useful in the often noisy environments of end-user monitoring. The Metric Browser also displays the percentiles.

You can:

- Enable/disable percentile metric display
- Set up to four different percentile levels to be applied to metrics

Access Configure Percentile Metrics

1. Open the browser application in which you are interested.
2. On the left navigation bar, select **Configuration**.
3. Click **Instrumentation >**.
4. Select the **Settings** tab.
5. Find the **Configure Percentile Metrics** section.

Configure Percentile Metrics

1. Check the **Enable Percentile Metrics** checkbox.
2. Enter up to four percentile levels to collect in the **Percentiles to Collect** fields. Each value must be a whole number between 1 and 99.

Configure Session Monitoring Timeouts

You can control the timeframe for browser sessions by setting an inactivity limit. No activity beyond this limit is connected to previous activity as part of an individual session. See Session Persistence to learn more about how the duration of a session is determined.

Access Session Monitoring

1. Open the browser application in which you are interested.
2. On the left navigation bar, select **Configuration**.
3. Click **Instrumentation >**.
4. Select the Settings tab.
5. Find the **Configure Session Monitoring** section.

Set Inactivity Timeout

From **Configure Session Monitoring**, set the number of minutes for the **Session Inactivity Timeout** field.

Configure Request IP Address Storage - Browser

For security and privacy reasons, Browser RUM does *not* store source IP addresses associated with requests. To have Browser RUM store IP addresses associated with requests, follow the instructions below.

1. Open the browser application in which you are interested.
2. On the left navigation bar, select **Configuration**.
3. Click **Instrumentation >**.
4. Select the Settings tab.
5. At the bottom of the Settings tab, check the **Store IP Address** checkbox to enable IP address storage. (By default, the box is *not* checked.)
6. Click **Save**.

Monitor Single-Page Applications

Browser RUM's single-page application (SPA) monitoring enables you to:

- Monitor the performance/throughput of user experiences of SPAs.
- Troubleshoot and resolve SPA problems within the context of the page load.
- Assist with business decisions by analyzing SPA data through analytics.
- Enable developers to quickly ship better web applications to the marketplace.

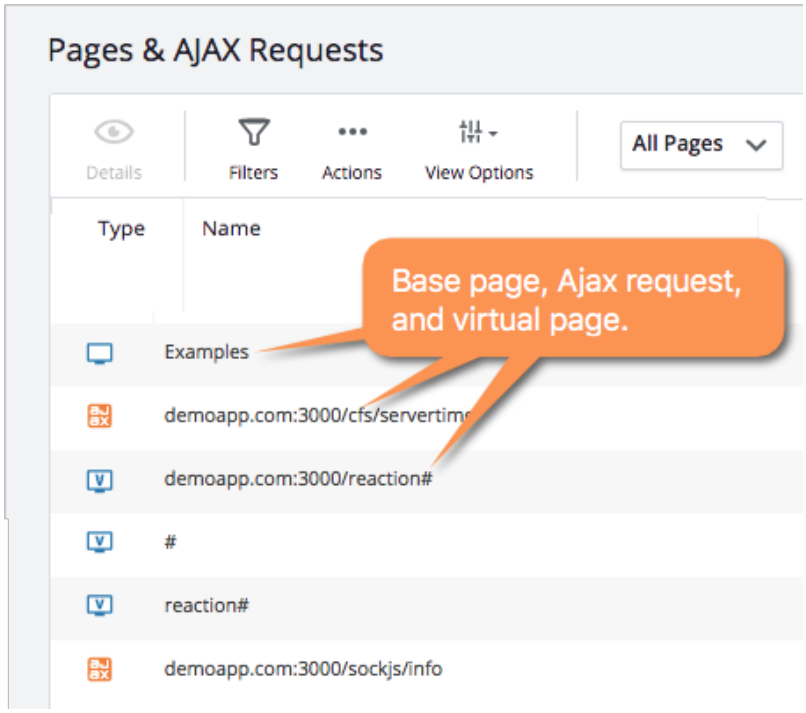
What Is SPA?

A single-page application (SPA) is a web technology and design paradigm that reduces browser-level page loads by using JavaScript to fetch resources and build pages. This creates a smoother, faster user experience more similar to a desktop or mobile application than a traditional web page. React and Angular are two popular JavaScript frameworks used to create SPAs.

Base Pages vs. Virtual Pages

When a user navigates to a SPA, the initial page download is considered the "base page." The base page includes the HTML skeleton, the core CSS, and the JavaScript framework for fetching and constructing new content. The ensuing pages are constructed from resources downloaded with the base page or fetched dynamically from the backend in response to user interaction. This new content (or views) constructed from different sources by the SPA are called "virtual pages."

For SPAs, you will have base pages, virtual pages, and Ajax requests. From **Pages & AJAX Requests**, you are able to view all three types and details for each as shown below.



SPA1 and SPA2 Monitoring

Until 4.4.3, Browser RUM only provided auto-instrumentation for AngularJS 1 SPA applications. For other frameworks, you had to manually report events with the JavaScript Agent API. This limited auto-instrumentation for SPA is known as *SPA1 monitoring*.

From JavaScript Agent >= 4.5.16, Browser RUM has been certified for Angular 1-9. This support for the auto-instrumentation for all SPA frameworks is known as *SPA2 monitoring*.

This table describes the support for SPA1 and SPA2 monitoring:

SPA Monitoring Version	Auto-Instrumentation	Metrics	JavaScript API

SPA1	AngularJS 1 only	• **Base / Virtual Pages:** Browser RUM page timing metrics • **Ajax Requests:** All available Ajax metrics	Yes
SPA2	All SPA frameworks. Auto-instrumentation is certified for React and Angular 1-9 (JavaScript Agent >=4.5.16.	• **Base pages:** Browser RUM page timing metrics, including End User Response Time (EURT), Visually Complete Time (VCT), and Page Complete Time (PCT) • **Virtual Pages:** End User Response Time (EURT), Visually Complete Time (VCT) • **Ajax Requests:** All available Ajax metrics	Yes

Which SPA Monitoring to Use?

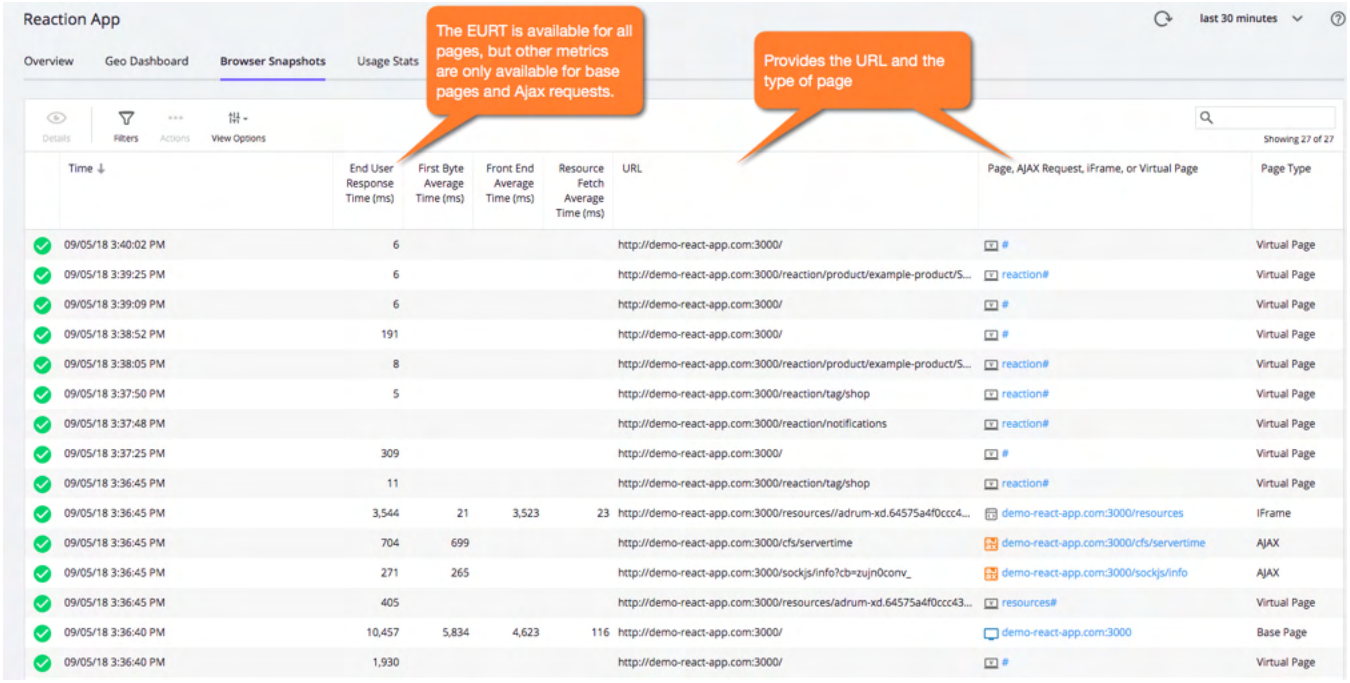
In general, you are highly recommended to use SPA2 monitoring. For most applications, the JavaScript Agent using SPA2 monitoring identifies the SPA transitions deterministically and captures the metrics and resources correctly without you having to write any special configuration or call methods from the JavaScript Agent API. If your SPA applications are built with AngularJS 1, you can simply add some configuration to enable auto-instrumentation.

View SPA2 Monitoring Data

Throughout the Controller, you can view SPA2 Metrics in various EUM dashboards in the Controller. You can also Configure the Primary Metric to choose a default displayed SPA2 metric for Browser RUM dashboards and widgets.

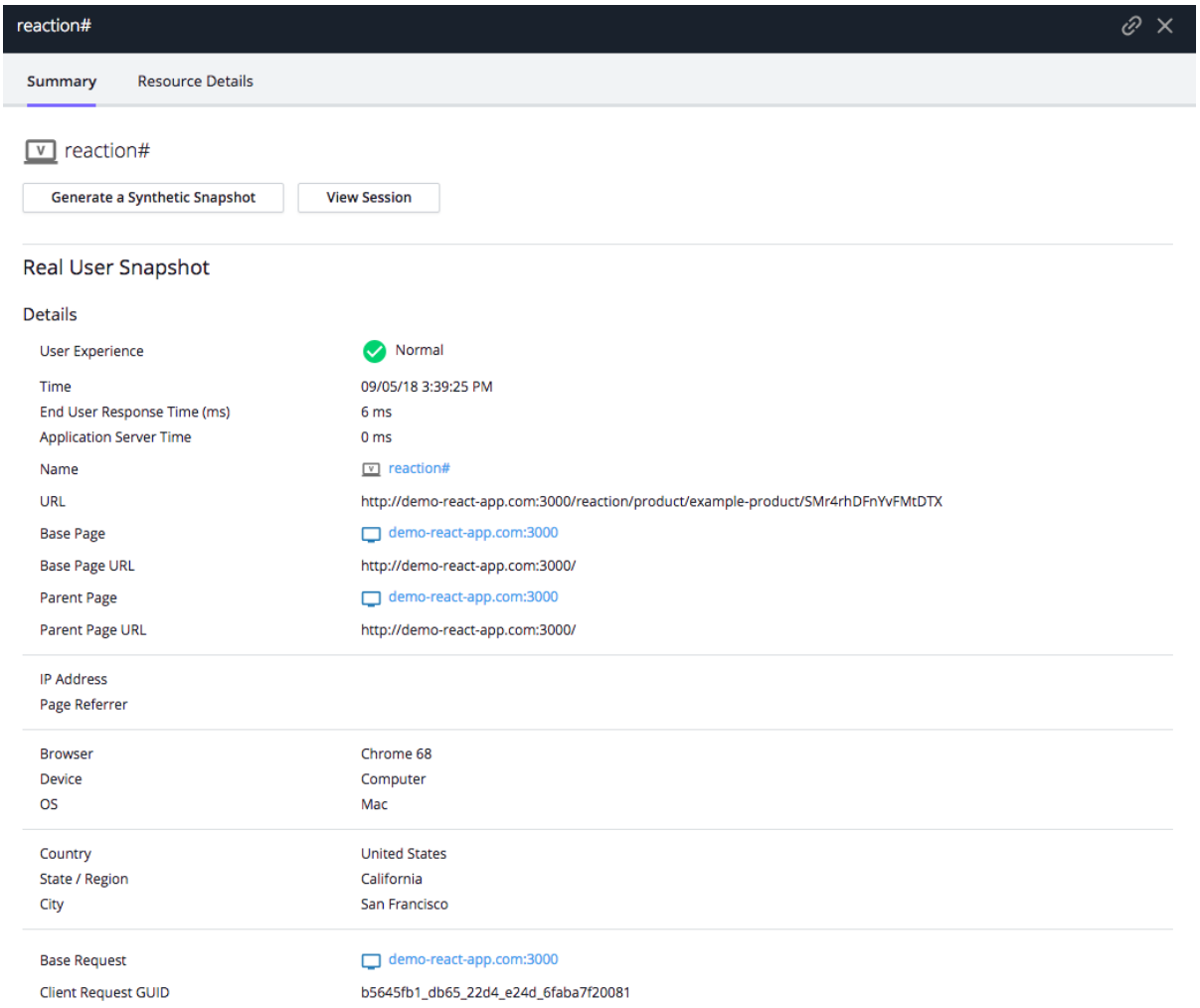
Browser Snapshots

The Browser Snapshots tab displays a list of different types of pages as shown below. You can view relevant metrics for all the pages and other metrics for base pages and Ajax requests.



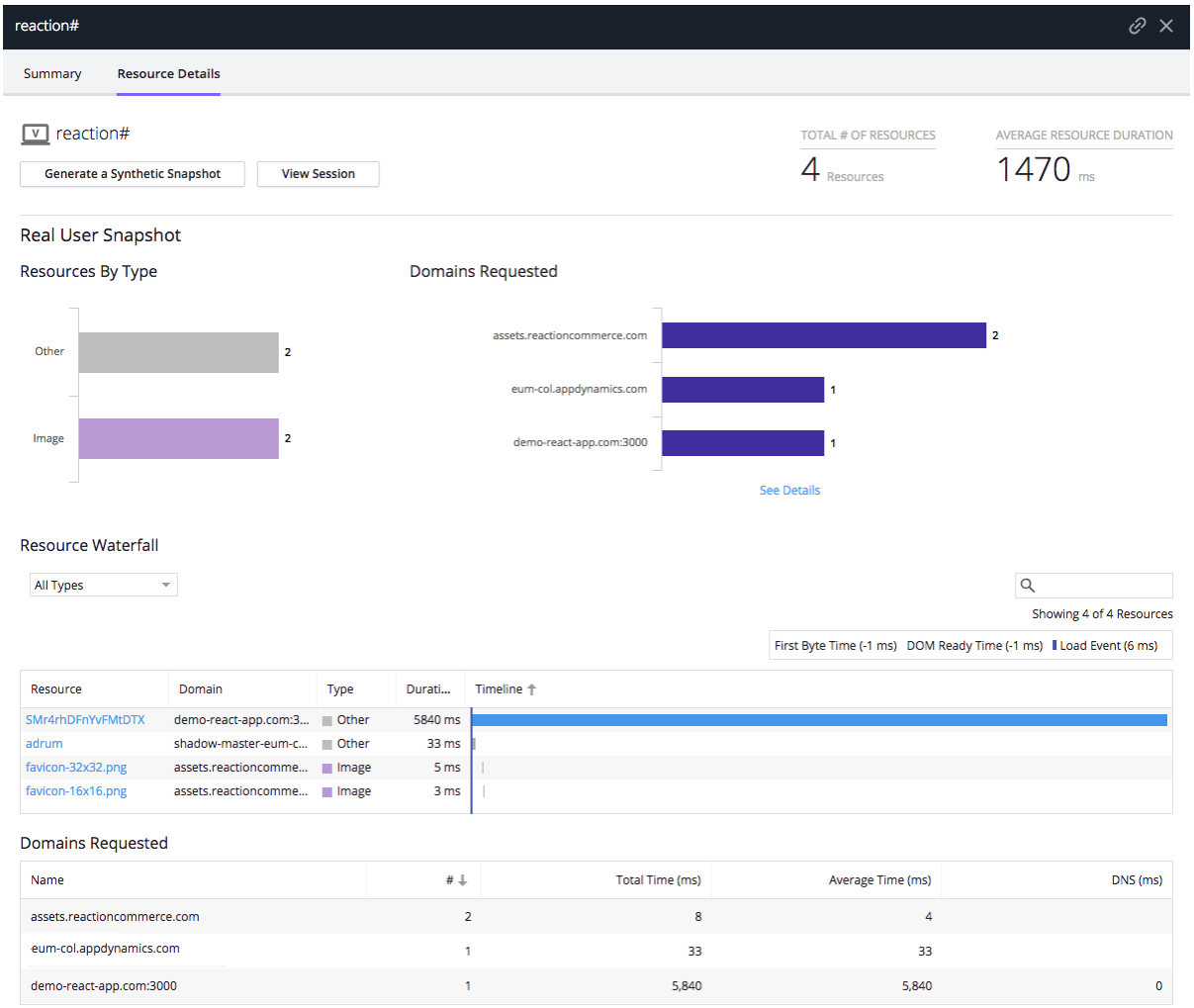
Snapshot Details: Summary

To access the **Snapshot Details** dialog shown below, you double-click one of the browser snapshots in the **Browser Snapshots** page. The Summary tab is the default tab for the dialog, and it gives some additional information such as the parent page URL.



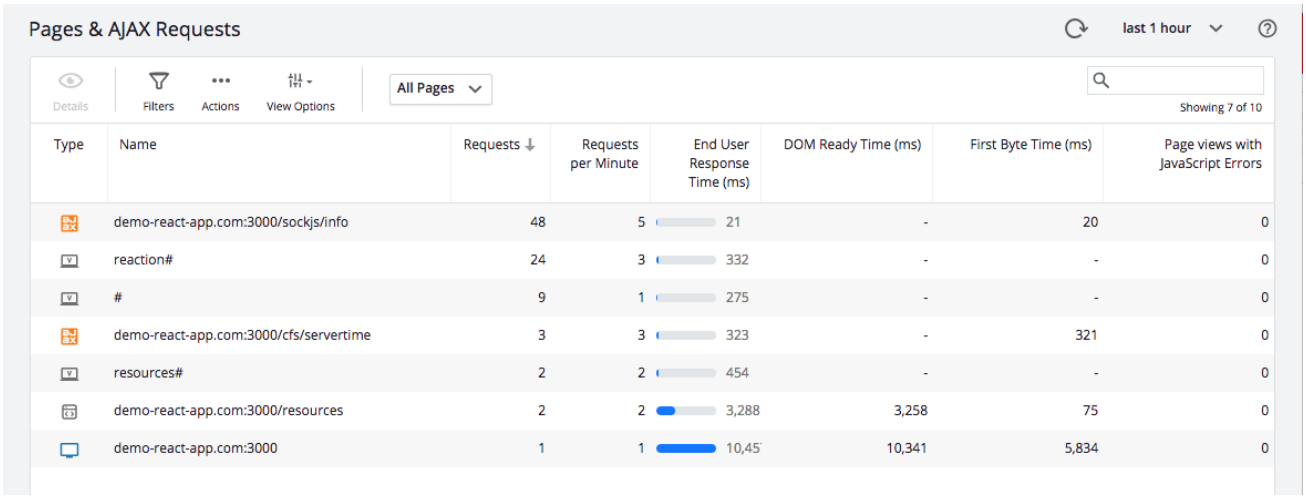
Snapshot Details: Resource Details

The Resource Details tab shown below provides details about resources such as the number and type of resources, the domains where the resources were requested, and resource load times.



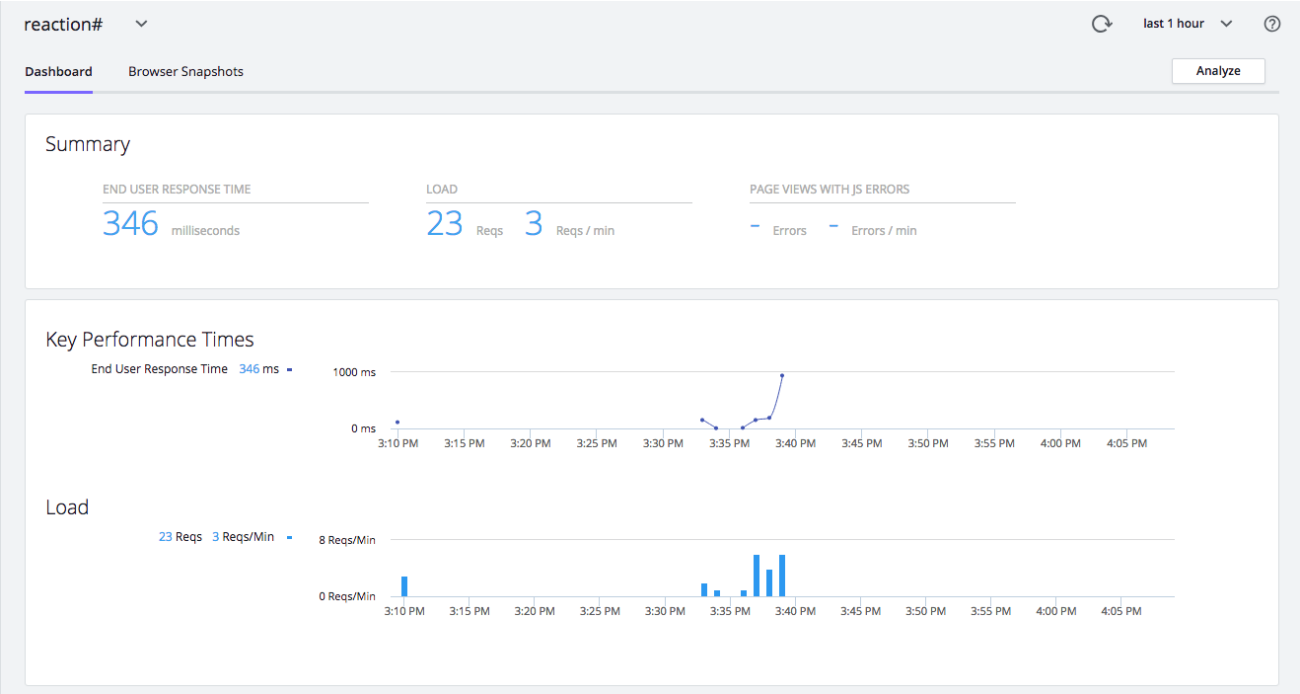
Pages & AJAX Requests

The **Pages & AJAX Requests** page as shown below enables you to view the number of requests for each type of page and the average metrics per page. You can also select which page types to view, such as virtual pages.



Page, Ajax, and Iframe Dashboards

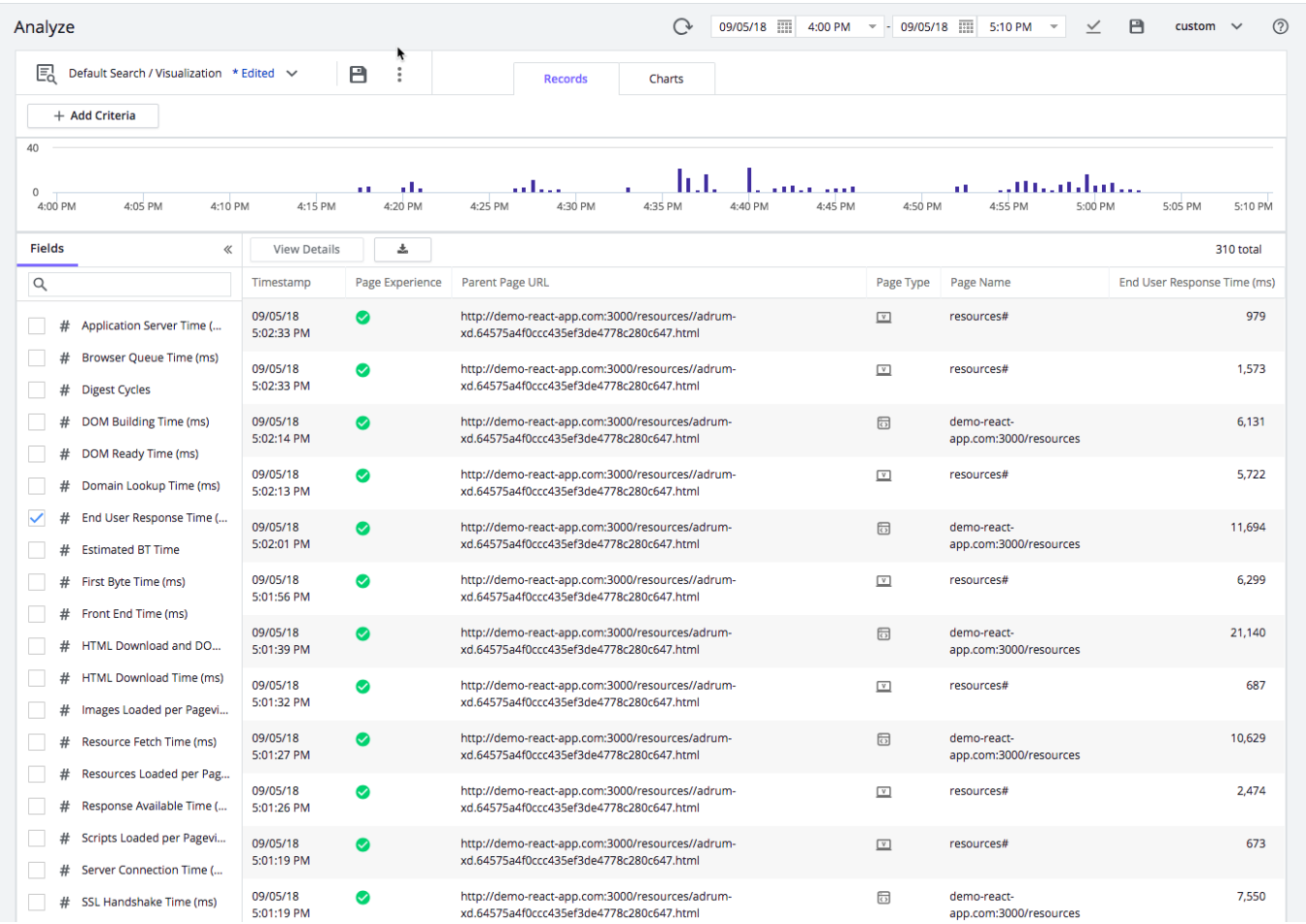
From the **Pages & AJAX Requests** page, you can double-click a page to view the Page and IFrame Dashboards. Dashboards for each page type will have a **Summary** section displaying the average metrics as well as a **Key Performance Times** section showing the requests and the metrics on a timeline**.**



The tabs below show the differences in the dashboard for each page type.

Browser Analyze

From **Browser Analyze**, you can filter by criteria, such as page types shown in the screenshot below, and configure the fields you want to view.



SPA1 Monitoring

We recommend you use SPA2 Monitoring for all SPA frameworks, especially Angular 1–5 and React.

This page describes how to use SPA1 monitoring for AngularJS 1 applications and manually report events for Ember.js applications. This page also provides information about the AngularJS 1 support for SPA1 monitoring.

SPA Requirements

- IE 10 and Edge is supported. IE >=10 and Edge *cannot* be in compatibility mode with <=IE 9 browsers.
- For non-AngularJS frameworks, the JavaScript Agent >=4.2 is required.

Manual Injection for Angular 1.x Applications

For non-Angular applications, `adrum.js` needs to be loaded before any other JavaScript on the page. When you are using AngularJS 1, however, you need to inject `adrum.js` after `angular.js`, but before `angular.js` is bootstrapped.

To learn which Angular versions have monitoring support, see Monitoring Support for AngularJS.

Manual Injection for Ember.js Applications

For Ember.js applications, you manually inject the JavaScript Agent much in the same way you would do for web pages loaded from an HTTP request. To monitor virtual pages, you manually start and end Virtual Page events based on events triggered when pages are dynamically created. You can also monitor virtual pages to report errors for pages.

The following sections will show you how to inject the JavaScript Agent and monitor virtual pages.

Manual Injection of the JavaScript Agent

The file `app/index.html` is the HTML skeleton for all dynamics pages in Ember.js. Thus, you can inject the JavaScript Agent in this file so it is included in every page.

Monitor Virtual Pages

When a user requests a new page, the route handler renders the associated template to form the new content of the virtual page. You can listen for events in the route handler indicating when the handler is started and completed, which in effect mark the lifetime of the virtual page. To monitor the virtual page, you start a Virtual Page Event when the `activate` event is triggered, close the Virtual Page when the `deactivate` event is triggered, and then report the completed virtual page.

The code snippet below listens to the `activate` and `deactivate` events and reports the created virtual page event.

```
import Ember from 'ember';
import config from '.././config/environment';

export default Ember.Route.extend({
  beforeEnterAbout: Ember.on('activate', function(){
    console.log('hello about');
    config.aboutVpView = new ADRUM.events.VPageView();
    config.aboutVpView.start();
  }),
  afterEnterRental: Ember.on('deactivate', function(){
    console.log('goodbye about');
    config.aboutVpView.end();
    ADRUM.report(config.aboutVpView);
  })
});
```

Use Virtual Pages to Capture Errors

You can also create actions in the routing handlers that can be monitored through virtual pages. For example, you might want to monitor Ajax calls. You can create an action that performs the Ajax call and then use a Virtual Page event to capture the results and errors as shown in the code snippet below.

```
 actions: {
    makeXhrCall() {
      config.xhrVpView = new ADRUM.events.VPageView();
      config.xhrVpView.start();
      var xmlHttp = new XMLHttpRequest();
      xmlHttp.onreadystatechange = function () {
        if (xmlHttp.readyState === 4) {
          console.log(xmlHttp.responseText);
          config.xhrVpView.end();
          ADRUM.report(config.xhrVpView);
        }
      };
      xmlHttp.open("GET", "http://localhost:3000", true);
      xmlHttp.send(null);
    }
}
```

Monitoring Support for AngularJS 1

The JavaScript Agent supports monitoring by default for AngularJS versions 1.x.

Routing Engines

AngularJS 1 applications that have multiple Views use a *route* to move from one virtual page to another. You can use Browser RUM to instrument any virtual page that uses either of two routing engines, `ngRoute` or `ui-router`.

Metrics

Because virtual pages are constructed in the browser, normal page view metrics must be adjusted. In essence, what a metric for AngularJS 1 must do is correlate the time between various routing events, using their timestamps. Metrics are calculated as follows:

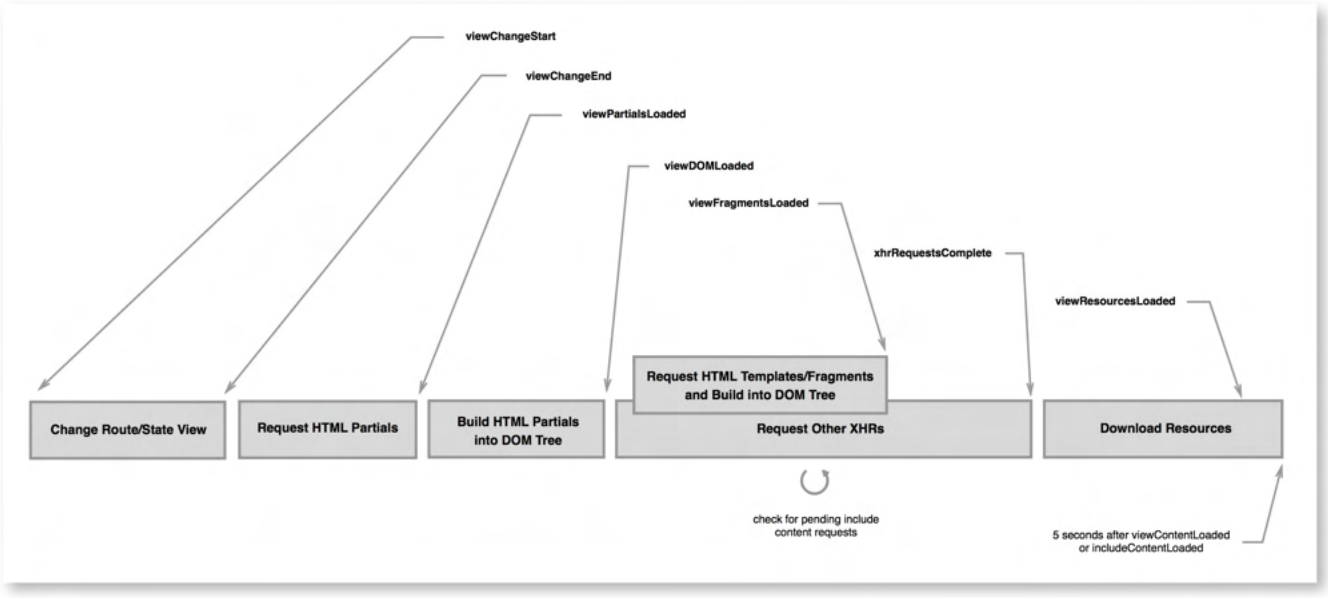
Full Metric Name	Short Metric Name	Calculation
End User Response Time (not used for waterfall UI)	PLT	`virtualPageStart` to `virtualPageEnd`
HTML Download Time	DDT	`viewChangeStart` to `viewChangeEnd`
HTML Download and DOM Building Time	DRT	`viewChangeStart` to `viewDOMLoaded`
DOM Building Time	DPT	`viewChangeEnd` to `viewDOMLoaded`
DOM Ready Time (used instead of PLT for waterfall UI)	DOM	`viewChangeStart` to `viewDOMLoaded`

Because the two routing engines function in slightly different ways, what the AppDynamics event consists of differs slightly, based on the engine.

AppDynamic Event Name	ngRoute Equivalent	ui-router Equivalent
`virtualPageStart`	`locationChangeStart`	`stateChangeStart`
`viewChangeStart`	`routeChangeStart`	`stateChangeStart`
`viewChangeEnd`	`routeChangeSuccess`	`stateChangeSuccess`
`viewDOMLoaded`	`viewContentLoaded`	`viewContentLoaded` (may happen multiple times - timestamp overwritten each time)
`viewFragmentsLoaded`	load time of the last HTML fragment	
`xhrRequestsCompleted`	response time of the last XHR requests	
`viewResourcesLoaded`	load time of the last resources	
`virtualPageEnd`	the latest one among `viewContentLoaded`, `xhrRequestsCompleted`, and `viewResourcesLoaded`	

Page Load Process Visualized

Visualized the page load process looks something like this:



Compare these to the standard page metrics, which are shown in Browser RUM Metrics.

Exclude Heartbeats or Background Requests from Timings

You may wish to exclude certain events from your virtual page timings. To do this, you can customize the JavaScript Agent when you inject it.

Add the following snippet *before* you add the JavaScript Agent file, `adrum.js`, to the page:

Option for excluding XHRs

```
<script type="text/javascript">
(function(config) {
   (function(spa) {
        (function(angular) {
            (function(vp) {
                vp.xhr = {
                    "exclude": {
                        "urls": {
                            pattern: '.*/dealActiveUsers'
                        }
                    }
                };
            })(angular.vp || (angular.vp = {}));
        })(spa.angular || (spa.angular = {}));
    })(config.spa || (config.spa = {}));
})(window["adrum-config"] || (window["adrum-config"] = {}));
</script>
```

Enable Resource Timing Collection for Virtual Pages

By default, Virtual Pages for AngularJS 1 do not include resource timing metrics. You need to set the Angular virtual page property `includeResTimingInEndUserResponseTiming` to `true`.

The JavaScript configuration below shows you how to enable resource timing collection for AngularJS 1 virtual pages. The configuration also sets limits for XHR calls per page, the buffer size for resource timings, and sets the flag for clearing resource timing metrics when a beacon is sent.

```
window['adrum-config'] = {
    "xhr": {
        "maxPerPageView": 10000
    },
    "resTiming": {
        bufSize: 300,
        "clearResTimingOnBeaconSend": true
    },
    "spa": {
        "angular": {
            "vp": {
                "metrics": {
                    "includeResTimingInEndUserResponseTiming": true
                }
            }
        }
    }
}
```

View Correlated Server Times

Since there isn't a regular HTML page timing to which correlated server timings can be linked, to view server times you must drill down from the Dashboard or Snapshot virtual page view to the component XHR requests. The server times can be seen there.

SPA2 Monitoring

For SPA2, you only need to set a flag to enable the JavaScript Agent to auto-instrument SPAs. When SPA2 auto-instrumentation is enabled, the JavaScript Agent:

- Detects virtual pages
- Measures the end-to-end time for virtual pages
- Correlates resources, JavaScript errors, and Ajax requests
- Names virtual pages

To enable SPA2 monitoring, see Configure SPA2 Monitoring.

How SPA2 Monitoring Works

The diagram below demonstrates how the JavaScript Agent defines the start and end points for the End User Response Time (EURT) metric and correlates with Ajax requests, resources, and JavaScript errors.

Step	Browser Activity	JavaScript Agent Actions
1	The user navigates to the base page of a SPA. The HTML skeleton, core CSS, and JavaScript are loaded into the browser.	The JavaScript Agent sends a beacon for the base page.
2	From the base page, the user clicks a button to view products. The URL changes as the virtual page is loaded through a combination of previously downloaded content and from new content fetched through Ajax requests.	The JavaScript marks the user action as the start time of the virtual page.
3	User actions and the browser activity cease for five seconds. The browser activity includes requesting resources, making Ajax calls, and so on.	The JavaScript Agent or your code using the JavaScript API marks the end time of the virtual page as the last time browser activity was seen after the URL changed. The JavaScript Agent then sends a beacon for the virtual page; however, if the virtual-page load stalls on one slow activity, the JavaScript Agent waits eight seconds before marking the end of the virtual page. Any activities such as Ajax requests, resource loads, and JavaScript errors that happened during the virtual page load will be correlated to that virtual page. There are some kinds of browser activities that aren't accessible to JavaScript. For example, the time that the browser takes render the DOM to the screen after the DOM has been updated. Therefore, if your virtual pages don't request any resources, you will likely see a very fast EURT (<10ms).

SPA2 Metrics

Browser RUM calculates metrics differently for base pages, Ajax requests, and virtual pages. The diagrams below demonstrate the differences between base page and virtual page metrics from the end-user perspective. See Browser RUM Metrics for metric definitions.

For base pages, Browser RUM calculates End User Response Time (EURT), Visually Complete Time (VCT), and Page Complete Time (PCT). For virtual pages, Browser RUM calculates EURT and VCT.

End User Response Time

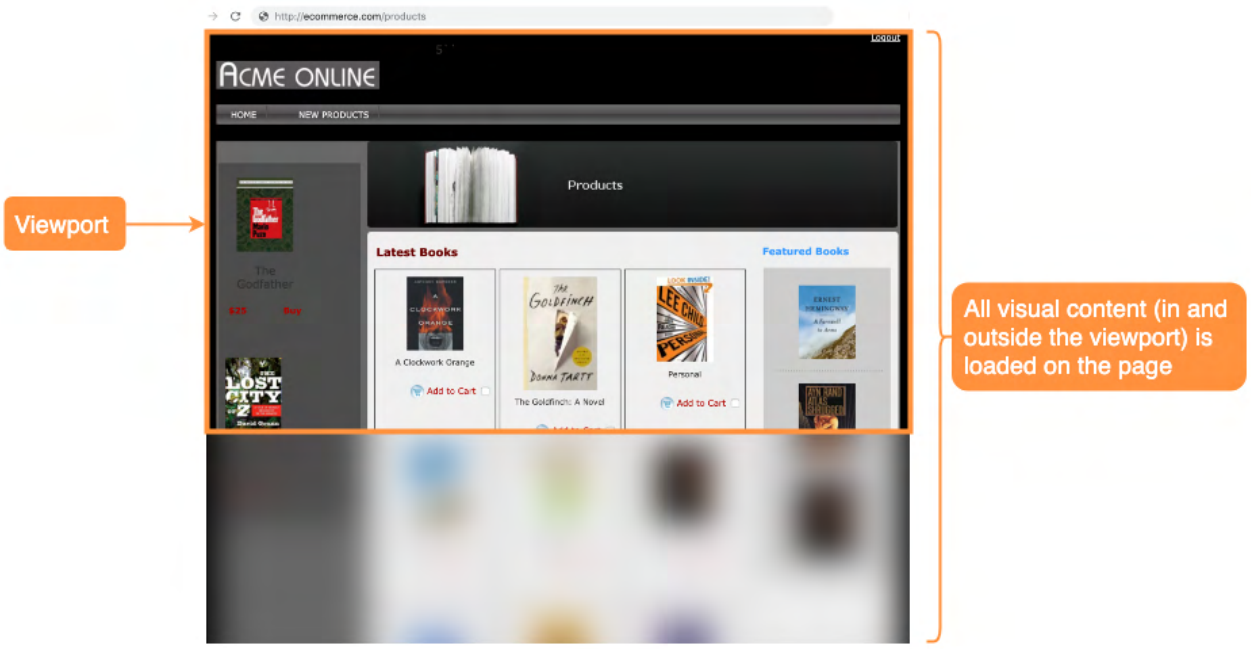
End User Response Time (EURT) calculates the total time for all content (visual and non-visual) to be loaded on a page.

Visually Complete Time

Visually Complete Time (VCT) calculates the point in time when the browser has finished loading all visual content in the viewport.

Page Complete Time

Page Complete Time (PCT) is a SPA2 metric for base pages only. PCT calculates the point in time when the browser has finished loading all visual content on the page, regardless of whether the content is in or outside the viewport.



Configure SPA2 Monitoring

SPA2 Requirements

- JavaScript Agent >= 4.4.3
- Controller / EUM Server >= 4.4.3 (for on-premises deployments)
- Enable SPA2 monitoring
- IE 10 and Edge are supported. IE >= 10 and Edge *cannot* be in compatibility mode with IE <= 9 browsers.
- For non-AngularJS frameworks, the JavaScript Agent >= 4.2 is required.

Angular 6-9 SPAs are supported by JavaScript Agent version >= 4.5.16. To manually report and name virtual pages with the JavaScript Agent API, you will need to use the JavaScript Agent >= 4.5. The JavaScript Agent >= 4.4.3 does not support the use of the JavaScript Agent API to manually report events.

Enable/Disable SPA2 Monitoring

In the JavaScript Agent configuration, set `spa2` to `true` and then inject the JavaScript Agent as shown below. You can use either manual or automatic inje ction. The default value for `spa2` is `false`, so to enable SPA2 monitoring, you must set `spa2` to `true`.

Make sure you set the configuration, including enabling SPA2 monitoring, *before* you load the JavaScript Agent.

```
<script charset='UTF-8'>
    window['adrum-start-time'] = new Date().getTime();
    (function(config){
        config.appKey = '<EUM_APP_KEY>';
        config.adrumExtUrlHttp = 'http://cdn.appdynamics.com';
        config.adrumExtUrlHttps = 'https://cdn.appdynamics.com';
        config.beaconUrlHttp = 'http://col.eum-appdynamics.com';
        config.beaconUrlHttps = 'https://col.eum-appdynamics.com';
        config.xd = {enable : false};
        config.spa = {
            "spa2": true
        };
    })(window['adrum-config'] || (window['adrum-config'] = {}));
</script>
<script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
```

Disable SPA2 Monitoring

To disable SPA2 auto-instrumentation, just set `spa2` to `false`. Although the default value for `spa2` is `false`, you are recommended to set the configuration to false and *not* just remove it.

Migrate from SPA1 to SPA2 Monitoring

To migration from SPA1 to SPA2 monitoring:

1. Read SPA1 and SPA2 Monitoring to confirm that your use case is suitable for SPA2.
2. Meet the SPA2 requirements.
3. Enable SPA2 monitoring.

Fetch API Support for SPA2 Monitoring

The JavaScript Agent monitors Fetch API calls by default for all SPAs except for Angular applications.

For Angular.js and Angular 2-9 applications, follow these steps to ensure Fetch API calls are monitored.

1. The JavaScript Agent is configured to monitor Fetch API calls by default. Make sure that your JavaScript Agent configuration *doesn't* have the following:

   ```
   config.fetch = false
   ```

2. Load the JavaScript Agent before Angular (recommended, but not required).

```
<script src="adrum.js"></script>
<script src="angular.js"></script>
```

3. Add the following configuration to ensure the JavaScript Agent monitors the Angular application correctly.

Troubleshoot SPA Monitoring

This page describes how to troubleshoot issues with monitoring SPAs.

Problems With Dynamically Loading the JavaScript Agent

Because Angular allows you to dynamically load modules with `AppdInitServlet`, you may be tempted to dynamically load `adrum.js`. The problem with dynamically loading `adrum.js`, however, is that this could cause a race condition between the time `adrum.js` is evaluated and the time `angular.js` is bootstrapped. This race condition can result in JavaScript Agent missing virtual pages or XHR events.

You can avoid this race condition by doing the following:

- Loading Angular first and then immediately loading the JavaScript Agent. See Load Angular and the JavaScript Agent.
- Bootstrap Angular after you have loaded the JavaScript Agent.

Load Angular and the JavaScript Agent

The key here is simply to make sure that you load `angular.js` first and then immediately load `adrum.js` as shown below.

```
<script src="angular.js"></script>
<script src="adrum.js"></script>
```

Bootstrap Angular

Manual (Recommended)

After you have loaded `adrum.js`, you can then bootstrap Angular through manual initialization. You should load both `angular.js` and `adrum.js` as soon as possible, so that the JavaScript Agent can get more accurate metrics about the page load and the HTML DOM Ready /Build time. The examples below follow the recommended procedure of first loading `angular.js`, then `adrum.js`, and then bootstrapping `angular.js`

Deferred (Optional)

If manual bootstrap is not possible, you can use deferred bootstrap. What this means is that you pause the bootstrap until `adrum.js` is successfully loaded, at which point, you resume the bootstrapping with `angular.resumeBootstrap()` as shown in the example below.

```
<!doctype html>
<html>
    <head>
        <script src="http://code.angularjs.org/snapshot/angular.js"></script>
        <script>
            // This defers the bootstrap.
            window.name = 'NG_DEFER_BOOTSTRAP!';
            angular.module('angularApp', [])
            .controller('TestController', ['$scope', function ($scope) {
                $scope.currentTime = new Date().toLocaleString();
            }]);
            angular.element(document).ready(function() {
                angular.bootstrap(document, ['angularApp']);
            });
        </script>
        // For simplicity, we're just showing the injection snippet using the AppDynamics CDN.
        <script charset='UTF-8'>
            window['adrum-start-time'] = new Date().getTime();
            (function(config){
                config.appKey = '<EUM_APP_KEY>';
                config.adrumExtUrlHttp = 'http://cdn.appdynamics.com';
                config.adrumExtUrlHttps = 'https://cdn.appdynamics.com';
                config.beaconUrlHttp = 'http://col.eum-appdynamics.com';
                config.beaconUrlHttps = 'https://col.eum-appdynamics.com';
                config.xd = {enable : false};
            })(window['adrum-config'] || (window['adrum-config'] = {}));
        </script>
        <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'><
/script>
  </head>
<body>
  <div ng-controller="TestController">
    It's {{currentTime}}!
  </div>
  <script>
    // Resume the bootstrapping.
    angular.resumeBootstrap()
  </script>
</body>
</html>
```

Manually Initializing the JavaScript Agent (Optional)

If `adrum.js` loads successfully, but does not initialize, you can manually initialize it by calling `ADRUM.ng.ngMonitor.init()` as shown below. This will force the JavaScript Agent to initialize even after the Angular bootstrap.

```
 <html>
     <head>
         <script src="http://code.angularjs.org/snapshot/angular.js"></script>
         <script>
             // This defers the bootstrap.
             window.name = 'NG_DEFER_BOOTSTRAP!';
             angular.module('angularApp', [])
             .controller('TestController', ['$scope', function ($scope) {
                 $scope.currentTime = new Date().toLocaleString();
             }]);
             angular.element(document).ready(function() {
                 angular.bootstrap(document, ['angularApp']);
             });
         </script>
<!doctype html>
<html>
    <head>
        <script src="http://code.angularjs.org/snapshot/angular.js"></script>
        <script>
            // This defers the bootstrap.
            window.name = 'NG_DEFER_BOOTSTRAP!';
            angular.module('angularApp', [])
            .controller('TestController', ['$scope', function ($scope) {
                $scope.currentTime = new Date().toLocaleString();
            }]);
            angular.element(document).ready(function() {
                angular.bootstrap(document, ['angularApp']);
            });
        </script>
        // For simplicity, we're just showing the injection snippet using the AppDynamics CDN.
        <script charset='UTF-8'>
            window['adrum-start-time'] = new Date().getTime();
            (function(config){
                config.appKey = '<EUM_APP_KEY>';
                config.adrumExtUrlHttp = 'http://cdn.appdynamics.com';
                config.adrumExtUrlHttps = 'https://cdn.appdynamics.com';
                config.beaconUrlHttp = 'http://col.eum-appdynamics.com';
                config.beaconUrlHttps = 'https://col.eum-appdynamics.com ';
                config.xd = {enable : false};
            })(window['adrum-config'] || (window['adrum-config'] = {}));
        </script>
        <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'><
/script>
    </head>
    <body>
        <div ng-controller="TestController">
        It's {{currentTime}}!
        </div>
        <script>
            // Resume the bootstrapping.
            angular.resumeBootstrap();
            ADRUM.ng.ngMonitor.init();
        </script>
    </body>
</html>
```

In some cases, you may be able to place `ADRUM.ng.ngMonitor.init()` before `angular.resumeBootstrap()`.

Enable the Content Security Policy

This page describes how to enable Content Security Police (CSP) so your application is compatible with Browser RUM.

Directives Required for CSP

To enable CSP for instrumented applications, you add the following required directives in the `Content-Security-Policy` header:

- `script-src`
- `connect-src`

In certain cases, you are also required to use the following directives:

- `child-src`
- `frame-ancestors`
- `img-src`

script-src

The `script-src` directive specifies the location of `adrum-ext.js`. By default, `adrum-ext.js` is loaded from our content delivery network (CDN) at `cdn.appdynamics.com`. The example below shows how you might use the `script-src` directive.

```
script-src cdn.appdynamics.com;
```

To measure first-byte time accurately, include the following line at the top of pages:

```
window["adrum-start-time"] = new Date().getTime();
```

For this line to be read, you also need to set the `script-src` directive to `'unsafe-inline'` as shown here:

```
script-src 'unsafe-inline';
```

connect-src

The `connect-src` directive specifies the location where beacons are sent. If you are using the SaaS-based EUM, you might use something like the following:

```
connect-src col.eum-appdynamics.com;
```

If you are using on-prem EUM, you would have `connect-src` point to your EUM Server.

child-src

For cross-domain sessions, we load `adrum-xd.html` into an iframe. By default, this is loaded from our CDN, so you need to have `child-src` specify a CDN as shown below.

```
child-src cdn.appdynamics.com;
```

frame-ancestors

If `adrum-xd.html` is hosted locally, you would use the `frame-ancestors` directives in the following way:

```
frame-ancestors /path/to/adrum-xd.html;
```

img-src

In older browsers, we send our beacons as image beacons. Although older browsers don't support CSP, you can configure the JavaScript Agent to always send image beacons. You do this using `img-src` directive to specify the beacon location as shown in the example below.

```
img-src col.eum-appdynamics.com;
```

Example Content-Security-Policy Header

The following `Content-Security-Policy` header loads the `adrum` files from our CDN and then sends beacons to our SaaS-based EUM.

```
Content-Security-Policy: connect-src 'self' col.eum-appdynamics.com; script-src 'unsafe-inline' cdn.appdynamics.
com; img-src cdn.appdynamics.com; child-src cdn.appdynamics.com
```

Host a Geo Server

Related pages:

- EUM Server Requirements
- EUM Server Deployment
- Configure the EUM Server
- Browser RUM Countries and Regions by Geo Dashboard

AppDynamics hosts a Geo Server that resolves the user's geographic location based on the request's reported IP address. You may prefer to host your own Geo Server if:

- You have intranet applications where the public IP address does not provide meaningful location information but the user's private IP does.
- You have a hybrid application where some users access the application from a private location and some access it from a public one. If a user doesn't come from a specific private IP range mapped by the custom geo server, the system can be set to default to the public Geo Server.

The custom Geo Server supports an HTTP header, `AD-X-Forwarded-For`. You can use this to declare an IP address specifically for the purpose of geo resolution. As of version 4.2, the custom Geo Server also supports modifying the JavaScript Agent itself to specify a particular IP or location. Custom Geo Server also supports the HTTP header `X-Real-IP`.

See Install and Host a Custom Geo Server for Browser RUM for more information on setting up your own Geo Server and private IP mapping file.

The AppDynamics Geo Server requires JDK >= 7.

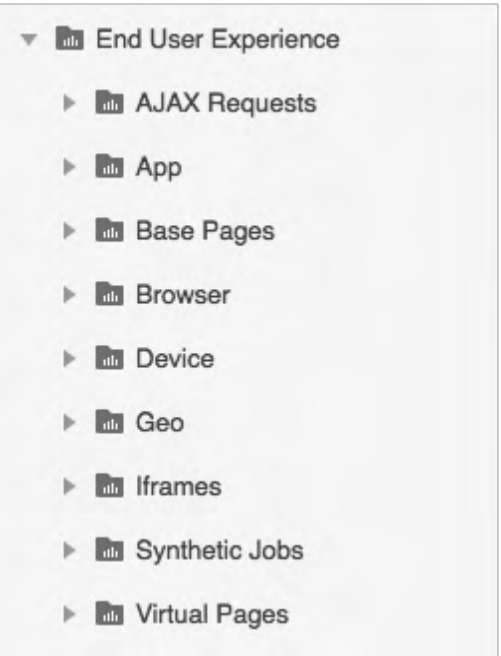
Browser RUM Metrics

Related pages:

- AppDynamics APIs
- The Pages & Ajax Requests View
- WC3 Navigation Timing API

Key Browser RUM metrics display on the **Geo**, **Pages & AJAX Requests**, and **Usage Stats** dashboards. They are also visible on the **All Pages** and **Top Pages** lists and in the Metric Browser.

In addition, the **Metric Browser** enables you to view these metrics in the context of Ajax requests, iframes, applications, base pages, browsers, devices, and geographic locations.



You can build custom health rules based on Browser RUM Page, Ajax, and iframe metrics in the health rule builder. Use these rules to automatically monitor key metrics in your installation. See Health Rules.

Browser RUM Timing Metrics Overview

The following provides an overview of the first Browser RUM page and iframe timing metrics. For key Ajax metrics, see Ajax Metrics Availability. Metrics in blue are available only from NavTime browsers. You may see a value of `unknown` for some metrics taken using older browsers.

When users begin to load the first instrumented page, the JavaScript Agent starts a timer. The timer or the NavTime `responseStart` starts the first-page timing.

> Because a NavTime browser always sets `navigationStart`, you can retrieve this information even on the first page a user loads from your site.

NavTiming-capable browsers also provide highly granular information on connection details.

Browser RUM Metrics Defined

Timing metrics are the average times, in milliseconds, over the time range selected in the Controller UI or REST API call. The three-letter abbreviation is the short name as it is recorded in the web beacon.

Name (Short Name)	NavTiming Capable Browsers[1]	Browsers without NavTiming Support	Available For . . .	Definition

Ajax Callback Execution Time (DPT)	`responseEnd` to `domContentLoadedEventStart` (context of the Ajax object)	`responseEnd` to `domContentLoadedEventStart` (context of the Ajax object)	Ajax Requests (XHR, Fetch API)	Time for the browser to process the Ajax response. This typically includes the time to apply the response data to the DOM.
Ajax Response Download Time (DDT)	`responseStart` to `responseEnd` (context of the Ajax object)	`responseStart` to `responseEnd` (context of the Ajax object)	Ajax Requests (XHR, Fetch API)	Time for the browser to download the complete Ajax response.
Ajax Request Errors per Minute (Short name based on ARE entries)			• Ajax Requests (XHR, Fetch API) • App • Browser • Device • Geo	Total number of Ajax requests that generate an error per minute.
Ajax Requests per Minute (Short name based on PLC entries)			• App • Browser • Device • Geo	Total number of Ajax requests per minute.
Application Server Calls per Minute			• Ajax Requests (XHR, Fetch API) • App • Base Pages • Browser • Device • Geo • iframes	Number of requests that actually hit the application server, rather than a cache.
Application Server Time (also called Server Time in the UI)			• Ajax Requests (XHR, Fetch API) • App • Base Pages • Browser • Device • Geo • iframe	Processing time for requests on the application server.
Browser Queue Time			• Ajax (XHR, Fetch API) • App • Base Pages • Browser • Device • Geo • iframes	
DOM Building Time (DPT)	`responseEnd` to `domContentLoadedEventStart`	N/A	• App • Base Pages • Browser • Device • Geo • iframes	Time for the browser to build the Document Object Model (DOM) and make it available for JavaScript to apply rendering logic.

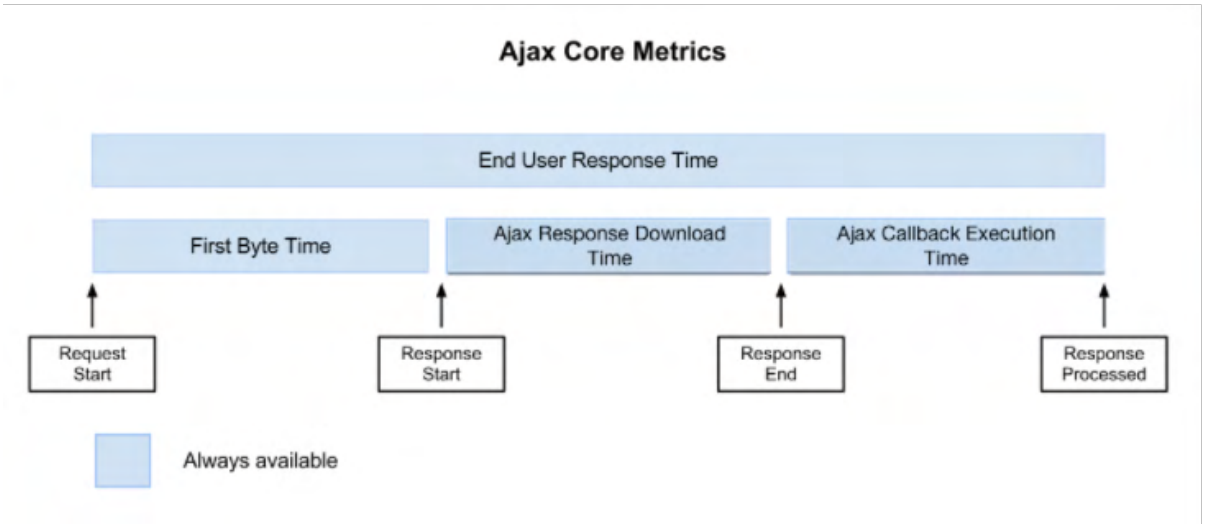
DOM Ready Time (DOM)[2]	`navigationStart` to `domContentLoadedEventStart`	Time between the writing of the `starttime` cookie on the previous page and an internal handler's `onready` event, similar to `jquery.onready()`.	• App • Base Pages • Geo	Interval between the time that a user initiates a request and the time that the `DOMContentLoaded` event (or the internal handler's `onready` event) occurs.
Domain Lookup Time (DNS)	`domainLookupStart` to `domainLookupEnd`	N/A	• App • Base Pages • Browser • Device • Geo • iframes	Time to complete the domain lookup portion of the server connection time.
End User Response Time (PLT)[2]	`navigationStart` to `loadEventEnd`	Time between the writing of the `starttime` cookie on the previous page and the `onload` event.	• Ajax Requests (XHR, Fetch API) • App • Base Pages • Browser • Device • Geo • iframes	Average interval between the time that a user initiates a request and the completion of the page load of the response in the user's browser. In the context of an Ajax request, it ends when the response has been completely processed. For information about End User Response Time for SPA2 pages, see SPA2 Metrics.
First Byte Time (FBT)[2]	`navigationStart` to `responseStart`	Time between the writing of the `starttime` cookie on the previous page and when the page's JavaScript Agent begins executing.	• Ajax Requests (XHR, Fetch API) • App • Base Pages • Browser • Device • Geo • iframes	Interval between the time that a user initiates a request and the time that the browser receives the first response byte. In the context of an Ajax request, First Byte Time is the interval between the Ajax request dispatch and the time that the browser receives the first response byte.
Front End Time (FET)	`responseStart` to `loadEventEnd`	Time between when the page's JavaScript Agent begins executing and the `onload` event.	• App • Base Pages • Browser • Device • Geo • iframes	Interval between the arrival of the first byte of text response and the completion of the response page rendering by the browser. Includes HTML Download, DOM Building Time, and Resource Fetch Time.
HTML Download and DOM Building Time (DRT)	`responseStart` to `domContentLoadedEventStart`	Time between when the page's JavaScript Agent begins executing and an internal handler's `onready` event, similar to `jquery.onready()`.	• App • Base Pages • Browser • Device • Geo • iframes	Time to make the complete HTML document (DOM) available for JavaScript to apply rendering logic. Includes the HTML Download and the DOM Building Time.
HTML Download Time (DDT)	`responseStart` to `responseEnd`	N/A	• App • Base Pages • Browser • Device • Geo • iframe	Time for the browser to download the complete HTML document content.

Iframe Requests per Minute (Short name based on PLC entries)			• App • Browser • Device • Geo	Total number of iframe requests per minute.
Images Loaded per Pageview			• Ajax (XHR, Fetch API) • App • Base Pages • Browser • Device • Geo • iframes	Total number of images in a `pageview`. Calculated after page has completely loaded.
Page Complete Time (PCT)	N/A	N/A	• App • Base Pages • Browser • Device • Geo	Total time for the browser to render all visual page elements (both in and out of the viewport). Available for SPA base pages only. For information about Page Complete Time for SPA2 pages, see SPA2 Metrics.
Page Requests per Minute (Short name based on PLC entries)			• App • Browser • Device • Geo	Total number of Page requests per minute. This is the metric displayed across most of the UI.
Page views with JavaScript Errors per minute (Short name based on EPM entries)			• App • Base Pages • Browser • Device • Geo • iframes	Total number of page views that contain JavaScript errors per minute.
Resource Fetch Time	`domContentLoadedEventStart` to `loadEventEnd`	Time between an internal handler's `onready` event, similar to `jquery.onready()`, and `onload`.	• App • Base Pages • Browser • Device • Geo • iframes	Time for the browser to complete the download of remaining resources, including images, and finish rendering the page.
Resources Loaded per Pageview			• Ajax (XHR, Fetch API) • App • Base Pages • Browser • Device • Geo • iframes	Total number of resources in a `pageview`. Calculated after the page has completely loaded.
Requests per Minute (Short name based on PLC entries)			• Ajax Requests (XHR, Fetch API) • App • Base Pages • Browser • Device • Geo • iframes	Total number of requests (Page + Ajax + iframe) per minute.

Scripts Loaded per Pageview			• Ajax (XHR, Fetch API) • App • Base Pages • Browser • Device • Geo • iframes	Total number of scripts in a `pageview`. Calculated after the page has completely loaded.
Server Connection Time (SCT)	`navigationStart` to `requestStart`	N/A	• App • Base Pages • Browser • Device • Geo • iframes	Interval between the time that a user initiates a request and the start of fetching the response document from the server or application task. Includes the time spent on redirects, domain lookups, TCP connects and SSL handshakes.
SSL Handshake Time (SSL)[3]	`secureConnectionStart` to `connectEnd`	N/A	• App • Base Pages • Browser • Device • Geo • iframes	Time taken to complete the SSL handshake.
TCP Connect Time (TCP)	`connectStart` to `connectEnd`	N/A	• App • Base Pages • Browser • Device • Geo • iframes	Time to complete the TCP connect portion of the server connection time, the equivalent of one network roundtrip of latency.
Total Resource Redirect Time			• Ajax (XHR, Fetch API) • App • Base Pages • Browser • Device • Geo • iframes	Time for all redirects associated with fetching resources. Calculated after the page has completely loaded.
Visually Complete Time (VCT)	N/A	N/A	• App • Base Pages • Browser • Device • Geo • iframes	Total time for all visual elements within the first screen (above-the-fold content) to load in an end user's browser. Calculated using the last visual change to the page in a browser window. For information about Visually Complete Time for SPA2 pages, see SPA2 Metrics.

Ajax Metrics Availability

Because Ajax requests occur in the context of the larger page, these four core metrics are always available, regardless of the browser type.



For more information on which browsers support the Navigation Timing API, see the Can I Use website.

Resource Timing Metrics

Resource timing metrics are the median times in milliseconds. The resource timing metrics shown in the Resource Performance Dashboard are raw data. For a visual representation of the resource timing, see the Resource Timing Overview diagram.

Metric Name	How It's Calculated	Description
Browser Wait	`startTime` or `redirectEnd` to `fetchStart`	Time from the redirection or request start until the browser begins to fetch the resource.
Redirect Time	`redirectStart` to `redirectEnd`	Time for all redirects associated with fetching resources.
DNS Wait Time	`fetchStart` to `domainLookupStart`	Time from when the browser starts to fetch the resource until the browser starts the domain name lookup for the resource.
DNS Time	`domainLookupStart` to `domainLookupEnd`	Time to complete the domain lookup portion of the server connection time.
TCP Wait Time	`domainLookupEnd` to `connectStart`	Time for the browser to find the domain name for the resource before a connection is made.
Connection Time	`connectStart` to `connectEnd`	Time to complete the establish the TCP connection for a resource: the equivalent of one network roundtrip of latency.
SSL Time	`secureConnectionStart` to `connectEnd` (if `secureConnectionStart` exists)	Time to establish a secure connection to the server.
Request Wait Time	connectEnd to requestStart	Time for the request is made for a resource from the server, cache, or local resource before the connection is closed.
Request Time	`requestStart` to `responseStart`	Time for the browser to complete the request for a resource from the server, cache, or local resource.
Response Time	responseStart to responseEnd	Time for the browser to download the complete HTML document content. In the context of an Ajax request, the time for the browser to download the complete Ajax response.
First Byte Time	`startTime` to `responseStart`	Time from when the request is made until the first byte of the resource is received by the browser.
Resource Load Time	`startTime` to `responseEnd`	Time for the browser to start a request and receive a response for a resource.
Browser Queue Time	Browser Wait + DNS Wait Time + Request Wait Time	Total wait time of the resource.

TCP Time	`secureConnectionStart` to `connectStart`	Time spent to establish a secure connection to the server (same as SSL Time).

Browser RUM Supported Environments

This page describes the supported environments and versions for Browser RUM.

Browser RUM Support

Browser Compatibility

- IE 10, 11, Edge
- Chrome >= 1.0.154.60, including Mobile
- Headless Chrome >= 59
- Firefox, including Mobile
- Safari >= 4 , including Mobile
- Opera >= 12

> Browsers are rapidly evolving, and not all versions have been individually tested with Browser RUM. You can view the browser versions *likely* to support the Resource Timing API.

Browser Requirements for Sessions

To use Browser RUM sessions, your browser is required to support for the following:

- cross-origin resource sharing (CORS) for beacons
- local storage for multiple-page sessions (single-page / multiple virtual page sessions don't need local storage)

> Browser RUM sessions do not support beacons implemented with GIFs.

JavaScript Agent Version Requirement for Sessions

You are required to use the JavaScript Agent >= 4.2. Older versions of the JavaScript Agent do not support sessions.

Injection Types for Browser RUM in Java Environments

Listed below are the injection types with their supported frameworks/technologies:

- **Automatic Injection**: JavaServer Pages (JSP) compiled using the Jasper compiler. Jasper is the default JSP compiler in Tomcat, Glassfish, and J Boss.
- **Assisted Injection:** All Servlet frameworks.
- **Manual Injection**: All technologies that generate HTML pages.

See Inject the JavaScript Agent for details.

Injection Types for Browser RUM in .NET Environments

AppDynamics certifies Browser RUM instrumentation for the following .NET environments:

- .NET Framework
- .NET Core

Browser RUM Instrumentation for .NET Framework

Supported Runtime Environments for Browser RUM .NET Framework include:
Microsoft IIS versions 6.0, 7.0, 7.5, 8.0, 8.5

- All listed frameworks support manual injection of the JavaScript Agent for Browser RUM.
- The Script Injection column lists additionally supported script injection strategies. See Inject the JavaScript Agent for details.

Web Application/AJAX Frameworks	Versions	Additional Supported Script Injection Methods
ASP.NET Web Forms (.aspx)	3, 4	Automatic, Using Attribute Injection
ASP.NET MVC Web Forms (.aspx)	3 - 5	Automatic, Using Attribute Injection
ASP.NET MVC Razor	3 - 5	Automatic, Using Attribute Injection
Microsoft SharePoint	2007, 2010	Automatic

AppDynamics does not support Browser RUM instrumentation of legacy ASP (`.asp`) pages.

Browser RUM Instrumentation for .NET Core

AppDynamics currently supports .NET Agent for Windows >= 21.1, and .NET Agent for Linux >= 21.5.

Web Application/AJAX Frameworks	Versions	Supported Script Injection Methods
ASP.NET Core MVC	3.1, 5	Automatic
ASP.NET Core Razor Pages	3.1, 5	Automatic

Browser RUM Countries and Regions by Geo Dashboard

The countries and regions in the sections below can be displayed in the **Geo Dashboard**. Each of these countries and regions also displays their own aggregate Browser RUM data. Data can be collected from areas not in the following list, but it will not be displayed in these parts of the interface. Some countries and regions are only displayed in the grid view.

Countries

Grid View

The following countries are displayed in the grid view of the **Geo Dashboard**.

A – D

Afghanistan
Aland Islands
Albania
Algeria
American Samoa
Andorra
Angola
Anguilla
Antarctica
Antigua and Barbuda
Argentina
Armenia
Aruba
Asia/Pacific Region
Australia
Austria
Azerbaijan
Bahamas
Bahrain
Bangladesh
Barbados
Belarus
Belgium
Belize
Benin
Bermuda
Bhutan
Bolivia
Bonaire, Saint Eustatius, and Saba
Bosnia and Herzegovina
Botswana
Bouvet Island
Brazil
British Indian Ocean Territory
Brunei Darussalam
Bulgaria
Burkina Faso
Burundi
Cambodia
Cameroon
Canada
Cape Verde
Cayman Islands
Central African Republic
Chad
Chile
China
Christmas Island
Cocos (Keeling) Islands
Colombia
Comoros
Congo
Congo, The Democratic Republic of the
Cook Islands
Costa Rica
Cote d'Ivoire
Croatia
Cuba
Curacao
Cyprus
Czech Republic
Denmark
Djibouti
Dominica
Dominican Republic

E – K

Ecuador
Egypt
El Salvador
Equatorial Guinea
Eritrea
Estonia
Ethiopia
Europe
Falkland Islands
(Malvinas)
Faroe Islands
Fiji
Finland
France
French Guiana
French Polynesia
French Southern
Territories
Gabon
Gambia
Georgia
Germany
Ghana
Gibraltar
Greece
Greenland
Grenada
Guadeloupe
Guam
Guatemala
Guernsey
Guinea
Guinea-Bissau
Guyana
Haiti
Heard Island and
McDonald Islands
Holy See (Vatican City
State)
Honduras
Hong Kong
Hungary
Iceland
India
Indonesia
Iran, Islamic Republic
of
Iraq
Ireland
Isle of Man
Israel
Italy
Jamaica
Japan
Jersey
Jordan
Kazakhstan
Kenya
Kiribati
Korea, Democratic
People's Republic of
Korea, Republic of
Kuwait
Kyrgyzstan

L – Q

Lao People's
Democratic Republic
Latvia
Lebanon
Lesotho
Liberia
Libyan Arab
Jamahiriya
Liechtenstein
Lithuania
Luxembourg
Macao
Macedonia
Madagascar
Malawi
Malaysia
Maldives
Mali
Malta
Marshall Islands
Martinique
Mauritania
Mauritius
Mayotte
Mexico
Micronesia, Federated
States of
Moldova, Republic of
Monaco
Mongolia
Montenegro
Montserrat
Morocco
Mozambique
Myanmar
Namibia
Nauru
Nepal
Netherlands
New Caledonia
New Zealand
Nicaragua
Niger
Nigeria
Niue
Norfolk Island
Northern Mariana
Islands
Norway
Oman
Pakistan
Palau
Palestinian Territory
Panama
Papua New Guinea
Paraguay
Peru
Philippines
Pitcairn
Poland
Portugal
Puerto Rico
Qatar

R – Z

Reunion
Romania
Russian Federation
Rwanda
Saint Barthelemy
Saint Helena
Saint Kitts and Nevis
Saint Lucia
Saint Martin
Saint Pierre and Miquelon
Saint Vincent and the Grenadines
Samoa
San Marino
Sao Tome and Principe
Saudi Arabia
Senegal
Serbia
Seychelles
Sierra Leone
Singapore
Sint Maarten
Slovakia
Slovenia
Solomon Islands
Somalia
South Africa
South Georgia and the South Sandwich Islands
Spain
Sri Lanka
Sudan
Suriname
Svalbard and Jan Mayen
Swaziland
Sweden
Switzerland
Syrian Arab Republic
Taiwan
Tajikistan
Tanzania, United Republic of
Thailand
Timor-Leste
Togo
Tokelau
Tonga
Trinidad and Tobago
Tunisia
Turkey
Turkmenistan
Turks and Caicos Islands
Tuvalu
Uganda
Ukraine
United Arab Emirates
United Kingdom
United States
United States Minor Outlying Islands
Uruguay
Uzbekistan
Vanuatu
Venezuela
Vietnam
Virgin Islands, British
Virgin Islands, U.S.
Wallis and Futuna
Western Sahara
Yemen
Zambia
Zimbabwe

Map View

The following countries are displayed in the map view of the **Geo Dashboard**.

A – D

Afghanistan
Aland Islands
Albania
Algeria
American Samoa
Andorra
Angola
Anguilla
Antigua and Barbuda
Argentina
Armenia
Aruba
Australia
Austria
Azerbaijan
Bahamas
Bahrain
Baker Island
Bangladesh
Barbados
Belarus
Belgium
Belize
Benin
Bermuda
Bhutan
Bolivia
Bosnia and
Herzegovina
Botswana
Bouvet Island
Brazil
British Indian Ocean
Territory
Brunei Darussalam
Bulgaria
Burkina Faso
Burundi
Cambodia
Cameroon
Canada
Cape Verde
Cayman Islands
Central African
Republic
Chad
Chile
China
Christmas Island
Cocos (Keeling) Island
Colombia
Comoros
Republic of Congo
the Democratic
Republic of the Congo
Cook Islands
Costa Rica
Cote d'Ivoire
Croatia
Cuba
Cyprus
Czech Republic
Denmark
Djibouti
Dominica
Dominican Republic

E – K

Ecuador
Egypt
El Salvador
Equatorial Guinea
Eritrea
Estonia
Ethiopia
Faeroe Islands
Falkland Islands
Fiji
Finland
France
French Guiana
French Polynesia
French Southern and
Antarctic Territories
Gabon
Gambia, The
Georgia
Germany
Ghana
Gibraltar
Glorioso Islands
Greece
Greenland
Grenada
Guadeloupe
Guatemala
Guam
Guinea
Guinea-Bissau
Guernsey
Guyana
Haiti
Heard and Mcdonald
Islands
Honduras
Hong Kong
Howland Island
Hungary
Iceland
India
Indonesia
Iran
Iraq
Ireland
Isle of Man
Israel
Italy
Jamaica
Japan
Jarvis Island
Jersey
Johnston Atoll
Juan De Nova Island
Jordan
Kazakhstan
Kenya
Kiribati

North Korea
South Korea
Kosovo
Kuwait
Kyrgyzstan

L – Q

Lao People's
Democratic Republic
Latvia
Lebanon
Lesotho
Liberia
Libya
Liechtenstein
Lithuania
Luxembourg
Macau
Macedonia
Madagascar
Malawi
Malaysia
Maldives
Mali
Malta
Marshall Islands
Martinique
Mauritania
Mauritius
Mayotte
Mexico
Federated States of
Micronesia
Midway Islands
Moldova
Monaco
Mongolia
Montenegro
Montserrat
Morocco
Mozambique
Myanmar
Namibia
Nauru
Nepal
Netherlands
New Caledonia
New Zealand
Nicaragua
Niger
Nigeria
Niue
Northern Mariana
Islands
Norway
Oman
Pakistan
Palau
Palestinian Territories
Panama
Papua New Guinea
Paraguay
Peru
Philippines
Pitcairn Islands
Poland
Portugal
Puerto Rico
Qatar

R – Z

Reunion
Romania
Russian Federation
Rwanda
Saint Helena
Saint Kitts and Nevis
Saint Lucia
Saint Pierre and Miquelon
Saint Martin
Saint Vincent and the Grenadines
Samoa
San Marino
Sao Tome and Principe
Saudi Arabia
Senegal
Serbia
Seychelles
Sierra Leone
Singapore
Slovakia
Slovenia
Solomon Islands
Somalia
South Africa
South Georgia and the South Sandwich Islands
Spain
Sri Lanka
Sudan
Suriname
Svalbard and Jan Mayen
Swaziland
Sweden
Switzerland
Syrian Arab Republic
Taiwan
Tajikistan
Tanzania
Thailand
Timor-Leste
Togo
Tonga
Tokelau
Trinidad and Tobago
Tunisia
Turkey
Turkmenistan
Turks and Caicos Islands
Tuvalu
Uganda
Ukraine
United Arab Emirates
United Kingdom
Uruguay
Vanuatu
Uzbekistan
Vatican City
Venezuela
Viet Nam
Virgin Islands, British
Virgin Islands, US
Wake Island
Western Sahara
Wallis and Futuna
Yemen
Zambia
Zimbabwe

Regions

Grid View

The following countries displayed in the grid view of the **Geo Dashboard** also report data by region.

A – G

Afghanistan,
Badakhshan
Afghanistan, Badghis
Afghanistan, Baghlan
Afghanistan, Balkh
Afghanistan, Bamian
Afghanistan, Daykondi
Afghanistan, Farah
Afghanistan, Faryab
Afghanistan, Ghazni
Afghanistan, Ghowr
Afghanistan, Helmand
Afghanistan, Herat
Afghanistan, Jowzjan
Afghanistan, Kabol
Afghanistan, Kandahar
Afghanistan, Kapisa
Afghanistan, Khowst
Afghanistan, Konar
Afghanistan, Kondoz
Afghanistan, Laghman
Afghanistan, Lowgar
Afghanistan,
Nangarhar
Afghanistan, Nimruz
Afghanistan, Nurestan
Afghanistan, Oruzgan
Afghanistan, Paktia
Afghanistan, Paktika
Afghanistan, Panjshir
Afghanistan, Parvan
Afghanistan,
Samangan
Afghanistan, Sar-e Pol
Afghanistan, Takhar
Afghanistan, Vardak
Afghanistan, Zabol
Albania, Berat
Albania, Diber
Albania, Durres
Albania, Elbasan
Albania, Fier
Albania, Gjirokaster
Albania, Korce
Albania, Kukes
Albania, Lezhe
Albania, Shkoder
Albania, Tirane
Albania, Vlore
Algeria, Adrar
Algeria, Ain Defla
Algeria, Ain
Temouchent
Algeria, Alger
Algeria, Annaba
Algeria, Batna
Algeria, Bechar
Algeria, Bejaia
Algeria, Biskra
Algeria, Blida
Algeria, Bordj Bou
Arreridj
Algeria, Bouira
Algeria, Boumerdes
Algeria, Chlef
Algeria, Constantine
Algeria, Djelfa
Algeria, El Bayadh
Algeria, El Oued
Algeria, El Tarf
Algeria, Ghardaia
Algeria, Guelma
Algeria, Illizi

Algeria, Jijel
Algeria, Khenchela
Algeria, Laghouat
Algeria, M'sila
Algeria, Mascara
Algeria, Medea
Algeria, Mila
Algeria, Mostaganem
Algeria, Naama
Algeria, Oran
Algeria, Ouargla
Algeria, Oum el Bouaghi
Algeria, Relizane
Algeria, Saida
Algeria, Setif
Algeria, Sidi Bel Abbes
Algeria, Skikda
Algeria, Souk Ahras
Algeria, Tamanghasset
Algeria, Tebessa
Algeria, Tiaret
Algeria, Tindouf
Algeria, Tipaza
Algeria, Tissemsilt
Algeria, Tizi Ouzou
Algeria, Tlemcen
Andorra, Andorra la Vella
Andorra, Canillo
Andorra, Encamp
Andorra, Escaldes-Engordany
Andorra, La Massana
Andorra, Ordino
Andorra, Sant Julia de Loria
Angola, Bengo
Angola, Benguela
Angola, Bie
Angola, Cabinda
Angola, Cuando Cubango
Angola, Cuanza Norte
Angola, Cuanza Sul
Angola, Cunene
Angola, Huambo
Angola, Huila
Angola, Luanda
Angola, Lunda Norte
Angola, Lunda Sul
Angola, Malanje
Angola, Moxico
Angola, Namibe
Angola, Uige
Angola, Zaire
Antigua and Barbuda, Barbuda
Antigua and Barbuda, Redonda
Antigua and Barbuda, Saint George
Antigua and Barbuda, Saint John
Antigua and Barbuda, Saint Mary
Antigua and Barbuda, Saint Paul
Antigua and Barbuda, Saint Peter
Antigua and Barbuda, Saint Philip
Argentina, Buenos Aires
Argentina, Catamarca
Argentina, Chaco
Argentina, Chubut
Argentina, Cordoba
Argentina, Corrientes

Argentina, Distrito Federal
Argentina, Entre Rios
Argentina, Formosa
Argentina, Jujuy
Argentina, La Pampa
Argentina, La Rioja
Argentina, Mendoza
Argentina, Misiones
Argentina, Neuquen
Argentina, Rio Negro
Argentina, Salta
Argentina, San Juan
Argentina, San Luis
Argentina, Santa Cruz
Argentina, Santa Fe
Argentina, Santiago del Estero
Argentina, Tierra del Fuego
Argentina, Tucuman
Armenia, Aragatsotn
Armenia, Ararat
Armenia, Armavir
Armenia, Geghark'unik'
Armenia, Kotayk'
Armenia, Lorri
Armenia, Shirak
Armenia, Syunik'
Armenia, Tavush
Armenia, Vayots' Dzor
Armenia, Yerevan
Australia, Australian Capital Territory
Australia, New South Wales
Australia, Northern Territory
Australia, Queensland
Australia, South Australia
Australia, Tasmania
Australia, Victoria
Australia, Western Australia
Austria, Burgenland
Austria, Karnten
Austria, Niederosterreich
Austria, Oberosterreich
Austria, Salzburg
Austria, Steiermark
Austria, Tirol
Austria, Vorarlberg
Austria, Wien
Azerbaijan, Abseron
Azerbaijan, Agcabadi
Azerbaijan, Agdam
Azerbaijan, Agdas
Azerbaijan, Agstafa
Azerbaijan, Agsu
Azerbaijan, Ali Bayramli
Azerbaijan, Astara
Azerbaijan, Baki
Azerbaijan, Balakan
Azerbaijan, Barda
Azerbaijan, Beylaqan
Azerbaijan, Bilasuvar
Azerbaijan, Cabrayil
Azerbaijan, Calilabad
Azerbaijan, Daskasan
Azerbaijan, Davaci
Azerbaijan, Fuzuli
Azerbaijan, Gadabay
Azerbaijan, Ganca
Azerbaijan, Goranboy
Azerbaijan, Goycay

Azerbaijan, Haciqabul
Azerbaijan, Imisli
Azerbaijan, Ismayilli
Azerbaijan, Kalbacar
Azerbaijan, Kurdamir
Azerbaijan, Lacin
Azerbaijan, Lankaran
Azerbaijan, Lankaran
Azerbaijan, Lerik
Azerbaijan, Masalli
Azerbaijan, Mingacevir
Azerbaijan, Naftalan
Azerbaijan, Naxcivan
Azerbaijan, Neftcala
Azerbaijan, Oguz
Azerbaijan, Qabala
Azerbaijan, Qax
Azerbaijan, Qazax
Azerbaijan, Qobustan
Azerbaijan, Quba
Azerbaijan, Qubadli
Azerbaijan, Qusar
Azerbaijan, Saatli
Azerbaijan, Sabirabad
Azerbaijan, Saki
Azerbaijan, Saki
Azerbaijan, Salyan
Azerbaijan, Samaxi
Azerbaijan, Samkir
Azerbaijan, Samux
Azerbaijan, Siyazan
Azerbaijan, Sumqayit
Azerbaijan, Susa
Azerbaijan, Susa
Azerbaijan, Tartar
Azerbaijan, Tovuz
Azerbaijan, Ucar
Azerbaijan, Xacmaz
Azerbaijan, Xankandi
Azerbaijan, Xanlar
Azerbaijan, Xizi
Azerbaijan, Xocali
Azerbaijan, Xocavand
Azerbaijan, Yardimli
Azerbaijan, Yevlax
Azerbaijan, Yevlax
Azerbaijan, Zangilan
Azerbaijan, Zaqatala
Azerbaijan, Zardab
Bahrain, Al Asimah
Bahrain, Al Hadd
Bahrain, Al Janubiyah
Bahrain, Al Manamah
Bahrain, Al Mintaqah al Gharbiyah
Bahrain, Al Mintaqah al Wusta
Bahrain, Al Mintaqah ash Shamaliyah
Bahrain, Al Muharraq
Bahrain, Al Wusta
Bahrain, Ar Rifa
Bahrain, Ash Shamaliyah
Bahrain, Jidd Hafs
Bahrain, Madinat
Bahrain, Madinat Hamad
Bahrain, Mintaqat Juzur Hawar
Bahrain, Sitrah
Bangladesh, Barisal
Bangladesh, Chittagong
Bangladesh, Dhaka
Bangladesh, Khulna
Bangladesh, Rajshahi
Bangladesh, Sylhet
Barbados, Christ

Church
Barbados, Saint Andrew
Barbados, Saint George
Barbados, Saint James
Barbados, Saint John
Barbados, Saint Joseph
Barbados, Saint Lucy
Barbados, Saint Michael
Barbados, Saint Peter
Barbados, Saint Philip
Barbados, Saint Thomas
Belarus, Brestskaya Voblasts'
Belarus, Homyel'skaya Voblasts'
Belarus, Hrodzyenskaya Voblasts'
Belarus, Mahilyowskaya Voblasts'
Belarus, Minsk
Belarus, Minskaya Voblasts'
Belarus, Vitsyebskaya Voblasts'
Belgium, Antwerpen
Belgium, Brabant Wallon
Belgium, Brussels Hoofdstedelijk Gewest
Belgium, Hainaut
Belgium, Liege
Belgium, Limburg
Belgium, Luxembourg
Belgium, Namur
Belgium, Oost-Vlaanderen
Belgium, Vlaams-Brabant
Belgium, West-Vlaanderen
Belize, Belize
Belize, Cayo
Belize, Corozal
Belize, Orange Walk
Belize, Stann Creek
Belize, Toledo
Benin, Alibori
Benin, Atakora
Benin, Atlanyique
Benin, Borgou
Benin, Collines
Benin, Donga
Benin, Kouffo
Benin, Littoral
Benin, Mono
Benin, Oueme
Benin, Plateau
Benin, Zou
Bermuda, Devonshire
Bermuda, Hamilton
Bermuda, Hamilton
Bermuda, Paget
Bermuda, Pembroke
Bermuda, Saint George
Bermuda, Saint George's
Bermuda, Sandys
Bermuda, Smiths
Bermuda,

Southampton
Bermuda, Warwick
Bhutan, Bumthang
Bhutan, Chhukha
Bhutan, Chirang
Bhutan, Daga
Bhutan, Geylegphug
Bhutan, Ha
Bhutan, Lhuntshi
Bhutan, Mongar
Bhutan, Paro
Bhutan, Pemagatsel
Bhutan, Punakha
Bhutan, Samchi
Bhutan, Samdrup
Bhutan, Shemgang
Bhutan, Tashigang
Bhutan, Thimphu
Bhutan, Tongsa
Bhutan, Wangdi
Phodrang
Bolivia, Chuquisaca
Bolivia, Cochabamba
Bolivia, El Beni
Bolivia, La Paz
Bolivia, Oruro
Bolivia, Pando
Bolivia, Potosi
Bolivia, Santa Cruz
Bolivia, Tarija
Bosnia and
Herzegovina,
Federation of Bosnia
and Herzegovina
Bosnia and
Herzegovina,
Republika Srpska
Botswana, Central
Botswana, Ghanzi
Botswana, Kgalagadi
Botswana, Kgatleng
Botswana, Kweneng
Botswana, North-East
Botswana, North-West
Botswana, South-East
Botswana, Southern
Brazil, Acre
Brazil, Alagoas
Brazil, Amapa
Brazil, Amazonas
Brazil, Bahia
Brazil, Ceara
Brazil, Distrito Federal
Brazil, Espirito Santo
Brazil, Goias
Brazil, Maranhao
Brazil, Mato Grosso
Brazil, Mato Grosso
do Sul
Brazil, Minas Gerais
Brazil, Para
Brazil, Paraiba
Brazil, Parana
Brazil, Pernambuco
Brazil, Piaui
Brazil, Rio de Janeiro
Brazil, Rio Grande do
Norte
Brazil, Rio Grande do
Sul
Brazil, Rondonia
Brazil, Roraima
Brazil, Santa Catarina
Brazil, Sao Paulo
Brazil, Sergipe
Brazil, Tocantins
Brunei Darussalam,
Alibori
Brunei Darussalam,

Belait
Brunei Darussalam, Brunei and Muara
Brunei Darussalam, Collines
Brunei Darussalam, Donga
Brunei Darussalam, Kouffo
Brunei Darussalam, Littoral
Brunei Darussalam, Oueme
Brunei Darussalam, Plateau
Brunei Darussalam, Temburong
Brunei Darussalam, Tutong
Brunei Darussalam, Zou
Bulgaria, Blagoevgrad
Bulgaria, Burgas
Bulgaria, Dobrich
Bulgaria, Gabrovo
Bulgaria, Grad Sofiya
Bulgaria, Khaskovo
Bulgaria, Kurdzhali
Bulgaria, Kyustendil
Bulgaria, Lovech
Bulgaria, Mikhaylovgrad
Bulgaria, Montana
Bulgaria, Pazardzhik
Bulgaria, Pernik
Bulgaria, Pleven
Bulgaria, Plovdiv
Bulgaria, Razgrad
Bulgaria, Ruse
Bulgaria, Shumen
Bulgaria, Silistra
Bulgaria, Sliven
Bulgaria, Smolyan
Bulgaria, Sofiya
Bulgaria, Stara Zagora
Bulgaria, Turgovishte
Bulgaria, Varna
Bulgaria, Veliko Turnovo
Bulgaria, Vidin
Bulgaria, Vratsa
Bulgaria, Yambol
Burkina Faso, Bale
Burkina Faso, Bam
Burkina Faso, Banwa
Burkina Faso, Bazega
Burkina Faso, Bougouriba
Burkina Faso, Boulgou
Burkina Faso, Boulkiemde
Burkina Faso, Ganzourgou
Burkina Faso, Gnagna
Burkina Faso, Gourma
Burkina Faso, Houet
Burkina Faso, Ioba
Burkina Faso, Kadiogo
Burkina Faso, Kenedougou
Burkina Faso, Komoe
Burkina Faso, Komondjari
Burkina Faso, Kompienga
Burkina Faso, Kossi
Burkina Faso, Koulpelogo
Burkina Faso,

Kouritenga
Burkina Faso,
Kourweogo
Burkina Faso, Leraba
Burkina Faso, Loroum
Burkina Faso,
Mouhoun
Burkina Faso,
Namentenga
Burkina Faso, Naouri
Burkina Faso, Nayala
Burkina Faso,
Noumbiel
Burkina Faso,
Oubritenga
Burkina Faso, Oudalan
Burkina Faso, Passore
Burkina Faso, Poni
Burkina Faso, Sanguie
Burkina Faso,
Sanmatenga
Burkina Faso, Seno
Burkina Faso, Sissili
Burkina Faso, Soum
Burkina Faso, Sourou
Burkina Faso, Tapoa
Burkina Faso, Tuy
Burkina Faso, Yagha
Burkina Faso, Yatenga
Burkina Faso, Ziro
Burkina Faso,
Zondoma
Burkina Faso,
Zoundweogo
Burundi, Bubanza
Burundi, Bujumbura
Burundi, Bururi
Burundi, Cankuzo
Burundi, Cibitoke
Burundi, Gitega
Burundi, Karuzi
Burundi, Kayanza
Burundi, Kirundo
Burundi, Makamba
Burundi, Muramvya
Burundi, Muyinga
Burundi, Mwaro
Burundi, Ngozi
Burundi, Rutana
Burundi, Ruyigi
Cambodia, Banteay
Meanchey
Cambodia,
Batdambang
Cambodia,
Batdambang
Cambodia, Kampong
Cham
Cambodia, Kampong
Chhnang
Cambodia, Kampong
Speu
Cambodia, Kampong
Thum
Cambodia, Kampot
Cambodia, Kandal
Cambodia, Koh Kong
Cambodia, Kracheh
Cambodia, Mondulkiri
Cambodia, Pailin
Cambodia, Phnum
Penh
Cambodia, Preah
Vihear
Cambodia, Prey Veng
Cambodia, Pursat
Cambodia, Ratanakiri
Kiri
Cambodia, Siem Reap

Cambodia, Stung Treng
Cambodia, Svay Rieng
Cambodia, Takeo
Cameroon, Adamaoua
Cameroon, Centre
Cameroon, Est
Cameroon, Extreme-Nord
Cameroon, Littoral
Cameroon, Nord
Cameroon, Nord-Ouest
Cameroon, Ouest
Cameroon, Sud
Cameroon, Sud-Ouest
Canada, Alberta
Canada, British Columbia
Canada, Manitoba
Canada, New Brunswick
Canada, Newfoundland
Canada, Northwest Territories
Canada, Nova Scotia
Canada, Nunavut
Canada, Ontario
Canada, Prince Edward Island
Canada, Quebec
Canada, Saskatchewan
Canada, Yukon Territory
Cape Verde, Boa Vista
Cape Verde, Brava
Cape Verde, Maio
Cape Verde, Mosteiros
Cape Verde, Paul
Cape Verde, Praia
Cape Verde, Ribeira Grande
Cape Verde, Sal
Cape Verde, Santa Catarina
Cape Verde, Santa Cruz
Cape Verde, Sao Domingos
Cape Verde, Sao Filipe
Cape Verde, Sao Miguel
Cape Verde, Sao Nicolau
Cape Verde, Sao Vicente
Cape Verde, Tarrafal
Cayman Islands, Creek
Cayman Islands, Eastern
Cayman Islands, Midland
Cayman Islands, South Town
Cayman Islands, Spot Bay
Cayman Islands, Stake Bay
Cayman Islands, West End
Cayman Islands, Western
Central African Republic, Bamingui-Bangoran

Central African Republic, Bangui
Central African Republic, Basse-Kotto
Central African Republic, Cuvette-Ouest
Central African Republic, Haut-Mbomou
Central African Republic, Haute-Kotto
Central African Republic, Kemo
Central African Republic, Lobaye
Central African Republic, Mambere-Kadei
Central African Republic, Mbomou
Central African Republic, Nana-Grebizi
Central African Republic, Nana-Mambere
Central African Republic, Ombella-Mpoko
Central African Republic, Ouaka
Central African Republic, Ouham
Central African Republic, Ouham-Pende
Central African Republic, Sangha-Mbaere
Chad, Batha
Chad, Biltine
Chad, Borkou-Ennedi-Tibesti
Chad, Chari-Baguirmi
Chad, Guera
Chad, Kanem
Chad, Lac
Chad, Logone Occidental
Chad, Logone Oriental
Chad, Mayo-Kebbi
Chad, Moyen-Chari
Chad, Ouaddai
Chad, Salamat
Chad, Tandjile
Chile, Aisen del General Carlos Ibanez del Campo
Chile, Antofagasta
Chile, Araucania
Chile, Arica y Parinacota
Chile, Atacama
Chile, Bio-Bio
Chile, Coquimbo
Chile, Libertador General Bernardo O'Higgins
Chile, Los Lagos
Chile, Los Lagos
Chile, Los Rios
Chile, Magallanes y de la Antartica Chilena
Chile, Maule
Chile, Region Metropolitana
Chile, Tarapaca
Chile, Tarapaca

Chile, Valparaiso
China, Anhui
China, Beijing
China, Chongqing
China, Fujian
China, Gansu
China, Guangdong
China, Guangxi
China, Guizhou
China, Hainan
China, Hebei
China, Heilongjiang
China, Henan
China, Hubei
China, Hunan
China, Jiangsu
China, Jiangxi
China, Jilin
China, Liaoning
China, Nei Mongol
China, Ningxia
China, Qinghai
China, Shaanxi
China, Shandong
China, Shanghai
China, Shanxi
China, Sichuan
China, Tianjin
China, Xinjiang
China, Xizang
China, Yunnan
China, Zhejiang
Colombia, Amazonas
Colombia, Antioquia
Colombia, Arauca
Colombia, Atlantico
Colombia, Bolivar
Colombia, Bolivar Department
Colombia, Boyaca
Colombia, Boyaca Department
Colombia, Caldas
Colombia, Caldas Department
Colombia, Caqueta
Colombia, Casanare
Colombia, Cauca
Colombia, Cesar
Colombia, Choco
Colombia, Cordoba
Colombia, Cundinamarca
Colombia, Distrito Especial
Colombia, Guainia
Colombia, Guaviare
Colombia, Huila
Colombia, La Guajira
Colombia, Magdalena
Colombia, Magdalena Department
Colombia, Meta
Colombia, Narino
Colombia, Norte de Santander
Colombia, Putumayo
Colombia, Quindio
Colombia, Risaralda
Colombia, San Andres y Providencia
Colombia, Santander
Colombia, Sucre
Colombia, Tolima
Colombia, Valle del Cauca
Colombia, Vaupes
Colombia, Vichada
Comoros, Anjouan

Comoros, Grande Comore
Comoros, Moheli
Congo, Bouenza
Congo, Brazzaville
Congo, Cuvette
Congo, Cuvette-Ouest
Congo, Kouilou
Congo, Lekoumou
Congo, Likouala
Congo, Niari
Congo, Plateaux
Congo, Pool
Congo, Sangha
Congo, The Democratic Republic of the, Bandundu
Congo, The Democratic Republic of the, Bas-Congo
Congo, The Democratic Republic of the, Equateur
Congo, The Democratic Republic of the, Kasai-Oriental
Congo, The Democratic Republic of the, Katanga
Congo, The Democratic Republic of the, Kinshasa
Congo, The Democratic Republic of the, Maniema
Congo, The Democratic Republic of the, Nord-Kivu
Congo, The Democratic Republic of the, Orientale
Congo, The Democratic Republic of the, Sud-Kivu
Costa Rica, Alajuela
Costa Rica, Cartago
Costa Rica, Guanacaste
Costa Rica, Heredia
Costa Rica, Limon
Costa Rica, Puntarenas
Costa Rica, San Jose
Cote D'Ivoire, Agneby
Cote D'Ivoire, Bafing
Cote D'Ivoire, Bas-Sassandra
Cote D'Ivoire, Denguele
Cote D'Ivoire, Dix-Huit Montagnes
Cote D'Ivoire, Fromager
Cote D'Ivoire, Haut-Sassandra
Cote D'Ivoire, Lacs
Cote D'Ivoire, Lagunes
Cote D'Ivoire, Marahoue
Cote D'Ivoire, Moyen-Cavally
Cote D'Ivoire, Moyen-Comoe
Cote D'Ivoire, N'zi-Comoe
Cote D'Ivoire, Savanes
Cote D'Ivoire, Sud-Bandama
Cote D'Ivoire, Sud-

Comoe
Cote D'Ivoire, Vallee du Bandama
Cote D'Ivoire, Worodougou
Cote D'Ivoire, Zanzan
Croatia, Bjelovarsko-Bilogorska
Croatia, Brodsko-Posavska
Croatia, Dubrovacko-Neretvanska
Croatia, Grad Zagreb
Croatia, Istarska
Croatia, Karlovacka
Croatia, Koprivnicko-Krizevacka
Croatia, Krapinsko-Zagorska
Croatia, Licko-Senjska
Croatia, Medimurska
Croatia, Osjecko-Baranjska
Croatia, Pozesko-Slavonska
Croatia, Primorsko-Goranska
Croatia, Sibensko-Kninska
Croatia, Sisacko-Moslavacka
Croatia, Splitsko-Dalmatinska
Croatia, Varazdinska
Croatia, Viroviticko-Podravska
Croatia, Vukovarsko-Srijemska
Croatia, Zadarska
Croatia, Zagrebacka
Cuba, Camaguey
Cuba, Ciego de Avila
Cuba, Cienfuegos
Cuba, Ciudad de la Habana
Cuba, Granma
Cuba, Guantanamo
Cuba, Holguin
Cuba, Isla de la Juventud
Cuba, La Habana
Cuba, Las Tunas
Cuba, Matanzas
Cuba, Pinar del Rio
Cuba, Sancti Spiritus
Cuba, Santiago de Cuba
Cuba, Villa Clara
Cyprus, Famagusta
Cyprus, Kyrenia
Cyprus, Larnaca
Cyprus, Limassol
Cyprus, Nicosia
Cyprus, Paphos
Czech Republic, Hlavni mesto Praha
Czech Republic, Jihocesky kraj
Czech Republic, Jihomoravsky kraj
Czech Republic, Karlovarsky kraj
Czech Republic, Kralovehradecky kraj
Czech Republic, Liberecky kraj
Czech Republic, Moravskoslezsky kraj
Czech Republic,

Olomoucky kraj
Czech Republic,
Pardubicky kraj
Czech Republic,
Plzensky kraj
Czech Republic,
Stredocesky kraj
Czech Republic,
Ustecky kraj
Czech Republic,
Vysocina
Czech Republic,
Zlinsky kraj
Denmark,
Hovedstaden
Denmark, Midtjylland
Denmark, Nordjylland
Denmark, Sjelland
Denmark, Syddanmark
Djibouti, Ali Sabieh
Djibouti, Arta
Djibouti, Dikhil
Djibouti, Djibouti
Djibouti, Obock
Djibouti, Tadjoura
Dominica, Saint
Andrew
Dominica, Saint David
Dominica, Saint
George
Dominica, Saint John
Dominica, Saint
Joseph
Dominica, Saint Luke
Dominica, Saint Mark
Dominica, Saint
Patrick
Dominica, Saint Paul
Dominica, Saint Peter
Dominican Republic,
Azua
Dominican Republic,
Baoruco
Dominican Republic,
Barahona
Dominican Republic,
Dajabon
Dominican Republic,
Distrito Nacional
Dominican Republic,
Distrito Nacional
Dominican Republic,
Duarte
Dominican Republic,
El Seibo
Dominican Republic,
Elias Pina
Dominican Republic,
Espaillat
Dominican Republic,
Hato Mayor
Dominican Republic,
Independencia
Dominican Republic,
La Altagracia
Dominican Republic,
La Romana
Dominican Republic,
La Vega
Dominican Republic,
Maria Trinidad
Sanchez
Dominican Republic,
Monsenor Nouel
Dominican Republic,
Monte Cristi
Dominican Republic,
Monte Plata
Dominican Republic,

Pedernales
Dominican Republic, Peravia
Dominican Republic, Peravia
Dominican Republic, Puerto Plata
Dominican Republic, Salcedo
Dominican Republic, Samana
Dominican Republic, San Cristobal
Dominican Republic, San Jose de Ocoa
Dominican Republic, San Juan
Dominican Republic, San Pedro De Macoris
Dominican Republic, Sanchez Ramirez
Dominican Republic, Santiago
Dominican Republic, Santiago Rodriguez
Dominican Republic, Santo Domingo
Dominican Republic, Valverde
Ecuador, Azuay
Ecuador, Bolivar
Ecuador, Canar
Ecuador, Carchi
Ecuador, Chimborazo
Ecuador, Cotopaxi
Ecuador, El Oro
Ecuador, Esmeraldas
Ecuador, Galapagos
Ecuador, Guayas
Ecuador, Imbabura
Ecuador, Loja
Ecuador, Los Rios
Ecuador, Manabi
Ecuador, Morona-Santiago
Ecuador, Napo
Ecuador, Orellana
Ecuador, Pastaza
Ecuador, Pichincha
Ecuador, Sucumbios
Ecuador, Tungurahua
Ecuador, Zamora-Chinchipe
Egypt, Ad Daqahliyah
Egypt, Al Bahr al Ahmar
Egypt, Al Buhayrah
Egypt, Al Fayyum
Egypt, Al Gharbiyah
Egypt, Al Iskandariyah
Egypt, Al Isma'iliyah
Egypt, Al Jizah
Egypt, Al Minufiyah
Egypt, Al Minya
Egypt, Al Qahirah
Egypt, Al Qalyubiyah
Egypt, Al Wadi al Jadid
Egypt, As Suways
Egypt, Ash Sharqiyah
Egypt, Aswan
Egypt, Asyut
Egypt, Bani Suwayf
Egypt, Bur Sa'id
Egypt, Dumyat
Egypt, Janub Sina'
Egypt, Kafr ash Shaykh
Egypt, Matruh
Egypt, Qina

Egypt, Shamal Sina'
Egypt, Suhaj
El Salvador,
Ahuachapan
El Salvador, Cabanas
El Salvador,
Chalatenango
El Salvador, Cuscatlan
El Salvador, La
Libertad
El Salvador, La Paz
El Salvador, La Union
El Salvador, Morazan
El Salvador, San
Miguel
El Salvador, San
Salvador
El Salvador, San
Vicente
El Salvador, Santa
Ana
El Salvador,
Sonsonate
El Salvador, Usulutan
Equatorial Guinea,
Annobon
Equatorial Guinea,
Bioko Norte
Equatorial Guinea,
Bioko Sur
Equatorial Guinea,
Centro Sur
Equatorial Guinea,
Kie-Ntem
Equatorial Guinea,
Litoral
Equatorial Guinea,
Wele-Nzas
Eritrea, Anseba
Eritrea, Debub
Eritrea, Debubawi
K'eyih Bahri
Eritrea, Gash Barka
Eritrea, Ma'akel
Eritrea, Semenawi
K'eyih Bahri
Estonia, Harjumaa
Estonia, Hiiumaa
Estonia, Ida-Virumaa
Estonia, Jarvamaa
Estonia, Jogevamaa
Estonia, Kohtla-Jarve
Estonia, Laane-
Virumaa
Estonia, Laanemaa
Estonia, Narva
Estonia, Parnu
Estonia, Parnumaa
Estonia, Polvamaa
Estonia, Raplamaa
Estonia, Saaremaa
Estonia, Sillamae
Estonia, Tallinn
Estonia, Tartu
Estonia, Tartumaa
Estonia, Valgamaa
Estonia, Viljandimaa
Estonia, Vorumaa
Ethiopia, Adis Abeba
Ethiopia, Afar
Ethiopia, Amara
Ethiopia, Binshangul
Gumuz
Ethiopia, Dire Dawa
Ethiopia, Gambela
Hizboch
Ethiopia, Hareri Hizb
Ethiopia, Oromiya
Ethiopia, Sumale

Ethiopia, Tigray
Ethiopia, YeDebub Biheroch Bihereseboch na Hizboch
Fiji, Central
Fiji, Eastern
Fiji, Northern
Fiji, Rotuma
Fiji, Western
Finland, Aland
Finland, Eastern Finland
Finland, Lapland
Finland, Oulu
Finland, Southern Finland
Finland, Western Finland
France, Alsace
France, Aquitaine
France, Auvergne
France, Basse-Normandie
France, Bourgogne
France, Bretagne
France, Centre
France, Champagne-Ardenne
France, Corse
France, Franche-Comte
France, Haute-Normandie
France, Ile-de-France
France, Languedoc-Roussillon
France, Limousin
France, Lorraine
France, Midi-Pyrenees
France, Nord-Pas-de-Calais
France, Pays de la Loire
France, Picardie
France, Poitou-Charentes
France, Provence-Alpes-Cote d'Azur
France, Rhone-Alpes
Gabon, Estuaire
Gabon, Haut-Ogooue
Gabon, Moyen-Ogooue
Gabon, Ngounie
Gabon, Nyanga
Gabon, Ogooue-Ivindo
Gabon, Ogooue-Lolo
Gabon, Ogooue-Maritime
Gabon, Woleu-Ntem
Gambia, Banjul
Gambia, Central River
Gambia, Lower River
Gambia, North Bank
Gambia, Upper River
Gambia, Western
Georgia, Abashis Raioni
Georgia, Abkhazia
Georgia, Adigenis Raioni
Georgia, Ajaria
Georgia, Akhalgoris Raioni
Georgia, Akhalk'alak'is Raioni
Georgia, Akhalts'ikhis Raioni

Georgia, Akhmetis Raioni
Georgia, Ambrolauris Raioni
Georgia, Aspindzis Raioni
Georgia, Baghdat'is Raioni
Georgia, Bolnisis Raioni
Georgia, Borjomis Raioni
Georgia, Ch'khorotsqus Raioni
Georgia, Ch'okhatauris Raioni
Georgia, Chiat'ura
Georgia, Dedop'listsqaros Raioni
Georgia, Dmanisis Raioni
Georgia, Dushet'is Raioni
Georgia, Gardabanis Raioni
Georgia, Gori
Georgia, Goris Raioni
Georgia, Gurjaanis Raioni
Georgia, Javis Raioni
Georgia, K'arelis Raioni
Georgia, K'ut'aisi
Georgia, Kaspis Raioni
Georgia, Kharagaulis Raioni
Georgia, Khashuris Raioni
Georgia, Khobis Raioni
Georgia, Khonis Raioni
Georgia, Lagodekhis Raioni
Georgia, Lanch'khut'is Raioni
Georgia, Lentekhis Raioni
Georgia, Marneulis Raioni
Georgia, Martvilis Raioni
Georgia, Mestiis Raioni
Georgia, Mts'khet'is Raioni
Georgia, Ninotsmindis Raioni
Georgia, Onis Raioni
Georgia, Ozurget'is Raioni
Georgia, P'ot'i
Georgia, Qazbegis Raioni
Georgia, Qvarlis Raioni
Georgia, Rust'avi
Georgia, Sach'kheris Raioni
Georgia, Sagarejos Raioni
Georgia, Samtrediis Raioni
Georgia, Senakis Raioni
Georgia, Sighnaghis Raioni
Georgia, T'bilisi
Georgia, T'elavis Raioni

Georgia, T'erjolis Raioni
Georgia, T'et'ritsqaros Raioni
Georgia, T'ianet'is Raioni
Georgia, Tqibuli
Georgia, Ts'ageris Raioni
Georgia, Tsalenjikhis Raioni
Georgia, Tsalkis Raioni
Georgia, Tsqaltubo
Georgia, Vanis Raioni
Georgia, Zestap'onis Raioni
Georgia, Zugdidi
Georgia, Zugdidis Raioni
Germany, Baden-Wurttemberg
Germany, Bayern
Germany, Berlin
Germany, Brandenburg
Germany, Bremen
Germany, Hamburg
Germany, Hessen
Germany, Mecklenburg-Vorpommern
Germany, Niedersachsen
Germany, NordrheinWestfalen
Germany, Rheinland-Pfalz
Germany, Saarland
Germany, Sachsen
Germany, Sachsen-Anhalt
Germany, Schleswig-Holstein
Germany, Thuringen
Ghana, Ashanti
Ghana, Brong-Ahafo
Ghana, Central
Ghana, Eastern
Ghana, Greater Accra
Ghana, Northern
Ghana, Upper East
Ghana, Upper West
Ghana, Volta
Ghana, Western
Greece, Aitolia kai Akarnania
Greece, Akhaia
Greece, Argolis
Greece, Arkadhia
Greece, Arta
Greece, Attiki
Greece, Dhodhekanisos
Greece, Drama
Greece, Evritania
Greece, Evros
Greece, Evvoia
Greece, Florina
Greece, Fokis
Greece, Fthiotis
Greece, Grevena
Greece, Ilia
Greece, Imathia
Greece, Ioannina
Greece, Iraklion
Greece, Kardhitsa
Greece, Kastoria
Greece, Kavala

Greece, Kefallinia
Greece, Kerkira
Greece, Khalkidhiki
Greece, Khania
Greece, Khios
Greece, Kikladhes
Greece, Kilkis
Greece, Korinthia
Greece, Kozani
Greece, Lakonia
Greece, Larisa
Greece, Lasithi
Greece, Lesvos
Greece, Levkas
Greece, Magnisia
Greece, Messinia
Greece, Pella
Greece, Pieria
Greece, Preveza
Greece, Rethimni
Greece, Rodhopi
Greece, Samos
Greece, Serrai
Greece, Thesprotia
Greece, Thessaloniki
Greece, Trikala
Greece, Voiotia
Greece, Xanthi
Greece, Zakinthos
Greenland,
Nordgronland
Greenland,
Ostgronland
Greenland,
Vestgronland
Grenada, Saint
Andrew
Grenada, Saint David
Grenada, Saint
George
Grenada, Saint John
Grenada, Saint Mark
Grenada, Saint Patrick
Guatemala, Alta
Verapaz
Guatemala, Baja
Verapaz
Guatemala,
Chimaltenango
Guatemala,
Chiquimula
Guatemala, El
Progreso
Guatemala, Escuintla
Guatemala,
Guatemala
Guatemala,
Huehuetenango
Guatemala, Izabal
Guatemala, Jalapa
Guatemala, Jutiapa
Guatemala, Peten
Guatemala,
Quetzaltenango
Guatemala, Quiche
Guatemala, Retalhuleu
Guatemala,
Sacatepequez
Guatemala, San
Marcos
Guatemala, Santa
Rosa
Guatemala, Solola
Guatemala,
Suchitepequez
Guatemala,
Totonicapan
Guatemala, Zacapa
Guinea-Bissau, Bafata

Guinea-Bissau, Biombo
Guinea-Bissau, Bissau
Guinea-Bissau, Bolama
Guinea-Bissau, Cacheu
Guinea-Bissau, Gabu
Guinea-Bissau, Oio
Guinea-Bissau, Quinara
Guinea-Bissau, Tombali
Guinea, Beyla
Guinea, Boffa
Guinea, Boke
Guinea, Conakry
Guinea, Coyah
Guinea, Dabola
Guinea, Dalaba
Guinea, Dinguiraye
Guinea, Dubreka
Guinea, Faranah
Guinea, Forecariah
Guinea, Fria
Guinea, Gaoual
Guinea, Gueckedou
Guinea, Kankan
Guinea, Kerouane
Guinea, Kindia
Guinea, Kissidougou
Guinea, Koubia
Guinea, Koundara
Guinea, Kouroussa
Guinea, Labe
Guinea, Lelouma
Guinea, Lola
Guinea, Macenta
Guinea, Mali
Guinea, Mamou
Guinea, Mandiana
Guinea, Nzerekore
Guinea, Pita
Guinea, Siguiri
Guinea, Telimele
Guinea, Tougue
Guinea, Yomou
Guyana, Barima-Waini
Guyana, Cuyuni-Mazaruni
Guyana, Demerara-Mahaica
Guyana, East Berbice-Corentyne
Guyana, Essequibo Islands-West Demerara
Guyana, Mahaica-Berbice
Guyana, Pomeroon-Supenaam
Guyana, Potaro-Siparuni
Guyana, Upper Demerara-Berbice
Guyana, Upper Takutu-Upper Essequibo

H – M

Haiti, Artibonite
Haiti, Centre
Haiti, Grand' Anse
Haiti, Nippes
Haiti, Nord
Haiti, Nord-Est
Haiti, Nord-Ouest
Haiti, Ouest
Haiti, Sud

Haiti, Sud-Est
Honduras, Atlantida
Honduras, Choluteca
Honduras, Colon
Honduras, Comayagua
Honduras, Copan
Honduras, Cortes
Honduras, El Paraiso
Honduras, Francisco Morazan
Honduras, Gracias a Dios
Honduras, Intibuca
Honduras, Islas de la Bahia
Honduras, La Paz
Honduras, Lempira
Honduras, Ocotepeque
Honduras, Olancho
Honduras, Santa Barbara
Honduras, Valle
Honduras, Yoro
Hungary, Bacs-Kiskun
Hungary, Baranya
Hungary, Bekes
Hungary, Bekescsaba
Hungary, Borsod-Abauj-Zemplen
Hungary, Budapest
Hungary, Csongrad
Hungary, Debrecen
Hungary, Dunaujvaros
Hungary, Eger
Hungary, Erd
Hungary, Fejer
Hungary, Gyor
Hungary, Gyor-Moson-Sopron
Hungary, Hajdu-Bihar
Hungary, Heves
Hungary, Hodmezovasarhely
Hungary, Jasz-Nagykun-Szolnok
Hungary, Kaposvar
Hungary, Kecskemet
Hungary, Komarom-Esztergom
Hungary, Miskolc
Hungary, Nagykanizsa
Hungary, Nograd
Hungary, Nyiregyhaza
Hungary, Pecs
Hungary, Pest
Hungary, Salgotarjan
Hungary, Somogy
Hungary, Sopron
Hungary, Szabolcs-Szatmar-Bereg
Hungary, Szeged
Hungary, Szekesfehervar
Hungary, Szekszard
Hungary, Szolnok
Hungary, Szombathely
Hungary, Tatabanya
Hungary, Tolna
Hungary, Vas
Hungary, Veszprem
Hungary, Veszprem
Hungary, Zala
Hungary, Zalaegerszeg
Iceland, Arnessysla
Iceland, Austur-Hunavatnssysla
Iceland, Austur-

Skaftafellssysla
Iceland,
Borgarfjardarsysla
Iceland,
Eyjafjardarsysla
Iceland,
Gullbringusysla
Iceland, Kjosarsysla
Iceland, Myrasysla
Iceland, Nordur-
Mulasysla
Iceland, Nordur-
Tingeyjarsysla
Iceland, Norourland
Eystra
Iceland, Norourland
Vestra
Iceland,
Rangarvallasysla
Iceland,
Skagafjardarsysla
Iceland, Snafellsnes-
og Hnappadalssysla
Iceland, Strandasysla
Iceland, Sudur-
Mulasysla
Iceland, Sudur-
Tingeyjarsysla
Iceland, Suourland
Iceland, Suournes
Iceland, Vestfiroir
Iceland, Vestur-
Bardastrandarsysla
Iceland, Vestur-
Hunavatnssysla
Iceland, Vestur-
Isafjardarsysla
Iceland, Vestur-
Skaftafellssysla
Iceland, Vesturland
India, Andaman and
Nicobar Islands
India, Andhra Pradesh
India, Arunachal
Pradesh
India, Assam
India, Bihar
India, Chandigarh
India, Chhattisgarh
India, Dadra and
Nagar Haveli
India, Daman and Diu
India, Delhi
India, Goa
India, Gujarat
India, Haryana
India, Himachal
Pradesh
India, Jammu and
Kashmir
India, Jharkhand
India, Karnataka
India, Kerala
India, Lakshadweep
India, Madhya Pradesh
India, Maharashtra
India, Manipur
India, Meghalaya
India, Mizoram
India, Nagaland
India, Orissa
India, Puducherry
India, Punjab
India, Rajasthan
India, Sikkim
India, Tamil Nadu
India, Tripura
India, Uttar Pradesh
India, Uttarakhand

India, West Bengal
Indonesia, Aceh
Indonesia, Bali
Indonesia, Banten
Indonesia, Bengkulu
Indonesia, Gorontalo
Indonesia, Irian Jaya Barat
Indonesia, Jakarta Raya
Indonesia, Jambi
Indonesia, Jawa Barat
Indonesia, Jawa Barat
Indonesia, Jawa Tengah
Indonesia, Jawa Timur
Indonesia, Kalimantan Barat
Indonesia, Kalimantan Selatan
Indonesia, Kalimantan Tengah
Indonesia, Kalimantan Timur
Indonesia, Kepulauan Bangka Belitung
Indonesia, Kepulauan Riau
Indonesia, Lampung
Indonesia, Maluku
Indonesia, Maluku
Indonesia, Maluku Utara
Indonesia, Nusa Tenggara Barat
Indonesia, Nusa Tenggara Timur
Indonesia, Papua
Indonesia, Papua
Indonesia, Riau
Indonesia, Riau
Indonesia, Sulawesi Barat
Indonesia, Sulawesi Selatan
Indonesia, Sulawesi Selatan
Indonesia, Sulawesi Tengah
Indonesia, Sulawesi Tenggara
Indonesia, Sulawesi Utara
Indonesia, Sulawesi Utara
Indonesia, Sumatera Barat
Indonesia, Sumatera Selatan
Indonesia, Sumatera Selatan
Indonesia, Sumatera Utara
Indonesia, Yogyakarta
Iran, Islamic Republic of, Ardabil
Iran, Islamic Republic of, Azarbayjan-e Bakhtari
Iran, Islamic Republic of, Bakhtaran
Iran, Islamic Republic of, Bushehr
Iran, Islamic Republic of, Chahar Mahall va Bakhtiari
Iran, Islamic Republic of, East Azarbaijan
Iran, Islamic Republic

of, Esfahan
Iran, Islamic Republic
of, Fars
Iran, Islamic Republic
of, Gilan
Iran, Islamic Republic
of, Golestan
Iran, Islamic Republic
of, Hamadan
Iran, Islamic Republic
of, Hormozgan
Iran, Islamic Republic
of, Ilam
Iran, Islamic Republic
of, Kerman
Iran, Islamic Republic
of, Kerman
Iran, Islamic Republic
of, Khorasan
Iran, Islamic Republic
of, Khorasan-e Janubi
Iran, Islamic Republic
of, Khorasan-e Razavi
Iran, Islamic Republic
of, Khorasan-e
Shemali
Iran, Islamic Republic
of, Khuzestan
Iran, Islamic Republic
of, Kohkiluyeh va
Buyer Ahmadi
Iran, Islamic Republic
of, Kordestan
Iran, Islamic Republic
of, Lorestan
Iran, Islamic Republic
of, Markazi
Iran, Islamic Republic
of, Markazi
Iran, Islamic Republic
of, Markazi
Iran, Islamic Republic
of, Mazandaran
Iran, Islamic Republic
of, Mazandaran
Iran, Islamic Republic
of, Qazvin
Iran, Islamic Republic
of, Qom
Iran, Islamic Republic
of, Semnan
Iran, Islamic Republic
of, Semnan Province
Iran, Islamic Republic
of, Sistan va
Baluchestan
Iran, Islamic Republic
of, Tehran
Iran, Islamic Republic
of, Yazd
Iran, Islamic Republic
of, Yazd
Iran, Islamic Republic
of, Zanjan
Iran, Islamic Republic
of, Zanjan
Iran, Islamic Republic
of, Zanjan
Iraq, Al Anbar
Iraq, Al Basrah
Iraq, Al Muthanna
Iraq, Al Qadisiyah
Iraq, An Najaf
Iraq, Arbil
Iraq, As Sulaymaniyah
Iraq, At Ta'mim
Iraq, Babil
Iraq, Baghdad
Iraq, Dahuk

Iraq, Dhi Qar
Iraq, Diyala
Iraq, Karbala'
Iraq, Maysan
Iraq, Ninawa
Iraq, Salah ad Din
Iraq, Wasit
Ireland, Carlow
Ireland, Cavan
Ireland, Clare
Ireland, Cork
Ireland, Donegal
Ireland, Dublin
Ireland, Galway
Ireland, Kerry
Ireland, Kildare
Ireland, Kilkenny
Ireland, Laois
Ireland, Leitrim
Ireland, Limerick
Ireland, Longford
Ireland, Louth
Ireland, Mayo
Ireland, Meath
Ireland, Monaghan
Ireland, Offaly
Ireland, Roscommon
Ireland, Sligo
Ireland, Tipperary
Ireland, Waterford
Ireland, Westmeath
Ireland, Wexford
Ireland, Wicklow
Israel, HaDarom
Israel, HaMerkaz
Israel, HaZafon
Israel, Hefa
Israel, Tel Aviv
Israel, Yerushalayim
Italy, Abruzzi
Italy, Basilicata
Italy, Calabria
Italy, Campania
Italy, Emilia-Romagna
Italy, Friuli-Venezia Giulia
Italy, Lazio
Italy, Liguria
Italy, Lombardia
Italy, Marche
Italy, Molise
Italy, Piemonte
Italy, Puglia
Italy, Sardegna
Italy, Sicilia
Italy, Toscana
Italy, Trentino-Alto Adige
Italy, Umbria
Italy, Valle d'Aosta
Italy, Veneto
Jamaica, Clarendon
Jamaica, Hanover
Jamaica, Kingston
Jamaica, Manchester
Jamaica, Portland
Jamaica, Saint Andrew
Jamaica, Saint Ann
Jamaica, Saint Catherine
Jamaica, Saint Elizabeth
Jamaica, Saint James
Jamaica, Saint Mary
Jamaica, Saint Thomas
Jamaica, Trelawny
Jamaica, Westmoreland

Japan, Aichi
Japan, Akita
Japan, Aomori
Japan, Chiba
Japan, Ehime
Japan, Fukui
Japan, Fukuoka
Japan, Fukushima
Japan, Gifu
Japan, Gumma
Japan, Hiroshima
Japan, Hokkaido
Japan, Hyogo
Japan, Ibaraki
Japan, Ishikawa
Japan, Iwate
Japan, Kagawa
Japan, Kagoshima
Japan, Kanagawa
Japan, Kochi
Japan, Kumamoto
Japan, Kyoto
Japan, Mie
Japan, Miyagi
Japan, Miyazaki
Japan, Nagano
Japan, Nagasaki
Japan, Nara
Japan, Niigata
Japan, Oita
Japan, Okayama
Japan, Okinawa
Japan, Osaka
Japan, Saga
Japan, Saitama
Japan, Shiga
Japan, Shimane
Japan, Shizuoka
Japan, Tochigi
Japan, Tokushima
Japan, Tokyo
Japan, Tottori
Japan, Toyama
Japan, Wakayama
Japan, Yamagata
Japan, Yamaguchi
Japan, Yamanashi
Jordan, Al Balqa'
Jordan, Al Karak
Jordan, Al Mafraq
Jordan, Amman
Jordan, Amman Governorate
Jordan, At Tafilah
Jordan, Az Zarqa
Jordan, Irbid
Jordan, Ma
Kazakhstan, Almaty
Kazakhstan, Almaty City
Kazakhstan, Aqmola
Kazakhstan, Aqtobe
Kazakhstan, Astana
Kazakhstan, Atyrau
Kazakhstan, Bayqonyr
Kazakhstan, East Kazakhstan
Kazakhstan, Mangghystau
Kazakhstan, North Kazakhstan
Kazakhstan, Pavlodar
Kazakhstan, Qaraghandy
Kazakhstan, Qostanay
Kazakhstan, Qyzylorda
Kazakhstan, South Kazakhstan
Kazakhstan, West

Kazakhstan
Kazakhstan, Zhambyl
Kenya, Central
Kenya, Coast
Kenya, Eastern
Kenya, Nairobi Area
Kenya, North-Eastern
Kenya, Nyanza
Kenya, Rift Valley
Kenya, Western
Kiribati, Gilbert Islands
Kiribati, Line Islands
Kiribati, Phoenix Islands
Korea, Democratic People's Republic of, Chagang-do
Korea, Democratic People's Republic of, Hamgyong-bukto
Korea, Democratic People's Republic of, Hamgyong-namdo
Korea, Democratic People's Republic of, Hwanghae-bukto
Korea, Democratic People's Republic of, Hwanghae-namdo
Korea, Democratic People's Republic of, Kaesong-si
Korea, Democratic People's Republic of, Kangwon-do
Korea, Democratic People's Republic of, Najin Sonbong-si
Korea, Democratic People's Republic of, Namp'o-si
Korea, Democratic People's Republic of, P'yongan-bukto
Korea, Democratic People's Republic of, P'yongan-namdo
Korea, Democratic People's Republic of, P'yongyang-si
Korea, Democratic People's Republic of, Yanggang-do
Korea, Republic of, Ch'ungch'ong-bukto
Korea, Republic of, Ch'ungch'ong-namdo
Korea, Republic of, Cheju-do
Korea, Republic of, Cholla-bukto
Korea, Republic of, Cholla-namdo
Korea, Republic of, Inch'on-jikhalsi
Korea, Republic of, Kangwon-do
Korea, Republic of, Kwangju-jikhalsi
Korea, Republic of, Kyonggi-do
Korea, Republic of, Kyongsang-bukto
Korea, Republic of, Kyongsang-namdo
Korea, Republic of, Pusan-jikhalsi
Korea, Republic of, Se oul Teukbyeolsi

Korea, Republic of, Seoul-t'ukpyolsi
Korea, Republic of, Taegu-jikhalsi
Korea, Republic of, Taejon-jikhalsi
Korea, Republic of, Ulsan-gwangyoksi
Kuwait, Al Ahmadi
Kuwait, Al Farwaniyah
Kuwait, Al Jahra
Kuwait, Al Kuwayt
Kuwait, Hawalli
Kuwait, Mubarak al Kabir
Kyrgyzstan, Batken
Kyrgyzstan, Bishkek
Kyrgyzstan, Chuy
Kyrgyzstan, Jalal-Abad
Kyrgyzstan, Naryn
Kyrgyzstan, Osh
Kyrgyzstan, Osh
Kyrgyzstan, Talas
Kyrgyzstan, Ysyk-Kol
Lao People's Democratic Republic, Attapu
Lao People's Democratic Republic, Champasak
Lao People's Democratic Republic, Houaphan
Lao People's Democratic Republic, Khammouan
Lao People's Democratic Republic, Louang Namtha
Lao People's Democratic Republic, Louangphrabang
Lao People's Democratic Republic, Oudomxai
Lao People's Democratic Republic, Phongsali
Lao People's Democratic Republic, Saravan
Lao People's Democratic Republic, Savannakhet
Lao People's Democratic Republic, Vientiane
Lao People's Democratic Republic, Xaignabouri
Lao People's Democratic Republic, Xiangkhoang
Latvia, Aizkraukles
Latvia, Aluksnes
Latvia, Balvu
Latvia, Bauskas
Latvia, Cesu
Latvia, Daugavpils
Latvia, Daugavpils
Latvia, Dobeles
Latvia, Gulbenes
Latvia, Jekabpils
Latvia, Jelgava
Latvia, Jelgavas
Latvia, Jurmala
Latvia, Kraslavas
Latvia, Kuldigas
Latvia, Liepaja

Latvia, Liepajas
Latvia, Limbazu
Latvia, Ludzas
Latvia, Madonas
Latvia, Ogres
Latvia, Preilu
Latvia, Rezekne
Latvia, Rezeknes
Latvia, Riga
Latvia, Rigas
Latvia, Saldus
Latvia, Talsu
Latvia, Tukuma
Latvia, Valkas
Latvia, Valmieras
Latvia, Ventspils
Latvia, Ventspils
Lebanon, Aakk
Lebanon, Al Janub
Lebanon, Baalbek-Hermel
Lebanon, Beqaa
Lebanon, Beqaa
Lebanon, Beyrouth
Lebanon, Liban-Nord
Lebanon, Liban-Nord
Lebanon, Liban-Sud
Lebanon, Mont-Liban
Lebanon, Nabatiye
Lesotho, Berea
Lesotho, Butha-Buthe
Lesotho, Leribe
Lesotho, Mafeteng
Lesotho, Maseru
Lesotho, Mohales Hoek
Lesotho, Mokhotlong
Lesotho, Qachas Nek
Lesotho, Quthing
Lesotho, Thaba-Tseka
Liberia, Bong
Liberia, Gbarpolu
Liberia, Grand Bassa
Liberia, Grand Cape Mount
Liberia, Grand Cape Mount
Liberia, Grand Gedeh
Liberia, Lofa
Liberia, Lofa
Liberia, Margibi
Liberia, Maryland
Liberia, Maryland
Liberia, Monrovia
Liberia, Montserrado
Liberia, Nimba
Liberia, River Cess
Liberia, River Gee
Liberia, Sino
Libyan Arab Jamahiriya, Ajdabiya
Libyan Arab Jamahiriya, Al Aziziyah
Libyan Arab Jamahiriya, Al Fatih
Libyan Arab Jamahiriya, Al Jabal al Akhdar
Libyan Arab Jamahiriya, Al Jufrah
Libyan Arab Jamahiriya, Al Khums
Libyan Arab Jamahiriya, Al Kufrah
Libyan Arab Jamahiriya, An Nuqat al Khams
Libyan Arab Jamahiriya, Ash Shati'

Libyan Arab
Jamahiriya, Awbari
Libyan Arab
Jamahiriya, Az
Zawiyah
Libyan Arab
Jamahiriya, Banghazi
Libyan Arab
Jamahiriya, Darnah
Libyan Arab
Jamahiriya, Ghadamis
Libyan Arab
Jamahiriya, Gharyan
Libyan Arab
Jamahiriya, Misratah
Libyan Arab
Jamahiriya, Murzuq
Libyan Arab
Jamahiriya, Sabha
Libyan Arab
Jamahiriya, Sawfajjin
Libyan Arab
Jamahiriya, Surt
Libyan Arab
Jamahiriya, Tarabulus
Libyan Arab
Jamahiriya, Tarhunah
Libyan Arab
Jamahiriya, Tubruq
Libyan Arab
Jamahiriya, Yafran
Libyan Arab
Jamahiriya, Zlitan
Liechtenstein, Balzers
Liechtenstein, Eschen
Liechtenstein, Gamprin
Liechtenstein,
Gbarpolu
Liechtenstein, Mauren
Liechtenstein, Planken
Liechtenstein, River
Gee
Liechtenstein, Ruggell
Liechtenstein, Schaan
Liechtenstein,
Schellenberg
Liechtenstein, Triesen
Liechtenstein,
Triesenberg
Liechtenstein, Vaduz
Lithuania, Alytaus
Apskritis
Lithuania, Kauno
Apskritis
Lithuania, Klaipedos
Apskritis
Lithuania,
Marijampoles Apskritis
Lithuania, Panevezio
Apskritis
Lithuania, Siauliu
Apskritis
Lithuania, Taurages
Apskritis
Lithuania, Telsiu
Apskritis
Lithuania, Utenos
Apskritis
Lithuania, Vilniaus
Apskritis
Luxembourg, Diekirch
Luxembourg,
Grevenmacher
Luxembourg,
Luxembourg
Macau, Ilhas
Macau, Macau
Macedonia, Aracinovo
Macedonia, Bac

Macedonia, Belcista
Macedonia, Berovo
Macedonia, Bistrica
Macedonia, Bitola
Macedonia, Blatec
Macedonia, Bogdanci
Macedonia, Bogomila
Macedonia, Bogovinje
Macedonia, Bosilovo
Macedonia, Brvenica
Macedonia, Cair
Macedonia, Capari
Macedonia, Caska
Macedonia, Cegrane
Macedonia, Centar
Macedonia, Centar Zupa
Macedonia, Cesinovo
Macedonia, Cucer-Sandevo
Macedonia, Debar
Macedonia, Delcevo
Macedonia, Delogozdi
Macedonia, Demir Hisar
Macedonia, Demir Kapija
Macedonia, Dobrusevo
Macedonia, Dolna Banjica
Macedonia, Dolneni
Macedonia, Dorce Petrov
Macedonia, Drugovo
Macedonia, Dzepciste
Macedonia, Gazi Baba
Macedonia, Gevgelija
Macedonia, Gostivar
Macedonia, Gradsko
Macedonia, Ilinden
Macedonia, Izvor
Macedonia, Jegunovce
Macedonia, Kamenjane
Macedonia, Karbinci
Macedonia, Karpos
Macedonia, Kavadarci
Macedonia, Kicevo
Macedonia, Kisela Voda
Macedonia, Klecevce
Macedonia, Kocani
Macedonia, Konce
Macedonia, Kondovo
Macedonia, Konopiste
Macedonia, Kosel
Macedonia, Kratovo
Macedonia, Kriva Palanka
Macedonia, Krivogastani
Macedonia, Krusevo
Macedonia, Kuklis
Macedonia, Kukurecani
Macedonia, Kumanovo
Macedonia, Labunista
Macedonia, Lipkovo
Macedonia, Lozovo
Macedonia, Lukovo
Macedonia, Makedonska Kamenica
Macedonia, Makedonski Brod
Macedonia, Mavrovi Anovi
Macedonia, Meseista
Macedonia, Miravci

Macedonia, Mogila
Macedonia, Murtino
Macedonia, Negotino
Macedonia, Negotino-Polosko
Macedonia, Novaci
Macedonia, Novo Selo
Macedonia, Oblesevo
Macedonia, Ohrid
Macedonia, Orasac
Macedonia, Orizari
Macedonia, Oslomej
Macedonia, Pehcevo
Macedonia, Petrovec
Macedonia, Plasnica
Macedonia, Podares
Macedonia, Prilep
Macedonia, Probistip
Macedonia, Radovis
Macedonia, Rankovce
Macedonia, Resen
Macedonia, Rosoman
Macedonia, Rostusa
Macedonia, Samokov
Macedonia, Saraj
Macedonia, Sipkovica
Macedonia, Sopiste
Macedonia, Sopotnica
Macedonia, Srbinovo
Macedonia, Star Dojran
Macedonia, Staravina
Macedonia, Staro Nagoricane
Macedonia, Stip
Macedonia, Struga
Macedonia, Strumica
Macedonia, Studenicani
Macedonia, Suto Orizari
Macedonia, Sveti Nikole
Macedonia, Tearce
Macedonia, Tetovo
Macedonia, Topolcani
Macedonia, Valandovo
Macedonia, Vasilevo
Macedonia, Veles
Macedonia, Velesta
Macedonia, Vevcani
Macedonia, Vinica
Macedonia, Vitoliste
Macedonia, Vranestica
Macedonia, Vrapciste
Macedonia, Vratnica
Macedonia, Vrutok
Macedonia, Zajas
Macedonia, Zelenikovo
Macedonia, Zelino
Macedonia, Zitose
Macedonia, Zletovo
Macedonia, Zrnovci
Madagascar, Antananarivo
Madagascar, Antsiranana
Madagascar, Fianarantsoa
Madagascar, Mahajanga
Madagascar, Toamasina
Madagascar, Toliara
Malawi, Balaka
Malawi, Blantyre
Malawi, Chikwawa
Malawi, Chiradzulu
Malawi, Chitipa

Malawi, Dedza
Malawi, Dowa
Malawi, Karonga
Malawi, Kasungu
Malawi, Likoma
Malawi, Lilongwe
Malawi, Machinga
Malawi, Mangochi
Malawi, Mchinji
Malawi, Mulanje
Malawi, Mwanza
Malawi, Mzimba
Malawi, Nkhata Bay
Malawi, Nkhotakota
Malawi, Nsanje
Malawi, Ntcheu
Malawi, Ntchisi
Malawi, Phalombe
Malawi, Rumphi
Malawi, Salima
Malawi, Thyolo
Malawi, Zomba
Malaysia, Johor
Malaysia, Kedah
Malaysia, Kelantan
Malaysia, Kuala
Lumpur
Malaysia, Labuan
Malaysia, Melaka
Malaysia, Negeri
Sembilan
Malaysia, Pahang
Malaysia, Perak
Malaysia, Perlis
Malaysia, Pulau
Pinang
Malaysia, Putrajaya
Malaysia, Sabah
Malaysia, Sarawak
Malaysia, Selangor
Malaysia, Terengganu
Maldives, Alifu
Maldives, Baa
Maldives, Dhaalu
Maldives, Faafu
Maldives, Gaafu Alifu
Maldives, Gaafu
Dhaalu
Maldives, Gnaviyani
Maldives, Haa Alifu
Maldives, Haa Dhaalu
Maldives, Kaafu
Maldives, Laamu
Maldives, Lhaviyani
Maldives, Maale
Maldives, Meemu
Maldives, Noonu
Maldives, Raa
Maldives, Seenu
Maldives, Shaviyani
Maldives, Thaa
Maldives, Vaavu
Mali, Bamako
Mali, Gao
Mali, Kayes
Mali, Kidal
Mali, Koulikoro
Mali, Mopti
Mali, Segou
Mali, Sikasso
Mali, Tombouctou
Mauritania, Adrar
Mauritania, Assaba
Mauritania, Brakna
Mauritania, Dakhlet
Nouadhibou
Mauritania, Gorgol
Mauritania, Guidimaka
Mauritania, Hodh Ech

Chargui
Mauritania, Hodh El Gharbi
Mauritania, Inchiri
Mauritania, Tagant
Mauritania, Tiris Zemmour
Mauritania, Trarza
Mauritius, Agalega Islands
Mauritius, Black River
Mauritius, Cargados Carajos
Mauritius, Flacq
Mauritius, Grand Port
Mauritius, Moka
Mauritius, Pamplemousses
Mauritius, Plaines Wilhems
Mauritius, Port Louis
Mauritius, Riviere du Rempart
Mauritius, Rodrigues
Mauritius, Savanne
Mexico, Aguascalientes
Mexico, Baja California
Mexico, Baja California Sur
Mexico, Campeche
Mexico, Chiapas
Mexico, Chihuahua
Mexico, Coahuila de Zaragoza
Mexico, Colima
Mexico, Distrito Federal
Mexico, Durango
Mexico, Guanajuato
Mexico, Guerrero
Mexico, Hidalgo
Mexico, Jalisco
Mexico, Mexico
Mexico, Michoacan de Ocampo
Mexico, Morelos
Mexico, Nayarit
Mexico, Nuevo Leon
Mexico, Oaxaca
Mexico, Puebla
Mexico, Queretaro de Arteaga
Mexico, Quintana Roo
Mexico, San Luis Potosi
Mexico, Sinaloa
Mexico, Sonora
Mexico, Tabasco
Mexico, Tamaulipas
Mexico, Tlaxcala
Mexico, Veracruz-Llave
Mexico, Yucatan
Mexico, Zacatecas
Micronesia, Chuuk
Micronesia, Kosrae
Micronesia, Pohnpei
Micronesia, Yap
Moldova, Republic of, Anenii Noi
Moldova, Republic of, Balti
Moldova, Republic of, Basarabeasca
Moldova, Republic of, Bender
Moldova, Republic of, Briceni

Moldova, Republic of, Cahul
Moldova, Republic of, Calarasi
Moldova, Republic of, Cantemir
Moldova, Republic of, Causeni
Moldova, Republic of, Chisinau
Moldova, Republic of, Cimislia
Moldova, Republic of, Criuleni
Moldova, Republic of, Donduseni
Moldova, Republic of, Drochia
Moldova, Republic of, Dubasari
Moldova, Republic of, Edinet
Moldova, Republic of, Falesti
Moldova, Republic of, Floresti
Moldova, Republic of, Gagauzia
Moldova, Republic of, Glodeni
Moldova, Republic of, Hincesti
Moldova, Republic of, Ialoveni
Moldova, Republic of, Leova
Moldova, Republic of, Nisporeni
Moldova, Republic of, Ocnita
Moldova, Republic of, Rezina
Moldova, Republic of, Riscani
Moldova, Republic of, Singerei
Moldova, Republic of, Soldanesti
Moldova, Republic of, Soroca
Moldova, Republic of, Stefan-Voda
Moldova, Republic of, Stinga Nistrului
Moldova, Republic of, Straseni
Moldova, Republic of, Taraclia
Moldova, Republic of, Telenesti
Moldova, Republic of, Ungheni
Monaco, La Condamine
Monaco, Monaco
Monaco, Monte-Carlo
Mongolia, Arhangay
Mongolia, Bayan-Olgiy
Mongolia, Bayanhongor
Mongolia, Bulgan
Mongolia, Darhan
Mongolia, Darhan-Uul
Mongolia, Dornod
Mongolia, Dornogovi
Mongolia, Dundgovi
Mongolia, Dzavhan
Mongolia, Erdenet
Mongolia, Govi-Altay

Mongolia, Govisumber
Mongolia, Hentiy
Mongolia, Hovd
Mongolia, Hovsgol
Mongolia, Omnogovi
Mongolia, Orhon
Mongolia, Ovorhangay
Mongolia, Selenge
Mongolia, Suhbaatar
Mongolia, Tov
Mongolia, Ulaanbaatar
Mongolia, Uvs
Montserrat, Saint Anthony
Montserrat, Saint Georges
Montserrat, Saint Peter
Morocco, Chaouia-Ouardigha
Morocco, Doukkala-Abda
Morocco, Fes-Boulemane
Morocco, Gharb-Chrarda-Beni Hssen
Morocco, Grand Casablanca
Morocco, Guelmim-Es Smara
Morocco, La
Morocco, Marrakech-Tensift-Al Haouz
Morocco, Meknes-Tafilalet
Morocco, Oriental
Morocco, Rabat-Sale-Zemmour-Zaer
Morocco, Souss-Massa-Dr
Morocco, Tadla-Azilal
Morocco, Tanger-Tetouan
Morocco, Taza-Al Hoceima-Taounate
Mozambique, Cabo Delgado
Mozambique, Gaza
Mozambique, Inhambane
Mozambique, Manica
Mozambique, Maputo
Mozambique, Maputo
Mozambique, Nampula
Mozambique, Niassa
Mozambique, Sofala
Mozambique, Tete
Mozambique, Zambezia
Myanmar, Chin State
Myanmar, Irrawaddy
Myanmar, Kachin State
Myanmar, Karan State
Myanmar, Kayah State
Myanmar, Magwe
Myanmar, Mandalay
Myanmar, Mon State
Myanmar, Pegu
Myanmar, Rakhine State
Myanmar, Rangoon
Myanmar, Sagaing
Myanmar, Shan State
Myanmar, Tenasserim
Myanmar, Yangon

N – S

Namibia, Bethanien
Namibia,

Boesmanland
Namibia, Caprivi
Namibia, Caprivi Oos
Namibia, Damaraland
Namibia, Erongo
Namibia, Gobabis
Namibia, Grootfontein
Namibia, Hardap
Namibia, Hereroland
Oos
Namibia, Hereroland
Wes
Namibia, Kaokoland
Namibia, Karas
Namibia, Karasburg
Namibia, Karibib
Namibia, Kavango
Namibia,
Keetmanshoop
Namibia, Kunene
Namibia, Luderitz
Namibia, Maltahohe
Namibia, Mariental
Namibia, Namaland
Namibia, Ohangwena
Namibia, Okahandja
Namibia, Okavango
Namibia, Omaheke
Namibia, Omaruru
Namibia, Omusati
Namibia, Oshana
Namibia, Oshikoto
Namibia, Otjiwarongo
Namibia, Otjozondjupa
Namibia, Outjo
Namibia, Owambo
Namibia, Rehoboth
Namibia, Swakopmund
Namibia, Tsumeb
Namibia, Windhoek
Nauru, Aiwo
Nauru, Anabar
Nauru, Anetan
Nauru, Anibare
Nauru, Baiti
Nauru, Boe
Nauru, Buada
Nauru, Denigomodu
Nauru, Ewa
Nauru, Ijuw
Nauru, Meneng
Nauru, Nibok
Nauru, Uaboe
Nauru, Yaren
Nepal, Bagmati
Nepal, Bheri
Nepal, Dhawalagiri
Nepal, Gandaki
Nepal, Janakpur
Nepal, Karnali
Nepal, Kosi
Nepal, Lumbini
Nepal, Mahakali
Nepal, Mechi
Nepal, Narayani
Nepal, Rapti
Nepal, Sagarmatha
Nepal, Seti
Netherlands, Drenthe
Netherlands, Flevoland
Netherlands, Friesland
Netherlands,
Gelderland
Netherlands,
Groningen
Netherlands, Limburg
Netherlands, Noord-
Brabant
Netherlands, Noord-

Holland
Netherlands, Overijssel
Netherlands, Overijssel
Netherlands, Utrecht
Netherlands, Zeeland
Netherlands, Zuid-Holland
New Zealand, Auckland
New Zealand, Bay of Plenty
New Zealand, Canterbury
New Zealand, Chatham Islands
New Zealand, Gisborne
New Zealand, Hawke's Bay
New Zealand, Manawatu-Wanganui
New Zealand, Marlborough
New Zealand, Nelson
New Zealand, Northland
New Zealand, Otago
New Zealand, Southland
New Zealand, Taranaki
New Zealand, Waikato
New Zealand, Wellington
New Zealand, West Coast
Nicaragua, Autonoma Atlantico Norte
Nicaragua, Boaco
Nicaragua, Carazo
Nicaragua, Chinandega
Nicaragua, Chontales
Nicaragua, Esteli
Nicaragua, Granada
Nicaragua, Jinotega
Nicaragua, Leon
Nicaragua, Madriz
Nicaragua, Managua
Nicaragua, Masaya
Nicaragua, Matagalpa
Nicaragua, Nueva Segovia
Nicaragua, Region Autonoma Atlantico Sur
Nicaragua, Rio San Juan
Nicaragua, Rivas
Nicaragua, Zelaya
Niger, Agadez
Niger, Diffa
Niger, Dosso
Niger, Maradi
Niger, Niamey
Niger, Niamey
Niger, Tahoua
Niger, Zinder
Nigeria, Abia
Nigeria, Adamawa
Nigeria, Akwa Ibom
Nigeria, Anambra
Nigeria, Bauchi
Nigeria, Bayelsa
Nigeria, Benue
Nigeria, Borno
Nigeria, Cross River

Nigeria, Delta
Nigeria, Ebonyi
Nigeria, Edo
Nigeria, Ekiti
Nigeria, Enugu
Nigeria, Federal Capital Territory
Nigeria, Gombe
Nigeria, Imo
Nigeria, Jigawa
Nigeria, Kaduna
Nigeria, Kano
Nigeria, Katsina
Nigeria, Kebbi
Nigeria, Kogi
Nigeria, Kwara
Nigeria, Lagos
Nigeria, Nassarawa
Nigeria, Niger
Nigeria, Ogun
Nigeria, Ondo
Nigeria, Osun
Nigeria, Oyo
Nigeria, Plateau
Nigeria, Rivers
Nigeria, Sokoto
Nigeria, Taraba
Nigeria, Yobe
Nigeria, Zamfara
Norway, Akershus
Norway, Aust-Agder
Norway, Buskerud
Norway, Finnmark
Norway, Hedmark
Norway, Hordaland
Norway, More og Romsdal
Norway, Nord-Trondelag
Norway, Nordland
Norway, Oppland
Norway, Oslo
Norway, Ostfold
Norway, Rogaland
Norway, Sogn og Fjordane
Norway, Sor-Trondelag
Norway, Telemark
Norway, Troms
Norway, Vest-Agder
Norway, Vestfold
Oman, Ad Dakhiliyah
Oman, Al Batinah
Oman, Al Wusta
Oman, Ash Sharqiyah
Oman, Az Zahirah
Oman, Masqat
Oman, Musandam
Oman, Zufar
Pakistan, Azad Kashmir
Pakistan, Balochistan
Pakistan, Federally Administered Tribal Areas
Pakistan, Islamabad
Pakistan, North-West Frontier
Pakistan, Northern Areas
Pakistan, Punjab
Pakistan, Sindh
Palestinian Territory, Occupied, Gaza
Palestinian Territory, Occupied, West Bank
Panama, Bocas del Toro

Panama, Chiriqui
Panama, Cocle
Panama, Colon
Panama, Darien
Panama, Herrera
Panama, Los Santos
Panama, Panama
Panama, San Blas
Panama, Veraguas
Papua New Guinea, Central
Papua New Guinea, Chimbu
Papua New Guinea, East New Britain
Papua New Guinea, East Sepik
Papua New Guinea, Eastern Highlands
Papua New Guinea, Enga
Papua New Guinea, Gulf
Papua New Guinea, Madang
Papua New Guinea, Manus
Papua New Guinea, Milne Bay
Papua New Guinea, Morobe
Papua New Guinea, National Capital
Papua New Guinea, New Ireland
Papua New Guinea, North Solomons
Papua New Guinea, Northern
Papua New Guinea, Sandaun
Papua New Guinea, Southern Highlands
Papua New Guinea, West New Britain
Papua New Guinea, Western
Papua New Guinea, Western Highlands
Paraguay, Alto Paraguay
Paraguay, Alto Parana
Paraguay, Amambay
Paraguay, Boqueron
Paraguay, Caaguazu
Paraguay, Caazapa
Paraguay, Canindeyu
Paraguay, Central
Paraguay, Chaco
Paraguay, Concepcion
Paraguay, Cordillera
Paraguay, Guaira
Paraguay, Itapua
Paraguay, Misiones
Paraguay, Neembucu
Paraguay, Nueva Asuncion
Paraguay, Paraguari
Paraguay, Presidente Hayes
Paraguay, San Pedro
Peru, Amazonas
Peru, Ancash
Peru, Apurimac
Peru, Arequipa
Peru, Ayacucho
Peru, Cajamarca
Peru, Callao
Peru, Cusco

Peru, Huancavelica
Peru, Huanuco
Peru, Ica
Peru, Junin
Peru, La Libertad
Peru, Lambayeque
Peru, Lima
Peru, Loreto
Peru, Madre de Dios
Peru, Moquegua
Peru, Pasco
Peru, Piura
Peru, Puno
Peru, San Martin
Peru, Tacna
Peru, Tumbes
Peru, Ucayali
Philippines, Abra
Philippines, Agusan del Norte
Philippines, Agusan del Sur
Philippines, Aklan
Philippines, Albay
Philippines, Angeles
Philippines, Antique
Philippines, Aurora
Philippines, Bacolod
Philippines, Bago
Philippines, Baguio
Philippines, Bais
Philippines, Basilan
Philippines, Basilan City
Philippines, Bataan
Philippines, Batanes
Philippines, Batangas
Philippines, Batangas City
Philippines, Benguet
Philippines, Bohol
Philippines, Bukidnon
Philippines, Bulacan
Philippines, Butuan
Philippines, Cabanatuan
Philippines, Cadiz
Philippines, Cagayan
Philippines, Cagayan de Oro
Philippines, Calbayog
Philippines, Caloocan
Philippines, Camarines Norte
Philippines, Camarines Sur
Philippines, Camiguin
Philippines, Canlaon
Philippines, Capiz
Philippines, Catanduanes
Philippines, Cavite
Philippines, Cavite City
Philippines, Cebu
Philippines, Cebu City
Philippines, Cotabato
Philippines, Dagupan
Philippines, Danao
Philippines, Dapitan
Philippines, Davao
Philippines, Davao City
Philippines, Davao del Sur
Philippines, Davao Oriental
Philippines, Dipolog
Philippines, Dumaguete
Philippines, Eastern

Samar
Philippines, General Santos
Philippines, Gingoog
Philippines, Ifugao
Philippines, Iligan
Philippines, Ilocos Norte
Philippines, Ilocos Sur
Philippines, Iloilo
Philippines, Iloilo City
Philippines, Iriga
Philippines, Isabela
Philippines, Kalinga-Apayao
Philippines, La Carlota
Philippines, La Union
Philippines, Laguna
Philippines, Lanao del Norte
Philippines, Lanao del Sur
Philippines, Laoag
Philippines, Lapu-Lapu
Philippines, Legaspi
Philippines, Leyte
Philippines, Lipa
Philippines, Lucena
Philippines, Maguindanao
Philippines, Mandaue
Philippines, Manila
Philippines, Marawi
Philippines, Marinduque
Philippines, Masbate
Philippines, Mindoro Occidental
Philippines, Mindoro Oriental
Philippines, Misamis Occidental
Philippines, Misamis Oriental
Philippines, Mountain
Philippines, Naga
Philippines, Negros Occidental
Philippines, Negros Occidental
Philippines, Negros Oriental
Philippines, North Cotabato
Philippines, Northern Samar
Philippines, Nueva Ecija
Philippines, Nueva Vizcaya
Philippines, Olongapo
Philippines, Ormoc
Philippines, Oroquieta
Philippines, Ozamis
Philippines, Pagadian
Philippines, Palawan
Philippines, Palayan
Philippines, Pampanga
Philippines, Pangasinan
Philippines, Pasay
Philippines, Puerto Princesa
Philippines, Quezon
Philippines, Quezon City
Philippines, Quirino
Philippines, Rizal
Philippines, Romblon

Philippines, Roxas
Philippines, Samar
Philippines, San Carlos
Philippines, San Carlos
Philippines, San Jose
Philippines, San Pablo
Philippines, Silay
Philippines, Siquijor
Philippines, Sorsogon
Philippines, South Cotabato
Philippines, Southern Leyte
Philippines, Sultan Kudarat
Philippines, Sulu
Philippines, Surigao
Philippines, Surigao del Norte
Philippines, Surigao del Sur
Philippines, Tacloban
Philippines, Tagaytay
Philippines, Tagbilaran
Philippines, Tangub
Philippines, Tarlac
Philippines, Tawitawi
Philippines, Toledo
Philippines, Trece Martires
Philippines, Zambales
Philippines, Zamboanga
Philippines, Zamboanga del Norte
Philippines, Zamboanga del Sur
Poland, Dolnoslaskie
Poland, Kujawsko-Pomorskie
Poland, Lodzkie
Poland, Lubelskie
Poland, Lubuskie
Poland, Malopolskie
Poland, Mazowieckie
Poland, Opolskie
Poland, Podkarpackie
Poland, Podlaskie
Poland, Pomorskie
Poland, Slaskie
Poland, Swietokrzyskie
Poland, Warminsko-Mazurskie
Poland, Wielkopolskie
Poland, Zachodniopomorskie
Portugal, Aveiro
Portugal, Azores
Portugal, Beja
Portugal, Braga
Portugal, Braganca
Portugal, Castelo Branco
Portugal, Coimbra
Portugal, Evora
Portugal, Faro
Portugal, Guarda
Portugal, Leiria
Portugal, Lisboa
Portugal, Madeira
Portugal, Portalegre
Portugal, Porto
Portugal, Santarem
Portugal, Setubal
Portugal, Viana do Castelo
Portugal, Vila Real

Portugal, Viseu
Qatar, Ad Dawhah
Qatar, Al Ghuwariyah
Qatar, Al Jumaliyah
Qatar, Al Khawr
Qatar, Al Wakrah
Qatar, Al Wakrah Municipality
Qatar, Ar Rayyan
Qatar, Jariyan al Batnah
Qatar, Madinat ach Shamal
Qatar, Umm Sa'id
Qatar, Umm Salal
Romania, Alba
Romania, Arad
Romania, Arges
Romania, Bacau
Romania, Bihor
Romania, Bistrita-Nasaud
Romania, Botosani
Romania, Braila
Romania, Brasov
Romania, Bucuresti
Romania, Buzau
Romania, Calarasi
Romania, Caras-Severin
Romania, Cluj
Romania, Constanta
Romania, Covasna
Romania, Dambovita
Romania, Dolj
Romania, Galati
Romania, Giurgiu
Romania, Gorj
Romania, Harghita
Romania, Hunedoara
Romania, Ialomita
Romania, Iasi
Romania, Ilfov
Romania, Maramures
Romania, Mehedinti
Romania, Mures
Romania, Neamt
Romania, Olt
Romania, Prahova
Romania, Salaj
Romania, Satu Mare
Romania, Sibiu
Romania, Suceava
Romania, Teleorman
Romania, Timis
Romania, Tulcea
Romania, Valcea
Romania, Vaslui
Romania, Vrancea
Russian Federation, Adygeya
Russian Federation, Aginsky Buryatsky AO
Russian Federation, Altaisky krai
Russian Federation, Amur
Russian Federation, Arkhangel'sk
Russian Federation, Astrakhan'
Russian Federation, Bashkortostan
Russian Federation, Belgorod
Russian Federation, Bryansk
Russian Federation, Buryat

Russian Federation,
Chechnya
Russian Federation,
Chechnya Republic
Russian Federation,
Chelyabinsk
Russian Federation,
Chita
Russian Federation,
Chukot
Russian Federation,
Chuvashia
Russian Federation,
Dagestan
Russian Federation,
Evenk
Russian Federation,
Gorno-Altay
Russian Federation,
Ingush
Russian Federation,
Irkutsk
Russian Federation,
Ivanovo
Russian Federation,
Kabardin-Balkar
Russian Federation,
Kaliningrad
Russian Federation,
Kalmyk
Russian Federation,
Kaluga
Russian Federation,
Kamchatka
Russian Federation,
Karachay-Cherkess
Russian Federation,
Karelia
Russian Federation,
Kemerovo
Russian Federation,
Khabarovsk
Russian Federation,
Khakass
Russian Federation,
Khanty-Mansiy
Russian Federation,
Kirov
Russian Federation,
Komi
Russian Federation,
Komi-Permyak
Russian Federation,
Koryak
Russian Federation,
Kostroma
Russian Federation,
Krasnodar
Russian Federation,
Krasnoyarsk
Russian Federation,
Krasnoyarskiy Kray
Russian Federation,
Kurgan
Russian Federation,
Kursk
Russian Federation,
Leningrad
Russian Federation,
Lipetsk
Russian Federation,
Magadan
Russian Federation,
Mariy-El
Russian Federation,
Mordovia
Russian Federation,
Moscow City
Russian Federation,

Moskva
Russian Federation,
Murmansk
Russian Federation,
Nenets
Russian Federation,
Nizhegorod
Russian Federation,
North Ossetia
Russian Federation,
Novgorod
Russian Federation,
Novosibirsk
Russian Federation,
Omsk
Russian Federation,
Orel
Russian Federation,
Orenburg
Russian Federation,
Penza
Russian Federation,
Perm'
Russian Federation,
Permskiy Kray
Russian Federation,
Primor'ye
Russian Federation,
Pskov
Russian Federation,
Rostov
Russian Federation,
Ryazan'
Russian Federation,
Saint Petersburg City
Russian Federation,
Sakha
Russian Federation,
Sakhalin
Russian Federation,
Samara
Russian Federation,
Saratov
Russian Federation,
Smolensk
Russian Federation,
Stavropol'
Russian Federation,
Sverdlovsk
Russian Federation,
Tambovskaya oblast
Russian Federation,
Tatarstan
Russian Federation,
Taymyr
Russian Federation,
Tomsk
Russian Federation,
Tula
Russian Federation,
Tuva
Russian Federation,
Tver'
Russian Federation,
Tyumen'
Russian Federation,
Udmurt
Russian Federation,
Ul'yanovsk
Russian Federation,
Ust-Orda Buryat
Russian Federation,
Vladimir
Russian Federation,
Volgograd
Russian Federation,
Vologda
Russian Federation,
Voronezh

Russian Federation, Yamal-Nenets
Russian Federation, Yaroslavl'
Russian Federation, Yevrey
Rwanda, Butare
Rwanda, Est
Rwanda, Gitarama
Rwanda, Kibungo
Rwanda, Kigali
Rwanda, Kigali
Rwanda, Nord
Rwanda, Ouest
Rwanda, Sud
Saint Helena, Ascension
Saint Helena, Saint Helena
Saint Helena, Tristan da Cunha
Saint Kitts and Nevis, Christ Church Nichola Town
Saint Kitts and Nevis, Saint Anne Sandy Point
Saint Kitts and Nevis, Saint George Basseterre
Saint Kitts and Nevis, Saint George Gingerland
Saint Kitts and Nevis, Saint James Windward
Saint Kitts and Nevis, Saint John Capisterre
Saint Kitts and Nevis, Saint John Figtree
Saint Kitts and Nevis, Saint Mary Cayon
Saint Kitts and Nevis, Saint Paul Capisterre
Saint Kitts and Nevis, Saint Paul Charlestown
Saint Kitts and Nevis, Saint Peter Basseterre
Saint Kitts and Nevis, Saint Thomas Lowland
Saint Kitts and Nevis, Saint Thomas Middle Island
Saint Kitts and Nevis, Trinity Palmetto Point
Saint Lucia, Anse-la-Raye
Saint Lucia, Castries
Saint Lucia, Choiseul
Saint Lucia, Dauphin
Saint Lucia, Dennery
Saint Lucia, Gros-Islet
Saint Lucia, Laborie
Saint Lucia, Micoud
Saint Lucia, Praslin
Saint Lucia, Soufriere
Saint Lucia, Vieux-Fort
Saint Vincent and the Grenadines, Charlotte
Saint Vincent and the Grenadines, Grenadines
Saint Vincent and the Grenadines, Saint Andrew
Saint Vincent and the Grenadines, Saint David
Saint Vincent and the

Grenadines, Saint George
Saint Vincent and the Grenadines, Saint Patrick
Samoa, Aiga-i-le-Tai
Samoa, Atua
Samoa, Fa
Samoa, Gaga
Samoa, Gagaifomauga
Samoa, Palauli
Samoa, Satupa
Samoa, Tuamasaga
Samoa, Va
Samoa, Vaisigano
San Marino, Acquaviva
San Marino, Borgo Maggiore
San Marino, Chiesanuova
San Marino, Domagnano
San Marino, Faetano
San Marino, Fiorentino
San Marino, Monte Giardino
San Marino, San Marino
San Marino, Serravalle
Sao Tome and Principe, Principe
Sao Tome and Principe, Sao Tome
Saudi Arabia, Al Bahah
Saudi Arabia, Al Hudud ash Shamaliyah
Saudi Arabia, Al Jawf
Saudi Arabia, Al Jawf
Saudi Arabia, Al Madinah
Saudi Arabia, Al Qasim
Saudi Arabia, Al Qurayyat
Saudi Arabia, Ar Riyad
Saudi Arabia, Ash Sharqiyah
Saudi Arabia, Ha'il
Saudi Arabia, Jizan
Saudi Arabia, Makkah
Saudi Arabia, Najran
Saudi Arabia, Tabuk
Senegal, Dakar
Senegal, Diourbel
Senegal, Fatick
Senegal, Kaolack
Senegal, Kolda
Senegal, Louga
Senegal, Matam
Senegal, Saint-Louis
Senegal, Tambacounda
Senegal, Thies
Senegal, Ziguinchor
Serbia, Kosovo
Serbia, Vojvodina
Seychelles, Anse aux Pins
Seychelles, Anse Boileau
Seychelles, Anse Etoile
Seychelles, Anse Louis
Seychelles, Anse Royale
Seychelles, Baie

Lazare
Seychelles, Baie Sainte Anne
Seychelles, Beau Vallon
Seychelles, Bel Air
Seychelles, Bel Ombre
Seychelles, Cascade
Seychelles, Glacis
Seychelles, Grand' Anse
Seychelles, Grand' Anse
Seychelles, La Digue
Seychelles, La Riviere Anglaise
Seychelles, Mont Buxton
Seychelles, Mont Fleuri
Seychelles, Plaisance
Seychelles, Pointe La Rue
Seychelles, Port Glaud
Seychelles, Saint Louis
Seychelles, Takamaka
Sierra Leone, Eastern
Sierra Leone, Northern
Sierra Leone, Southern
Sierra Leone, Western Area
Slovakia, Banska Bystrica
Slovakia, Bratislava
Slovakia, Kosice
Slovakia, Nitra
Slovakia, Presov
Slovakia, Trencin
Slovakia, Trnava
Slovakia, Zilina
Slovenia, Ajdovscina
Slovenia, Beltinci
Slovenia, Bled
Slovenia, Bohinj
Slovenia, Borovnica
Slovenia, Bovec
Slovenia, Brda
Slovenia, Brezice
Slovenia, Brezovica
Slovenia, Celje
Slovenia, Cerklje na Gorenjskem
Slovenia, Cerknica
Slovenia, Cerkno
Slovenia, Crensovci
Slovenia, Crna na Koroskem
Slovenia, Crnomelj
Slovenia, Divaca
Slovenia, Dobrepolje
Slovenia, Dobrova-Horjul-Polhov Gradec
Slovenia, Dol pri Ljubljani
Slovenia, Domzale
Slovenia, Dornava
Slovenia, Dravograd
Slovenia, Duplek
Slovenia, Gorenja Vas-Poljane
Slovenia, Gorisnica
Slovenia, Gornja Radgona
Slovenia, Gornji Grad
Slovenia, Gornji Petrovci
Slovenia, Grosuplje

Slovenia, Hrastnik
Slovenia, Hrpelje-Kozina
Slovenia, Idrija
Slovenia, Ig
Slovenia, Ilirska Bistrica
Slovenia, Ivancna Gorica
Slovenia, Izola-Isola
Slovenia, Jesenice
Slovenia, Jursinci
Slovenia, Kamnik
Slovenia, Kanal
Slovenia, Kidricevo
Slovenia, Kobarid
Slovenia, Kobilje
Slovenia, Kocevje
Slovenia, Komen
Slovenia, Koper-Capodistria
Slovenia, Kozje
Slovenia, Kranj
Slovenia, Kranjska Gora
Slovenia, Krsko
Slovenia, Kungota
Slovenia, Kuzma
Slovenia, Lasko
Slovenia, Lenart
Slovenia, Litija
Slovenia, Ljubljana
Slovenia, Ljubno
Slovenia, Ljutomer
Slovenia, Logatec
Slovenia, Loska Dolina
Slovenia, Loski Potok
Slovenia, Luce
Slovenia, Lukovica
Slovenia, Majsperk
Slovenia, Maribor
Slovenia, Medvode
Slovenia, Menges
Slovenia, Metlika
Slovenia, Mezica
Slovenia, Miren-Kostanjevica
Slovenia, Mislinja
Slovenia, Moravce
Slovenia, Moravske Toplice
Slovenia, Mozirje
Slovenia, Murska Sobota
Slovenia, Muta
Slovenia, Naklo
Slovenia, Nazarje
Slovenia, Nova Gorica
Slovenia, Novo Mesto
Slovenia, Odranci
Slovenia, Ormoz
Slovenia, Osilnica
Slovenia, Pesnica
Slovenia, Piran
Slovenia, Pivka
Slovenia, Podcetrtek
Slovenia, Postojna
Slovenia, Preddvor
Slovenia, Ptuj
Slovenia, Puconci
Slovenia, Racam
Slovenia, Radece
Slovenia, Radenci
Slovenia, Radlje ob Dravi
Slovenia, Radovljica
Slovenia, Ribnica
Slovenia, Rogaska Slatina

Slovenia, Rogasovci
Slovenia, Rogatec
Slovenia, Ruse
Slovenia, Semic
Slovenia, Sencur
Slovenia, Sentilj
Slovenia, Sentjernej
Slovenia, Sentjur pri Celju
Slovenia, Sevnica
Slovenia, Sezana
Slovenia, Skocjan
Slovenia, Skofja Loka
Slovenia, Skofljica
Slovenia, Slovenj Gradec
Slovenia, Slovenska Bistrica
Slovenia, Slovenske Konjice
Slovenia, Smarje pri Jelsah
Slovenia, Smartno ob Paki
Slovenia, Sostanj
Slovenia, Starse
Slovenia, Store
Slovenia, Sveti Jurij
Slovenia, Tolmin
Slovenia, Trbovlje
Slovenia, Trebnje
Slovenia, Trzic
Slovenia, Turnisce
Slovenia, Velenje
Slovenia, Velike Lasce
Slovenia, Videm
Slovenia, Vipava
Slovenia, Vitanje
Slovenia, Vodice
Slovenia, Vojnik
Slovenia, Vrhnika
Slovenia, Vuzenica
Slovenia, Zagorje ob Savi
Slovenia, Zalec
Slovenia, Zavrc
Slovenia, Zelezniki
Slovenia, Ziri
Slovenia, Zrece
Solomon Islands, Central
Solomon Islands, Choiseul
Solomon Islands, Guadalcanal
Solomon Islands, Isabel
Solomon Islands, Makira
Solomon Islands, Malaita
Solomon Islands, Rennell and Bellona
Solomon Islands, Temotu
Solomon Islands, Western
Somalia, Awdal
Somalia, Bakool
Somalia, Banaadir
Somalia, Bari
Somalia, Bay
Somalia, Galguduud
Somalia, Gedo
Somalia, Hiiraan
Somalia, Jubbada Dhexe
Somalia, Jubbada Hoose

Somalia, Mudug
Somalia, Nugaal
Somalia, Nugaal
Somalia, Sanaag
Somalia, Shabeellaha Dhexe
Somalia, Shabeellaha Hoose
Somalia, Sool
Somalia, Togdheer
Somalia, Woqooyi Galbeed
Somalia, Woqooyi Galbeed
South Africa, Eastern Cape
South Africa, Free State
South Africa, Gauteng
South Africa, KwaZulu-Natal
South Africa, Limpopo
South Africa, Mpumalanga
South Africa, North-West
South Africa, North-Western Province
South Africa, Northern Cape
South Africa, Western Cape
Spain, Andalucia
Spain, Aragon
Spain, Asturias
Spain, Canarias
Spain, Cantabria
Spain, Castilla y Leon
Spain, Castilla-La Mancha
Spain, Catalonia
Spain, Comunidad Valenciana
Spain, Extremadura
Spain, Galicia
Spain, Islas Baleares
Spain, La Rioja
Spain, Madrid
Spain, Murcia
Spain, Navarra
Spain, Pais Vasco
Sri Lanka, Amparai
Sri Lanka, Anuradhapura
Sri Lanka, Badulla
Sri Lanka, Batticaloa
Sri Lanka, Central
Sri Lanka, Colombo
Sri Lanka, Galle
Sri Lanka, Gampaha
Sri Lanka, Hambantota
Sri Lanka, Jaffna
Sri Lanka, Kalutara
Sri Lanka, Kandy
Sri Lanka, Kegalla
Sri Lanka, Kurunegala
Sri Lanka, Mannar
Sri Lanka, Matale
Sri Lanka, Matara
Sri Lanka, Moneragala
Sri Lanka, Mullaittivu
Sri Lanka, North Central
Sri Lanka, North Western
Sri Lanka, Northern
Sri Lanka, Nuwara Eliya
Sri Lanka,

Polonnaruwa
Sri Lanka, Puttalam
Sri Lanka, Ratnapura
Sri Lanka,
Sabaragamuwa
Sri Lanka, Southern
Sri Lanka, Trincomalee
Sri Lanka, Uva
Sri Lanka, Vavuniya
Sri Lanka, Western
Sudan, Al Istiwa'iyah
Sudan, Al Khartum
Sudan, Al Wahadah
State
Sudan, Al Wusta
Sudan, Ash
Shamaliyah
Sudan, Ash Sharqiyah
Sudan, Bahr al Ghazal
Sudan, Central
Equatoria State
Sudan, Darfur
Sudan, Kurdufan
Sudan, Upper Nile
Suriname, Brokopondo
Suriname,
Commewijne
Suriname, Coronie
Suriname, Marowijne
Suriname, Nickerie
Suriname, Para
Suriname, Paramaribo
Suriname, Saramacca
Suriname, Sipaliwini
Suriname, Wanica
Swaziland, Hhohho
Swaziland, Lubombo
Swaziland, Manzini
Swaziland, Praslin
Swaziland, Shiselweni
Sweden, Blekinge Lan
Sweden, Dalarnas Lan
Sweden, Gavleborgs
Lan
Sweden, Gotlands Lan
Sweden, Hallands Lan
Sweden, Jamtlands
Lan
Sweden, Jonkopings
Lan
Sweden, Kalmar Lan
Sweden, Kronobergs
Lan
Sweden, Norrbottens
Lan
Sweden, Orebro Lan
Sweden,
Ostergotlands Lan
Sweden, Skane Lan
Sweden,
Sodermanlands Lan
Sweden, Stockholms
Lan
Sweden, Uppsala Lan
Sweden, Varmlands
Lan
Sweden,
Vasterbottens Lan
Sweden,
Vasternorrlands Lan
Sweden,
Vastmanlands Lan
Sweden, Vastra
Gotaland
Switzerland, Aargau
Switzerland, Ausser-
Rhoden
Switzerland, Basel-
Landschaft

Switzerland, Basel-Stadt
Switzerland, Bern
Switzerland, Fribourg
Switzerland, Geneve
Switzerland, Glarus
Switzerland, Graubuenden
Switzerland, Inner-Rhoden
Switzerland, Jura
Switzerland, Luzern
Switzerland, Neuchatel
Switzerland, Nidwalden
Switzerland, Obwalden
Switzerland, Sankt Gallen
Switzerland, Schaffhausen
Switzerland, Schwyz
Switzerland, Solothurn
Switzerland, Thurgau
Switzerland, Ticino
Switzerland, Uri
Switzerland, Valais
Switzerland, Vaud
Switzerland, Zug
Switzerland, Zuerich
Syrian Arab Republic, Al Hasakah
Syrian Arab Republic, Al Ladhiqiyah
Syrian Arab Republic, Al Qunaytirah
Syrian Arab Republic, Ar Raqqah
Syrian Arab Republic, As Suwayda'
Syrian Arab Republic, Dar
Syrian Arab Republic, Dayr az Zawr
Syrian Arab Republic, Dimashq
Syrian Arab Republic, Halab
Syrian Arab Republic, Hamah
Syrian Arab Republic, Hims
Syrian Arab Republic, Idlib
Syrian Arab Republic, Rif Dimashq
Syrian Arab Republic, Tartus

T – Z

Taiwan, Fu-chien
Taiwan, Kao-hsiung
Taiwan, T'ai-pei
Taiwan, T'ai-wan
Tajikistan, Khatlon
Tajikistan, Kuhistoni Badakhshon
Tajikistan, Sughd
Tanzania, Arusha
Tanzania, Dar es Salaam
Tanzania, Dodoma
Tanzania, Iringa
Tanzania, Kagera
Tanzania, Kigoma
Tanzania, Kilimanjaro
Tanzania, Lindi
Tanzania, Manyara
Tanzania, Mara
Tanzania, Mbeya

Tanzania, Morogoro
Tanzania, Mtwara
Tanzania, Mwanza
Tanzania, Pemba North
Tanzania, Pemba South
Tanzania, Pwani
Tanzania, Rukwa
Tanzania, Ruvuma
Tanzania, Shinyanga
Tanzania, Singida
Tanzania, Tabora
Tanzania, Tanga
Tanzania, Zanzibar Central
Tanzania, Zanzibar North
Tanzania, Zanzibar Urban
Thailand, Amnat Charoen
Thailand, Ang Thong
Thailand, Buriram
Thailand, Chachoengsao
Thailand, Chai Nat
Thailand, Chaiyaphum
Thailand, Chanthaburi
Thailand, Chiang Mai
Thailand, Chiang Rai
Thailand, Chon Buri
Thailand, Chumphon
Thailand, Kalasin
Thailand, Kamphaeng Phet
Thailand, Kanchanaburi
Thailand, Khon Kaen
Thailand, Krabi
Thailand, Krung Thep
Thailand, Lampang
Thailand, Lamphun
Thailand, Loei
Thailand, Lop Buri
Thailand, Mae Hong Son
Thailand, Maha Sarakham
Thailand, Mukdahan
Thailand, Nakhon Nayok
Thailand, Nakhon Pathom
Thailand, Nakhon Phanom
Thailand, Nakhon Phanom
Thailand, Nakhon Ratchasima
Thailand, Nakhon Sawan
Thailand, Nakhon Si Thammarat
Thailand, Nan
Thailand, Narathiwat
Thailand, Nong Bua Lamphu
Thailand, Nong Khai
Thailand, Nonthaburi
Thailand, Pathum Thani
Thailand, Pattani
Thailand, Phangnga
Thailand, Phatthalung
Thailand, Phayao
Thailand, Phetchabun
Thailand, Phetchaburi
Thailand, Phichit

Thailand, Phitsanulok
Thailand, Phra Nakhon Si Ayutthaya
Thailand, Phrae
Thailand, Phuket
Thailand, Prachin Buri
Thailand, Prachuap Khiri Khan
Thailand, Ranong
Thailand, Ratchaburi
Thailand, Rayong
Thailand, Roi Et
Thailand, Sa Kaeo
Thailand, Sakon Nakhon
Thailand, Samut Prakan
Thailand, Samut Sakhon
Thailand, Samut Songkhram
Thailand, Saraburi
Thailand, Satun
Thailand, Sing Buri
Thailand, Sisaket
Thailand, Songkhla
Thailand, Sukhothai
Thailand, Suphan Buri
Thailand, Surat Thani
Thailand, Surin
Thailand, Tak
Thailand, Trang
Thailand, Trat
Thailand, Ubon Ratchathani
Thailand, Ubon Ratchathani
Thailand, Udon Thani
Thailand, Uthai Thani
Thailand, Uttaradit
Thailand, Yala
Thailand, Yasothon
The Bahamas, Acklins and Crooked Islands
The Bahamas, Bimini
The Bahamas, Cat Island
The Bahamas, Exuma
The Bahamas, Freeport
The Bahamas, Fresh Creek
The Bahamas, Governor's Harbour
The Bahamas, Green Turtle Cay
The Bahamas, Harbour Island
The Bahamas, High Rock
The Bahamas, Inagua
The Bahamas, Kemps Bay
The Bahamas, Long Island
The Bahamas, Marsh Harbour
The Bahamas, Mayaguana
The Bahamas, New Providence
The Bahamas, Nicholls town and Berry Islands
The Bahamas, Ragged Island
The Bahamas, Rock Sound
The Bahamas, San

Salvador and Rum Cay
The Bahamas, Sandy Point
Togo, Centrale
Togo, Kara
Togo, Maritime
Togo, Plateaux
Togo, Savanes
Tonga, Ha
Tonga, Tongatapu
Tonga, Vava
Trinidad and Tobago, Arima
Trinidad and Tobago, Caroni
Trinidad and Tobago, Mayaro
Trinidad and Tobago, Nariva
Trinidad and Tobago, Port-of-Spain
Trinidad and Tobago, Saint Andrew
Trinidad and Tobago, Saint David
Trinidad and Tobago, Saint George
Trinidad and Tobago, Saint Patrick
Trinidad and Tobago, San Fernando
Trinidad and Tobago, Tobago
Trinidad and Tobago, Victoria
Tunisia, Aiana
Tunisia, Al Mahdia
Tunisia, Al Munastir
Tunisia, Bajah
Tunisia, Ben Arous
Tunisia, Bizerte
Tunisia, El Kef
Tunisia, Gabes
Tunisia, Jendouba
Tunisia, Kairouan
Tunisia, Kasserine
Tunisia, Kebili
Tunisia, Madanin
Tunisia, Manouba
Tunisia, Nabeul
Tunisia, Qafsah
Tunisia, Sfax
Tunisia, Sidi Bou Zid
Tunisia, Siliana
Tunisia, Sousse
Tunisia, Tataouine
Tunisia, Tozeur
Tunisia, Tunis
Tunisia, Zaghouan
Turkey, Adana
Turkey, Adiyaman
Turkey, Afyonkarahisar
Turkey, Agri
Turkey, Aksaray
Turkey, Amasya
Turkey, Ankara
Turkey, Antalya
Turkey, Ardahan
Turkey, Artvin
Turkey, Aydin
Turkey, Balikesir
Turkey, Bartin
Turkey, Batman
Turkey, Bayburt
Turkey, Bilecik
Turkey, Bingol
Turkey, Bitlis
Turkey, Bolu

Turkey, Burdur
Turkey, Bursa
Turkey, Canakkale
Turkey, Cankiri
Turkey, Corum
Turkey, Denizli
Turkey, Diyarbakir
Turkey, Duzce
Turkey, Edirne
Turkey, Elazig
Turkey, Erzincan
Turkey, Erzurum
Turkey, Eskisehir
Turkey, Gaziantep
Turkey, Giresun
Turkey, Gumushane
Turkey, Hakkari
Turkey, Hatay
Turkey, Igdir
Turkey, Isparta
Turkey, Istanbul
Turkey, Izmir
Turkey,
Kahramanmaras
Turkey, Karabuk
Turkey, Karaman
Turkey, Kars
Turkey, Kastamonu
Turkey, Kayseri
Turkey, Kilis
Turkey, Kirikkale
Turkey, Kirklareli
Turkey, Kirsehir
Turkey, Kocaeli
Turkey, Konya
Turkey, Kutahya
Turkey, Malatya
Turkey, Manisa
Turkey, Mardin
Turkey, Mersin
Turkey, Mugla
Turkey, Mus
Turkey, Nevsehir
Turkey, Nigde
Turkey, Ordu
Turkey, Osmaniye
Turkey, Rize
Turkey, Sakarya
Turkey, Samsun
Turkey, Sanliurfa
Turkey, Siirt
Turkey, Sinop
Turkey, Sirnak
Turkey, Sivas
Turkey, Tekirdag
Turkey, Tokat
Turkey, Trabzon
Turkey, Tunceli
Turkey, Usak
Turkey, Van
Turkey, Yalova
Turkey, Yozgat
Turkey, Zonguldak
Turkmenistan, Ahal
Turkmenistan, Balkan
Turkmenistan,
Dashoguz
Turkmenistan, Lebap
Turkmenistan, Mary
Uganda, Adjumani
Uganda, Apac
Uganda, Arua
Uganda, Bugiri
Uganda, Bundibugyo
Uganda, Bushenyi
Uganda, Busia
Uganda, Gulu
Uganda, Hoima
Uganda, Iganga

Uganda, Jinja
Uganda, Kabarole
Uganda, Kaberamaido
Uganda, Kalangala
Uganda, Kampala
Uganda, Kamuli
Uganda, Kamwenge
Uganda, Kanungu
Uganda, Kapchorwa
Uganda, Kasese
Uganda, Katakwi
Uganda, Kayunga
Uganda, Kibale
Uganda, Kiboga
Uganda, Kisoro
Uganda, Kitgum
Uganda, Kotido
Uganda, Kumi
Uganda, Kyenjojo
Uganda, Lira
Uganda, Luwero
Uganda, Masaka
Uganda, Masindi
Uganda, Mayuge
Uganda, Mbale
Uganda, Mbarara
Uganda, Moroto
Uganda, Moyo
Uganda, Mpigi
Uganda, Mubende
Uganda, Mukono
Uganda, Nakapiripirit
Uganda, Nakasongola
Uganda, Nebbi
Uganda, Ntungamo
Uganda, Pader
Uganda, Pallisa
Uganda, Rakai
Uganda, Rukungiri
Uganda, Sembabule
Uganda, Sironko
Uganda, Soroti
Uganda, Tororo
Uganda, Wakiso
Uganda, Yumbe
Ukraine, Cherkas'ka Oblast'
Ukraine, Chernihivs'ka Oblast'
Ukraine, Chernivets'ka Oblast'
Ukraine, Dnipropetrovs'ka Oblast'
Ukraine, Donets'ka Oblast'
Ukraine, Ivano-Frankivs'ka Oblast'
Ukraine, Kharkivs'ka Oblast'
Ukraine, Khersons'ka Oblast'
Ukraine, Khmel'nyts'ka Oblast'
Ukraine, Kirovohrads'ka Oblast'
Ukraine, Krym
Ukraine, Kyyiv
Ukraine, Kyyivs'ka Oblast'
Ukraine, L'vivs'ka Oblast'
Ukraine, Luhans'ka Oblast'
Ukraine, Mykolayivs'ka Oblast'
Ukraine, Odes'ka Oblast'
Ukraine, Poltavs'ka

Oblast'
Ukraine, Rivnens'ka Oblast'
Ukraine, Sevastopol'
Ukraine, Sums'ka Oblast'
Ukraine, Ternopil's'ka Oblast'
Ukraine, Vinnyts'ka Oblast'
Ukraine, Volyns'ka Oblast'
Ukraine, Zakarpats'ka Oblast'
Ukraine, Zaporiz'ka Oblast'
Ukraine, Zhytomyrs'ka Oblast'
United Arab Emirates, Abu Dhabi
United Arab Emirates, Ajman
United Arab Emirates, Dubai
United Arab Emirates, Fujairah
United Arab Emirates, Ras Al Khaimah
United Arab Emirates, Sharjah
United Arab Emirates, Umm Al Quwain
United Kingdom, Aberdeen City
United Kingdom, Aberdeenshire
United Kingdom, Angus
United Kingdom, Antrim
United Kingdom, Ards
United Kingdom, Argyll and Bute
United Kingdom, Armagh
United Kingdom, Ballymena
United Kingdom, Ballymoney
United Kingdom, Banbridge
United Kingdom, Barking and Dagenham
United Kingdom, Barnet
United Kingdom, Barnsley
United Kingdom, Bath and North East Somerset
United Kingdom, Bedfordshire
United Kingdom, Belfast
United Kingdom, Bexley
United Kingdom, Birmingham
United Kingdom, Blackburn with Darwen
United Kingdom, Blackpool
United Kingdom, Blaenau Gwent
United Kingdom, Bolton
United Kingdom, Bournemouth

United Kingdom,
Bracknell Forest
United Kingdom,
Bradford
United Kingdom, Brent
United Kingdom,
Bridgend
United Kingdom,
Brighton and Hove
United Kingdom,
Bristol
United Kingdom,
Bromley
United Kingdom,
Buckinghamshire
United Kingdom, Bury
United Kingdom,
Caerphilly
United Kingdom,
Calderdale
United Kingdom,
Cambridgeshire
United Kingdom,
Camden
United Kingdom,
Cardiff
United Kingdom,
Carmarthenshire
United Kingdom,
Carrickfergus
United Kingdom,
Castlereagh
United Kingdom,
Ceredigion
United Kingdom,
Cheshire
United Kingdom,
Clackmannanshire
United Kingdom,
Coleraine
United Kingdom,
Conwy
United Kingdom,
Cookstown
United Kingdom,
Cornwall
United Kingdom,
Coventry
United Kingdom,
Craigavon
United Kingdom,
Croydon
United Kingdom,
Cumbria
United Kingdom,
Darlington
United Kingdom,
Denbighshire
United Kingdom,
Derby
United Kingdom,
Derbyshire
United Kingdom, Derry
United Kingdom,
Devon
United Kingdom,
Doncaster
United Kingdom,
Dorset
United Kingdom, Down
United Kingdom,
Dudley
United Kingdom,
Dumfries and
Galloway
United Kingdom,
Dundee City
United Kingdom,
Dungannon

United Kingdom, Durham
United Kingdom, Ealing
United Kingdom, East Ayrshire
United Kingdom, East Dunbartonshire
United Kingdom, East Lothian
United Kingdom, East Renfrewshire
United Kingdom, East Riding of Yorkshire
United Kingdom, East Sussex
United Kingdom, Edinburgh
United Kingdom, Eilean Siar
United Kingdom, Enfield
United Kingdom, Essex
United Kingdom, Falkirk
United Kingdom, Fermanagh
United Kingdom, Fife
United Kingdom, Flintshire
United Kingdom, Gateshead
United Kingdom, Glasgow City
United Kingdom, Gloucestershire
United Kingdom, Greenwich
United Kingdom, Gwynedd
United Kingdom, Hackney
United Kingdom, Halton
United Kingdom, Hammersmith and Fulham
United Kingdom, Hampshire
United Kingdom, Haringey
United Kingdom, Harrow
United Kingdom, Hartlepool
United Kingdom, Havering
United Kingdom, Herefordshire
United Kingdom, Hertford
United Kingdom, Highland
United Kingdom, Hillingdon
United Kingdom, Hounslow
United Kingdom, Inverclyde
United Kingdom, Isle of Anglesey
United Kingdom, Isle of Wight
United Kingdom, Islington
United Kingdom, Kensington and Chelsea

United Kingdom, Kent
United Kingdom,
Kingston upon Hull
United Kingdom,
Kingston upon Thames
United Kingdom,
Kirklees
United Kingdom,
Knowsley
United Kingdom,
Lambeth
United Kingdom,
Lancashire
United Kingdom, Larne
United Kingdom,
Leeds
United Kingdom,
Leicester
United Kingdom,
Leicestershire
United Kingdom,
Lewisham
United Kingdom,
Limavady
United Kingdom,
Lincolnshire
United Kingdom,
Lisburn
United Kingdom,
Liverpool
United Kingdom,
London
United Kingdom, Luton
United Kingdom,
Magherafelt
United Kingdom,
Manchester
United Kingdom,
Medway
United Kingdom,
Merthyr Tydfil
United Kingdom,
Merton
United Kingdom,
Middlesbrough
United Kingdom,
Midlothian
United Kingdom,
Milton Keynes
United Kingdom,
Monmouthshire
United Kingdom,
Moray
United Kingdom,
Moyle
United Kingdom,
Neath Port Talbot
United Kingdom,
Newcastle upon Tyne
United Kingdom,
Newham
United Kingdom,
Newport
United Kingdom,
Newry and Mourne
United Kingdom,
Newtownabbey
United Kingdom,
Norfolk
United Kingdom,
North Ayrshire
United Kingdom,
North Down
United Kingdom,
North East
Lincolnshire
United Kingdom,
North Lanarkshire
United Kingdom,

North Lincolnshire
United Kingdom,
North Somerset
United Kingdom,
North Tyneside
United Kingdom,
North Yorkshire
United Kingdom,
Northamptonshire
United Kingdom,
Northumberland
United Kingdom,
Nottingham
United Kingdom,
Nottinghamshire
United Kingdom,
Oldham
United Kingdom,
Omagh
United Kingdom,
Orkney
United Kingdom,
Oxfordshire
United Kingdom,
Pembrokeshire
United Kingdom,
Perth and Kinross
United Kingdom,
Peterborough
United Kingdom,
Plymouth
United Kingdom, Poole
United Kingdom,
Portsmouth
United Kingdom,
Powys
United Kingdom,
Reading
United Kingdom,
Redbridge
United Kingdom,
Redcar and Cleveland
United Kingdom,
Renfrewshire
United Kingdom,
Rhondda Cynon Taff
United Kingdom,
Richmond upon
Thames
United Kingdom,
Rochdale
United Kingdom,
Rotherham
United Kingdom,
Rutland
United Kingdom,
Salford
United Kingdom,
Sandwell
United Kingdom,
Scottish Borders
United Kingdom,
Sefton
United Kingdom,
Sheffield
United Kingdom,
Shetland Islands
United Kingdom,
Shropshire
United Kingdom,
Slough
United Kingdom,
Solihull
United Kingdom,
Somerset
United Kingdom,
South Ayrshire
United Kingdom,
South Gloucestershire

United Kingdom,
South Lanarkshire
United Kingdom,
South Tyneside
United Kingdom,
Southampton
United Kingdom,
Southend-on-Sea
United Kingdom,
Southwark
United Kingdom, St.
Helens
United Kingdom,
Staffordshire
United Kingdom,
Stirling
United Kingdom,
Stockport
United Kingdom,
Stockton-on-Tees
United Kingdom,
Stoke-on-Trent
United Kingdom,
Strabane
United Kingdom,
Suffolk
United Kingdom,
Sunderland
United Kingdom,
Surrey
United Kingdom,
Sutton
United Kingdom,
Swansea
United Kingdom,
Swindon
United Kingdom,
Tameside
United Kingdom,
Telford and Wrekin
United Kingdom,
Thurrock
United Kingdom,
Torbay
United Kingdom,
Torfaen
United Kingdom,
Tower Hamlets
United Kingdom,
Trafford
United Kingdom, Vale
of Glamorgan
United Kingdom,
Wakefield
United Kingdom,
Walsall
United Kingdom,
Waltham Forest
United Kingdom,
Wandsworth
United Kingdom,
Warrington
United Kingdom,
Warwickshire
United Kingdom, West
Berkshire
United Kingdom, West
Dunbartonshire
United Kingdom, West
Lothian
United Kingdom, West
Sussex
United Kingdom,
Westminster
United Kingdom,
Wigan
United Kingdom,
Wiltshire
United Kingdom,

Windsor and Maidenhead
United Kingdom, Wirral
United Kingdom, Wokingham
United Kingdom, Wolverhampton
United Kingdom, Worcestershire
United Kingdom, Wrexham
United Kingdom, York
United States, Alabama
United States, Alaska
United States, American Samoa
United States, Arizona
United States, Arkansas
United States, Armed Forces Americas
United States, Armed Forces Europe
United States, Armed Forces Pacific
United States, California
United States, Colorado
United States, Connecticut
United States, Delaware
United States, District of Columbia
United States, Federated States of Micronesia
United States, Florida
United States, Georgia
United States, Guam
United States, Hawaii
United States, Idaho
United States, Illinois
United States, Indiana
United States, Iowa
United States, Kansas
United States, Kentucky
United States, Louisiana
United States, Maine
United States, Marshall Islands
United States, Maryland
United States, Massachusetts
United States, Michigan
United States, Minnesota
United States, Mississippi
United States, Missouri
United States, Montana
United States, Nebraska
United States, Nevada
United States, New Hampshire
United States, New Jersey
United States, New Mexico
United States, New York

United States, North Carolina
United States, North Dakota
United States, Northern Mariana Islands
United States, Ohio
United States, Oklahoma
United States, Oregon
United States, Palau
United States, Pennsylvania
United States, Puerto Rico
United States, Rhode Island
United States, South Carolina
United States, South Dakota
United States, Tennessee
United States, Texas
United States, Utah
United States, Vermont
United States, Virgin Islands
United States, Virginia
United States, Washington
United States, West Virginia
United States, Wisconsin
United States, Wyoming
Uruguay, Artigas
Uruguay, Canelones
Uruguay, Cerro Largo
Uruguay, Colonia
Uruguay, Durazno
Uruguay, Flores
Uruguay, Florida
Uruguay, Lavalleja
Uruguay, Maldonado
Uruguay, Montevideo
Uruguay, Paysandu
Uruguay, Rio Negro
Uruguay, Rivera
Uruguay, Rocha
Uruguay, Salto
Uruguay, San Jose
Uruguay, Soriano
Uruguay, Tacuarembo
Uruguay, Treinta y Tres
Uzbekistan, Andijon
Uzbekistan, Bukhoro
Uzbekistan, Farghona
Uzbekistan, Jizzakh
Uzbekistan, Khorazm
Uzbekistan, Namangan
Uzbekistan, Nawoiy
Uzbekistan, Qashqadaryo
Uzbekistan, Qoraqalpoghiston
Uzbekistan, Samarqand
Uzbekistan, Sirdaryo
Uzbekistan, Surkhondaryo
Uzbekistan, Toshkent
Uzbekistan, Toshkent
Vanuatu, Ambrym

Vanuatu, Aoba
Vanuatu, Efate
Vanuatu, Epi
Vanuatu, Malakula
Vanuatu, Malampa
Vanuatu, Paama
Vanuatu, Penama
Vanuatu, Pentecote
Vanuatu, Sanma
Vanuatu, Shefa
Vanuatu, Shepherd
Vanuatu, Tafea
Vanuatu, Torba
Venezuela, Amazonas
Venezuela, Anzoategui
Venezuela, Apure
Venezuela, Aragua
Venezuela, Barinas
Venezuela, Bolivar
Venezuela, Carabobo
Venezuela, Cojedes
Venezuela, Delta Amacuro
Venezuela, Dependencias Federales
Venezuela, Distrito Federal
Venezuela, Falcon
Venezuela, Guarico
Venezuela, Lara
Venezuela, Merida
Venezuela, Miranda
Venezuela, Monagas
Venezuela, Nueva Esparta
Venezuela, Portuguesa
Venezuela, Sucre
Venezuela, Tachira
Venezuela, Trujillo
Venezuela, Vargas
Venezuela, Yaracuy
Venezuela, Zulia
Vietnam, An Giang
Vietnam, An Giang
Vietnam, Ba Ria-Vung Tau
Vietnam, Ben Tre
Vietnam, Binh Dinh
Vietnam, Binh Thuan
Vietnam, Can Tho
Vietnam, Cao Bang
Vietnam, Da Nang
Vietnam, Dac Lac
Vietnam, Dak Lak
Vietnam, Dak Nong
Vietnam, Dien Bien
Vietnam, Dong Nai
Vietnam, Dong Thap
Vietnam, Dong Thap
Vietnam, Ha Giang
Vietnam, Ha Nam
Vietnam, Ha Noi
Vietnam, Ha Tay
Vietnam, Ha Tinh
Vietnam, Hai Duong
Vietnam, Hai Phong
Vietnam, Hau Giang
Vietnam, Ho Chi Minh
Vietnam, Ho Chi Minh
Vietnam, Hoa Binh
Vietnam, Hung Yen
Vietnam, Khanh Hoa
Vietnam, Kien Giang
Vietnam, Kien Giang
Vietnam, Kon Tum
Vietnam, Lai Chau
Vietnam, Lam Dong

Vietnam, Lang Son
Vietnam, Lao Cai
Vietnam, Long An
Vietnam, Nam Dinh
Vietnam, Nam Ha
Vietnam, Nghe An
Vietnam, Ninh Binh
Vietnam, Ninh Thuan
Vietnam, Phu Tho
Vietnam, Phu Yen
Vietnam, Quang Binh
Vietnam, Quang Nam
Vietnam, Quang Ngai
Vietnam, Quang Ninh
Vietnam, Quang Tri
Vietnam, Quang Tri
Vietnam, Soc Trang
Vietnam, Son La
Vietnam, Song Be
Vietnam, Tay Ninh
Vietnam, Thai Binh
Vietnam, Thai Nguyen
Vietnam, Thanh Hoa
Vietnam, Thua Thien
Vietnam, Tien Giang
Vietnam, Tra Vinh
Vietnam, Tuyen Quang
Vietnam, Vinh Long
Vietnam, Vinh Phu
Vietnam, Vinh Puc
Province
Yemen, Abyan
Yemen, Adan
Yemen, Al Bayda'
Yemen, Al Ghaydah
Yemen, Al Hudaydah
Yemen, Al Jawf
Yemen, Al Mahrah
Yemen, Al Mahwit
Yemen, Dhamar
Yemen, Hadramawt
Yemen, Hajjah
Yemen, Ibb
Yemen, Lahij
Yemen, Ma'rib
Yemen, Sa
Yemen, San
Yemen, Shabwah
Yemen, Ta
Zambia, Central
Zambia, Copperbelt
Zambia, Eastern
Zambia, Luapula
Zambia, Lusaka
Zambia, North-
Western
Zambia, Northern
Zambia, Southern
Zambia, Western
Zimbabwe, Bulawayo
Zimbabwe, Harare
Zimbabwe, Manicaland
Zimbabwe,
Mashonaland Central
Zimbabwe,
Mashonaland East
Zimbabwe,
Mashonaland West
Zimbabwe, Masvingo
Zimbabwe,
Matabeleland North
Zimbabwe,
Matabeleland South
Zimbabwe, Midlands

Map View

The following countries displayed in the map view of the **Geo Dashboard** also report data by region.

A – G

Australia, Australian Capital Territory
Australia, New South Wales
Australia, Northern Territory
Australia, Queensland
Australia, South Australia
Australia, Tasmania
Australia, Victoria
Australia, Western Australia

Belgium, Antwerpen
Belgium, Hainaut
Belgium, Liege
Belgium, Limburg
Belgium, Luxembourg
Belgium, Namur
Belgium, Oost-Vlaanderen
Belgium, West-Vlaanderen
Belgium, Brabant Wallon
Belgium, Brussels Hoofdstedelijk Gewest
Belgium, Vlaams-Brabant
Belgium, Flanders
Belgium, Wallonia

Brazil, Acre
Brazil, Alagoas
Brazil, Amapa
Brazil, Amazonas
Brazil, Bahia
Brazil, Ceara
Brazil, Distrito Federal
Brazil, Espirito Santo
Brazil, Goias
Brazil, Maranhao
Brazil, Mato Grosso
Brazil, Mato Grosso do Sul
Brazil, Minas Gerais
Brazil, Para
Brazil, Paraiba
Brazil, Parana
Brazil, Permambuco
Brazil, Piaui
Brazil, Rio de Janeiro
Brazil, Rio Grande do Norte
Brazil, Rio Grande do Sul
Brazil, Rondonia
Brazil, Roraima
Brazil, Santa Catarina
Brazil, Sao Paulo
Brazil, Sergipe
Brazil, Tocantins

Canada, Alberta
Canada, British Columbia
Canada, Manitoba
Canada, New Brunswick
Canada, Newfoundland and Labrador
Canada, Northwest Territories
Canada, Nova Scotia
Canada, Nunavut
Canada, Ontario
Canada, Prince Edward Island
Canada, Quebec
Canada, Saskatchewan
Canada, Yukon Territory

China, Anhui
China, Beijing
China, Chongqing
China, Fujian
China, Gansu
China, Guangdong
China, Guangxi
China, Guizhou
China, Hainan
China, Hebei
China, Heilongjiang
China, Henan
China, Hubei
China, Hunan
China, Jiangsu
China, Jiangxi
China, Jilin
China, Liaoning
China, Nei Mongol
China, Ningxia
China, Qinghai
China, Shaanxi
China, Shandong
China, Shanghai
China, Shanxi
China, Sichuan
China, Tianjin
China, Xinjiang
China, Xizang
China, Yunnan
China, Zhejiang

France, Alsace
France, Aquitaine
France, Auvergne
France, Basse-Normandie
France, Bourgogne
France, Bretagne
France, Centre
France, Champagne-Ardenne
France, Corse
France, Franche-Comte
France, Haute-Normandie
France, Ile-de-France
France, Languedoc-Roussillon
France, Limousin
France, Lorraine
France, Midi-Pyrenees
France, Nord-Pas-de-Calais
France, Pays de la Loire
France, Picardie
France, Poitou-Charentes
France, Provence-Alpes-Cote d'Azur
France, Rhone-Alpes

Germany, Baden-Wurttemberg
Germany, Bayern
Germany, Berlin
Germany, Brandenburg
Germany, Bremen
Germany, Hamburg
Germany, Hessen
Germany, Mecklenburg-Vorpommern
Germany, Niedersachsen
Germany, Nordrhein-Westfalen
Germany, Rheinland-Pfalz
Germany, Saarland
Germany, Sachsen
Germany, Sachsen-Anhalt
Germany, Schleswig-Holstein
Germany, Thuringen

H – M

India, Andaman and Nicobar Islands
India, Andhra Pradesh
India, Arunachal Pradesh
India, Assam
India, Bihar
India, Chandigarh
India, Chhattisgarh
India, Dadra and Nagar Haveli
India, Daman and Diu
India, Delhi
India, Goa
India, Gujarat
India, Haryana
India, Himachal Pradesh
India, Jammu and Kashmir
India, Jharkhand
India, Karnataka
India, Kerala
India, Lakshadweep
India, Madhya Pradesh
India, Maharashtra
India, Manipur
India, Meghalaya
India, Mizoram
India, Nagaland
India, Orissa
India, Puducherry
India, Punjab
India, Rajasthan
India, Sikkim
India, Tamil Nadu
India, Tripura
India, Uttar Pradesh
India, Uttarakhand
India, West Bengal

Indonesia, Aceh
Indonesia, Bali
Indonesia, Banten
Indonesia, Bengkulu
Indonesia, Gorontalo
Indonesia, Irian Jaya Barat
Indonesia, Jakarta Raya
Indonesia, Jambi
Indonesia, Jawa Barat
Indonesia, Jawa Tengah
Indonesia, Jawa Timur
Indonesia, Kalimantan Barat
Indonesia, Kalimantan Selatan
Indonesia, Kalimantan Tengah
Indonesia, Kalimantan Timur
Indonesia, Kepulauan Bangka Belitung
Indonesia, Kepulauan Riau
Indonesia, Lampung
Indonesia, Maluku
Indonesia, Maluku Utara
Indonesia, Nusa Tenggara Barat
Indonesia, Nusa Tenggara Timur
Indonesia, Papua
Indonesia, Riau
Indonesia, Sulawesi Barat
Indonesia, Sulawesi Selatan
Indonesia, Sulawesi Tengah
Indonesia, Sulawesi Tenggara
Indonesia, Sulawesi Utara
Indonesia, Sumatera Barat
Indonesia, Sumatera Selatan
Indonesia, Sumatera Utara
Indonesia, Yogyakarta

Italy, Abruzzi
Italy, Aosta Valley
Italy, Basilicata
Italy, Calabria
Italy, Campania
Italy, Corsica (Corse) - France
Italy, Emilia-Romagna
Italy, Friuli-Venezia Giulia
Italy, Lazio
Italy, Liguria
Italy, Lombardia
Italy, Malta
Italy, Marche
Italy, Molise
Italy, Piemonte
Italy, Puglia
Italy, San Marino
Italy, Sardignia
Italy, Sicily
Italy, Tuscany
Italy, Trentino-Alto Adige
Italy, Umbria
Italy, Vatican City (Holy See)
Italy, Veneto

Japan, Aichi
Japan, Akita
Japan, Aomori
Japan, Chiba
Japan, Ehime
Japan, Fukui
Japan, Fukuoka
Japan, Fukushima
Japan, Gifu
Japan, Gumma
Japan, Hiroshima
Japan, Hokkaido
Japan, Hyogo
Japan, Ibaraki
Japan, Ishikawa
Japan, Iwate
Japan, Kagawa
Japan, Kagoshima
Japan, Kanagawa
Japan, Kochi
Japan, Kumamoto
Japan, Kyoto
Japan, Mie
Japan, Miyagi
Japan, Miyazaki
Japan, Nagano
Japan, Nagasaki
Japan, Nara
Japan, Niigata
Japan, Oita
Japan, Okayama
Japan, Okinawa
Japan, Osaka
Japan, Saga
Japan, Saitama
Japan, Shiga
Japan, Shimane
Japan, Shizuoka
Japan, Tochigi
Japan, Tokushima
Japan, Tokyo
Japan, Tottori
Japan, Toyama
Japan, Wakayama
Japan, Yamagata
Japan, Yamaguchi
Japan, Yamanashi

Malaysia, Johor
Malaysia, Kedah
Malaysia, Kelantan
Malaysia, Kuala Lumpur
Malaysia, Labuan
Malaysia, Melaka
Malaysia, Negeri Sembilan
Malaysia, Pahang
Malaysia, Perak
Malaysia, Perlis
Malaysia, Pulau Pinang
Malaysia, Putrajaya
Malaysia, Sabah
Malaysia, Sarawak
Malaysia, Selangor
Malaysia, Terengganu

N – S

Netherlands, Drenthe
Netherlands, Friesland
Netherlands, Gelderland
Netherlands, Groningen
Netherlands, Limburg
Netherlands, Noord-Brabant
Netherlands, Noord-Holland
Netherlands, Utrecht
Netherlands, Zeeland
Netherlands, Zuid-Holland
Netherlands, Overijssel
Netherlands, Flevoland

Poland, Dolnoslaskie
Poland, Kujawsko-Pomorskie
Poland, Lodzkie
Poland, Lubelskie
Poland, Lubuskie
Poland, Malopolskie
Poland, Mazowieckie
Poland, Opolskie
Poland, Podkarpackie
Poland, Podlaskie
Poland, Pomorskie
Poland, Slaskie
Poland, Swietokrzyskie
Poland, Warminsko-Mazurskie
Poland, Wielkopolskie
Poland, Zachodniopomorskie

Russian Federation, Altai Krai
Russian Federation, Altai Republic
Russian Federation, Amur Oblast
Russian Federation, Arkhangelsk Oblast
Russian Federation, Astrakhan Oblast
Russian Federation, Belgorod Oblast
Russian Federation, Bryansk Oblast
Russian Federation, Buryat Republic
Russian Federation, Chechen Republic

Russian Federation,
Chelyabinsk Oblast
Russian Federation,
Chukotka
Autonomous Okrug
Russian Federation,
Chuvash Republic
Russian Federation,
Irkutsk Oblast
Russian Federation,
Ivanovo Oblast
Russian Federation,
Jewish Autonomous
Oblast
Russian Federation,
Kabardino-Balkar
Republic
Russian Federation,
Kaliningrad Oblast
Russian Federation,
Kaluga Oblast
Russian Federation,
Kamchatka Krai
Russian Federation,
Karachay-Cherkess
Republic
Russian Federation,
Kemerovo Oblast
Russian Federation,
Khabarovsk Krai
Russian Federation,
Khanty-Mansi
Autonomous Okrug—
Yugra
Russian Federation,
Kirov Oblast
Russian Federation,
Komi Republic
Russian Federation,
Kostroma Oblast
Russian Federation,
Krasnodar Krai
Russian Federation,
Krasnoyarsk Krai
Russian Federation,
Kurgan Oblast
Russian Federation,
Kursk Oblast
Russian Federation,
Leningrad Oblast
Russian Federation,
Lipetsk Oblast
Russian Federation,
Magadan Oblast
Russian Federation,
Mari El Republic
Russian Federation,
Moscow
Russian Federation,
Moscow Oblast
Russian Federation,
Murmansk Oblast
Russian Federation,
Nenets Autonomous
Okrug
Russian Federation,
Nizhny Novgorod
Oblast
Russian Federation,
Novgorod Oblast
Russian Federation,
Novosibirsk Oblast
Russian Federation,
Omsk Oblast
Russian Federation,
Orenburg Oblast
Russian Federation,
Oryol Oblast
Russian Federation,

Penza Oblast
Russian Federation,
Perm Krai
Russian Federation,
Primorsky Krai
Russian Federation,
Pskov Oblast
Russian Federation,
Republic of Adygea
Russian Federation,
Republic of
Bashkortostan
Russian Federation,
Republic of Dagestan
Russian Federation,
Republic of Ingushetia
Russian Federation,
Republic of Kalmykia
Russian Federation,
Republic of Karelia
Russian Federation,
Republic of Khakassia
Russian Federation,
Republic of Mordovia
Russian Federation,
Republic of North
Ossetia-Alania
Russian Federation,
Republic of Tatarstan
Russian Federation,
Rostov Oblast
Russian Federation,
Ryazan Oblast
Russian Federation,
Saint Petersburg
Russian Federation,
Sakha (Yakutia)
Republic
Russian Federation,
Sakhalin Oblast
Russian Federation,
Samara Oblast
Russian Federation,
Saratov Oblast
Russian Federation,
Smolensk Oblast
Russian Federation,
Stavropol Krai
Russian Federation,
Sverdlovsk Oblast
Russian Federation,
Tambov Oblast
Russian Federation,
Tomsk Oblast
Russian Federation,
Tula Oblast
Russian Federation,
Tver Oblast
Russian Federation,
Tyumen Oblast
Russian Federation,
Tyva Republic
Russian Federation,
Udmurt Republic
Russian Federation,
Ulyanovsk Oblast
Russian Federation,
Vladimir Oblast
Russian Federation,
Volgograd Oblast
Russian Federation,
Vologda Oblast
Russian Federation,
Voronezh Oblast
Russian Federation,
Yamalo-Nenets
Autonomous Okrug
Russian Federation,
Yaroslavl Oblast

Russian Federation, Zabaykalsky Krai

Spain, Islas Baleares
Spain, La Rioja
Spain, Madrid
Spain, Murcia
Spain, Navarra
Spain, Asturias
Spain, Cantabria
Spain, Andalucia
Spain, Aragon
Spain, Canarias
Spain, Castilla-La Mancha
Spain, Castilla y Leon
Spain, Catalonia
Spain, Extremadura
Spain, Galicia
Spain, Pais Vasco
Spain, Comunidad Valenciana

T – Z

Taiwan, Fu-chien
Taiwan, Kao-hsiung
Taiwan, T'ai-pei
Taiwan, T'ai-wan

Thailand, Amnat Charoen
Thailand, Ang Thong
Thailand, Buriram
Thailand, Chachoengsao
Thailand, Chai Nat
Thailand, Chaiyaphum
Thailand, Chanthaburi
Thailand, Chiang Mai
Thailand, Chiang Rai
Thailand, Chon Buri
Thailand, Chumphon
Thailand, Kalasin
Thailand, Kamphaeng Phet
Thailand, Kanchanaburi
Thailand, Khon Kaen
Thailand, Krabi
Thailand, Lampang
Thailand, Lamphun
Thailand, Loei
Thailand, Lop Buri
Thailand, Mae Hong Son
Thailand, Maha Sarakham
Thailand, Mukdahan
Thailand, Nakhon Nayok
Thailand, Nakhon Pathom
Thailand, Nakhon Phanom
Thailand, Nakhon Ratchasima
Thailand, Nakhon Sawan
Thailand, Nakhon Si Thammarat
Thailand, Nan
Thailand, Narathiwat
Thailand, Nong Bua Lamphu
Thailand, Nong Khai
Thailand, Nonthaburi
Thailand, Pathum Thani
Thailand, Pattani

Thailand, Phangnga
Thailand, Phatthalung
Thailand, Phayao
Thailand, Phetchabun
Thailand, Phetchaburi
Thailand, Phichit
Thailand, Phitsanulok
Thailand, Phrae
Thailand, Phra
Nakhon Si Ayutthaya
Thailand, Phuket
Thailand, Prachin Buri
Thailand, Prachuap
Khiri Khan
Thailand, Ranong
Thailand, Ratchaburi
Thailand, Rayong
Thailand, Roi Et
Thailand, Sa Kaeo
Thailand, Sakon
Nakhon
Thailand, Samut
Prakan
Thailand, Samut
Sakhon
Thailand, Samut
Songkhram
Thailand, Saraburi
Thailand, Satun
Thailand, Sing Buri
Thailand, Sisaket
Thailand, Songkhla
Thailand, Sukhothai
Thailand, Suphan Buri
Thailand, Surat Thani
Thailand, Surin
Thailand, Tak
Thailand, Trang
Thailand, Trat
Thailand, Ubon
Ratchathani
Thailand, Udon Thani
Thailand, Uthai Thani
Thailand, Uttaradit
Thailand, Yala
Thailand, Krung Thep

Turkey, Adanag
Turkey, Adyamang
Turkey,
Afyonkarahisar (Afyon)
g
Turkey, Arig
Turkey, Aksarayg
Turkey, Amasyag
Turkey, Ankarag
Turkey, Antalyag
Turkey, Ardahang
Turkey, Artving
Turkey, Aydng
Turkey, Balkesirg
Turkey, Bartng
Turkey, Batmang
Turkey, Bayburtg
Turkey, Bilecikg
Turkey, Bingölg
Turkey, Bitlisg
Turkey, Bolug
Turkey, Burdurg
Turkey, Bursag
Turkey, Çanakkaleg
Turkey, Çankrg
Turkey, Çorumg
Turkey, Denizlig
Turkey, Diyarbakrg
Turkey, Düzceg
Turkey, Edirneg
Turkey, Elazg
Turkey, Erzincang
Turkey, Erzurumg

Turkey, Eskiehirg
Turkey, Gaziantepg
Turkey, Giresung
Turkey, Gümühaneg
Turkey, Hakkarig
Turkey, Hatayg
Turkey, Idirg
Turkey, Ispartag
Turkey, stanbulg
Turkey, zmirg
Turkey,
Kahramanmarag
Turkey, Karabükg
Turkey, Karamang
Turkey, Karsg
Turkey, Kastamonug
Turkey, Kayserig
Turkey, Kilisg
Turkey, Krkkaleg
Turkey, Krklarelig
Turkey, Krehirg
Turkey, Kocaelig
Turkey, Konyag
Turkey, Kütahyag
Turkey, Malatyag
Turkey, Manisag
Turkey, Marding
Turkey, Mersing
Turkey, Mulag
Turkey, Mug
Turkey, Nevehirg
Turkey, Nideg
Turkey, Ordug
Turkey, Osmaniyeg
Turkey, Rizeg
Turkey, Sakaryag
Turkey, Samsung
Turkey, anlurfag
Turkey, Siirtg
Turkey, Sinopg
Turkey, rnakg
Turkey, Sivasg
Turkey, Tekirdag
Turkey, Tokatg
Turkey, Trabzong
Turkey, Tuncelig
Turkey, Uakg
Turkey, Vang
Turkey, Yalovag
Turkey, Yozgatg
Turkey, Zonguldakg

United Kingdom,
Channel Islands

United Kingdom,
England

United Kingdom,
Ireland

United Kingdom, Isle
of Man

United Kingdom,
Northern Ireland

United States,
Alabama
United States, Alaska
United States, Arizona
United States,
Arkansas
United States,
California
United States,
Colorado
United States,
Connecticut
United States,

Delaware
United States, District of Columbia
United States, Florida
United States, Georgia
United States, Hawaii
United States, Idaho
United States, Illinois
United States, Indiana
United States, Iowa
United States, Kansas
United States, Kentucky
United States, Louisiana
United States, Maine
United States, Maryland
United States, Massachusetts
United States, Michigan
United States, Minnesota
United States, Mississippi
United States, Missouri
United States, Montana
United States, Nebraska
United States, Nevada
United States, New Hampshire
United States, New Jersey
United States, New Mexico
United States, New York
United States, North Carolina
United States, North Dakota
United States, Ohio
United States, Oklahoma
United States, Oregon
United States, Pennsylvania
United States, Rhode Island
United States, South Carolina
United States, South Dakota
United States, Tennessee
United States, Texas
United States, Utah
United States, Vermont
United States, Virginia
United States, Washington
United States, West Virginia
United States, Wisconsin
United States, Wyoming

Troubleshoot Browser RUM

These pages provide methods to solve common problems:

- Browser RUM License Troubleshooting
- Browser RUM Metrics Not Reported
- Pages Not Monitored
- Injection Problems
- Browser Snapshot Problems
- Connection Problems
- Create a HAR File to Troubleshoot Web Download Issues

Browser RUM License Troubleshooting

Related pages:

- EUM Accounts, Licenses, and App Keys

The following sections discuss how to solve common license problems.

No EUM Account License Key

The EUM account license is separate from the Controller license. Although the license file includes the EUM license key, the EUM account license is provisioned separately.

Controllers cannot share an EUM license key, but applications can.

If you need an EUM license, call your AppDynamics sales representative or email salesops@appdynamics.com.

Browser RUM Not Working in Production

If you are running more than one Controller, each instance requires a unique license. Make sure you are not trying to use the same license on more than one Controller.

Browser RUM Metrics Not Reported

This page provides a high-level overview to check your Browser RUM setup.

Check for Load

Make sure there is load running on your app for the pages that you want to monitor for the selected period.

When Browser RUM discovers a new page, iframe, or Ajax call for the first time, there will be a several minute delay before data for it appears in the Controller UI.

- It takes up to two or three minutes for the agent to register with the Controller. No metrics are captured during this time.
- After the page is registered and traffic starts flowing, it can take an additional two or three minutes for the metrics to appear in the AppDynamics console.
- After that, the delay between an end-user click and the appearance of Browser RUM data is two to three minutes.

Confirm Browser RUM Is Enabled

Make sure you enable Browser RUM for the app. Enable it if it is disabled. See Enable and Disable Browser Monitoring.

Ensure Pages Are Instrumented

Examine the source of your web page to ensure that you have instrumented the pages that you want to monitor.

Check Connectivity

Make sure there is connectivity from the browser to the EUM Cloud.

Examine Beacons for Issues

If beacons are not present or their status is not valid, verify your Browser RUM configuration. See Set Up and Access Browser RUM.

To examine your instrumented application for issues, verify the following:

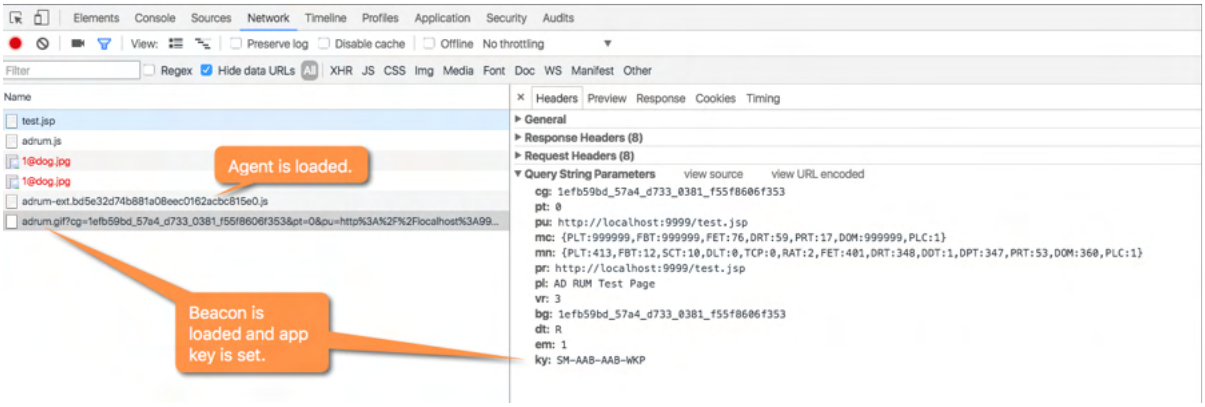
- The JavaScript Agent extension (`adrum.ext`) is loaded and its status is OK (200).
- The EUM beacon is sent:
    - As a `.gif` request—older browsers
    - A GET request for `adrum.gif` is loaded, and its status is OK (200).
    - The `ky` parameter in the beacon is set to your application key.
    - As a CORS-based request—deprecated
    - A POST request has been sent with a status of OK (200).
    - The app key is sent in the body of the beacon.
    - As a CORS-based request
    - A POST request has been sent with a status of OK (200).
    - The app key is sent in the URL.

Beacons for using image requests have an inherent length limitation. Because of this, URLs longer than 180 characters, page names longer than 50 characters, and user data (key and value combined) longer than 113 characters are not supported in these browsers.

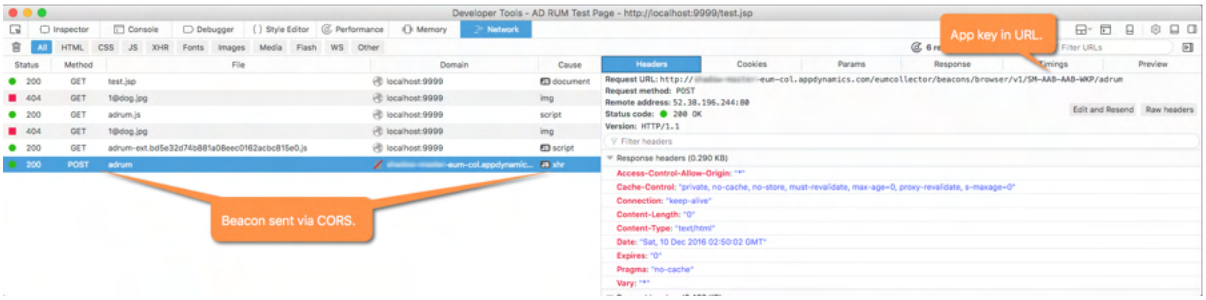
Use Developer Tools to Examine Your Application

You can use the developer tools in your web browser to examine your application.

This screenshot shows a `.gif`-based beacon in Chrome.



This screenshot shows a CORS-based beacon with a key in the URL path.



Create a HAR file to confirm that the browser is downloading the expected resources for your web page.

Pages Not Monitored

You may find that some pages have not been monitored as expected. This page may help you determine the reason why these pages have not been monitored.

Verify Pages Have Been Injected

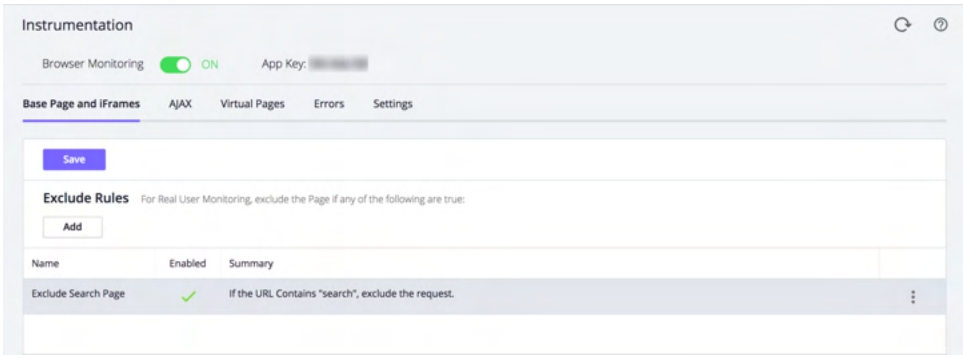
If only some web pages are not reporting data, first verify that those pages have been injected with the JavaScript Agent. See Verifying that the JavaScript Agent for Browser RUM was injected.

Check Exclude Rules

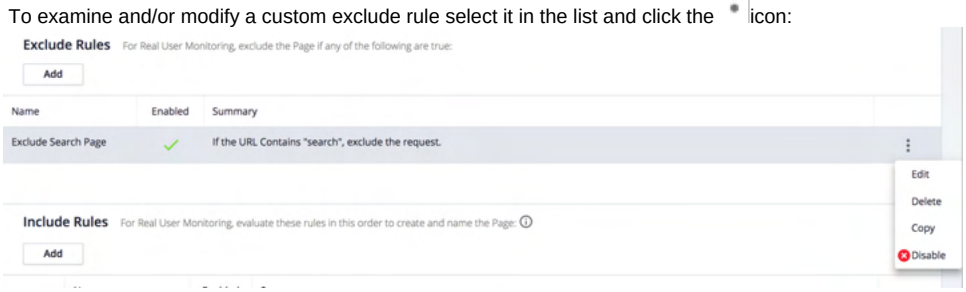
It the agent has been injected, the page may have been excluded from monitoring by custom exclude rules. You can check and modify these rules.

To access custom exclude rules for pages:

1. Click **Configuration > Instrumentation**.
2. Click the Base Pages and iFrames tab.
3. From **Base Pages and iFrames**, check the custom rules under the **Exclude Rules** section.



4. To examine and/or modify a custom exclude rule select it in the list and click the icon:



5. If you want to remove a custom exclude rule, select and click **Delete**.

Check Automatic Injection Configuration

In addition, certain pages could have been excluded by the injection configuration. This can happen when automatic injection is used with a limited set of pages enabled for injection. If you used automatic injection, check your automatic injection configuration to see if the missing pages are enabled for injection.

Examine the **Request Match Rules** and **Request Exclude Rules** lists under **Only enable Automatic Injection for certain Pages** in the Automatic JavaScript Injection tab. See Automatic Injection of the JavaScript Agent. Pages can also be bypassed by assisted injection using injection rules when an injection rule specifies only classes and methods to be injected. If you used assisted injection with injection rules, check your injection rules. See Assisted Injection.

Configure the JavaScript Agent to Monitor Older Browsers

The JavaScript Agent by default supports modern browser and no longer tests or certifies the collection and reporting of data from older browsers. If you find that older browsers are accessing your web pages and that they are not being monitored, you can configure the JavaScript Agent to monitor older browser with the following configuration.

```
(function(config){
  ...
  "beacon": { "neverSendImageBeacon": false },
  ...
})(window['adrum-config'] || (window['adrum-config'] = {}));
```

Ajax Requests Not Monitored

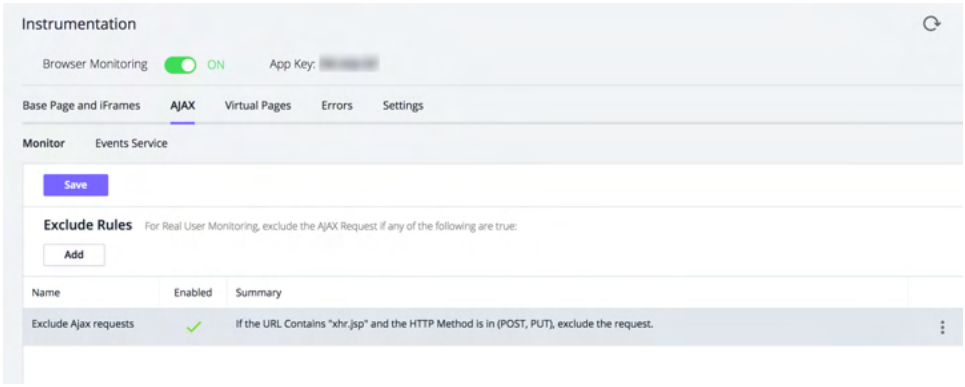
You may find that some Ajax requests have not been monitored as expected. The page may help you determine why these pages have not been monitored.

Check Exclude Rules

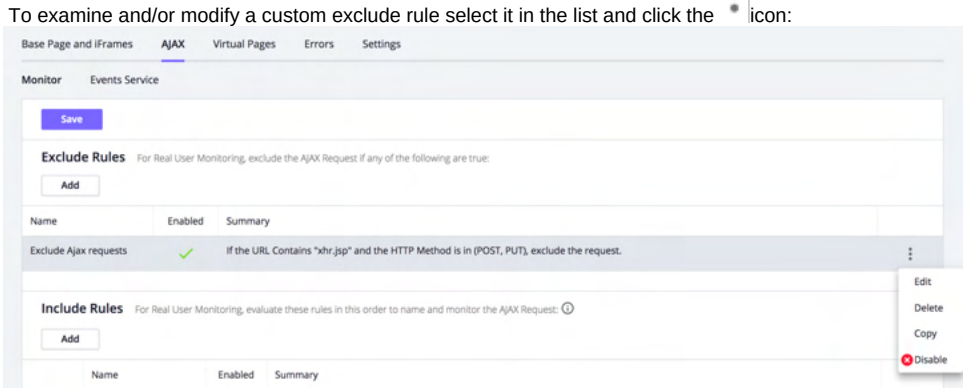
It the agent has been injected, the Ajax event may have been excluded from monitoring by custom exclude rules. You can check and modify these rules.

To access custom exclude rules for pages:

1. Click **Configuration > Instrumentation**.
2. From the AJAX tab, check the custom rules under the **Exclude Rules** section.



3. To examine and/or modify a custom exclude rule select it in the list and click the icon:



4. If you want to remove a custom exclude rule, select and click **Delete**.

Verify That XHR Objects Were Reused Correctly

Browser RUM can monitor multiple Ajax calls made from a reused `XMLHttpRequest` object, but if the `XMLHttpRequest` object is reused (opened) before the last request is completed, the JavaScript agent cannot collect all the metrics for all the Ajax requests.

Incorrect Method to Reuse XMLHttpRequest Object

In the example below, the `xhr` object is making a request to `/multi-bt` before the request to `/xhr/uptime` has completed, so the JavaScript Agent only reports the second Ajax request to `/multi-bt`.

```
function reuseXHRBeforeReady() {
    var xhr = newXhr();
    xhr.open("GET", "/xhr/uptime");
    xhr.onreadystatechange = safe(function() {
        if (xhr.readyState == 4) {
            if (typeof(console) !== "undefined" && console !== null && console.log)
                console.log("onreadystatechange for xh1: " + xhr.responseText);
                document.getElementById("result").innerHTML = xhr.responseText;
        }
    });
    xhr.send(null);
    xhr.open("GET", "/multi-bt");
    xhr.onreadystatechange = safe(function() {
        if (xhr.readyState == 4) {
            if (typeof(console) !== "undefined" && console !== null && console.log)
                console.log("onreadystatechange for xhr2: " + xhr.responseText);
                document.getElementById("result").innerHTML = xhr.responseText;
        }
    });
    xhr.send(null);
}
```

When an Ajax request cannot be reported because the `XMLHttpRequest` object is reused (opened) before the last request is completed, the JavaScript Agent logs that an Ajax request was *not* reported in a message similar to the one below. You will most likely have to modify your JavaScript so that new Ajax requests are only made after previous requests are completed.

```
EXT~ The reused XHR object calls open() before finishes last job. No event of XHR to '" + this._adrumAjaxT.url()
+ "' is reported.
```

Correct Method to Reuse XMLHttpRequest Object

The correct method to reuse the `XMLHttpRequest` object to make more than one request is to nest the calls so that new calls are only made when previous calls have been completed. In the example below, the call to `/xhr/multi-bt2` is only made when the call to `/multi-bt` has completed, so both calls are reported as Ajax requests.

```
function reuseXHR() {
    var xhr = newXhr();
    xhr.open("GET", "/xhr/uptime");
    xhr.onreadystatechange = safe(function() {
        if (xhr.readyState == 4) {
            document.getElementById("result").innerHTML = xhr.responseText;
            xhr.open("GET", "/multi-bt");
            xhr.onreadystatechange = safe(function() {
                if (xhr.readyState == 4) {
                    if (typeof(console) !== "undefined" && console !== null && console.log)
                        console.log("onreadystatechange for xhr2: " + xhr.responseText);
                    document.getElementById("result").innerHTML = xhr.responseText;

                    // Reuse xhr again
                    xhr.open("GET", "/xhr/multi-bt2");
                    xhr.onreadystatechange = safe(function() {
                        console.log("/xhr/multi-bt2 responded.");
                    });
                    xhr.send(null);
                }
            });
            xhr.send(null);
        }
    });
    xhr.send(null);
}
```

Check Injection Configuration

In addition, certain Ajax requests could have been filtered out by the injection configuration. This can happen when the injection uses XHR filters to limit which Ajax requests are monitored. Check your injection configuration to verify that the filtering did not prevent the calls from being monitored.

Injection Problems

You must inject the JavaScript Agent for Browser RUM into every page that you want to monitor.

Verify Injection

View the source of your web page. When automatic or assisted injection is used, you should see the script for the JavaScript Agent inline in the web page. When manual injection is used, you will see:

```
<script src="/path_to_adrum.js"/>
```

If you used manual injection, use the normal procedures that you use to verify other types of code changes in your web pages. Keep in mind that various caches, such as the server page, CDN or browser caches, can prevent the page from actually being reloaded. If you cannot get manual injection to work, try one of the other injection schemes if they are available for your platform. See Inject the JavaScript Agent for information about the various injection strategies.

Verify Automatic Injection is Enabled

To verify that automatic injection is enabled:

1. Open the business application that is injecting the JavaScript Agent.
2. Click **Configuration > User Experience App Integration > JavaScript Agent Injection**.
3. From **Inject the JavaScript Agent configured for this Browser App**, confirm that your browser app receiving the automatic injection is selected.
4. Select the Automatic Injection tab.
5. Confirm that the **Enable Automatic Injection of JavaScript** checkbox is checked.
6. Verify that automatic injection is enabled for all of the business transactions that you want to monitor. If some of those business transactions are in the **Automatic injection possible, but not enabled list**, move them to the **Automatic injection enabled** list. If the business transaction that you want to monitor does not appear in either list, automatic injection is not possible for that business transaction.
7. Click **Save** if you made any changes.

See Automatic Injection of the JavaScript Agent for information about creating and enabling injection rules.

Verify Assisted Injection is Enabled

To verify that assisted attribute injection is enabled:

1. Open the business application that is injecting the JavaScript Agent.
2. Click **Configuration > User Experience App Integration > JavaScript Agent Injection**.
3. From **Inject the JavaScript Agent configured for this Browser App**, confirm that your browser app receiving the automatic injection is selected.
4. Select the Configure JavaScript Injection tab.
5. Confirm that the **Request Attribute Injection** is checked.
6. Click **Save** if you made any changes.

To verify that assisted injection using injection rules is enabled:

1. Open the business application that is injecting the JavaScript Agent.
2. Click **Configuration > User Experience App Integration > JavaScript Agent Injection**.
3. From **Inject the JavaScript Agent configured for this Browser App**, confirm that your browser app receiving the automatic injection is selected.
4. Select the Configure JavaScript Injection tab.
5. Confirm that there are enabled injection rules in the table under **Create Injection Rules**.
6. From the Where to Inject JavaScript tab, confirm that the rules are correct.
7. From the Inject for these Business Transactions tab, confirm that the rules are applied for the desired business transactions.
8. Click **Save** if you made any changes.

See Assisted Injection for information about creating and enabling injection rules.

Change Your Injection Strategy

If you try one way to inject the JavaScript Agent and it does not work, it is best to undo the current injection configuration before implementing another one.

- To undo automatic injection, clear the **Enable Automatic Injection of JavaScript** check box.
- To undo assisted injection using attribute injection, clear the **Request Attribute Injection** check box, and manually delete the code snippets.
- To undo manual injection, manually delete the JavaScript Agent code from your web pages.
- To undo assisted injection using injection rules, clear the **Enable** check box for each injection rule in the injection rules list.

If multiple copies of the agent exist on a page, the second copy does not execute.

Fix Page Rendering Issues

The rendering of pages in some browsers, in particular, Internet Explorer (IE), may be adversely affected by the JavaScript Agent injection.

After you have confirmed that the JavaScript Agent injection is causing the issue, try the following:

1. Inject the JavaScript Agent after the last <meta> element.
2. Inject the JavaScript Agent just before the closing <head> element.

If you are using automatic injection, see Troubleshoot Automatic Injection.

> Injecting the JavaScript Agent just before the closing <head> tag is *not* the recommended practice because the JavaScript Agent will not capture the resource timing metrics for resources loaded before it. Thus, only do this as a last resort.

Inject After Last Meta Element

To avoid rendering issues and capture resource timing metrics, you are recommended to inject the JavaScript injection right after the last <meta> element in the <head> as shown below.

```
<head>
  <meta charset="UTF-8">
  <meta name="description" content="Fancy Web Page">
  <meta name="keywords" content="HTML,CSS,XML,JavaScript">
  <meta name="author" content="John Doe">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  // Injection of the JavaScript Agent after the last <meta> tag
  <script charset='UTF-8'>
    window['adrum-start-time'] = new Date().getTime();
    (function(config){
      config.appKey = 'SH-AAB-AAC-XAR';
      config.adrumExtUrlHttp = 'http://cdn.appdynamics.com';
      config.adrumExtUrlHttps = 'https://cdn.appdynamics.com';
      config.beaconUrlHttp = 'http://eum-col.appdynamics.com';
      config.beaconUrlHttps = 'https://eum-col.appdynamics.com';
      config.xd = {enable : false};
    })(window['adrum-config'] || (window['adrum-config'] = {}));
  </script>
  <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'/>
  // The rest of the JS and CSS files
  <script src="https://code.jquery.com/jquery-<version>.js" integrity="sha256-DZAnKJ/6XZ9si04Hgrsxu
/8s717jcIzLy3oi35EouyE=" crossorigin="anonymous"></script>
  <script src="js/custom-script.js"></script>
  <link type="text/css" rel="stylesheet" href="css/custom-style.css" />
  ...
</head>
```

Inject Before the Closing Head Element

On rare occasions, some browsers, in particular, Internet Explorer (IE), will still render the page incorrectly because of where the JavaScript Agent is injected. In such cases, as a last resort, try injecting the JavaScript Agent right before the closing <head> element:

```
<head>
  <meta charset="UTF-8">
  <meta name="description" content="Fancy Web Page">
  <meta name="keywords" content="HTML,CSS,XML,JavaScript">
  <meta name="author" content="John Doe">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <script src="https://code.jquery.com/jquery-<version>.js" integrity="sha256-DZAnKJ/6XZ9si04Hgrsxu
/8s717jcIzLy3oi35EouyE=" crossorigin="anonymous"></script>
  <script src="js/custom-script.js"></script>
  <link type="text/css" rel="stylesheet" href="css/custom-style.css" />
  ...
  // Injection of the JavaScript Agent before the closing <head> tag
  <script charset='UTF-8'>
    window['adrum-start-time'] = new Date().getTime();
    (function(config){
      config.appKey = 'SH-AAB-AAC-XAR';
      config.adrumExtUrlHttp = 'http://cdn.appdynamics.com';
      config.adrumExtUrlHttps = 'https://cdn.appdynamics.com';
      config.beaconUrlHttp = 'http://eum-col.appdynamics.com';
      config.beaconUrlHttps = 'https://eum-col.appdynamics.com';
      config.xd = {enable : false};
    })(window['adrum-config'] || (window['adrum-config'] = {}));
  </script>
  <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'/>
</head>
```

Troubleshoot Automatic Injection Issues

If you are using the .NET Agent to automatically inject the JavaScript Agent into pages and the pages are not rendering correctly:

1. Use manual injection and make sure that the JavaScript Agent is injected after the last <meta> element.
2. If you cannot use manual injection, add the registered node property `eum-header-beforeendofheadtag` to a Web tier or particular nodes. This node property configures the .NET Agent in the specified tier or nodes to inject the JavaScript Agent right before the closing `<head>` tag.

Browser Snapshot Problems

This page provides help with troubleshooting problems with browser snapshots.

No Browser Snapshots

If you do not see any browser snapshots, it is possible that browser snapshot collection has been disabled. If periodic collection and error collection and slow collection are all disabled, the agent does not collect any browser snapshots. See Configure Browser Snapshot Collection.

Also, check the thresholds that define slow end-user experience. AppDynamics collects browser snapshots only for slow-performing requests, so if the thresholds are set too high, no requests are flagged as slow. See Configure Browser RUM Performance Thresholds.

No Correlation between Browser Snapshots and Business Transactions

You get server-side correlation with browser snapshots only if the business transactions associated with the browser snapshot are running on application servers instrumented with AppDynamics app agents. This could explain why you do not see any or do not see all of the business transactions that you expect to see. Check which of your servers are instrumented by app agents and which are not. You may need to get more AppDynamics app agent licenses to get correlation.

If the app servers are all instrumented with AppDynamics app agents, it is possible that the business transactions that you expect to see were not injected with the JavaScript Agent for Browser Monitoring. This can happen when automatic injection is used with a limited set of business transactions enabled for injection. If you used automatic injection, check your automatic injection configuration to see if the missing business transactions are enabled for injection. See Verifying that the JavaScript Agent for Browser RUM was injected and Automatic Injection of the JavaScript Agent. Business transactions can also be excluded with assisted injection using injection rules when an injection rule specifies only certain business transactions to be injected. If you used assisted injection with injection rules, check your injection rules. See Assisted Injection.

No Transaction Snapshots Associated with Browser Snapshots

Even if all your app servers are instrumented with AppDynamics app agents, it is possible that no associated transaction snapshots were captured at the time of the browser snapshot. For example, if no transactions were slow at the time of the browser snapshot, you probably will not see any transaction snapshots. See Transaction Snapshots for information about when transaction snapshots are captured. You can modify transaction snapshot capture.

On the browser side, if a browser snapshot is associated with a transaction snapshot, you will see it in the **Transaction Snapshots** section of the browser snapshots. See Business Transactions in Browser Snapshots. On the server side, if a transaction snapshot is associated with a browser snapshot, you will see an EUEM GUID in the **ADDITIONAL DATA** tab in the transaction snapshot. See Transaction Snapshots.

Not Getting Full Timing Data for Business Transactions Associated with Browser Snapshots

To ensure full business transaction timing information using older agents and controllers (3.8 and before), you need to inject the JavaScript footer for Browser RUM into the footer of your web pages. Manual injection of the agent does not inject into the footer, so you need to use another injection method to get this functionality.

See Getting Full Timing Data for Associated Business Transactions and Choosing Your Injection Method.

No Correlation between Cross Domain AJAX Requests and Business Transactions

Your JavaScript Agent may not be able to correlate Ajax requests with business transactions because browsers restrict access to certain HTTP response headers. Your application, however, can explicitly give browsers access to HTTP headers by sending the HTTP response header `Access-Control-Expose-Headers` with a list of the headers that you want the JavaScript Agent to access.

The process of correlating cross-domain Ajax requests and business transactions consists of the following two steps:

1. Set the agent, tier, or application node property `always-add-eum-metadata-in-http-headers` to `true`. If this property doesn't exist, you need to add and register it as described in Add a Registered Node Property.
2. Modify the application code, so that the returned response header includes the field `Access-Control-Expose-Headers` as shown below:

```
Access-Control-Expose-Headers: ADRUM_0,ADRUM_1,ADRUM_2,ADRUM_3,ADRUM_4,ADRUM_5,ADRUM_6,ADRUM_7,ADRUM_8,
ADRUM_9,ADRUM_10,ADRUM_11,ADRUM_12,ADRUM_13,ADRUM_14,ADRUM_15,ADRUM_16,ADRUM_17,ADRUM_18
```

Connection Problems

If your browser cannot connect to the AppDynamics EUM cloud and you use an on-premises Controller, but not an on-premises EUM Server, it is possible that:

- You have no Internet connectivity
- A firewall is blocking the port

Verify Connectivity

Navigate to SaaS Domains and select your geolocation. Find the EUM API Domain and add the following endpoint: `eumaggregator/ping`

If you receive a "ping" in the panel, you should be able to connect to the EUM Cloud.

> If you are using IE on a Windows system, make sure the browser itself does not have a proxy (with authentication) set up. If it does, the test link may work but not the actual connection.

Make sure you have also unblocked any firewalls and verified the keystore entries.

Unblock Your Firewall

The Controller needs to be able to use HTTP over SSL (HTTPS) on port 443 to reach the EUM Cloud Aggregator in your region.

Navigate to SaaS Domains and select your geolocation. Find the EUM API Domain and add the following endpoint: `eumaggregator/ping`

If your Controller is behind a firewall, you can either open your Controller's firewall or use a forward proxy.

To open the firewall, see the instructions unique to your firewall.

> You only need to open the firewall for the specific host and port, not for the entire `*.eum-appdynamics.com` domain.

Use a Forward Proxy:

1. Set up an HTTP proxy to your SaaS EUM API Server. Navigate to SaaS Domains and select your geolocation. Find the EUM API Domain and add the following endpoint: `eumaggregator/ping`

   > This is a cleartext/pass-through proxy. Authentication is not supported on the first level. If the client network itself requires authentication, you must set up an intermediate proxy between your Controller and this proxy to pass on the credentials you need to get out of your network.

2. Configure the HTTP proxy host and port in the `<Controller-Installation-Directory>/appserver/glassfish/domains/domain1/config/domain.xml` file by adding the following JVM options to the existing `java-config` element:

   ```
   <jvm-options>
        -Dappdynamics.controller.http.proxyHost=myhost
   </jvm-options>
   <jvm-options>
        -Dappdynamics.controller.http.proxyPort=8888
   </jvm-options>
   ```

3. Restart the Controller's app server.

JavaScript Errors

This page provides helpful tips for finding and solving common JavaScript errors.

Enable Error Reporting

It is possible that reporting is disabled or that certain JavaScript or Ajax errors that you would like to be reported have been configured to be ignored. See Configure JavaScript and Ajax Error Detection.

If another script on your monitored pages sets the JavaScript `window.onerror` event, this setting can interfere with Browser RUM error capture. See Handle the window.onerror Event.

Visualize JavaScript Errors

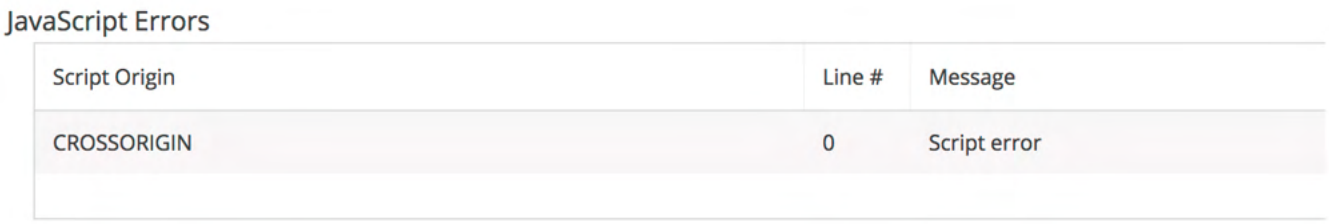
You can use Analytics to query your Browser Request event data and create a JavaScript exception and error summary dashboard.

See Visualize JavaScript Errors.

View Error Information for Cross-Domain JavaScript

By default and for security reasons, most modern browsers do not provide access to the error information in `window.onerror` for scripts that are loaded from other domains. To access this error information, you need to enable cross-origin resource sharing (CORS). Firefox and Chrome both allow you to enable CORS by making small changes to your server and `script` element in your HTML.

If you have **not** enabled CORS, the error message "CROSSORIGIN" displays:



Once CORS is enabled, you will able to see error details like those shown here.

Also, when you can navigate to your page from Chrome and then open **JavaScript Console > Network > Headers**, you'll see the HTTP response header `access-control-allow-origin: *`.

In the following sections, we're going to show you how to make changes to access the error information in `window.onerror` cross-domain scripts.

Server Change

To enable CORS, you need to add this header to responses that are not from the same domain as the caller:

```
Access-Control-Allow-Origin: *
```

Script Tag Change

The `script` element has a new non-standard attribute called `crossorigin`. The most secure value for `crossorigin` would be `anonymous`. So, you'll have to modify your script tags to look like the following.

```
<script src="http://sub.domain.com/adrum.js" crossorigin="anonymous"></script>
```

Browser Support for CORS

In the future, we expect most browsers to enable CORS. Internet Explorer 10+ already has native support for CORS and reports errors to `window.onerror` for both local and cross-domain scripts. See CORS: Browser support to see a list of browsers/layout engines that support CORS.

Browser Synthetic Monitoring

Related pages:

- Browser Synthetic Monitoring versus Browser Real User Monitoring
- Get Started with Browser Synthetic Monitoring

Browsers Synthetic Monitoring uses geographically distributed Synthetic Agents to continuously test key user workflows in your application. This allows you to monitor the correctness and performance of multi-step flows independently of the user-generated load.

There are two ways of using Synthetic:

- **Scheduled jobs**: Use this to test your pages on a recurring basis to ensure they continue to perform well. Upload a Python WebDriver script to exercise multi-step workflows. Using this feature requires a Synthetic Pro license. See Browser Synthetic Licenses for more information.
- **On-demand snapshots**: Use this to collect performance data right now. This is useful for ad-hoc analysis. This feature is available with both Synthetic Lite and Pro licenses.

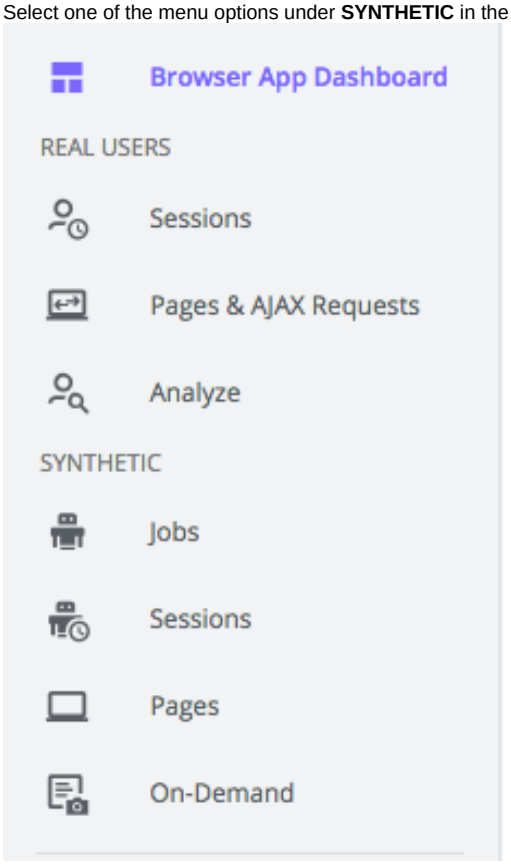
There are two deployment types for Synthetic Agents:

- **Synthetic Hosted Agents:** AppDynamics hosts these Synthetic Agents. This deployment is recommended for testing public websites.
- **Synthetic Private Agents:** You install and host Synthetic Agents on your machines. This enables you to test internal sites and services. For more information, see Synthetic Private Agent Deployment.

Separate licenses are required for the Synthetic Hosted Agent and the Synthetic Private Agent. See Synthetic Agent Licenses for details.

Access Synthetic

1. Open the browser application that you want to test with Synthetic or create a new one if you're just getting started.
2. Select one of the menu options under **SYNTHETIC** in the left navigation bar.



Scheduled Jobs

The **Jobs**, **Sessions**, and **Pages** options are only displayed if you have a Browser Synthetic Pro license.

- **Jobs:** Create and manage recurring jobs. See Get Started with Browser Synthetic Monitoring.
- **Sessions:** View individual executions of your jobs (i.e., "sessions") and perform an analysis to understand where the problems are.
- **Pages:** View aggregated page-level statistics collected from Synthetic. This is particularly useful for comparing Browser RUM with Synthetic.

You can also see some synthetic results in the context of real user traffic in the Browser App Dashboard.

On-Demand Snapshots

Create and view on-demand snapshots using On-Demand.

Website Authentication Methods

The following table lists the authentication methods that are supported and not supported in Synthetic Monitoring.

Supported	Not Supported
Websites that authenticate through an explicit login page	Integrated Windows Authentication (see windows authentication)

Browser Support

Agent	Supported Browsers
Hosted Agent	• Chrome 86 • Chrome 63 • Firefox 72.0.1 • IE 11
Private Synthetic Agent (Windows)	• Chrome 81.0.4044.0 • Firefox 72.0.1 • IE 11
Private Synthetic Agent (K8S Container Agent)	• Chrome 86

Limitations

Browser Extensions

Browser Synthetic Monitoring does not support monitoring for the browser extensions Adobe Flash, Microsoft Silverlight, or Java applets.

Single-Page Applications (SPAs)

Browser Synthetic Monitoring does not fully support SPA. Instead of reporting the base page and virtual pages separately like Browser RUM does, Browser Synthetic Monitor reports all the virtual pages with the base page, so you will only see one page in the Synthetic Sessions.

Multiple Window Tests

Browser Synthetic Monitoring supports running tests in multiple windows for Firefox and Chrome, but *not* for Internet Explorer Browser.

Chrome Version 86

This page provides details about the impact of updating the current browser version to headless Chrome Version 86.

- Chrome version 86 is now available in all 16 AWS locations and the 5 Azure locations will be available shortly.
- When you select Chrome browser, jobs are executed only from the 16 AWS locations even if other locations are selected.
- When a browser other than chrome is selected, jobs continue to be executed in that browser from all the selected locations.
- Chrome version 86 is available only for the SaaS controllers.

Browser Version

Current Browsers	New Browser	Impact	Action
Chrome versions - 64, 79,81, and 83	Headless Chrome Version 86	Headless version includes major security enhancements and changes in the browser engine that results in increased page load time and session duration.	• Increase the timeout configuration as required in the scripts. • Update the performance and alerting configurations as required.
		Some Python based Selenium scripts that run successfully in Chrome version 64 fail with "Script broken" error in Chrome version 86. This happens due to incompatibility between the Selenium commands and the latest Chrome web-driver.	• Update the scripts to fix this issue. • If the scripts pass locally with Chrome version 86 (Headless Chrome), reach out to the AppDynamics support team.
		Current Behaviour: Sessions with 4xx errors are not treated as client errors in certain cases	No action required.

		Headless Chrome does not support extensions or embedded PDF viewer. The PDF is downloaded and viewable in resource waterfall but screenshot is not present. Hence, you can verify the availability of a PDF file but cannot take a screenshot.	You can verify the availability of the PDF document by sending an HTTP request using Python's request library and asserting on the response code.
			Following is a sample script:
			`import requests`
			
			`response = requests.get ("https://www.mohfw.gov.in /pdf /PreventionandManagementof COVID19FLWEnglish.pdf") print ("Got response " + str(response.status_code)) assert (response. status_code == 200)`

Agent Machine Configuration

Current Browser	New Browser	Impact	Action
Machine configuration: 2vCPU 8GB RAM	• Container configuration: min 0.75 vCPU - max 1.25 vCPU, 2GB RAM. • Uses burst-able limits of Kubernetes for better resource utilisation.	The page load time and session duration might increase.	• Increase the timeout configuration as required in the scripts. • Update the performance and alerting configurations as required.

Job execution environment

Current Browser	New Browser	Impact	Action
Jobs are executed in Windows environment.	Jobs are executed in K8S based Alpine containers.	If the website serves content based on operating system (OS), the scripts created based on windows environment fail.	Update the scripts with Linux environment based content.

		Scripts fail with 4xx error in some cases. This issue might occur when the server denies access to Linux agents or when website access is restricted only in Windows environment.	Provide access to Linux agent.
		• The NAT based address can be different from the existing whitelisted IPs. So the existing whitelisting may not work. • Session might fail with '4XX client error'	Update the IP whitelisting based on the AWS IP ranges. IP ranges used in our AWS locations are available here.
		The jobs with OS or file system based commands fail. As per AppDynamics security standards, jobs are executed in stateless environments meant only for the execution of web tests. Hence, OS/file system based commands are not supported.	If you have used OS or file system based commands in your scripts, remove those commands.

Network Throttling

Current Browser	New Browser	Impact	Action
Network throttling feature is implemented using a third party library.	Network throttling is implemented using Chrome browser dev tools protocols.	• The network throttling accuracy is improved. • Jobs which have configured network throttling may observe increased session duration.	Either increase the timeout value or re-configure the throttling speed of the job.

Browser Synthetic Monitoring Versus Browser Real User Monitoring

Browser Synthetic User Monitoring (Browser SUM) and Browser Real-User Monitoring (Browser RUM) both have the same goal of helping you improve your user's experience. The difference between the two is how they are used to accomplish this goal.

This page describes the monitoring differences, and how to use the two monitoring types together.

Identify Issues

Although Browser RUM can detect certain kinds of problems (like JavaScript exceptions), it cannot comprehensively test for functional correctness. For example, you may want to verify that your online store has reasonable prices a list of items. If your site is down entirely, then the JavaScript Agent will never be loaded, so errors or verifications will not be reported. Fortunately, Browser SUM will keep running, discover the error, alert you, and provide detailed information about the problem.

Control Environmental Factors

Performance analysis in Browser RUM is complicated by the wide range of hardware, browsers, operating systems, and networks with which users access your site. Browser SUM uses consistent hardware, software, and network configurations, so if you see deviations in performance, you can be fairly certain a problem exists.

Use Synthetic Metrics and Screenshots to Understand the User Experience

Browser SUM can collect certain data that Browser RUM cannot. For example, Browser SUM provides screenshots, which shows you the current situation. You can also use the Visually Complete and related metrics to understand how users experience page load time.

Performance Versus Workflows

Browser RUM excels at capturing the full breadth of performance that your real users experience. Browser SUM gives you confidence that your key workflows are always working and performing.

Differences Between Synthetic and Browser RUM Metrics

Although Browser SUM and Browser RUM report similar metrics, you should be wary of comparing them because of these differences:

- Hardware
- Network connections
- Different browsers
- Browser caching is not present in synthetic sessions

If you see a sudden change in any of those metrics, however, you should compare the results of Browser SUM and Browser RUM to see if there is an existing problem.

Get Started with Browser Synthetic Monitoring

Related pages:

- Configure Synthetic Jobs
- Synthetic Scripts
- Synthetic Sessions

This page describes Browser Synthetic Monitoring, including how to access the **Jobs** page, create a job that measures a URL, view the results from the **Sessions** page, and finally how to edit and delete your job.

Create a Job

1. Open one of your browser applications.
2. Click **Jobs** to open the **Jobs** page.
3. If you don't have any previous jobs, click **Create a New Job >**. If you have previous jobs, click **Add**.
4. From the **New Job** dialog:
    a. For **Specify your Test**, enter a URL in the **URL** field and "Example Job" in the **Name** field.
    b. For **Choose Browsers**, select Chrome.
    c. For **Choose Locations**, click **+** and select any location.
    d. Click **Save**.

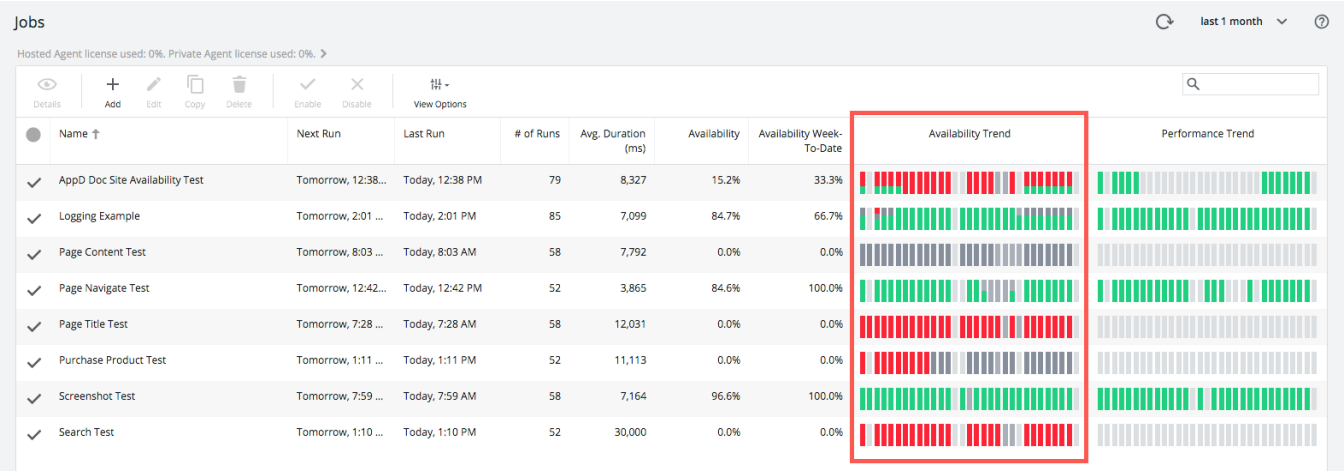
See Configure Synthetic Jobs.

Monitor Job Trends

From the Synthetic Job Dashboard, you can quickly monitor the availability and performance trends of synthetic jobs.

Availability Trends

The **Availability Trend** column that displays the status of the executed jobs as shown in the screenshot below. The **Availability Trend** shows the session status based on the coloring codes listed in Availability Trend Status Codes. The session status is not based on health rules. See Session Status.



> The session status is not based on health rules. See Session Status.

Availability Trend Status Codes

The **Availability Trend** shows the session status based on the coloring codes listed in this table.

Availability Trend Status Color	Session Status
Green	OK
Yellow	WARNING
Red	FAILED
Dark gray	BROKEN

Light gray	INTERNAL_ERROR

Understanding the Availability Metric

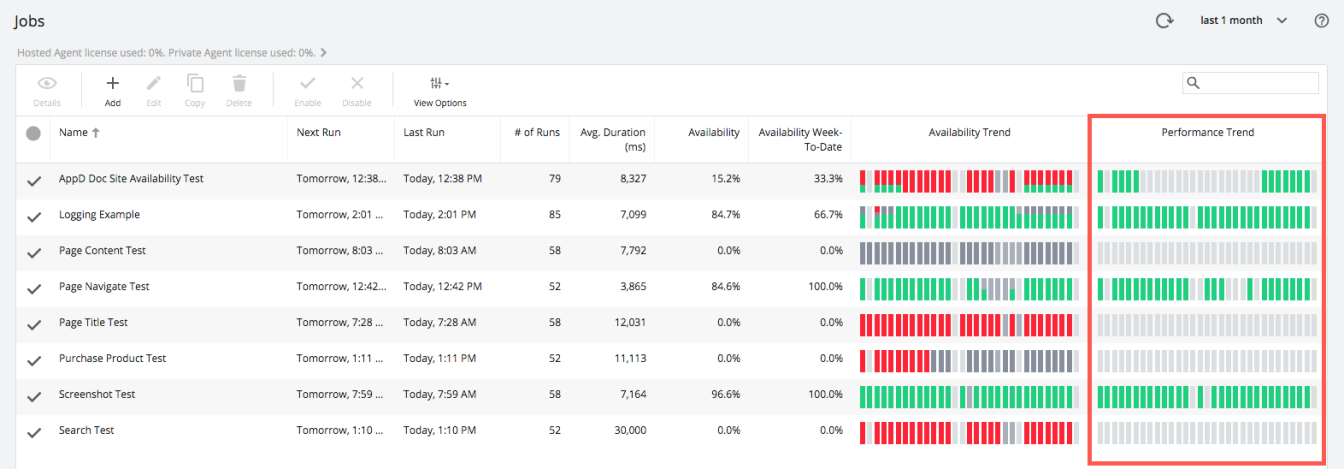
The Availability metric is the percentage of successful sessions for a given time range. That is the percentage of sessions with status as OK or WARNING within the given time range.

The **View Options** enables you to view the availability over periods not restricted to the specified time frame. The following list describes the availability options:

- **Availability Last-24-Hours** - The availability for the last 24 hours from the moment the page is refreshed/opened.
- **Availability Month-to-Date** - The availability from the starting moment (12 AM) of the current month according to user's timezone until the current time (page refreshed/opened).
- **Availability Week-to-Date** - The availability from the starting moment (12 AM) of the current week Monday according to user's timezone, considering week starts on Monday, till now (page refreshed/opened).

Performance Trends

The **Performance Trend** column displays the performance threshold status of the executed jobs as shown in the screenshot below. The **Performance Trend** column only displays successful sessions: those with the status of OK or WARNING. The performance trend statuses are based on the configured performance thresholds for the synthetic job.



For example, if the **Availability Trend** column shows 5 OK and 5 FAILED status counts (50% green, 50% red), then the corresponding **Performance Trend** column will show only 5 OK counts (100% green). If the **Availability Trend** column has 5 failed and 5 broken (50% red, 50% gray) jobs, that is all unsuccessful, then corresponding Performance Trend column will show no count that is an empty bucket (light gray).

Performance Trend Status Codes

As with the availability status, colors are used in the trend column to indicate performance status:

Performance Trend Status Color	Description
Green	No threshold violated: Green.
Yellow	Warning threshold violated.
Red	Critical threshold violated.

A job can have a warning violation for one metric and a critical violation for another metric. In this case, the critical violation will override the warning violation.

View the Results

1. From the **Jobs** page, verify that your job has a check next to it indicating that the job has been enabled.
2. After a few minutes, you should see a time that your job was last run and some metrics.
3. Select your job and click **Details**.
4. You should now see the results for your job on the **Sessions** page.

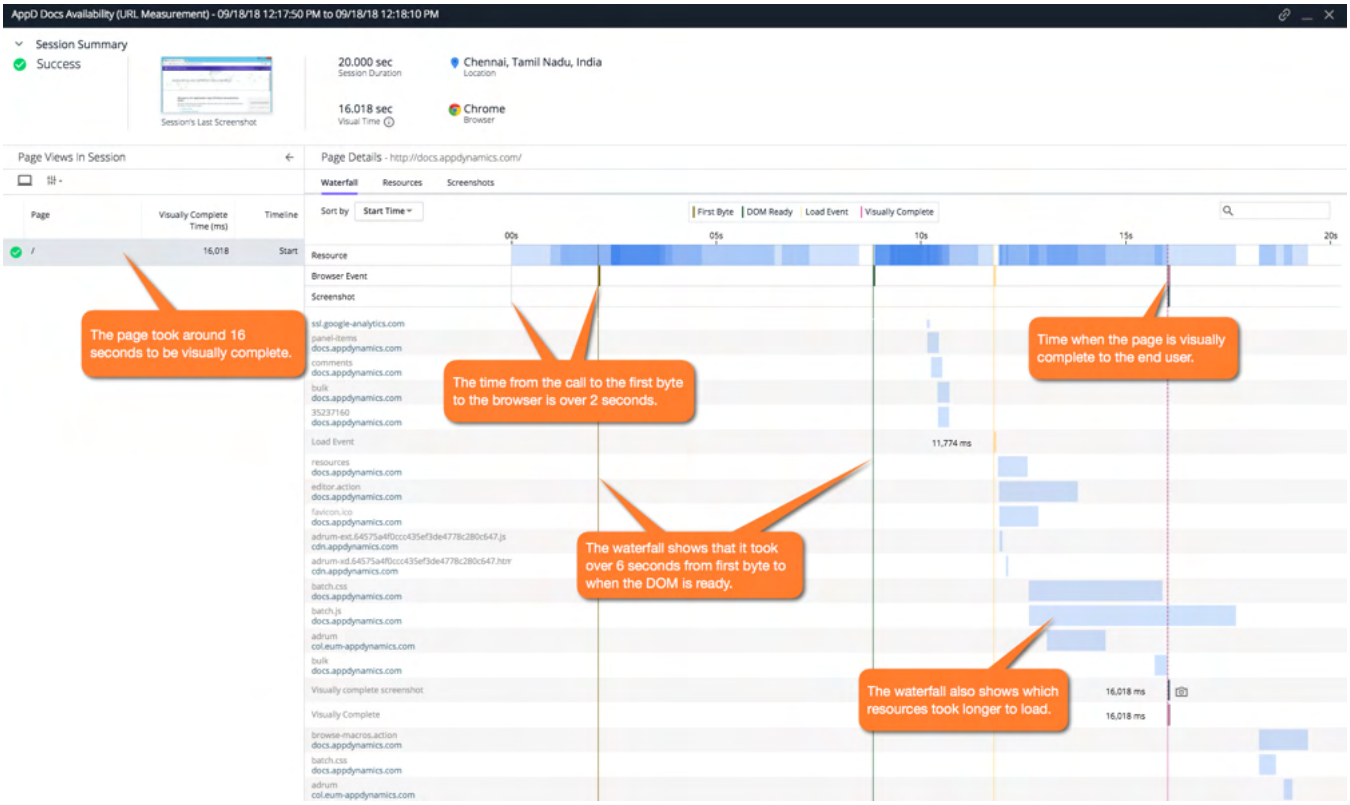
Understand the Results

You can understand the results in several different ways and levels from the **Sessions** page. You can sort session results by criteria and then examine the details of a specific session in the **Session Details.** From the Charts tab of the **Sessions** page, you can then view widgets representing aggregated session data in charts and a configurable set of fields for each session.

Examine Session Details

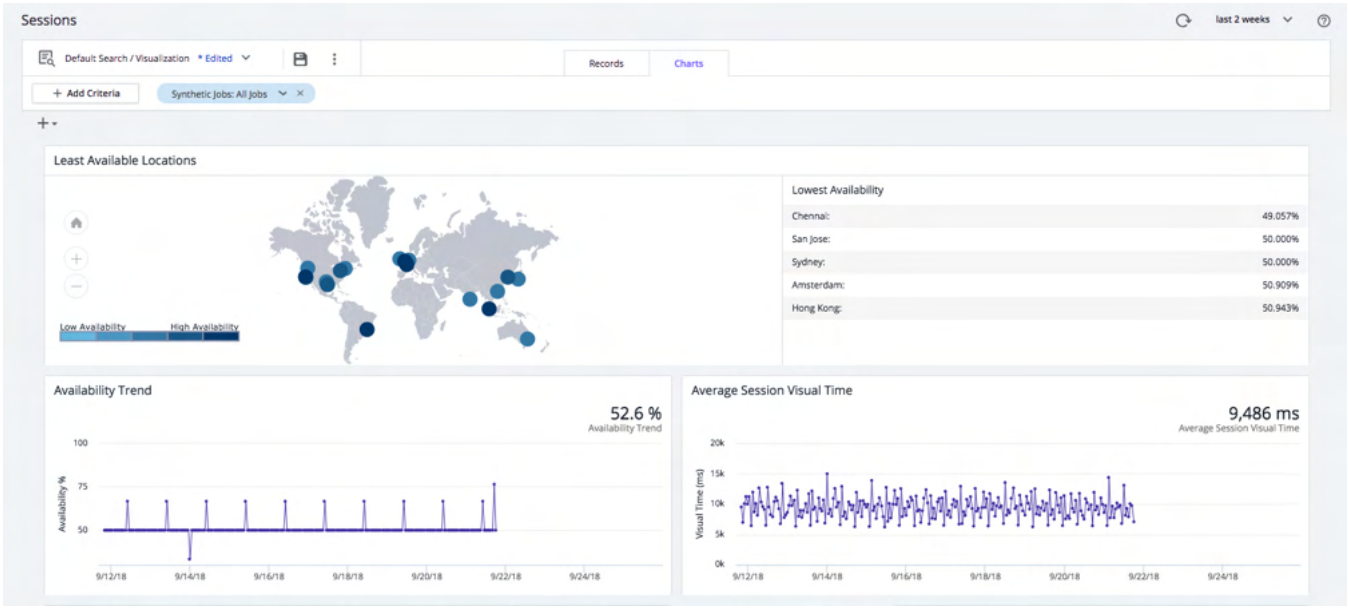
You can double-click any session to get a summary of the session, page details, and a breakdown of details for each page in the session. You'll be able to view how the page loaded, the resources that were requested, and the business transaction for each page. You can use this information to find where network latency occurs or where scripts break.

For example, from the **Session Details** dialog, you can determine that the visually complete (when the user would see the completely loaded page) took around 16 seconds and then examine the duration of the page load from the HTTP request to the first byte, from the first byte to DOM ready, and finally from DOM ready to visually complete.



Get a Quick Overview

From the **Sessions** page, you can click the Charts tab to view widgets representing aggregated session data. For example, the **Lowest Availability** widget shows you what locations are the least available, so you can confine and focus on issues in specific locations.


Sessions
last 2 weeks
Default Search / Visualization * Edited
Records
Charts
+ Add Criteria
Synthetic Jobs: All Jobs
Least Available Locations
Low Availability
High Availability
Lowest Availability
Chennai: 49.057%
San Jose: 50.000%
Sydney: 50.000%
Amsterdam: 50.909%
Hong Kong: 50.943%
Availability Trend
52.6 %
Availability Trend
Availability %
100
75
50
9/12/18
9/14/18
9/16/18
9/18/18
9/20/18
9/22/18
9/24/18
Average Session Visual Time
9,486 ms
Average Session Visual Time
Visual Time (ms)
20k
15k
10k
5k
0k
9/12/18
9/14/18
9/16/18
9/18/18
9/20/18
9/22/18
9/24/18

Synthetic Jobs

Synthetic jobs are configurable, scheduled tests that mimic end-user actions in remote browsers in supported locations. Synthetic jobs consist of URL measurements or synthetic scripts. To create a job, see Get Started with Browser Synthetic Monitoring.

Support

The table below describes supported versions and libraries for Synthetic Jobs.

Support Type	Description
Browser Versions	• IE 11 • Chrome 81.0.4044.0 • Firefox 72
Python Versions	• Python 2 • Python 3
Python Libraries	• selenium • Requests • pysftp

Locations

For URL measurements and synthetic script jobs, the Synthetic Agent is always executed in isolation within a Docker container. This location is then mapped to the configured browser location. The synthetic script, however, is run on a different machine or location from the configured browser for security reasons. See Synthetic Agent Locations for more details.

Maturity Levels

When you create a synthetic job, it is added to a queue based on the creation timestamp. Scheduled jobs are assigned a maturity level that influences when they will be executed in the synthetic job queue. The table below describes the two supported maturity levels.

Maturity Level	Description
Junior	The Junior maturity level consists of synthetic jobs that are created or updated within two hours. Once two hours have passed since the last update or creation of the job, the Junior job maturity level is promoted to Senior maturity level. The promotion from Junior to Senior maturity is only performed only if enough capacity has been allocated. This avoids negatively affecting jobs which are currently executing.
Senior	The Senior maturity level consists of synthetic jobs that have been created or updated more than two hours ago. Synthetic jobs with Senior maturity level have ample allocated resources to be executed.

Execution Order

The execution order of synthetic jobs depends on the job priority and the job creation time. Synthetic jobs are only executed if they are in the job queue. Job priority is based on several factors: job type, job maturity, and your Browser Synthetic Monitoring license. The table below shows how the job priority is calculated. The actual job execution order depends on the job priority and job creation timestamp.

Synthetic jobs are only executed if they are in the job queue. If the max job queue size has been reached, however, no additional synthetic jobs will be added to the queue until the queue size decreases. The max queue size is defined by the priority, license, type, and maturity level.

Queue	Job Type	Job Seniority
High-priority	Scheduled	Senior
	On-demand	N/A
Low-priority	Scheduled	Junior

Execution Cadence

The synthetic job scheduler executes jobs according to the following cron-based expression:

```
<second> <minute> <hour> <day-of-month> <month> <day-of-week> <year>
```

When you schedule a job, you choose how frequently the job runs. It is recommended you choose a larger unit of time (hours or days) that can evenly divide into the smaller unit of time (minutes or hours). For example, if a job is scheduled to run every 15 minutes, the job runs four times within the hour. This results in the job consistently repeating every hour without interruption to the schedule.

However, if you schedule a job such that a larger unit of time that does not divide evenly into the smaller unit of time, such as running every 50 minutes, then the job does not run at consistent times. For example, if the first job starts at 7:03, then the job will run at 7:53, 8:03, 8:53, and so on, resulting in a disruption to the schedule.

Execution Errors

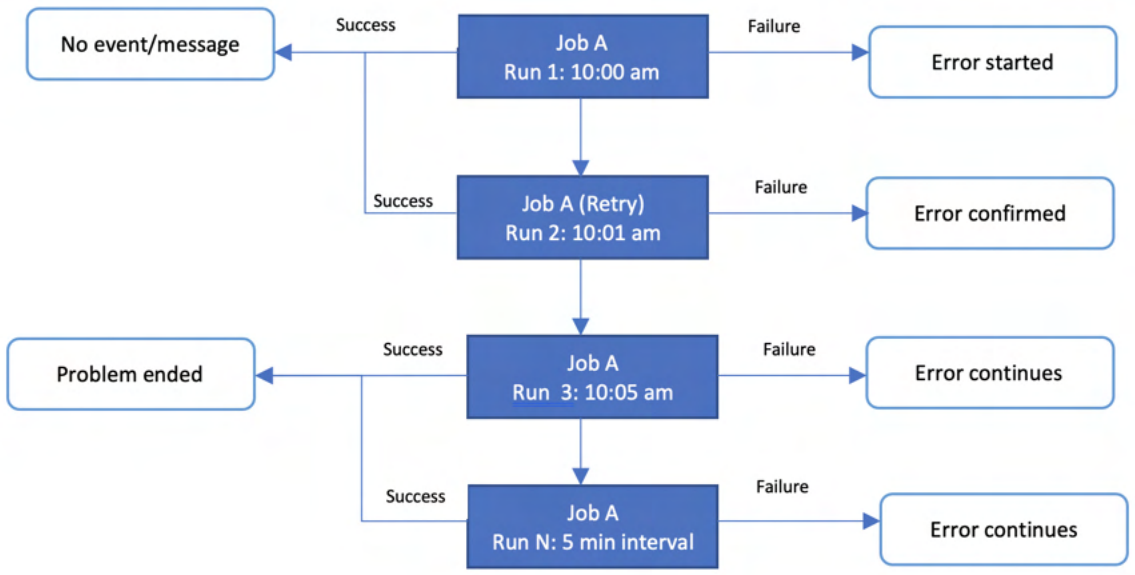
The following table provides job execution error messages, the cause of the error, and the associated error code.

Error Message	Cause of Error	Error Code
Skipped; still waiting for a previous job execution	An attempt was made to queue a high-priority measurement request from a scheduled job before the previous measurement request from the same job, and location-browser combination has been processed.	`TARDY`
Skipped while new capacity is being added	An attempt was made to queue a low-priority measurement request from a scheduled job before the previous measurement request from the same job, and location-browser combination has been processed.	`TARDY_ONBOARDING`
Skipped while new capacity is being added	An attempt was made to queue a measurement request from a junior job beyond the maximum respective queue capacity.	`ONBOARDING`
Testing location is overloaded	An attempt was made to request measurement from a senior job beyond the maximum respective queue capacity.	`THROTTLED`

Health Rule Violation Events

You can configure health rule violation events, such triggering an event when a job fails. To set health rule violation events, create an alerting policy for synthetic jobs.

The diagram below shows the retest process when a job fails, and what events are triggered. In this example, a job is scheduled to run at 10:00 am. If successful, no event or message is triggered; if it fails, "Error started" is triggered and the job retests. If successful, no even or message is triggered; if it fails, "Error started" is triggered and the job retests. If successful, "Problem ended" is triggered; if it fails, "Problem ended" is triggered and the job retests. As long as the job continues to fail, the job will retest every 5 minutes and continue triggering "Error continues."



Configure Synthetic Jobs

Related Pages:

- Get Started with Browser Synthetic Monitoring
- Alerts for Browser Synthetic Monitoring

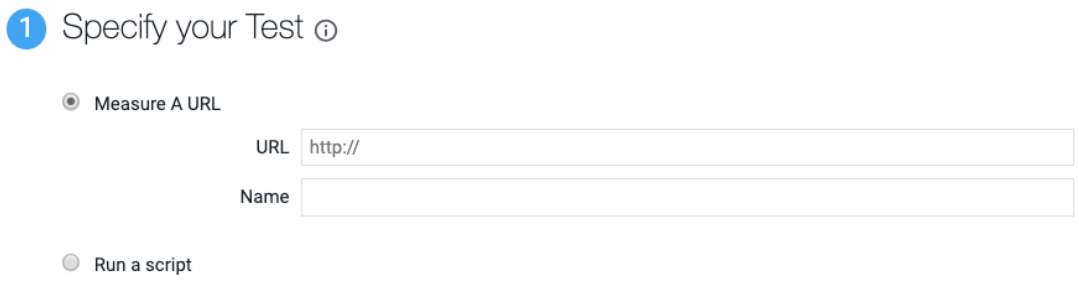
This page describes how to use the Controller UI to configure a synthetic job. You can schedule when, where, and how often a job runs, set timeouts, configure performance thresholds to trigger warning and critical events, and customize connection speeds. Before configuring a job, see Getting Started with Browser Synthetic Monitoring.

Specify your Test

Synthetic Jobs use either a specific URL or webdriver script to test one or more web pages. If your site requires authentication, you must create a script which performs the authentication to log onto your site. See Synthetic Scripts for more information.

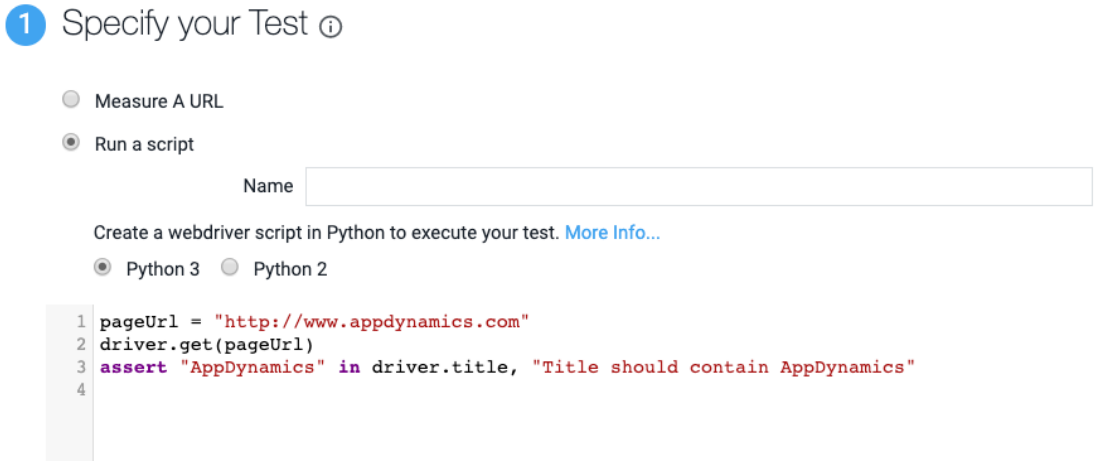
To use a URL:

1. Select "Measure a URL".
2. In the URL field, enter a site or page URL.
3. In the Name field, enter a name for the job. This name appears in the Jobs list in the Controller UI.



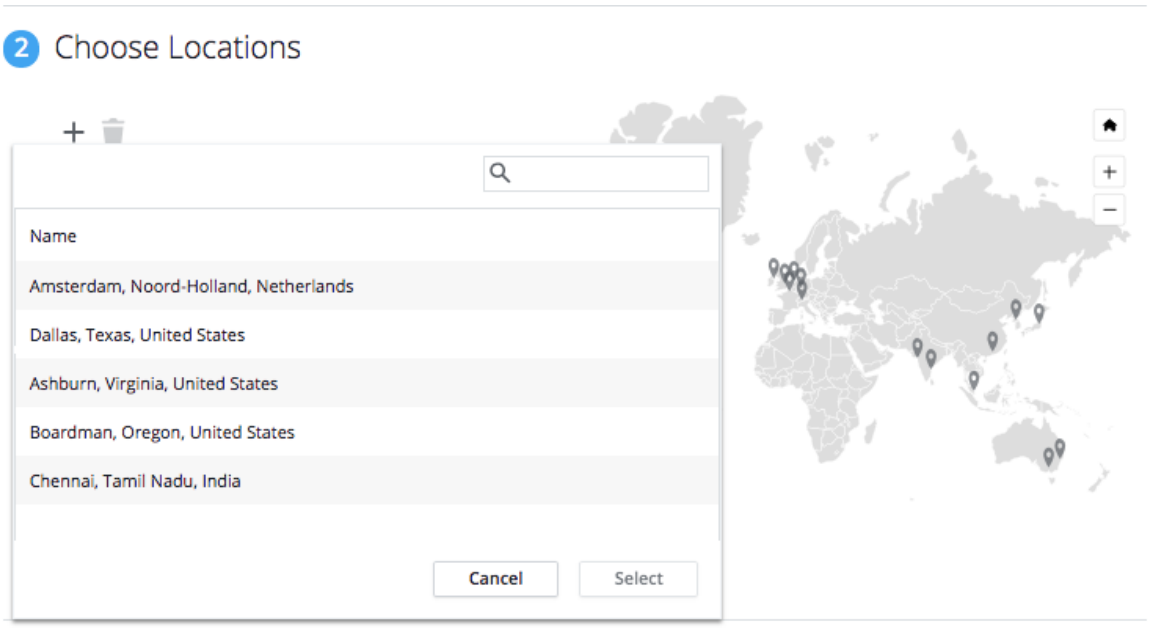
To use a webdriver script:

1. Select "Run a Script".
2. Select Python 3 or Python 2.



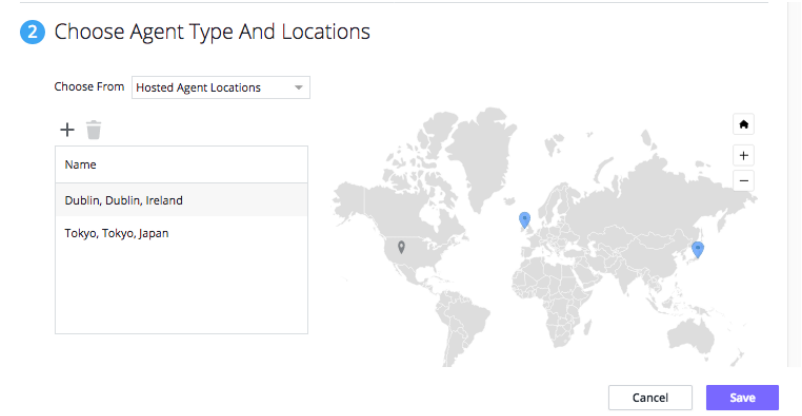
Choose Locations

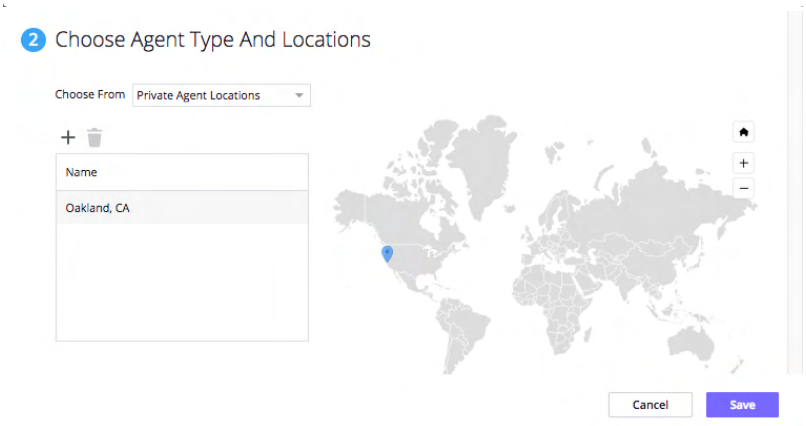
You can choose one or more locations in which the job runs. This can also be configured to test all of the locations every time or one location each time the job runs. For a complete list of Synthetic Agent locations, see Synthetic Agent Locations.



Hosted Agent Locations and Synthetic Private Locations

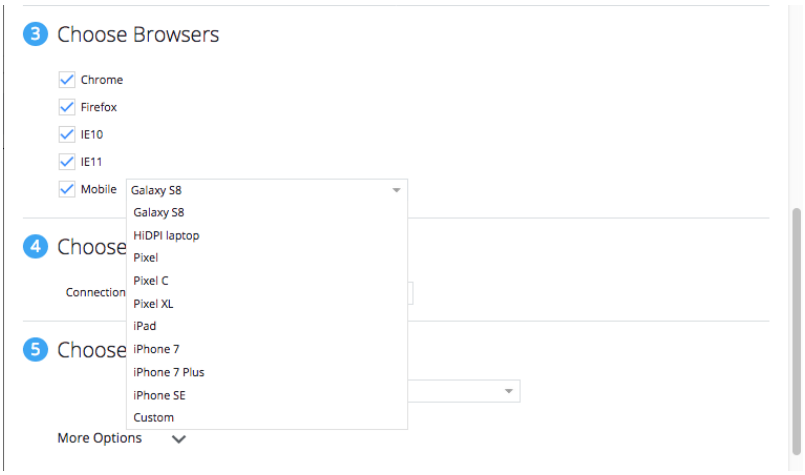
If you have deployed a Synthetic Private Agent Deployment, you can choose from **Hosted Agent Locations** or **Private Agent Locations**. The **Hosted Agent Locations** are where AppDynamics hosts public synthetic agents. The **Private Agent Locations** are where you are hosting your synthetic private agents.





Choose Browsers

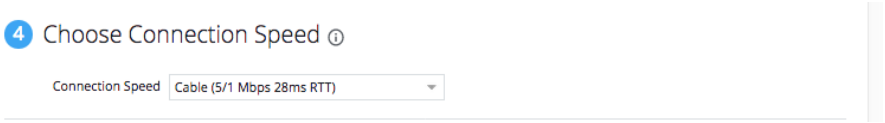
A job can run on one or more different types of browsers. For mobile browsers, you can specify a platform, such as the iPhone 7 Plus or Pixel. When a job runs on mobile browsers, it is actually running a Chrome browser emulator with the specified platform properties.



All the browsers are running on either Windows 2012 or Windows 2012 R2.

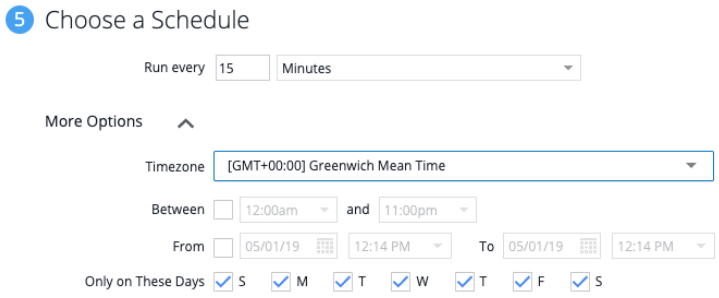
Customize Connection Speeds

The connection speed helps keep performance consistent and realistic. If you choose **Native Connection**, the job will run at the maximum speed available to AppDynamics data centers. The default is **Cable (5/1 Mbps 28ms RTT)**.



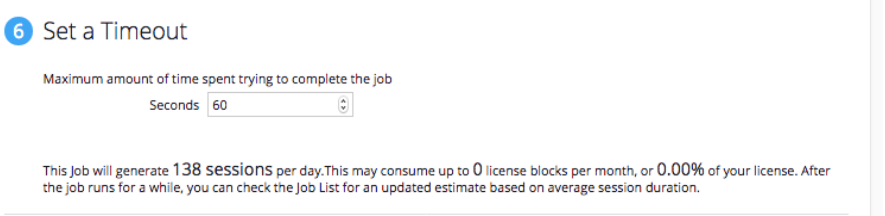
Schedule Jobs

When you create or edit your job, you can define the job schedule. The job will run within the specified timezone. When you choose a timezone, the job schedule will not be affected by time changes in any other timezone, such as daylight saving time. The default timezone is GMT.



Set Timeouts

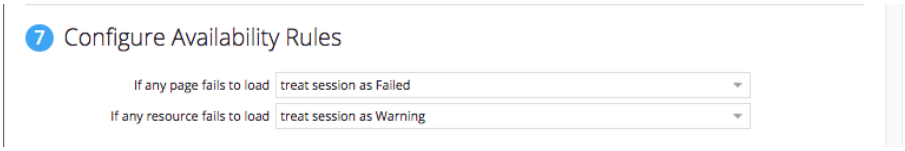
Synthetic jobs consume licenses based on the duration of the job, so you can set a timeout to help limit the license consumption. If the job times out, data will still be collected until the timeout expires. The default suggestion for job timeout is 15 seconds for a URL and 30 seconds for a script. You can adjust the timeout as preferred, but keep in mind that short timeouts might result in unsuccessful sessions, and long timeouts might result in overconsumption of licenses.



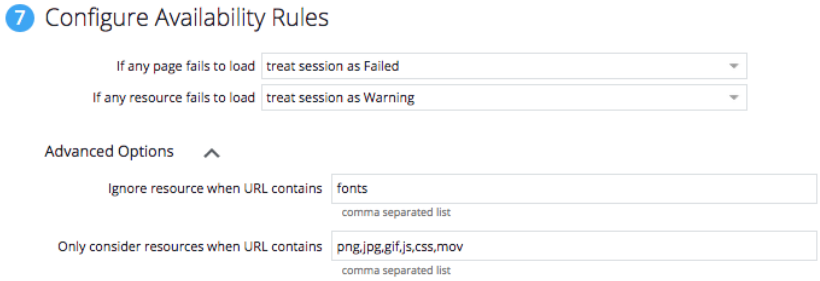
Configure Availability Rules

You can configure the synthetic job to check the availability of pages and resources. For example, you can set the session status for when a page fails to load or when a resource is missing or inaccessible.

The screenshot below is a configuration to treat the session as **Failed** if any page fails to load and to treat the session as **Warning** if any resource fails to load. If the job navigates to a page with an HTTP 4xx or 5xx status code, the session status will be set to `FAILED`. This will trigger a Critical Event. If a resource fails to load, the session status will be set to `WARNING`, which will trigger a Warning Event. To set up alerts, see Alerts for Browser Synthetic Monitoring.



Use the **Advanced Options** to ignore or only consider selected resources. For example, the screenshot below might be used to ignore the font resource (which browsers may not have) and only consider high-impact resources like CSS, JavaScript, etc.

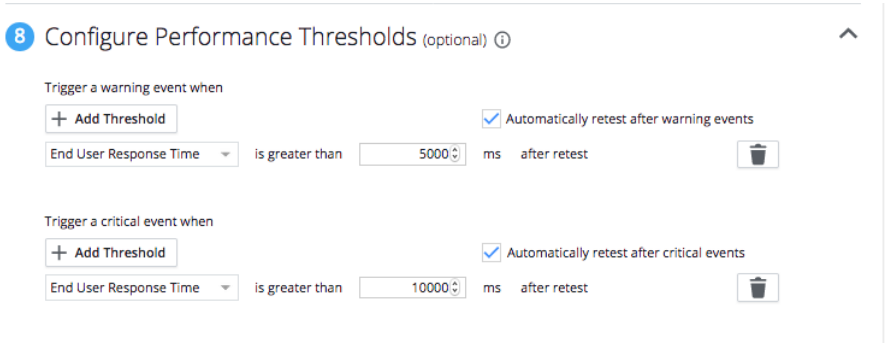


> Favicons are always ignored when checking the availability of resources.

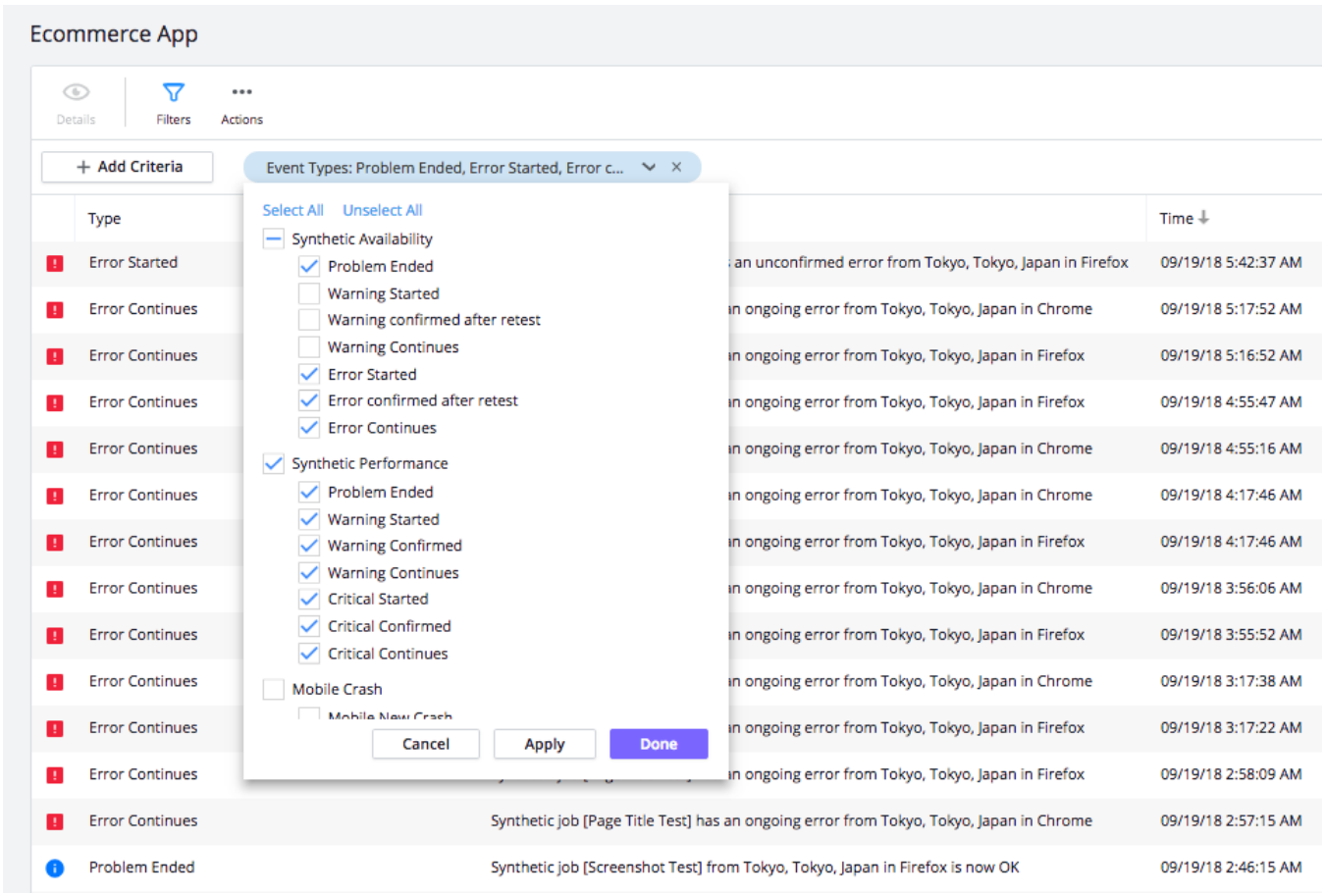
Configure Performance Thresholds

You can specify the job performance thresholds to send alerting events. For an example of the health rule violation process, see Health Rule Violation Events.

If you check one or both of the "**Automatically retest after warning events"** or "**Automatically retest after critical events"** boxes, and the event is generated, then the job will be triggered to re-execute immediately per unconfirmed warning or critical event. For example, if the **Critical Events** threshold is exceeded, the event **Critical Started** will be triggered and the job will rerun until the event **Critical Continues** is confirmed. Learn more about the synthetic job retest process.



From the **Events** page shown below, you can see that the results of the first and successive tests exceeded the threshold for synthetic availability and triggered the **Error Started**, **Error confirmed after restart**, **Error Continues**, and **Problem Ended** events.


Ecommerce App
Details
Filters
Actions
+ Add Criteria
Event Types: Problem Ended, Error Started, Error c...
Select All
Unselect All
Synthetic Availability
Problem Ended
Warning Started
Warning confirmed after retest
Warning Continues
Error Started
Error confirmed after retest
Error Continues
Synthetic Performance
Problem Ended
Warning Started
Warning Confirmed
Warning Continues
Critical Started
Critical Confirmed
Critical Continues
Mobile Crash
Cancel
Apply
Done
Type
Time
Error Started
an unconfirmed error from Tokyo, Tokyo, Japan in Firefox
09/19/18 5:42:37 AM
Error Continues
n ongoing error from Tokyo, Tokyo, Japan in Chrome
09/19/18 5:17:52 AM
Error Continues
n ongoing error from Tokyo, Tokyo, Japan in Firefox
09/19/18 5:16:52 AM
Error Continues
n ongoing error from Tokyo, Tokyo, Japan in Firefox
09/19/18 4:55:47 AM
Error Continues
n ongoing error from Tokyo, Tokyo, Japan in Chrome
09/19/18 4:55:16 AM
Error Continues
n ongoing error from Tokyo, Tokyo, Japan in Chrome
09/19/18 4:17:46 AM
Error Continues
n ongoing error from Tokyo, Tokyo, Japan in Firefox
09/19/18 4:17:46 AM
Error Continues
n ongoing error from Tokyo, Tokyo, Japan in Chrome
09/19/18 3:56:06 AM
Error Continues
n ongoing error from Tokyo, Tokyo, Japan in Firefox
09/19/18 3:55:52 AM
Error Continues
n ongoing error from Tokyo, Tokyo, Japan in Chrome
09/19/18 3:17:38 AM
Error Continues
n ongoing error from Tokyo, Tokyo, Japan in Firefox
09/19/18 3:17:22 AM
Error Continues
n ongoing error from Tokyo, Tokyo, Japan in Firefox
09/19/18 2:58:09 AM
Error Continues
Synthetic job [Page Title Test] has an ongoing error from Tokyo, Tokyo, Japan in Chrome
09/19/18 2:57:15 AM
Problem Ended
Synthetic job [Screenshot Test] from Tokyo, Tokyo, Japan in Firefox is now OK
09/19/18 2:46:15 AM

Synthetic Scripts

Browser Synthetic Monitoring allows you to write Python scripts using the Selenium WebDriver library to create scheduled tests that can mimic end-user actions in remote browsers in supported locations. For supported Python versions and libraries, see Synthetic Jobs.

You can use synthetic scripts to test performance, features, end-user flows, and the end-user experience.

For example, your synthetic script might do the following:

1. Visit your website.
2. Search for products.
3. Add items to a shopping cart.
4. Check out the shopping cart and assert that the total price of the cart is correct.
5. Enter payment details.
6. Complete the purchase.

Benefits of Using Synthetic Scripts

Using synthetic scripts, you can:

- find broken user flows
- improve the performance of your site by finding where response times in your web applications are slow
- test new features

How Scripts Are Run

The entire process can be simplified into the following steps:

1. You add your script to a synthetic job in the Controller UI.
2. The synthetic job is transmitted to the Synthetic servers, where it is scheduled to be run.
3. When it's time for your job to run, a container is created just for your script.
4. Your script runs in the container, controlling a web browser using Remote WebDriver.
5. After the script finishes running, the agent collects browser metrics and sends them to the Controller UI.

The Controller UI receives and displays the metrics from your synthetic job.

Get Help

In the Knowledge Base forum, read our Synthetic Scripts FAQ. If you can't find the answer to your question, submit your questions to the community.

Write Your First Script

Related pages:

- Locate DOM Elements
- Wait for DOM Elements
- Work with Screenshots
- Add Logs to Troubleshoot
- Verify Program Correctness

This page describe how to create a synthetic job and view the results.

Write a Script

Before you create a Synthetic Job with a custom script, you will need to write a Python WebDriver script and get it working locally. For a good introduction to scripting, AppDynamics recommends reading the Selenium Python Bindings documentation.

Before running your scripts in Synthetic Jobs, AppDynamics recommends reading the differences between Python and synthetic scripts as you may need to make some modifications.

Tools for Writing Scripts

For a full-fledged script recorder, use the Firefox add-on Katalon Recorder (Selenium IDE for Firefox 55+). The resulting script will work with Browser Synthetic, but as with most auto-generated code, expect to make manual changes to improve it.

Constraints for Writing Scripts

`driver.get` should always point to a proper HTML page and not to resources.

- `driver.get()` works well for HTML pages.
- Some of the session data might be incomplete or incorrect if you try to get resources directly.

Test Scripts Locally

Testing scripts locally is one of the best ways to take advantage of utilities to help you write scripts and then run them locally. To run scripts locally, complete the installation instructions given in the Selenium Python Bindings documentation and then run the script as you would any Python script.

Understand the Differences Between Python and Synthetic Scripts

When you are ready to write synthetic scripts, you should be aware that although Browser Synthetic can run unmodified Python scripts, there are few differences:

- The driver initialization code can be *removed* because the variable `driver` is automatically created and initialized with the browser selected in the **Job** creation UI. Thus, the example code below that initializes a driver would be replaced by our logic that then returns the variable `driver`.

```
# This initialization code or any initialization code
# will be replaced with Synthetic Monitoring logic, and
# the singleton `driver` will be available.
from selenium import webdriver

driver = webdriver.Firefox()
```

If your script initializes a driver, it is okay to keep it: Synthetic will just ignore your initialized driver and instead return the driver it automatically created.

- Only one driver can be run within a synthetic session. Any attempt at creating new drivers will return the automatically instantiated driver (a singleton).
- Scripts interacting with multiple windows will work; however, the session results will merge the results for all windows into one waterfall.

Create a Synthetic Job with Your Script

Once you have completed and tested your scripts, the next step is to create a Synthetic Job with your script:

1. From the **New Job** dialog, check the **Run a script** radio button.
2. Enter **My First Script** in the **Name** field.
3. Take a look at the sample WebDriver script, but do not change it.
4. For **Choose Browsers**, select **Chrome** and **Firefox.**
5. For **Choose Locations**, add a couple of different locations.
6. Click **Save**.

View Results

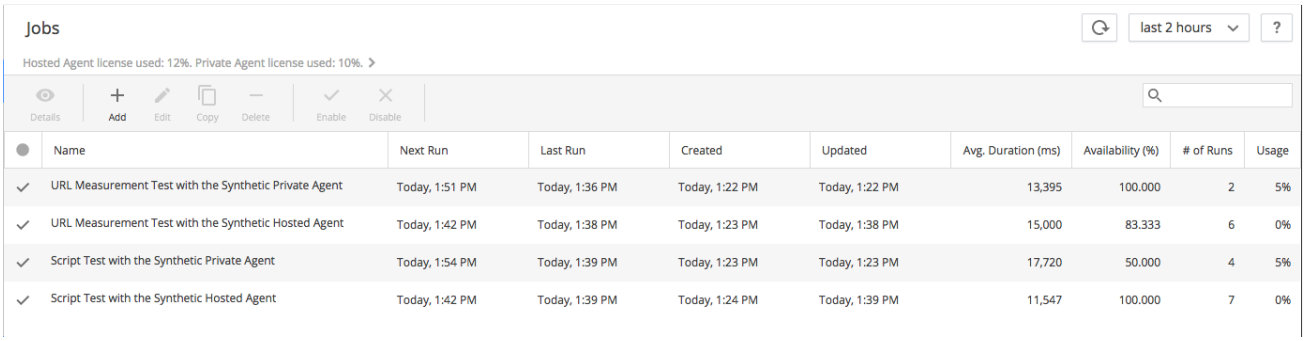
All date and time functions in the synthetic script return time in the UTC time zone. For example, if a script is scheduled on July 30 at 12am PST from the San Francisco location, and in the script it calls the following function:

```
import datetime
.....
dt = datetime.datetime.now()
print(str(dt))
```

It will return the time as:

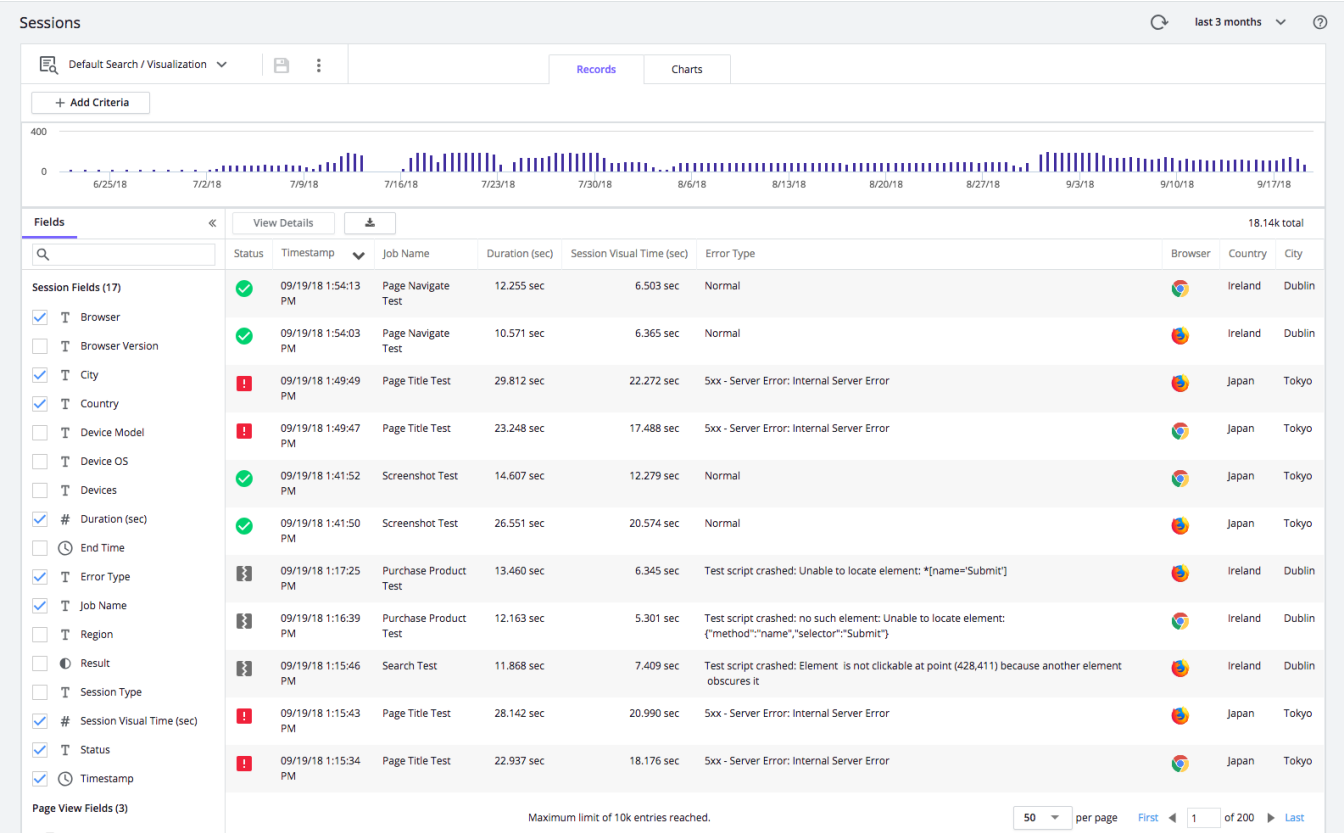
```
2020-07-30 07:00:00.000000
```

You have two types of views for the results. From the **Jobs** page, where you create a job, you can see your job and a summary of the results after the job has been run for both Synthetic Hosted/Private Agents.

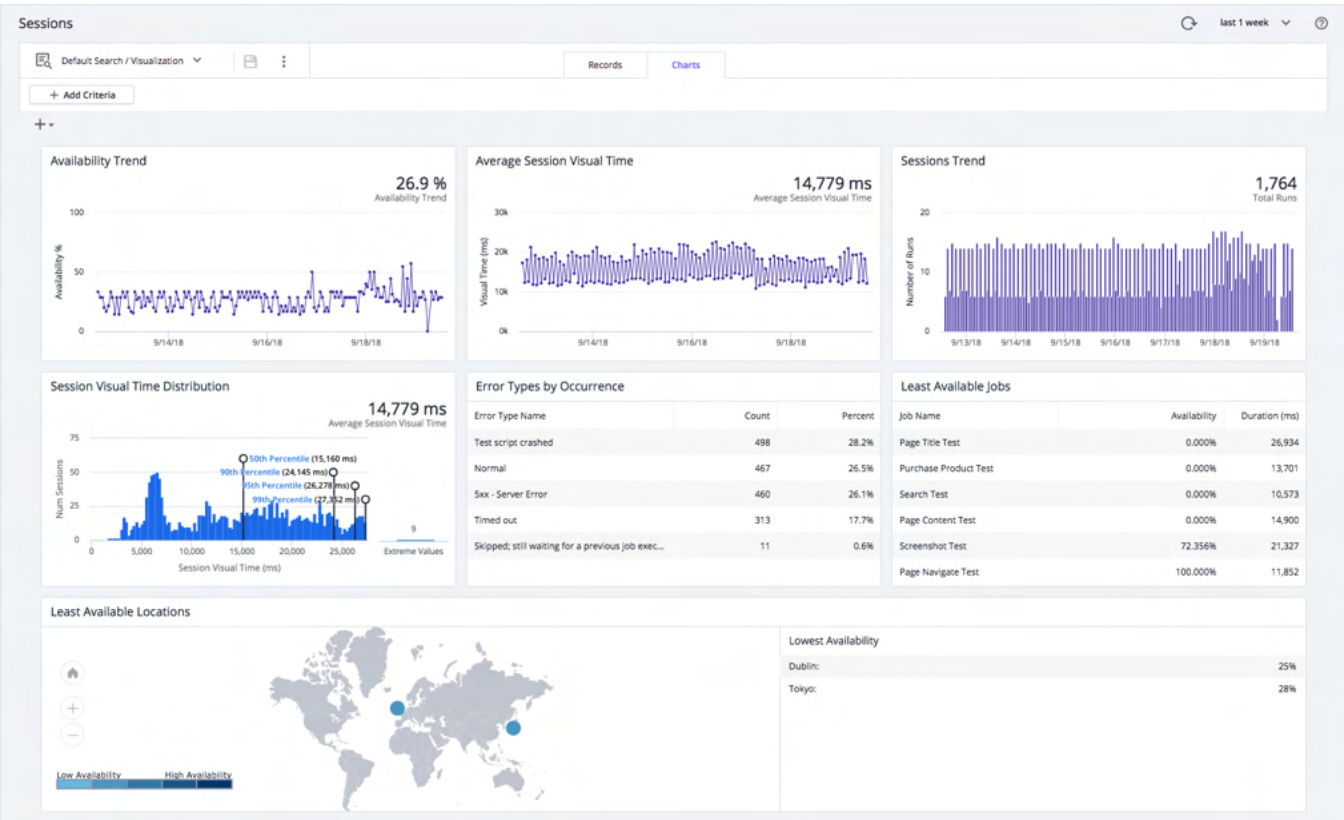


Name	Next Run	Last Run	Created	Updated	Avg. Duration (ms)	Availability (%)	# of Runs	Usage
URL Measurement Test with the Synthetic Private Agent	Today, 1:51 PM	Today, 1:36 PM	Today, 1:22 PM	Today, 1:22 PM	13,395	100.000	2	5%
URL Measurement Test with the Synthetic Hosted Agent	Today, 1:42 PM	Today, 1:38 PM	Today, 1:23 PM	Today, 1:38 PM	15,000	83.333	6	0%
Script Test with the Synthetic Private Agent	Today, 1:54 PM	Today, 1:39 PM	Today, 1:23 PM	Today, 1:23 PM	17,720	50.000	4	5%
Script Test with the Synthetic Hosted Agent	Today, 1:42 PM	Today, 1:39 PM	Today, 1:24 PM	Today, 1:39 PM	11,547	100.000	7	0%

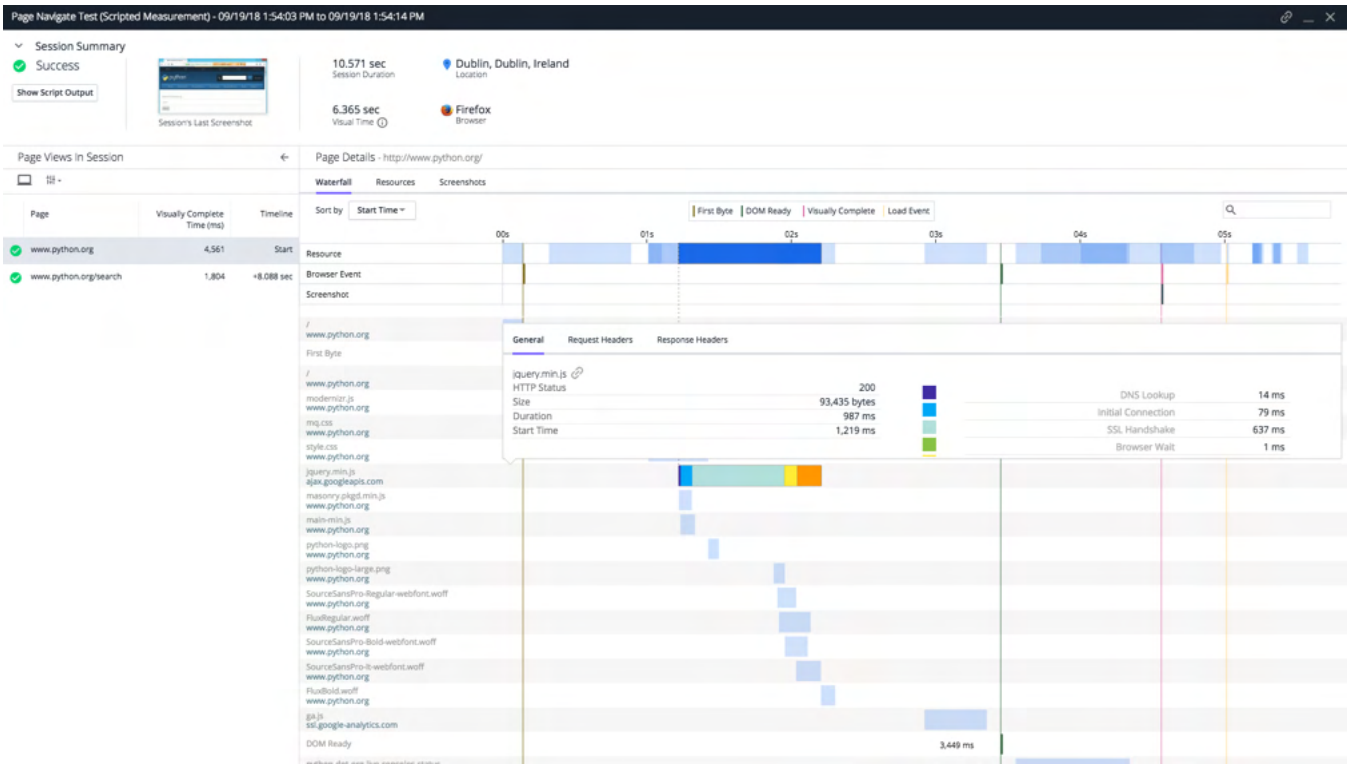
You can also see more detailed results from the **Sessions** page. You can get there by either double-clicking your job or by clicking **Sessions** in the left navigation bar.



To see the metrics as charts, click the Charts tab:



From the Records tab, you can double-click a session to view the **Session Details** dialog:


Page Navigate Test (Scripted Measurement) - 09/19/18 1:54:03 PM to 09/19/18 1:54:14 PM
Session Summary
Success
Show Script Output
Session's Last Screenshot
10.571 sec
Session Duration
Dublin, Dublin, Ireland
Location
6.365 sec
Visual Time
Firefox
Browser
Page Views In Session
Page
Visually Complete Time (ms)
Timeline
www.python.org
4,561
Start
www.python.org/search
1,804
+8.088 sec
Page Details - http://www.python.org/
Waterfall
Resources
Screenshots
Sort by
Start Time
First Byte
DOM Ready
Visually Complete
Load Event
00s
01s
02s
03s
04s
05s
Resource
Browser Event
Screenshot
General
Request Headers
Response Headers
jquery.min.js
HTTP Status
200
Size
93,435 bytes
Duration
987 ms
Start Time
1,219 ms
DNS Lookup
14 ms
Initial Connection
79 ms
SSL Handshake
637 ms
Browser Wait
1 ms
DOM Ready
3,449 ms

Locate DOM Elements

Related pages:

- Write Your First Script
- Wait for DOM Elements
- Work with Screenshots
- Add Logs to Troubleshoot
- Verify Program Correctness

Your synthetic script simulates end-user interactions, so you'll need to be able to locate the DOM elements of your web pages. For example, your script might click on buttons, links, or enter text into text fields: this requires your script being able to locate and then select relevant HTML elements.

Methods for Locating Elements

The Selenium WebDriver library provides CSS selectors and XPath statements for selecting HTML elements. See 4. Locating Elements for the list of the library methods and usage examples.

Best Practices for Locating Elements

When locating elements, you are recommended to do the following. Think of it as a checklist.

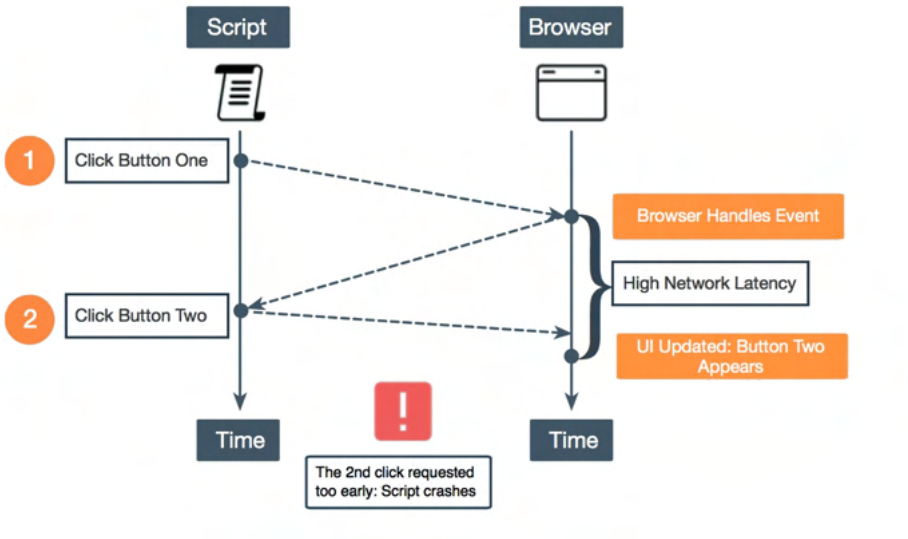
- Understand the following about your application:
    - the DOM structure
    - which pages are dynamically and statically loaded
    - which elements of a page are loaded and visible.
- Use unique IDs for elements and selectors that are as short as possible: Selectors break all the time. Long hierarchical selectors break more easily than shorter ones, and using short selectors will reduce script maintenance over time.
- Click on the user-visible element instead, or send a **Return** key instead of submitting forms using the method `submit()`.
- Pay attention to element visibility. The specification is complex, and the results are not always what users expect. See the WebDriver specification for more information.

Wait for DOM Elements

Related pages:

- Write Your First Script
- Locate DOM Elements
- Work with Screenshots
- Add Logs to Troubleshoot
- Verify Program Correctness

Your script might be accessing a UI component that is not available in the DOM. The cause could be network latency, as shown in the diagram below, or the DOM simply isn't ready to access.

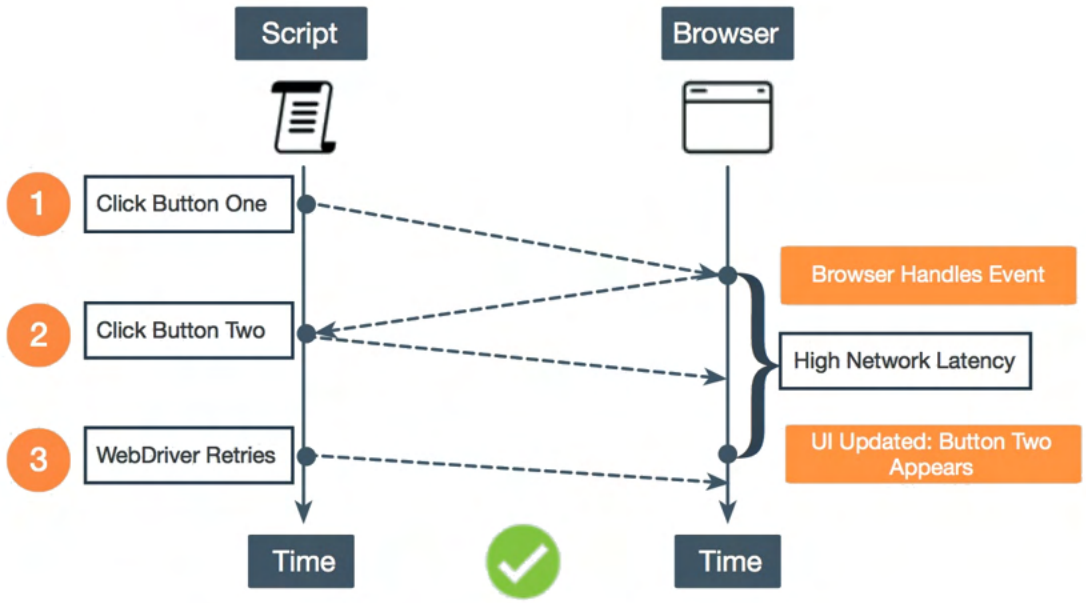


You can avoid this issue by using either an explicit or implicit wait. The following section discusses the different types of waits and when to use each.

Explicit Versus Implicit

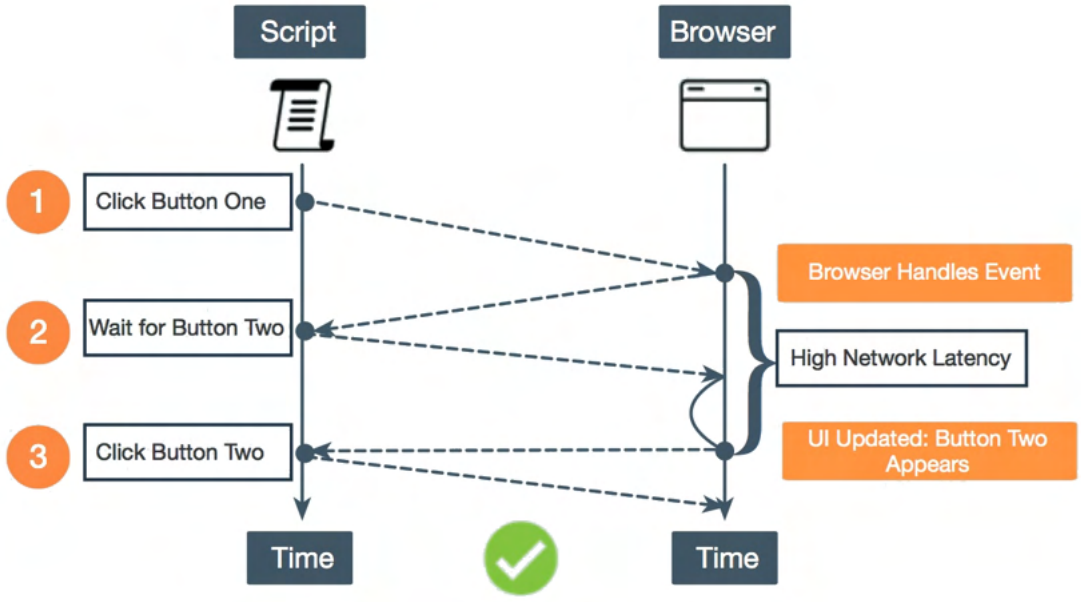
The WebDriver function `driver.implicitly_wait(n)` configures WebDriver for the remainder of a session. If you try to access an element that is not available, WebDriver will retry for up to n seconds to find that element in the DOM. This simple approach is often sufficient to make your scripts work reliably.

In the diagram below, the `WebDriver` retries to click **Button 2** until it's available.



An explicit wait is code that requires a certain condition to occur before continuing to execute code. Although you may be tempted to use `time.sleep()` to wait for an element to be available, you should avoid doing this. Instead, use the WebDriver API methods given in 5.1. Explicit Waits of the Selenium Python Bindings documentation that enable you to wait for *expected conditions*.

In the diagram below, the explicit wait stipulates to click **Button 2** (action) only when the button is available (the condition).



Work with Screenshots

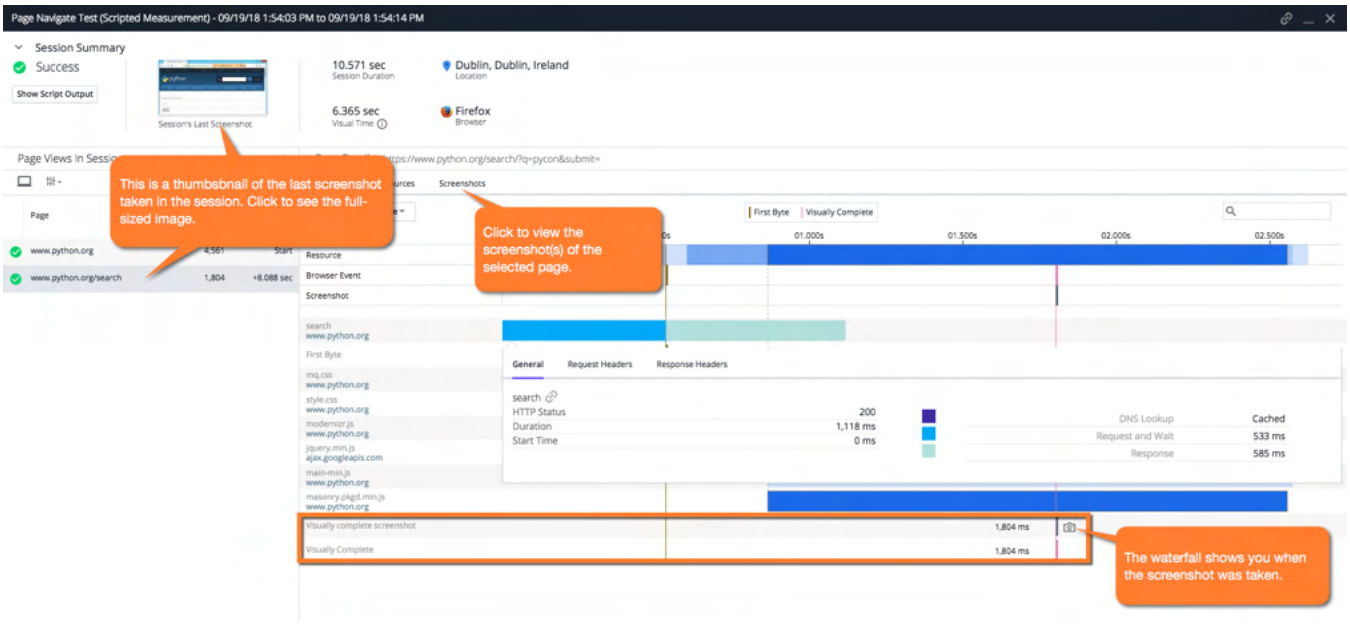
Related pages:

- Write Your First Script
- Locate DOM Elements
- Wait for DOM Elements
- Add Logs to Troubleshoot
- Verify Program Correctness

Synthetic jobs will automatically take screenshots for you, but you might want to manually take screenshots to debug issues.

Where to Find Screenshots

The **Session Details** dialog displays screenshots for each page, the last screenshot taken in the session, and a screenshot icon in the waterfall showing when the screenshot was taken. You can also view the larger version of the page screenshot from the Screenshots tab.



When to Use Screenshots to Debug

If interactions with a page are resulting in failures, you can use a screenshot to detect whether the UI component is on the page. For example, a screenshot can be used to determine if you are trying to click a button that isn't on the page or that is being overlapped by another HTML component. You may also want to manually take a screenshot if your job is timing out to diagnose an issue with the page because screenshots are not be automatically taken for timed out jobs.

How to Take Manual Screenshots

Although screenshots are taken by default, you can manually take screenshots at any point during the execution of your script using either of the following:

```
driver.get_screenshot_as_file("homepage.png")
driver.save_screenshot("myscreenshot.png")
```

Add Logs to Troubleshoot

Related pages:

- Write Your First Script
- Locate DOM Elements
- Wait for DOM Elements
- Work with Screenshots
- Verify Program Correctness

You can use logs to track the progress of your scripts and to better understand what is happening while the script is executing.

Print Statements

The simplest way to log messages is with print statements. You can use them to inspect objects or to determine where an issue is located.

For example, you can determine whether an element was selected on a page by displaying the contents of the object:

```
elem = driver.find_element_by_id("wiki-content")
print(elem)
```

If the script results are incomplete, you can use a series of print statements to pinpoint where the problem might have occurred:

```
print("Getting the web page 'appdynamics.com'.")
driver.get("http://docs.appdynamics.com")

print("Getting the content for the container with the ID 'wiki-content'.")
elem = driver.find_element_by_id("wiki-content")

element = driver.find_element_by_xpath("//select[@name='name']")
all_options = element.find_elements_by_tag_name("option")
for option in all_options:
    print("Value is: %s" % option.get_attribute("value"))
    option.click()
```

Logging Package

You can import the `logging` package to log different types of error messages.

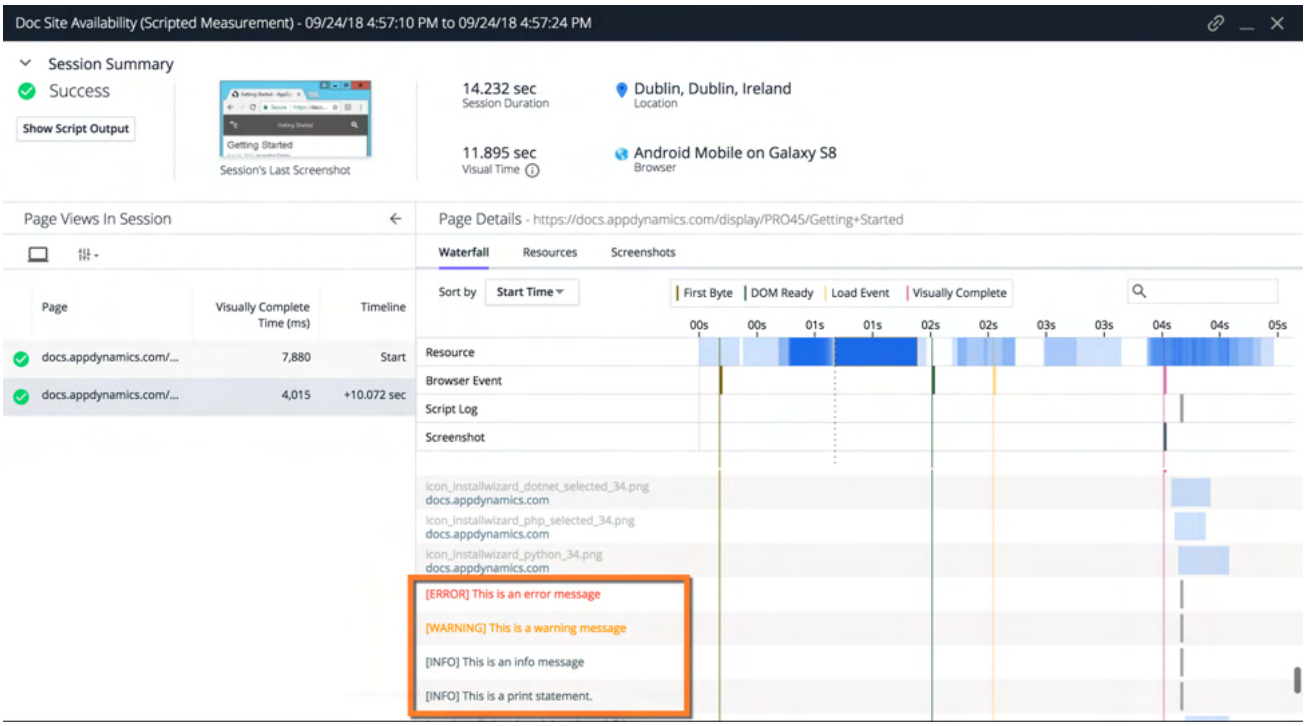
To import the `logging` package:

```
import logging
```

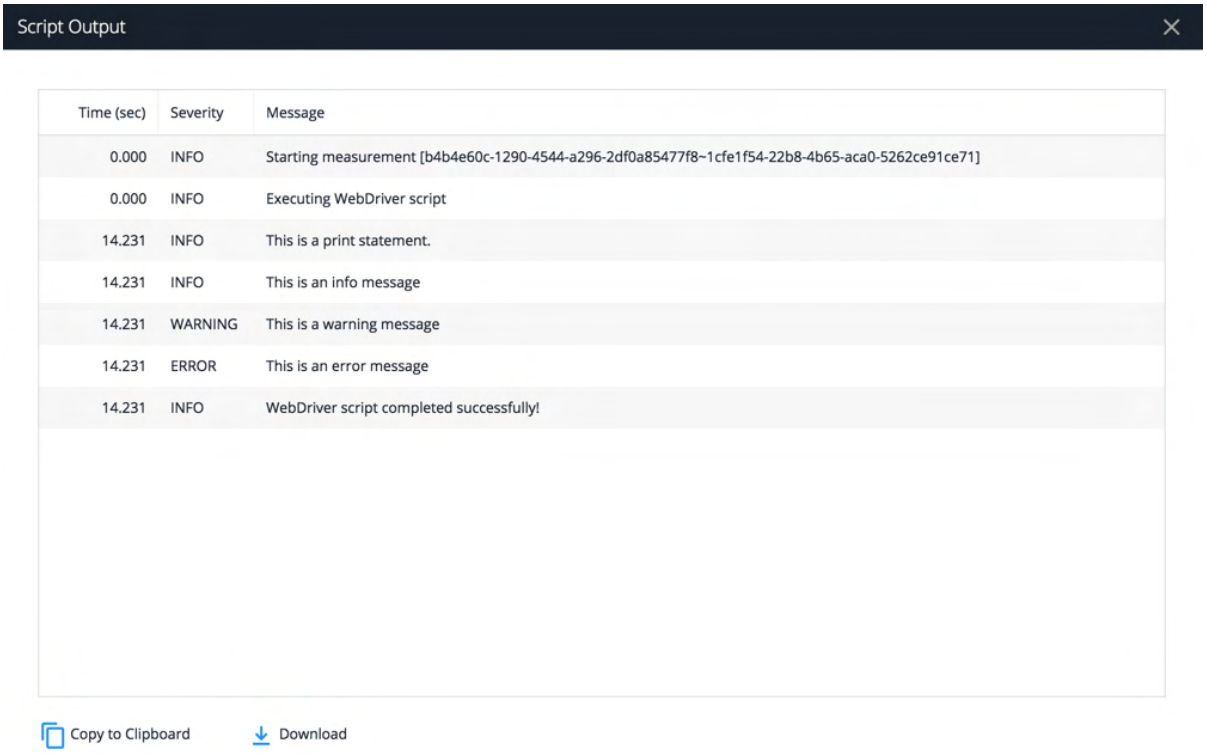
Once you have imported the `logging` package, you can generate different types of logs:

```
print("This is a print statement.")
logging.info("This is an info message")
logging.warn("This is a warning message")
logging.debug("This is a debug message")
logging.error("This is an error message")
```

The different types of errors are then color-coded in the waterfall of the **Session Details** dialog.



The script output will label the different log messages by the severity as shown below.



Import Python Packages in Synthetic Scripts

Synthetic scripts are executed in short-lived containers where you can use pre-installed Python packages. Your script is restricted to the container and the container's lifespan.

Pre-installed Libraries

The following modules are installed in containers and can be accessed by importing them in your synthetic scripts.

- os
- selenium
- Requests
- pysftp

- ‘os’, ‘socket’, and, ‘subprocess’ modules will be disallowed for any new job.
- All existing jobs will continue to run.

Environmental Variables

With the `os` module available, you can access the environment variables through `os.environ`.

For example, to display the variables, you can use a script similar to the following:

```
import os
for i, j in os.environ.items():
        print(i, j)
```

Upload and Download Files with SFTP

One of the packages installed on these containers is `pysftp`, which you can use to upload and download files with SFTP.

The following code sample shows the basic functionality of the package.

```
import pysftp
cnopts = pysftp.CnOpts()
cnopts.hostkeys = None
# The <hostname> can be in either of the following formats: ftp.domain.ca || http://ftp.domain.ca
with pysftp.Connection('<hostname>', username='xxxx', password='xxxxxx', cnopts=cnopts) as sftp:
        # Change to the dictory 'public'
        with sftp.cd('public'):
            # Fetch the remote file
            sftp.get('<remote_file_name>')
            # Upload the same file to a remote location
                      sftp.put('<local_file_name>')
```

Make HTTP Requests

You can use the HTTP library `requests` to make HTTP requests within your synthetic scripts. The following example makes a GET request to the public GitHub Events API. See Requests: HTTP for Humans™ to learn more about the library's functionality and usage.

```
import requests

r = requests.get('https://api.github.com/events?per_page=1')
print("Status Code: %s\n" %(r.status_code))
print ("Headers: \n%s\n" %(r.headers))
print("Response: \n%s" %(r.text))
```

Verify Program Correctness

You can use assertion statements to determine that your script is receiving the expected results. You use the session status to determine session status whether your jobs have failed because of an uncaught exception or an assertion statement failed.

This page provides an assertion example that shows how to use the session status to determine which scripts have issues.

Use Assertions to Verify Expected Results

Assert statements use a simple syntax to verify an expected result. You can also provide a message to display if the assertion fails. In the example below, the message "Title should contain AppDynamics" will be displayed if the assertion fails.

```
# Get page and check the title
driver.get("http://https://ecommerce.com/view.html/ref=nav_cart")
assert "E-Commerce Shopping Cart" in driver.title, "Title should contain E-Commerce Shopping Cart"

# Click link and confirm URL is correct
driver.find_element_by_link_text("Place your order").click()
assert "ThankYouCart" in driver.current_url
```

You can also verify that a string is not in the result or the truth of a mathematical expression. Assert statements that are false will result in a session status of `FAILED`.

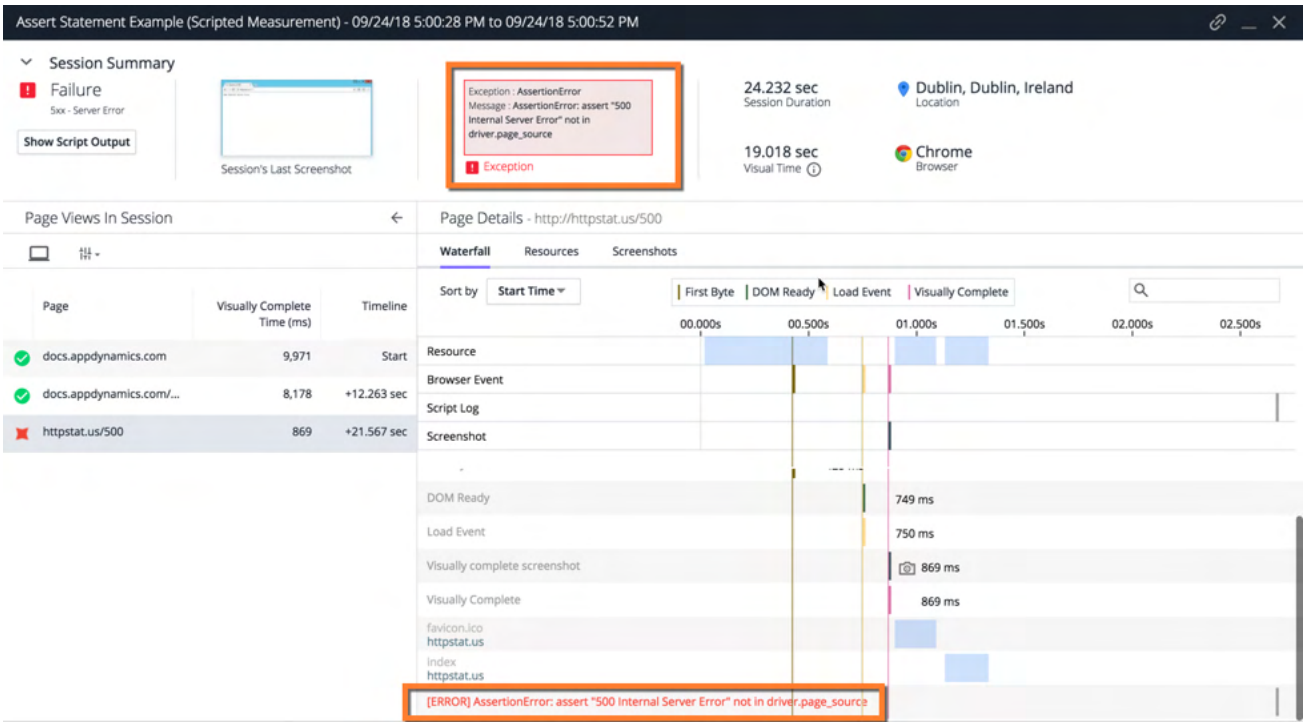
```
 # Confirm the page doesn't gives a 500 error.
assert "500 Internal Server Error" not in driver.page_source
# Confirm the footer is on the page.
footer = driver.find_elements_by_class_name('footer-body')
assert len(footer) > 0, "Footer couldn't be found."
```

Use the Session Status to Find Script Issues

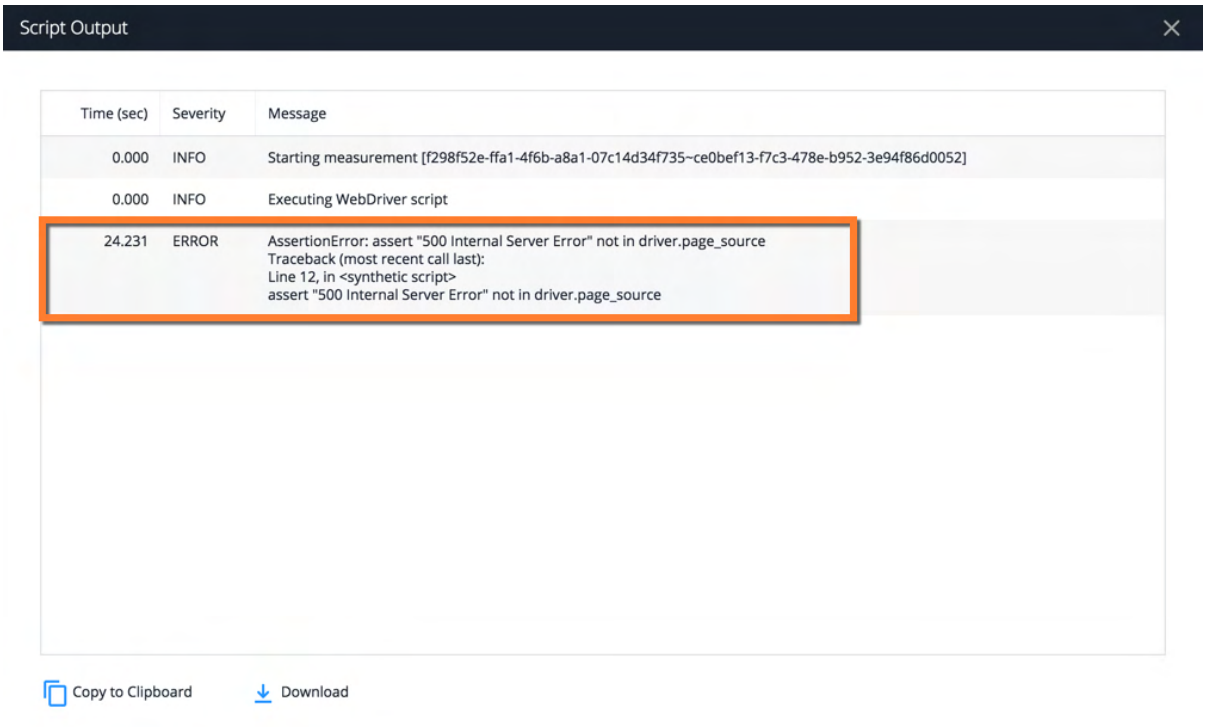
The two-session statuses `BROKEN` and `FAILED` indicate that the script had an issue. If the session status is `BROKEN`, it means that your script threw an uncaught exception. You'll most likely need to review your code and look for errors. See Exceptions for a list of common exceptions.

If the session state is `FAILED`, an assert statement failed or if you checked "Fail on page load error" and the page failed to load. You'll need to determine why your assertion statement failed or why the page couldn't be loaded.

You can view the results of assertion statements in the session status detail and script output. From the **Session Details** page, you can view failed assert statements in a red exception box at the top and from the Waterfall tab:



Failed assert statements will also appear in the **Script Output** dialog:



Select Client Certificates

This page is solely intended for use with the Synthetic Private Agent. You will *not* be able to run some of the synthetic scripts below with the Synthetic Hosted Agent.

When using client certificates, you may need to select the certificate from a dialog. To do this, you will need to install the Python pyautogui library on the host machine of your Synthetic Private Agent. The `pyautogui` library enables you to navigate the UI to select a client certificate.

Install the PyAutoGUI Library

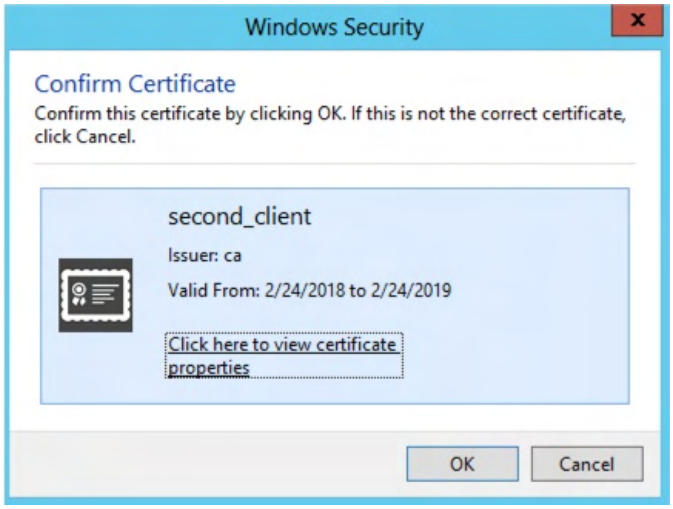
To install the PyAutoGUI Library:

1. Log in to the host machine for the Synthetic Private Agent as the `agent_user`.
2. Open a PowerShell console.
3. Use `pip` to install the library:

```
```

Select the Only Client Certificate

When you manually navigate to a site that uses client certification, IE10 may prompt you with an alert dialog to select one client certificates as seen below:

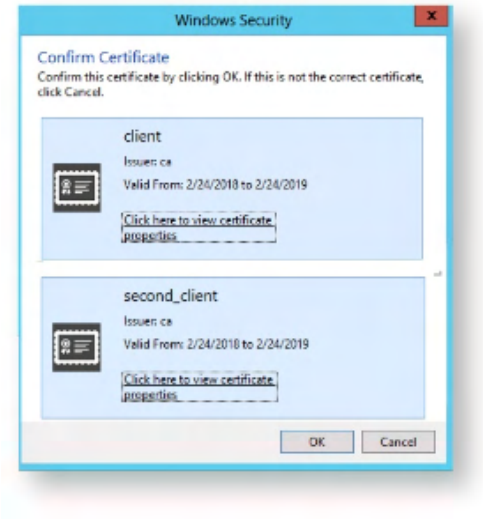


To select the client certificate from a synthetic script, your code could focus on the alert dialog and accept the given client certificate as performed by the synthetic script below.

```
sslUrl = "https://yourdomain.com/"
driver.get(sslUrl)
# Move focus to the dialog
alert = driver.switch_to_alert()
# Accept the client cert
alert.accept()
```

Select Certificate from List

In the case where your synthetic job needs to select one client certificate from a list presented in the alert dialog as shown below, you can use the PyAutoGUI Library to select and accept the client certificate that you want.



In the example below, you use `switch_to_alert` to focus on the alert dialog, and then use the `pyautogui` library to navigate to the correct certificate (tab), select it (enter), and then navigate to the OK button to select it (tab, enter).

```
import pyautogui
pyautogui.FAILSAFE = False
sslUrl = "https://yourdomain.com/"
driver.get(sslUrl)
driver.switch_to_alert()
pyautogui.press(['tab', 'enter', 'tab', 'enter'])
```

You may have many more client certificates to choose from, so you will have to figure out the best way to use the PyAutoGUI APIs to navigate around the alert dialogue to select and accept the correct client certificate. See Keyboard Control Functions in the PyAutoGUI documentation for more information.

Synthetic Agent Locations

The primary purpose of Browser Synthetic Monitoring is to perform browser-based testing. To help you test browsers in the locations of your users, Browser Synthetic provides agents in locations around the world. Your scripts, however, are run in a separate location for the security reasons discussed in the next section.

Security Considerations for Scripting

The browsers configured in synthetic jobs are run on shared Windows machines. Because synthetic scripts are written in native Python, they are not run on the shared Windows machines. Instead, each synthetic script is executed in isolation within a Docker container in a location mapped to the configured browser location. This isolation of the agents and scripts guarantees the safety of the data handled by the scripts and has no effect on the performance of the agent.

How This Affects Your Synthetic Jobs

You should understand how the locations of the agent/browser and where the script is executed may affect your synthetic job. For example, if the location of the execution container and the configured browser are not in the same region, you may expect to see some network latency. Also, if you are using the Python Requests library to make HTTP requests, you should know where the request is being made from.

The following sections provide some insight as to how your job may be affected.

URL Measurements

URL Measurements are used to test site availability and will not be affected because no script is being executed.

Synthetic Scripts

Your script can use different libraries to make HTTP requests. The WebDriver API is used to mimic the browser making HTTP requests. To make cURL-like requests to test REST APIs or to retrieve data, you can use an HTTP library such as the Requests library. The tables below describe how the script is executed and the types of problems you may encounter.

Mapping of Execution Containers and Browser Locations

The table below lists the available browser locations as well as the corresponding provider and container location.

Browser Location	Provider	Container Location
Ashburn, Virginia	AWS	US East (N. Virginia)
Montreal, Québec	AWS	
Toronto, Ontario	Azure	
Amsterdam, Netherlands	Azure	EU (Ireland)
Dublin, Ireland	AWS	
Frankfurt, Germany	AWS	
London, United Kingdom	AWS	
Milan, Italy	AWS	
Paris, France	AWS	
Hong Kong, China	AWS	Asia Pacific (Tokyo)
Seoul, South Korea	AWS	
Tokyo, Japan	AWS	
Chennai, India	Azure	Asia Pacific (Singapore)
Mumbai, India	AWS	
Singapore	AWS	

Melbourne, Australia	Azure	Asia Pacific (Sydney)
Sydney, Australia	AWS	
Boardman, Oregon	AWS	US West (Oregon)
San Francisco, California	AWS	
Sao Paulo, Brazil	AWS	
San Antonio, Texas	Azure	

Hosted Agent Coverage Variations

Synthetic hosted agents have the following coverage variations:

- The US EUM cloud has access to both AWS and Azure locations.
- The Frankfurt EUM cloud has access to only AWS locations.
- The Sydney EUM cloud has access to only AWS locations.
- The Bombay EUM cloud has access to only AWS locations.

Synthetic Credential Vault

Deployment Support

Synthetic Credential Vault securely stores credentials used in synthetic jobs. When you create a job with a synthetic script, you can add a credential, such as a username and password to log in to an application, and inject a reference to the stored credential into the script. When the job runs, the synthetic script uses that reference to retrieve the credentials stored in Synthetic Credential Vault.

How to Use Synthetic Credential Vault

1. Go to ⚙ > **Tools** > **Manage Synthetic Credentials**.
2. Add a credential in a key-value pair to Synthetic Credential Vault.
3. Create a synthetic job using a synthetic script.
4. Inject the credential key into the synthetic script.
5. The synthetic job runs and retrieves the credential value associated with the credential key.

Add Synthetic Credentials

In Synthetic Credential Vault, you can add credentials one-by-one or import multiple credentials.

Syntax Rules

This table describes credential syntax rules used for creating credentials and adding credentials to synthetic scripts.

Element	Rule
Keys	• 1-100 characters • Alphanumeric characters, hyphens, and underscores only • Keys must be unique per EUM account
Values	• 5000 characters • Any non-control, printable UTF-16 characters
Credential key name (used in synthetic scripts)	`"<%key%>"`

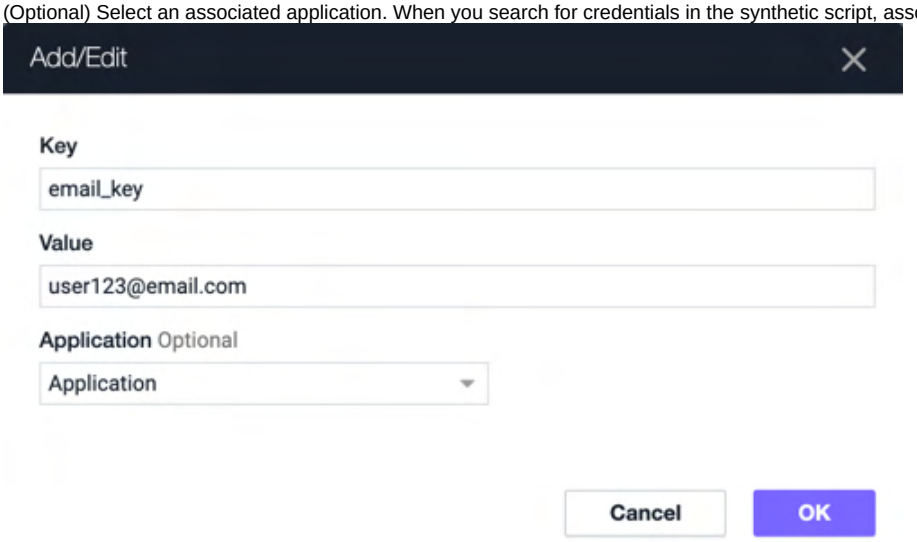
Add a Single Credential

> ⓘ If you want to use a combination of credentials, such as a username and password combination, you must create two separate credentials: one credential for the username and one for the password.

To add a single credential for a sample email:

1. Go to ⚙ > **Tools** > **Manage Synthetic Credentials**.
2. Click **Add**.
3. Enter a key and value. Make sure you follow syntax rules.

4. (Optional) Select an associated application. When you search for credentials in the synthetic script, associated credentials are suggested first.



Import Multiple Credentials

You can add multiple credentials at once using the **Import** button.

To add multiple credentials for sample emails and passwords:

1. Go to > **Tools** > **Manage Synthetic Credentials**.
2. Click **Import**.
3. Copy and paste credentials or type them line by line. Make sure you follow syntax rules.

- The maximum number of credentials that can be imported at once is 250.
- You cannot add associated applications to imported credentials. After importing, you must edit each credential individually to add the associated application.

Example

```
email_key=user123@email.com
password_key=Password123
email_key=user456@email.com
password_key=Password456
```

Use Credentials in a Synthetic Script

You can retrieve a credential value by inserting the corresponding credential key in a synthetic script. Credentials associated to a specific application appear first when you start typing the key in the script.

To create a synthetic job:

1. Go to **User Experience** > **Jobs** > **Add**.
2. Add your synthetic script. See Write Your First Script.
3. Locate the DOM element associated with your credential.
4. Enter the credential key using the syntax `"<%key%>"`.

Synthetic Script

```
driver.find_element_by_id("email").send_keys("<%email_key%>")
driver.find_element_by_id("pass").send_keys("<%password_key%>")
```

User Permissions

In the Controller UI, you can configure roles, users, and groups with Account-level, read-only permission for Synthetic Credential Vault. See Roles and Permissions for more information. As an admin-level user, we recommend you add a read-only role for Synthetic Credential Vault and assign that role to multiple users and/or groups.

To create a role in the Controller UI with read-only permission:

1. In the Controller UI, go to  > **Administration** > **Roles**.
2. Under **Roles**, click **Create**.
3. Add a role name, such as "Synthetic Credential Vault (read-only)."
4. Under **Account**, click **Add+**.
5. Check the "View Synthetic Credential Vault" box.
6. Under **User and Groups within this Role**, add users and/or groups.

The table below describes permission differences between admin and read-only users for using Synthetic Credential Vault.

User Level	Permission
Admin	Requires "Administration," "Agents," and "Getting Started Wizard" permissions to: • Manage all users • Add, edit, and delete credentials • View both keys and values of stored credentials • View users, timestamps, and associated applications
Read-only	• View keys of stored credentials • View users, timestamps, and associated applications

Alerts for Browser Synthetic Monitoring

Related pages:

- Policies
- Health Rules
- Actions

You can set policies and alerts based on synthetic events, or you can set thresholds to trigger alerts when you create or edit a synthetic job.

Synthetic Event Types

The following two types of synthetic events are supported and can be used to trigger policies and actions:

- Synthetic Availability
- Synthetic Performance

You can set policies with alerts for these synthetic events, selecting either or both warning or critical conditions.

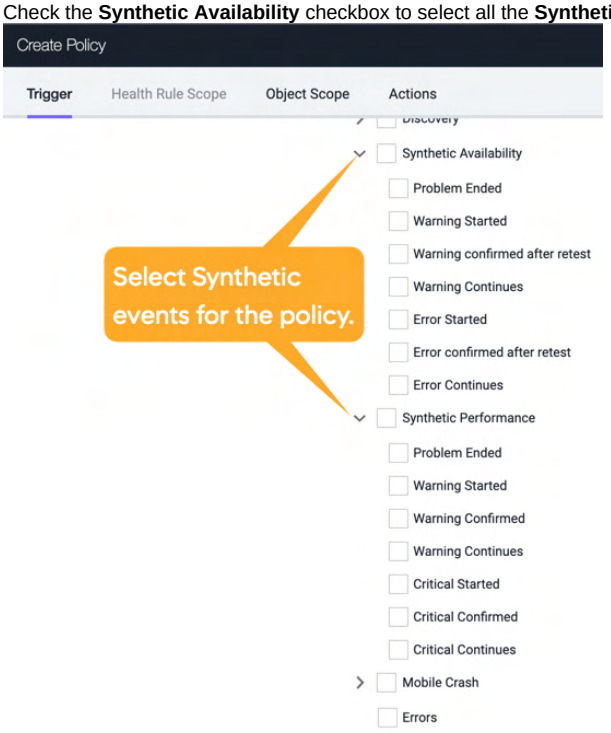
Synthetic Policies

You can use Synthetic Availability and Performance events to trigger policies, which can, for example, send an alerting email. See Configure Policies for more information.

Create Policy for Synthetic Availability

You can select warning and/or error events for policies for Synthetic Availability.

1. From the **Browser App** menu, click **Alert & Respond**.
2. Click **Policies**.
3. From the **Policies** page, click **Create Policy Manually**.
4. From the **Create Policy** dialog, enter a name for your policy.
5. Check the **Synthetic Availability** checkbox to select all the **Synthetic Availability** events or check one or more specific events.



6. Click **Next**.
7. Create an action to be executed when any of the **Synthetic Availability** events are triggered. See Actions for more information.

Site Availability

Availability is based on the session status:

- Single URL jobs
  - Warning = 4xx response, (session status is either `WARNING` or `BROKEN`)

    - Error = 5xx response, timeout, cannot contact application (`FAILED`)
- Scripted jobs
    - Warning = your script threw an uncaught exception (`BROKEN`) or the session status is `WARNING`.
    - Error = your script had an assertion failure or timed out (`FAILED`)

Performance Issues

Performance events are based on the criteria you set up when you set up scheduled jobs. For an example of the health rule violation process, see Health Rule Violation Events.

Create Policy for Synthetic Performance Events

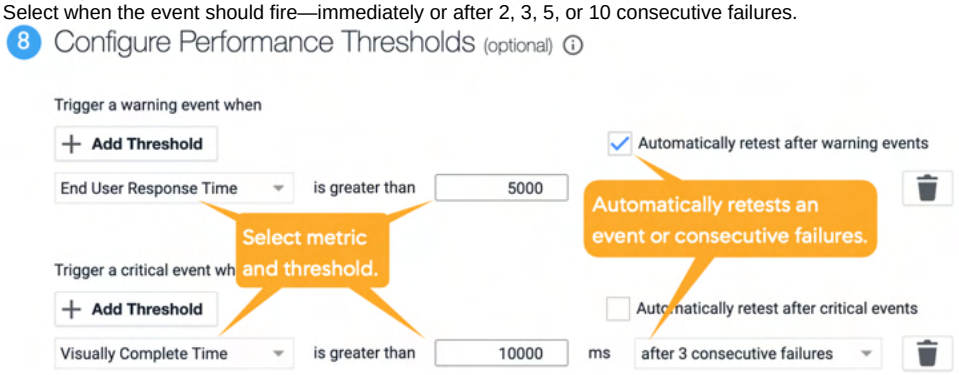
You can select warning and/or error events for policies for Synthetic Performance.

1. From the **Browser App** menu, click **Alert & Respond**.
2. Click **Policies**.
3. From the **Policies** page, click **Create Policy Manually**.
4. From the **Create Policy** dialog, enter a name for your policy.
5. Check the **Synthetic Performance** check box to select all the **Synthetic Performance** events or check one or more specific events.
6. Click **Next**.
7. Create an action to be executed when any of the **Synthetic Performance** events are triggered. See Actions for more information.

Configure Performance Thresholds

You can also configure performance thresholds that will trigger events when the thresholds are exceeded.

1. Click **+ Add** to add a new job.
2. In the **New Job** popup, scroll down to **Configure Performance Thresholds**:
3. From either **Trigger a warning event when** or **Trigger a critical event when**:
    - Click **+ Add Threshold** to add a performance threshold for triggering a warning/critical event.
    - Select a performance threshold, enter a value that will trigger a warning event.
    - Select when the event should fire—immediately or after 2, 3, 5, or 10 consecutive failures.



4. You can select multiple performance thresholds by clicking **+ Add Threshold** multiple times.

Performance Thresholds

The following are the supported performance thresholds. Follow the links for descriptions of the performance thresholds.

- End User Response Time
- Fully Loaded Time
- Visually Complete Time
- First Byte Time
- Start Render Time
- Bytes per Pageview

Synthetic Sessions

You use the **Session** page to analyze synthetic sessions. A synthetic session is a record of the complete sequence of events that make up one synthetic test. A session can cover one page or, in the case of scripted jobs, can extend to multiple pages. All session timestamps are in local time, regardless of the configured scheduled job's timezone.

A session begins at the first navigation start in the sequence and continues until the last page is fully loaded. In the case of single-page sessions, the session begins and ends when the *page* is fully loaded (the session duration represented by the metric Visually Complete). Like Browser RUM Analyze and Network Request Analyze, the Synthetic Sessions view enables you to analyze results from *all* synthetic sessions.

> For single-page applications (SPAs), sessions will include the base page and its virtual pages as *one* page: the base page.

Session Status

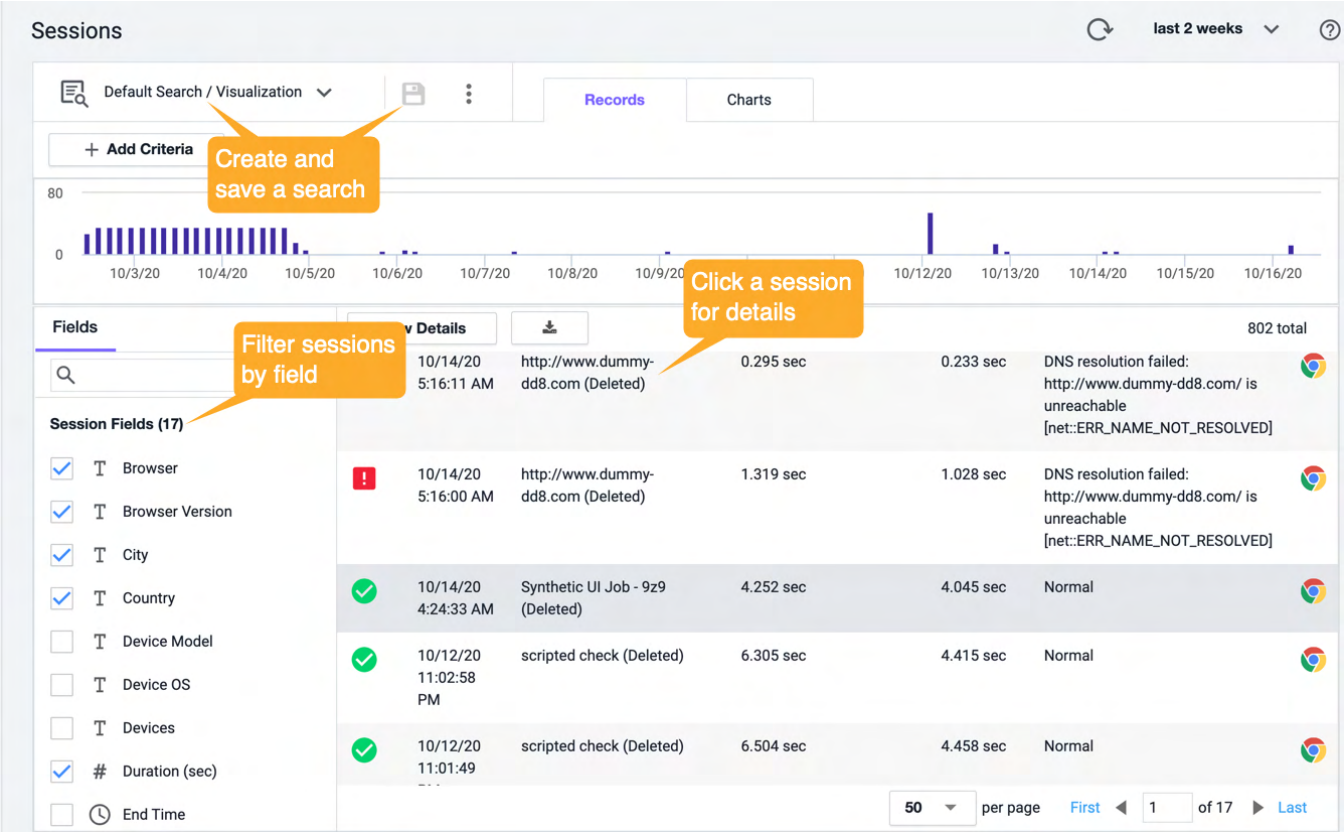
Sessions can have one of four statuses:

- `OK`: The test ran successfully and returned data.
- `WARNING`: The test ran successfully, but there were inaccessible resources.
- `BROKEN`: Your job has an error. You can find the error in the script console output. When the status is `BROKEN`, no event is created and the job is not retested.
    - In single URL jobs, the test received a 4xx response.
    - In scripted jobs, your script threw an uncaught exception. This can, for example, happen if you try to perform an action, like a click, on an element which is not on the page you are testing.
- `FAILED`: The synthetic test has detected a problem with your site. When failed jobs have the status `FAILED`, an event is created and the job is run again to confirm there was an error.
    - In single URL jobs, one of the following occurred:
        - the browser couldn't reach the URL
        - the browser received a 5xx response
        - the session timed out
    - In scripted jobs, your script had an assertion failure or timed out.

Session UI

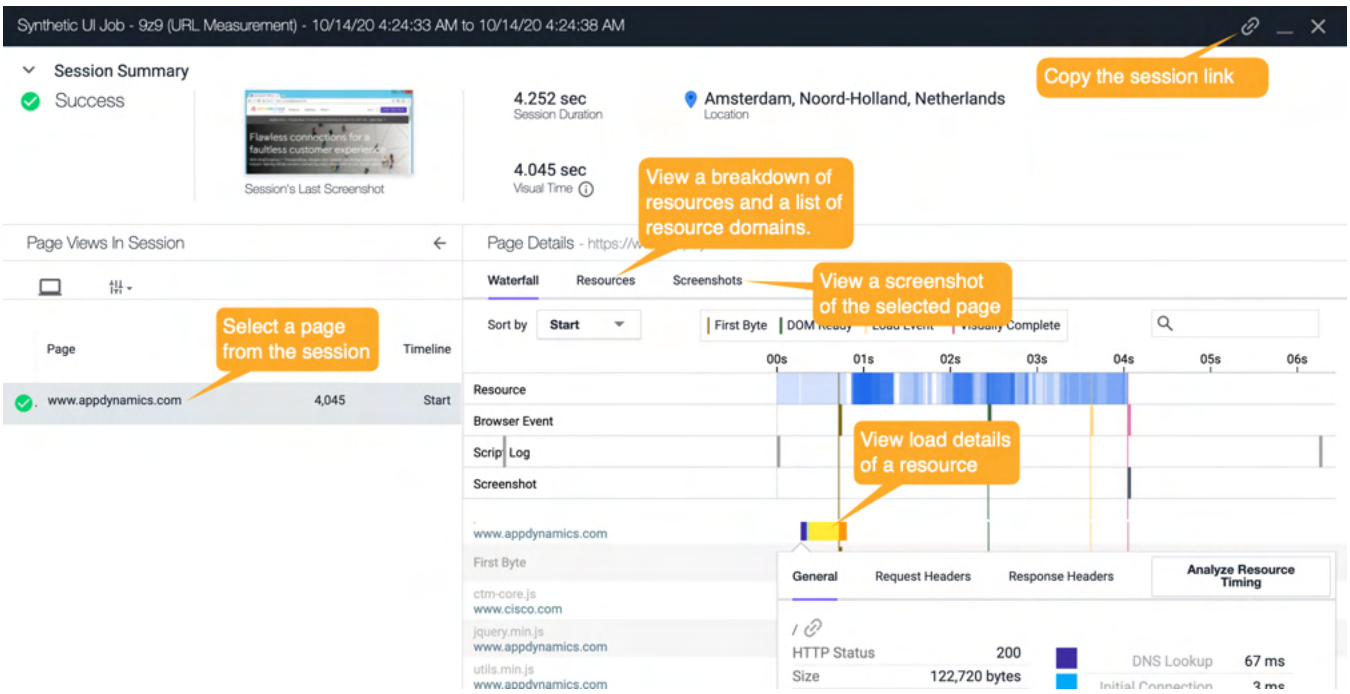
Records

The Records tab enables you to scan individual requests and allows you to filter and sort to get exactly the data set in which you are interested.



Session Details

Select a specific session and click **View Details** to see detailed information, including a page load waterfall and tabular details for resources. In multi-page sessions, the left navigation allows you to select each page from the sequence.



Business Transactions

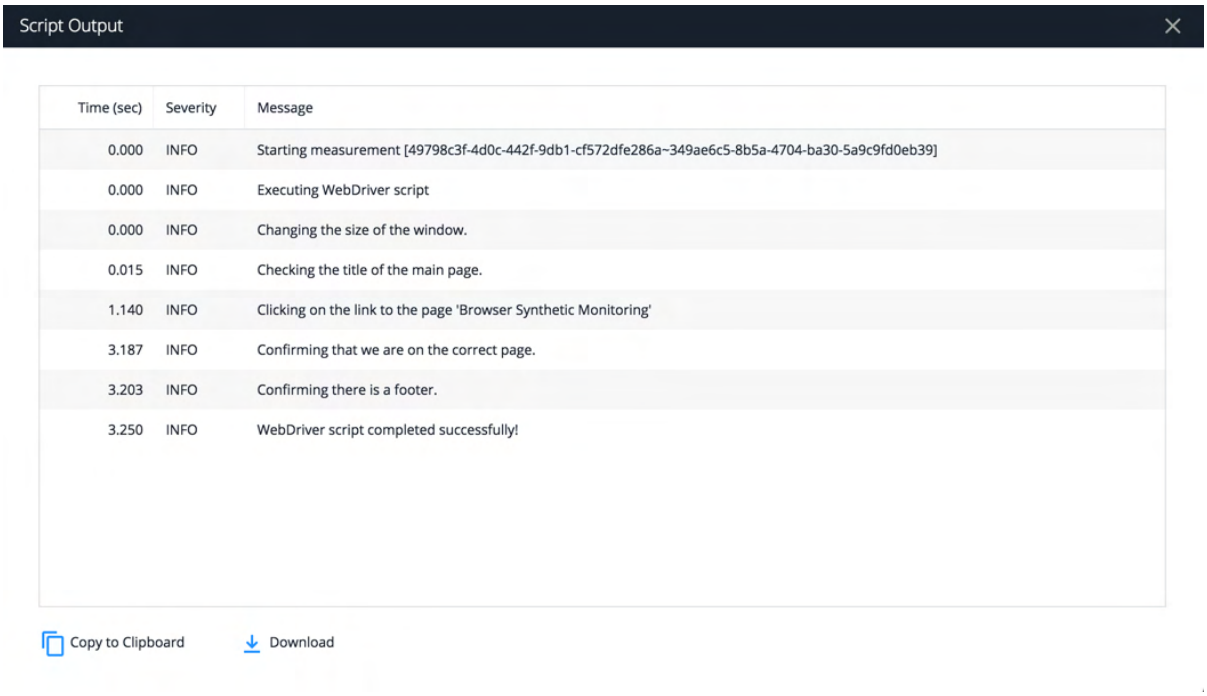
When a business and a browser application have correlated business transactions, you can also create synthetic jobs from that browser application that makes requests to the business application to get business transactions in synthetic sessions. You can view business transactions in the Waterfall and the Business Transactions tabs of the **Session Details** dialog.

If you see **View Snapshot >** next to resource in the waterfall, that indicates that there was a correlated business transaction. Clicking it will take you to the business transaction snapshot from the backend.

You can also click the Business Transactions tab to see a list of the server-side business transactions this request initiated. Click through to the related **Business Transaction** dashboards.

Script Output

If you used a script, you can click through to see any script output.



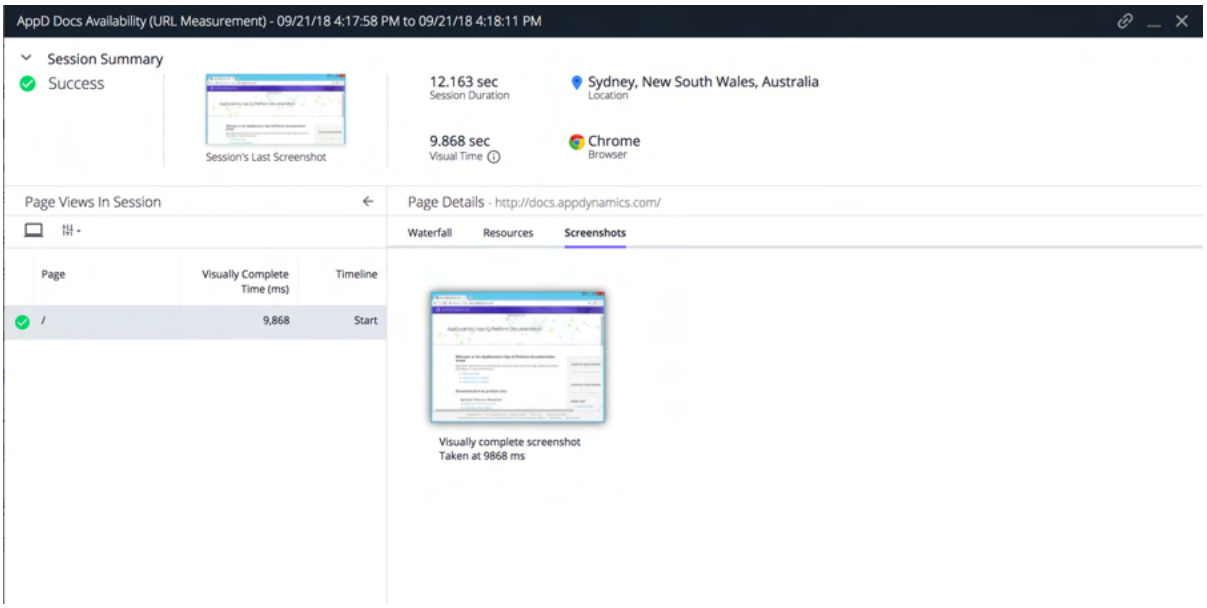
Click **Copy to Clipboard** to copy the script output to your clipboard or **Download** to download the script output.

Session Screenshots

Synthetic captures screenshots of the browser as your job runs. By default, it takes one screenshot per page, when the page is finished rendering. Screenshots are retained for 30 days.

In scripted jobs, you can call an API from your script to take a maximum of 10 screenshots per page. The screenshot will be of the visibly complete content in the viewport (won't include the browser chrome).

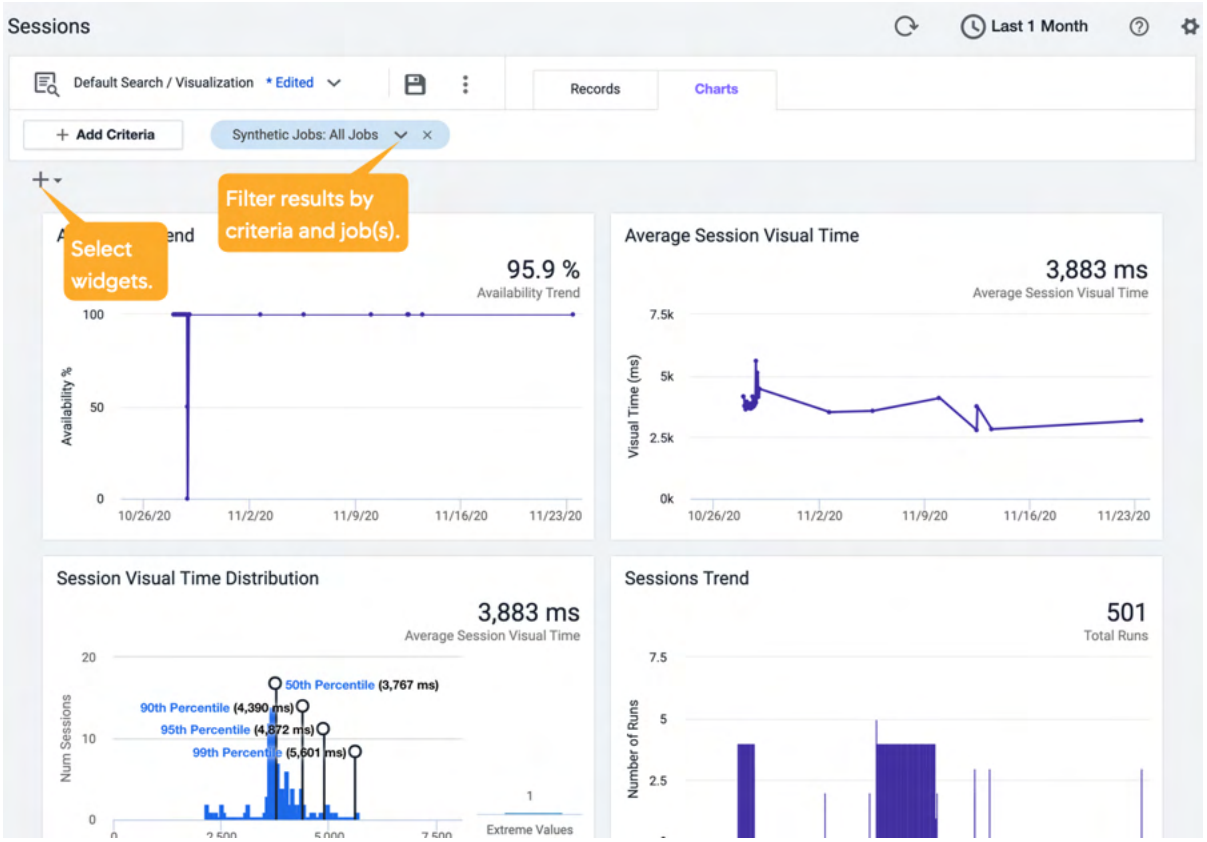
The Screenshots tab shown below displays the screenshot as a 200 x 150 image. The resolution of the screenshot will be the same as the device where the screenshot was taken.



The Waterfall tab will contain the visually complete screenshot and any custom screenshots (e.g., `screenshot2` in the screenshot below) triggered by the script.

Charts

The Charts tab provides you with a set of widgets that offer predefined visualizations of the data set you have created, allowing you to create a dashboard. You can delete, re-add, resize, and drag-and-drop to move the widgets.



Analyze Session Results

You understand the results of your synthetic jobs through sessions. From session details, you can determine if there were errors, what type of errors, and diagnose what factors are affecting performance. The following sections show you how to analyze synthetic session results for errors and performance issues.

Errors

Determine Type of Error

At this time, you should be able to determine the type of error by the **Status** and **Error Type** fields. The session status indicates whether your jobs are passing, failing (there is a problem with the system being tested), or broken (there is a problem with the job itself). The **Error Type** field gives more information about the cause of the error.

Understand the Error

1. From the **Sessions** > **Records** tab, select a session with an error and click **View Details**.
2. From the **Session Details** dialog, you can view when the error occurred and the returned error message in the Waterfall tab.
3. If your job used a custom script, you can click **Show Script Output** to view the script results. You can download or copy the results as well.
4. You can also view the screenshot to see how the error might be manifested to end users. You might want to display the errors in a more user-friendly way and possibly omit error details.

Performance

One of the best ways to see if your job is having performance issues is to view widgets representing aggregated metrics in the Charts tab. With a quick glance, you can see if there are site availability issues, where end users might be having bad user experiences, how long sessions are lasting, and how long it's taking for pages to be visually complete.

Network Latency

The Waterfall tab shows you a timeline of when resources are loaded and plots the events First Byte, DOM Ready, Load Event, and Visually Complete that mark significant and discreet moments from the time the browser receives data until the user can see a complete page.

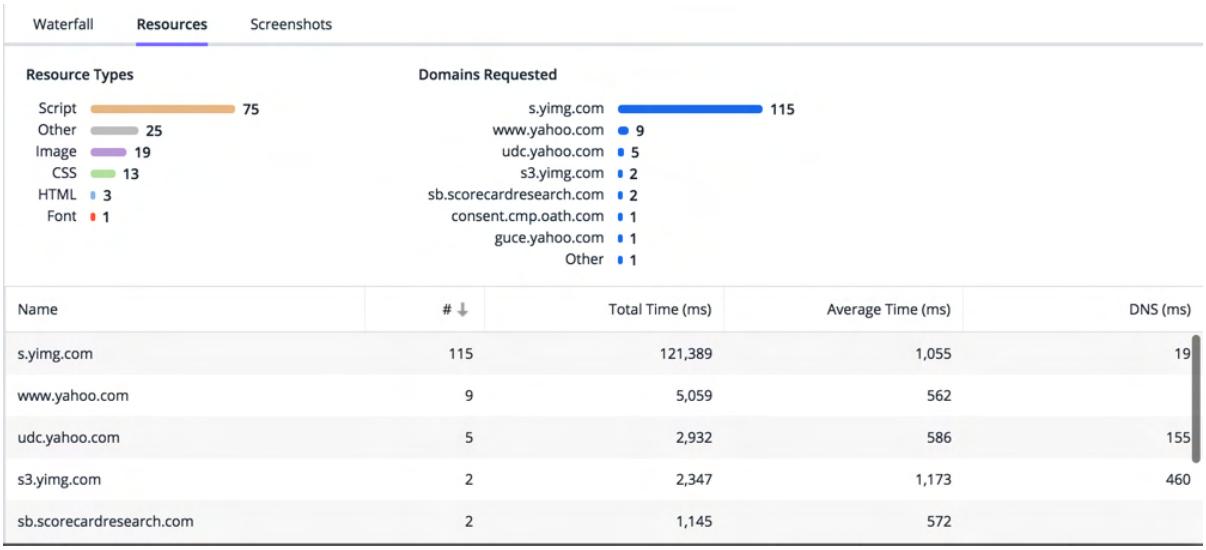
To understand session performance, you can compare the end user response time (the same as the `onload` time) of a page with the session duration. This gives you an idea of which pages are taking too long to load and how that might be affecting the session.

You can also click one of the bars to determine if a resource is taking a long time to load. For example, if an image is taking too long to load, you can take actions to optimize the loading time such as caching the image, reducing its size, confirm that the image is the size that you're displaying it, or not use the image at all.

Problematic Resources

The **Session Details** dialog shows you the final session state and the session state set by each page. The final session state is indicated in the top-left corner, whereas, the session state set by each page is shown in the **Page Views in Session** sidebar. In the screenshot below, the ⚠ icon indicates that the first page set the session status to "Warning". From the Waterfall tab, you can also discover which resources didn't load, either marked in yellow for the "Warning" state or in red for the "Failed" state.

The Resources tab shows you the number of resources by category, the domains requested, and key performance metrics that you can use to sort the results. You can quickly identify resources that are slowing the page load and potential DNS problems.



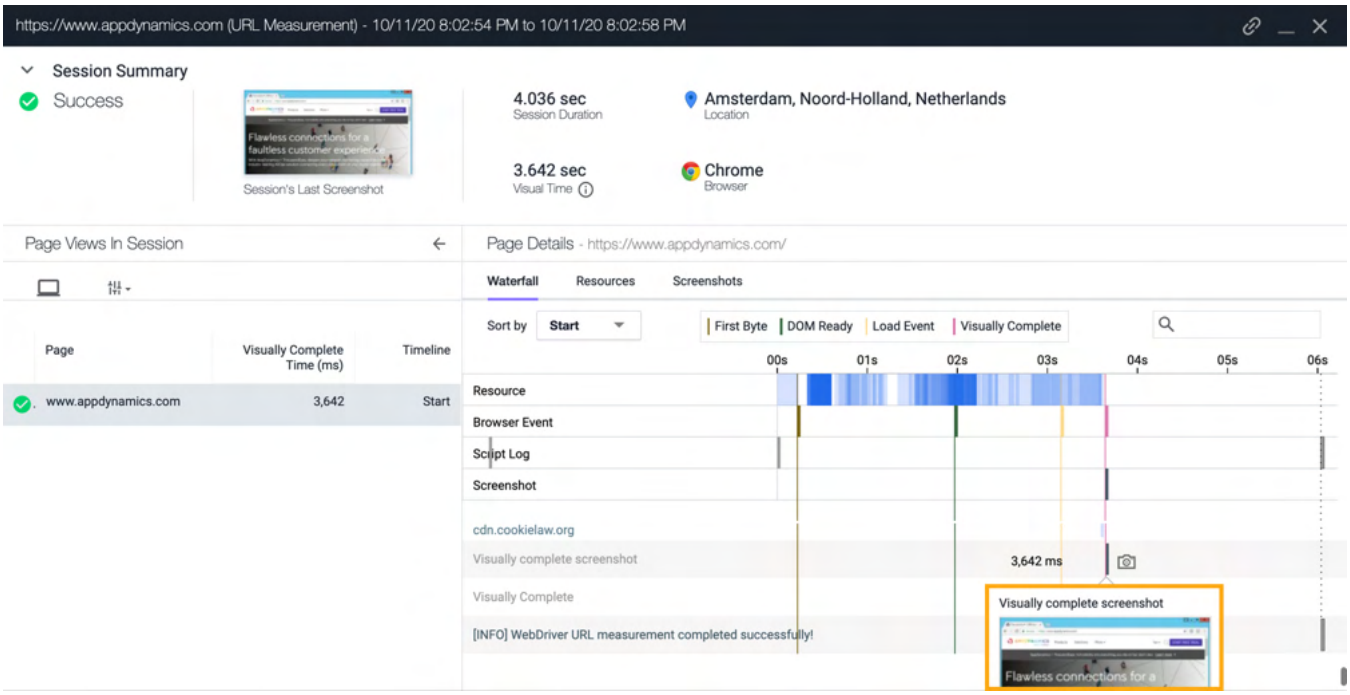
Name	# ↓	Total Time (ms)	Average Time (ms)	DNS (ms)
s.yimg.com	115	121,389	1,055	19
www.yahoo.com	9	5,059	562	
udc.yahoo.com	5	2,932	586	155
s3.yimg.com	2	2,347	1,173	460
sb.scorecardresearch.com	2	1,145	572	

You can also configure your synthetic jobs to check the availability of resources. You can set the session status to **Failed** or **Warning** when resources are unavailable. See Configure Availability Rules for configuration instructions. When the session status is set to **Failed** or **Warning**, Browser Synthetic Monitoring triggers corresponding events that can be used to create alerts. This enables you to be alerted when resources are missing. See Alerts for Browser Synthetic Monitoring to learn how to create availability policies.

View the Effects of Performance Issue

Screenshots will show you what your user is seeing. From the screenshot, you can determine whether the content has completely loaded (text, images, containers, UI components) and the page is styled correctly (CSS).

From the Waterfall tab, you can determine when the screenshot (visually complete) and view a small thumbnail by hovering over the camera icon.

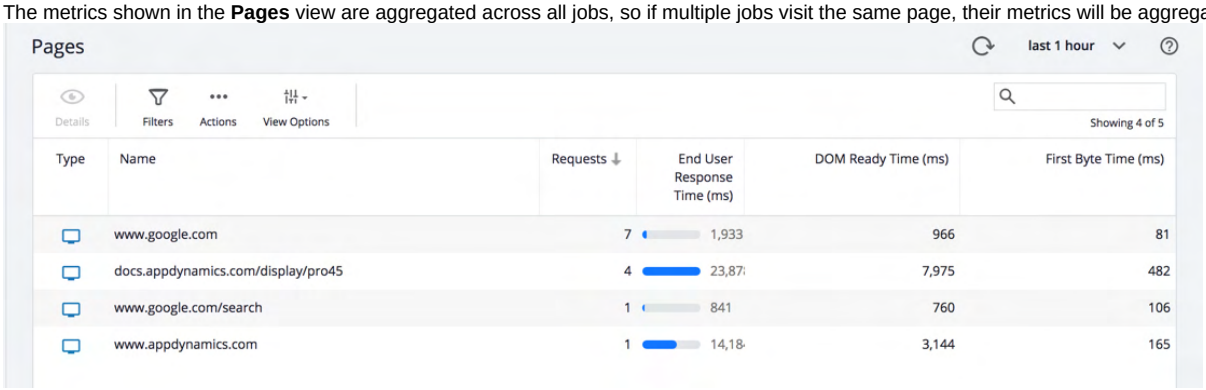


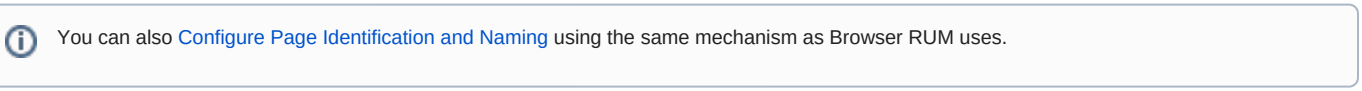
By navigating to the Screenshots tab and double-clicking the screenshot, you can view a larger screenshot to see details such as the styling, UI components, and content.

Synthetic Pages

You use the **Pages** view to see an aggregated view of how each page visited in your jobs is performing over time.

The metrics shown in the **Pages** view are aggregated across all jobs, so if multiple jobs visit the same page, their metrics will be aggregated.



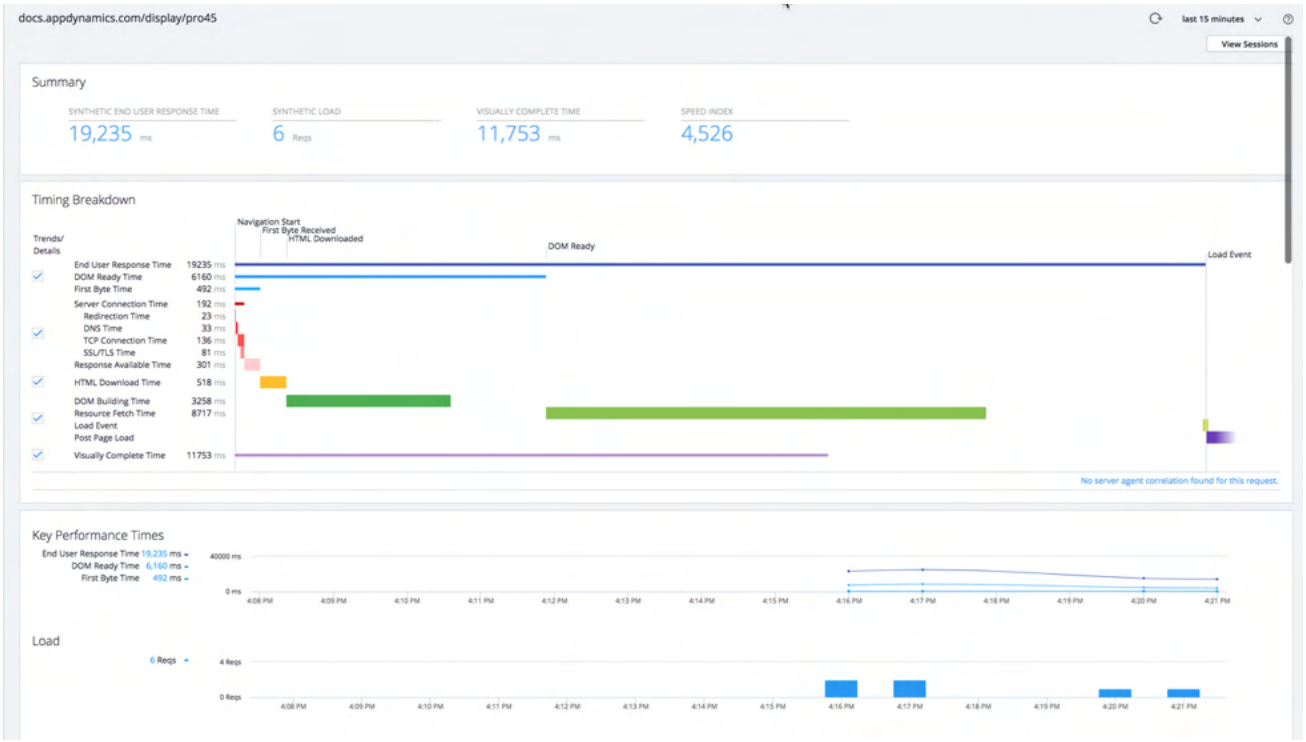


You can also Configure Page Identification and Naming using the same mechanism as Browser RUM uses.

Pages Details

Just as in the Browser RUM view you can filter the dataset and manage which pages are displayed and which metrics are shown. If you select an individual page and click **Details**, you are shown a **Page** dashboard very similar to the Browser RUM Pages Dashboard.

The **Synthetic Pages Dashboard**, however, includes some additional metrics. To understand these added synthetic metrics, you can hover over the metric name and a tooltip explaining the metric popup. See Browser Synthetic Metrics.



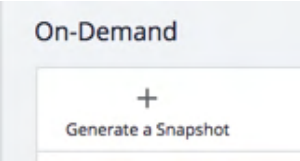
Request Synthetic Snapshots On-Demand

Browser Synthetic On-Demand allows you to take quick snapshots of single pages whenever you wish. See Browser Synthetic Metrics.

> Browser Synthetic On-Demand is only available on SaaS deployments of the Controller.

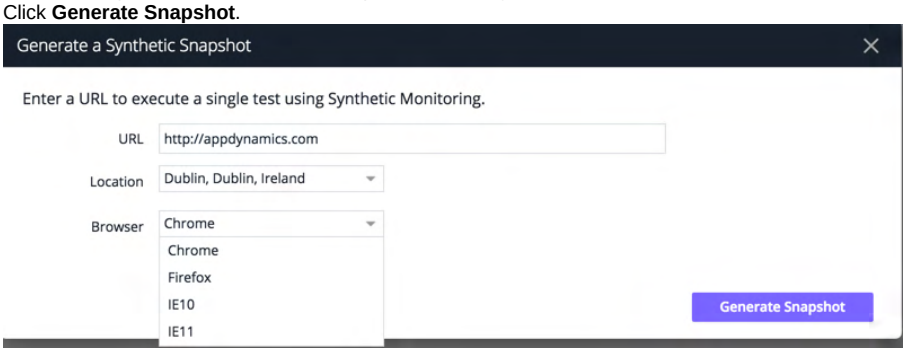
Create an On-Demand Snapshot

To create an on-demand session, click **+ Generate a Snapshot** at the top of the **On-Demand** list page.



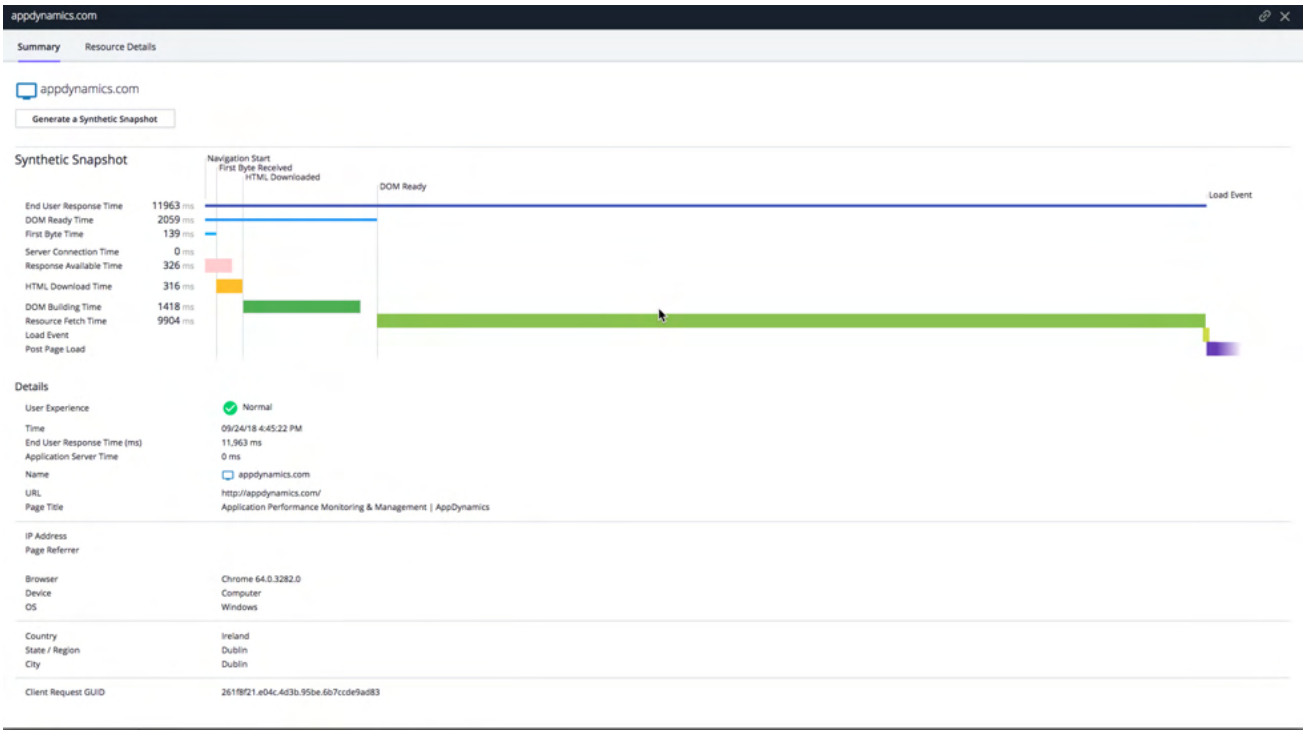
After the **Generate a Synthetic Snapshot** popup appears:

1. Provide the URL of the page you wish to test.
2. From the dropdown, select where the request should originate and which browser should be used.
3. Click **Generate Snapshot**.

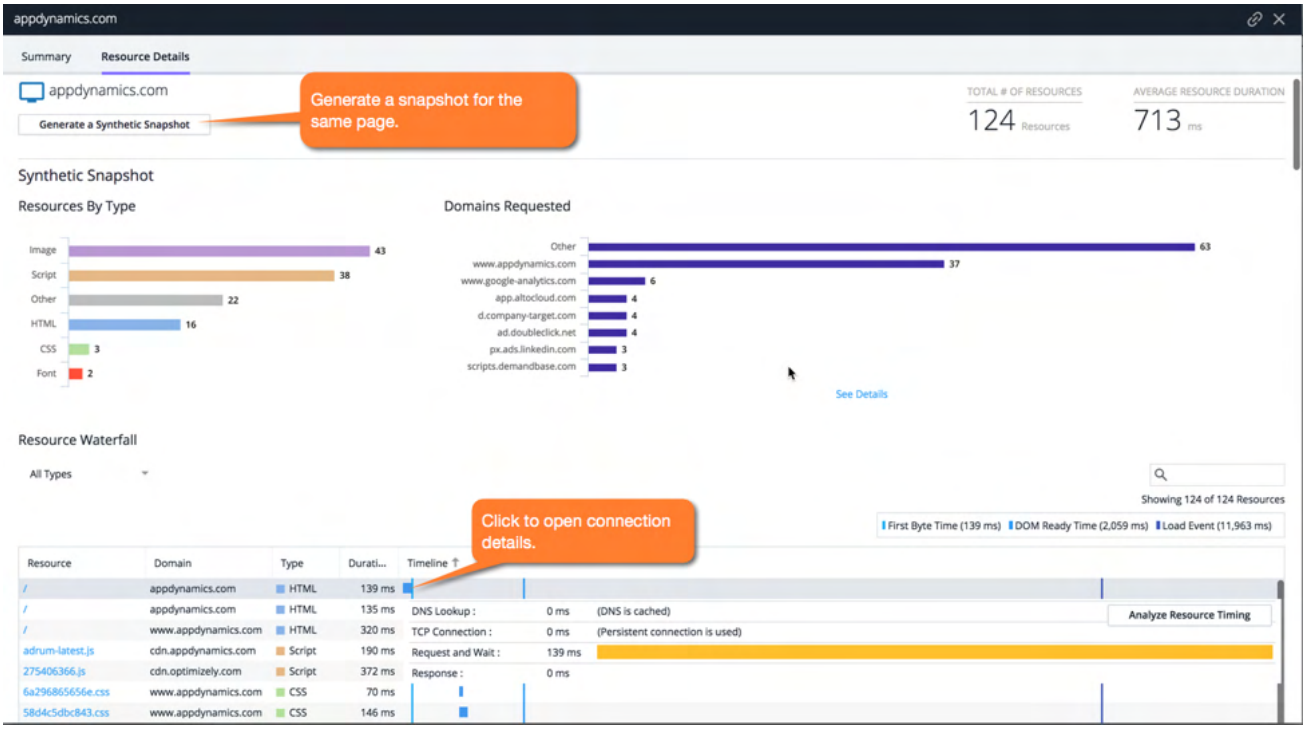


View On-Demand Snapshot Results

When the test completes, it is marked as **Done** in the Snapshot list. To see the detailed results, click the test. The Summary tab provides a waterfall of the page load and additional request details.



The Resource Details tab provides visualizations of **Resources by Type** and **Domains Requested** and a waterfall displaying the download of every resource. Click a waterfall bubble to see connection details. **Domains Requested** is also presented as a table.



Browser Synthetic Metrics

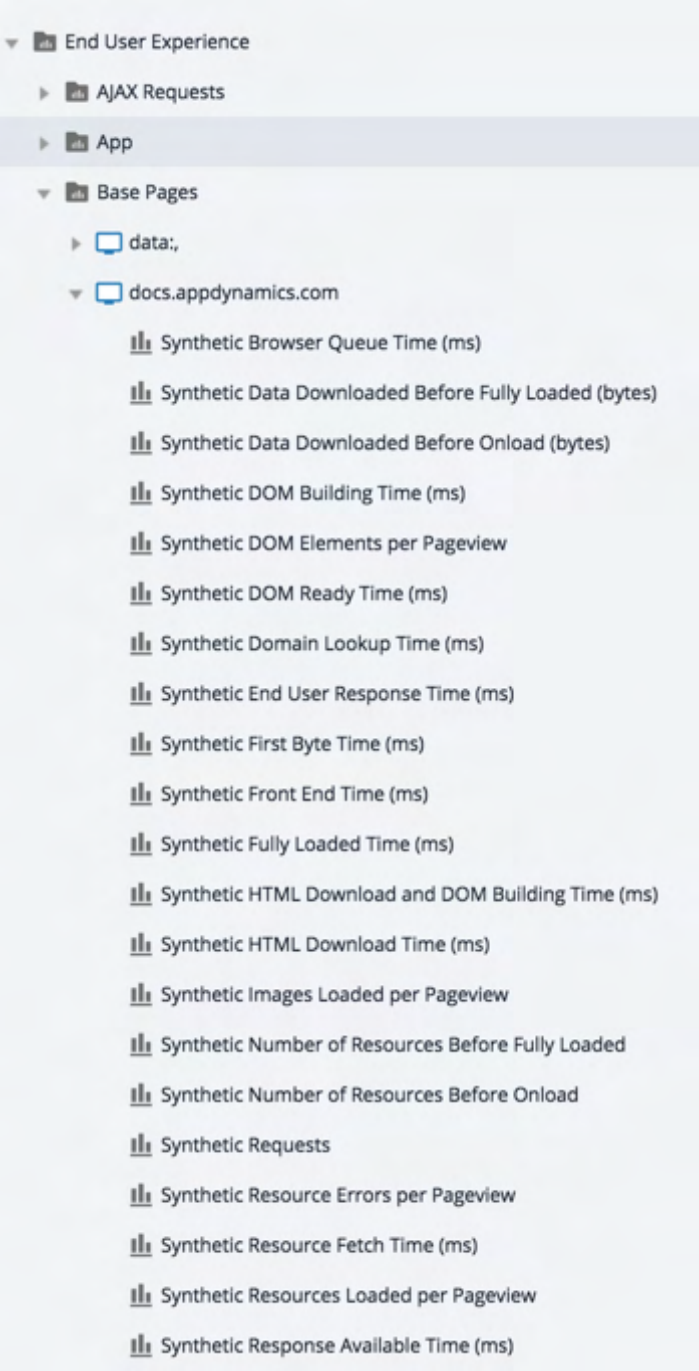
Related pages:

- Browser RUM Metrics
- Analytics Synthetic Sessions Data

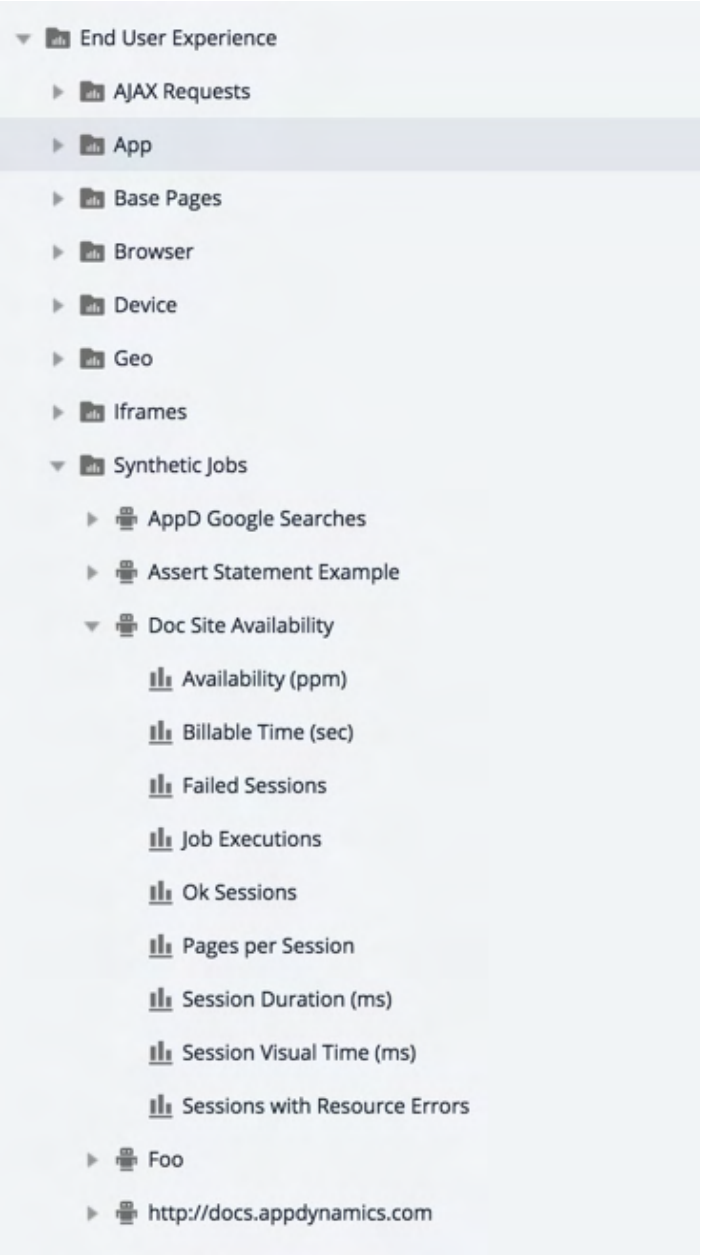
Many of the metrics collected by Browser Synthetic are identical to those collected by Browser RUM. Additional synthetic-only metrics are defined in Browser Synthetic Metrics Defined.

Key Browser RUM metrics are displayed on the **Geo**, **Sessions**, and **Pages** dashboards. They can also be seen in the **Metric Browser**, just like RUM metrics, giving you the ability to view these metrics in the context of Ajax requests, iframes, applications, base pages, browsers, devices, and geographic locations.

Metrics will only appear in the **Metric Browser** if there is synthetic data. For example, if your synthetic jobs do not have failed sessions, the metric **Failed Sessions** will not appear.



There are also metrics for specific tests and, under the **Synthetic Jobs** heading, for Synthetic job health overall.



Browser Synthetic Metrics Defined

Name	Definition
Availability	The ratio of OKAY sessions (successful) to all sessions, expressed per-million: 0 (for all failed) to 1,000,000 (for all succeeded). The ratio is calculated using the following: (# of successful sessions/# of total sessions) * 1,000,000. Availability is represented in per-million instead of as a percentage because Controller metrics are given as integers, making expressing the availability as a percentage less precise. For example, although the availability percentage in actuality might be 99.99%, it would only be able to be expressed as an integer such as 100% or 99%. To convert per-million to a percentage, just divide by Availability (ppm) by 10,000. For example, if Availability (ppm) is 999,800, then Availability (%) is 99.98%. Most Controller UI components display metrics as percentages, but the **Metric Browser**, which shows "raw data", shows the value per-million.
Billable Time	The seconds that were billed against your Synthetic license.
Broken Sessions	The number of sessions that completed with the session status BROKEN, indicating that the job has an error. You can find the error in the script console output.

Data Downloaded Before Fully Loaded	The bytes of data downloaded by the browser between the navigation start of a page and the navigation start of the next page or the end of the session. In particular, includes any resources loaded after the page `load` event.
Data Downloaded Before Onload	The bytes of data downloaded by the browser between the navigation start for a page and the `load` event for that page.
DOM Elements per Pageview	The number of DOM elements present on a page at the time of the page's `load` event. Includes sub-documents in frames.
Failed Sessions	The number of sessions that completed with the session status `FAILED`, indicating the synthetic test has detected a problem with your site. See Session Status for more information.
Fully Loaded Time	The time from the start of the initial navigation until 2 seconds of no network activity has passed after Document Complete. This will usually include any activity that is triggered by JavaScript after the main page loads. The Fully Loaded Time is given in milliseconds and percentiles.
Job Executions	Total number of sessions (per browser, per location) for a given job.
Number of Resources Loaded Before Fully Loaded	See **Data Downloaded Before Fully Loaded** above.
Number of Resources Loaded Before Onload	See **Data Downloaded Before Onload** above.
Ok Sessions	The number of sessions that completed with the session status `OK`, indicating the synthetic job ran successfully and returned data.
Pages per Session	The number of pages per session.
Session Duration	For scripted jobs: the time needed to execute the script from the first line to the last line. For URL jobs: the time from when the browser initiated the first HTTP request until the last resource was loaded.
Sessions with Resource Errors	The number of sessions that had missing or inaccessible resources.
Session Visual Time (SVT)	This metric captures the time the browser spent loading pages. Relative to **Session Duration**, this excludes the script process time and the time your script was interacting with content on a given page. Specifically, the **Session Visual Time** for a session is the sum of the **Visually Complete Times** for each page.
Speed Index	This metric captures whether a page renders incrementally or all at once. For example, if two pages both take five seconds to finish rendering, but one is mostly rendered after a second, while the other is blank until the last moment. Then the first page will have a **Speed Index** close to 1000, while the second will have a **Speed Index** close to 5000 (the **Visually Complete** time in milliseconds).
Visually Complete	The time when an end-user would determine that the page is visually complete in the viewport. This time is calculated by using the last visual change to the page and is in milliseconds. The page represents the viewable content (the viewport) in a browser window. The default size for a browser window is 1024 x 768.
Warning Sessions	The number of sessions that completed with the session status `WARNING`, indicating the test ran successfully, but there were inaccessible resources.

Private Synthetic Agent (Linux Based Agent in Kubernetes Container

This page describes the hardware components required for the Private Synthetic Agent (PSA) to run on the Kubernetes (K8S) cluster.

Linux based PSA supports these platforms:

- SaaS Controllers
- Python >=3
- Chrome browser

Prerequisites

- Kubernetes cluster version from 1.15 to 1.18
- License units:
  - Browser Synthetic Monitoring - Private Agent - Per Location (SaaS) or
  - Browser Synthetic Monitoring - Private Agent - Unlimited Locations (SaaS)

Hardware Requirements

This table lists the hardware components and their recommended configuration:

Component	Description	Minumum Configuration		Recommended Configuration	
		# Instances	Per Instance	# Instances	Per Instance
Heimdall (Orchestrator)	Orchestrator that connects to the Synthetic Cloud to fetch measurement jobs for the PSA cluster.	1	Processor: 2 vCPU Memory: 4 GB RAM	2	Processor: 3 vCPU Memory: 5 GB RAM
Postgres (DB)	Database (DB) for the local storage of in-progress measurement metadata. This DB does not store any business data. Any DB data loss or crash impacts on-going measurements only. Instead of setting up a new DB, you can use the existing Postgres DB set up with HA (High Availability) as a DB Java Database Connectivity (JDBC) url (part of configuration to Heimdall).	1	Processor: 2 vCPU Memory: 4 GB RAM	1	Processor: 3 vCPU Memory: 6 GB RAM
Measurement Container	A temporary or short-lived container auto-orchestrated by Heimdall to execute every measurement. The measurement container image can be customised to install additional libraries.	Based on need, a temporary container instance is scheduled for every measurement	Burstable CPU-RAM Min: 0.5 vCPU, 1 GB RAM Max: 1.25 CPU, 2 GB RAM	Based on need, a temporary container instance is scheduled for every measurement	Burstable CPU-RAM Min: 0.5 vCPU, 1 GB RAM Max: 1.25 CPU, 2 GB RAM

Number of Measurement Containers

You can calculate the number of required concurrent containers using this formula:

((Number of Jobs) x (Average job duration in minutes) / (Frequency in minutes))

For example, the number of containers required for 10 jobs with an average duration of 30 seconds per job executed at 5 minute interval needs:

(10 x (30/60)) / 5) = 1 container

Concurrent measurements

The Kubernetes cluster specifications determine the number of concurrent measurements that can be executed. Any submission of jobs beyond the infra limits leads to the queuing of jobs in the Kubernetes cluster. To scale the number of concurrent jobs automatically, enable cluster auto-scaler. K8S auto-scaler functionality depends on the cloud or the managed service provider.

To set up Private Synthetic Agent in any of the these environments, see:

- Set up Private Synthetic Agent in Amazon Elastic Kubernetes Service
- Set up the Private Synthetic Agent in Minikube
- Set up the Private Synthetic Agent in Azure AKS
- Set up the Private Synthetic Agent in Bare Metal K8s
- Configure Private Synthetic Agent

Set up Private Synthetic Agent in Amazon Elastic Kubernetes Service

To set up the Synthetic Private Agent in Amazon Elastic Kubernetes Service (Amazon EKS), perform the following:

1. Create the Kubernetes Cluster.
2. Build and customise the Docker image.
3. Tag and push images to the Registry.
4. Deploy the Private Synthetic Agent.
5. Monitor the Kubernetes cluster.

> This document contains links to AWS CLI documentation. AppDynamics makes no representation as to the accuracy of AWS CLI documentation because AWS CLI controls its own documentation.

Create the Kubernetes Cluster

To create a Kubernetes cluster in Amazon EKS:

1. Install and configure AWS CLI.
2. Based on your platform, install `eksctl` following the instructionshere.
3. Create a Kubernetes cluster using one of these valid Kubernetes version options: 1.14, 1.15, 1.16, 1.17, or 1.18.

```
EKSCTL_CLUSTER_NAME=eks-heimdall-onprem-cluster
EKSCTL_NODEGROUP_NAME=eks-heimdall-onprem-worker-nodes
EKSCTL_KUBERNETES_VERSION=1.18


eksctl create cluster \
--name ${EKSCTL_CLUSTER_NAME} \
--version ${EKSCTL_KUBERNETES_VERSION} \
--region us-west-2 \
--zones us-west-2a,us-west-2b,us-west-2c \
--nodegroup-name ${EKSCTL_NODEGROUP_NAME} \
--node-type t3.2xlarge \
--nodes 4 \
--nodes-min 2 \
--nodes-max 6 \
--ssh-access \
--ssh-public-key ~/.ssh/id_rsa.pub \
--managed \
--vpc-nat-mode Disable
```

> The node-type, nodes, nodes-min, and nodes-max in the code snippet are selected based on the recommended configuration type. You can specify a configuration of your choice with a different type and number of nodes. See EC2 instance types.

Access the Cluster

To access the Kubernetes cluster, follow these instructions to install kubectl, a utility to interact with the cluster.

To verify that the cluster is running, enter:

```
kubectl get nodes
```

Build and Customise the Docker Image

You can download the zip file for Simple Synth PSA installation from the Appdynamics Downloads Portal or from the beta upload tool. This file contains Dockerfiles for sum-chrome-agent, sum-heimdall, and Helm charts used to install the agent and set up monitoring. To build an image for sum-chrome-agent and sum-heimdall, ensure that Docker is installed. If it is not installed, you can download and install Docker from here.

For sum-chrome-agent:

1. Unzip the zip file to access the sum-chrome-agent directory.
2. Navigate to the directory and run the following command:

```
docker build -f Dockerfile-PSA -t sum-chrome-agent:<agent-tag> .
```

For sum-heimdall:

1. Unzip the zip file to access the sum-heimdall directory.
2. Navigate to this directory and run the following command:

```
docker build -f Dockerfile-PSA -t sum-heimdall:<heimdall-tag> .
```

You can use any value for <heimdall-tag> and <agent-tag>, but ensure that you use the same value in the subsequent steps.

Add Custom Python Libraries

In addition to the available standard set of libraries, you can add custom Python libraries to the agent to use in scripted measurements. You build a new image based on the image you loaded as the base image.

1. Create a Dockerfile and then create `RUN` directives to run `python pip`. For example, to install the library `algorithms` you can create a Dockerfile:

```
# Use the sum-chrome-agent image we just loaded as the base image
FROM sum-chrome-agent:<agent-tag>

# Install algorithm for python3 on top of that
RUN python3 -m pip install algorithms==0.1.4


# We can add more RUN directives for installing more libraries
# RUN python3 -m pip install ...
```

   You can create any number of `RUN` directives to install the required libraries.

2. To build the new image, enter:

```
docker build -t sum-chrome-agent:<agent-tag> - < Dockerfile
```

   The newly built agent image contains the required libraries.

Tag and Push Images to the Registry

You must tag and push the images to a registry for the cluster to access it. The Amazon EKS clusters pull the images from Elastic Container Registry (ECR), which is the managed registry provided by AWS.

To tag the images, enter:

```
docker tag sum-heimdall:<heimdall-tag> <aws_account_id>.dkr.ecr.<region>.amazonaws.com/sum/sum-heimdall:
<heimdall-tag>
docker tag sum-chrome-agent:<agent-tag> <aws_account_id>.dkr.ecr.<region>.amazonaws.com/sum/sum-chrome-agent:
<agent-tag>
```

You need to replace `<aws_account_id>` & `<region>` with your account id and region values.

To create repositories, enter:

```
aws ecr create-repository --repository-name sum/sum-heimdall
aws ecr create-repository --repository-name sum/sum-chrome-agent
```

To push the images, enter:

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-stdin <aws_account_id>.
dkr.ecr.<region>.amazonaws.com
docker push <aws_account_id>.dkr.ecr.<region>.amazonaws.com/sum/sum-heimdall:<heimdall-tag>
docker push <aws_account_id>.dkr.ecr.<region>.amazonaws.com/sum/sum-chrome-agent:<agent-tag>
```

Deploy the Private Synthetic Agent

The application is deployed to the cluster after the images are in the Registry. You use Helm chart to deploy and create all Kubernetes resources in the required order.

1. Install Helm following these instructions.
2. Create a new measurement namespace where Heimdall, Postgres database, and measurement pods will run.

   To create a new measurement namespace, enter:

   ```
   kubectl create namespace measurement
   ```

   Using a single command, you can deploy the Helm chart which contains the deployment details. To deploy the agent, you use the Helm chart `sum-psa-heimdall.tgz` in the zip file that you downloaded previously. Before you deploy the Private Synthetic Agent, you must set some configuration options. To view the configuration options, navigate to the previously downloaded `sum-psa-heimdall.tgz` file and enter:

   ```
   helm show values sum-psa-heimdall.tgz > values.yaml
   ```

   These are the configuration key value pairs that you need to edit in the `values.yaml` file:

Configuration Key	Value
`heimdall > repository`	`<aws_account_id>.dkr.ecr.<region>.amazonaws.com/sum/sum-heimdall`
`heimdall > tag`	`<heimdall-tag>`
`agent > repository`	`<aws_account_id>.dkr.ecr.<region>.amazonaws.com/sum/sum-chrome-agent`
`agent > tag`	`<agent-tag>`
`shepherd > url`	`Shepherd URL`
`shepherd > credentials`	`credentials`
`shepherd > location`	`agent location`

   You can leave the rest of the values set to their defaults or configure them based on your requirements. See Configure Private Synthetic Agent for details on shepherd URL, credentials, location, and optional key-value pairs.

   > You need to replace `<aws_account_id>` and `<region>` with your account and region values.

3. To deploy the Helm chart using the above mentioned configuration, navigate to the previously downloaded `sum-psa-heimdall.tgz` file and enter:

   ```
   helm install heimdall-onprem sum-psa-heimdall.tgz --values values.yaml --namespace measurement
   ```

   All the Kubernetes resources are created in the cluster and you can use Heimdall. After a few seconds, Heimdall initializes and is visible in the Controller.
4. To verify if the pods are running, enter:

   ```
   kubectl get pods --namespace measurement
   ```

   To make any changes to the `values.yaml` after the initial deployment, navigate to the previously downloaded `sum-psa-heimdall.tgz` file and enter:

```
helm upgrade heimdall-onprem sum-psa-heimdall.tgz --values values.yaml --namespace measurement
```

To remove the deployment:

```
helm uninstall heimdall-onprem --namespace measurement
```

This is not recommended unless it required.

Monitor the Kubernetes Cluster

The Helm chart `sum-psa-monitoring.tgz` in the zip you downloaded installs the monitoring stack. This Helm chart installs kube-prometheus-stack alon g with a custom Grafana dashboard to monitor the Private Simple Synthetic Agent.

Monitoring the deployment is optional; however, we highly recommended that you monitor the cluster to periodically check its health.

Install the Monitoring Stack

1. To create a separate `monitoring` namespace, enter:

   ```
   kubectl create namespace monitoring
   ```

   To review configuration options, enter:

   ```
   helm show values sum-psa-monitoring.tgz > values.yaml
   ```

   This generates a `values.yaml` file which contains all the configuration options. To modify and pass the generated `values.yaml` file while installing the Helm chart, enter:

   ```
   helm install psa-monitoring sum-psa-monitoring.tgz --values values.yaml --namespace monitoring
   ```

2. After the monitoring stack is installed, you can Launch Grafana (which runs inside the cluster) to view the dashboard. To access Grafana from outside of the cluster, you can configure port forwarding or set up Ingress. To configure port forward to access it locally, enter:

   ```
   kubectl port-forward svc/psa-monitoring-grafana 3000:80 --namespace monitoring
   ```

3. Launch localhost:3000 from the browser and log in using the default credentials with username as `admin` and password as `prom-operator`. A dashboard named Private Simple Synthetic Agent displays and provides details about the Kubernetes cluster, Heimdall, Postgres, and running measurements.

Set up the Private Synthetic Agent in Minikube

To set up the Private Synthetic Agent in Minikube, follow the steps given below:

1. Create the Kubernetes Cluster.
2. Build and customise the Docker image.
3. Save Images to Minikube's Docker Daemon.
4. Deploy the Private Synthetic Agent.
5. Monitor the Kubernetes cluster.

> This document contains links to Kubernetes documentation. AppDynamics makes no representation as to the accuracy of Kubernetes documentation because Kubernetes controls its own documentation.

Create the Kubernetes Cluster

To create a Kubernetes cluster in Minikube:

1. Install Minikube following these instructions.
2. To start the cluster, enter:

```
minikube start --kubernetes-version=v1.18.3
```

> Minikube installs the latest version of Kubernetes available at the time of Minikube release by default. For latest information on supported versions, see `OldestKubernetesVersion` and `NewestKubernetesVersion` in constants.go.

Access the Cluster

To access the Kubernetes cluster, follow these instructions to install kubectl, a utility to interact with the cluster.

To verify that the cluster is running, enter:

```
kubectl get nodes
```

Build and Customise Image

You can download the zip file for Simple Synth PSA installation from the Appdynamics Downloads Portal or from the beta upload tool. This file contains Dockerfiles for sum-chrome-agent, sum-heimdall, and Helm charts used to install the agent and set up monitoring. To build an image for sum-chrome-agent and sum-heimdall, ensure that Docker is installed. If it is not installed, you can download and install Docker from here.

For sum-chrome-agent:

1. Unzip the zip file to access the sum-chrome-agent directory.
2. Navigate to the directory and run the following command:

```
docker build -f Dockerfile-PSA -t sum-chrome-agent:<agent-tag> .
```

For sum-heimdall:

1. Unzip the zip file to access the sum-heimdall directory.
2. Navigate to this directory and run the following command:

```
docker build -f Dockerfile-PSA -t sum-heimdall:<heimdall-tag> .
```

> You can use any value for <heimdall-tag> and <agent-tag>, but ensure that you use the same value in the subsequent steps.

Add Custom Python Libraries

In addition to the available standard set of libraries, you can add custom Python libraries to the agent to use in scripted measurements. You build a new image based on the image you loaded as the base image.

1. Create a Dockerfile and create RUN directives to run python pip. For example, to install the library `algorithms` you can create a Dockerfile:

```
# Use the sum-chrome-agent image we just loaded as the base image
FROM sum-chrome-agent:<agent-tag>

# Install algorithm for python3 on top of that
RUN python3 -m pip install algorithms==0.1.4


# We can add more RUN directives for installing more libraries
# RUN python3 -m pip install ...
```

> You can create any number of RUN directives to install the required libraries.

2. To build the new image, enter:

```
docker build -t sum-chrome-agent:<agent-tag> - < Dockerfile
```

The newly built agent image contains the required libraries.

Save Images to Minikube's Docker Daemon

You must tag and push the images to a registry for cluster to access it. Execute the following command to save the images to Minikube docker:

```
docker save sum-heimdall:<heimdall-tag> | (eval $(minikube docker-env) && docker load)
docker save sum-chrome-agent:<agent-tag> | (eval $(minikube docker-env) && docker load)
```

Deploy the Private Synthetic Agent

The application is deployed to the cluster after the images are in the Registry. You use Helm chart to deploy and create all Kubernetes resources in the required order.

1. Install Helm following these instructions.
2. Create a new measurement namespace where Heimdall, Postgres database, and measurement pods will run.

   To create a new measurement namespace, enter:

```
kubectl create namespace measurement
```

   Using a single command, you can deploy the Helm chart which contains the deployment details. To deploy the agent, you use the Helm chart `sum-psa-heimdall.tgz` in the zip file that you downloaded previously. Before you deploy the Private Synthetic Agent, you must set some configuration options. To view the configuration options, navigate to the previously downloaded `sum-psa-heimdall.tgz` file and enter:

```
helm show values sum-psa-heimdall.tgz > values.yaml
```

   These are the configuration key value pairs that you need to edit in the `values.yaml` file:

Configuration Key	Value
heimdall > repository	sum-heimdall
heimdall > tag	<heimdall-tag>
heimdall > pullPolicy	Never
agent > repository	sum-chrome-agent
agent > tag	<agent-tag>
shepherd > url	Shepherd URL
shepherd > credentials	credentials

`shepherd > location`	`agent location`

You can leave the rest of the values set to their defaults or configure them based on your requirements. See Configure Private Synthetic Agent for details on shepherd URL, credentials, location, and optional key-value pairs.

> You need to replace `<aws_account_id>` and `<region>` with your account and region values.

3. To deploy the Helm chart using the above mentioned configuration, navigate to the previously downloaded `sum-psa-heimdall.tgz` file and enter:

```
helm install heimdall-onprem sum-psa-heimdall.tgz --values values.yaml --namespace measurement
```

All the Kubernetes resources are created in the cluster and you can use Heimdall. After a few seconds, Heimdall initializes and is visible in the Controller.

4. To verify if the pods are running, enter:

```
kubectl get pods --namespace measurement
```

To make any changes to the `values.yaml` after the initial deployment, navigate to the previously downloaded `sum-psa-heimdall.tgz` file and enter:

```
helm upgrade heimdall-onprem sum-psa-heimdall.tgz --values values.yaml --namespace measurement
```

> To remove the deployment:
>
> ```
> helm uninstall heimdall-onprem --namespace measurement
> ```
>
> This is not recommended unless it required.

Monitor the Kubernetes Cluster

The Helm chart `sum-psa-monitoring.tgz` in the zip you downloaded installs the monitoring stack. This Helm chart installs kube-prometheus-stack alon g with a custom Grafana dashboard to monitor the Private Simple Synthetic Agent.

> Monitoring the deployment is optional; however, we highly recommended that you monitor the cluster to periodically check its health.

Install the Monitoring Stack

1. To create a separate `monitoring` namespace, enter:

```
kubectl create namespace monitoring
```

To review configuration options, enter:

```
helm show values sum-psa-monitoring.tgz > values.yaml
```

This generates a `values.yaml` file which contains all the configuration options. To modify and pass the generated `values.yaml` file while installing the Helm chart, enter:

```
helm install psa-monitoring sum-psa-monitoring.tgz --values values.yaml --namespace monitoring
```

2. After the monitoring stack is installed, you can Launch Grafana (which runs inside the cluster) to view the dashboard. To access Grafana from outside of the cluster, you can configure port forwarding or set up Ingress. To configure port forward to access it locally, enter:

```
kubectl port-forward svc/psa-monitoring-grafana 3000:80 --namespace monitoring
```

3. Launch localhost:3000 from the browser and log in using the default credentials with username as `admin` and password as `prom-operator`. A dashboard named Private Simple Synthetic Agent displays and provides details about the Kubernetes cluster, Heimdall, Postgres, and running measurements.

Set up the Private Synthetic Agent in Azure AKS

To set up the Private Synthetic Agent in Azure AKS, perform the following:

1. Create the Kubernetes Cluster.
2. Build and customise the Docker image..
3. Tag and push images to the Registry.
4. Deploy the Private Synthetic Agent.
5. Monitor the Kubernetes cluster.

This document contains links to Azure CLI documentation. AppDynamics makes no representation as to the accuracy of Azure CLI documentation because Azure CLI controls its own documentation.

Create the Kubernetes Cluster

To create a Kubernetes cluster in Azure AKS:

1. Install and authenticate Azure CLI.
2. To create a resource group, enter

```
RESOURCE_GROUP=heimdall-onprem
az group create --name $RESOURCE_GROUP --location eastus
```

3. To create a container registry, enter:

```
ACR_NAME=heimdallonprem
az acr create --resource-group $RESOURCE_GROUP --name $ACR_NAME --sku Basic
```

4. To create a cluster, enter:

```
CLUSTER_NAME=heimdall-onprem
az aks create \
    --resource-group $RESOURCE_GROUP \
    --name $CLUSTER_NAME \
    --enable-managed-identity \
    --kubernetes-version 1.18.10 \
    --node-count 4 \
    --node-vm-size Standard_D8s_v3 \
    --generate-ssh-keys \
    --attach-acr $ACR_NAME
```

The `node-vm-size` and `node-count` in the above code are selected according to the recommended configuration type. You can specify a configuration of your choice with a different type and number of nodes. See node-vm-size.

Access the Cluster

To access the Kubernetes cluster, follow these instructions to install kubectl, a utility to interact with the cluster.

To verify that the cluster is running, enter:

```
kubectl get nodes
```

Build and Customise the Docker Image

You can download the zip file for Simple Synth PSA installation from the Appdynamics Downloads Portal or from the beta upload tool. This file contains Dockerfiles for sum-chrome-agent, sum-heimdall, and Helm charts used to install the agent and set up monitoring. To build an image for sum-chrome-agent and sum-heimdall, ensure that Docker is installed. If it is not installed, you can download and install Docker from here.

For sum-chrome-agent:

1. Unzip the zip file to access the sum-chrome-agent directory.
2. Navigate to the directory and run the following command:

```
docker build -f Dockerfile-PSA -t sum-chrome-agent:<agent-tag> .
```

For sum-heimdall:

1. Unzip the zip file to access the sum-heimdall directory.
2. Navigate to this directory and run the following command:

```
docker build -f Dockerfile-PSA -t sum-heimdall:<heimdall-tag> .
```

You can use any value for <heimdall-tag> and <agent-tag>, but ensure that you use the same value in the subsequent steps.

Add Custom Python Libraries

In addition to the standard set of libraries, you can add the custom Python libraries to the agent for the scripted measurements. To add the custom Python libraries, build an image using the downloaded base image.

1. Create a Dockerfile and create `RUN` directives to run python pip. For example, to install the library `algorithms` you can create a Dockerfile:

```
# Use the sum-chrome-agent image we just loaded as the base image
FROM sum-chrome-agent:<agent-tag>

# Install algorithm for python3 on top of that
RUN python3 -m pip install algorithms==0.1.4


# We can add more RUN directives for installing more libraries
# RUN python3 -m pip install ...
```

   You can create any number of `RUN` directives to install the required libraries.

2. To build the new image, enter:

```
docker build -t sum-chrome-agent:<agent-tag> - < Dockerfile
```

   The newly built agent image contains the required libraries.

Tag and Push Images to the Registry

You must tag and push the images to a registry for cluster to access it. You have to use the `ACR_NAME` environment variable while creating the cluster.

To tag the images, enter:

```
ACR_LOGIN_SERVER=$ACR_NAME.azurecr.io
docker tag sum-heimdall:<heimdall-tag> $ACR_LOGIN_SERVER/sum-heimdall:<heimdall-tag>
docker tag sum-chrome-agent:<agent-tag> $ACR_LOGIN_SERVER/sum-chrome-agent:<agent-tag>
```

You need to replace `<aws_account_id>` & `<region>` with your account id and region values.

To push the images, enter:

```
az acr login --name $ACR_NAME
docker push $ACR_LOGIN_SERVER/sum-heimdall:<heimdall-tag>
docker push $ACR_LOGIN_SERVER/sum-chrome-agent:<agent-tag>
```

Deploy the Private Synthetic Agent

The application is deployed to the cluster after the images are in the Registry. You use Helm chart to deploy and create all Kubernetes resources in the required order.

1. Install Helm following the instructions here.
2. Create a new measurement namespace where Heimdall, postgres database, and measurement pods will run.

   To create measurement namespace:

   ```
   kubectl create namespace measurement
   ```

   Using a single command, you can deploy the Helm chart which contains the deployment details. To deploy the agent, you use the Helm chart `sum-psa-heimdall.tgz` in the zip file that you downloaded previously. Before you deploy the Private Synthetic Agent, you must set some configuration options. To view the configuration options, navigate to the previously downloaded `sum-psa-heimdall.tgz` file and enter:

   ```
   helm show values sum-psa-heimdall.tgz > values.yaml
   ```

   These are the configuration key value pairs that you need to edit in the `values.yaml` file:

Configuration Key	Value
`heimdall > repository`	`$ACR_LOGIN_SERVER/sum-heimdall`
`heimdall > tag`	`<heimdall-tag>`
`agent > repository`	`$ACR_LOGIN_SERVER/sum-chrome-agent`
`agent > tag`	`<agent-tag>`
`shepherd > url`	`Shepherd URL`
`shepherd > credentials`	`credentials`
`shepherd > location`	`agent location`

   You can leave the rest of the values set to their defaults or configure them based on your requirements. See Configure Private Synthetic Agent for details on shepherd URL, credentials, location, and optional key-value pairs.

   > You need to replace `<aws_account_id>` and `<region>` with your account and region values.

3. To deploy the Helm chart using the above mentioned configuration, navigate to the previously downloaded `sum-psa-heimdall.tgz` file and enter:

   ```
   helm install heimdall-onprem sum-psa-heimdall.tgz --values values.yaml --namespace measurement
   ```

   All the Kubernetes resources are created in the cluster and you can use Heimdall. After a few seconds, Heimdall initializes and is visible in the Controller.
4. To verify if the pods are running, enter:

   ```
   kubectl get pods --namespace measurement
   ```

   To make any changes to the `values.yaml` after the initial deployment, navigate to the previously downloaded `sum-psa-heimdall.tgz` file and enter:

   ```
   helm upgrade heimdall-onprem sum-psa-heimdall.tgz --values values.yaml --namespace measurement
   ```

   > To remove the deployment:
   >
   > ```
   > helm uninstall heimdall-onprem --namespace measurement
   > ```
   >
   > This is not recommended unless it required.

Monitor the Kubernetes Cluster

The Helm chart `sum-psa-monitoring.tgz` in the zip you downloaded installs the monitoring stack. This Helm chart installs kube-prometheus-stack alon g with a custom Grafana dashboard to monitor the Private Simple Synthetic Agent.

> Monitoring the deployment is optional; however, we highly recommended that you monitor the cluster to periodically check its health.

Install the Monitoring Stack

1. To create a separate `monitoring` namespace, enter:

   ```
   kubectl create namespace monitoring
   ```

   To review configuration options, enter:

   ```
   helm show values sum-psa-monitoring.tgz > values.yaml
   ```

   This generates a `values.yaml` file which contains all the configuration options. To modify and pass the generated `values.yaml` file while installing the Helm chart, enter:

   ```
   helm install psa-monitoring sum-psa-monitoring.tgz --values values.yaml --namespace monitoring
   ```

2. After the monitoring stack is installed, you can Launch Grafana (which runs inside the cluster) to view the dashboard. To access Grafana from outside of the cluster, you can configure port forwarding or set up Ingress. To configure port forward to access it locally, enter:

   ```
   kubectl port-forward svc/psa-monitoring-grafana 3000:80 --namespace monitoring
   ```

3. Launch localhost:3000 from the browser and log in using the default credentials with username as `admin` and password as `prom-operator`. A dashboard named Private Simple Synthetic Agent displays and provides details about the Kubernetes cluster, Heimdall, Postgres, and running measurements.

Set up the Private Synthetic Agent in Bare Metal K8s

To set up the Private Synthetic Agent in Bare Metal K8s, perform the following:

1. Create the Kubernetes Cluster.
2. Build and customise the Docker image.
3. Save Images to Minikube's Docker Daemon.
4. Deploy the Private Synthetic Agent.
5. Monitor the Kubernetes cluster.

> This document contains links to AWS CLI documentation. AppDynamics makes no representation as to the accuracy of AWS CLI documentation because AWS CLI controls its own documentation.

Create the Kubernetes Cluster

> You can use kops to create your own managed Kubernetes cluster on AWS. If you want to create self-managed cluster on a different cloud or your own datacenter, you might want to look into other tools like Kubeadm or Kubespray. See installing Kubernetes with deployment tools.

To create a Kubernetes cluster in Bare Metal K8s:

1. Install and configure AWS CLI.
2. To create IAM Role, enter.

```
aws iam create-group --group-name kops

aws iam attach-group-policy --policy-arn arn:aws:iam::aws:policy/AmazonEC2FullAccess --group-name kops
aws iam attach-group-policy --policy-arn arn:aws:iam::aws:policy/AmazonRoute53FullAccess --group-name
kops
aws iam attach-group-policy --policy-arn arn:aws:iam::aws:policy/AmazonS3FullAccess --group-name kops
aws iam attach-group-policy --policy-arn arn:aws:iam::aws:policy/IAMFullAccess --group-name kops
aws iam attach-group-policy --policy-arn arn:aws:iam::aws:policy/AmazonVPCFullAccess --group-name kops

aws iam create-user --user-name kops

aws iam add-user-to-group --user-name kops --group-name kops

aws iam create-access-key --user-name kops
```

3. Configure AWS CLI credentials using `aws configure` with the access key obtained in the previous step.
4. Based on your platform, install kops following these instructions.
5. To create S3 bucket, enable versioning by entering:

```
bucket_name=heimdall-onprem-kops-state-store
aws s3api create-bucket \
--bucket ${bucket_name} \
--create-bucket-configuration LocationConstraint=us-west-2


aws s3api put-bucket-versioning --bucket ${bucket_name} --versioning-configuration Status=Enabled
```

6. To create cluster, enter:

```
export KOPS_CLUSTER_NAME=heimdall-onprem.k8s.local
export KOPS_STATE_STORE=s3://${bucket_name}
export KOPS_KUBERNETES_VERSION=1.18.10


kops create cluster \
--node-count=4 \
--node-size=t3.2xlarge \
--zones=us-west-2a \
--kubernetes-version=${KOPS_KUBERNETES_VERSION} \
--name=${KOPS_CLUSTER_NAME}


kops update cluster --name ${KOPS_CLUSTER_NAME} --yes
```

The `node-size` and `node-count` in the above code snippet are selected according to recommended configuration type. You can specify a configuration of your choice with a different type and number of nodes. See EC2 instance types.

7. To validate if the cluster is running (might take some time for cluster to set up and run), enter:

```
kops validate cluster
```

Access the Cluster

To access the Kubernetes cluster, follow these instructions to install kubectl, a utility to interact with the cluster.

To verify that the cluster is running, enter:

```
kubectl get nodes
```

Build and Customise the Docker Image

You can download the zip file for Simple Synth PSA installation from the Appdynamics Downloads Portal or from the beta upload tool. This file contains Dockerfiles for sum-chrome-agent, sum-heimdall, and Helm charts used to install the agent and set up monitoring. To build an image for sum-chrome-agent and sum-heimdall, ensure that Docker is installed. If it is not installed, you can download and install Docker from here.

For sum-chrome-agent:

1. Unzip the zip file to access the sum-chrome-agent directory.
2. Navigate to the directory and run the following command:

```
docker build -f Dockerfile-PSA -t sum-chrome-agent:<agent-tag> .
```

For sum-heimdall:

1. Unzip the zip file to access the sum-heimdall directory.
2. Navigate to this directory and run the following command:

```
docker build -f Dockerfile-PSA -t sum-heimdall:<heimdall-tag> .
```

You can use any value for <heimdall-tag> and <agent-tag>, but ensure that you use the same value in the subsequent steps.

Add Custom Python Libraries

In addition to the available standard set of libraries, you can add custom Python libraries to the agent to use in scripted measurements. You build a new image based on the image you loaded as the base image

1. Create a Dockerfile and create `RUN` directives to run python pip. For example, to install the library `algorithms` you can create a Dockerfile:

```
# Use the sum-chrome-agent image we just loaded as the base image
FROM sum-chrome-agent:<agent-tag>

# Install algorithm for python3 on top of that
RUN python3 -m pip install algorithms==0.1.4


# We can add more RUN directives for installing more libraries
# RUN python3 -m pip install ...
```

You can create any number of `RUN` directives to install the required libraries.

2. To build the new image, enter:

```
docker build -t sum-chrome-agent:<agent-tag> - < Dockerfile
```

The newly built agent image contains the required libraries.

Tag and Push Images to the Registry

You must tag and push the images to a registry for cluster to access it. It can be done in the following ways:

Bare Metal K8S using EC2

Vanilla K8S runs on AWS infrastructure. As kops create and assign appropriate roles to the cluster nodes, they can directly access Elastic Container Registry (ECR), without any other configuration.

To tag images, enter:

```
docker tag sum-heimdall:<heimdall-tag> <aws_account_id>.dkr.ecr.<region>.amazonaws.com/sum/sum-heimdall:
<heimdall-tag>
docker tag sum-chrome-agent:<agent-tag> <aws_account_id>.dkr.ecr.<region>.amazonaws.com/sum/sum-chrome-agent:
<agent-tag>
```

Replace `<aws_account_id>` and `<region>` with your account and region values.

To create repositories, enter:

```
aws ecr create-repository --repository-name sum/sum-heimdall
aws ecr create-repository --repository-name sum/sum-chrome-agent
```

To push the images, enter:

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-stdin <aws_account_id>.
dkr.ecr.<region>.amazonaws.com
docker push <aws_account_id>.dkr.ecr.<region>.amazonaws.com/sum/sum-heimdall:<heimdall-tag>
docker push <aws_account_id>.dkr.ecr.<region>.amazonaws.com/sum/sum-chrome-agent:<agent-tag>
```

Bare Metal K8S using Private Registry

If you are managing your own Kubernetes cluster which is not on AWS, then you must deploy your own registry server. See deploy a registry server.

To tag images, enter:

```
docker tag sum-heimdall:<heimdall-tag> <REGISTRY_HOST>:<REGISTRY_PORT>/sum-heimdall:<heimdall-tag>
docker tag sum-chrome-agent:<agent-tag> <REGISTRY_HOST>:<REGISTRY_PORT>/sum-chrome-agent:<agent-tag>
```

Replace <REGISTRY_HOST> and <REGISTRY_PORT> to what you configured while deploying the registry.

To push the images, enter:

```
docker login <REGISTRY_HOST>:<REGISTRY_PORT>
docker push <REGISTRY_HOST>:<REGISTRY_PORT>/sum-heimdall:<heimdall-tag>
docker push <REGISTRY_HOST>:<REGISTRY_PORT>/sum-chrome-agent:<agent-tag>
```

Deploy the Private Synthetic Agent

The application is deployed to the cluster after the images are in the Registry. You use Helm chart to deploy and create all Kubernetes resources in the required order.

1. Install Helm following these instructions.
2. Create a new measurement namespace where Heimdall, Postgres database, and measurement pods will run.

   To create a new measurement namespace, enter:

   ```
   kubectl create namespace measurement
   ```

   Using a single command, you can deploy the Helm chart which contains the deployment details. To deploy the agent, you use the Helm chart `sum-psa-heimdall.tgz` in the zip file that you downloaded previously. Before you deploy the Private Synthetic Agent, you must set some configuration options. To view the configuration options, navigate to the previously downloaded `sum-psa-heimdall.tgz` file and enter:

   ```
   helm show values sum-psa-heimdall.tgz > values.yaml
   ```

   These are the configuration key value pairs that you need to edit in the `values.yaml` file:

   Using EC2

Configuration Key	Value
heimdall > repository	<aws_account_id>.dkr.ecr.<region>.amazonaws.com/sum/sum-heimdall
heimdall > tag	<heimdall-tag>
agent > repository	<aws_account_id>.dkr.ecr.<region>.amazonaws.com/sum/sum-chrome-agent
agent > tag	<agent-tag>
shepherd > url	Shepherd URL
shepherd > credentials	credentials
shepherd > location	agent location

   Using Private Registry

Configuration Key	Value
heimdall > repository	<REGISTRY_HOST>:<REGISTRY_PORT>/sum-heimdall
heimdall > tag	<heimdall-tag>
agent > repository	<REGISTRY_HOST>:<REGISTRY_PORT>/sum-chrome-agent
agent > tag	<agent-tag>
privateRegistry	true
shepherd > url	Shepherd URL
shepherd > credentials	credentials
shepherd > location	agent location

After configuring using Private Registry

Create registry credentials:

```
kubectl create secret docker-registry regcred --docker-server=<REGISTRY_HOST>:<REGISTRY_PORT> --
docker-username=<your-name> --docker-password=<your-pword> --docker-email=<your-email> --
namespace measurement
```

Patch the default service account of measurement namespace to use the `regcred` registry credentials:

```
kubectl patch serviceaccount default -p '{"imagePullSecrets": [{"name": "regcred"}]}' --
namespace measurement
```

You can leave the rest of the values set to their defaults or configure them based on your requirements. See Configure Private Synthetic Agent for details on shepherd URL, credentials, location, and optional key-value pairs.

You need to replace `<aws_account_id>` and `<region>` with your account and region values.

3. To deploy the Helm chart using the above mentioned configuration, navigate to the previously downloaded `sum-psa-heimdall.tgz` file and enter:

```
helm install heimdall-onprem sum-psa-heimdall.tgz --values values.yaml --namespace measurement
```

All the Kubernetes resources are created in the cluster and you can use Heimdall. After a few seconds, Heimdall initializes and is visible in the Controller.

4. To verify if the pods are running, enter:

```
kubectl get pods --namespace measurement
```

To make any changes to the `values.yaml` after the initial deployment, navigate to the previously downloaded `sum-psa-heimdall.tgz` file and enter:

```
helm upgrade heimdall-onprem sum-psa-heimdall.tgz --values values.yaml --namespace measurement
```

To remove the deployment:

```
helm uninstall heimdall-onprem --namespace measurement
```

This is not recommended unless it required.

Monitor the Kubernetes Cluster

The Helm chart `sum-psa-monitoring.tgz` in the zip you downloaded installs the monitoring stack. This Helm chart installs kube-prometheus-stack along with a custom Grafana dashboard to monitor the Private Simple Synthetic Agent.

Monitoring the deployment is optional; however, we highly recommended that you monitor the cluster to periodically check its health.

Install the Monitoring Stack

1. To create a separate `monitoring` namespace, enter:

```
kubectl create namespace monitoring
```

To review configuration options, enter:

```
helm show values sum-psa-monitoring.tgz > values.yaml
```

This generates a `values.yaml` file which contains all the configuration options. To modify and pass the generated `values.yaml` file while installing the Helm chart, enter:

```
helm install psa-monitoring sum-psa-monitoring.tgz --values values.yaml --namespace monitoring
```

2. After the monitoring stack is installed, you can Launch Grafana (which runs inside the cluster) to view the dashboard. To access Grafana from outside of the cluster, you can configure port forwarding or set up Ingress. To configure port forward to access it locally, enter:

```
kubectl port-forward svc/psa-monitoring-grafana 3000:80 --namespace monitoring
```

3. Launch localhost:3000 from the browser and log in using the default credentials with username as `admin` and password as `prom-operator`. A dashboard named Private Simple Synthetic Agent displays and provides details about the Kubernetes cluster, Heimdall, Postgres, and running measurements.

Configure Private Synthetic Agent

This page describes how to configure the Private Synthetic Agent (PSA).

In-Script Metadata Configuration

When you run Synthetic Jobs as scripted measurements, you can specify certain configuration options using in-script metadata. You must include these yaml based configurations at the beginning of the script in the form of a comment block.

Set Custom User Agent for Synthetic Scripts

To modify the default browser specific `user-agent` header to a custom `user-agent` header, add this in-script metadata at the beginning of the Synthetic script:

```
'''yml
requestHeader:
  userAgent: "custom user agent"
'''
driver.get('http://127.0.0.1:20000/green.html')
```

Include Synthetic Traffic in Browser RUM Traffic

The `user-agent` request headers in the Synthetic scripts are tagged so that the requests are not included in Browser RUM pages. You can set `tagUserAgent` to `false` to include the Synthetic traffic in the Browser RUM pages.

Enter the following in-script metadata at the beginning of the Synthetic script:

```
'''yml
requestHeader:
  userAgent: "custom user agent"
  tagUserAgent: false
'''
driver.get('http://127.0.0.1:20000/green.html')
```

Kubernetes Configuration

After you set up the Kubernetes cluster, push the images to the Docker registry, and deploy the Private Synthetic Agent using `sum-psa-heimdall.tgz` Helm chart, a `values.yaml` file is generated. Use the `values.yaml` file to dynamically configure the values while deploying the PSA.

Key-Value Pairs Configuration

You can replace the optional configuration values in the `values.yaml` file with the values in the table:

Configuration	Key-Value Pair	Description
Heimdall replica count	`heimdall > replicaCount`	Indicates the Heimdall replica count.
Heimdall Resource allocations	• `heimdallResources > min_cpu` • `heimdallResources > max_cpu` • `heimdallResources > min_mem` • `heimdallResources > max_mem`	Indicates the Heimdall Resource allocation.

Measurement Container Resource allocations	• `agentResources > min_cpu` • `agentResources > max_cpu` • `agentResources > min_mem` • `agentResources > max_mem`	Indicates the measurement container resource allocations.
Postgres DB Resource allocations	• `postgresResources > min_cpu` • `postgresResources > max_cpu` • `postgresResources > min_mem` • `postgresResources > max_mem`	Indicates the Postgres DB resource allocations.
Custom Postgres DB	• postgres > enable to false • databaseConfig >	To set up Postgres DB instance and to configure Heimdall to connect to the DB instance, ensure that the DB is running in the same namespace as Heimdall (measurement namespace).
Logging level	`logging > level to ERROR, WARN, INFO, DEBUG, or TRACE`	Indicates the logging level. The default value is `DEBUG`.
Node selectors	• `heimdallNodeSelector >` • `agentNodeSelector >`	Apply Node selectors for Heimdall and measurement containers to schedule these components on specific nodes.

Connect Agent to the Synthetic Server

When deploying the Synthetic Private Agent in the Kubernetes cluster, you must specify the agent registration information in the `values.yaml` file. Heimdall uses this information to register itself with the Synthetic Server. Once registered you can view the agent in the Controller and run measurements.

Shepherd URL

To connect the agent to a specific region, enter one of these shepherd URLs in the `values.yaml` file.

Region	Shepherd URL
us-west-2	https://synthetic.api.appdynamics.com
eu-central-1	https://fra-synthetic.eum-appdynamics.com
ap-southeast-2	https://syd-synthetic.eum-appdynamics.com
ap-south-1	https://bom-synthetic.eum-appdynamics.com

You can connect the Agent to only one Synthetic Cloud only. You cannot connect it to your on-premise Controllers and on-premise Synthetic Servers.

EUM Account and License Key

To connect the agent to the Controller, you need the EUM account and license key.

Follow the given steps to obtain this information:

1. Navigate to the SaaS Controller.
2. Click the settings icon at the top-right corner to navigate to **License > Account Usage**.
3. Scroll down to the **User Experience** section.
   Enter Account Name and License Key in the `values.yaml` file.

Configuration Key	Configuration Value
`shepherd > credentials > eumAccount`	`Account Name`

`shepherd > credentials > eumKey`	`License Key`

Location Information

To identify the location of the agent, enter the following `shepherd > location` details in the `values.yaml` file:

- `name` - A user-friendly string for the location.
- `code` - A unique alphanumeric string with four to ten characters which identifies the location where the agent is running. You must create this ID based on the location name.
- `latitude, longitude` - Navigate to the GPS Coordinates with Google Maps to get the latitude and longitude of your location.

Mobile Real User Monitoring

Related pages:

- Set Up and Access Mobile RUM

Mobile Real User Monitoring (Mobile RUM) allows you to understand your native iOS, Android, Xamarin, Cordova-based, or React Native mobile application as your end users actually use it.

It provides you with visibility into the functioning of the application itself and the application's interactions with the network it uses and any server-side applications it may talk to. This page introduces Mobile RUM.

Benefits of Mobile RUM

With Mobile RUM, you can:

- Understand and improve your mobile application's performance
  - Know when your application is slow because of networking problems. See Network Requests List.
  - Determine whether a request is slow because of your servers or because of the network connection. See Network Request Dashboard.
  - Trace an individual request from the initial user action in the mobile application through the associated business transaction(s) on the application server(s). See Network Request Snapshots.
  - Estimate the network performance you can expect for different requests, carriers, devices, and geographies by viewing current metrics. See **Usage Stats** in Mobile App Dashboard.
  - Detect slow UI issues known as "application not responding" (ANR). You can track ANRs in the Code Issues Dashboard, identify sessions where ANRs occurred, and configure ANR thresholds to define severity levels.
- Reduce crashes
  - Observe when your application crashes and what caused each crash. See Crash Analyze and Crash Snapshots.
  - Learn which environments experience the most crashes. See Crash Dashboard.
- Reduce errors
  - Observe how many network errors occur and which requests caused them. See Network Requests List and Network Request Dashboard.
  - Capture errors and exceptions from your mobile applications, so that they are displayed in the Code Issues Dashboard and the session records and details. To learn how to use the SDKs to report errors, see Report and Exceptions (iOS/Android).
- Learn about your users
  - Learn which devices and technologies most of your users are running. See **Usage Stats** in Mobile App Dashboard and Crash Dashboard.
  - View where your users are located in the world. See **Geo Dashboard** in Mobile App Dashboard.
  - Understand how users are navigating through your application and what actions they are taking. See Mobile Sessions.
  - Capture user interactions with your mobile application such as button clicks and when text is edited. See "Enable User Interaction Capture Mode" (iOS/Android) to learn how.
- Customize the data that Mobile RUM returns
  - Create new metrics and timers. See Customize the iOS Instrumentation, Customize the Android Instrumentation, Customize the Xamarin Instrumentation, and Customize the Cordova Instrumentation.
  - Collect diagnostic data from your executing code via information points. See Customize the iOS Instrumentation, Customize the Android Instrumentation, Customize the Xamarin Instrumentation, and Customize the Cordova Instrumentation.

How It Works

To use Mobile RUM, you add a small piece of highly performant code, the Mobile Agent, to the source of your mobile application. This process is called *instrumenting* and is described in Instrument iOS Applications, Instrument Android Applications, Instrument Xamarin Applications, Instrument Cordova Applications, and Instrument React Native Applications.

As your end users interact with your application, the agent collects metrics and any error information on the application's performance and sends that information to the EUM Server, where it is processed. The EUM Server then makes the data available to the AppDynamics controller UI, where it is displayed in a series of dashboards and charts.

If your application crashes, the agent also creates a *crash snapshot* with information to help you analyze what happened, including the crashed function, the source file containing the crashed function, the line number, if available, and a stack trace of the application at the time of the crash, along with various other identifying data. And if the server-side application with which your mobile app interacts is also instrumented, you can get correlated metric information for the entire round-trip request life cycle.

Data Collection

When the agent detects that something of interest has occurred in the application, it collects data describing that event and stores it in a local buffer. Typically, this data consists of approximately 200-400 bytes per event. The agent periodically flushes this local buffer by sending the events in the buffer to the EUM Server in a request called a beacon. To minimize power consumption, the agent schedules these flushes, if possible, when the application is using the network.

If a network connection to the EUM Server is not available, the agent deletes older events to prevent unbounded growth in the buffer. The maximum size of the buffer is 200 events.

If the EUM Server is unavailable for extended periods of time, the agent may delete older data to prevent unbounded growth in the buffer.

License and Enable Mobile RUM

- Mobile RUM requires a separate license from that of your application server. For information about licensing, see License Entitlements and Restrictions.
- For information on enabling or disabling Mobile RUM monitoring, see Enable Mobile RUM on Your Controller.

AppDynamics Mobile RUM allows you to investigate the performance of your mobile application. AppDynamics Mobile App allows you to monitor your Controller from your mobile device.

Set Up and Access Mobile RUM

You set up and configure Mobile RUM in two areas:

- Your AppDynamics Controller
- Your iOS, Android, Xamarin, or Cordova application

To prepare Mobile RUM you need to make changes in both these areas in a particular order.

You need to do the following:

1. Check your prerequisites
2. Review Controller capacity
3. Evaluate your mobile application
4. Complete the Getting Started Wizard
5. Instrument and verify your app
6. Configure network request naming and thresholds

Check Your Prerequisites

Before you can use AppDynamics to monitor your mobile application, you need to make sure you have the following prerequisites:

- An AppDynamics account, with access to an AppDynamics Controller
- A Mobile RUM License. See License Entitlements and Restrictions.
- Access to your mobile application source code

Review Your Controller Capacity

If you use an on-premises Controller and plan to monitor mobile applications, assess your Controller's capacity to accommodate the increase in the number of metrics that Mobile RUM will generate. The number of individual metric data points generated depends on the level of activity of your mobile applications. As a rough guide, the use of Mobile RUM can increase the number of metric data points by as much as 15 to 25K per instrumented application, if your applications are heavily accessed by mobile users. The actual number depends on how many network requests your applications receive.

> The number of separate RUM metric *names* saved in the Controller database can be larger than the kinds of individual data points saved. For example, a metric name for a metric for iOS 5 might still be in the database even if all your users have migrated away from iOS 5. So, the metric name would no longer have an impact on resource utilization, but it would count against the default limit in the Controller for metric names per application. The default limit for *names* is 200,000 for Browser RUM and 100,000 for Mobile RUM.

For more information about Controller sizing, see Hardware Requirements per Performance Profile.

Evaluate Your Mobile Application

By default, Mobile RUM can monitor your application in two ways: network requests and crash reporting.

If you want to collect information on network requests, your application must make HTTP calls using specific classes:

- An iOS application must use `NSURLConnection` or `NSURLSession` to generate network requests.
- An Android application must itself (i.e., not via an external framework) use `HttpURLConnection`, `HttpsURLConnection`, `HttpClient`, `OkHttp`, or `ch.boye.httpclientandroidlib` to generate network requests.
- Using the SDK, custom HTTP libraries can also be monitored and used by the agent itself. See Customize the iOS Instrumentation and Customize the Android Instrumentation for more information.

Even if your application does not generate network requests, or if you do not want to monitor that activity, you can use AppDynamics purely to monitor crashes. You can also extend the Mobile Agents using the SDK to collect other data, like how your application is functioning internally.

Complete the Getting Started Wizard

Use the **Getting Started Wizard** to create a mobile app group, get an EUM App Key, and guide you through the instrumentation process. Once you have successfully completed the steps in the wizard, Mobile RUM will automatically be enabled for the created mobile app group.

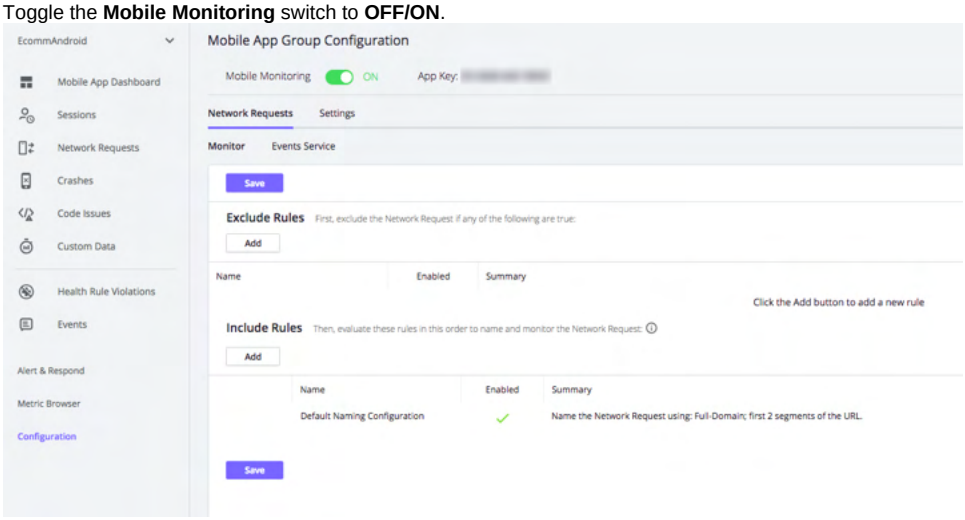
> Your Controller must have a Mobile RUM license before you can enable the Mobile RUM functionality.

Disable Mobile RUM

To later disable/enable Mobile RUM:

1. Open your mobile application.

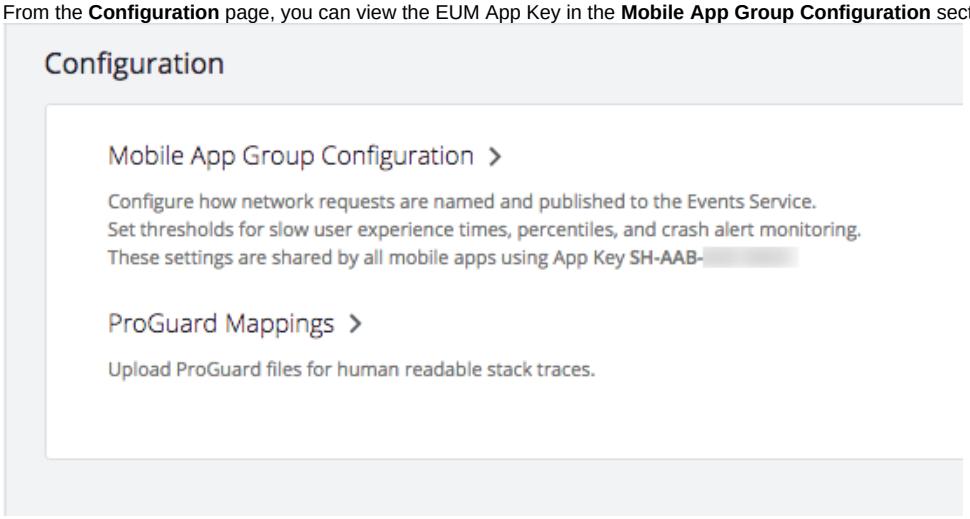
2. From the left-hand navigation bar, click **Configuration**.
3. Toggle the **Mobile Monitoring** switch to **OFF/ON**.



Get Your Application Key

To find your App Key after you've completed the **Getting Started Wizard**:

1. Open your mobile application.
2. From the left-hand navigation bar, click **Configuration**.
3. From the **Configuration** page, you can view the EUM App Key in the **Mobile App Group Configuration** section as shown below.



Instrument and Verify Your App

The method you follow to instrument your app depends on your platform:

- Instrument iOS Applications
- Instrument Android Applications
- Instrument Xamarin Applications
- Instrument Cordova Applications

Configure Network Request Naming and Thresholds

You can customize how you want the Controller to name your application's requests and what performance thresholds you want to be in effect. See Configure Mobile Network Request Naming and Thresholds.

External Access

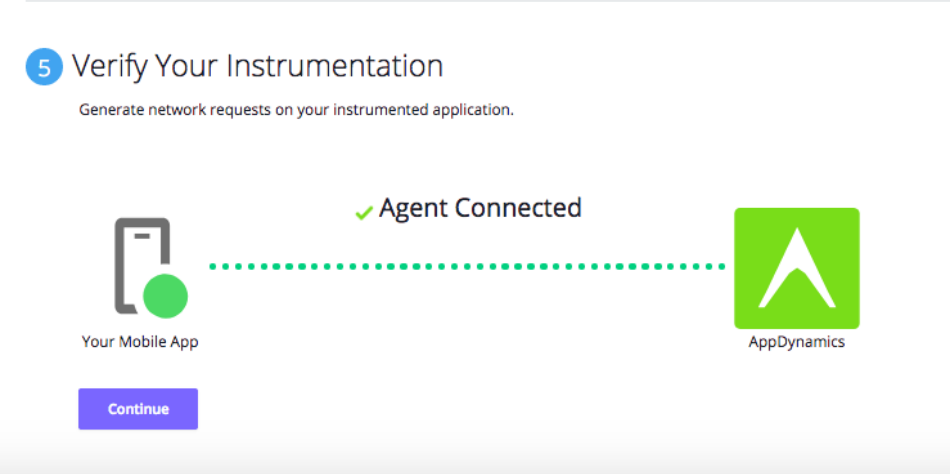
Mobile RUM is made up of several components, any or all of which can either be located on the Internet or hosted inside your own data center/network. On-premises access points are configured at installation or through the UI. If your installation requires access to any of these components on the Internet, see Access the SaaS EUM Server for more information.

Confirm the Mobile Agent Connected to the Controller

Related pages:

- Instrument iOS Applications
- Instrument Android Applications
- Instrument Xamarin Applications

After you have instrumented your application, the **Getting Started Wizard** will show you a verification that the Mobile Agent has connected to the Controller.



If you left the **Getting Started Wizard**, you can always verify the instrumentation by doing the following:

1. In the Controller UI, open **User Experience > Mobile Apps.**
2. Check the list of registered mobile applications to verify that the application is registered with the Controller. You can also use view some basic information about the app such as the health, sessions, crashes, network request information, and whether the monitoring is enabled.

3. Start monitoring your application! See Overview of the Controller UI for Mobile RUM.

Correlate Business Transactions for Mobile RUM

Related pages:

- Correlate Business Transactions for EUM
- Business Transactions

You can correlate mobile network requests with business transactions. In actuality, the correlation is made between instances of network requests (network request snapshots) and instances of business transactions (transaction snapshots).

Mobile Network Requests Correlation

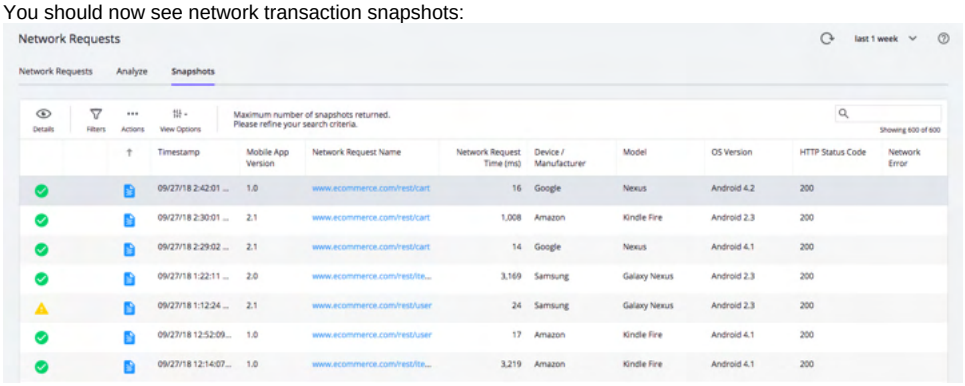
When an end user takes an action with your mobile application that requires a network request:

1. The App Agent:
    - Sends HTTP headers identifying the business transaction to the end user's mobile device.
    - Aggregates backend metrics and sends them along with the business transaction identifiers to the Controller. This serves as the content for the transaction snapshot.
2. The Mobile Agent sends metrics and the business transaction identifiers (from the HTTP header) to the EUM Server. This serves as the content for the network request snapshot.
3. The Controller fetches the metrics and business transaction identifiers from the EUM Server. Using the business transaction identifiers, the Controller correlates the network request snapshot with the transaction snapshots.

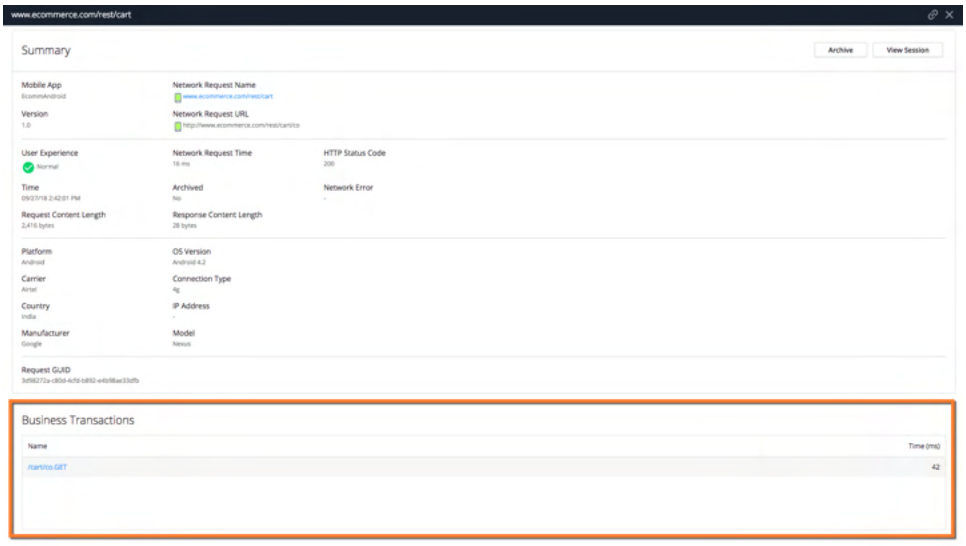
View Business Transactions Correlated with Mobile Applications

There are several ways to navigate from a mobile network request snapshot to its correlated business transaction. This procedure provides a sample navigation:

1. From the **Mobile App Dashboard**, click **Network Requests**.
2. Click **Snapshots.**
3. You should now see network transaction snapshots:



4. Double-click one of the browser snapshots to open the **Network Request Snapshot Details** dialog. If a correlated transaction snapshot exists, a link displays in the **Business Transactions** section:



5. You can then click links in the transaction snapshot or transaction snapshot itself to view corresponding pages in APM.

Monitor Your Applications with Mobile RUM

This page provides an overview of the Controller UI for Mobile RUM applications. Mobile RUM presents your application information in the following ways:

- Mobile App Dashboard: An overview dashboard, with tabs for
  - Widgets for visually displaying common metrics
  - Map-based performance display
  - Usage statistics
- Mobile Sessions: A detailed list of mobile sessions, following users as they interact with your application
- Network Requests Lists: A detailed list of network requests with dashboards
- Crashes: A dashboard displaying of application crash trends and detailed snapshots of individual crashes with stack traces
- Custom Data: A dashboard specific data you want the agent to collect for you

Access a Browser RUM Application

To see application data in the Controller UI:

1. In the top tab bar, select **User Experience**. The list of instrumented Browser and Mobile Apps displays.
2. Select the Mobile App tab.
3. Double-click a mobile application.

Mobile App Dashboard

The **Mobile App Dashboard** provides a high-level overview of how your application is performing, including the following:

- Widget-based overview
- Geo dashboard, displaying where your requests originate, including a variety of key performance metrics
- Set of aggregated usage statistics, by device, carrier, connection type, OS version, and app version

Access the Mobile App Dashboard View

1. Open the application you want.
2. From the left navigation bar of your application, select **Mobile App Dashboard**.
3. Click the tab for the view you want to access.

Overview Dashboard

The **Overview Dashboard** displays a set of configurable widgets, showing multiple graphs and lists featuring common high-level indicators of application performance. You can delete widgets, re-add them, resize them, and drag them to different locations.

Geo Dashboard

The Mobile App **Geo Dashboard** displays key performance metrics by geographic location.

The dashboard is divided into three panels:

- A main panel in the upper left that displays geographic distribution of mobile users on a map, if you clicked the map view icon, or on a grid, if you clicked the grid view icon.
- A panel on the right displaying key timings.
- A lower panel with dynamic trend graphs of KPIs.

The metrics displayed throughout the geo dashboard are for the country currently selected on the map or in the grid. For example, if you zoom down from the world view to France on the map, the panel on the right disappears and the trend graphs display data for France.

See Mobile RUM Metrics for definitions of the metrics.

Map View Labels

The map view displays load circles with labels for countries that are in the key timing metrics given in the right panel. Some countries and regions, however, are only displayed in the grid view.

Regions and Countries

Regions are subdivisions of a country, such as a state, province, or city. In the default map view, key performance metrics are displayed by country. The grid view, like the map view, by default displays key performance metrics by country, but can also be configured to show the metrics by region. Because the map view displays fewer regions than the grid view, if you do not see a region displayed in the map view, switch to the grid view.

See Browser RUM Countries and Regions by Geo Dashboard for a list of the countries and regions available in the map and grid views.

Unknown Locations in Map and Grid Views

An unknown location is one for which the agent cannot determine the country from which the request originated.

You may also see metrics reported for a location named **Anonymous Proxy**. The data for **Anonymous Proxy** represents the aggregated metrics from one or more private IP addresses that the agent cannot identify.

One of the effects of Unknown regions is that it possible for a country to display as slow (red circles) on the global map, but when you drill down to the country all its regions appear normal (green circles). Or a country may display as normal on the global map, but some sub-regions may display as slow when you drill down.

Usage Stats

This tab shows graphical and tabular breakdowns of usage by device, carrier, connection type, OS version, and app version. The breakdowns for the connection type are based on network requests. The other usage breakdowns are based on "app opens", *not* network requests. App opens are counted when a mobile application is booted or brought to the foreground. In the case of Android, the app opens occur when the method `onResume` is called for an activity. For iOS, the app opens occur when the method `applicationDidBecomeActive` is called.

The Android Agent obtains the carrier name by calling the Android SDK method `android.telephony.TelephonyManager.getSimOperatorName` and obtains the device name with the Android SDK property `android.os.Build.MANUFACTURER`. If the value for either the carrier or device name is "unknown", this indicates that the method `getSimOperatorName` or the property `MANUFACTURER` did not return a value.

Mobile Sessions

Related pages:

- EUM Data
- Mobile RUM Metrics

In most cases, a user's interaction with your app is not limited to one action in one panel view. Sessions allow you to track your users' interactions across time, as they navigate an entire session with your app.

By default, sessions begin when a user starts using your application and ends after a configurable period of user inactivity. You can also programmatically control sessions through the Mobile Agents SDKs (Android, iOS, Xamarin, Cordova). Using Mobile Sessions, you can analyze *sessionized* results from all requests, as stored in the AppDynamics Events Service.

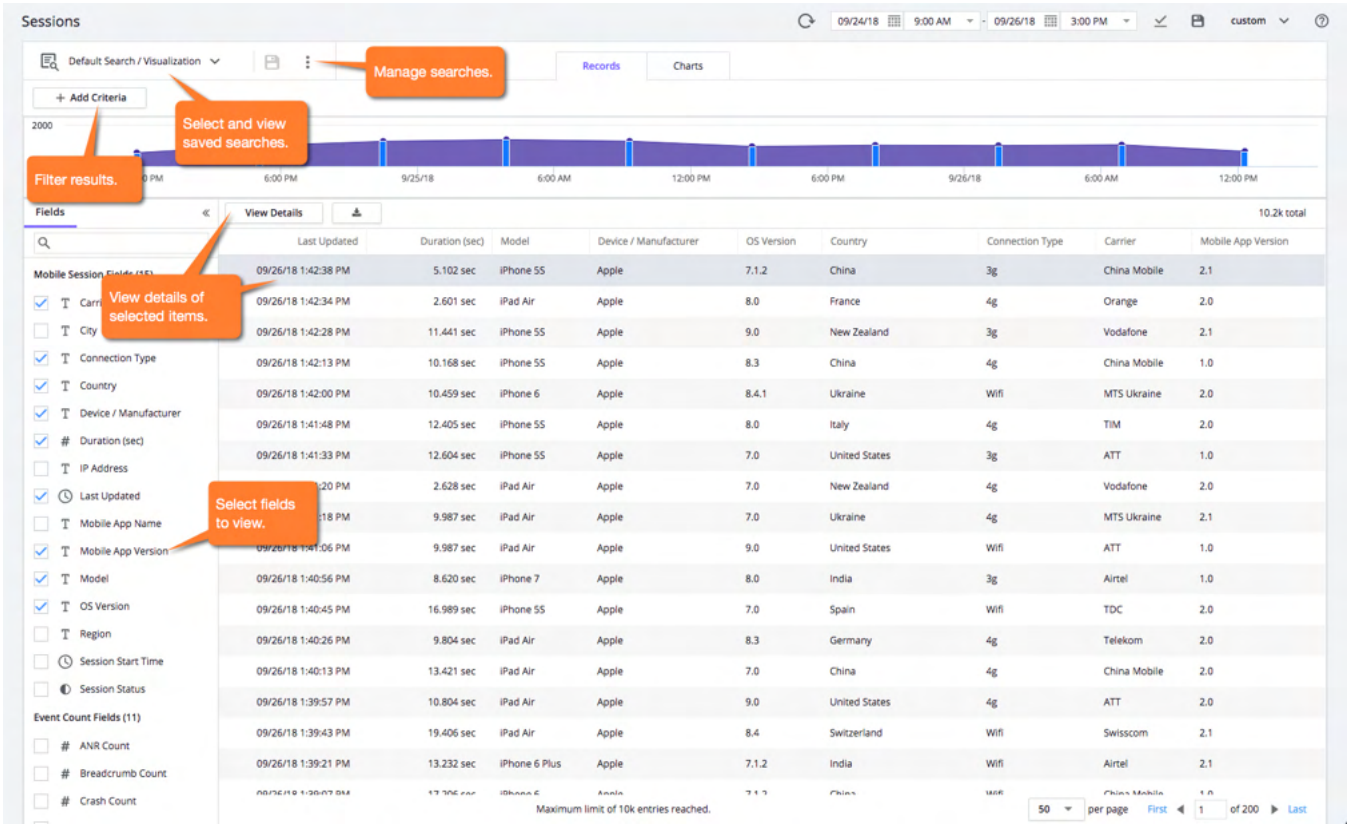
As in Network Request Analyze, the main Sessions panel is comprised of these tabs:

- Records
- Charts

You can also view detailed information about each session in the Session Details dialog.

Records

The Records tab lets you scan individual sessions and allows you to filter and sort to get exactly the data in which you are interested.

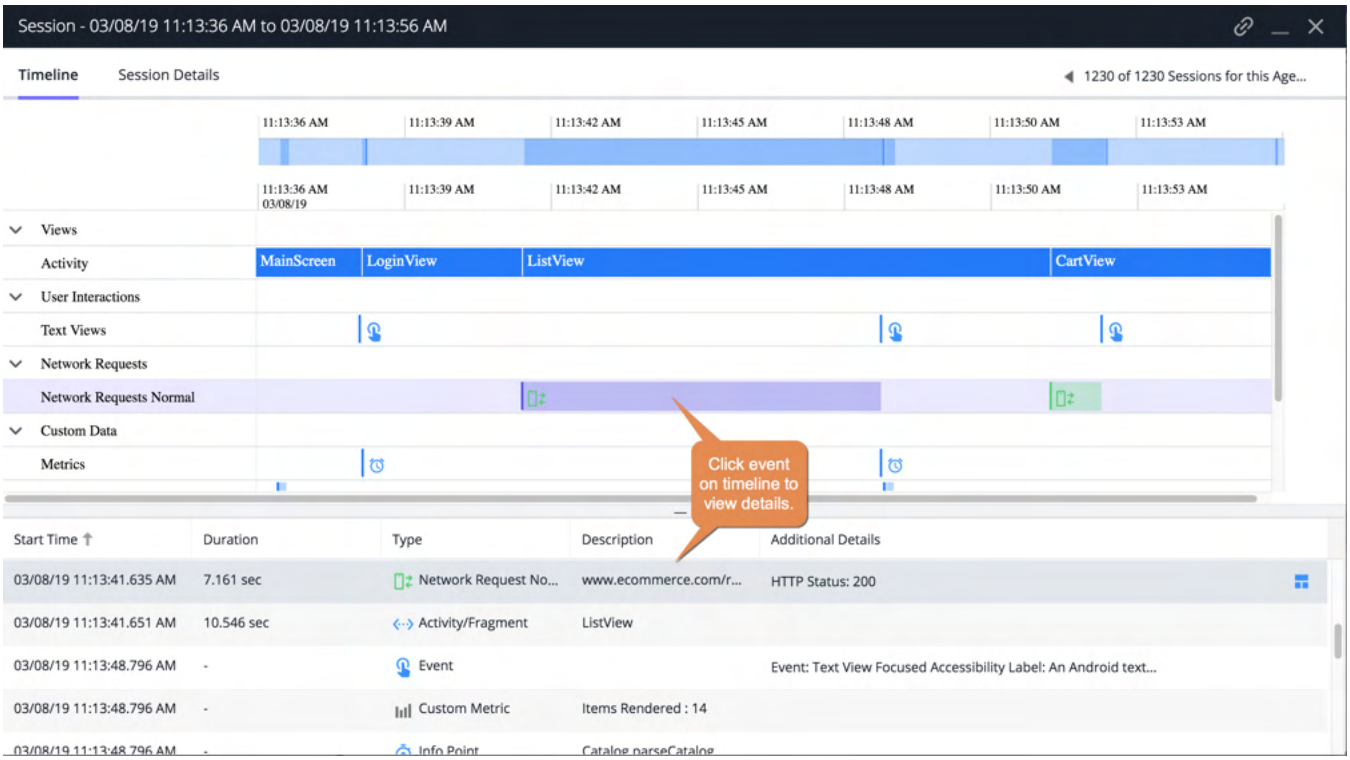


Session Details

Click **View Details** or double-click an item to see the information for a specific session.

Session Timeline

The sequence of session frames for the session is shown on the session timeline and on the bottom of the panel. Select a specific frame by clicking on an icon on the timeline to see details highlighted at the bottom of the panel. You can also select a frame row at the bottom to see it highlighted on the timeline.



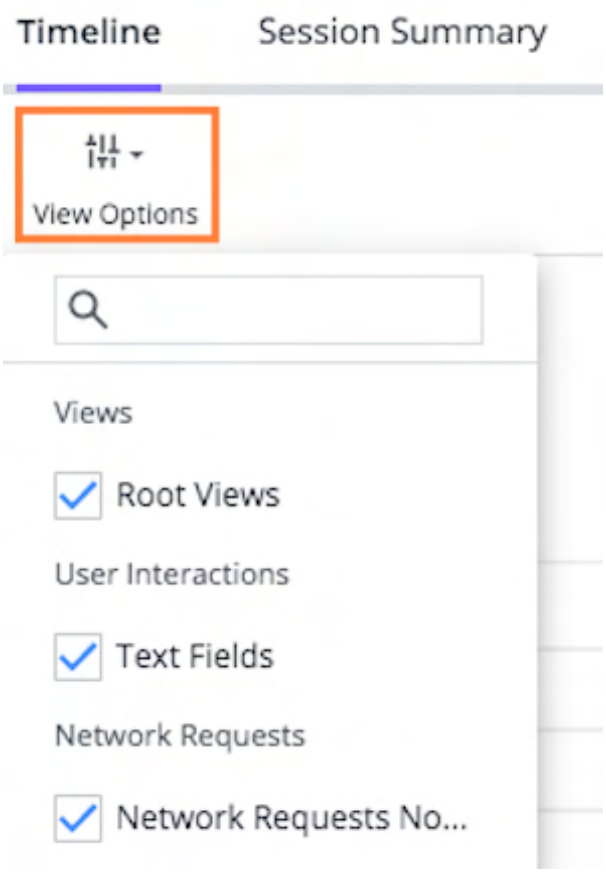
Session Details Categories and Types

Categories	Subcategories	Type	About
Views	Activity	Activity /Fragment	The current Activity in the foreground of an Android app. If the current Activity is not sufficient to understand what the user is doing, consider using the Session Frame APIs.
	RootViews	RootView	The current RootView in the foreground of the iOS App. If the current RootView is not sufficient to understand what the user is doing, consider using the Session Frame APIs.
	Session Frames	Session Frame	A manual API to label large user activities that are not captured out of the box by Activity or RootView tracking.
Screenshots	-	Screenshot	If you have configured mobile screenshots, they will appear in the timeline. They capture user events, and they can also be collected when in automatic mode.
User Interactions	Touches	Event	If you have enabled the Mobile Agent to capture UI events, you can view the UI events triggered by user interactions in the **Timeline**. See "Enable User Interaction Capture Mode" (iOS/Android) to learn how to capture UI events. If automatic screenshots are enabled, touches will be captured and include the screenshot that was captured at the same time. The touches are drawn on the screenshot.
	Buttons		
	Table Cells		
	Text Views		
	Text Fields		
Network Requests	Network Requests Normal	Network Request Normal	Network requests include the requested URL, duration, and status details of the HTTP request. To jump to more Network Requests details, click .
	Network Requests Slow	Network Requests Slow	
	Network Requests Very Slow	Network Requests Very Slow	
	Network Requests Stall	Network Requests Stall	

Errors And Warnings	Application Not Responding Issues (ANRs)	 ANR	You can view when the application was non-responsive in the timeline. ANRs are reported when the UI thread does not respond to an event for two seconds or more. To download or view the ANR Summary, click  or  , respectively.
	Crashes	 Crash	When an app crashes the subsequent activity will appear in the timeline. To download or view the Crash Summary, click  or  , respectively.
	Errors	 Error	If you have reported errors with the Mobile Agent SDKs, the errors will also appear in the timeline. See "Report Errors and Exceptions" (iOS/Android) to learn how to instrument your mobile apps to report errors and exceptions. To download or view the Error Summary, click  or  , respectively.
Custom Data	Metrics	 Custom Metric	Custom metrics display integer-based data on any metric you define in your application.
	Timers	 Custom Timer	Custom timers display how long something takes between any arbitrary start point and end point, even if they span multiple methods.
	Info Points	 Info Point	Information points display information on how a specific method in your code is performing.
	Breadcrumbs	 Breadcrumb	Breadcrumbs are short messages to mark interesting events in a session, controlled by the Breadcrumb APIs in the agent.
System Events	Connection Transition	 System Event	Connection transition indicates when a broadband transition has taken place, such as from Wifi to cellular.
	Agent Init	 System Event	Agent Init indicates when the Mobile Agent was first initialized. It can help with debugging the application lifecycle.

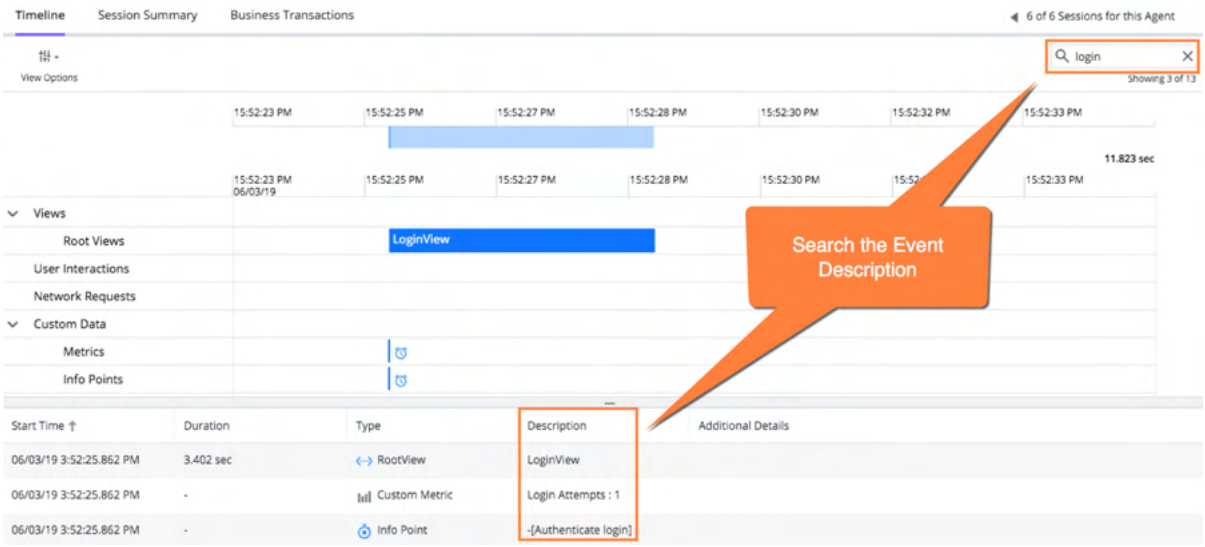
View Options

You can filter the categories in your timeline view by clicking **View Options**. Check all relevant categories for your search. All view options are checked by default.



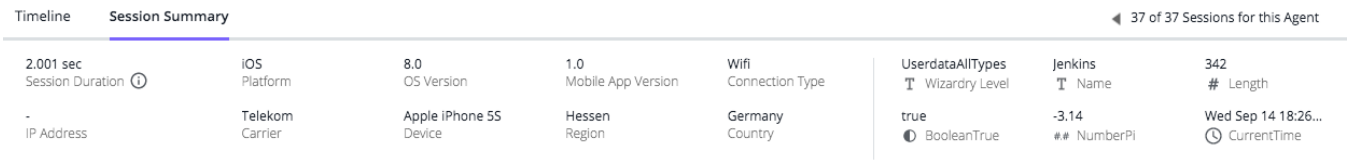
Search Events

You can search for a specific event **Description** using the search box located in the top right of the session timeline panel.



Session Summary

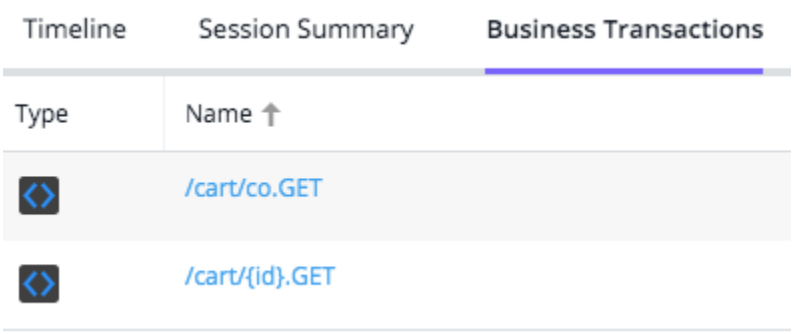
You can see a summary of the session by clicking **Session Summary**.



Business Transactions

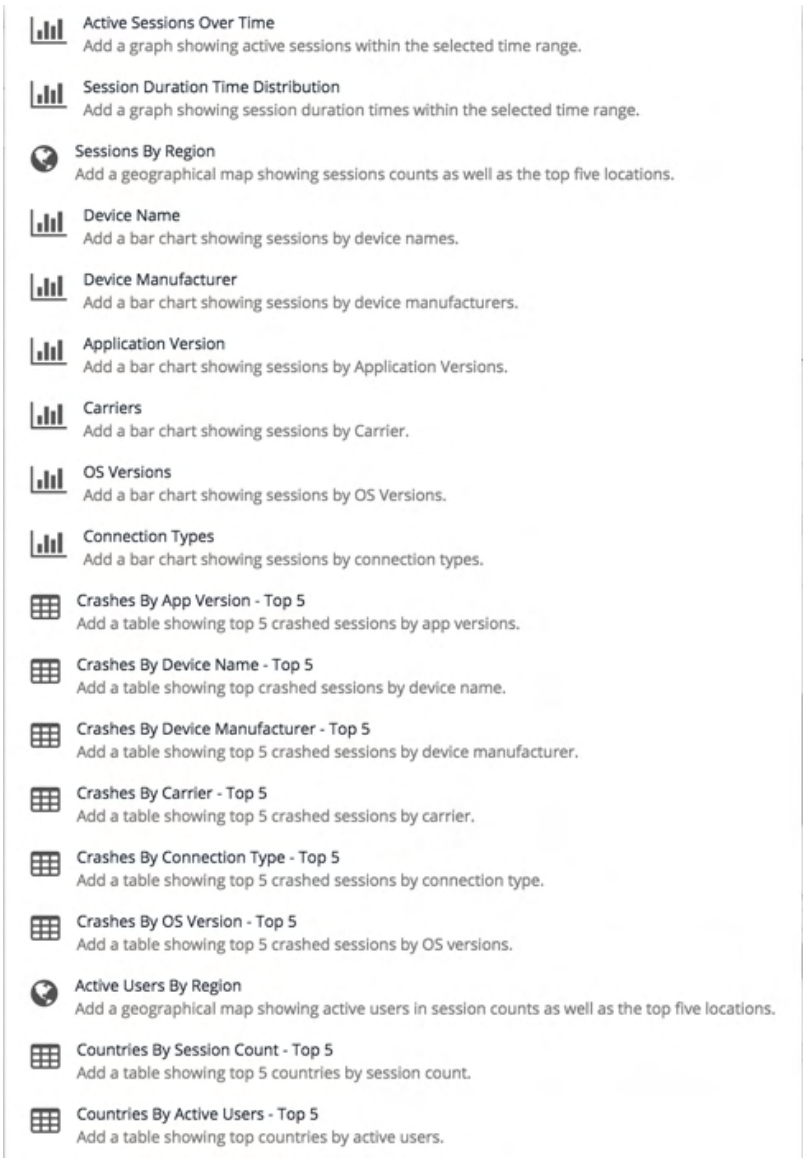
You can see a list of business transactions in the session if applicable. This tab only appears if you have business transactions in the selected session.

Click the business transaction name to redirect to the transaction flow map.



Charts

The **Charts** page provides you with a set of predefined widgets that offer visualizations of the data set you have created. As with the **Charts** pages of the other Analyze/Session features of the UI, you can delete, re-add, resize, and drag-and-drop to move all of the widgets.


Active Sessions Over Time
Add a graph showing active sessions within the selected time range.
Session Duration Time Distribution
Add a graph showing session duration times within the selected time range.
Sessions By Region
Add a geographical map showing sessions counts as well as the top five locations.
Device Name
Add a bar chart showing sessions by device names.
Device Manufacturer
Add a bar chart showing sessions by device manufacturers.
Application Version
Add a bar chart showing sessions by Application Versions.
Carriers
Add a bar chart showing sessions by Carrier.
OS Versions
Add a bar chart showing sessions by OS Versions.
Connection Types
Add a bar chart showing sessions by connection types.
Crashes By App Version - Top 5
Add a table showing top 5 crashed sessions by app versions.
Crashes By Device Name - Top 5
Add a table showing top crashed sessions by device name.
Crashes By Device Manufacturer - Top 5
Add a table showing top 5 crashed sessions by device manufacturer.
Crashes By Carrier - Top 5
Add a table showing top 5 crashed sessions by carrier.
Crashes By Connection Type - Top 5
Add a table showing top 5 crashed sessions by connection type.
Crashes By OS Version - Top 5
Add a table showing top 5 crashed sessions by OS versions.
Active Users By Region
Add a geographical map showing active users in session counts as well as the top five locations.
Countries By Session Count - Top 5
Add a table showing top 5 countries by session count.
Countries By Active Users - Top 5
Add a table showing top countries by active users.

Mobile Screenshots

Mobile screenshots provide context to the data collected in the Controller UI. They also allow you to better understand the user experience.

Screenshots enable you to:

- **Verify Presentation:** Determine if an app is displaying correctly on different devices.
- **Understand User Behavior**: Determine where users are clicking on the screen.
- **Troubleshoot Bugs:** Analyze bugs and errors by correlating code issues with screenshots. You'll be able to see what the user is doing when issues happen.

AppDynamics Version Requirements

You must use these AppDynamics software versions:

- Controller >= 4.3.2
- iOS/Android Agents >= 4.3.2
- AppDynamics Cordova Plugin >= 1

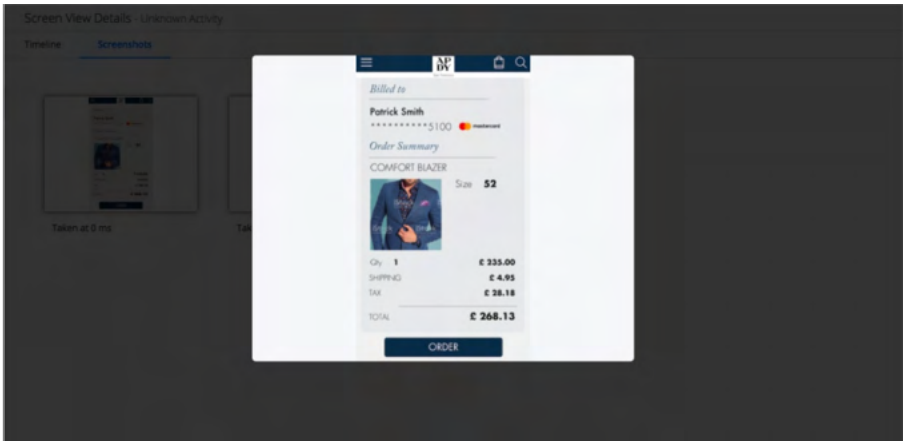
Screenshot Limitations

Mobile screenshots have the following limitations:

- Only iOS/Android/Cordova platforms are supported.
- You can enable or disable screenshots or explicitly exclude screenshots for specific screens/activities.
- You cannot set a minimum/maximum number of screenshots per session.
- Touch points are not collected unless screenshots are in automatic mode.
- The touch points will not be included in screenshots from past sessions.
- The UI Event feature is not related to screenshots or touch points in any way. Thus, the screenshot will not be directly correlated with a UI event in the **Waterfall.**

Types of Screenshots

Mobile RUM captures conventional screenshots and touch-point screenshots that capture user touch events. Both types of screenshots have timestamps, but the latter will also plot "user touches" on the panel.

Conventional Screenshots	Touch Point Screenshots
	
The conventional screenshot will simply show what your users viewing. These screenshots can be taken programmatically or by enabling automatic screenshots in the Controller UI.	The touch point screenshot shows where users tou points are only collected when automatic capture r

When Screenshots Are Taken

Screenshots can only be captured if the Mobile Agent and the Controller UI have enabled screenshots.

Once screenshots have been enabled, screenshots will be only taken if one or more of the following are true:

- The Mobile Agent uses the SDK API to manually take a screenshot.
- You have enabled automatic capture in the Controller UI.

Configure Mobile Screenshots

You have several methods for controlling when screenshots are taken. This page lists the settings for disabling/enabling and taking screenshots.

Screenshot Control Settings

This table lists the different settings for screenshots, the user tasks required for the settings, and a description of what the settings accomplish. Screenshots are enabled by default in the iOS and Android Agents, but disabled in the Controller UI. To use screenshots, you need to enable screenshots in the Controller UI.

Screenshot Setting	User Tasks	Result
Disable Screenshots From Being Taken	Perform one of the following: • Disable screenshots with the iOS /Android SDK. • Disable screenshots in the Controller UI.	**Only Disable Screenshots with the iOS/Android SDK** Screenshots are globally disabled. The Controller UI cannot override the Mobile Agent setting. **Only Disable Screenshots with the Controller UI** The Controller will disable screenshot capture in the Mobile Agent. Because the Controller setting is propagated to the Mobile Agents through the EUM Server, the Mobile Agent may take screenshots before its screenshot setting has been updated. Once the Controller setting has been applied, however, the Mobile Agent will no longer take screenshots.
Enable Screenshots	• Enable screenshots in the Controller UI.	The Mobile Agent can take screenshots, and the Controller UI will display them in the session results.
Enable Screenshots - Manual Screenshots (Wi-Fi)	• Enable screenshots in the Controller UI. • Manually take screenshots using the iOS /Android SDKs.	You call the Mobile SDK to take screenshots. The screenshots are only uploaded to the Controller when the device is using Wi-Fi.
Enable Screenshots - Automatic Screenshots (Wi-Fi)	• Enable screenshots in the Controller UI. • Configure the Controller UI to automatically take screenshots.	Screenshots are automatically taken periodically. The screenshots are only transmitted when the device is using Wi-Fi.

Enable Screenshots - Manual Screenshots (Wi-Fi / Cellular Data)	• Enable screenshots in the Controller UI. • Configure the Controller UI to allow the use of cellular data. • Manually take screenshots using the iOS /Android SDKs.	You call the Mobile SDK to take screenshots.
Enable Screenshots - Automatic Screenshots (Wi-Fi / Cellular Data)	• Enable screenshots in the Controller UI. • Configure the Controller UI to automatically take screenshots. • Configure the Controller UI to allow the use of cellular data.	Screenshots are automatically taken periodically.

Take Mobile Screenshots

Related pages:

- Customize the iOS Instrumentation
- Customize the Android Instrumentation

This page describes how to enable/disable screenshots and take manual/automatic screenshots.

Enable/Disable Screenshots

The following sections show you how to enable or disable screenshots. If you want screenshots taken automatically, see Enable Automatic Screenshots.

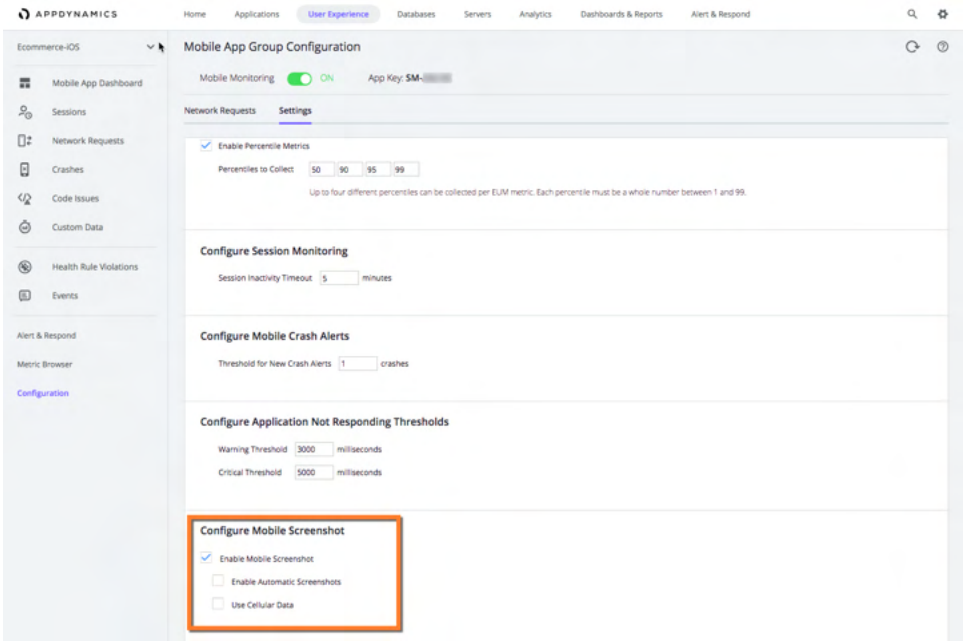
> Mobile screenshots are disabled by default for security and privacy reasons as screenshots may contain sensitive information.

Use the Mobile SDKs to Disable/Enable Screenshots

Mobile screenshots are enabled by default in the iOS and Android Agents. To disable screenshots, see the iOS SDK and Android SDK documentation.

Configure the Controller UI

1. Navigate to one of your mobile apps in the Controller UI.
2. Open the **Configuration** page.
3. Click **Mobile App Group Configuration.**
4. Scroll down to **Configure Mobile Screenshot** and check the **Enable Mobile Screenshot** check box.



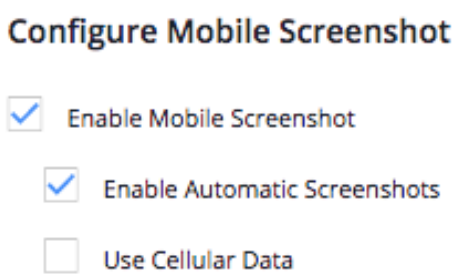
5. (Optional) If you want to allow the application to take screenshots when using cellular data, check the **User Cellular Data** check box.

Enable Automatic Screenshots

Configure the Controller UI to Automatically Take Screenshots

1. Navigate to one of your mobile apps in the Controller UI.
2. Open the **Configuration** page.
3. Click **Mobile App Group Configuration.**
4. Scroll down to **Configure Mobile Screenshot** and confirm that the **Enable Mobile Screenshot** checkbox is checked.

5. Check the **Enable Automatic Screenshots** check box.



6. (Optional) If you want to allow the application to take screenshots when using cellular data, check the **User Cellular Data** check box.

Programmatically Take Screenshots

You can use the Android or iOS SDK to programmatically take screenshots. See:

- Android SDK: Configure and Take Screenshots
- iOS SDK: Configure and Take Screenshots

View Mobile Screenshots

Mobile screenshots are included in the **Session Details**. You can view a timeline of the screenshots and touchpoints that were taken during the user's activity.

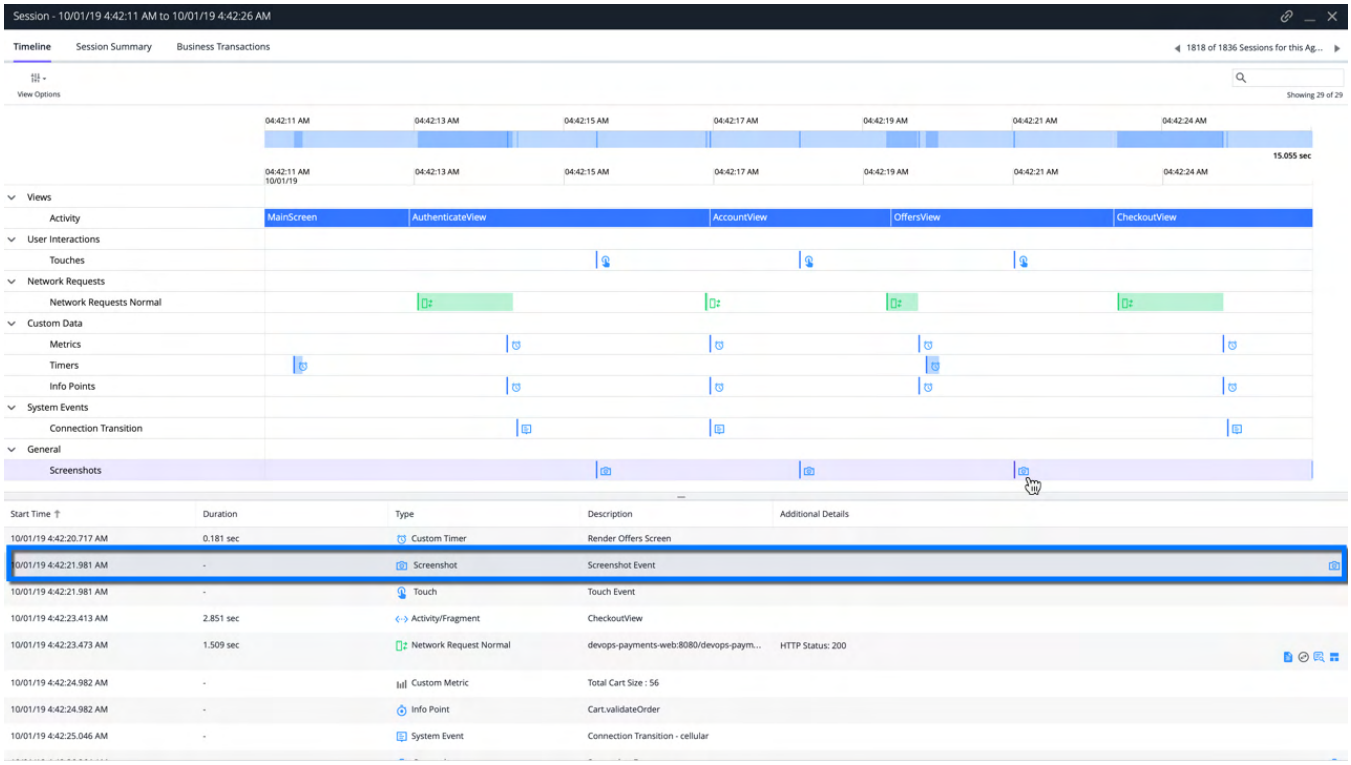
This page describes how to view screenshot and touch events in the timeline of the Timeline tab, and view thumbnails and full-size images of the screenshots and touches in the chronological event list.

Screenshot Events in the Timeline

Both conventional and touch-point screenshot events appear in the Timeline tab at the time that it was taken.

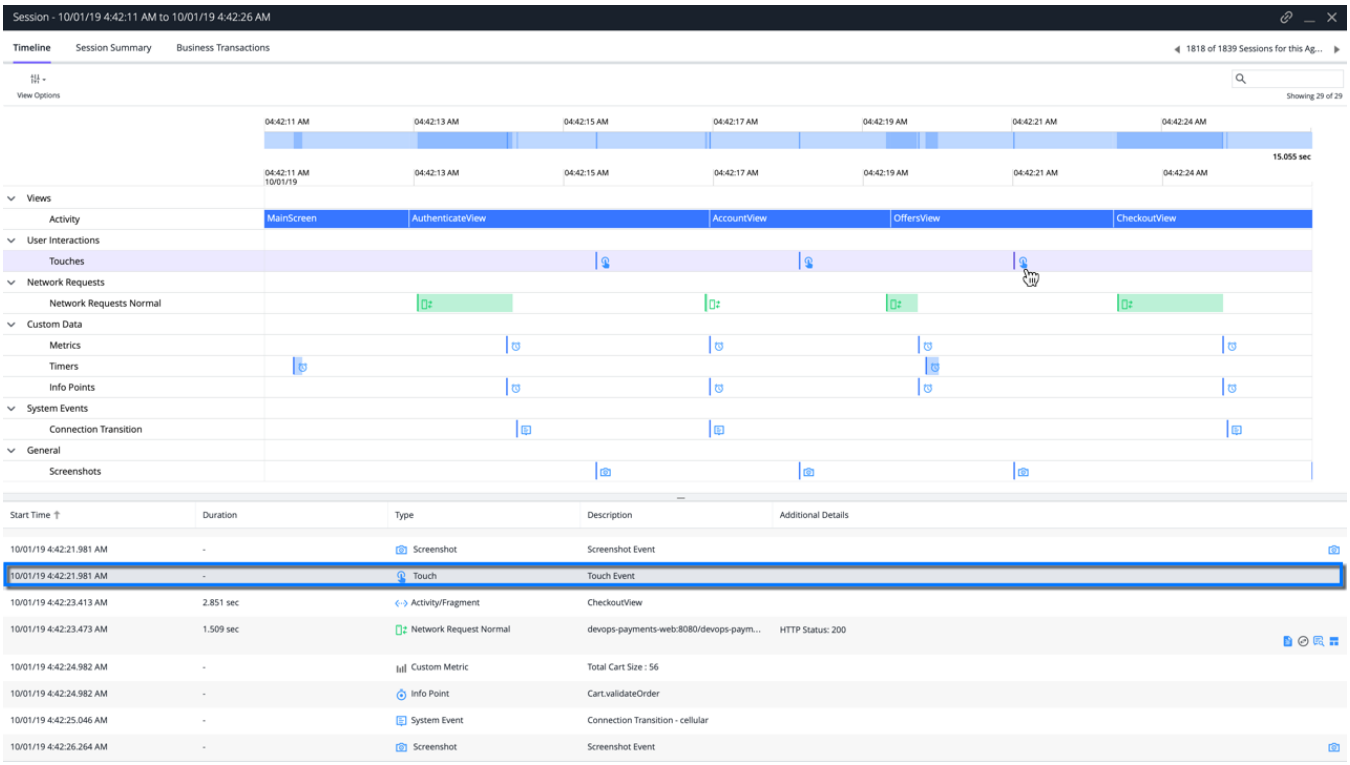
Conventional Screenshots

Conventional screenshots appear in **General > Screenshots**. Click the **camera** icon to view details of the screenshot event in the chronological event list at the bottom.



Screenshots of Touches

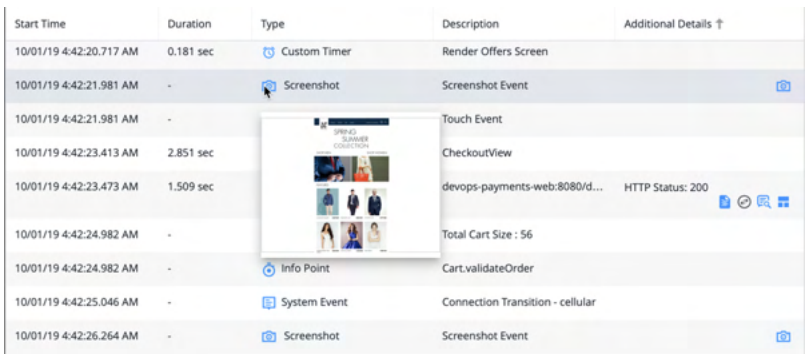
Screenshots of touches appear in **User Interactions > Touches**. Click the **screen touch** icon to view details of the touch event in the chronological event list at the bottom.



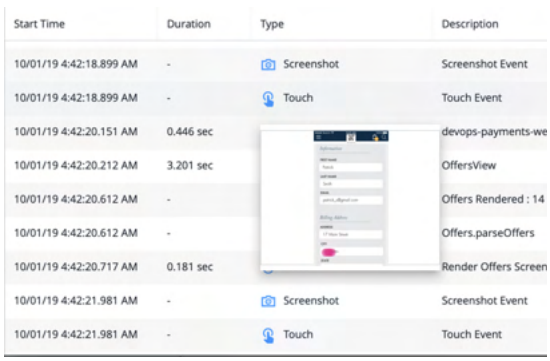
Screenshot Thumbnails

From the chronological list of events at the bottom of the page, you can hover over a screenshot or touch event to view a thumbnail. The screenshot of the touches displays the touch locations in pink.

Conventional Screenshot Thumbnail

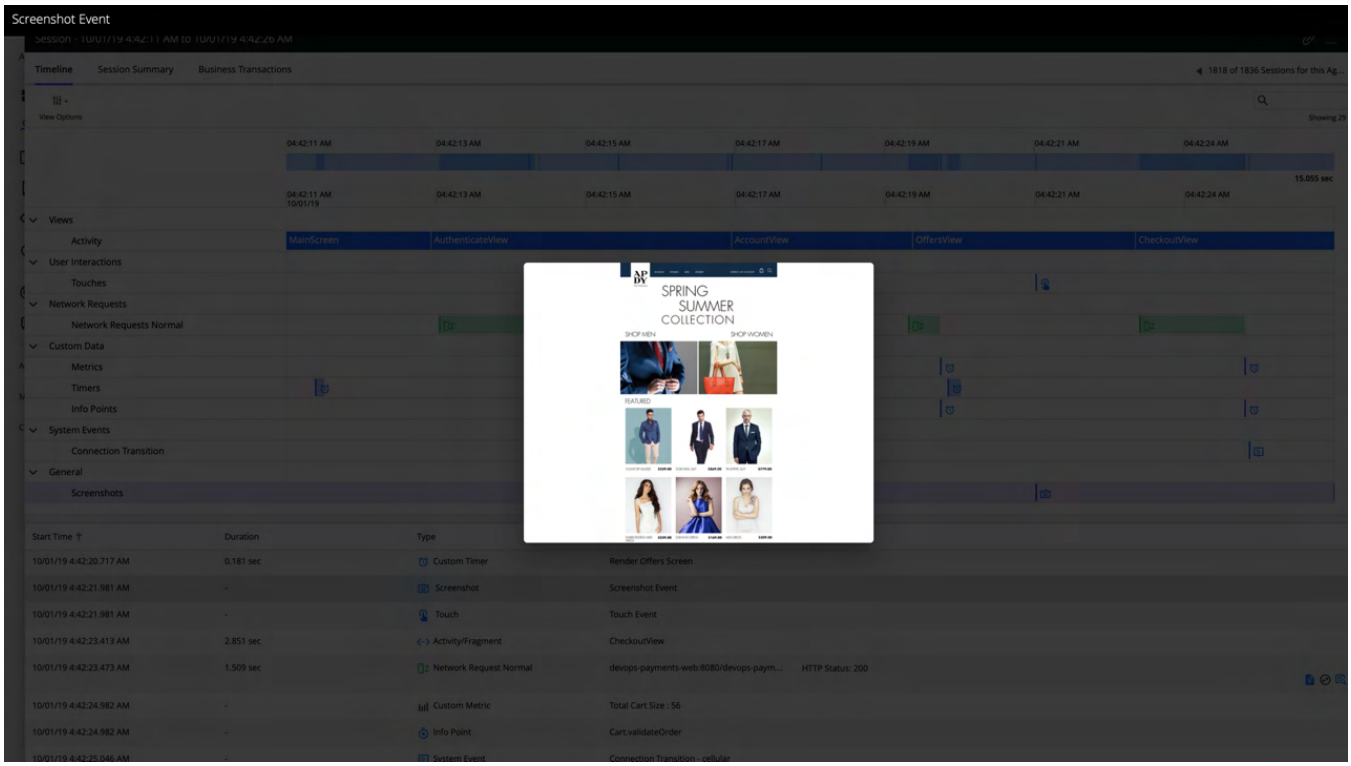


Screenshot Thumbnail of Touches



Full-Size Screenshots

To view full-size images of the conventional screenshots, click the **camera** icon on the right of the event details in the chronological list. Full-sized images are not available for screenshots of touches.


Screenshot Event
Timeline
Session Summary
Business Transactions
Views
Activity
MainScreen
AuthenticateView
AccountView
OffersView
CheckoutView
User Interactions
Touches
Network Requests
Network Requests Normal
Custom Data
Metrics
Timers
Info Points
System Events
Connection Transition
General
Screenshots
SPRING
SUMMER
COLLECTION
SHOP MEN
SHOP WOMEN
FEATURED
Start Time
Duration
Type
10/01/19 4:42:20.717 AM
0.181 sec
Custom Timer
Render Offers Screen
10/01/19 4:42:21.981 AM
Screenshot
Screenshot Event
10/01/19 4:42:21.981 AM
Touch
Touch Event
10/01/19 4:42:23.413 AM
2.851 sec
Activity/Fragment
CheckoutView
10/01/19 4:42:23.473 AM
1.509 sec
Network Request Normal
devops-payments-web:8080/devops-paym...
HTTP Status: 200
10/01/19 4:42:24.982 AM
Custom Metric
Total Cart Size : 56
10/01/19 4:42:24.982 AM
Info Point
Cart.validateOrder

Mobile Health Rules

Related pages:

- Policies
- Troubleshoot Health Rule Violations

You can set mobile health rules to trigger actions when certain conditions are met or exceeded based on configured thresholds for mobile application metrics.

To configure alerts and responses, you can use the **Getting Started Wizard** or manually create alerting policies based on mobile health rules. Alerts and responses help you to anticipate and take action for problems with your mobile applications.

This page describes the two types of mobile health rules, how to select mobile health rule types, create new mobile health rules, and how to view the health results of your mobile applications.

Mobile Health Rule Types

When you select the health rule scope for mobile applications, you are given a set of health rules based on the two health rule types: **Mobile Apps** and **Network Requests**. You can create new health rules using the mobile health rule types or use a custom health rule type.

Mobile Apps Health Rule Type

The **Mobile Apps** health rule type can be used to add conditions based on a wide range of mobile application metrics such as application starts, crashes, or network requests for the application. The network request metrics for the **Mobile Apps** health rule type refer to all the network requests of the application. For example, when you configure a threshold for the metric **Slow Network Requests**, the **Mobile App** type health rule will evaluate all the network requests of the mobile application in determining which network requests are slow.

Network Requests Health Rule Type

The **Network Requests** health rule can only be used to add conditions based on network-related metrics such as HTTP errors, network errors, network request time, and network requests per minute. The network request metrics for the **Network Requests** health rule type refers to specific network requests. For example, when you configure a threshold for the metric **Slow Network Requests**, the **Network Requests** type health rule will evaluate one network requests, such as calls to http://google.com in determining which network requests are slow.

Custom Health Rule Type

The **Custom** health rule type allows you to use any metric. This enables you to set health rules that have metrics from different health rule types. For example, you could have a custom health rule that is based on configured thresholds for mobile metrics and business transaction metrics.

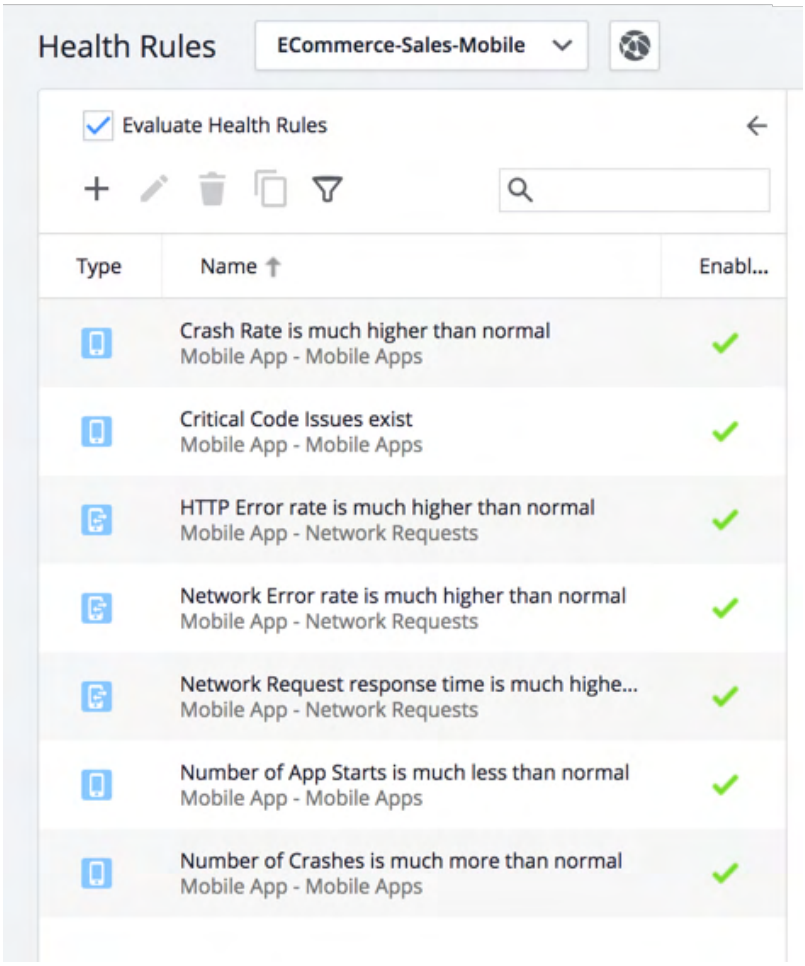
Create Health Rules

The process and user flow for creating health rules for mobile applications are similar to creating health rules for other applications. Thus, if you haven't created policies or health rules before, see Configure Policies and Configure Health Rules for general instructions. The sections below are not going to cover creating policies, but instead focus strictly on the important differences when creating health rules for mobile applications.

The first example will show you how to use the default set of mobile health rules, modify one of the default mobile health rules, and finally, how to create a new health rule using one of the mobile health rule types.

Using Default Health Rules

When you use the health rule scope for mobile applications, you are given a set of default health rules based on mobile health rule types. To enable these default health rules, click **Alert & Respond > Health Rules** to see the list of default health rules for mobile applications. You can use these default health rule types for your mobile applications by checking the **Evaluate Health Rules** checkbox:



Modify Default Health Rules

The default health rules for mobile applications are based on mobile health rule types and have default conditions based on preset values for mobile metrics. You may want to retain the default health rules, but want to customize the conditions for your mobile application. To do this, click the default health rule you want to modify and navigate to either the **Critical Warning** or **Warning Condition** panel. You can then change the parameters of the existing conditions or add new conditions.

Select Mobile Applications for the Health Rules

You can choose whether to apply the health rules for mobile applications. Depending on the mobile health rule type, you have different criteria to choose when to apply the health rules.

For health rules based on the health rule type **Network Requests**, you can apply the health rules for:

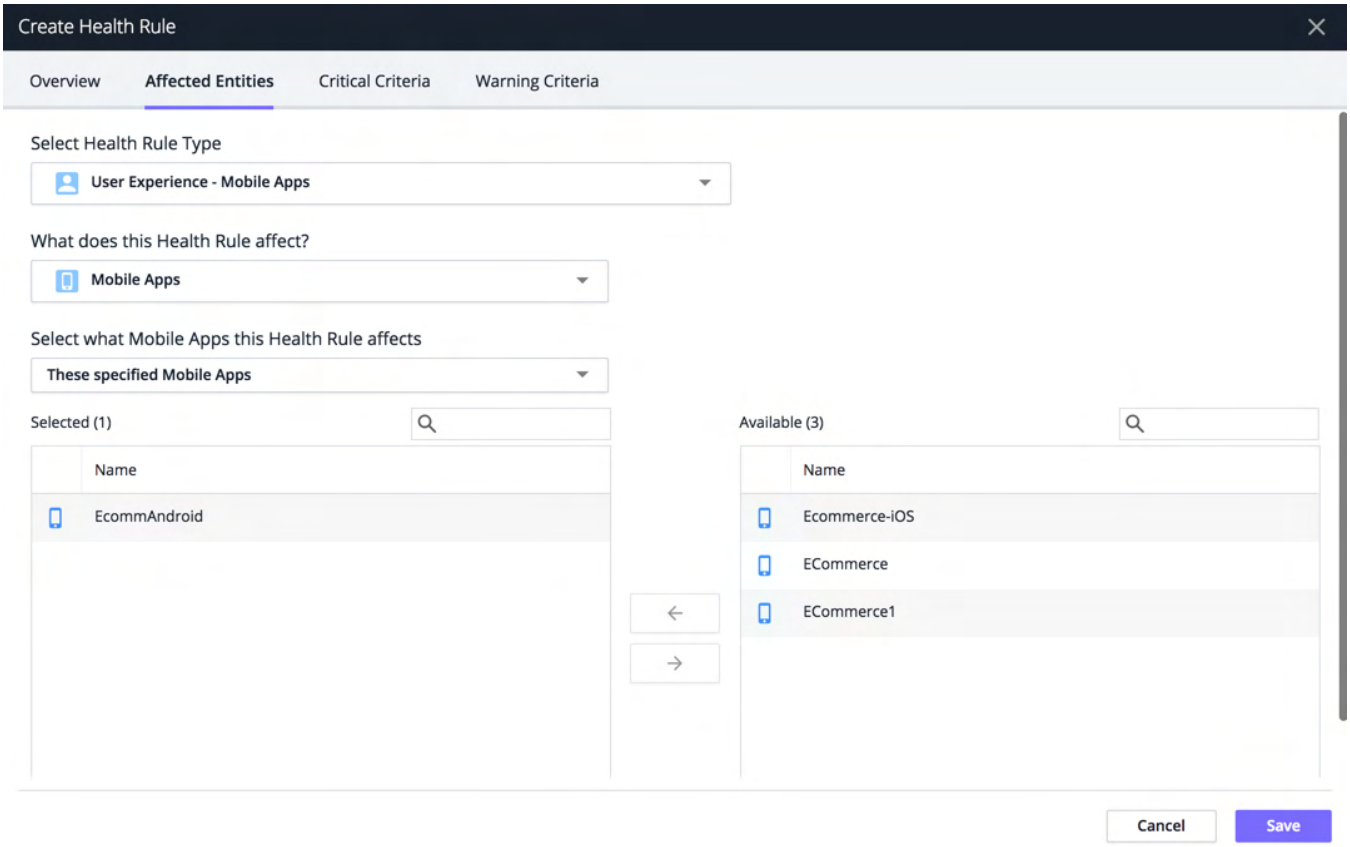
- All network requests in the mobile app group
- Specified network requests
- Network requests matching criteria
- Network requests for specific mobile applications

For health rules based on the health rule type **Mobile Apps**, you can apply the health rules for:

- All mobile applications in a mobile app group
- Specified mobile applications
- Mobile applications matching criteria

Create New Health Rules

You may want to create additional health rules based on one of the mobile health rules or a custom health rule. You do this in the same way you would create any health rule, except from the **Create Health Rule** dialog, you would select either the mobile health rule type **Mobile Apps** or **Network Requests**:



You can choose to apply the new health rule to mobile applications that you select, or those that match given criteria.

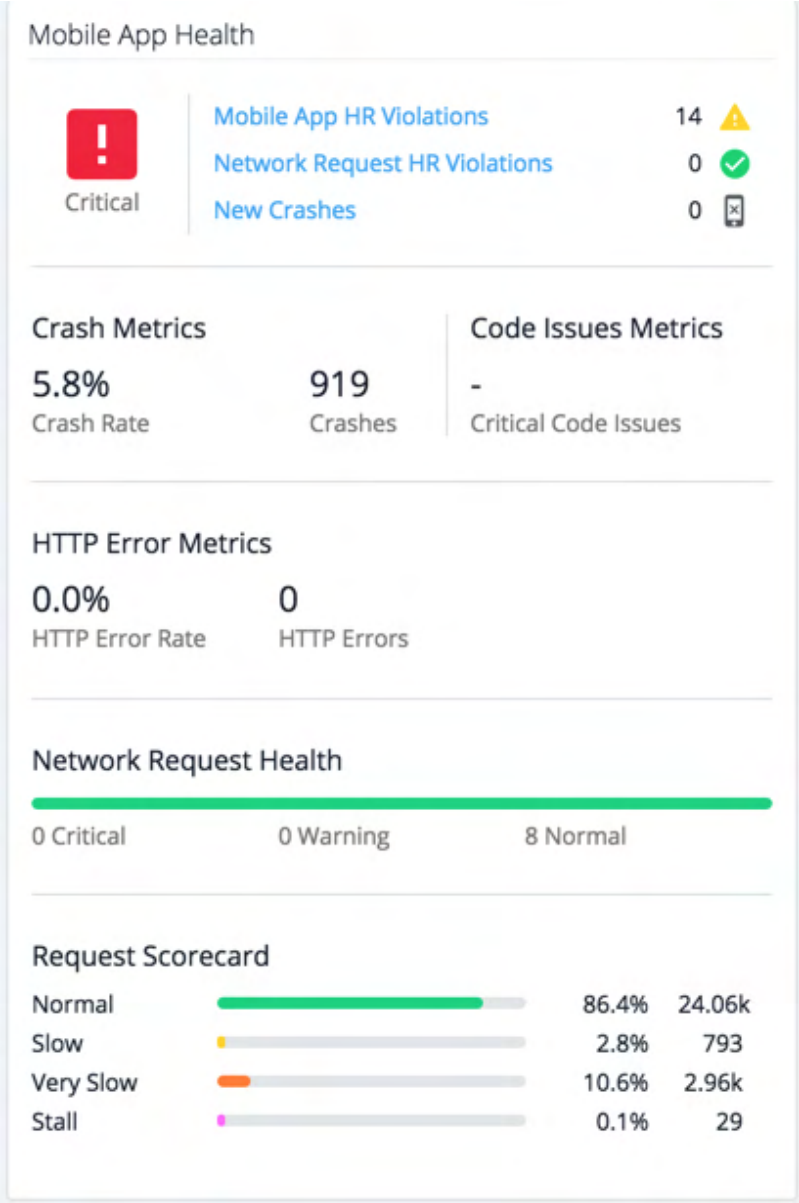
From the **Critical Criteria** and/or **Warning Criteria** panels, you can add conditions and define parameters with mobile metrics.

View Health Results

In addition to getting alerts when mobile applications are unhealthy, you can also view the health of your mobile application from certain widgets the **Mobile App Dashboard** and in the Events and Health Rule Violations tabs.

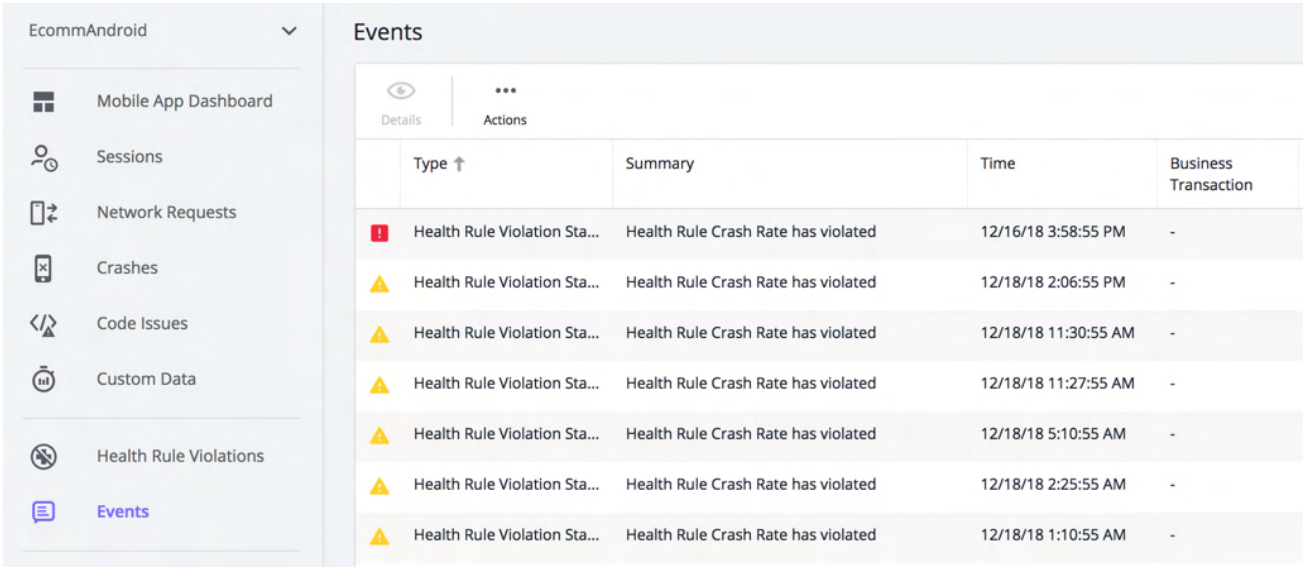
Overview Widgets

From the **Mobile Dashboard**, you should see the **Mobile App Health** widget. The widget displays health rule violations, crash metrics, and the network request health.



Events

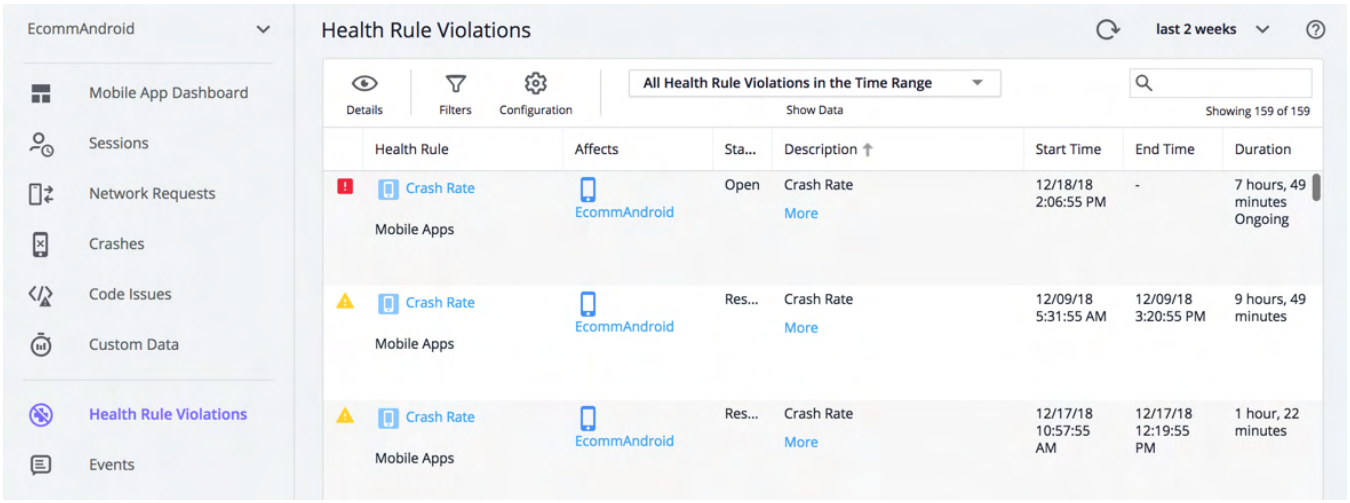
The Events tab displays the health rule violations for your mobile application. See Monitor Events.



You can sort the health rule violations for your mobile application by event types, health rules, or custom events you created.

Health Rule Violations

The Health Rule Violations tab lists the health rule violations that have happened over the specified time or that are still open.



Double-clicking one of the health rule violations in the list opens the **Health Rule Violation Details** dialog showing the type of health rule violation, the timeline, the events, and a description of the violation.

Network Requests

A network request is an HTTP request from your mobile app to a server-side application.

The iOS Agent detects network requests when the underlying implementation is handled by the `NSURLConnection` or `NSURLSession` classes.
The Android Agent detects network requests when the underlying implementation is handled by the `HttpURLConnection`, `HttpsURLConnection`, `HttpClient`, `OkHttp`, or `ch.boye.httpclientandroidlib` classes.

You can use the agent SDK to set up other HTTP classes.

View Network Requests

There are different ways of viewing network request data in the **Network Requests View**:

- The Network Requests list displays current network request types to your applications. You can sort the list according to key metrics such as the slowest response time, highest error rate, highest load, etc. You can view a network request dashboard that summarizes aggregate performance for a specific network request type.
- Network Request Analyze allows you to sort and filter a store of *all* the network request data your agents have collected and to see visualizations based on that data.
- A Network Request Snapshot reports information on an individual instance of a network request. Snapshots are useful for examining the details of the worst-performing requests. Access these snapshots from the network request snapshots list.

Network request data is also displayed in the Mobile App Dashboard.

- The Overview tab has the following widgets showing information about network requests:
  - **Network Request Scorecard:** Displays the number and percentage of normal, slow, very slow, and stalled network requests. Clicking on the widget opens the Network Requests list.
  - **Network Request Health:** Displays the number of network requests evaluated as normal, critical, or warning based on default or configured health rules. Clicking on the widget opens the Network Requests list.
  - **Network Request Time By Country:** Displays a heat map of the world based on the number of network requests. Clicking on the widget opens the geographic view.
  - **Requests Per Minute:** Displays a bar graph showing the number of requests per minute over the specified time period. Clicking on the widget opens the geographic view.
  - **Network Requests Time Distribution:** Displays a bar graph showing the number of requests made at different network request times. In addition, the graph gives the percentage rank of the number of network requests in its frequency distribution of network request time. For example, the **95th percentile (5,419 ms)** indicates that 95% of the network requests had a network request time of 5,419 ms or less. Clicking on the widget opens the Charts tab of Network Request Analyze.
  - **Network Request Time Trend:** Displays a line graph showing the average network response time over a specified time period. Clicking on the widget opens the geographic view.
  - **HTTP Errors:** Displays the number and rate of HTTP errors. Clicking on the widget opens the Network Requests list.
  - **HTTP and Network Errors Trend:** Displays a line graph comparing the number of HTTP and network errors over a specified time period. Clicking on the widget opens the geographic view.
- The geographic view reports aggregated mobile data by geographic location. Monitor the geographic view to learn which countries have the highest number of requests, the longest request times, and the most errors.
- Usage stats display key network request metrics by various criteria: device, carrier, operating system version, connection type, and application. For example, you can see which carriers are the slowest or which devices are producing the most errors.

The Controller processes a maximum of 2000 network requests per mobile app group and 500 network requests per mobile application. See Network Request Limits for suggestions on how to configure network request detection to stay under this limit.

Access the Network Requests View

1. Open the application you want.
2. In the left navigation bar, select **Network Requests**.
3. Click the tab for the view you want to access.

Network Requests List

Related pages:

- Network Request Dashboard
- Network Request Limits

The **Network Requests** list shows all the network requests types from your instrumented mobile application, along with their key performance indicators.

Network Requests List Organization

The **Network Requests** list is a table that displays aggregated metrics for current requests, with one row for each request type. The columns display the name of the network request and the aggregated metrics associated with it. See Mobile RUM Metrics for descriptions of these metrics.

Click a column header to sort the list based on the column's metric. For example, if you want to sort by the slowest requests, click the **Network Request Time (ms)** column header. You can toggle the column to switch between ascending and descending order. Use **View Options** to configure the table. Check **With Load** if you want to show only network request types that have experienced active load, that is, those for which there were one or more measured network requests over the selected time period.

To view a network request dashboard:

1. Select the network request type in the list.
2. Click **Details** or double-click the network request.

More Actions Menu

Use the **More Actions** menu to select one or more requests in the list and perform these actions:

- **Exclude Request(s)**: Use this option to direct AppDynamics to ignore the selected request(s) and stop reporting metrics for them. You can use the **View Excluded Requests** option to see requests that have been excluded and then you can "un-exclude" them.
- **Rename Request**: Use this option to rename one selected request in the AppDynamics console.
- **Delete Request(s)**: Use this option to remove the request(s) from the list. If AppDynamics discovers a deleted request again it will reappear in the list. To prevent it from re-appearing, use **Exclude Request(s)**.

Network Request Dashboard

Related pages:

- Network Requests
- Network Requests List
- Network Request Analyze
- Mobile RUM Metrics

Each network request type has its own dashboard that graphically displays key performance indicators for that type over the selected time range. To select the time range, use the general time range dropdown at the top right of the UI.

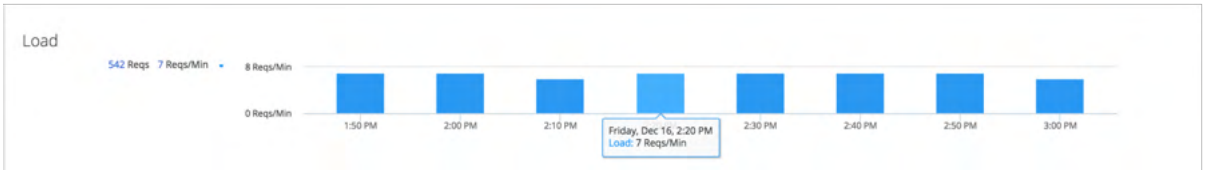
Network Request Dashboard Organization

The **Network Request Dashboard** displays summary key network request metrics for the time selected. To see any particular metric in the metric browser, click the metric value (in link blue).

The trend graphs for the key performance indicators are:

- **Network Request Time**: Average times in milliseconds.
- **Total Server Time:** Displayed only If the mobile request is correlated with a server-side application.
  The total server time is the interval between the time that the server-side application receives the network request to the time that it finishes processing the request. Use this graph to determine, on average, how much time is spent on the network versus how much time is spent on the server to process the user's request.
- **Load**: Total Requests and Requests per Minute.
- **Errors**: Network Errors and HTTP Errors in total and per minute.
- **Related Business Transactions**: If the request is correlated with a server-side application, the dashboard lists business transactions associated with the request below the performance metrics.
  You can click the link to a related business transaction to see its business transaction dashboard.
  If transaction snapshots were taken at the same time as the network request, the dashboard lists the transaction snapshots below the business transactions. See Transaction Snapshots.

You can hover over any data point on any of the trend graphs to see the metric for a precise point:

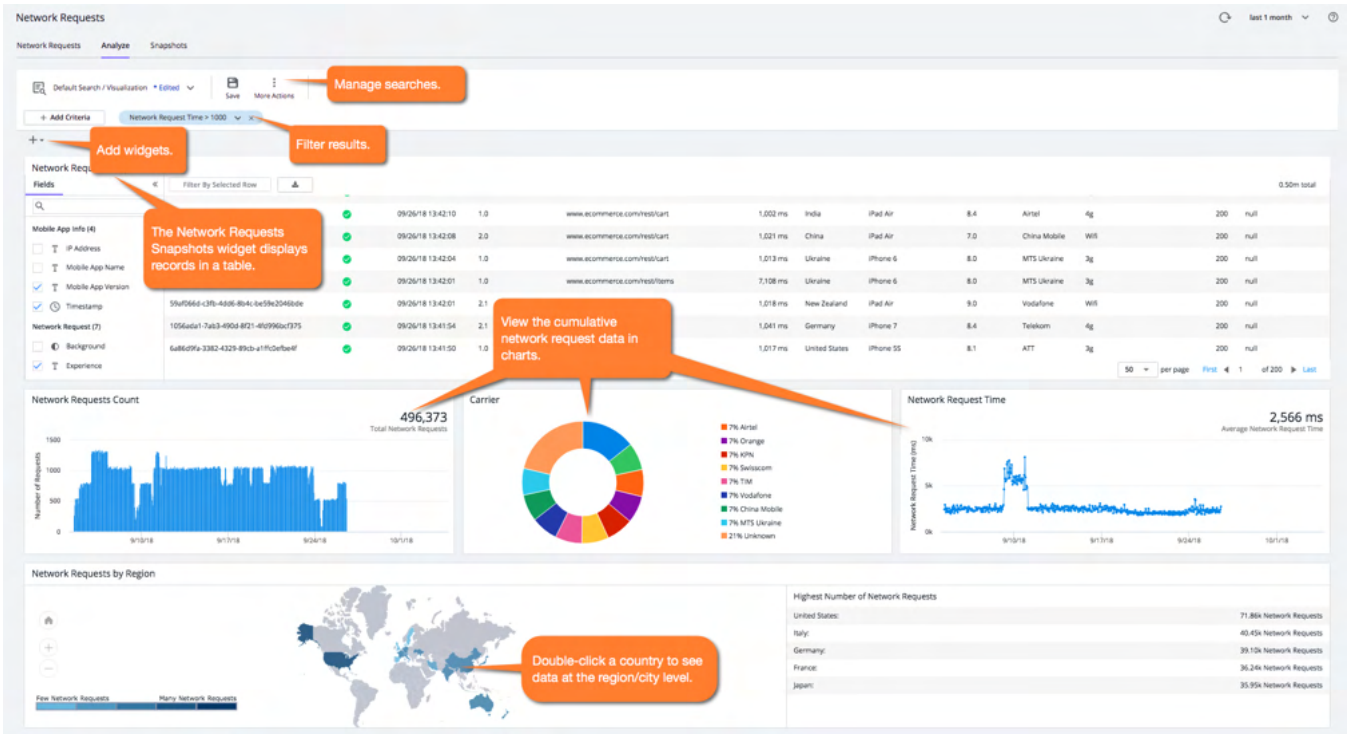


Network Request Analyze

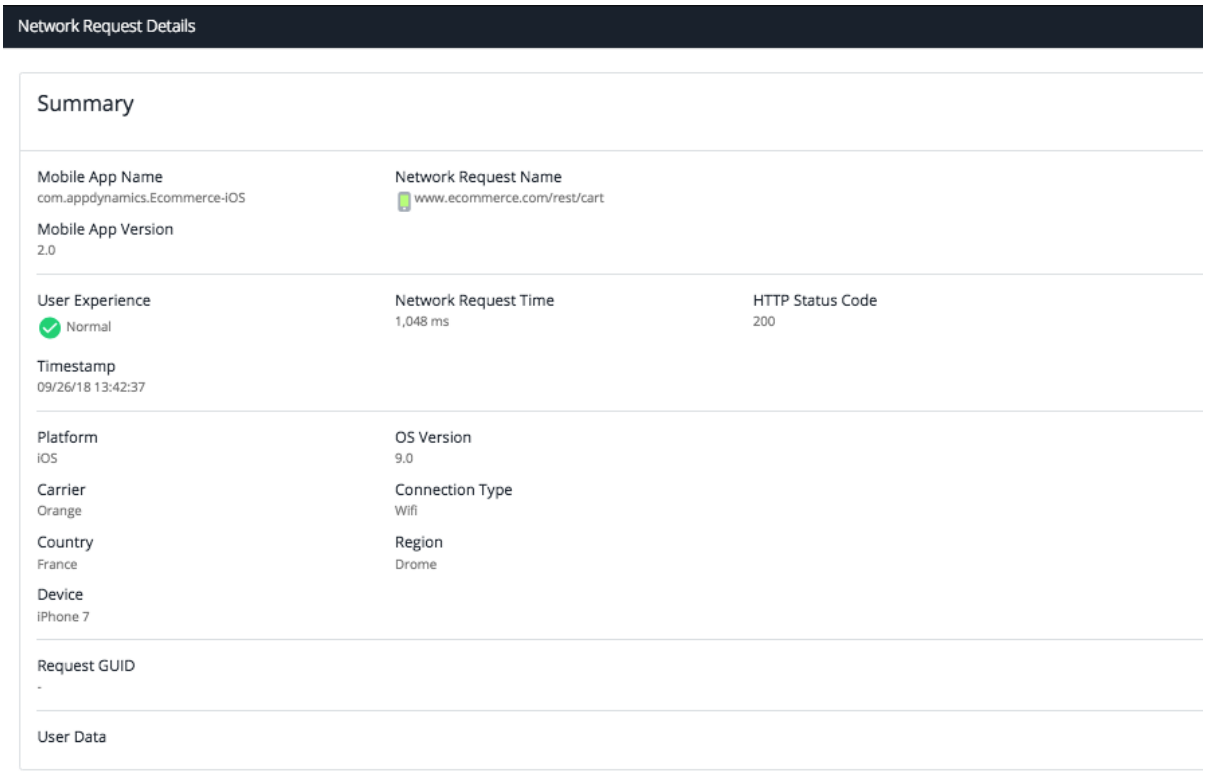
Mobile RUM collects data on every network request that you have instrumented, and it also takes detailed snapshots periodically in case performance issues have been detected.

Based on network request performance data, Mobile RUM calculates metrics which are associated with each network request and with predefined aggregations across network requests. You can focus on certain aspects of the performance data using more specific or flexible criteria, such as all network requests from a specific country *and* originating from a specific carrier.

With **Analyze**, every single network request event is collected and stored by the AppDynamics Events Service. Using **Analyze**, you can see results based on this cumulative data in tabular form or as charts.



Double-click a network request snapshot in the **Network Request Snapshots** widget to see the information for a specific request.



Mobile RUM Network Request Analyze Versus Mobile Requests Analytics

The data shown in the **Analyze** page is processed and stored by the AppDynamics Platform Events Service. **Network Request Analyze** provides a number of widely used visualization types to let you explore your application's performance. A separate product, AppDynamics Mobile Analytics, has components called Mobile Requests and Mobile Crash Reports. These components are based on the same Events Service and use the same data, but offer additional capabilities, including:

- Additional predefined widgets, such as the funnel widget
- ADQL for searching the data
- Creating custom widgets
- Manipulating multiple dashboard types
- Longer retention time for storing data

AppDynamics Mobile Requests Analytics requires a license separate from the Mobile RUM license. See Analytics Mobile Requests Data.

Network Request Snapshots List

A network request snapshot captures the details of one instance of a network request. Examine these details to troubleshoot the causes of slow network request performance. See Network Request Snapshots.

You access the list using the Network Request Snapshots tab. The network request snapshots list is a table that displays a row for each network request snapshot. The columns describe the properties of each snapshot.

To view a network request snapshot:

1. Select the network request snapshot in the list.
2. Click **Details** or double-click the request.

Network Request Snapshots

Related pages:

- Network Request Snapshots List
- Transaction Snapshots

Network request snapshots capture detailed information about individual network request instances by your application. They can help you troubleshoot the causes of poorly performing mobile applications.

When Network Request Snapshots are Captured

Mobile RUM starts capturing snapshots when user experience becomes slow, based on how you have configured the thresholds for slow, very slow, and stalled. See Configure Mobile Network Request Thresholds.

Periodic snapshots of normal user experience are also captured at least once per minute.

Network Request Snapshot Content

A network request snapshot contains summary data about the individual request as well as any business transactions associated with the request if correlation with an instrumented server-side application is available.

The snapshot contains extensive metrics for the request in the Summary data, including:

- **User Experience**: Normal, Slow, Very Slow, or Stalled.
- **Time**: When the request was received by the EUM Cloud or EUM Server collector, in UNIX epoch time.
- **Mobile Network Request**: Link to the network request dashboard for the network request of which this snapshot is an instance. See Network Request Dashboard.

Business Transactions in Network Request Snapshots

When a network request snapshot is associated with one or more business transactions on an instrumented server-side application, the business transactions are listed in the **Business Transactions** panel in the network request snapshot. You can click the link to see the business transaction dashboard for the associated business transaction.

If transaction snapshots for an associated business transaction were captured at the same time as the network request snapshot, they are linked in the **Transaction Snapshots** panel of the network request snapshot. If a call graph icon is displayed for a snapshot in the transaction snapshot list, a full or partial call graph is available for that transaction snapshot. This allows you to examine the cause of performance problems on the server side. Click the link to see the associated transaction snapshot.

Transaction snapshots are triggered on the server when slow or stalled business transactions are identified when a diagnostic session is started or periodically based on a configured interval. In general, slow, very slow, and stalled transactions are more likely to trigger a transaction snapshot on the server than transactions operating within the normal range. For more information about when server-side transaction snapshots are captured, see Transaction Snapshots.

Access Network Request Snapshots from Transaction Snapshots

If a transaction snapshot seen from the server side also has generated a correlated network request snapshot, a mobile snapshot link appears in the top right of the transaction snapshot flow map. Click the link to open the network request snapshot.

Archive Network Request Snapshots

Normally network request snapshots are purged after two weeks. You can archive a snapshot beyond the normal snapshot lifespan to retain it for future analysis.

To archive a snapshot, click the **Archive** button in the upper right corner of the snapshot panel.

You can view archived snapshots by checking **Archived** as a view option in the network request snapshots list.

Customers with on-premises Controllers can modify the default two-week period by configuring the `event.retention.period` property in the **Controller Settings** section of the Administration console.

Crashes

Related pages:

- Crash Analyze
- Crash Snapshots
- Crash Snapshot Properties

Use crashes to get trend and detailed information on mobile application crashes.

A crash snapshot is a detailed report on a particular crash including the code that was executing when the application crashed. Crash snapshots help you understand the causes of crashes.

View Crash Information in Widgets

The **Crash Metrics** and **Events** widgets are on the **Mobile App Dashboard**. The **Crash Metrics** widget displays the crash rate and the number of crashes for your mobile application. The **Events** widget displays new crash events.

1. Open the application you want.
2. From the **Mobile App Dashboard**, both **Events** and **Crash Metrics Scorecard** widgets display.
3. Click **Crash Metrics** or click one of the new crash events in the **Events** widget to access the **Crash Dashboard**.

Access the Crash Dashboard

The Crash Dashboard is where you can view both summary and detailed information about crashes.

1. Open the application you want.
2. On the left navigation bar of your application, select **Crashes**.

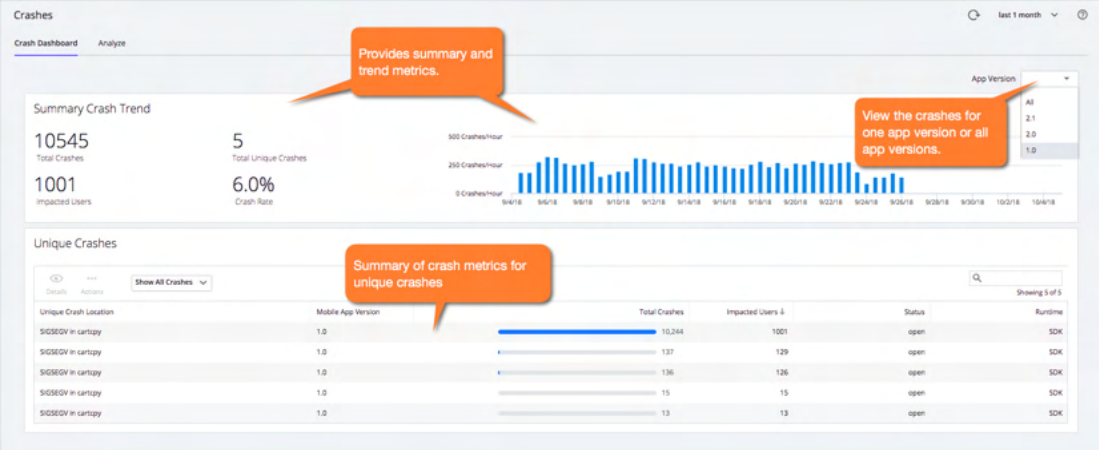
Crash Dashboard

Related pages:

- Crash Snapshots
- Crash Snapshot Properties

The **Crash Dashboard** aggregates mobile application crash data over time, using the Events Service. This service collects and stores *all* the data collected by the mobile agent.

The **Crash Dashboard** is divided into two panels and has the **App Version** dropdown that enables you to view crash data for different versions of your application.

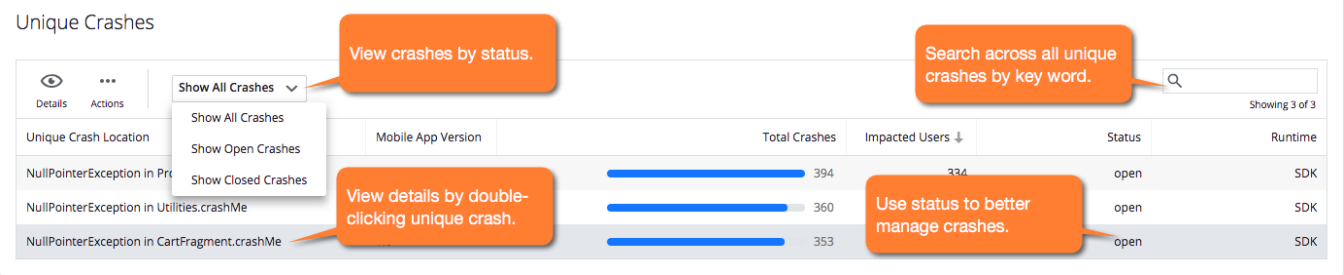


Summary Crash Trend

This panel displays a running total of crashes, unique crashes, impacted users, crash rate, and crash trends. The crash trends is a timeline of crash rates. It also reports any iOS crash reports that were uploaded without the accompanying dSYM file. For more information on how to use dSYM files with crashing reports, see Get Human-Readable Crash Snapshots.

Unique Crashes

Multiple crashes can be caused by the same underlying code issue. The **Unique Crashes** panel displays a list of crashes grouped by common characteristics and displays basic information about the crash. You can view open, closed, or all crashes.

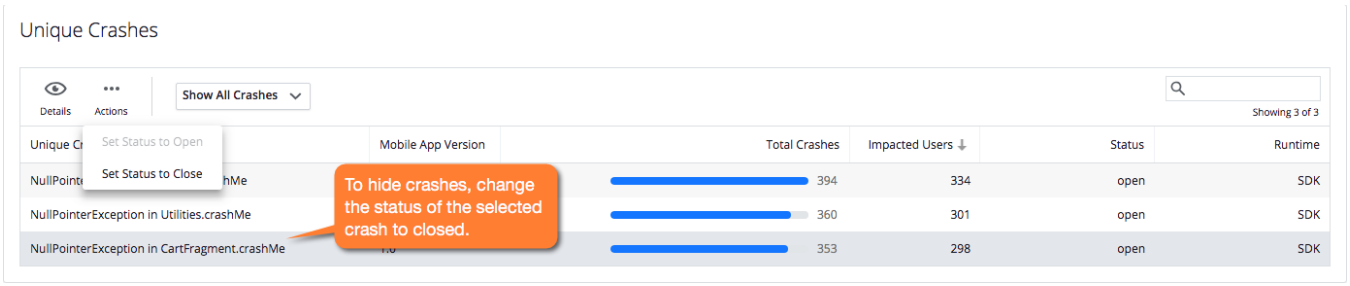


Unique Crash Details

To see more detail per crash, click the crash that interests you, in blue. The dashboard for that crash appears, with a header, trend bar graph, crash distribution charts, and a snapshot of the crucial details common to all the crash snapshots. See the Crash Summary section on the Crash Analyze page.

Crash Status

In addition to viewing crash details, from the **Unique Crashes** panel, you can select a unique crash, click **Actions**, and set the status to either open or closed. You can set the status to mark those unique crashes that you want to ignore or have fixed the root cause of.



When a crash is marked as closed, the crash will no longer trigger a new crash event, so you will not see the crash in the **Events** widget of the **Mobile App Dashboard,** the **Events** page, and it will not be included in alerts for new crashes.

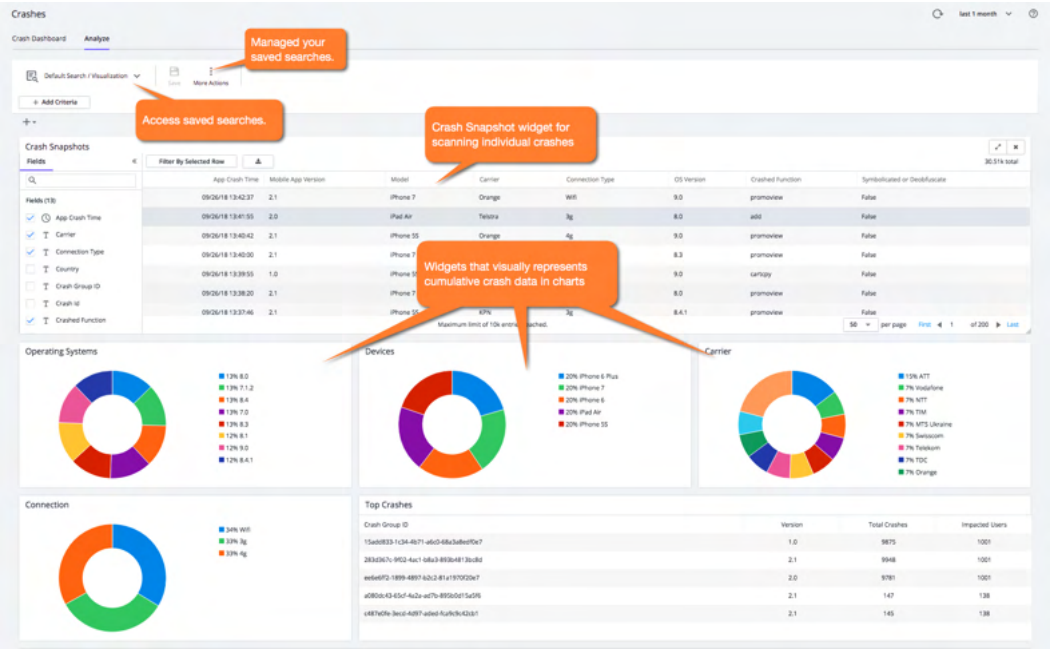
Crash Analyze

Related pages:

- Instrument an iOS Application
- Instrument an Android Application Manually

Crash Analyze lets you view and analyze the results based on the cumulative data. With **Crash Analyze**, every crash event is collected and stored by the AppDynamics Events Service.

The **Crash Analyze** page has the **Crash Snapshots** widget for viewing a list of current crash snapshots, with one row for each snapshot, and a selection of different widgets for visualizing the cumulative data in charts.



Crash Snapshots Widget

The Crash Snapshots tab lets you scan individual crashes and allows you to filter and sort to get exactly the crash set in which you are interested. The columns of the table are fields representing the crash snapshot properties. See Crash Snapshot Properties. Click **View Details** to see information for a specific crash.

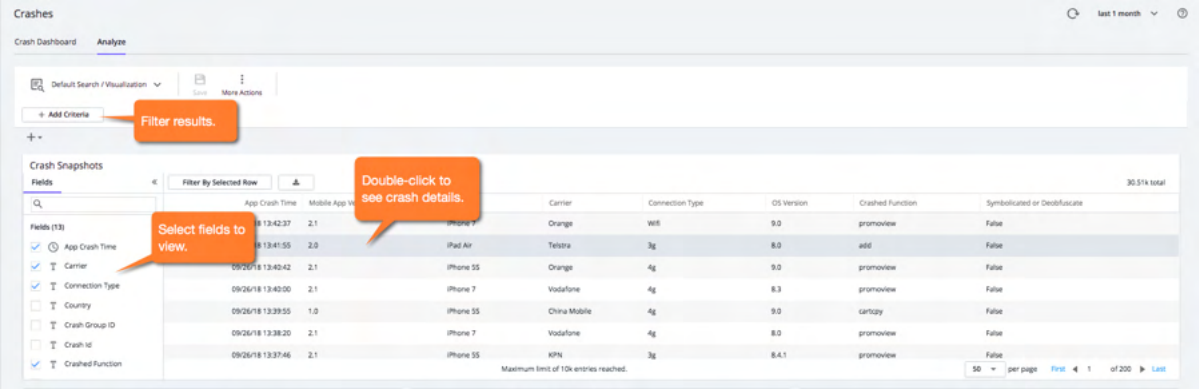
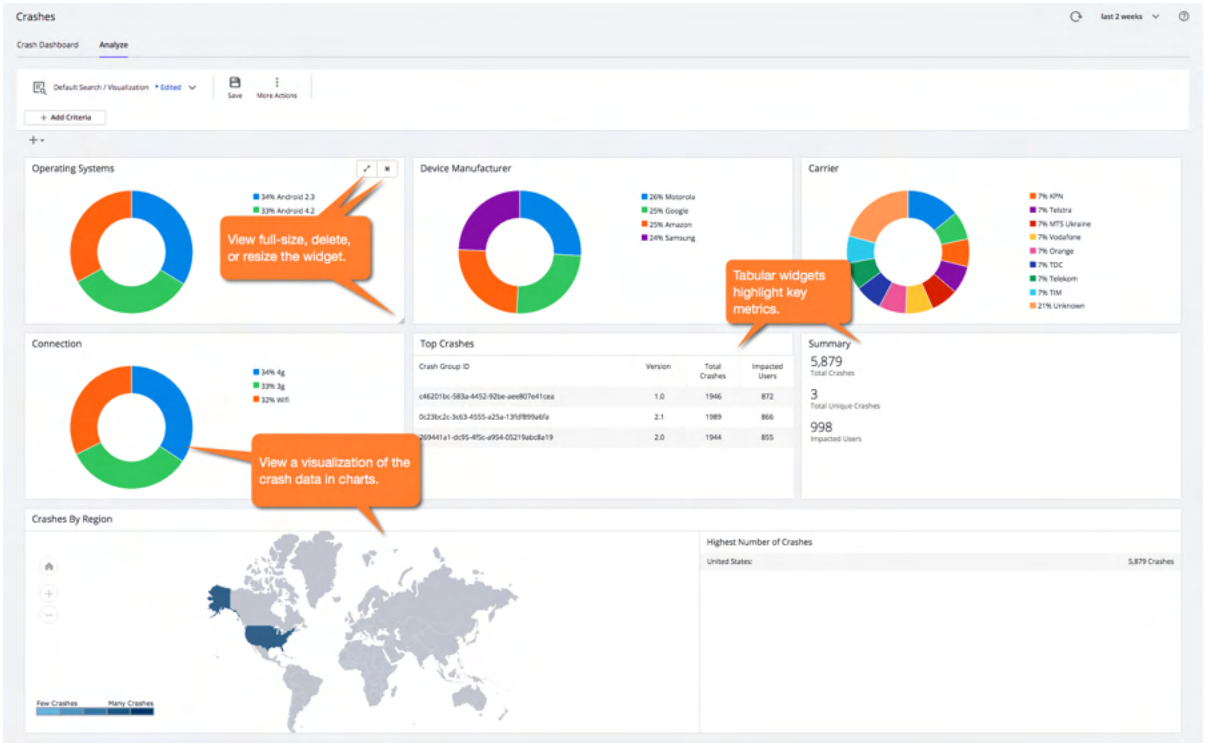


Chart and Tabular Widgets

The chart widgets provide you with a set of predefined widgets that offer visualizations of the data set you have created. The tabular widgets highlight key metric points such as top crashes. You can delete, re-add, re-size, and drag-and-drop to move all of the widgets. You can also set criteria to visualize a subset of the data.



You can also view crash snapshots in the context of a separate product, AppDynamics Mobile Analytics. It offers additional mechanisms to analyze and visualize crash patterns, including:

- Additional predefined widgets, such as the funnel widget
- ADQL for searching the data
- Custom widgets
- Manipulating multiple dashboard types
- Longer retention time for storing data

AppDynamics Mobile Crash Report Analytics requires a license separate from the Mobile RUM license.

Crash Snapshots

Related pages:

- Crash Analyze
- Crash Snapshot Properties

Crash snapshots provide detailed information for one application crash. When an instance of an instrumented application crashes, a crash snapshot is created.

The snapshot provides information to help you analyze the cause of the crash, including:

- Crashed function
- Source file containing the crashed function
- Line number in the source file, if available
- Stack trace of the application at the time of the crash

For iOS applications, crash snapshots are based on:

- Fatal signals (SIGSEGV, and so on)
- Unhandled Objective-C exceptions

For Android applications, crash snapshots are based on unhandled Java exceptions.

Crash Snapshot Details

The **Crash Snapshot Details** dialog has the two panels **Crash Summary** and **Crash Distribution**. The **Crash Summary** panel displays basic crash information and the stack trace. The **Crash Distribution** panel displays widgets with distributed crash metrics by the crash group and the stack trace.

To view crash snapshot details:

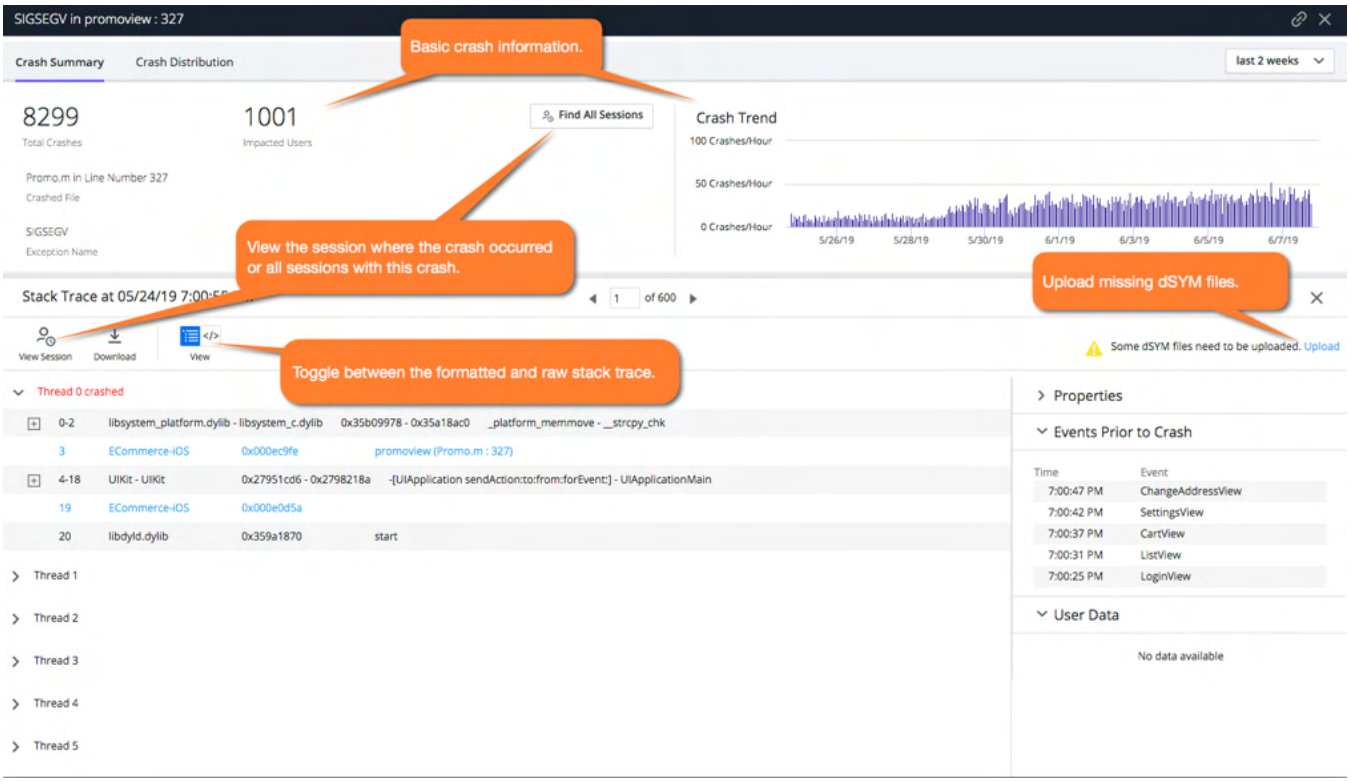
1. From the Records tab, select a crash snapshot in the list.
2. Click **View Details**.

Crash Summary

The **Crash Summary** panel of the snapshot displays the key properties of the snapshot. If you have customized your instrumentation (Android/iOS) to include breadcrumbs, they are displayed in the **Events Prior to Crash** panel. Up to 99 breadcrumbs display, beginning with the most recent. Not all of this information may be available. If you added user data (Android/iOS), it also displays here.

The **Stack Trace** panel displays the call stack of the crashed application, showing the thread in which the crash occurred. This is the full stack trace for this specific crash, not the aggregated stack trace shown in the **Unique Crash Dashboard**. You can view the formatted stack trace that groups threads and parses the output or the raw log output for the stack trace.

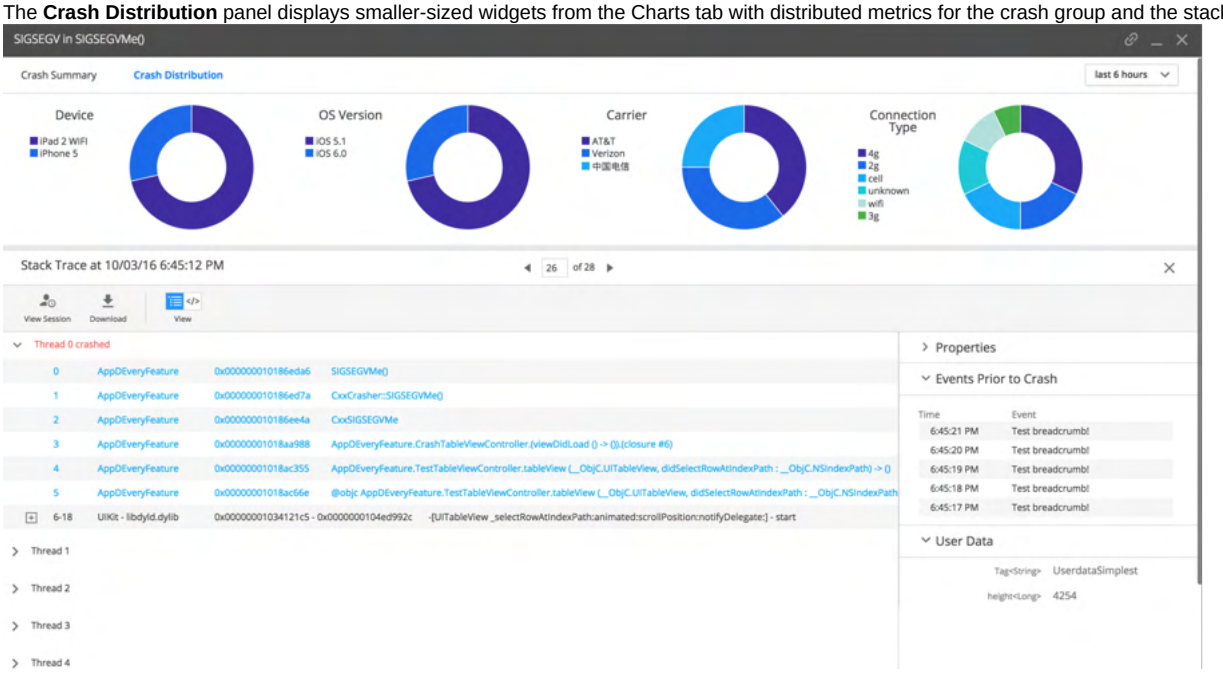
Click **Download** to access the stack trace in a file that you can forward to developers or port to logging platforms. To see the crash in the context of the mobile session, click **View Session**.



If the information in the stack trace is cryptic, it is possible that source code for your iOS app was not symbolicated or the source code for your Android app was obfuscated. See Get Human-Readable Crash Snapshots for information about why this happens and what you can do about it.

Crash Distribution

The **Crash Distribution** panel displays smaller-sized widgets from the Charts tab with distributed metrics for the crash group and the stack trace.



Crash Snapshot Properties

Related pages:

- Network Requests List
- Network Request Analyze
- Network Request Snapshots List
- Network Request Snapshots
- Analytics Mobile Crash Reports Data

This page describes the list of crash snapshot properties. They display in the **Crash Snapshot Details** panel when you open a crash snapshot from **Crash Analyze**.

- **Carrier**: Name of the mobile carrier.
- **Connection Type**: Active connection type at the time of the crash, if known.
- **Crashed File/Line/Function**: Name of the source file containing the crashed function, with the line number, if available. The name of the topmost function on the crashed thread's call stack. If this function is an Objective-C method, this name includes the class name. For Android, this name is the fully qualified name of the topmost method on the uncaught exception's stack trace.
- **Crash Time**: Timestamp when the crash occurred, based on the mobile device's clock.
- **Device/Manufacturer**: Model or manufacturer name of the mobile device on which the crash occurred.
- **Events Prior to Crash**: Any breadcrumbs you might have set up.
- **Exception Name**: Name of the fatal signal (iOS) or uncaught exception (Android) associated with the crash.
- **Memory Usage:** Memory used by the app when the crash occurred.
- **Mobile App**: Name of the app that crashed.
- **OS Version**: Operating system version of the mobile device on which the crash occurred.
- **Platform**: iOS or Android.
- **Request Timestamp**: Time when the beacon arrived at the Collector. Snapshot only.
- **User Data**: Any user data you might have added.
- **Version**: Version string of the crashed application.

Crash Snapshot list only

- **Country: O**riginating country.
- **Crash id: U**nique identifier for the crash.
- **Deobfuscated:** For Android: **true**, if this crash report has been matched with a ProGuard mapping file and deobfuscated; otherwise, **false**.
  See Upload the Proguard Mapping File in Instrument an Android Application Manually and Get Human-Readable Crash Snapshots. A **false** value for this property does necessarily indicate that the crash report will not be human-readable since it is possible that the application in question was not obfuscated.
- **Symbolicated:** For iOS: **true,** if this crash report has been matched with a dSYM file and symbolicated; otherwise, **false**.
  The application must have been compiled with the **Debugging Information Format** set to `DWARF with dSYM File` for a crash report to exist.
  See Upload the dSYM File and Get Human-Readable Crash Snapshots.

Get Human-Readable Crash Snapshots

Related pages:

- Crash Snapshots

The information in most raw crash stack traces is not fully human-readable. To make your crash stack traces more easily understood, you need to provide a platform-specific mapping file that can translate the raw data into human-readable output. Normally, you upload the file at the time that you instrument your mobile application.

- For iOS, see Upload the dSYM File.
- For Android, see Manually Upload Mapping Files.

This page explains the advantages of providing these files.

iOS dSYM File

For iOS applications, the raw data in the stack traces in crash snapshots consists of memory addresses of stack frames that point to executable application code. It also includes symbols and memory offsets for the system library code used by the application. Such a partially symbolicated stack trace looks something like this:

```
Thread 11 Crashed:
0   libobjc.A.dylib                 0x38cb50fc objc_retain + 12
1   SomeApp                         0x003a2204 0xc6000 + 2998788
2   SomeApp                         0x003a0854 0xc6000 + 2992212
3   SomeApp                         0x003a09d4 0xc6000 + 2992596
4   SomeApp                         0x003948e4 0xc6000 + 2943204
5   Foundation                      0x2f3a2dc2 __NSThread__main__ +
1058
6   libsystem_pthread.dylib         0x392cec5a _pthread_body + 138
7   libsystem_pthread.dylib         0x392cebca _pthread_start + 98
8   libsystem_pthread.dylib         0x392ccccc thread_start + 4
```

AppDynamics attempts to display stack traces with the names of functions with offsets into those functions to help you identify the line of code that was executing when the application crashed. To get the symbols that map to the executable code, it needs the dSYM (desymbolication) file for the crashed application.

If the dSYM file for the crashed application has been uploaded, the symbolicated stack trace shows the function name and the offset into the function where the app crashed. It looks something like this:

```
Thread 0 Crashed:
0   libobjc.dylib               objc_release + 0x14
1   CoreFoundation              CFStringCreateWithFormat + 0x4
2   UIKit                       -[UIButton init] + 0x96
3   MyApp                       createUI (MyAppDelegate.m:42)
```

The dSYM file is created when the application source code is compiled with the **Debugging Information Format** set to `DWARF with dSYM file`. AppDynamics recommends that you build all the iOS apps that you want to monitor using this option and then upload the dSYM file to AppDynamics. The best time to do this is when you instrument the app.

If a dSYM has been uploaded for a crashed application, in the crash list the **Symbolicated** column for the associated crash snapshot is true.

If the `symbolicated` property is `false` and you want to see user-friendly stack traces in your crash snapshots for this application, you need to locate and upload the dSYM file for the crashed application.

ProGuard Mapping File for Android

If an Android app was not obfuscated to prevent reverse engineering, you should see human-readable stack traces in your crash snapshots by default.

If the code was obfuscated, however, AppDynamics needs the ProGuard mapping file to be able to deobfuscate the app. The best procedure is to upload this file at the time you build the app.

If a ProGuard mapping file has been uploaded for a crashed application, in the crash list the **Deobfuscated** column for the crash snapshot is `true`.

If the `deobfuscated` property is `false` and the stack traces you see in the crash snapshots are obfuscated, you need to locate and upload the mapping file for the application.

Crash Alerts

You can configure AppDynamics to send you alerts when your mobile application crashes. You can configure email alerts for health rule violates based on some default health rules or by manually creating a policy with a notification action or creating an email digest.

> Crashes that have the status "Closed" do not trigger alerts.

Set Up Alerts for Crashes

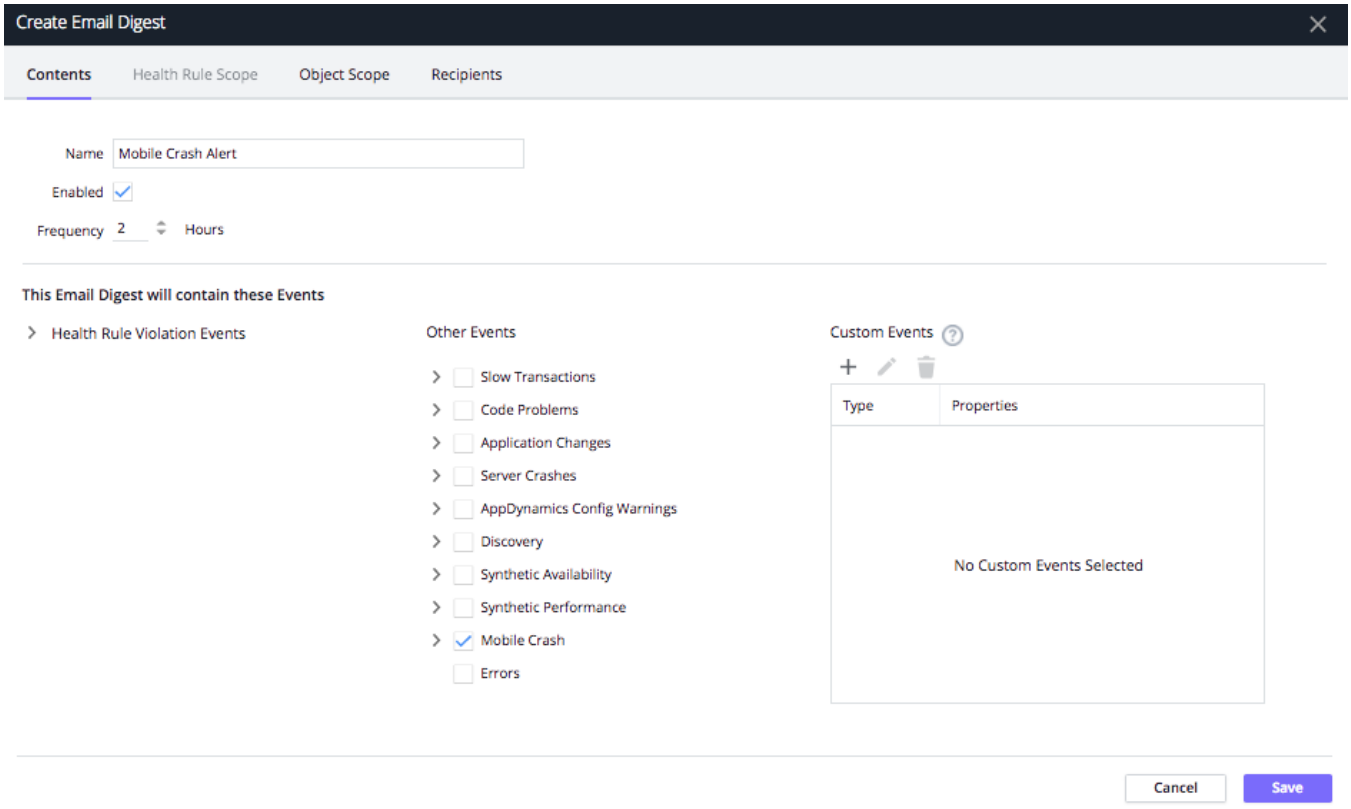
Setting up alerts for crashes is similar to setting up alerts for other health rule violations:

1. Configure Mobile Crash Alerts. This will configure the default number of crashes.
2. If desired, update the **Critical Criteria** and **Warning Criteria** for the health rule **Crash Rate is much higher than normal**. See Configure Health Rules.
3. Create a policy that will be triggered when the health rule for crashes is violated. You'll need to configure an action to be taken when the policy is triggered.
4. After you set up the email alerts, when a health rule is violated, an email will be sent indicating whether there was a warning or critical alert and providing links to view crash and session details.

Create Email Digest for Crashes

You can create the same type of notification for crashes by creating an email digest.

From the **Create Email Digest** dialog, check the **Mobile Crash** checkbox:



You can then use the **Email Digest** wizard to add recipients, a custom message, and schedule the frequency to send emails.

Code Issues

Code issues consist of caught non-fatal exceptions or application not responding (ANR) issues. A code issue snapshot is a detailed report on a particular event including the code that was executing when the exception or ANR occurred.

Code issue snapshots help you understand the causes of these issues. Use **Code Issues** to get trend and detailed information on application non-responsive issues (ANRs) and caught errors and exceptions.

Catch and Report Code Issues

All the Mobile Agents except the iOS Agent will automatically detect and report ANRs. For iOS, you have to enable ANR detection. To catch and report errors and exceptions, you instrument your mobile applications with iOS, Android, and Xamarin SDKs.

See the section "Report Errors and Exceptions" (iOS, Android, Xamarin) to learn how to use the SDKs to manually catch and report errors and exceptions.

Access the Code Issues Dashboard

The Code Issues Dashboard is where you can view both summary and detailed information about code issues.

1. Open the application you want.
2. On the left navigation bar of your application, select **Code Issues**.

View Code Issue Widgets

The **Summary Code Issue Trend** and **Unique Code Issues** widgets are on the **Code Issues Dashboard**. The **Summary Code Issue Trend** widget displ ays the number of code issues, impacted users, and unique code issues as well as graph the code issues on a timeline. The **Unique Code Issues** widget d isplays details about the unique code issues such as the severity, the issue type, runtime, impacted users, and mobile app version.

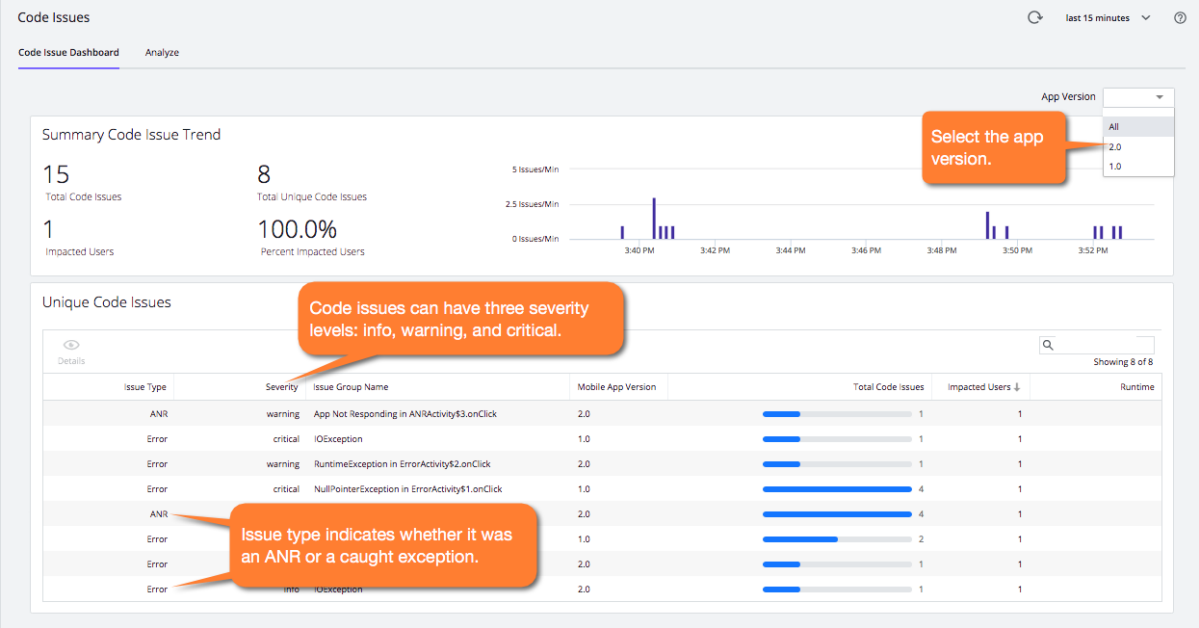
To view code issue information:

1. Open the application you want.
2. From the **Code Issues Dashboard**, both **Summary Code Issue Trend** and **Unique Code Issues** widgets display.
3. You can click one of the unique code issues in the **Unique Code Issues** widget to view the **Code Issues Details** dialog.

Code Issues Dashboard

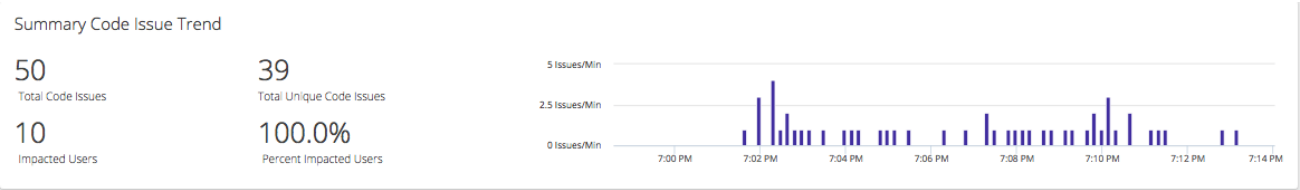
The **Code Issues Dashboard** aggregates mobile application caught exceptions and ANRs over time, using the Events Service. This service collects and stores *all* the data collected by the mobile agent.

The **Code Issues Dashboard** is divided into two panels and has the dropdown **App Version** to view code issues for different versions of your application.



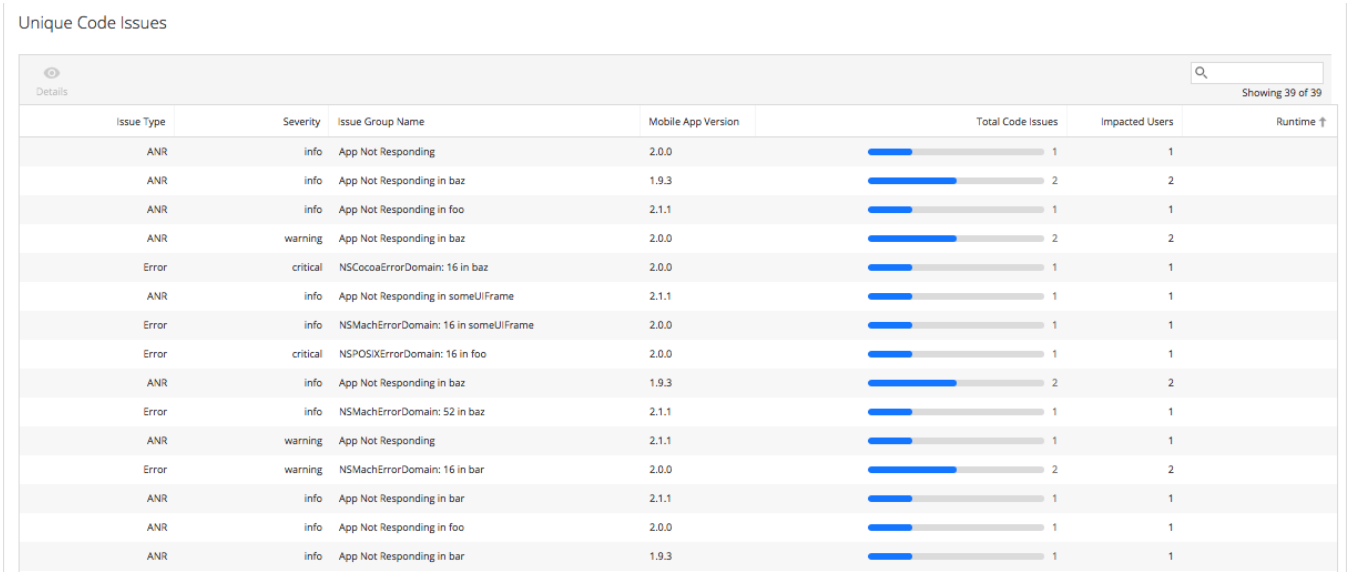
Summary Code Issue Trend

This panel displays a running total of code issues, unique code issues, impacted users, the percent of impacted users, and code issue trends. The code issues trends is a timeline of code issues.



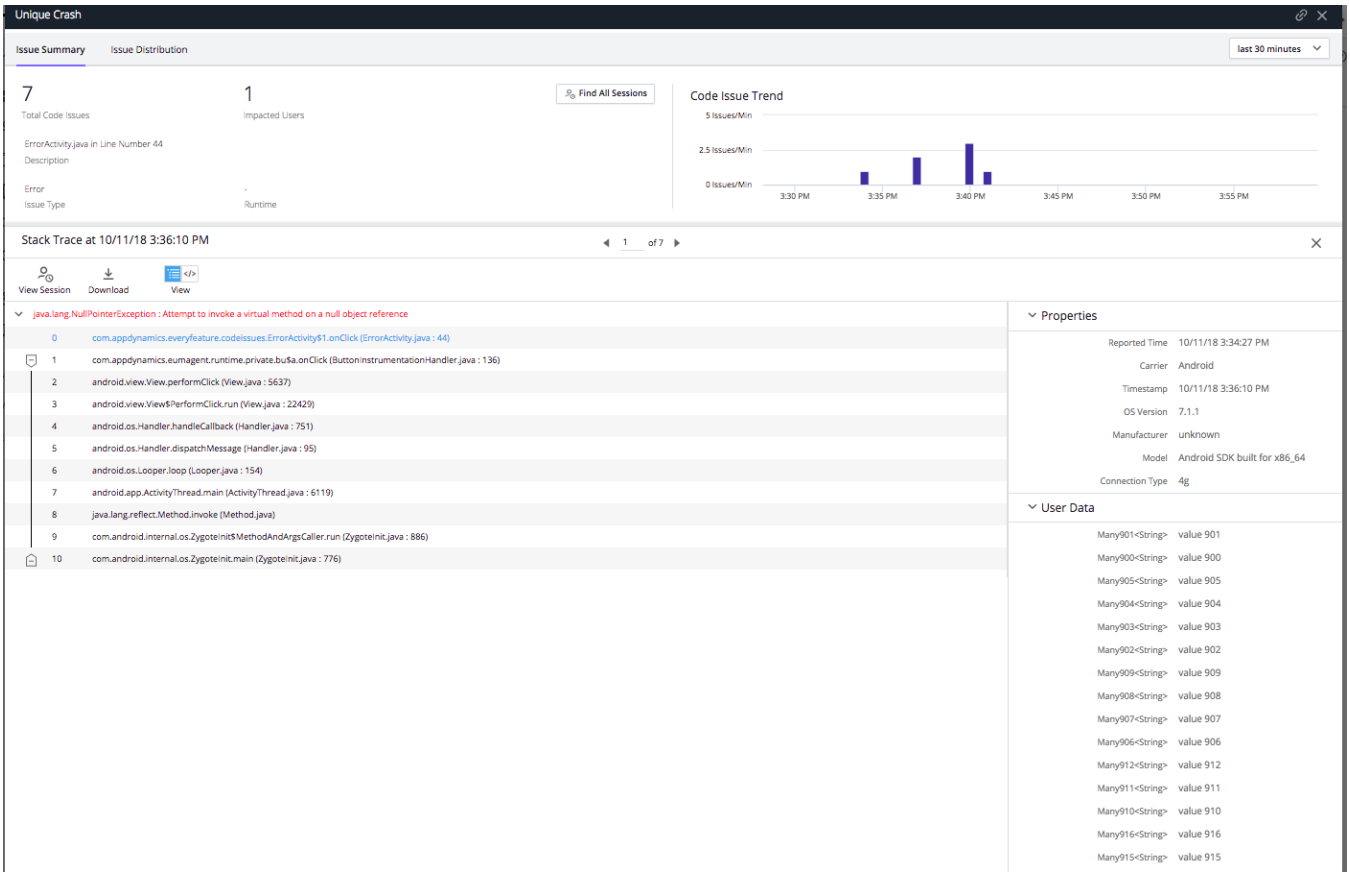
Unique Code Issues

Multiple code issues can be caused by the same underlying issue. The **Unique Code Issues** panel displays a list of code issues grouped by common characteristics and displays basic information about the issue.



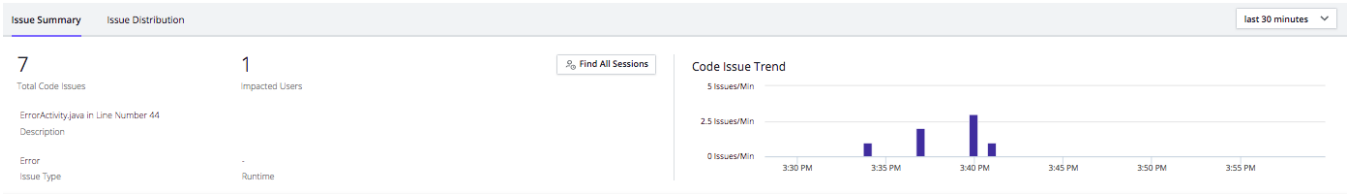
Unique Issue Details

To see more detail per code issue, select the unique code issue that interests you and click **Details**. The **Unique Issue** dialog has the two tabs Issue Summary and Issue Distribution.



Issue Summary

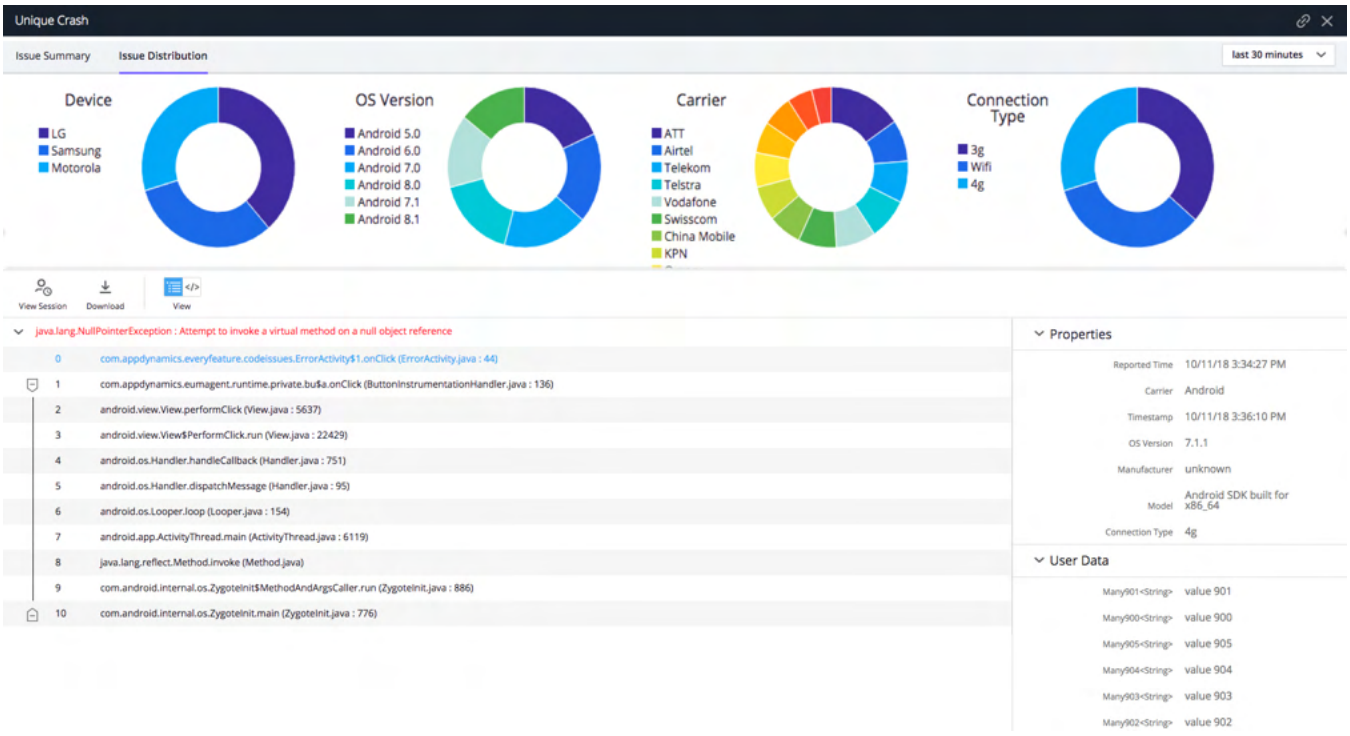
The Issue Summary tab has two main panels. The top panel shows summary information such as the total code issues, the issue type, impacted users, the runtime, and a code issue trend bar graph.



The bottom panel shows a sequence of threads in the stack trace that you can expand for more information, a **Properties** section with general information about the device, and a **User Data** section with any set user data.

Issue Distribution

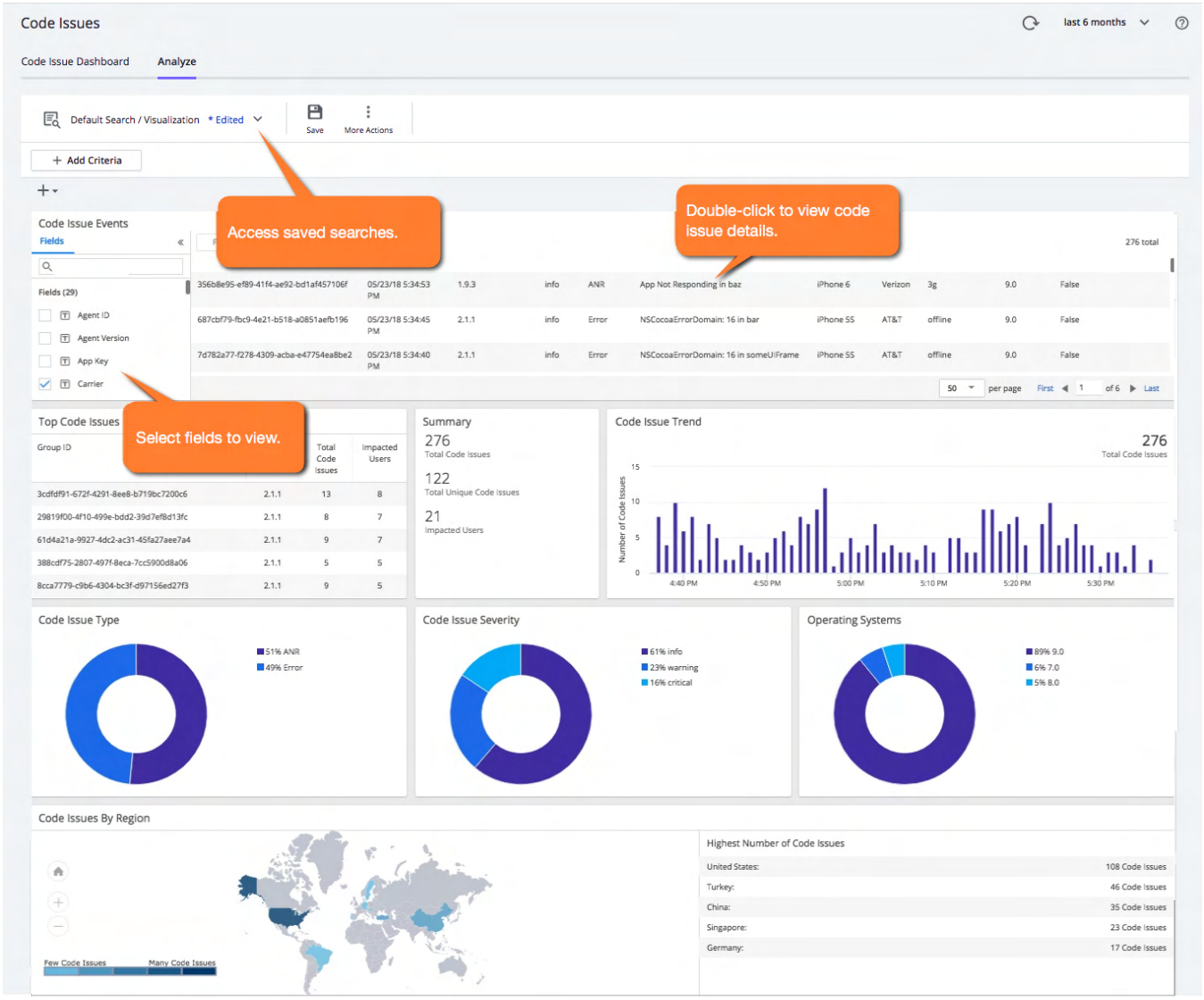
The Issue Distribution tab displays the same stack trace panel, but the top panel dashboard instead displays the distribution charts for the code issue.



Code Issues Analyze

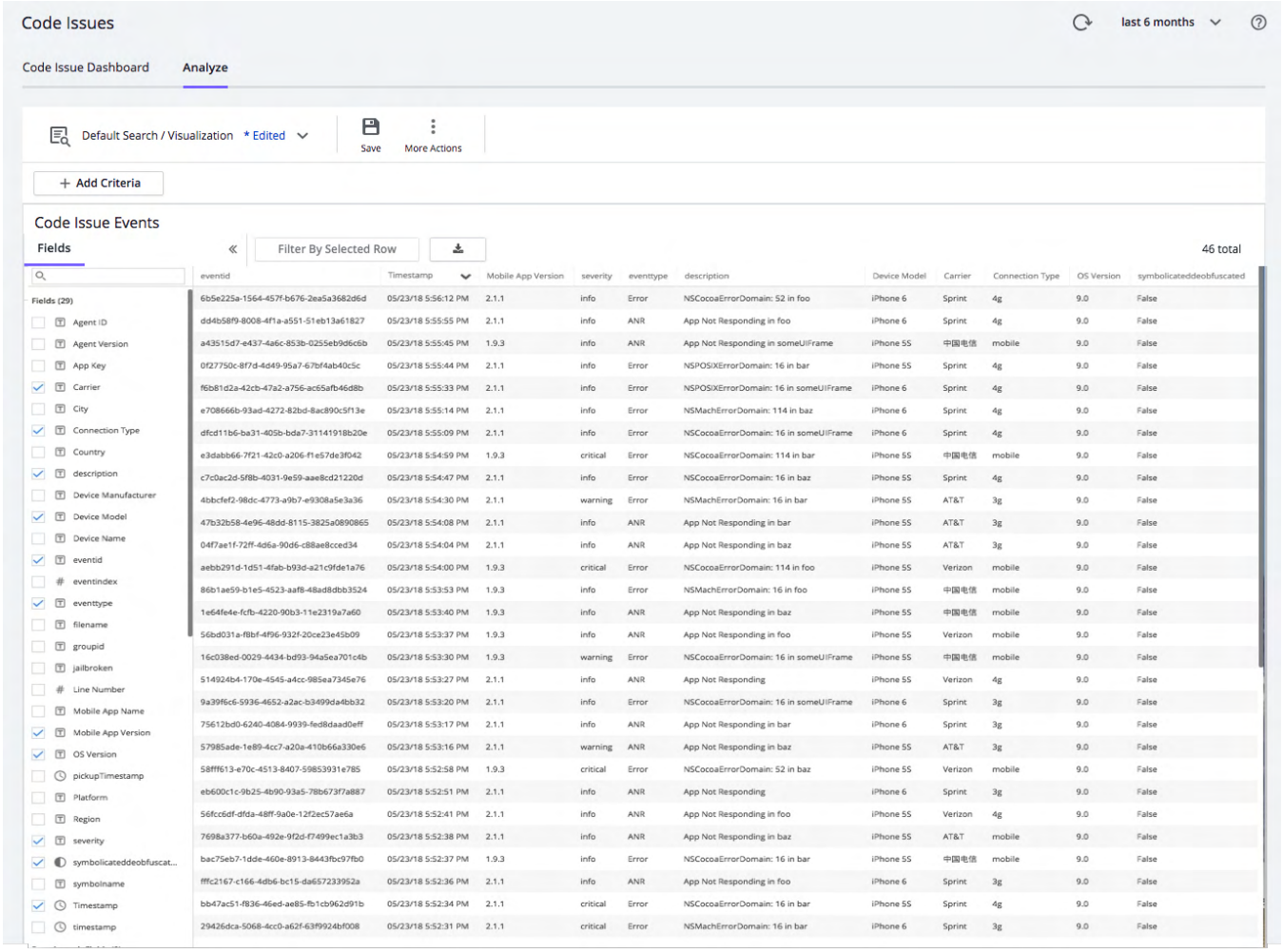
The **Code Issues Analyze** page has the **Code Issue Events** widget for viewing a list of current code issue snapshots, with one row for each snapshot, widgets for visualizing code issues data, and widgets for highlighting key data points.

Code Issues Analyze enables you to view and analyze the results based on cumulative data based on configured criteria. You can delete, re-add, re-size, and drag-and-drop to move all of the widgets. You can set criteria to visualize a subset of the data. Every code issue event in **Code Issues Analyze** is collected and stored by the AppDynamics Events Service.



Code Issue Events Widget

The **Code Issue Events** widget enables you to scan individual code issues, filter, and sort to get exactly the code issue in which you are interested. The columns of the table are fields representing the code issue properties. See Code Issue Snapshot Properties.



Other Code Issue Widgets

The other widgets fall into the following three categories: tables, charts, a summary, and a hybrid.

Table Widgets

The table widgets display rows of code issues in tabular form. The **Code Issue Events** widget lists the code issue snapshots.

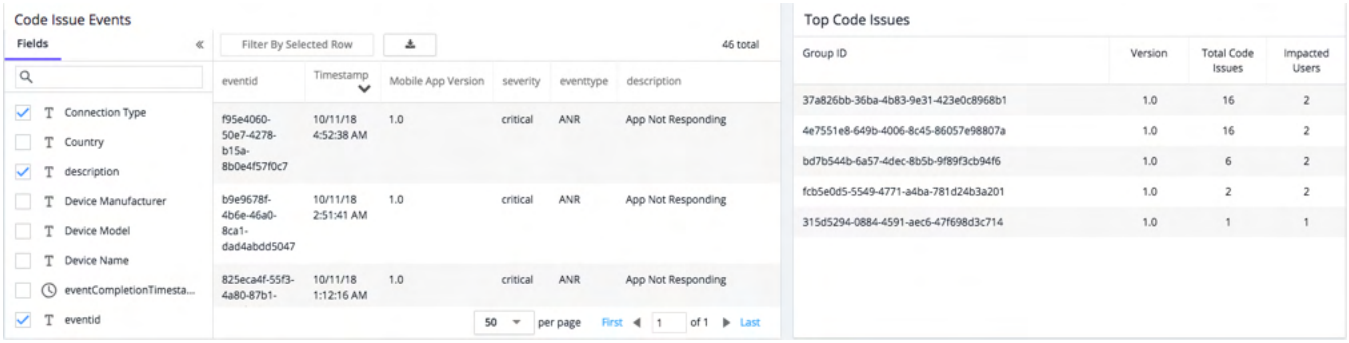
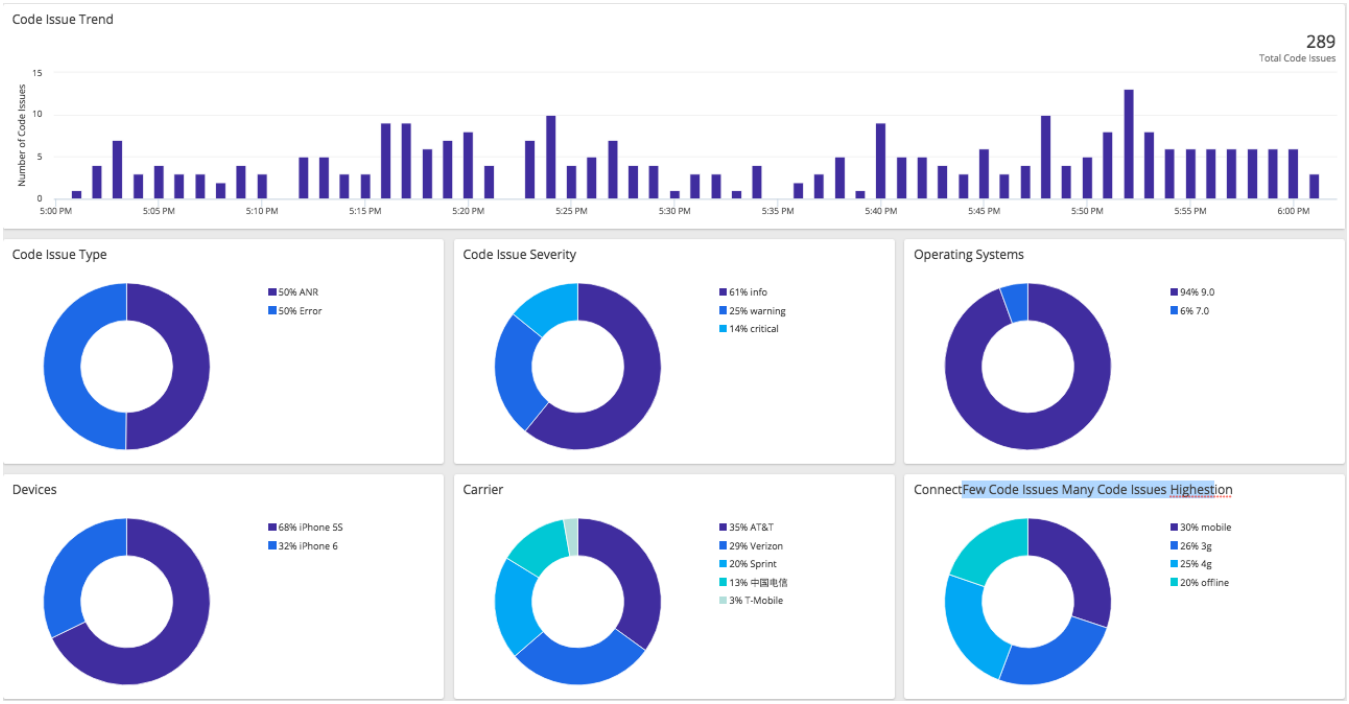


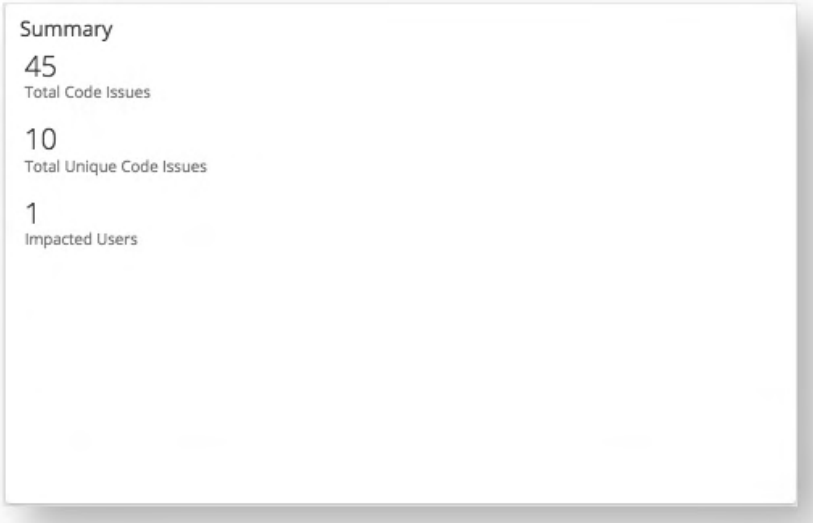
Chart Widgets

The chart widgets visualize the code issue data. For example, you can view the code issues data in pie charts showing the ratio of ANRs to caught errors /exceptions, severity levels, operating systems, and so on.



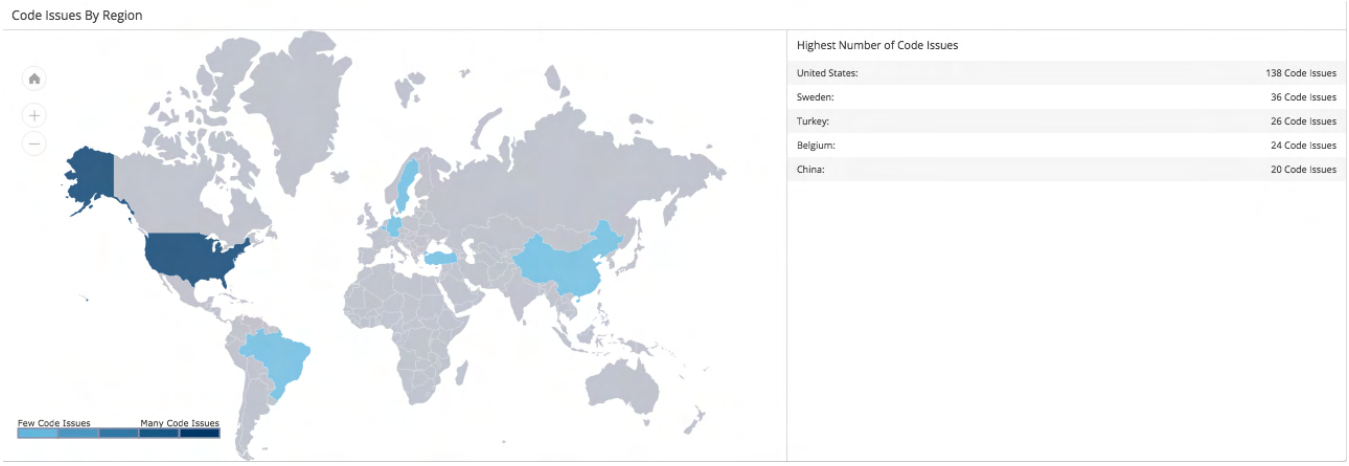
Summary Widget

The **Summary** widget lists totals for key statistics.



Hybrid Widget

The **Code Issues By Region** widget is a hybrid, displaying geographic data in a map and listing the data points in a table.



Code Issues in Mobile Analytics

Code issue snapshots can also be viewed in the context of a separate product, AppDynamics Mobile Analytics. It offers additional mechanisms to analyze and visualize code issue patterns, including:

- Additional predefined widgets, such as the funnel widget
- ADQL for searching the data
- Custom widgets
- Manipulating multiple dashboard types
- Longer retention time for storing data

AppDynamics Analytics Mobile Non-Fatal Issues Data requires a license separate from the Mobile RUM license.

Code Issue Snapshots

Code issue snapshots provide the detailed information for one code issue. A code issue snapshot is created when an instance of an instrumented application experiences an ANR, slow UI, or manually reports a caught exception. To manually report caught exceptions, you use either the iOS SDK or the Android SDK.

- Thread that contains a function that is not responding or has a caught exception
- Source file containing the function that is not responding or has the caught exception
- Line number in the source file, if available
- Stack trace of the application at the time of the code issue

For iOS applications, code issue snapshots are based on:

- ANRs
- Handled Objective-C exceptions
- Slow UI

For Android applications, crash snapshots are based on:

- ANRs
- Handled Java exceptions
- Slow UI

Code Issue Snapshot Details

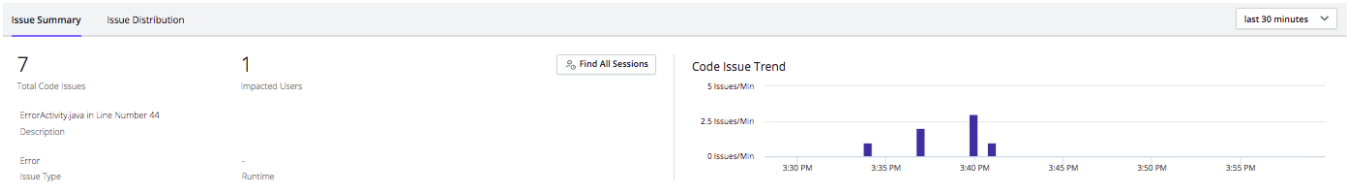
The **Issue Snapshot Details** dialog has the two panels **Issue Summary** and **Issue Distribution**. The **Issue Summary** panel displays basic information about the code issue and the stack trace. The **Issue Distribution** panel displays widgets with distributed issue metrics by code issue group as well as the stack trace.

View Code Issue Snapshot Details

1. Navigate to **Code Issues > Analyze**.
2. From the **Code Issue Events** widget, double-click one of the code issue snapshots.

Issue Summary

The **Issues Summary** panel of the snapshot displays the key properties of the snapshot. Unlike with crashes, code issues do not include breadcrumbs.

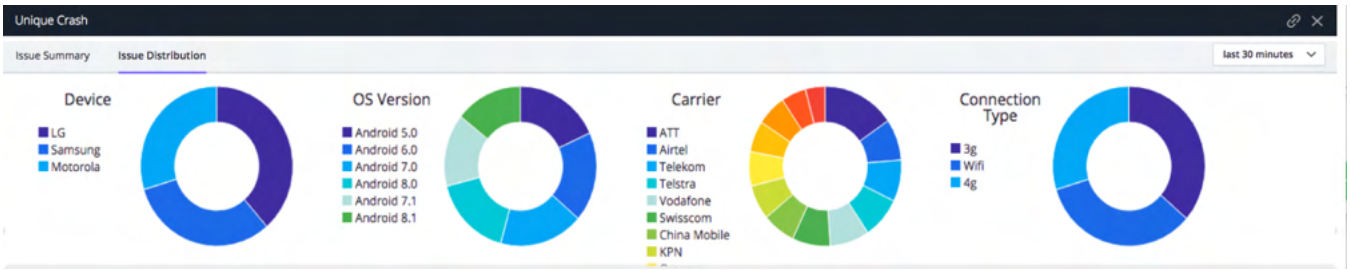


The **Stack Trace** panel displays the call stack of the code issue, showing the thread in which the caught exception or ANR occurred. This is the full stack trace for this specific code issue, not the aggregated stack trace shown in the **Unique Issue Dashboard**. You can view the formatted stack trace that groups threads and parses the output or the raw log output for the stack trace.

Click **Download** to grab the stack trace in a file that you can forward to developers or port to logging platforms. To see the code issue in the context of the mobile session, click **View Session**.

Issue Distribution

The **Issue Distribution** panel displays smaller-sized widgets from the **Analyze** tab with distributed metrics for the code issue group.



Code Issue Snapshot Properties

This page lists the code issue snapshot properties. They appear in the **Code Issue Snapshot Details** panel when you open a code issue snapshot from **Code Issue Analyze**.

- **Agent ID:** The ID of the Mobile Agent reporting the code issue.
- **Agent Version:** The version of the Mobile Agent reporting the code issue.
- **Application Key:** The application key.
- **Carrier**: The name of the mobile carrier.
- **City:** The originating city.
- **Client Time:** The timestamp of a device when the beacon was made.
- **Connection Type**: The active connection type at the time of the code issue, if known.
- **Country:** The originating country.
- **Description:** The type of code issue, such as an application not responding (ANR) or a caught non-fatal exception.
- **Device/Manufacturer/Model**: The model or manufacturer name of the mobile device on which the code issue occurred.
- **Event Type:** Type of code issue (for example, ANR, Caught Exception)
- **Jailbroken:** Flag indicating whether the mobile device has been jailbroken.
- **Group ID:** The event group id after grouping.
- **Mobile App**: The name of the app that had the code issue.
- **OS Version**: The operating system version of the mobile device on which the code issue occurred.
- **Platform**: iOS or Android.
- **Region:** The originating region, such as the province or state.
- **Request Timestamp**: When the beacon arrived at the Collector. Snapshot only.
- **Severity:** The severity level set for the code issue.
- **Symbolication/Obfuscation:** Flag indicating whether the application code has been symbolicated/obfuscated.
- **User Data**: Any user data you might have added.
- **Version**: The version string of the application that had the code issue.

Code Issue Alerts

You can configure AppDynamics to send you alerts when your mobile application has code issues. You can configure email alerts for health rule violates based on the default health rule or by manually creating a policy with a notification action.

Set Up Alerts for Code Issues

This basic workflow describes how to set up alerts for code issues:

1. Report Errors: In your application code, use the iOS, Android, or Xamarin SDK to report errors. Set the severity to either `INFO`, `WARNING`, or `CRITICAL`. Only errors with the severity of `CRITICAL` trigger alerts with the default health rule.
2. Configure Thresholds: Set the warning and critical thresholds for application not responding issues:
    a. Navigate to **Configuration > Mobile App Group Configuration > Settings**.
    b. From **Configure Application Not Responding Thresholds**, set the values for **Warning Threshold** and **Critical Threshold**. Any value that is less than the warning threshold will be labeled as **Info**.
3. View alerts for code issues.

Report Errors with the Mobile SDKs

See "Report Errors and Exceptions" (iOS/Android/Xamarin) to report errors and set the severity for iOS, Android, or Xamarin applications.

Configure Threshold for Code Issue Alerts

You can configure the thresholds for ANRs that must be reached before you are alerted. See Configure Application Not Responding Thresholds.

View Code Issue Alerts

The health rule **Critical Code Issues exist** is used to send alerts when one if the following occurs:

- You have used the Android or iOS SDK to report an error with the severity of `CRITICAL`.
- The time that your application has not responded exceeded the critical threshold.

After you set up the email alerts, when a health rule is violated, an email will be sent indicating whether there was a warning or critical alert and providing links to view code issue and session details.

Custom Data

Mobile RUM provides substantial information on crashes and network and server-related performance. You may want, however, to customize the Mobile Agent to collect other metrics and data to understand how your application is performing for your users.

For information on customizing the agent, including sample code, see:

- Customize the iOS Instrumentation
- Customize the Android Instrumentation

To see the results of your data collection, you use the **Custom Data** view.

Custom Data Types

There are three kinds of custom data that you can collect:

- Info Points
- Custom Timers
- Custom Metrics

> There are two other types of custom data: the breadcrumb and user data. Breadcrumbs display only in Crash Snapshots. User data is available in **Crash Analyze** and **Network Request Analyze**. See the agent SDK docs for more information.

Info Points

An information point allows you to collect information on how a specific method in your code is performing. How many times was a particular method executed? Was an exception thrown? If you need to monitor multiple methods, you can create multiple info points.

If you know there are important methods you always want to monitor, create static info points as described in Customize the iOS Instrumentation and Customize the Android Instrumentation.

Custom Timers

Custom timers allow you to time how long something takes between any arbitrary start point and end point, even if they span multiple methods. For example, how long did it take to repaint a frame buffer? How long did the app take to start up?

Custom Metrics

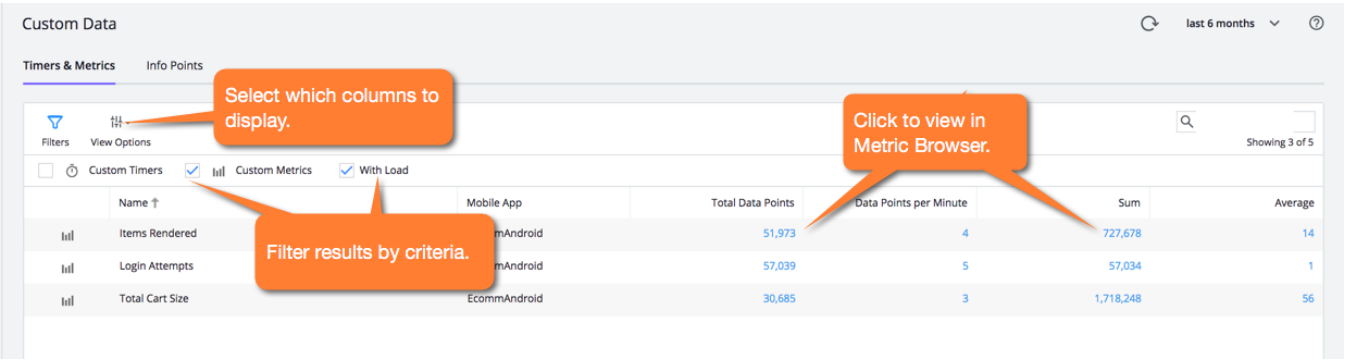
Custom metrics allow you to collect data on any metric you define in your application. Metrics can be any integer-based data. Over the specified time frame, Mobile RUM calculates the minimum, maximum, average, sum, and count of these values.

View Custom Data

Your data is shown in tabular format in two tabs.

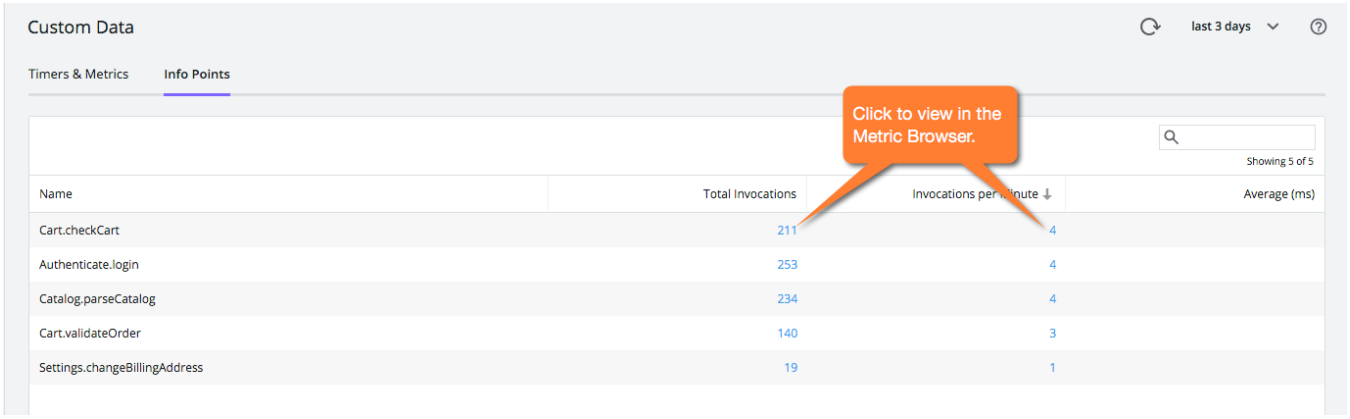
Custom Timers & Metrics

Click on any value (in blue) to display the same metric in the Metric Browser.



Info Points

The Info Points tab shows any current info points.


Custom Data
last 3 days
Timers & Metrics
Info Points
Click to view in the Metric Browser.
Showing 5 of 5
Name
Total Invocations
Invocations per Minute
Average (ms)
Cart.checkCart
211
4
Authenticate.login
253
4
Catalog.parseCatalog
234
4
Cart.validateOrder
140
3
Settings.changeBillingAddress
19
1

Configure the Controller UI for Mobile RUM

Using the Controller UI, you can configure:

- How to rename mobile requests
- Thresholds that cause network request snapshots to be considered slow, very slow or stalled
- Percentile levels to display, if any
- Which network requests are sent to the Event Service
- If the IP address from which the request comes should be stored

To configure Mobile RUM from the Controller UI, your user account must belong to a role that has the **Configure EUM** permission. See End User Monitoring Permissions.

Access the Mobile Container Configuration

To access mobile request configuration:

1. Open the mobile application in which you are interested.
2. From the left-hand navigation menu, click **Configuration**.
3. From the **Configuration** page, click **Mobile App Group Configuration >**.

Configure Mobile Network Request Naming

This page describes how to modify the default naming configuration for network requests and create include/exclude naming rules for network requests.

> If you create a rule for capturing mobile network requests and later disable the rule, the URLs that were formerly captured by the rule will be displayed with the network request name "/" unless you have another rule that captures the URL. To avoid this from occurring, you can create a custom rule that serves as a catch-all for URLs that are not captured by custom rules. See Network Request Limits.

Default Network Request Naming Configuration

By default, AppDynamics names network requests using:

- Hostname
- First two segments of the URL

For example, if an application makes this HTTP request:

```
http://myapp.com/friends/profiles/12345
```

The default name that is displayed in the Controller UI for that request is:

```
myapp.com/friends/profiles
```

If this is adequate for your needs, you can leave the default setting.

> The naming rules you configure here apply to all the mobile applications that are in the same Mobile App Group.

Modify the Default Naming Configuration Rule

You may want to configure a different default rule for naming your network requests to help you visualize the parts of your application more clearly. The task is similar to configuring naming rules for business transactions on the server side. Try to group logically related requests together while keeping unrelated requests in separate groups.

- If the default hostname and first two segments of the URL for all your requests are identical, you might want to name the requests based on the last segments or a selection of non-contiguous segments of the URL to distinguish among requests in the network requests list.
- You can also name the requests based on query parameters. For example, if the request passes an order number, you could specify that the value of the `order-number` query parameter is used in the network request name.
- You can also base the name on a regular expression run on the URL. AppDynamics uses the Java libraries for regular expressions. See:
  - Tutorial: http://download.oracle.com/javase/tutorial/essential/regex/index.html
  - Javadoc: http://download.oracle.com/javase/1.5.0/docs/api/java/util/regex/Pattern.html

Modify the Default Network Request Naming Rule

The default configuration covers how all your requests are named if you do not customize them further.

1. From the **Network Request** tab, scroll down to the **Include Rules** section.
2. Double-click **Default Naming Configuration**.
3. In the **Include Rule** dialog, select the elements you want to use for your default network request naming.
4. Click **OK**.
5. Click **Save**.

Create Mobile Include Rules

By default, the same request naming rule is applied to every URL that your application requests. If you want to apply different naming rules to different URLs, create include rules.

For example, if some requests call your own in-house server and others call out to a third-party API, you may want to see all the third-party API calls as one network request and use the default naming rules for the calls to your own server. You would create a custom naming rule that matches the third party calls and uses only the host in the default rule name or perhaps also include certain query parameters.

Create an Include Rule

1. From the **Network Request** tab, scroll down to the **Include Rules** section.
2. Click **Add**.

3. In the **Include Rule** dialog, enter a name for the custom rule that you are creating.
4. Check the **Enabled** check box to enable the rule.
5. Select the check boxes and radio buttons and enter the match criteria for AppDynamics to use to name network requests.
6. Click **OK**.

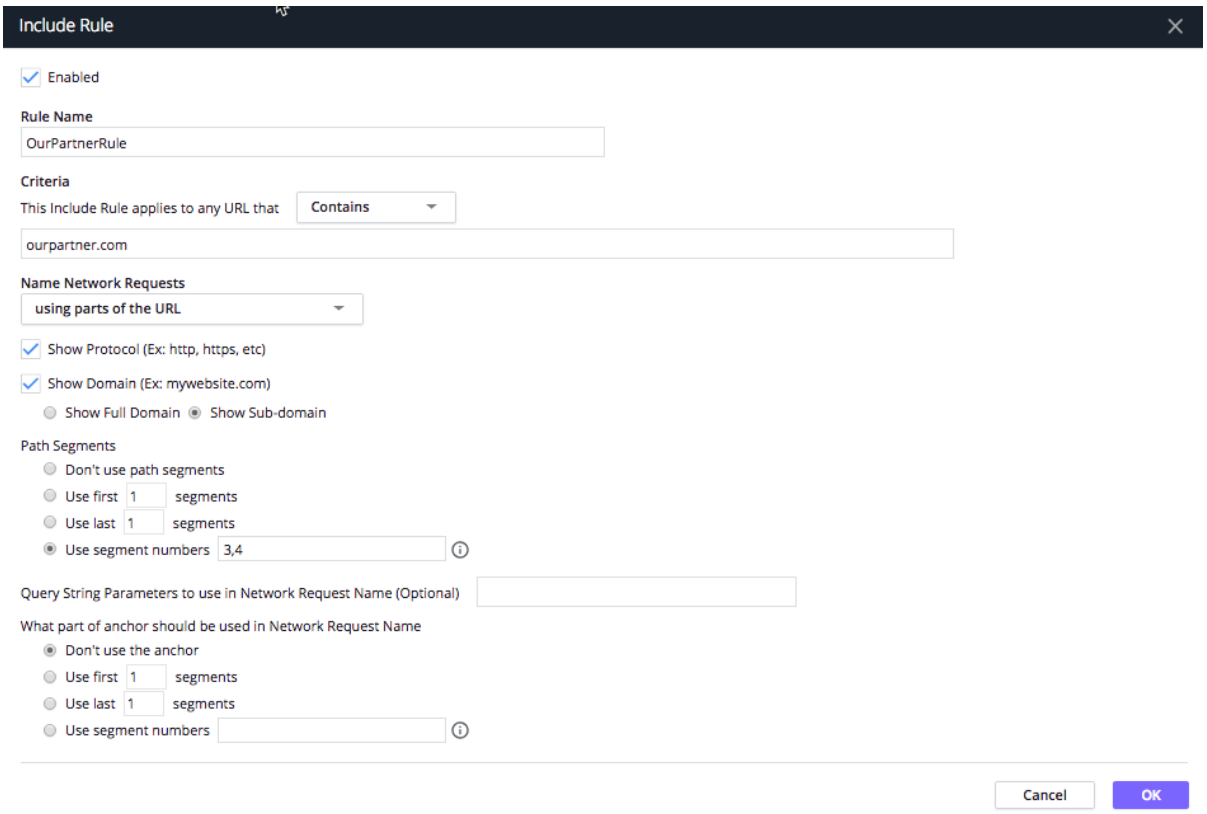
Guidelines for Using Regular Expressions in Mobile Include Rules

When using regular expressions to match URLs in your include rules, you should note the following:

- URL strings are case-sensitive. So, although the page names displayed in the Network Request Dashboard are converted to lowercase, your regular expressions still need to match the case used in URLs that your include rule are trying to capture.
- Your regular expression should match the entire URL, from beginning to end, not just a section; otherwise, the rule will not match. This differs from using regular expressions in custom match rules for naming transactions, which only need to match sections of the URL.

Sample Include Rule

The following rule creates a custom match rule for requests in which the URL contains "ourpartner.com". This rule uses the protocol, the subdomain and the third and fourth segments of the URL in the network request name.



You can temporarily cancel the application of a custom naming rule by clearing the **Enabled** checkbox in the custom rule configuration. In this case, the default naming rule is applied to requests that would have been named by the disabled custom rule. To remove the rule permanently, select the custom rule in the **Custom Naming Rules** list and click the **Delete** icon.

Create Mobile Exclude Rules

If there are certain types of requests that you do not want to monitor, create custom exclude rules for them based on the URL and/or the application name. Excluded network requests are not reported or counted toward the network request limit of 500 requests per controller application.

Create an Exclude Rule

1. From the Network Request tab, scroll down to the **Exclude Rules** section.
2. Click **Add**.
3. In the **Exclude Rule** dialog, enter a name for the exclude rule that you are creating.
4. Check the **Enabled** check box to enable the rule.
5. Select the check boxes and radio buttons and enter the match criteria for AppDynamics to use to name network requests.
6. Click **OK**.

You can temporarily cancel the application of an exclude rule by clearing the **Enabled** checkbox in the exclude rule configuration. To remove the rule permanently, select the exclude rule in the **Exclude Rules** list and click the **Delete** icon.

Change Priority of Rules

Rules are evaluated in the order that they appear in the include or exclude list. You can change the priority of the rules by dragging and dropping rules towards the top (higher priority) or towards the bottom of the list (lower priority). Custom rules are always evaluated before the default naming rule, beginning with the custom rule that has the highest priority.

Configure Mobile Network Request Thresholds

Related pages:

- Network Request Limits
- Network Requests

The Mobile Agent uses configurable thresholds to determine whether network request time is normal, slow, very slow or stalled. It uses these thresholds:

- To decide whether to create a mobile request snapshot
- For labeling network request experience in a network request snapshot

Default Network Request Thresholds

By default, the Mobile Agent uses these default values to determine whether a request is slow or stalled:

- **Slow:** Greater than 3 standard deviations
- **Very slow:** Greater than 4 standard deviations
- **Stall:** Greater than 45000 ms

You should configure these defaults to conform to your own criteria for your mobile applications. An absolute threshold rather than one based on the standard deviation is often more appropriate for mobile applications.

Configure Network Request Thresholds

1. From the **Mobile App Group Configuration** page, click the Settings tab.
2. In the **Thresholds for Slow End User Experience** section, set the thresholds for slow, very slow, and stalled in milliseconds or by standard deviations.
3. Click **Save**.

Configure Mobile Percentile Metrics

Parts of the Controller UI for Mobile RUM rely on the processing done by the Events Service, including some of the widgets in the **Mobile App Dashboard Overview**, **Network Requests Analyze**, and **Crash Analyze**.

For the Mobile App Dashboard, **Network Requests Analyze**, and **Crash Analyze**, you can choose widgets to display the metrics collected using either averages or percentiles.

A *percentile* is a measure that indicates a value below which a given percentage of values in a set falls: for example, the 99th percentile means that 99% of all values are below this level.

Using percentiles can be a good way to reduce the impact of extreme outliers in performance metrics, which can be useful in the often noisy environments of end-user experience monitoring. Percentiles also display in the **Metric Browser**.

You can:

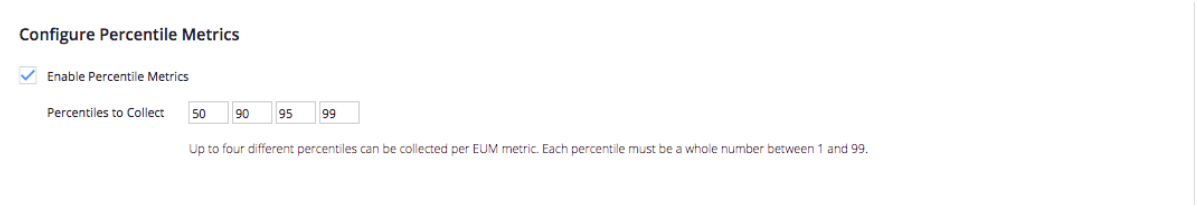
- Enable or disable percentile display of metrics.
- Set up to four different percentile levels to be applied to metrics.

Access Configure Percentile Metrics

From the **Configuration > Mobile App Group Configuration**, click the Settings tab and navigate to the **Configure Percentile Metrics** section.

Configure Percentile Metrics

You can change the percentiles which are used to evaluate data via the Events Service.



1. Check the box to enable percentile metric display.
2. Add up to four percentile levels to collect. Each value must be a whole number between 1 and 99.

If you change the percentiles to collect new values, it takes time for the recalculation to occur. If you review the **Metric Browser,** you will see metrics based on the old percentiles until the moment that you update the values, when they change to reflect the change.

Configure Which Network Requests Are Sent to the Event Service

Related pages:

- Network Request Limits

Your mobile app may make various kinds of network requests, and not all of them may be equally important to monitor in detail. For example, any requests to Google Analytics that your app may make are useful but probably aren't as important to analyze as the requests it makes to your backend.

To manage the impact on your overall Event Service usage, you can create rules which specify which of these network requests should be sent on to the Event Service, either by excluding a request entirely, including a particular request or a sample of that request types by percentage, or by simply allowing the request to be sent on.

The behavior follows this pattern:

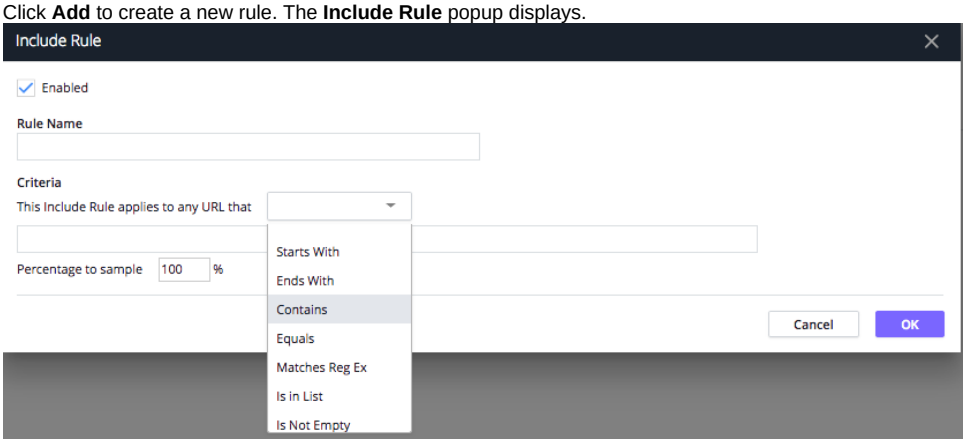
- If no rules are specified, data on all network requests are sent on.
- If exclude rules are specified, and a network request satisfies a rule, that data is *not* sent on.
- If include rules are specified, any network request that satisfies a rule is sent on, based on sampling defined by the percentage indicated in the rule.
- If both include and exclude rules are specified, a network request that satisfies an include rule but does *not* satisfy an exclude rule is sent on.

Access Network Requests Rules

From the **Mobile App Group Configuration** page, select the Event Service tab.

Configure Include Rules

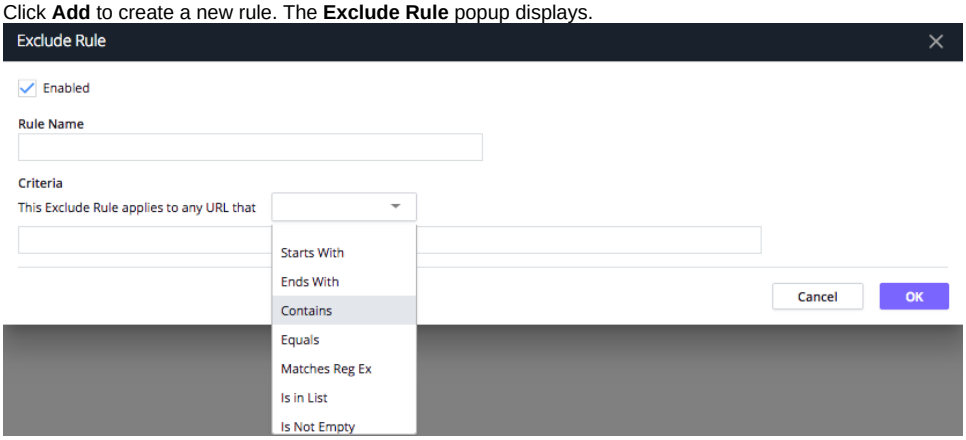
1. Click **Add** to create a new rule. The **Include Rule** popup displays.



2. Give your rule a display name.
3. Check **Enabled** to place the rule in force.
4. Specify the Network Request **URL** using any of the options in the dropdown.
5. If you want to sample this request type, select a percentage for sampling.
6. Click **OK**.

Configure Exclude Rules

1. Click **Add** to create a new rule. The **Exclude Rule** popup displays.



2. Specify a display name for the rule.
3. Check **Enabled** to place the rule in force.
4. Specify the Network Request **URL** using any of the options in the dropdown.
5. Specify the percentage of URLs to sample.
6. Click **OK**.

Change Priority of Rules

Rules are evaluated in the order that they appear in the include or exclude list. You can change the priority of the rules by dragging and dropping rules towards the top (higher priority) or towards the bottom of the list (lower priority). Custom rules are always evaluated before the default naming rule, beginning with the custom rule that has the highest priority.

Configure Request IP Address Storage - Mobile

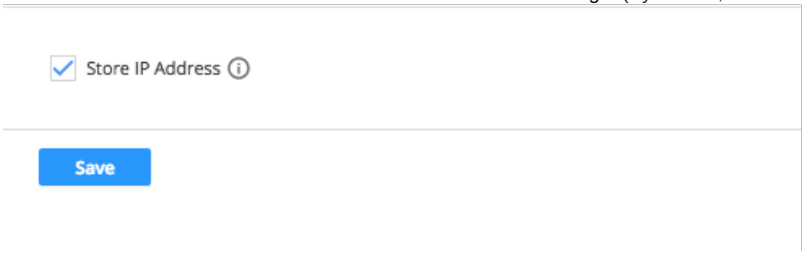
Related pages:

- Network Request Limits
- Network Requests

For security and privacy reasons, Mobile RUM does not store source IP addresses associated with network requests.

To have Mobile RUM store source IP addresses associated with network requests, following the instructions below.

1. From the **Mobile App Group Configuration** page, select the Settings tab.
2. Click the **Mobile Network Request Naming,** Thresholds & Percentiles tab.
3. Check the **Store IP Address** checkbox to enable IP address storage. (By default, the box is not checked.)



4. Click **Save**.

Configure Session Monitoring

Sessions begin when a user starts using your application and ends after a configurable period of user inactivity. Increasing the period of user inactivity enables sessions to capture more user events, but this could impact the session limits of your account. See Mobile Sessions.

Access Session Monitoring

1. Open the mobile application in which you are interested.
2. From the left-hand navigation menu, select **Configuration**.
3. Click **Mobile App Group Configuration >**.
4. Select the Settings tab.
5. Find the **Configure Session Monitoring** section.

Set Inactivity Timeout

From **Configure Session Monitoring:**

1. Set the number of minutes for the **Session Inactivity Timeout** field.
2. Click **Save**.

Configure Mobile Crash Alerts

New crashes are shown in the **Events** widget, and you can configure Mobile RUM to send you alerts when new crashes occur. See Crash Alerts.

To restrict the number of alerts for new crashes, follow these instructions configure the number of crashes that must occur before you are alerted.

Access Configure Mobile Crash Alerts

1. Open the mobile application in which you are interested.
2. On the left navigation bar, select **Configuration**.
3. Click **Mobile App Group Configuration >**.
4. Select the Settings tab.
5. Navigate to the **Configure Mobile Crash Alerts** section.

Set Threshold for New Crash Alerts

From **Configure Mobile Crash Alerts:**

1. Set the threshold for new crash alerts. The default value is 1.
2. Click **Save**.

Configure Application Not Responding Thresholds

The Mobile Agent uses configurable thresholds to set the severity level for application-not-responding (ANR) errors. The severity levels for ANRs are **warni ng** and **critical,** and these levels can be used to set health rules and receive alerts for health rule violations.

You can use Controller UI to configure thresholds for warning and critical severity levels. These thresholds will be propagated to the Mobile Agents without the need to redeploy the application.

Default Network Request Thresholds

The Mobile Agent uses the following default values to determine the severity level:

- **Warning:** greater than 3000 ms
- **Critical:** greater than 5000 ms

You should set custom thresholds best suited for your mobile applications.

Configure Application Not Responding Thresholds

1. From the **Mobile App Group Configuration** page, click the Settings tab.
2. In the **Configure Application Not Responding Thresholds** section, set the thresholds for **Warning Threshold** and **Critical Threshold** in milliseconds.
3. Click **Save**.

Instrument iOS Applications

With the iOS 14 release (supported by iOS Agent 20.10.0), Apple introduced a new data collection policy. See iOS Data Collection Disclosure.

Before you can monitor your iOS application, you will need to instrument your application to enable the iOS Agent to collect mobile metrics.

After you have set up and accessed Mobile RUM, follow these instructions:

1. Install the iOS SDK
2. Instrument an iOS Application
3. Customize the iOS Instrumentation (Optional)
4. Troubleshoot the iOS Instrumentation (Optional)

Install the iOS SDK

Related pages:

- Instrument an iOS Application
- Customize the iOS Instrumentation
- Troubleshoot the iOS Instrumentation

With the iOS 14 release (supported by iOS Agent 20.10.0), Apple introduced a new data collection policy. See iOS Data Collection Disclosure for details.

You can install the iOS SDK using CocoaPods or by manually downloading and installing it. We recommended that you use CocoaPods to install the iOS SDK because it handles the dependencies, the build settings, and simplifies upgrading.

Follow the instructions for your preferred installation method:

- CocoaPods Install
- Manual Install

iOS SDK Install Requirements

To install the iOS SDK, you must be running Xcode >= 7.

CocoaPods Install

1. Add the line below to a `target` block in your Podfile:

   ```
   pod 'AppDynamicsAgent'
   ```

   For example:

   ```
   platform :ios, '8.0'
   use_frameworks!

   target 'YourApp' do
     pod 'AppDynamicsAgent'
   end
   ```

2. In your project directory, run this command:

   ```
   pod install
   ```

Manual Install

To manually install the iOS SDK, follow these steps:

Download the iOS SDK

1. Navigate to the AppDynamics Download page.
2. From the **APP AGENT** list, check the **Mobile RUM Agent - iOS** checkbox.
3. Click **Download** for the latest iOS Agent that will appear in the **Releases** results. This downloads a file named `iOSAgent-<version>.zip`.

Add the Framework to the App

You can only use a Swift framework if that framework was built with the exact same version of Swift as the Xcode compiling your app.

1. Unzip `iOSAgent-<version>.zip`.
2. Import the `ADEUMInstrumentation.framework` folder into your project Xcode.

If you don't see that Xcode has automatically added the correct path(s), add a project-relative path to your frameworks.

Add Required Libraries

The AppDynamics iOS Agent requires these libraries:

- `SystemConfiguration.framework`
- `CoreTelephony.framework`
- `libz.dylib` or `.tbd`

To add the libraries:

1. Select the target that builds your app in Xcode.
2. Select the Build Phases tab.
3. Expand the **Link Binary With Libraries** section.
4. If any of the above libraries are not listed:
    - Click the **+** button.
    - Locate the missing library in the list.
    - Click **Add**.

    Repeat this step for each missing library.

Set the -ObjC Flag

You also need to add the `-ObjC` flag to Other Linker Flags.

1. Select your project in the **Project Navigator**.
2. In the target list, select the target that builds your application.
3. Select the Build Settings tab.
4. Scroll to **Linking** and open.
5. Go to **Other Linker Flags** and double-click to open the popup.
6. If the `-ObjC` flag is not in your list, click **+** and add it.

The `-ObjC` flag is necessary because the iOS Agent defines categories with methods that can be called at runtime, and by default, these methods are not loaded by the linker. As a result, you'll get an "unrecognized selector" runtime exception. The use of `-ObjC` ensures the methods will be loaded.

Upgrade the iOS SDK

Upgrade with CocoaPods

1. In your project directory, run this command: `$ pod update`
2. Rebuild.

Upgrade Manually

To pick up new features or to get crucial bug fixes you want to upgrade the iOS SDK in your app.

1. Download the updated SDK.
2. Replace the `.framework` file using the updated SDK.
3. Rebuild your app.

Instrument an iOS Application

Follow these steps to get your EUM App Key and instrument your iOS apps.

Get Your Application Key

After you completed the **Getting Started Wizard**, you were given an EUM App Key. You will need this key when you modify the source code. In some cases, multiple mobile applications can share the same key.

If you have completed the **Getting Started Wizard**, but don't have your EUM App Key, see Get Your Application Key.

Initialize the Agent

Generate a dSYM File

To enable the agent to provide human-readable information in the crash snapshots that are produced if the application crashes, compile with the `DWARF with dSYM` file option to generate a debug symbols (dSYM) file for the application. For more details about why you would want to do this, see Get Human-Readable Crash Snapshots.

1. In Xcode, select your project in the **Project Navigator**.
2. In the target list, select the target that builds your application.
3. Select the **Build Settings** tab.
4. In the **Build Options** section, make sure that the **Debug Information Format** is set to `DWARF with dSYM File`.
5. Rebuild the Xcode project.

Monitor Crashes with the dSYM File

This step is optional but highly recommended if you plan to monitor crashes. AppDynamics needs the dSYM file for the application to produce human-readable stack traces for crash snapshots.

For instructions, see Upload the dSYM File.

Customize the Instrumentation (Optional)

The `ADEumInstrumentation` class has additional methods to allow you to extend the kinds of data you can collect and aggregate using AppDynamics. There are five basic kinds of extensions that you can create:

- Custom timers: any arbitrary sequence of events within your code timed, even spanning multiple methods
- Custom metrics: any integer-based data you wish to collect
- User data: any string key/value pair you think might be useful
- Information points: how often a single method is invoked, and how long it takes to run
- Breadcrumbs: context for a crash

See Customize the iOS Instrumentation.

Configure the iOS Agent for On-Premises Deployments (Optional)

By default, the agent is configured to send its beacons to the EUM Cloud, which is an instance of the EUM Server running on AWS. If you wish to instrument your app in an environment that is using an on-premises version of the EUM Server, you need to modify the URL to which the agent sends its beacons. You do this using the `ADEumAgentConfiguration` object to set the Collector URL and the Screenshot Service URL to your on-premises EUM Server URL.

> The iOS Agent knows which path to use to make calls to different services (Collector/Screenshot Service). For example, if the EUM Server URL is `https://myEUMServerURL.com:7001`, the iOS Agent will know to use `https://myEUMServerURL.com:7001/eumcollector` to make requests to the EUM Collector. By specifying the Collector URL, you will *not* be able to use the SaaS deployment of the EUM Cloud for the Screenshot Service.

To get the EUM Server URL:

1. Open the Administration Console.
2. From the left navigation bar, click **Controller Settings**.

3. In the search field, enter **eum.beacon.host** or **eum.beacon.https.host** if you're using HTTPS.
4. Copy the value for the configuration. This is your EUM Server URL.

The code examples below show how to set the Collector URL and Screenshot Service URL using Objective-C and Swift.

Enable HTTP to Send Beacons to On-Premises EUM Servers

If you use an on-premises EUM Server and you wish to use HTTP to dispatch your beacons to the EUM Server, starting with iOS 9 you need to set a flag in your app's `info.plist` file to allow it to use the unsecured connection. By default, HTTPS is enforced in all iOS 9 applications by App Transport Security (ATS), and the iOS Agent complies with ATS when used with the EUM Cloud.

Customize the iOS Instrumentation

Related pages:

- Install the iOS SDK
- Instrument an iOS Application
- iOS SDK Documentation

Once you have instrumented your iOS application with the Mobile iOS SDK, you can also use the APIs exposed by the SDK to customize the data for your app that appears in the Controller UI.

The following sections show you how to use the iOS SDK to customize your instrumentation.

Because the agent stores data about events in a local buffer before reporting the information, you are recommended to use the APIs with discretion.

Collect Additional Types of Data

You can use methods available in the ADEUMInstrumentation class to collect six additional types of data:

Type of Data	Description	Specifications	Where Data is Displayed
Info points	How often a method is invoked, and how long it takes to run.	• Data is numeric • Names must consist of alphanumeric characters and/or spaces	• Metric Browser • Custom Data • Network Request Snapshots • Mobile Sessions • Network Request Analyze
Custom timers	Any arbitrary sequence of events within your code timed, even spanning multiple methods.	• Data is numeric • Metric names must consist of alphanumeric characters and/or spaces	• Metric Browser • Custom Data
Custom metrics	Any integer-based data you wish to collect.	• Data is numeric • Metric names must consist of alphanumeric characters and/or spaces	• Metric Browser • Custom Data
User data	Any string key/value pair you think might be useful.	• Data can be any type • Metric names have no restrictions	• Network Request Snapshots • Mobile Sessions • Network Request Analyze • Crash Snapshots
Breadcrumbs	The context for a crash.	• Data can be any data type • Metric names have no restrictions	• Network Request Snapshots • Mobile Sessions • Network Request Analyze • Crash Snapshots

User interaction	Capture when users press buttons, click on lists, and select text.	• Data can be any data type • Metric names have no restrictions	• Network Request Snapshots • Mobile Sessions • Network Request Analyze

When you have set up additional data types, the Mobile Agent packages that data in a mobile beacon. Normally, the beacon is transmitted when the instrumented app sends an HTTP request or when the app is restarted following a crash, but if custom data has been collected and neither of those events has occurred for at least five minutes, the custom data is sent at that time.

Info Points

Information points allow you to track how your own code is running. You can see how often a method is invoked, and how long it takes to run, by using `beginCall` and `endCall`, something like the following:

If an exception is thrown, it is also reported. This information appears in the **Custom Data** view in the Controller UI.

Custom Timers

Custom timers allow you to time any arbitrary sequence of events within your code, even spanning multiple methods, by using `startTimer` and `stopTimer`. For example, to track the time a user spends viewing a screen, the instrumentation could look like this:

This information appears in the **Custom Data** view of the Controller UI.

Calling `startTimerWithName` again with the same name value resets a named timer.

Custom Metrics

Any integer-based data can be passed to the agent. The first parameter to the `report.MetricWithName` call is the name you want the metric to appear under in the Controller UI. The metric name should only contain alphanumeric characters and spaces. Illegal characters are replaced by their ASCII hex value.

Reporting a metric called "My custom metric", for example, would look something like this:

```
[ADEumInstrumentation reportMetricWithName:@"My custom metric" value:<#VALUE HERE#>];
```

This information appears in the **Custom Data** view of the Controller UI.

User Data

You can set any string key/value pair you think might be useful. The first parameter to the `setUserData` call is the key you want to use, which must be unique across your application. The second is the value that you want to be assigned to the key.

For example:

This information is available in **Network Request Analyze** and is added to any crash snapshots that may be taken. Keys and values are limited to 2048 characters each.

You can also set user data with values of other types (Long, Boolean, Double, Date) using the following methods:

- `setUserDataLong:value`
- `setUserDataBoolean:value:`
- `setUserDataDouble:value:`
- `setUserDataDate:value:`

Breadcrumbs

Breadcrumbs allow you to situate a crash in the context of your user's experience. Set a breadcrumb when something interesting happens. If your application crashes at some point in the future, the breadcrumb will be displayed along with the crash report.

There are two ways of leaving breadcrumbs:

- Crash Reports Only
- Modal

Using this method means that breadcrumbs are reported in crash reports only.

```
+ (void)leaveBreadcrumb:(NSString *)breadcrumb
```

Using this method lets you fine tune where the breadcrumbs are reported, either only in crash reports or in crash reports *and* sessions.

```
+ (void)leaveBreadcrumb:(NSString *)breadcrumb mode:(ADEumBreadcrumbVisibility)mode
```

Where `mode` is either:

- `ADEumBreadcrumbVisibilityCrashesOnly`
- `ADEumBreadcrumbVisibilityCrashesAndSessions`

If the `breadcrumb` is over 2048 characters, it is truncated. If it is empty or `nil`, no breadcrumb is recorded. Each crash report displays the *most recent* 99 breadcrumbs.

Capture User Interactions

You can enable the iOS Agent to track certain UI events triggered by user interactions. Once user interactions have been captured, you can sort sessions by UI event and view UI events in the timeline of the session waterfall.

You can capture when users do one or all of the following:

- press buttons
- select table cells
- select text fields
- select text views

Security and Privacy Concerns

The interaction capture mode is disabled by default for security and privacy reasons as user interactions may contain sensitive information. Moreover, this potential security and privacy issue may be compounded if you enable both the capturing of UI interactions and screenshots.

Enable User Interaction Capture Mode

To enable user interaction capture mode, you assign the capture mode to the property `interactionCaptureMode` of the `ADEumAgentConfiguration` object. The instrumentation code example below configures the iOS Agent to capture all the supported types of user interactions.

```
ADEumAgentConfiguration *config = [[ADEumAgentConfiguration alloc] initWithAppKey: <#EUM_APP_KEY#>];
config.interactionCaptureMode = ADEumInteractionCaptureModeAll;
[ADEumInstrumentation initWithConfiguration:config];
```

You can also configure the iOS Agent to only capture one type of user interaction:

```
ADEumAgentConfiguration *config = [[ADEumAgentConfiguration alloc] initWithAppKey: <#EUM_APP_KEY#>];
config.interactionCaptureMode = ADEumInteractionCaptureModeButtonPressed;
[ADEumInstrumentation initWithConfiguration:config];
```

Configure and Take Screenshots

Mobile screenshots are enabled by default. The `takeScreenshot()` function limits screenshots to a maximum of 1 per 10s.

You can configure the Controller UI to automatically take screenshots or use the iOS SDK to manually take a screenshot as shown below:

Disable Screenshots

You can disable screenshots from the Controller UI or with the iOS SDK. To disable screenshots with the iOS SDK, set the property `screenshotsEnable` of the `ADEumAgentConfiguration` object to `NO` for Objective-C and `false` for Swift as shown below.

Block and Unblock Screenshots

You can also use the iOS SDK to block screenshots from being taken during the execution of a code block. This just temporarily blocks screenshots from being taken until you unblock screenshots. This enables you to stop taking screenshots in situations where users are entering personal data, such as on login and account screens.

The `ADEumInstrumentation` class provides the methods `blockScreenshots` and `unblockScreenshots` to block and unblock screenshots. If screenshots are disabled through the property `screenshotsEnabled` of the `ADEumAgentConfiguration` object or through the Controller UI, these methods have no effect. You can also call `screenshotsBlocked` to check if screenshots are being blocked.

The following example demonstrates how you could use the API to block and unblock screenshots for a user login.

Disable the Agent to Stop Sending User Data to the Collector

You can disable the agent to stop sending all data to the collector while the agent is initialized and running. For example, you can disable the agent if your app has an option for users to opt-out of monitoring for privacy reasons.

`shutdownAgent`

The `shutdownAgent` call stops outgoing data to the collector, and does not persist data on the device.

- The call only stops the traffic out of the agent.
- Once the agent has been initialized, the call cannot be removed, and a license will have been consumed.
- If you want to make this state permanent for a device, add code in UserDefaults to save the state and use that flag to conditionally initialize the agent in your code.

`restartAgent`

To re-enable the agent and reverse `shutdownAgent`, use `restartAgent`.

- This call will respect the server side calls that can remotely shutdown the agent in a similar way.
- The call is only in effect while the app is running.
- The call will be ignored if the agent has been remotely disabled.
- If the call is removed from memory and the app restarts, or the device is rebooted, the agent will be initialized as normal.

Add a Crash Reporting Callback

You may want to make crash report information that Mobile RUM collects available to other parts of your code, for example, to Google Analytics, if you are using it. To enable you to pass on summary crash information, you can set up a crash report runtime callback. To get a callback when the iOS Agent detects and then reports a crash, you need to implement the following protocol in your code:

```
@protocol ADEumCrashReportCallback <NSObject>

- (void)onCrashesReported:(NSArray<ADEumCrashReportSummary *> *)crashReportSummaries;

@end
```


This callback is invoked on your app's UI thread, so any significant work should be done on a separate work thread.

Each `ADEumCrashReportSummary` passed in has the following properties:

```
@interface ADEumCrashReportSummary : NSObject

/** Uniquely defines the crash, can be used as key to find full crash report. */
@property (nonatomic, readonly) NSString *crashId;

/** The exception name, may be `nil` if no `NSException` occured. */
@property (nonatomic, readonly) NSString * ADEUM_NULLABLE exceptionName;

/** The exception reason, may be `nil` if no `NSException` occured. */
@property (nonatomic, readonly) NSString * ADEUM_NULLABLE exceptionReason;

/** The Mach exception signal name */
@property (nonatomic, readonly) NSString *signalName;

/** The Mach exception signal code */
@property (nonatomic, readonly) NSString *signalCode;

@end
```

If you are sending the information to another analytics tool, such as Google Analytics, it is best to include all five properties:

- `exceptionName` and `exceptionReason` are optional and useful for a quick identification of what the crash is. These are only present if the crash cause occurred within an exception reporting runtime, such as Objective-C.
- `signalName` and `signalCode` are useful for quick identification of the crash. These are from the system and are independent of the runtime.
- For additional information, `crashId` can be used to look up the crash in the AppDynamics Controller UI.

For example, to print the crash information to iOS's logger, you could implement an `ADEumCrashReportCallback` class like this:

```
// assumes the containing object has "adopted" the protocol
- (void)onCrashesReported:(NSArray<ADEumCrashReportSummary *> *)summaries {
    for (ADEumCrashReportSummary *summary in summaries) {
        NSLog(@"Crash ID: %@", summary.crashId);
        NSLog(@"Signal: %@ (%@)", summary.signalName, summary.signalCode);
        NSLog(@"Exception Name:\n%@", summary.exceptionName);
        NSLog(@"Exception Reason:\n%@", summary.exceptionReason);
    }
}
```

You set the object that implements the `ADEumCrashReportCallback` protocol during agent configuration:

```
ADEumAgentConfiguration *config = [ADEumAgentConfiguration new];
config.crashReportCallback = myCrashReportCallback;
```

Your callback is invoked, on the main/UI thread, if a crash from a previous run is detected and collected. See the latest iOS SDK documentation for more information.

Disable Crash Reporting

Crash reporting is enabled by default, but you can manually disable crash reporting through the instrumentation configuration. If you are using other crash reporting tools, you might disable crash reporting to minimize conflicts and optimize the crash report results.

You can disable crash reporting by configuring the instrumentation with the `crashReportingEnabled` property as shown in the following code example.

Report Errors and Exceptions

You can report exceptions using the method `reportError` from the `ADEumInstrumentation` class. Reported exceptions will appear in session details.

The method can have the following two signatures:

Objective-C Function Signature	Description

`(void)reportError:(NSError *)error withSeverity (ADEumErrorSeverityLevel) severity;`	Use this signature to report errors, set the severity level of the issue, and send the stack trace. This function signature sends the stack trace by default. If you don't want to send the stack trace, use the function signature below with the additional argument `andStackTrace` and set its value to `NO`.
`(void)reportError:(NSError *)error withSeverity:(ADEumErrorSeverityLevel) severity andStackTrace:(BOOL)stacktrace;`	Use this signature to report errors, set the severity level of the issue, and explicitly specify whether the stack trace should be included. If you include the stack trace with the reported error by setting `stacktrace` to `YES`, you can view the stack trace in the **Code Issues Details** dialog. To report the error without the stack trace, set `stacktrace` to `NO`.

Severity Levels

You can also set one of the following severity levels for an issue. With the severity level, you can filter errors in the **Code Issues Dashboard** or **Code Issues Analyze**.

- `ADEumErrorSeverityLevelInfo`
- `ADEumErrorSeverityLevelWarning`
- `ADEumErrorSeverityLevelCritical`

Examples of Reporting Errors

The example below uses the API to report possible exceptions and set the severity level to `ADEumErrorSeverityLevelCritical` for a failed attempt to perform a file operation.

You can also create and report custom errors with the following. Note that because `reportError` is not passed the argument `andStackTrace`, by default, the stack trace is automatically included with the error.

Configure Application-Not-Responding (ANR) Detection

By default, the iOS Agent does not detect ANR issues, and when ANR detection is enabled, the ANR issues are reported without stack traces. You must manually enable ANR detection and set a flag to include stack traces through the iOS Agent configuration. For more information about ANR monitoring, see Code Issues. To specify thresholds for ANR issues, see Configure Application Not Responding Thresholds.

Enable ANR Detection

You enable the detection of ANR issues by configuring the instrumentation with the `anrDetectionEnabled` property as shown below.

Report Stack Traces with ANRs

In addition to enabling ANR detection, you set the property `anrStackTraceEnabled` to `YES` (Objective-C) or `true` (Swift) to report stack traces with the ANRs.

Configure Hybrid Application Support

By default, the iOS Agent instruments iOS WKWebViews, but does not collect and report Ajax calls. See Hybrid Application Support for an overview and an explanation of how it works.

You can configure the static or runtime configuration to disable hybrid application support or modify its behavior. The sections below show you how to change the defaults for hybrid support through either runtime or static configuration.

Runtime Configuration for Hybrid Application Support

The code example below disables the injection of the JavaScript Agent. By disabling the injection, the WKWebViews in your application will not be instrumented and Ajax calls will not be reported.

```
ADEumAgentConfiguration *adeumAgentConfig = [[ADEumAgentConfiguration alloc] initWithAppKey: <#EUM_APP_KEY#>];
// Disable the JavaScript Agent Injection
adeumAgentConfig.jsAgentEnabled = NO;
[ADEumInstrumentation initWithConfiguration:adeumAgentConfig];
```

The JavaScript Agent injection is enabled by default. To also enable the collection and reporting of Ajax calls:

```
ADEumAgentConfiguration *adeumAgentConfig = [[ADEumAgentConfiguration alloc] initWithAppKey: <#EUM_APP_KEY#>];
// Enable the collection and reporting of Ajax calls
adeumAgentConfig.jsAgentAjaxEnabled  = YES;
[ADEumInstrumentation initWithConfiguration:adeumAgentConfig];
```

Static Configuration for Hybrid Application Support

You should use static configuration for the following reasons:

- force the instrumentation of WKWebViews and/or Ajax calls (override the runtime configuration)
- disable hybrid support and override the runtime configuration
- set the URL to your self-hosted JavaScript Extension file

The table below describes the supported properties and provides the default value for the `info.plist` file.

Property	Default Value	Description
`serverJsAgentEnabled`	true	If the client receives a `false` for this flag, then the JavaScript Agent will be disabled. Thus, the WKWebViews and Ajax requests will not be monitored. The injection occurs during the creation of a new WKWebView. So, if a WKWebView is created when this flag is set to `false`, that particular WKWebView won't be instrumented even if the flag is subsequently set to `true`.
`ForceWebviewInstrumentation`	false	When set to `true`, the iOS Agent will inject the JavaScript Agent into the WKWebViews regardless of the runtime configuration.
`ForceAjaxInstrumentation`	`true`	When set to `true`, Ajax operations will always be collected and reported regardless of the runtime configuration.
`ADRUMExtUrlHttp`	http://cdn.appdynamics.com	The JavaScript Agent consists of two components: the base JavaScript Agent and the JavaScript Agent extension. The base JavaScript Agent is built into the Mobile Agent binary and injected according to the rules above. After initialization, the JavaScript Agent fetches the JavaScript Agent extension from the URLs specified by these properties.
`ADRUMExtUrlHttps`	https://cdn.appdynamics.com/	

Example Configuration

The example `info.plist` below forces the instrumentation of WKWebViews (overriding the runtime configuration), but does not force the collection and reporting of Ajax requests. The configuration also sets the URL where the JavaScript Extension file is obtained.

```
<plist>
  <dict>
    ...
    <key>ADEUM_Settings</key>
    <dict>
      <key>ForceWebviewInstrumentation</key>
      <true/>
      <key>ForceAjaxInstrumentation</key>
      <false/>
      <key>ADRUMExtUrlHttp</key>
      <string>http://<your-domain>/adrum.cdn</string>
      <key>ADRUMExtUrlHttps</key>
      <string>https://<your-domain>/adrum.cdn</string>
    </dict>
    ...
 </dict>
</plist>
```

Programmatically Control Sessions

By default, a mobile session ends after a period of user inactivity. For example, when a user opens your application, the session begins and only ends after the user stops using the app for a set period of time. When the user begins to use the application again, a new session begins.

Instead of having a period of inactivity to define the duration of a session, however, you can use the following API to programmatically control when sessions begin and end:

```
- (void)startNextSession
```

When you call the method `startNextSession` from the `ADEumInstrumentation` class, the current session ends and a new session begins. The API enables you to define and frame your sessions so that they align more closely with business goals and expected user flows. For example, you could use the API to define a session that tracks a purchase of a product or registers a new user.

Excessive use of this API will cause sessions to be throttled (excessive use is >10 calls per minute per iOS Agent, but is subject to change). When not using the API, sessions will fall back to the default of ending after a period of user inactivity.

Example of a Programmatically Controlled Session

In the example below, the current session ends and a new one begins when the check out is made.

Start and End Session Frames

You can use the `SessionFrame` API to create session frames that will appear in the session activity. Session frames provide context for what the user is doing during a session. With the API, you can improve the names of user screens and chronicle user flows within a business context.

Use Cases

The following are common use cases for the `SessionFrame` API:

- One `ViewController` performs multiple functions and you want more granular tracking of the individual functions.
- A user flow spans multiple ViewController or user interactions. For example, you could use the API to create the session frames "Login", "Product Selection", and "Purchase" to chronicle the user flow for purchases.
- You want to capture dynamic information based on user interactions to name session frames, such as an order ID.

SessionFrame API

The table below lists the three methods you can use with session frames. In short, you start a session frame with `startSessionFrame` and then use the returned `ADeumSessionFrame` object to rename and end the session frame.

Session Frame Example

In the following example, the `SessionFrame` API is used to track user activity during the checkout process.

Configure the Agent for Custom App Names

By default, AppDynamics automatically detects the name of your application. The application name is a string form of the bundle ID. Thus, if the bundle ID is `com.example.appdynamics.HelloWorld`, the application name will be "com.example.appdynamics.HelloWorld".

There may be cases, however, where you deploy essentially the same app binary with different bundle IDs to various regional app stores. To make sure all the data belonging to one app is collected and displayed together, despite varying bundle IDs, you can set a common name by giving the apps a custom name. To do this, set the application name property in the `ADEumAgentConfiguration` instance that you use to set up `ADEumInstrumentation`. See the latest iOS SDK documentation for more information.

```
@property (nonatomic, strong) NSString *applicationName;
```

Configure the Agent for Ignoring Some HTTP Requests

In some cases, HTTP requests using NSURL are used for internal purposes in an application and do not represent actual network requests. Metrics created based on these requests are not normally useful in tracking down issues, so preventing data on them from being collected can be useful. To ignore specific NSURL requests, set the excluded URL patterns property in the `ADEumAgentConfiguration` instance that you use to set up `ADEumInstrumentation`. Use the simplest regex possible. See the latest iOS SDK documentation for more information.

```
@property (nonatomic, strong) NSSet * excludedUrlPatterns;
```

Use the Agent with a Custom HTTP Library

The iOS Agent automatically detects network requests when the underlying implementation is handled by either by the `NSURLConnection` or the `NSURLSession` classes. This covers the great majority of iOS network requests. In some cases, however, mobile applications use custom HTTP libraries.

- To have the iOS Agent detect requests from a custom library, add request tracking code to your application manually, using the `ADEumHTTPRequestTracker` class.
- To set headers to allow correlation with server-side processing, use the `ADEumServerCorrelationHeaders` class.
- To configure the agent *to use your custom library* to deliver its beacons over HTTP, use the `ADEumCollectorChannel` protocol and the `ADEumAgentConfiguration` class.

Add Request Tracking

To add request tracking manually, you tell the agent when the request begins and when it ends. You also set properties to tell the agent the status of the response.

Start and complete tracking a request

To begin tracking an HTTP request, call the following method immediately before sending the request.

> ⚠ You must initialize the agent using one of the `ADEumInstrumentation`'s `initWithKey` methods before using this method.

```
@interface ADEumHTTPRequestTracker : NSObject
...
+ (ADEumHTTPRequestTracker *)requestTrackerWithURL:(NSURL *)url;
```

Where `url` is the URL being requested. This parameter must not be `nil`.

To complete tracking an HTTP request, immediately after receiving a response or an error, set the appropriate properties on the tracker object and call the following method to report the outcome of the request back to the agent. You should not continue to use this object after calling this method. To track another request, call `requestTrackerWithURL` again.

```
- (void)reportDone;
```

Properties to be set

The following properties should be set on the `requestTrackerWithURL` object to describe to the agent the results of the call.

```
@property (copy, nonatomic) NSError *error;
```

Indicates the failure to receive a response, if this occurred. If the request was successful, this should be `nil`.

```
@property (copy, nonatomic) NSNumber *statusCode;
```

Reports the HTTP status code of the response, if one was received.

- If a response was received, this should be an integer.
- If an error occurred and a response was not received, this should be `nil`.

```
@property (copy, nonatomic) NSDictionary *allHeaderFields;
```

Provides a dictionary representing the keys and values from the server's response header. The format of this dictionary should be identical to the `allHTTPHeadersFields` property of `NSURLRequest`. The dictionary elements consist of key/value pairs, where the key is the header key name and the value is the header value.

If an error occurred and a response was not received, this should be `nil`.

Example:

Given a request snippet like this:

```
- (NSData *)sendRequest:(NSURL *) url error:(NSError **)error {
      // implementation omitted
      NSData *result = nil;
      if (errorOccurred) {
          *error = theError;
      } else {
          result = responseBody;
      }
      return result;
  }
```

Adding the tracker could look something like this:

```
- (NSData *)sendRequest:(NSURL *)url error:(NSError **)error {
      ADEumHTTPRequestTracker *tracker = [ADEumHTTPRequestTracker requestTrackerWithURL:url];
      // implementation omitted
      NSData *result = nil;
      if (errorOccurred) {
          *error = theError;
          tracker.error = theError;
      } else {
          tracker.statusCode = theStatusCode;
          tracker.allHeaderFields = theResponseHeaders;
          result = responseBody;
      }
      [tracker reportDone];
      return result;
  }
```

Enable Server-Side Correlation

To enable correlation between your request and server-side processing, add specific headers to outgoing requests that the server-side agent can detect and return the headers obtained from the server-side agent in the response to make them available to the iOS Agent.

This is done automatically for standard HTTP libraries.

```
@interface ADEumServerCorrelationHeaders : NSObject
+ (NSDictionary *)generate;
@end
```

You must:

1. Call the `generate` method and set the generated headers *before* sending a request to the backend.
2. Report back the response headers, using the allHeaderFields property shown above.

Configure Agent to Use Custom HTTP Library

The iOS Agent uses HTTP to deliver its beacons. To have the agent use your custom HTTP library for this purpose, do the following.

1. Implement a class that conforms to this protocol:

```
/**
* Protocol for customizing the connection between the agent SDK and the collector.
*/
@protocol ADEumCollectorChannel <NSObject>

/**
* Sends a request synchronously and returns the response received, or an error.
*
* The semantics of this method are exactly equivalent to NSURLConnection's
* sendSynchronousRequest:returningResponse:error: method.
*
* @param request The URL request to load.
* @param response Out parameter for the URL response returned by the server.
* @param error Out parameter used if an error occurs while processing the request. May be NULL.
*/
- (NSData *)sendSynchronousRequest:(NSURLRequest *)request returningResponse:(NSURLResponse **)response
error:(NSError **)error;
@end
```

2. Set the `collectorChannel` property in `ADEumAgentConfiguration` before initializing `ADEumInstrumentation`, passing in an instance of your class that implements `ADEumCollectorChannel`. See the latest iOS SDK documentation for more information.

```
@property (nonatomic, strong) id<ADEumCollectorChannel> collectorChannel;
```

Transform URLs for Network Requests

When your application makes network requests, you may not want to report URLs containing sensitive information to the EUM Server. You can instead transform the network request URL before reporting it or ignore it altogether.

To do so:

1. Implement a network request callback that modifies or ignores specific URLs.
2. Register the network request callback in the initialization code.

Implement the Network Request Callback

The callback that modifies or ignore specific URLs is an implementation of the protocol below. The callback method networkRequestCallback is synchronous, so it is recommended that you return from the function quickly.

```
- (BOOL)networkRequestCallback:(ADEumHTTPRequestTracker *)networkRequest
```

Transforming URLs

The `networkRequestCallback` method, in general, should follow the steps below to transform URLs:

1. Identify specific URLs using techniques such as regex or pattern matching.
2. Modify the `url` property of the `ADEumHTTPRequestTracker` object. (Modifying other properties of the `ADEumHTTPRequestTracker` object will be ignored.)
3. Assign a valid URL to the `url` property.
4. Return `YES` (Objective-C) or `true` (Swift).

The first step is optional as you could choose to transform the URLs of all network requests.

In general, however, you would want to identify and transform URLs that contain sensitive information as implied in the example below.

Ignoring URLs

If the `networkRequestCallback` method returns false, the beacon is dropped. The general process for ignoring beacons is as follows:

1. Identify specific URLs using techniques such as regex or pattern matching.
2. Return `false`.

You could theoretically ignore all network requests by having the callback `networkRequestCallback` always return `NO` (Objective-C) or `false` (Swift):

In general, though, you would identify network requests that you didn't want to monitor and return `NO` (Objective-C) or `false` (Swift) to ignore the network request as implied by this example.

Register the Callback

After implementing the callback, you register the object implementing the protocol method in the initialization code as shown below. When the iOS Agent is ready to create a network request beacon, it will first call the callback with an `ADEumHTTPRequestTracker` object.

Enable Logging and Set Logging Level

You use the method `loggingLevel` to enable and set the logging level. You can set logging to one of the following levels:

- `ADEumLoggingLevelOff`
- `ADEumLoggingLevelAll`
- `ADEumLoggingLevelVerbose`
- `ADEumLoggingLevelDebug`
- `ADEumLoggingLevelInfo`
- `ADEumLoggingLevelWarn`
- `ADEumLoggingLevelError`

Use verbose, all, and debug levels of logging only for troubleshooting and be sure to turn off for production.

Examples:

iOS SDK Documentation

See the latest iOS SDK documentation or the previous versions listed below:

- https://sdkdocs.appdynamics.com/ios-sdk/21.5/html/
- https://sdkdocs.appdynamics.com/ios-sdk/21.2/html/
- https://sdkdocs.appdynamics.com/ios-sdk/20.12/html/
- https://sdkdocs.appdynamics.com/ios-sdk/20.10/html/
- https://sdkdocs.appdynamics.com/ios-sdk/20.8/html
- https://sdkdocs.appdynamics.com/ios-sdk/20.7/html
- https://sdkdocs.appdynamics.com/ios-sdk/2020.3/html

Upload the dSYM File

Related pages:

- Manage the dSYM Files with Bitcode Enabled
- Troubleshoot the iOS Instrumentation

AppDynamics needs the dSYM file for the application to produce human-readable stack traces for crash snapshots. For details about why you should do this, see Get Human-Readable Crash Snapshots.

If you update the application, you need to provide the new dSYM file for the new application version. The dSYM file contains a UUID that links it to a specific Xcode build, so AppDynamics can unambiguously match the correct dSYM file with an incoming crash report with no additional information.

To upload the dSYM file:

- Enable the dSYM file
- Set up your environment to upload the file automatically each time you build or upload the file manually

Enable the dSYM File

To enable the agent to provide human-readable information in the crash snapshots that are produced if the application crashes, compile with the `DWARF with dSYM` file option to create a debug symbols file for the application.

To enable dSYM:

1. In Xcode, select your project in the **Project Navigator**.
2. In the target list, select the target that builds your application.
3. Select the Build Settings tab.
4. In the **Build Options** section, make sure that the **Debugging Information Format** is set to `DWARF with dSYM File`.

Upload the dSYM File to AppDynamics Automatically with Each Build

Automating the upload of your dSYM file reduces the number of manual steps required for each build and ensures that all builds have appropriate dSYM files available for AppDynamics to use.

1. In Xcode, select your project from the **Project Navigator**.
2. Click on the application target.
3. Select the **Build Phase** tab in the **Settings** editor.
4. Click the **+** icon in the upper left corner of the main panel.
5. Select **New Run Script Phase** from the dropdown.
6. In the script box, add these lines:

```
export ADRUM_ACCOUNT_NAME="<Account_Name_HERE>" // From the View License - End User Monitoring section
of the License Page
export ADRUM_LICENSE_KEY="<License_Key_HERE>"  // From the View License - End User Monitoring section of
the License Page
SCRIPT=$(/usr/bin/find "${SRCROOT}" -name xcode_build_dsym_upload.sh | head -n 1)
/bin/sh "${SCRIPT}"
```

7. There are also some optional parameters you can set if desired. To set them, add the following line(s) after the second `export` statement above. Set to `1` to enable.

```
export ADRUM_UPLOAD_WHEN_BUILT_FOR_SIMULATOR=0
export ADRUM_TREAT_UPLOAD_FAILURES_AS_ERRORS=0
export ADRUM_EUM_PROCESSOR="<EUM_SERVER_URL>"
```

   The last statement should be used to set the URL for an on-prem version of the EUM Server.

Upload the dSYM File to AppDynamics Manually

To upload the file manually:

1. Get the dSYM file from Xcode
2. Upload the dSYM file to AppDynamics using the UI
   or
   Upload the dSYM File to AppDynamics Using the API
3. Check Uploaded dSYMs Using the REST API

Get the dSYM file from Xcode

1. In Xcode, run the Xcode build: **Product > Build**.
2. View the log navigator: **View > Navigators > Show Report Navigator**.

   > Older versions of Xcode used **Show Log Navigator**.

3. Click the log entry for the most recent build.
4. Near the end of the log, find and mouse over the log entry named `Generate <Your_App_Name>.app.dSYM`.
5. Click the button on the right side of the entry to expand it.
   The end of the displayed command is the path to the dSYM file.
6. Navigate to this dSYM file in the Finder.
7. Right-click on the dSYM file and choose **Compress**.
8. Upload to AppDynamics the `.zip` file that **Finder** generates.

Upload the dSYM File to AppDynamics Using the UI

1. From the Mobile App menu, click **Configuration**.
2. Click **Mobile App Configuration >**.
3. From **dSYM Mappings**, click **Upload dSYM package file for iOS crashes**.
4. From the **XCode dSYM package upload** dialog, click **Choose File**.
   The uploader expects a file with a `.zip` extension.
5. In the file browser locate the zipped dSYM file for the application that you are instrumenting and click **Open**.
6. Click **Upload**.

Upload the dSYM File to AppDynamics Using the REST API

The API uses HTTP basic authentication. The username is your EUM account name and the password is your EUM license key.

Set up your HTTP basic authentication credentials

1. In the upper-right corner of the Controller UI, click the gear icon ( ) and choose **License**.
2. From the Account Usage tab, scroll down to the **User Experience** section.
3. Click **Show** next to License Key to display the EUM license key. This is your password for authentication.

4. URL-encode the EUM account name and the EUM license key.

Send the dSYM File

Send the dSYM as a zip archive in the body of a `PUT` request to the following URI:

```
https://api.eum-appdynamics.com/v2/account/<EUM_Account_Name>/ios-dsym
```

You must set a Content-Type header, `-H Content-Type:application/octet-stream`, and your URL-encoded account name (the username) and license key (the password) to the call.

Sample Request and Response

This is a sample request and response using the REST API.

Upload Request

The following example uses `curl` to send a dSYMm file named `UISampleApp.app.dSYM.zip`. The EUM account name is "Example account" and the password is "Example-License-Key-4e8ec2ae6cfe", the EUM License Key. The plus signs replace spaces in the account name when the account name is URL-encoded.

```
curl -v -H Content-Type:application/octet-stream --upload-file UISampleApp.app.dSYM.zip --user Example account:
Example-License-Key-4e8ec2ae6cfe https://api.eum-appdynamics.com/v2/account/Example+account/ios-dsym
```

Upload Response

The successful output of the sample request looks like this:

```
* About to connect() to api.eum-appdynamics.com port 443 (#0)*   Trying ::1...
* connected
* Connected to api.eum-appdynamics.com (::1) port 443 (#0)
* Server auth using Basic with user 'Example+account'
> PUT /v2/account/Example+Account/ios-dsym  HTTP/1.1
> Authorization: Basic SW50ZXJuYWwrdGVzdCthY2NvdW50OlRlc3RBY2N0LTFlMzktNDVkMy05MzAzLTRlOGVjMmFlNmNmZQ==
> User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8y zlib/1.2.5
> Host: localhost:7001
> Accept: */*
> Content-Length: 0
> Expect: 100-continue
>
< HTTP/1.1 100 Continue
< HTTP/1.1 200 OK
< Content-Length: 0
< Server: Jetty(8.1.4.v20120524)
<
* Connection #0 to host api.eum-appdynamics.com left intact
* Closing connection #0
```

Check Uploaded dSYMs Using the REST API

You can check to make sure that your dSYMs have successfully uploaded using two REST APIs.

1. Get a list of the UUIDs for the last 50 dSYMs you have uploaded
2. Check if a specific dSYM has been uploaded

List of the Last 50 dSYMs Uploaded

The `dsymQuery` method allows you to retrieve a list of UUIDs for up to the last 50 dSYM files that have been uploaded to your account, along with the time they were uploaded. The response is displayed as JSON, by upload time, with the most recent first.

1. Set up your authentication as described in Upload the dSYM File to AppDynamics Using the REST API.
2. Create a GET request of the form:

   ```
   curl --user Example+account:Example-License-Key-4e8ec2ae6cfe https://<EUM_Cloud/Server_Host:Port>/v2
   /account/Example+account/crash-symbol-file-query/dsym
   ```

   where the value for `--user` is the authentication string you created in Upload the dSYM File to AppDynamics Using the REST API, `EUM_Cloud/Server` is `api.eum-appdynamics.com:443` for SaaS-based EUM Cloud accounts or as configured for on-premises EUM Servers.

Sample Request

```
curl --user Example+account:Example-License-Key-4e8ec2ae6cfe  https://api.eum-appdynamics.com:443/v2/account
/Example+account/crash-symbol-file-query/dsym
```

Sample Response

```
{"dSymFiles":[
{"uploadTime":"mm/dd/yyyy 14:15:32","UUID2":"my_uuid2"},
{"uploadTime":"mm/dd/yyyy 14:15:32","UUID":"my_uuid"}
]}
```

Check for Specific dSYM by UUID

The `checkForDSymFile` method allows you to check if a specific dSYM by UUID has been uploaded. The upload time is returned in the response.

1. Set up your authentication as described in Upload the dSYM File to AppDynamics Using the REST API.
2. Create a GET request of the form:

   ```
   curl --user Example+account:Example-License-Key-4e8ec2ae6cfe https://<EUM_Cloud/Server_Host:Port>/v2
   /account/Example+account/crash-symbol-file-query/dsym/uuid/<UUID_to_Check>
   ```

where the value for `--user` is the authentication string you created in Upload the dSYM File to AppDynamics Using the REST API, `EUM_Cloud/Server` is `api.eum-appdynamics.com:443` for SaaS-based EUM Cloud accounts or as configured for on-prem EUM Servers, and `UUID_to_Check` is the UUID of the dSYM in which you are interested.

Sample Request

```
curl --user Example+account:Example-License-Key-4e8ec2ae6cfe  https://<EUM_Cloud/Server_Host:Port>/v2
/account/Example+account/crash-symbol-file-query/dsym/uuid/<UUID_to_Check>
```

Sample Response

```
{"uploadTime":"mm/dd/yyyy 14:15:32","UUID":"my_uuid"}
```

Manage the dSYM Files with Bitcode Enabled

Related pages:

- Upload the dSYM File

With iOS 9, Apple introduced a new mode of building and distributing iOS apps called App Thinning. As part of this initiative, apps containing bitcode are delivered.

If you choose to create your app containing bitcode, the process of acquiring your dSYM files for upload while you are in local development, debugging, and testing mode (Xcode on simulators or Devices) is exactly the same as in earlier versions.

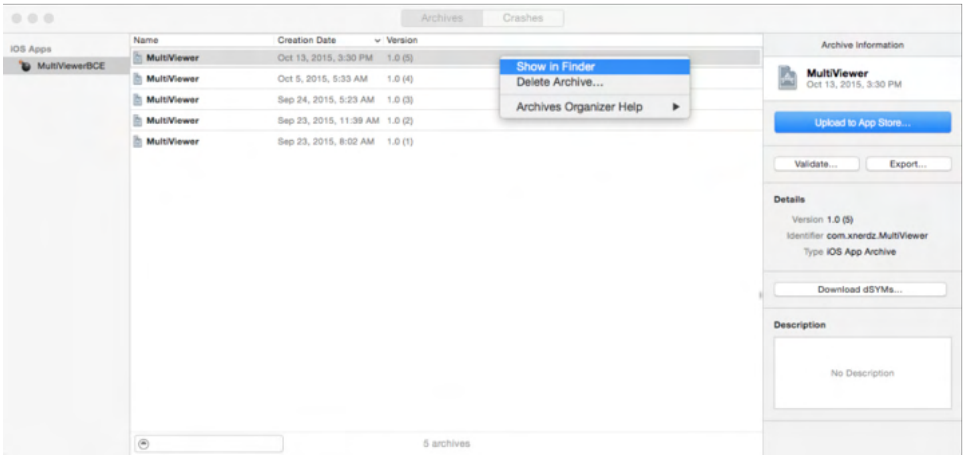
It also works if you are using Ad Hoc Provisioning. But once you have used iTunes Connect to upload your application to Apple, for either TestFlight beta testing or application release, you have to take a few additional steps to get your dSYM files for use with AppDynamics.

Get and Upload the Correct dSYM File

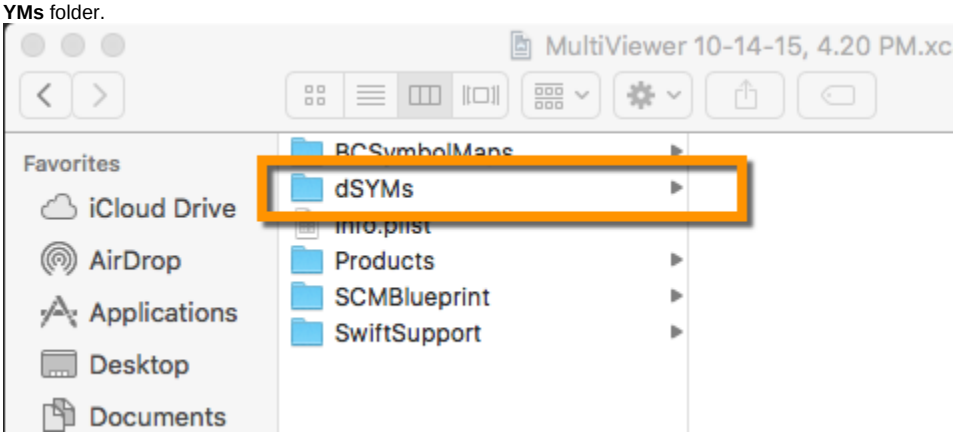
As part of the process of preparing to upload your application using iTunes Connect, you create an archive. After the app is uploaded to iTunes Connect, it is processed by Apple. To have the correct version of the dSYM for AppDynamics, you must download this processed version to your local archive.

To download the processed dSYM file, in the Xcode **Archives Organizer**:

1. Make sure this version of the application is available for download (either TestFlight or the iTunes store).
2. In the **Archives** organizer, select the archive for this version.
3. Click the **Download dSYMs** button.



4. Right-click the `.xcarchive` file you receive and select **Show in Finder**. Right-click again and select **Show Package Contents**, and select the **dSYMs** folder.

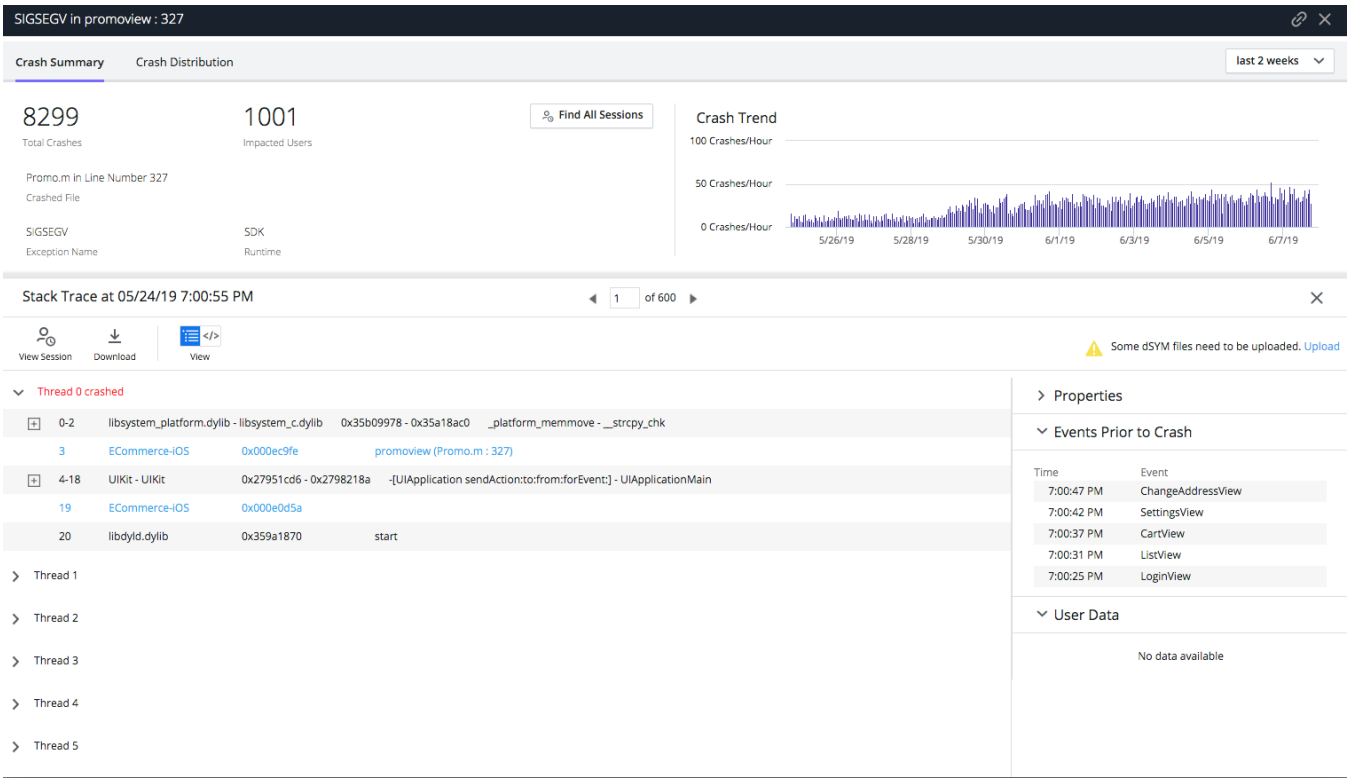


5. Zip this directory and upload.

See the Apple Developer docs.

Upload Missing dSYM Files

If you have crashes that AppDynamics could not symbolicate because it did not have a correct dSYM file, you get an error message in the **Crashes Details** dialog.



If you click **Upload missing dSYM files in Instrumentation**, a list of dSYM hashes appears. Match the missing files with the dSYMs in your archives, as shown above. If necessary, click **Download dSYMs** to update the archive. Upload the necessary files to AppDynamics.

> ⚠ Some of the missing dSYM files that AppDynamics reports may belong to simulation/debug builds that have crashed. In this case, the dSYMs may no longer be available.

Troubleshoot the iOS Instrumentation

Related pages:

- Install the iOS SDK
- Instrument an iOS Application
- Customize Your Instrumentation With the iOS SDK

This page provides instructions and tips for solving some common iOS instrumentation issues.

Agent Not Detecting Requests

First, confirm that you are using either the `NSURLConnection` or the `USURLSession` class. By default, you are required to make network requests. If you are using a custom HTTP library, make sure you follow the instructions given in Use the Agent with a Custom HTTP Library.

If you are using `NSURLSession`, be sure to create an instance of `NSURLSession` *after* initializing the iOS agent. This is because the agent may not be aware of objects created before the instrumentation is initialized.

Unrecognized Selector Error

If you are instrumenting your app using the manual method and you see this error message:

```
+[NSURLConnection ADEumInsertHooks]: unrecognized selector sent to class <hex value>
```

Then, you did not add `-ObjC` to your link flags. See Set the -ObjC Flag for instructions.

Instrument Android Applications

Before you can monitor your Android application, you need to build and instrument your application to enable the Android Agent to collect mobile metrics. The Android Agent supports monitoring for Android applications written in Java or any JVM-based language such as Kotlin.

After you have set up and accessed Mobile RUM, use one of the following ways to build and instrument your application:

- Instrument an Application with the Android Agent Installer - The Android Agent Installer is an Android Studio plugin that will configure the build and insert the instrumentation code for you.
- Instrument an Android Application Manually - If you are unable to use the Android Studio plugin, follow these instructions to configure your build and instrument your application.

For further customization, see Customize the Android Build and Customize the Android Instrumentation.

Instrument an Application with the Android Agent Installer

This page describes how to use the AppDynamics Android Agent Installer plugin to instrument an app with the AppDynamics Android SDK. The instructions will include how to get the EUM App Key, install the plugin, and use the plugin to instrument the application.

Follow the steps below to get your EUM App Key and instrument your Android applications:

> The plugin is recommended for initial basic instrumentation. If your app requires custom initialization or more advanced configuration, see Customize the Android Build and Customize the Android Instrumentation.

Check the Prerequisites

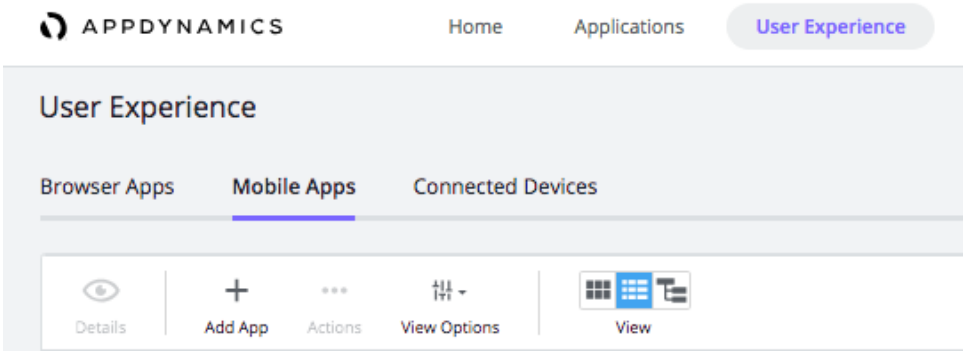
To use the plugin, you need the following:

- An AppDynamics account - If you do not have an account, sign up for a free account.
- Android Studio 2.1.0 - 2.9.9, and 3.2.1 - 3.9.9 (Not supported: 3.0.0 - 3.2.0, and 4.0)
- Java >= 7

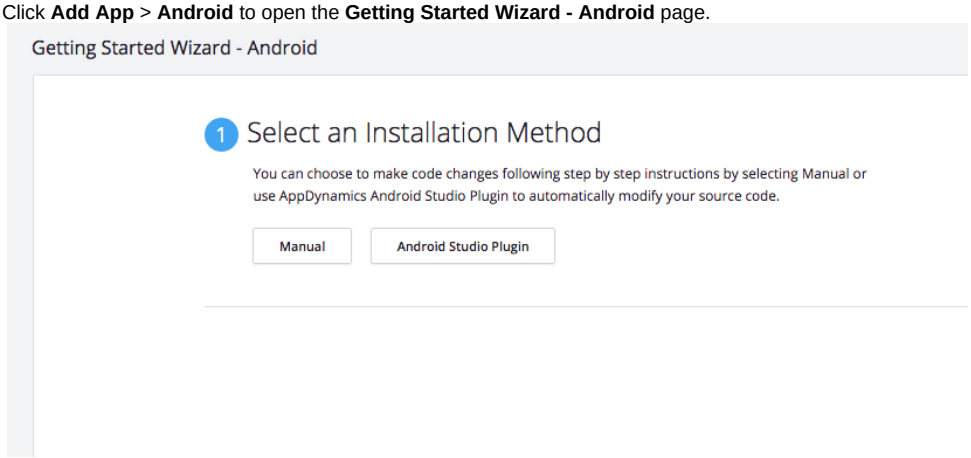
If you are running an on-premises deployment, you also need the URL to your EUM Server.

Get the EUM App Key

1. From the top navigation bar of your Controller UI, go to **User Experience** > **Mobile Apps**.

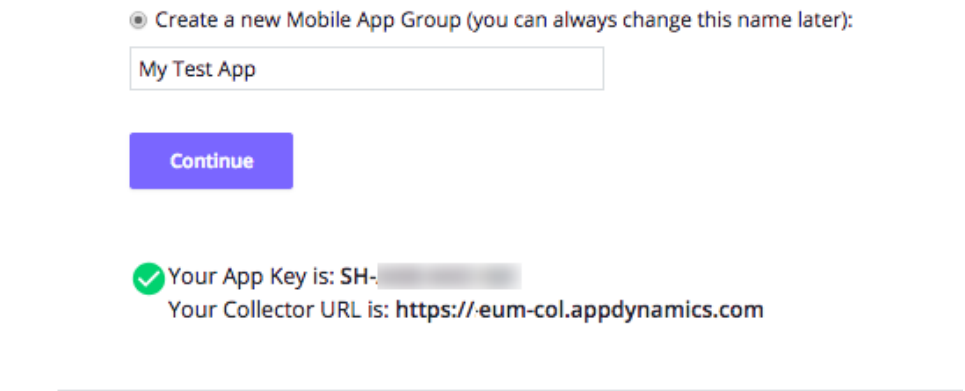


2. Click **Add App** > **Android** to open the **Getting Started Wizard - Android** page.

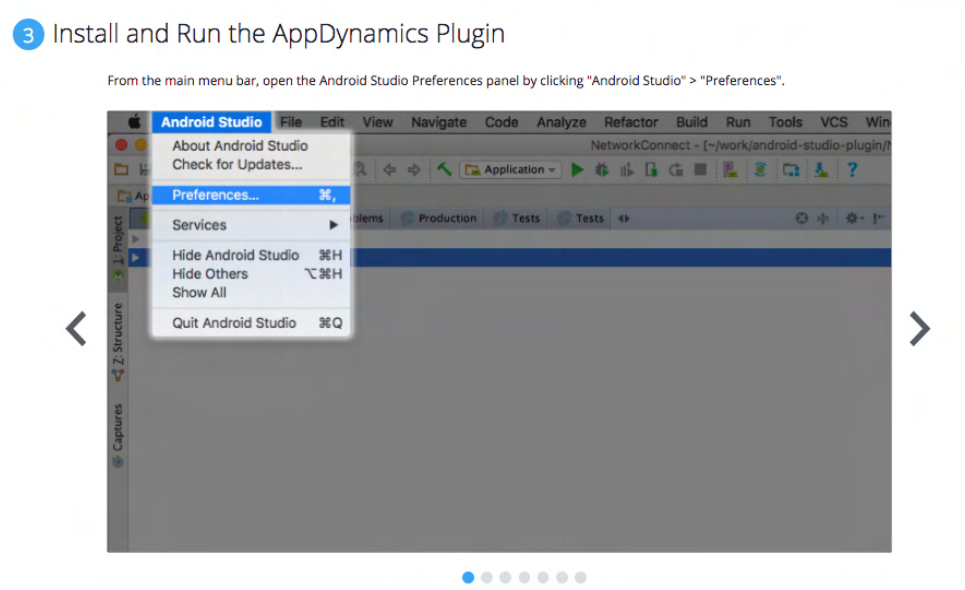


3. Click **Android Studio Plugin**.
4. Select the **Create a new Mobile App Group** radio button, enter an application name, and click **Continue**.

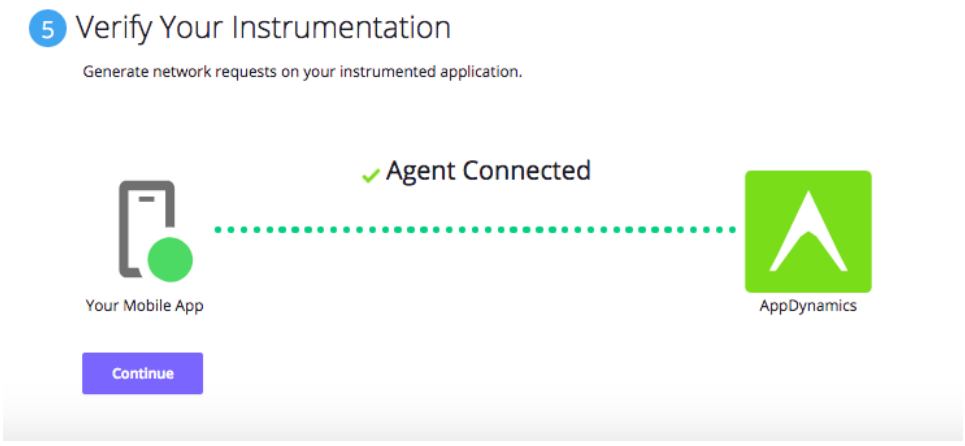
5. Copy the displayed App Key and the Collector URL.



6. Follow the instructions in 3 Install and Run the AppDynamics Plugin.



7. Keep the wizard open until the instrumentation has been verified in the wizard.



Instrument an Android Application Manually

Related pages:

- Verify the Android Instrumentation
- Confirm the Mobile Agent Connected to the Controller
- Customize the Android Build
- Customize the Android Instrumentation

To instrument Android Applications, you need to first build the application for the platform you are using and then add the instrumentation code.

To get started, follow the instructions below:

1. Build the Android Application
2. Instrument the Android Application

Build the Android Application

To build your application, follow the instructions for your platform:

- Gradle/Android Studio
- Maven
- Ant

If you are fetching the AppDynamics Android SDK from the AppDynamics Download page, see Download Manually.

Gradle/Android Studio

Complete the following steps to configure the build for your Android application:

1. Confirm the compatibility of your Gradle, Android Tools with the AppDynamics plugin versions.
2. Install the Android Agent.
3. Activate the plugin.

Install the Android Agent

Use the native package system to install the Android Agent. In the app module `build.gradle`, add the class path of the AppDynamics Gradle Plugin to the build path dependencies clause. Use `com.appdynamics:appdynamics-gradle-plugin:21.4.0` unless you need to use another version of the AppDynamics plugin for compatibility between your Gradle and Android Tools.

```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:1.1.0'
        classpath 'com.android.tools.build:gradle:3.4.1' // >=3.4.1
        classpath 'com.appdynamics:appdynamics-gradle-plugin:21.4.0' // this line added for AppDynamics
    }
}
allprojects {
    repositories {
        jcenter()
    }
}
```

If you are unable to use the Android Gradle plugin >=3.4.1, you will have to use the Android Agent <=20.4.0.

Activate the Plugin

In your module-level `build.gradle`, add the `adeum` plugin immediately after the `com.android.application` plugin, so that it looks similar to the example below:

```
apply plugin: 'com.android.application'
apply plugin: 'adeum' // this line added for AppDynamics
```

Apache Maven Project

If your application is a Maven Project:

1. Add the following code to the `<dependencies>` section:

```
<dependency>
   <groupId>com.appdynamics</groupId>
   <artifactId>appdynamics-runtime</artifactId>
   <version>1.0</version>
</dependency>
```

2. Add the following code to the `<plugins>` section:

```
<plugin>
    <groupId>com.appdynamics</groupId>
    <artifactId>appdynamics-maven-plugin</artifactId>
    <version>1.0</version>
    <executions>
        <execution>
            <phase>compile</phase>
            <goals>
                <goal>adinject</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

Ant/Eclipse

See the Knowledge Base article Use Ant to Build Android Apps with the AppDynamics Android SDK for instructions.

Instrument the Android Application

After you have completed building your application, follow the steps below:

1. Get the Application Key
2. Add the Required Permissions
3. Modify the Source
4. Run the Build

Get Application Key

After you completed the **Getting Started Wizard**, you were given an EUM App Key. You will need this key when you modify the source code. In some cases, multiple mobile applications can share the same key.

If you have completed the **Getting Started Wizard**, but don't have your EUM App Key, see Get Your Application Key.

Add the Required Permissions

Open your application's `AndroidManifest.xml` file and verify that it has these permissions:

```
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"></uses-permission>
```

If both of these permissions are not present, add them.

Modify the Source

1. In the source file that defines your application's primary Activity, add the following import:

```
import com.appdynamics.eumagent.runtime.Instrumentation;
```

2. In your primary Activity's `onCreate()` method, add the following lines, passing in the EUM App Key from step 2 above:

```
Instrumentation.start(<EUM_APP_KEY>, getApplicationContext());
```

3. Save the file.
   Your code should look something like this.

```
import com.appdynamics.eumagent.runtime.Instrumentation;
...
@Override public void onCreate(Bundle savedInstanceState) {
  Instrumentation.start(<EUM_APP_KEY>, getApplicationContext());
  ...
}
```

4. Configure the Android Agent to report metrics and screenshots to the SaaS EUM Server and Screenshot Service in your region when initializing the agent with the methods `withCollectorURL` and `withScreenshotURL`. (If you are using an on-premises EUM Server, see Customize the Agent Configuration for implementation details.)

Verify the Instrumentation

See Verify the Android Instrumentation for build and verification instructions.

Upgrade the Android Mobile Agent

As new features are added to the agent, you will need to upgrade the Android SDK in your app.

To upgrade, you simply update the build file for your platform:

- Gradle/Android Studio
- Maven

The process of installing and updating the latest Android SDK is the same.

Verify the Android Instrumentation

To verify your Android instrumentation:

1. Run the build.
2. Verify the instrumentation.
3. Confirm the agent connected to the Controller.

Run the Build

Build your Android app following the instructions for your build system:

- Android Studio
- Gradle
- Maven

In the console, you should see something like this:

```
[injector]                                /=======================================\
[injector]                                | AppDynamics BCI Instrumentation summary  |
[injector]                                \=======================================/
[injector]
[injector]
[injector]                 - Total number of classes visited (#720 classes)
[injector]                 - Total number of classes instrumented (#1 classes)
[injector]                 - Total number of classes failed to instrument (#2 classes)
[injector]                 - Total number of features discovered (#3)
[injector]
```

Verify the Instrumentation

Based on the build system you used, verify that the instrumentation was successful:

- Gradle/Android Agent Installer Plugin
- Maven
- Ant

Gradle/Android Agent Installer Plugin

If you didn't use the `-i` flag, check to make sure there is a line in your console output that contains "`inject`". If you don't see this information printed in your console, either your project is incorrectly configured or the injector failed to run completely. There is a very detailed log of this process either at `<project>/target/appdynamics_eum_android_bci.log` or `<module>/target/appdynamics_eum_android_bci.log`.

Maven

If you don't see this information printed in your console, either your project is incorrectly configured or the injector failed to run completely. There is a very detailed log of this process either at `<project>/target/appdynamics_eum_android_bci.log` or `<module>/target/appdynamics_eum_android_bci.log`.

Ant

If you don't see this information printed in your console, either your project is incorrectly configured or the injector failed to run completely. There is a very detailed log of this process either at `<project>/target/appdynamics_eum_android_bci.log` or `<module>/target/appdynamics_eum_android_bci.log`.

Customize the Android Build

Related pages:

- Instrument an Android Application Manually
- Customize the Android Instrumentation

Build the Android Application showed you how to use the bare minimum configuration to build your application.

The following pages show you how to add additional configuration to customize your build:

- Automatically Upload Mapping Files
- Configure ProGuard to Prevent Obfuscation and Class Removal
- Exclude Classes from Being Instrumented
- Enable/Disable Instrumentation for Build Types
- Enable/Disable Native Crash Reporting

Automatically Upload Mapping Files

You can configure the build to add your EUM account information so that you can automatically upload your ProGuard or DexGuard mapping files for crash reports with each build.

This is the recommended way of managing these types of mapping files, although manual modes are available for uploading mapping files. If you do not use ProGuard or DexGuard to obfuscate your files you can ignore this section. See Configure ProGuard to Prevent Obfuscation and Class Removal.

> **DexGuard Limitation**
>
> DexGuard mapping files must be in the default location. This means that their dexguard configurations should not specify `--printMapping`, which changes where to print the mapping file.

Upload to the SaaS EUM Server

If you are using the SaaS EUM Server, you just need to provide your account name and license key to automate uploading as shown below.

```
adeum {
        url COLLECTOR_URL // Only needed for on-prem. This is used to upload the proguard mapping file by the
buildscript. For on-prem it should be the public collector url.
        account {
                licenseKey EUM_LICENSE_KEY name EUM_ACCOUNT_NAME
                }
        proguardMappingFileUpload {
                failBuildOnUploadFailure false enabled true // This specifies whether the entire build should
fail if mapping file upload fails. Enabled refers to whether auto-uploading of mapping file is enabled.
                }
        excludeClasses = ['android.support.multidex.*', 'okio.**'] // This specifies any classes customers
would like to exclude from auto-instrumentation. The notation is standard java package/class specification used
in Gradle.
        enabledForDebugBuilds = true // Whether auto-instrumentation is enabled for debug builds, true by
default.
        enabledForReleaseBuilds = true // Whether auto-instrumentation is enabled for release builds, true by
default.
        webViewCallbackCrashReportingEnabled = false // Auto-instrumentation is enabled for WebChromeClient and
WebViewClient
...


// This block specifies how to handle native crashes. By default, this is not enabled.
        nativeCrashHandling {
                if (BUILD_NDKLIB) {
                        enabled = true
                        symbolUpload {
                                variantLibraryPaths = ["appRelease": "ndkLib/obj/local"]
                        }
                }
        }
}
```

Upload to the On-Premises EUM Server

If you are using an on-premises deployment, in addition to supplying your account name and license, you must also assign the URL to your on-premises EUM Server to the `url` property as shown:

```
adeum{
    // The account information is also needed for on-prem deployments.
    account {
      name "The EUM Account Name from the License screen"
      licenseKey "The EUM License Key from the License screen"
    }
    ...
    // Add this information to point to the on-prem EUM Server.
    url "https://<your-on-prem-eum-server>:7001"
}
```

Modify the Default Upload Behavior

You can also modify the default upload behavior for both on-premises and SaaS deployments. For on-premises deployments, you will need to provide the URL to your on-premises EUM Server with the `url` property.

The configuration example below sets `failBuildOnUploadFailure` to `true`, so that the build will fail if the upload to the URL specified by `url` is unrea chable. The object `proguardMappingFileUpload` is used for both Proguard and DexGuard mapping files. If you don't want your build to fail because the ProGuard or DexGuard mapping file couldn't be uploaded, do *not* modify the default setting.

```
adeum{
    ...
    // Add this information to point to the on-prem EUM Server.
    url "https://<your-on-prem-eum-server>:7001"

   // Add this information if you want to modify upload behavior.
    proguardMappingFileUpload {
      failBuildOnUploadFailure true // If true, will fail build. Defaults to false.
      enabled true //enables automatic uploads.  Defaults to true
    }
}
```

Configure Proguard to Prevent Obfuscation and Class Removal

If you use ProGuard to verify or optimize your code, add the following lines to the ProGuard configuration file `proguard-rules.pro` under Gradle Scripts to prevent ProGuard from obfuscating or removing classes needed for proper instrumentation.

```
-keep class com.appdynamics.eumagent.runtime.DontObfuscate
-keep @com.appdynamics.eumagent.runtime.DontObfuscate class * { *; }
```

Configure ProGuard to Prevent Obfuscation and Class Removal

Related pages:

- Automatically Upload Mapping Files

If you use ProGuard to verify or optimize your code, add the following lines to the ProGuard configuration file `proguard-rules.pro` under Gradle Scripts to prevent ProGuard from obfuscating or removing classes needed for proper instrumentation.

```
-keep class com.appdynamics.eumagent.runtime.DontObfuscate
-keep @com.appdynamics.eumagent.runtime.DontObfuscate class * { *; }
```

Exclude Classes from Being Instrumented

You can exclude one or more class names from being instrumented, using the "adeum" `excludeClasses` snippet.

Class names can contain wildcards:

- '?' for a single character
- '*' for any number of characters, but *not* the package separator
- '**' for any number of (any) characters
- '$' prefix for an inner class name

To exclude named classes from instrumentation, add the class names to the `excludeClasses` array, using matching patterns where needed:

```
...
adeum {
    // Excludes all the classes in the android.support.multidex and okio packages.
    excludeClasses = ['android.support.multidex.*', 'okio.**']
}
...
```

Enable and Disable Instrumentation for Build Types

This page describes how to enable and disable instrumentation for Android build types.

Default Instrumentation for Build Types

For `release` builds, auto-instrumentation is enabled by default.

```
adeum {
...
enabledForDebugBuilds = true // Whether auto-instrumentation is enabled for debug builds, true by default.
        enabledForReleaseBuilds = true // Whether auto-instrumentation is enabled for release builds, true by
default.
        webViewCallbackCrashReportingEnabled = false // Auto-instrumentation is enabled for WebChromeClient and
WebViewClient
...
}
```

Verify the Instrumentation State at Runtime

If one or more of your builds have disabled instrumentation, you need to disable the instrumentation check at runtime. You can set boolean fields in the build configuration to disable the runtime verification of instrumentation. The runtime verification of instrumentation is on by default. You can turn it off using the method `withAutoInstrument`.

In the build configuration, you can set a boolean value for the `CHECK_ENABLED` field for build types. The table below shows the config value, the instrumentation state, and then describes the runtime behavior of your applications.

Config Value	Instrumentation State	Runtime Behavior
`true`	Enabled	The agent will verify that instrumentation has been enabled before initializing.
	Disabled	The agent will verify that the instrumentation has not been enabled and then thrown an `IllegalState` exception.
`false`	Enabled	The agent will initialize without checking whether instrumentation has been enabled.
	Disabled	

For example, in the build configuration below, the field `CHECK_ENABLED` is set to `true` for the release build and `false` for the debug build.

The agent for the `debug` build will not verify if instrumentation has been enabled before executing instrumentation code, whereas, the application code for the `release` build will check before executing the instrumentation code.

```
...
android {
    // usual stuff
    buildTypes {
    // usual stuff
        release {//these lines added for AppDynamics
            //release based configuration
            // The release build by default is not "debuggable".
            // The build config "CHECK_ENABLED" will be accessible in the runtime environment.
            // This enables the Android Agent to verify that the instrumentation has been enabled before
running the initialization code.
            // If instrumentation has not been enabled, an "IllegalState" exception is thrown.
            buildConfigField "boolean", "CHECK_ENABLED", "true"
        }
        debug {
            // Setting 'CHECKED_ENABLED' to "false" means the Android Agent will run the initialization code
without confirming that
            // instrumentation has been enabled. No exception will be thrown.
            buildConfigField "boolean", "CHECK_ENABLED", "false"
        }
    }
}
...
```

If the value of `CHECK_ENABLED` is `true`, the Android Agent will confirm that instrumentation has been enabled before executing the initialization code. If the value is `false`, the Android Agent executes the initialization code regardless of whether instrumentation has been enabled.

The following instrumentation initialization code shows you how to check the value of the build config `CHECK_ENABLED` with the method `withCompileTimeInstrumentationCheck`.

```
import com.appdynamics.eumagent.runtime.Instrumentation;
...
Instrumentation.start(
        AgentConfiguration config = AgentConfiguration.builder()
            .withAppKey(<EUM_APP_KEY>)
            .withContext(this)
            .withCompileTimeInstrumentationCheck(BuildConfig.CHECK_ENABLED)
            .build();
);
...
```

Enable and Disable Native Crash Reporting

You can report native crashes caused by native libraries. This feature is available for all architectures supporting the Android Native Development Kit (NDK) .

Native Crash Handler

When instrumentation is initialized, the native crash handler loads the AppDynamics library `libadeum`. At entry points of your code, a signal handler is injected into the running process. After a signal is handled, `libadeum` restores the previous signal handler. (This enables you to use different methods to handle signals.)

The handler collects and writes the information about the native crash into a temporary file. When the application is restarted, the temporary file is read, processed, and the crash information is displayed in the Controller UI.

Configuration for Native Crash Reporting

By default, however, this feature is turned off, so you need to add the section `nativeCrashHandling` and set `enabled` to true. In addition, you point the library paths in the `variantLibraryPaths` array for each build variant to write the symbol files locally and transmit the symbol files to the EUM Server. You can also specify a build ID for a project build. After you build the project, the build ID can be found in `build/appDynamics /ndkSym<VariantName>/<AppBuildID>`. If not specified, the build ID of the most recently built project will be used.

```
adeum {
    // Other configuration if needed.

    // Enable native crash handling; the default is false.
    nativeCrashHandling {
        enabled = true
        symbolUpload {
            buildId = "<your_custom_build_id>"
            variantLibraryPaths = ["release": "ndkLib/obj/local", "debug": "ndkLib/obj/debug", ...]
        }
    }
}
```

Configuration for the Android App Bundle

If you using Android App Bundle, you need to add the following line to your app module `build.gradle` for crashes to symbolicate correctly.

```
android.bundle.enableUncompressedNativeLibs=false
```

View Source Code Information in the Stack Trace

You have two different ways to view symbol information in the stack trace:

1. From the **Crash Details** dialog, download the crash report and use a utility like `ndk-stack` to view the crash report with source code level symbolications.
2. Upload the symbol files and then view the stack trace in the **Crash Details** dialog.

Download the Crash Report and Use ndk-stack

1. From the **Crash Dashboard**, double-click one of the crashes listed in the **Unique Crashes** widget.
2. From the **Crash Details** dialog, click **Download** to download the crash report.
3. Run ndk-stack (or a similar utility) on the downloaded crash report as the input file to generate a crash report with source code level symbolications.

Upload the Symbol File and View the Stack Trace

You are recommended to automatically generate and upload symbol files, but you can also manually generate and upload the symbol files, too. After you upload the symbol files, when the app with the same build ID as the UUID of the uploaded `ndkSYM` file crashes, you will see the file name and the line number next to the C/C++ function name for each frame of the stack trace shown in the **Crash Details** dialog.

Automatically Generate and Upload Symbol Files

You can run the following `gradle` command to generate and upload the symbol files to the EUM Server. Replace `<VariantName>` with the build variant names you defined in the configuration. For example, in the configuration above, the `<VariantName>` could be release or debug.

```
$ gradle appDynamicsUploadNDKSymbolFile<VariantName>
```

Manually Generate and Upload Symbol Files

To generate symbol files, run the following gradle command, where `<VariantName>` is the build variant names. The generated symbol file will be written to `build/appDynamics/ndkSym<VariantName>/<AppBuildID>/<AppBuildID>.ndkSYM.zip`.

```
$ gradle appDynamicsGenerateNDKSymbolFile<VariantName>
```

To manually upload the generated symbol file, use the following cURL command, replacing the `<ndkSymZipFile>` with your generated symbol file, `<Account Name>` with your EUM Account name, and `<License Key>` with your EUM License Key.

```
$ curl -v -H Content-Type:application/octet-stream --upload-file <ndkSymZipFile> --user <Account Name>:<License Key> https://api.eum-appdynamics.com/v2/account/<Account Name>/android-ndksym
```

Customize the Android Instrumentation

Related pages:

- Instrument an Android Application
- Android SDK Documentation

Once you have instrumented your Android application with the Android SDK, you can also use the APIs exposed by the SDK to customize the data for your application that appears in the Controller UI. The following sections show you how to use the Android API to customize your instrumentation:

Because the Android Agent stores data about events in a local buffer before reporting the information, we recommend you use the APIs with discretion.

Collect Additional Types of Data

The `Instrumentation` class has additional methods to allow you to extend the kinds of application data you can collect and aggregate using Mobile RUM. There are six basic kinds of extensions that you can create:

Extension	Description	Specification	Where Data is Displayed in the Controller UI
Info points	How often a method is invoked, and how long it takes to run	• Data is numeric • Names must consist of alphanumeric characters and/or spaces	• Metric Browser • Custom Data • Network Request Snapshots • Mobile Sessions • Network Request Analyze
Custom timers	Any arbitrary sequence of events within your code timed, even spanning multiple methods	• Data is numeric • Metric names must consist of alphanumeric characters and/or spaces	• Metric Browser • Custom Data
Custom metrics	Any integer-based data you wish to collect	• Data is numeric • Metric names must consist of alphanumeric characters and/or spaces	• Metric Browser • Custom Data
User data	Any string key/value pair you might find useful	• Any data type • Metric names have no restrictions	• Network Request Snapshots • Mobile Sessions • Network Request Analyze • Crash Snapshots
Breadcrumbs	The context of a crash	• Any data type • Metric names have no restrictions	• Network Request Snapshots • Mobile Sessions • Network Request Analyze • Crash Snapshots

User interactions	Captured when users press buttons, click on lists, and select text	• Any data type • Metric names have no restrictions	• Network Request Snapshots • Mobile Sessions • Network Request Analyze

When you have set up info points, custom timers, custom metrics, and/or user data, the Mobile Agent packages that data in a mobile beacon. Normally, the beacon is transmitted when the instrumented application sends an HTTP request or when the application starts after a crash, but if custom data has been collected and neither of those events has occurred for at least 5 minutes, the custom data is sent at that time.

Info Points

Information points allow you to track how your own code is running. You can see how often a method is invoked, how long it takes to run, and if an exception is thrown. The simplest way to set up an information point is to use the `@InfoPoint` annotation. For example:

```
@InfoPoint
 public void infoPointMethod(String arg1, int arg2, long value) {
   System.out.println("Executing infoPointMethod!");
 }
```

You can also do this manually, using the `CallTracker` interface. For example, to collect information on the `downloadImage` method, you could use code similar to this:

```
private void downloadImage(URL url) {
     CallTracker tracker =
          Instrumentation.beginCall("com.example.android.awesomeapp.ImageDownloader", "downloadImage")
              .withArguments(url);
     try {
          //download image.
          tracker.reportCallEnded()
     } catch(Exception e) {
         //handle exception thrown
         tracker.reportCallEndedWithException(e);
     }
 }
```

This information appears in the **Custom Data** view of the Controller UI.

Custom Timers

Custom timers allow you to time any arbitrary sequence of events within your code, even spanning multiple methods, by using `startTimer` and `stopTimer`.

```
public class MyActivity extends Activity {
   @Override
   protected void onStart(){
       Instrumentation.startTimer("Time Spent on MyActivity");
       //your code here.
   }

   @Override
   protected void onStop(){
       Instrumentation.stopTimer("Time Spent on MyActivity");
       //your code here.
   }
 }
```

The methods `startTimer(String)` and `stopTime(String)` can be called from different threads. Calling `startTimer` again with the same name value resets a named timer.

This information appears in the **Custom Data** view of the Controller UI.

Custom Metrics

Any integer-based data can be passed to the Android Agent. The first parameter to the `reportMetric` call is the name you want the metric to appear under in the Controller UI. The metric name should only contain alphanumeric characters and spaces. Illegal characters are replaced by their ASCII hex value.

For example, to track the number of times your users click the checkout button in your UI, you could use code similar to this.

```
findViewById(R.id.checkout_button).setOnClickListener(new View.OnClickListener(){
      @Override
      public void onClick(View view){
          //run your checkout routine.
          Instrumentation.reportMetric("Checkout Count", 1);
      }
 });
```

This information appears in the **Custom Data** view of the Controller UI.

User Data

You can set any string key/value pair you think might be useful. The first parameter to the `setUserData` call is the key you want to use, which must be unique across your application. The second is the value you want assign to the key.

For example:

```
void onUserLoggedIn(String userid) {
    Instrumentation.setUserData("User ID", userid);
    ...
}
```

This information is available in **Network Request Analyze** and is added to any crash snapshot. Keys and values are limited to 2048 characters each.

You can also set user data with values of other types (Long, Boolean, Double, Date) using the following methods:

- `setUserDataLong`
- `setUserDataBoolean`
- `setUserDataDouble`
- `setUserDataDate`

Breadcrumbs

Breadcrumbs allow you to situate a crash in the context of your user's experience. Set a breadcrumb when you want to capture a series of events. If your application crashes at some point in the future, the breadcrumb will be displayed along with the crash report.

There are two ways of leaving breadcrumbs:

- Crash Reports Only
- Modal

Crash Reports Only

Using this method means that breadcrumbs are reported in crash reports only.

```
public static void leaveBreadcrumb(java.lang.String breadcrumb)
```

Modal

Using this method lets you fine tune where the breadcrumbs are reported, either only in crash reports or in crash reports *and* sessions.

```
public static void leaveBreadcrumb(java.lang.String breadcrumb, int mode)
```

Where `mode` is either:

- `CRASHES_ONLY`
- `CRASHES_AND_SESSIONS`

If the `breadcrumb` is over 2048 characters, it is truncated. If it is empty, no breadcrumb is recorded. Each crash report displays the most recent 99 breadcrumbs.

Add a Crash Reporting Callback

You may want to make crash report information (that the Android Agent collects) available to other parts of your code, for example, to Google Analytics. To enable passing on summary crash information, you can set up a crash report runtime callback. To get a callback when the Android Agent detects and then reports a crash, implement the following interface in your code:

```
public interface CrashReportCallback {
    void onCrashesReported(Collection<CrashReportSummary> summaries);
}
```

The Android Agent sends a collection of crash report summaries instead of an individual callback in the event that there is more than one crash, producing multiple crash report summaries.

The method `onCrashesReported` is invoked during the next initialization of the Android Agent after a crash has occurred.

This callback is invoked on your app's UI thread, so any work should be done on a separate work thread.

Each `CrashReportSummary` has the following properties:

```
public class CrashReportSummary {
    public final String crashId;
    public final String exceptionClass;
    public final String exceptionMessage;
}
```

If you are sending the information to another analytics tool, such as Google Analytics, we recommend to include all three properties: `exceptionClass` an d `exceptionMessage` are useful for quick identification of the crash, and `crashId` is useful to look up the crash in the Controller UI.

For example, to print the crash information to Android's logger, you could implement a `CrashReportCallback` class like this:

```
public static class MyCrashReportCallback implements CrashReportCallback {
    @Override
    public void onCrashesReported(Collection<CrashReportSummary> summaries) {
        for (CrashReportSummary crash : summaries) {
            Log.e("MyApp", "Crash Detected: " + crash.exceptionClass + " : " + crash.exceptionMessage + " (" +
crash.crashId + ")");
        }
    }
}
```

You set your callback as using the AgentConfiguration object:

```
final AgentConfiguration config = AgentConfiguration.builder()
        .withAppKey(appKey)
        .withContext(context)
        .withCrashCallback(new MyCrashReportCallback())
        .build();
```

Your callback is invoked after a crash, during the next initialization, on the main thread. For more information, see the latest JavaDocs for the complete Android SDK API.

Disable Crash Reporting

Crash reporting is enabled by default, but you can manually disable crash reporting through the instrumentation configuration. If you are using other crash reporting tools, you might disable crash reporting to minimize conflicts and optimize the crash report results.

You can disable crash reporting by configuring the instrumentation with the `crashReportingEnabled` property as shown in the following:

```
let config = ADEumAgentConfiguration(appKey: <#EUM_APP_KEY#>);
config.crashReportingEnabled = false;
ADEumInstrumentation.initWith(config);
```

Report Errors and Exceptions

You can report exceptions using the method `reportError` from the `Instrumentation` class. Reported exceptions will appear in session details.

You can also set one of the severity levels below for an issue. With the severity level, you can filter errors in the **Code Issues Dashboard** or **Code Issues Analyze**.

- `ErrorSeverityLevel.INFO`
- `ErrorSeverityLevel.WARNING`
- `ErrorSeverityLevel.CRITICAL`

The example below uses the API to report possible exceptions and sets the severity level to `ErrorSeverityLevel.CRITICAL` (critical) when writing to a file.

```
private void writeToFile(String filePath, String data) {
    try {
        OutputStream outputStream = new FileOutputStream(filePath);
        Writer outputStreamWriter = new OutputStreamWriter(outputStream);
        outputStreamWriter.write(data);
        outputStreamWriter.close();
    } catch (IOException e) {
        Log.e("Exception", "File write failed: " + e.toString());
        Instrumentation.reportError(e, ErrorSeverityLevel.CRITICAL);
    }
}
```

Disable the Agent to Stop Sending User Data to the Collector

You can disable the agent to stop sending all data to the collector while the agent is initialized and running. For example, you can disable the agent if your app has an option for users to opt-out of monitoring for privacy reasons.

`shutdownAgent`

The `shutdownAgent` call stops outgoing data to the collector, and does not persist data on the device.

```
Instrumentation.shutdownAgent();
```

- The call only stops the traffic out of the agent.
- Once the agent has been initialized, the call cannot be removed, and a license will have been consumed.
- If you want to make this state permanent for a device, add code in UserDefaults to save the state and use that flag to conditionally initialize the agent in your code.

`restartAgent`

To re-enable the agent and reverse `shutdownAgent`, use `restartAgent`.

```
Instrumentation.restartAgent();
```

- This call will respect the server side calls that can remotely shutdown the agent in a similar way.
- The call is only in effect while the app is running.
- The call will be ignored if the agent has been remotely disabled.
- If the call is removed from memory and the app restarts, or the device is rebooted, the agent will be initialized as normal.

Configure Hybrid Support

By default, the Android Agent instruments Android WebViews by injecting the JavaScript Agent into WebViews. See Hybrid Application Support for an overview and an explanation of how it works.

Runtime Configuration for Hybrid Support

The code example below disables the injection of the JavaScript Agent. If the client receives a `false` for this flag, then the JavaScript Agent will be disabled. Thus, WebViews will not be instrumented, and Ajax requests will not be monitored.

```
Instrumentation.start(
    AgentConfiguration.builder()
        .withAppKey(getString(R.string.app_key))
        .withContext(applicationContext)
        .withJSAgentInjectionEnabled(false)
        .build())
```

The injection occurs during the creation of a new WKWebView. So, if a WKWebView is created when this flag is set to `false`, that particular WKWebView won't be instrumented even if the flag is subsequently set to `true`.

The collection and reporting of Ajax calls are disabled by default. To enable the injection and the collection and reporting of Ajax calls, pass `true` to the method `jsAgentEnabled` in the instrumentation configuration as shown below.

```
Instrumentation.start(
    AgentConfiguration.builder()
    .withAppKey(getString(R.string.app_key))
    .withContext(applicationContext)
    .withJSAgentAjaxEnabled(true)
    .build())
```

Programmatically Control Sessions

By default, a mobile session ends after a period of user inactivity. For example, when a user opens your application, the session begins and only ends after the user stops using the application for a set period of time. When the user begins to use the application again, a new session begins.

Instead of having a period of inactivity to define the duration of a session, however, you can use the following API to programmatically control when sessions begin and end:

```
void startNextSession()
```

When you call the method `startNextSession` from the Instrumentation class, the current session ends and a new session begins. The API enables you to define and frame your sessions so that they align more closely with business goals and expected user flows. For example, you could use the API to define a session that tracks a purchase of a product or registers a new user.

Excessive use of this API will cause sessions to be throttled (excessive use is >10 calls per minute per Android Agent, but is subject to change). When not using the API, sessions will fall back to the default of ending after a period of user inactivity.

Example of a Programmatically Controlled Session

In the code example below, the current session ends and a new one begins when the check out is made.

```
public void checkoutCart(){
    if (currentCartItems!=null && currentCartItems.size()>0){
        CheckoutTask checkoutReq = new CheckoutTask();
        checkoutReq.execute(getEndpoint() + "cart/co");
        currentCartItemsMap.clear();
        convertItemsMaptoList();
        Instrumentation.startNextSession();
    } else {
        displayToast("There are no items in the cart");
    }
}
```

Start and End Session Frames

You can use the `SessionFrame` API to create session frames that will appear in the session activity. Session frames provide context for what the user is doing during a session. With the API, you can improve the names of user screens and chronicle user flows within a business context.

Use Cases

The following are common use cases for using the `SessionFrame` API:

- One Activity performs multiple functions and you want more granular tracking of the individual functions.
- A user flow spans multiple activities or user interactions. For example, you could use the API to create the session frames "Login", "Product Selection", and "Purchase" to chronicle the user flow for purchases.
- You want to capture dynamic information based on user interactions to name session frames, such as an order ID.

SessionFrame API

The table below lists the three methods you can use with session frames.

Class	Method	Description
`Instrumentation`	`static SessionFrame startSessionFrame (String sessionFrameName)`	Use this to start and name your session frame. Naming session frames enable you to easily identify and track the frames in the **Sessions Dialog.**
`SessionFrame`	`static void updateName(String updatedSessionFrameName)`	Use this to start and name your session frame. Naming session frames enable you to easily identify and track the frames in the **Sessions Dialog.**

`SessionFrame`	`static void end()`	End the session frame. You call this method from the `SessionFrame` object returned from `startSessionFrame`.

Session Frame Example

In the following example, the `ShoppingCartActivity` class uses the `SessionFrame` API to track user activity during the checkout process.

```
public class ShoppingCartActivity extends Activity {

  SessionFrame checkoutSessionFrame;

  public void onCheckoutCartButtonClicked() {
    // The user starts the checkout by clicking the checkout button.
    // This may be after they have updated the quantities of items in the cart, etc.
    checkoutSessionFrame = Instrumentation.startSessionFrame("Checkout");
  }

  public void onConfirmOrderButtonClicked() {
    // Once they have confirmed payment info and shipping information, and they
    // are clicking the "Confirm" button to start the backend process of checking out,
    // we may know more information about the order itself, such as an order ID.
    checkoutSessionFrame.updateName("Checkout: Order ID " + orderId);
  }

  public void onProcessOrderCompleted() {
    // Once the order is processed, the user is done "checking out", so we end the session frame.
    checkoutSessionFrame.end();
    checkoutSessionFrame = null;
  }

  public void onCheckoutCanceled() {
    // If the user cancels or returns to the cart, you'll want to end the session frame also, or else it will be
    // left open and appear to have never ended.
    checkoutSessionFrame.end();
    checkoutSessionFrame = null;
  }
}
```

Use a Custom HTTP Library

The Android Agent automatically detects network requests when the underlying implementation is handled by any one of the supported network libraries. To have the Android Agent detect requests from a custom library, add request tracking code to your application manually, using the `HttpRequestTracker` interface.

Supported Network Libraries

The libraries below cover the great majority of Android network requests. In some cases, however, mobile applications use custom HTTP libraries.

- `HttpURLConnection`
- `HttpsURLConnection`
- `HttpClient` classes
- `OkHttp`
- `OkHttp3`
- `ch.boye.httpclientandroidlib`

To set headers to allow correlation with server-side processing, use the `ServerCorrelationHeaders` class.

Add Request Tracking

To add request tracking manually, you use an `HttpRequestTracker` object to tell the agent when the request begins and when it ends and to report fields of the response to the agent.

Tracking a request

To begin tracking an HTTP request, use an instance of the following interface.

You must initialize the agent using the `Instrumentation.start` method before using this interface.

```
public interface HttpRequestTracker {
    public Exception getException();
    public HttpRequestTracker withException(Exception e);

    public String getError();
    public HttpRequestTracker withError(String error);

    public int getResponseCode();
    public HttpRequestTracker withResponseCode(int responseCode);

    public Map<String, List<String>> getResponseHeaderFields();
    public HttpRequestTracker withResponseHeaderFields(Map<String, List<String>> responseHeaderFields);

   /**
    * Stops tracking an HTTP request.
    *
    * Immediately after receiving a response or an error, set the appropriate fields and call this method to
    * report the outcome of the HTTP request. You should not continue to use this object after calling this
    * method -- if you need to track another request, obtain a new instance.
    */
    public void reportDone();
}
```

Example

Given a request snippet like this:

```
public byte[] sendRequest(URL url) throws HttpException {
    try {
        // implementation omitted
        return responseBody;
    } catch (UnderlyingException e) {
        throw new HttpException(e);
    }
}
```

Adding the tracker could look like this:

```
public byte[] sendRequest(URL url) throws HttpException {
    HttpRequestTracker tracker = Instrumentation.beginHttpRequest(url);
    try {
        // implementation omitted
        tracker.withResponseCode(theResponseCode)
               .withResponseHeaderFields(theResponseHeaderFields)
               .reportDone();
        return responseBody;
    } catch (UnderlyingException e) {
        tracker.withException(e)
               .reportDone();
        throw new HttpException(e);
    }
}
```

Attach a custom user data property to a network request

You can attach a custom user data property to a network request by adding `withUserData` to `HttpRequestTracker`:

```
public HttpRequestTracker withUserData(String key, String value)
```

Enable Server-Side Correlation

To enable correlation between your request and server-side processing, add specific headers to outgoing requests that the server-side agent can detect.

This is done automatically for standard HTTP libraries.

```
public class ServerCorreleationHeaders {
    public static Map<String, List<String>> generate();
}
```

You must:

1. Call the `generate` method and set the generated headers *before* sending a request to the backend.
2. Report back the response headers, using data from the `withResponseHeaderFields` field.

Override the Request/Response Content-Length

- You can generally obtain the content lengths of the network request and response by passing the headers with `HttpRequestTracker.withRequestHeaderFields()` and `HttpRequestTracker.withResponseHeaderFields()`.
- If for some reason this does not work for your custom HTTP tracking (i.e. the network library doesn't populate those fields until its being transmitted), then you can still report the request and response content lengths using `HttpRequestTracker.withRequestContentLength(Long length)` and `HttpRequestTracker.withResponseContentLength(Long length)`.
- For example, you want to track a request that has a `byte` array of content. You can report the request content length by passing the size of the byte array:

```
byte[] requestContent;
HttpRequestTracker tracker;
tracker.withRequestContentLength(requestContent.size());
```

Customize the Agent Configuration

To customize the behavior of the agent itself, you pass the `AgentConfiguration` object to the `Instrumentation.start` method. The `AgentConfiguration` object allows you to:

- Point to an on-premises EUM Server
- Enable logging
- Custom-set the application name, which is useful if you deploy the same application binary with different package names to different geographic areas. This ensures that all the data ends up being processed under the same name.
- Ignore HTTP requests internal to your application that are not used for network requests
- Configure the agent to use your custom HTTP library to send its beacons

Syntax

See the latest JavaDocs for more information.

```
Instrumentation.start(AgentConfiguration.builder()
      .withAppKey("<EUM_APP_KEY>")
      .withContext(getApplicationContext())
      .withCollectorURL(collectorURL*) // The URL of the EUM Server(on-prem)
      .withCompileTimeInstrumentationCheck(true) // Set to false if you are using features of the SDK only,
like custom HTTP support, but not to instrument your app.
      .withLoggingEnabled(true)//set default INFO logging. Tagged "AppDynamics".
      .withApplicationName(applicationName)//set a custom app name
      .withExcludedUrlPatterns(excludedUrlPatterns) // Set excluded url regex patterns for http tracking
      .withCollectorChannelFactory(collectorChannelFactory()) // The custom HTTP implementation to use
      .build());
```

*For a list of URLs of EUM servers, see External Access Locations.

Configure the Agent to Use Custom HTTP Library

The Android Agent uses HTTP to deliver beacons. If you want the agent use your custom HTTP library to deliver beacons:

1. Implement a class that extends the following abstract class:

```
public abstract class CollectorChannel {
    private URL url;
    private int connectTimeout;
    private int readTimeout;
    private Map<String, List<String>> requestProperties = new HashMap<String, List<String>>();
    private String requestMethod;

    public void setURL(URL url) {
        this.url = url;
    }

    public URL getURL() {
        return url;
    }

    public void setConnectTimeout(int connectTimeout) {
        this.connectTimeout = connectTimeout;
    }

    public int getConnectTimeout() {
        return connectTimeout;
    }

    public void setReadTimeout(int readTimeout) {
        this.readTimeout = readTimeout;
    }

    public int getReadTimeout() {
        return readTimeout;
    }

    public void addRequestProperty(String property, String value) {
        if (!requestProperties.containsKey(property)) {
            requestProperties.put(property, new ArrayList<String>());
        }
        requestProperties.get(property).add(value);
    }

    public Map<String, List<String>> getRequestProperties() {
        return Collections.unmodifiableMap(requestProperties);
    }

    public void setRequestMethod(String requestMethod) {
        this.requestMethod = requestMethod;
    }

    public String getRequestMethod() {
        return requestMethod;
    }

    public abstract OutputStream getOutputStream() throws IOException;

    public abstract InputStream getInputStream() throws IOException;

    public abstract int getResponseCode() throws IOException;

    public abstract Map<String,List<String>> getHeaderFields() throws IOException;
}
```

This interface is loosely based on `HttpURLConnection`.

2. Implement a version of the `CollectorChannelFactory` interface:

```
public interface CollectorChannelFactory {

    /**
     * Returns a new instance of CollectorChannel.
     *
     * If you want to supply a custom CollectorChannel, implement this interface, and return
     * an instance of your concrete implementation of CollectorChannel from this method.
     */
    public CollectorChannel newCollectorChannel();
}
```

The implementation of `newCollectorChannel` should return a new instance of your implementation of `CollectorChannel`.

3. Pass the `CollectorChannelFactory` to the `AgentConfiguration` object.

Capture User Interactions

You can enable the agent to track certain user interactions. Once user interactions have been captured, you can sort sessions by UI event and view the UI event in the timeline of the session waterfall.

You can capture when users do one or all of the following:

- Press buttons
- Select a text field
- Click a list item

Security and Privacy Concerns

The interaction capture mode is disabled by default for security and privacy reasons as user interactions may contain sensitive information. This potential security and privacy issue may be compounded if you enable both the capturing of UI interactions and screenshots.

Enable User Interaction Capture Mode

To enable user interaction capture mode, pass the capture mode to the method `withInteractionCaptureMode()` from an `AgentConfiguration` obj ect. The instrumentation code example below configures the Android Agent to capture all the supported types of user interactions.

```
Instrumentation.start(AgentConfiguration.builder()
      .withAppKey("<EUM_APP_KEY>")
      .withContext(getApplicationContext())
      .withInteractionCaptureMode(InteractionCaptureMode.All)
      .build());
```

You can also configure the Android Agent to only capture one type of user interaction:

```
Instrumentation.start(AgentConfiguration.builder()
      .withAppKey("<EUM_APP_KEY>")
      .withContext(getApplicationContext())
      .withInteractionCaptureMode(InteractionCaptureMode.ButtonPressed)
      .build());
```

Configure and Take Screenshots

Mobile screenshots are enabled by default in the agent. You can configure the Controller UI to automatically take screenshots or use the Android SDK to manually take a screenshot:

```
Instrumentation.takeScreenshot();
```

For example, you might want to take a screenshot after you load a UI element to view how the element is displayed to customers:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_spinner_with_toast);
   spinner = (Spinner) findViewById(R.id.spnOptions);
   btnSpinnerVal = (Button) findViewById(R.id.btnSpinnerValue);
   loadSpinner();
   Instrumentation.takeScreenshot();
}
```

Disable Screenshots

You can disable screenshots from the Controller UI or with the Android SDK. To disable screenshots with the Android SDK, use the method `withScreens hotsEnabled(false)` from the `AgentConfiguration` class:

```
Instrumentation.start(AgentConfiguration.builder()
     .withAppKey("<EUM_APP_KEY>")
     .withContext(getApplicationContext())
     .withScreenshotsEnabled(false)
     .build());
}
```

Block/Unblock Screenshots

You can also use the Android SDK to block screenshots from being taken during the execution of a code block. This just temporarily blocks screenshots from being taken until you unblock screenshots. This enables you to stop taking screenshots in situations where users are entering personal data, such as on login and account screens.

You use the methods `Instrumentation.blockScreenshots()` and `Instrumentation.unblockScreenshots()` to block and unblock screenshots. If screenshots are disabled through `AgentConfiguration.Builder.withScreenshotsEnabled(true)` or through the Controller UI, these methods have no effect. You can call `Instrumentation.screenshotsBlocked()` to check if screenshots are being blocked.

The following example demonstrates how you could use the API to block and unblock screenshots when displaying user information.

```
public class UserAccountActivity extends Activity {
  ...
  public void displayCustomerAccount() {
    // Check to see if screenshots are blocked
    if (! Instrumentation.screenshotsBlocked()) {
        // If screenshots aren't blocked, block them before showing customer details
        Instrumentation.blockScreenshots();
    }
    // Code to display customer details
    displayAccount(this.user);
    // After you're done, unblock screenshots
    Instrumentation.unblockScreenshots();
  }
  ...
}
```

Transform URLs for Network Requests

When your application makes network requests, you may not want to report URLs containing sensitive information to the EUM Server. You can instead transform the network request URL before reporting it or ignore it altogether:

1. Implement a network request callback that modifies or ignores specific URLs.
2. Register the network request callback in the initialization code.

Implement the Network Request Callback

The callback that modifies or ignores specific URLs is an implementation of the interface below. The method `onNetworkRequest` is synchronous, we recommend that you return from the function quickly.

```
public interface com.appdynamics.eumagent.runtime.NetworkRequestCallback  {
    boolean onNetworkRequest(HttpRequestTracker httpRequestTracker);
}
```

Transforming URLs

The `onNetworkRequest` method, in general, should follow the steps below to transform URLs:

1. Identify specific URLs using techniques such as regex or pattern matching.
2. Modify the `url` property of the `HttpRequestTracker` object.
3. Assign a valid URL to the `url` property. Modifying other properties of the `HttpRequestTracker` object will be ignored.
4. Return `true`.

The first step is optional as you could choose to transform the URLs of all network requests.

```
private static class myNetworkRequestCallback implements com.appdynamics.eumagent.runtime.
NetworkRequestCallback  {
    @Override
    public boolean onNetworkRequest(HttpRequestTracker httpRequestTracker) {
        URL urlMask = new URL("http://networkrequest-mask.com/");
        httpRequestTracker.withURL(urlMask);
        return true;
    }
}
```

In general, however, you would want to identify and transform URLs that contain sensitive information. For example:

```
private static class myNetworkRequestCallback implements com.appdynamics.eumagent.runtime.
NetworkRequestCallback  {
    @Override
    public boolean onNetworkRequest(HttpRequestTracker httpRequestTracker) {
        String urlString = httpRequestTracker.getURL().toString();
        try {
            URL url = new URL("http://customer-account.com/");
            if (urlString.contains("accountInfo")) {
                // Change the URL for calls to Facebook
                httpRequestTracker.withURL(url);
                return true;
            }
        } catch (MalformedURLException e) {
            return false;
        }
        return true;
    }
}
```

Ignoring URLs

If the `onNetworkRequest` method returns `false`, the beacon is dropped. The general process for ignoring beacons is:

1. Identify specific URLs using techniques such as regex or pattern matching.
2. Return `false`.

You could also ignore all network requests with the following implementation of `onNetworkRequest:`

```
private static class myNetworkRequestCallback implements com.appdynamics.eumagent.runtime.
NetworkRequestCallback  {
    @Override
    public boolean onNetworkRequest(HttpRequestTracker httpRequestTracker) {
        return false;
    }
}
```

In general, you would identify network requests that you didn't want to monitor and return `false` to ignore the network request. For example:

```
private static class myNetworkRequestCallback implements com.appdynamics.eumagent.runtime.
NetworkRequestCallback  {
    @Override
    public boolean onNetworkRequest(HttpRequestTracker httpRequestTracker) {
        String urlString = httpRequestTracker.getURL().toString();
        try {
            URL url = new URL("http://socialnetworksite.com/");
            if (urlString.contains("avatar")) {
                // Ignore calls for avatars
                return false;
            }
        } catch (MalformedURLException e) {
            return false;
        }
        return true;
    }
}
```

Register the Network Request Callback

After implementing the callback, you register it in the initialization code. When the Android Agent is ready to create a network request beacon, it will first call the callback with an `HttpRequestTracker` object.

To register the callback in the initialization code:

```
Instrumentation.start(AgentConfiguration.builder()
        .withContext(getApplicationContext())
        .withAppKey("<EUM_APP_KEY>")
        .withNetworkRequestCallback(new myNetworkRequestCallback())
        .build());
```

Enable Logging and Set Logging Level

You use the method `withLoggingLevel` of the class `AgentConfiguration` to enable logging and set the logging level. You can set logging to one of the following levels:

- `LOGGING_LEVEL_NONE`
- `LOGGING_LEVEL_INFO`
- `LOGGING_LEVEL_VERBOSE`

Use verbose logging only for troubleshooting. Make sure to disable for production.

Example:

```
AgentConfiguration config = AgentConfiguration.builder()
config.withAppKey(appKey)
config.withContext(context)
 .withLoggingLevel(Instrumentation.LOGGING_LEVEL_VERBOSE)
        .build();
    Instrumentation.start(config);
```

Android SDK Documentation

For the complete SDK API documentation, see the latest JavaDocs or the previous versions listed below:

- https://sdkdocs.appdynamics.com/android-sdk/21.5/
- https://sdkdocs.appdynamics.com/android-sdk/21.2/
- https://sdkdocs.appdynamics.com/android-sdk/20.11/
- https://sdkdocs.appdynamics.com/android-sdk/20.10/
- https://sdkdocs.appdynamics.com/android-sdk/20.7/
- https://sdkdocs.appdynamics.com/android-sdk/20.5/
- https://sdkdocs.appdynamics.com/android-sdk/20.3/

Manually Upload Mapping Files

Related pages:

- Automatically Upload Mapping Files

If you have obfuscated your code and want to monitor crashes, you will need to upload the ProGuard or DexGuard mapping files. AppDynamics needs the mapping file for the application to produce human-readable stack traces for crash snapshots. For details about why you should do this, see Get Human-Readable Crash Snapshots.

DexGuard is built on ProGuard, so the Controller UI and some of the REST API for uploading mapping files will refer to ProGuard for both ProGuard and DexGuard files.

This page describes the requirements, the available methods, and instructions to manually upload the mapping files.

It is highly recommended to set up automatic uploading of the ProGuard or DexGuard mapping files using Gradle and your `build.gradle` file.

Requirements

To manually upload the mapping files, you need to associate the mapping file with the correct version of the application by providing the following:

- the package name of the Android package for the application
- the version code for that application specified in either the `AndroidManifest.xml` file or the `build.gradle` file

Upload Methods

You can then either upload the mapping file using the **Upload missing ProGuard Mappings** dialog in the Controller UI or use a special REST API. Perform the upload separately for each ProGuard mapping file that you are providing. You can also check your uploads with a separate REST API.

If you update your application, you need to upload the new version of the mapping file.

Upload with the Controller UI

1. From the Mobile App menu, click **Configuration**.
2. Click **ProGuard Mappings >**.
3. From the **Upload missing ProGuard Mapping** dialog:
    a. Enter the version code (a number) for the package. This is the `versionCode` property specified in either the `AndroidManifest.xml` file or the `build.gradle` file of the application for which this mapping file was generated.
    b. Click **Choose File.**
    The uploader expects a file with the `.txt` extension. The file is named `mapping.txt`.
    c. In the file browser, locate and select the mapping file and click **Open**.
    d. Click **Upload**.

Upload with the REST API

The API uses HTTP basic authentication. The username is your AppDynamics EUM account name and the password is your EUM license key.

Send the mapping file

Send the mapping file as a text file in the body of the PUT request to the following URL:

```
https://api.eum-appdynamics.com/v2/account/<MyAccountName>/<androidPackageName>/<versionCode>/proguard-mapping
```

These parameters are required:

- `MyAccountName`: the URL-encoded version of your account name.
- `androidPackagename`: the name of the Android package for which this mapping file was generated.
- `versionCode`: the string representation of the "versionCode" property specified in either the `AndroidManifest.xml` file or the `build.gradle` file of the application for which this mapping file was generated.

The request body contains the mapping file. You need to add a `Content-Type` header, `-H Content-Type:text/plain,` and your account name and license key/password to the call.

Sample Request and Response Using the REST API

This is a sample request and response using the REST API.

Upload Request

The following example uses curl to send a ProGuard mapping file. You would use a similar command to send a DexGuard mapping file. The account name is "Example account" and the license key/password is "Example-License-Key-4e8ec2ae6cfe". The plus signs replace spaces in the account name when the account name is URL-encoded. The package name of the Android application is "com.example.networklogger". The mapping file corresponds to the version with `versionCode`.

```
curl -v -H Content-Type:text/plain --upload-file mapping.txt --user Example+account:Example-License-Key-
4e8ec2ae6cfe https://api.eum-appdynamics.com/v2/account/Example+account/com.example.networklogger/1/proguard-
mapping
```

Upload Response

The successful output of the example request looks like this:

```
* About to connect() to api.eum-appdynamics.com port 443 (#0)
*   Trying ::1...
* connected
* Connected to api.eum-appdynamics.com  (::1) port 443 (#0)
* Server auth using Basic with user 'Example+account'
> PUT /v2/account/Example+account/com.example.networklogger/1/proguard-mapping HTTP/1.1
> Authorization: Basic SW50ZXJuYWwrdGVzdCthY2NvdW50OlRlc3RBY2N0LTFlMzktNDVkMy05MzAzLTRlOGVjMmFlNmNmZQ==
> User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8y zlib/1.2.5
> Host: app.eum-appdynamics.com
> Accept: */*
> Content-Length: 4
> Expect: 100-continue
>
< HTTP/1.1 100 Continue
* We are completely uploaded and fine
< HTTP/1.1 200 OK
< Content-Length: 0
< Server: Jetty(8.1.4.v20120524)
<
* Connection #0 to host app.eum-appdynamics.com left intact
* Closing connection #0
```

Check Uploaded Mapping Files Using the REST API

You can check to make sure that your mapping files have successfully uploaded using two REST APIs.

1. Get a list of the GUIDs for the last 50 mapping files you have uploaded.
2. Check if a specific mapping file has been uploaded.

List of the Last 50 Mapping Files Uploaded

The `proguardQuery` method allows you to retrieve a list of GUIDs for up to the last 50 mapping files (ProGuard or DexGuard) that have been uploaded to your account, along with the time they were uploaded. The response is displayed as JSON, by upload time, with the most recent first.

1. Set up your authentication as described in Upload with the API.
2. Create a GET request of the form:

   ```
   curl --user Example+account:Example-License-Key-4e8ec2ae6cfe https://<EUM_Cloud/EUM_Server_Host:port>/v2
   /account/<EUM_Account_Name>/crash-symbol-file-query/proguard
   ```

   where the value for `--user` is the authentication string you created in step 1, **`EUM_Cloud`** refers to `api.eum-appdynamics.com:443` for SaaS-based EUM Cloud accounts or **EUM_Server_Host**, which refers to the URL where your on-premises EUM Server is hosted, and **`EUM_Account_Name`** is your EUM account name.

Sample Request

```
curl --user Example+account:Example-License-Key-4e8ec2ae6cfe  https://api.eum-appdynamics.com:443/v2/account
/<EUM_Account_Name>/crash-symbol-file-query/proguard
```

Sample Response

```
{"proguardFiles": [
{"packageName":"my_package_name1", "version":"1", "uploadTime":"mm/dd/yyyy 16:09:23","GUID":"my_build_id1"},
{"packageName":"my_package_name2", "version":"1", "uploadTime":"mm/dd/yyyy 16:09:23","GUID":"my_build_id2"}
]}
```

Check for Specific Mapping File by GUID

The `checkForProguardFile` method allows you to check if a specific ProGuard or DexGuard file by GUID has been uploaded. The upload time is returned in the response.

a. Set up your authentication as described in Upload with the REST API.
b. Create a GET request of the form:

```
curl --user Example+account:Example-License-Key-4e8ec2ae6cfe https://<EUM_Cloud/EUM_Server_Host:
port>/v2/account/<EUM_Account_Name>/crash-symbol-file-query/proguard/guid/<GUID_To_Check>
```

where the value for `--user` is the authentication string you created in step 1, **`EUM_Cloud`** `refers` to `api.eum-appdynamics.com:443` for SaaS-based EUM Cloud accounts or **EUM_Server_Host** refers to the URL pointing to your on-premises EUM Server, **`EUM_Account_Name`** is your EUM account name and **`GUID_To_Check`** is the GUID of the ProGuard file in which you are interested.

Sample Request

```
curl --user Example+account:Example-License-Key-4e8ec2ae6cfe  https://api.eum-appdynamics.com:443
/v2/account/<EUM_Account_Name>/crash-symbol-file-query/proguard/guid/my_build_id1
```

Sample Response

```
{"packageName":"mypackagename","version":"1","uploadTime":"mm/dd/yyyy 16:09:23","GUID":"
my_build_id1"}
```

Troubleshoot the Android Instrumentation

Related pages:

- Instrument Android Applications
- Customize the Android Build
- Customize the Android Instrumentation

This page provide instructions and tips for solving some common Android instrumentation issues.

Dex Failure After Upgrading Agent

If you build your application with Gradle and you get a dex failure after upgrading the Mobile Agent, you likely need to clear the Gradle cache.

Use Android Studio

To clear the cache using Android Studio:

1. Select **Build** > **Clean Project.**
2. Select **File** > **Invalidate Caches.**

Sometimes a restart is also necessary.

Using the Command Line

To clear the cache using the command line:

1. Stop Gradle.

```
$ gradlew --stop
```

2. You may also need to refresh dependencies.

```
$ gradlew --refresh-dependencies
```

You can also try removing the AppDynamics cached images.

1. Find the images.

```
$ find ~/.gradle/caches -name com.appdynamics
```

2. Delete them.

```
$ find ~/.gradle/caches -name com.appdynamics -print0 | xargs -0 rm -r
```

Instrumentation Appears to Be Running After Being Disabled

If you disabled the instrumentation in Android using the Gradle flag `enabledForDebugBuilds`, but the instrumentation task still runs, this is because of the Transform API.

Android Gradle Plugin 1.5.0 introduced the Transform API that the Android Agent uses to do the bytecode injection. Due to the limitations of the Transform API, even when instrumentation is disabled, the `transformClassesWithAppDynamicsForDebug` task will still appear in the Gradle task log. To ensure that it is not actually instrumenting, run `gradle` with `--info` flag to show the info logs.

You should see the following log:

```
Instrumentation is disabled for this build variant. Just copying the input files to fulfill Transform contract.
```

Crashes Are Not Reported After Using Custom Default Uncaught Exception Handler

Install your exception handler before you start the Android Agent. When an uncaught exception is handled by the agent, it will first store the crash report, and then call your exception handler.

Network Requests Are Not Being Reported

Please see the list of supported networking libraries. If your library is not on the list, you can always report these network requests manually.

For builds with the `debuggable` attribute, auto-instrumentation (the `enableInstrumentation` build attribute) is disabled to improve build speeds, and you cannot enable auto-instrumentation for those builds. Debuggable builds using Android Agent 20.5.0-20.7.0 do not auto-instrument network requests.

Exclude Classes from Being Instrumented

- `excludeClasses`

Enforce a Different Runtime Version from the Plugin Version

- ```
  dependencies.compile 'com.appdynamics:appdynamics-runtime:4.5.+'
  adeum.dependencyInjection.enabled = false
  ```
  ```

Instrument Xamarin Applications

Before you can monitor your Xamarin application, you will need to instrument your application to enable the Xamarin Agent to collect mobile metrics.

After you have set up and accessed Mobile RUM, follow these instructions:

1. Confirm that the Xamarin Agent supports your platform
2. Understand the limitations of the Xamarin Agent
3. Instrument a Xamarin Application
4. Customize the Xamarin Instrumentation (Optional)

Xamarin Agent Support

Supported Platforms

- iOS and Android platforms

> All other platforms will build and run without errors, but no monitoring will occur.

Supported Libraries

- .Net Standard 2.0 (with Xamarin Agent >= 20.10.0)

> **Recommendations**
>
> Xamarin <= 50.2 only supported the instrumentation of applications that reference the Portable Class Libraries (PCL). From Xamarin >= 50.3, the Xamarin Agent only supports the .NET Standard library and the instrumentation of applications that reference the .NET Standard library. If you still need PCL support, you are recommended to use Xamarin <= 50.2.

Limitations

The Xamarin Agent has these limitations:

- Automatic instrumentation is not supported, so you will need to manually report events and metrics.
- Symbolication is not supported, although the Xamarin Agent *does* report uncaught exceptions and native application crashes.

Instrument Xamarin Applications

Follow the steps below to manually instrument your Xamarin iOS, Android, and Forms apps.

Add the Xamarin Agent Package

1. Get the Xamarin Agent from the NuGet Gallery. Follow the instructions given in Adding a Package to add the package `AppDynamics Xamarin Agent` from nuget.org.
2. In the `Xamarin.Android` project, add the following to `MainActivity.cs` under `OnCreate`:

```
AppDynamics.Droid.Agent.Init(this, bundle);
```

Example

```
public class MainActivity : global::Xamarin.Forms.Platform.Android.FormsAppCompatActivity {
    protected override void OnCreate(Bundle bundle) {
        base.OnCreate(bundle);  //existing code
        global::Xamarin.Forms.Forms.Init(this, bundle);

        AppDynamics.Droid.Agent.Init(this, bundle); //initialize the agent on the Android Platform

        LoadApplication(new App());
    }
}
```

Get Your Application Key

Complete the **Getting Started Wizard** to get an EUM App Key. You will need this key when you modify the source code. In some cases, multiple mobile applications can share the same key.

Because there is no Xamarin platform option, you will need to choose either Android or iOS. For Android, you will need to select **Manual**.

If you have completed the **Getting Started Wizard**, but don't have your EUM App Key, see Get Your Application Key.

Add the Required Permissions (Android Deployments Only)

Open the file `Properties/AndroidManifest.xml` and verify that it has these permissions:

```
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"></uses-permission>
```

If these permissions are not present, add them.

Link the AppDynamics Agent Assembly

Forms Solution

Xamarin Forms is the unified development environment for running all platforms. If you try to run something on an unsupported platform, linking our agent won't allow you to monitor the application, but it also will not cause any errors.

To use the Xamarin Agent for your iOS/Android apps, add the `using` directive at the top of the `App.xaml.cs` file:

```
using Xamarin.Forms;
using AppDynamics.Agent;

namespace <AppName>
{
    public partial class App : Application
    {
        ...
    }
}
```

iOS Solution

To use the Xamarin Agent in iOS apps, add the `using` directive at the top of the `AppDelegate.cs` file:

```
using Foundation;
using UIKit;
using AppDynamics.Agent;

public class AppDelegate : UIApplicationDelegate
{
    ...
}
```

Android Solution

To use the Xamarin Agent in Android apps, add the using directive at the top of the `MainActivity.cs` file:

```
using Android.App;
using Android.Widget;
using Android.OS;
using System;
using Android.Content;
using AppDynamics.Agent;

namespace <AppName>
{
    [Activity(Label = "Phoneword", MainLauncher = true, Icon = "@mipmap/icon")]
    public class MainActivity : Activity
    {
       ...
    }
    ...
}
```

Initialize the Agent

To initialize the Xamarin Agent, you use the code below for iOS and Android. Use the EUM app key (enter as a string) that you received after completing step 2.

```
var config = AppDynamics.Agent.AgentConfiguration.Create(<EUM_APP_KEY>);
AppDynamics.Agent.Instrumentation.InitWithConfiguration(config);
```

If you are running an on-premises EUM Server, you need to specify the URL to the EUM Server. See Point to an On-Premises EUM Server (Optional) to learn how.

Forms Solution

For Forms Solutions, you only need to place the initialize code in the constructor of the `App.xaml.cs` file for the Xamarin Agent to instrument both Android and iOS applications.

```
public App()
{
    InitializeComponent();
    // This initialization code is used by both iOS and Android apps.
    var config = AppDynamics.Agent.AgentConfiguration.Create(<EUM_APP_KEY>);
    AppDynamics.Agent.Instrumentation.InitWithConfiguration(config);
    MainPage = new FormsExamplePage();
}
```

If you have application code in the `MainActivity.cs` file for Android or `AppDelegate.cs` for iOS that you want to instrument, however, you should initialize the Xamarin Agent in those files as you would do for iOS Solutions and Android Solutions.

iOS Solution

For iOS apps, you place the initialize code In the `AppDelegate.cs` file in the method `FinishedLaunching` of the class `AppDelegate` as shown below.

```
public class AppDelegate : UIApplicationDelegate
{
   // class-level declarations
   public override UIWindow Window
   {
      get;
      set;
   }
   public override bool FinishedLaunching(UIApplication application, NSDictionary launchOptions)
   {
       // The two lines below initialize the AppDynamics instrumentation.
       var config = AppDynamics.Agent.AgentConfiguration.Create(<EUM_APP_KEY>);
       AppDynamics.Agent.Instrumentation.InitWithConfiguration(config);
       ...
       return true;
    }
    ...
}
```

You may also consider placing it in the `Main.cs` int the method `Main`.

Android Solution

In the `MainActivity.cs` file, place the initialization code in the method `OnCreate`:

```
class MainActivity {

  protected override void OnCreate(Bundle savedInstanceState) {
    // The two lines below initialize the AppDynamics instrumentation.
    var config = AppDynamics.Agent.AgentConfiguration.Create(<EUM_APP_KEY>);
    AppDynamics.Agent.Instrumentation.InitWithConfiguration(config);
    ...
  }
}
```

Build the Application

Run and build your application from Visual Studio. From the **Getting Started Wizard**, you should see that the application has connected and the instrumentation has been verified.

For iOS projects with Xamarin Agent 20.10.0 - 21.4.0, you must add the additional `MtouchExtraArgs` argument `--gcc_flags "-ObjC -lz"` for each build configuration, such as the `debug` and `release` builds. If the iOS project file is edited directly, the build configuration should contain the `MtouchExtraArgs` element:

```
<PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|iPhoneSimulator' ">
    <DebugSymbols>true</DebugSymbols>
    <DebugType>full</DebugType>
    <Optimize>false</Optimize>
    <OutputPath>bin\iPhoneSimulator\Debug</OutputPath>
    <DefineConstants>DEBUG;ENABLE_TEST_CLOUD;</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
    <CodesignKey>iPhone Developer</CodesignKey>
    <MtouchDebug>true</MtouchDebug>
    <MtouchFastDev>true</MtouchFastDev>
    <MtouchLink>SdkOnly</MtouchLink>
    <MtouchArch>x86_64</MtouchArch>
    <MtouchHttpClientHandler>HttpClientHandler</MtouchHttpClientHandler>
    <MtouchTlsProvider>Default</MtouchTlsProvider>
    <DeviceSpecificBuild>false</DeviceSpecificBuild>
    <MtouchExtraArgs>--gcc_flags "-ObjC -lz"</MtouchExtraArgs>
  </PropertyGroup>
```

Customize Your Instrumentation (Optional)

The Xamarin SDK has additional classes to allow you to extend the kinds of application data you can collect and aggregate using Mobile RUM. See Customize the Xamarin Instrumentation.

Point to an On-Premises EUM Server (Optional)

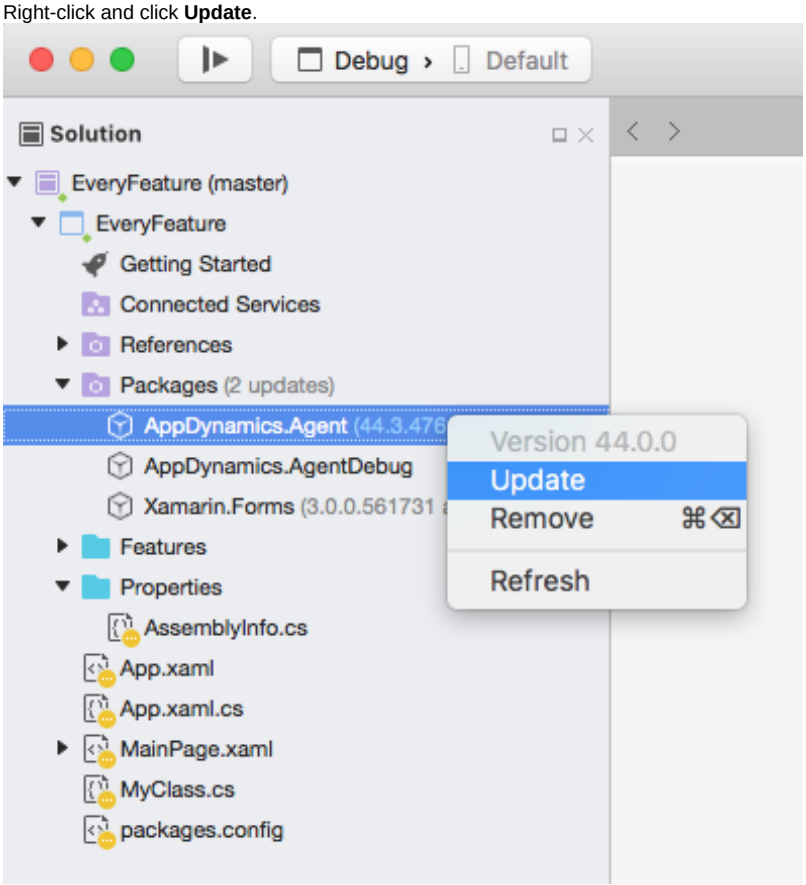
To use an on-premises EUM Server, you pass the URL to the on-premises EUM Server when you initialize the instrumentation with the EUM App Key from Get Your Application Key:

```
var config = AppDynamics.Agent.AgentConfiguration.Create(<EUM_APP_KEY>);
config.CollectorURL = <COLLECTOR_URL:PORT>;
AppDynamics.Agent.Instrumentation.InitWithConfiguration(config);
```

Upgrade the Xamarin Agent

As new features are added to the agent, you will need to upgrade the Xamarin Agent package in your app.

1. From Visual Studio, open the Xamarin application that has the AppDynamics Agent package.
2. From the **Packages** folder, select the AppDynamics Agent.
3. Right-click and click **Update**.



Customize the Xamarin Instrumentation

Related pages:

- Instrument Xamarin Applications
- Xamarin SDK Documentation

The following sections show you how to use the Xamarin SDK to customize your instrumentation.

Because the agent stores data about events in a local buffer before reporting the information, you are recommended to use the APIs with discretion.

Track Calls

You can instrument methods to see how often the instrumented a method is invoked and how long it takes to run. To do this, add a call at the beginning and end of the method you'd like to instrument.

In the example below, the code executed in the constructor for the class `ShoppingCart` will be tracked and reported. In your own code, start tracking calls by specifying the class and method in `BeginCall` and then complete the tracking and report the data by calling `ReportCallEnded`.

```
using AppDynamics.Agent;
...
public class ShoppingCart {
    public ShoppingCart() {

        // Code to create the shopping cart
    }
    void CheckOut(int custId, int transactionId) {
      var tracker = Instrumentation.BeginCall("ShoppingCart", "CheckOut", custId, transactionId);
       // The code placed here will be tracked and reported.
       tracker.ReportCallEnded();
    }
}
```

Timing Events

Sometimes you want to time an event in your application that spans multiple methods. You can do this by calling `StartTimerWithName` when the event starts, and then `StopTimerWithName` when it ends. For example, to track the time a user spends viewing a screen, the instrumentation might look something like the following:

```
using AppDynamics.Agent;
...
async private void StartCapturePreview_Click(object sender, RoutedEventArgs e) {
    capturePreview.Source = captureManager;
    Instrumentation.StartTimerWithName("CapturePreview");
    await captureManager.StartPreviewAsync();
}
async private void StopCapturePreview_Click(object sender, RoutedEventArgs e) {
    await captureManager.StopPreviewAsync();
    Instrumentation.StopTimerWithName("CapturePreview");
}
```

Report Metrics

To report other types of data, you can use a metric. The metric name should only contain alphanumeric characters and spaces. Illegal characters are replaced by their ASCII hex value. The metric value must be a long integer.

The snippet below shows how you might report a metric.

```
using AppDynamics.Agent;
...
Instrumentation.ReportMetricWithName("Database Rows", 5123);
```

Add Tracking to HTTP Requests

Semi-automatic HTTP Tracking

The `HttpMessageHandler` handles all the tracking and error handling. It can also include other inner handlers if you are already using a custom `HttpMessageHandler` for other purposes, such as for logging.

To add semi-automatic tracking, instantiate a `HttpClient` and pass the `HttpRequestTrackerHandler`:

```
var client = new HttpClient(new HttpRequestTrackerHandler());
```

Then, all the requests sent using the client will be already instrumented:

```
 response = await client.GetAsync(uri);
```

If you already have `HttpMessageHandler` passed to the `HttpClient` (for example, adding a logging handler), you must instantiate `HttpRequestTrackerHandler` and pass the existing handler to the constructor:

```
var loggingHandler = new MyLoggingHandler();
var client = new HttpClient(new HttpRequestTrackerHandler(loggingHandler));
```

Manual HTTP Tracking

You can manually report a network request using the `AppDynamics.Agent.HTTPRequestTracker` class.

The example below uses `HTTPRequestTracker` with the `System.Net.Http.HttpClient` class. The `tracker` object synchronously captures and reports the network request as well as any network errors.

```
using AppDynamics.Agent;
...
public async Task<string> Fetch(Uri uri) {
  var client = new HttpClient();
  // Create AppDynamics Tracker
  var tracker = HTTPRequestTracker.Create(uri);
  // Add AppDynamics Server Correlation Headers
  foreach (var header in ServerCorrelationHeaders.Generate) {
      // Each header could have multiple values
      foreach (var value in header.Value) {
          client.DefaultRequestHeaders.Add(header.Key, value);
      }
  }
  HttpResponseMessage response = null;
  try {
      response = await client.GetAsync(uri);
  } catch (Exception ex) {
      // Capture any network errors.
      tracker.Exception = ex;
      tracker.ReportDone();
      throw ex; //you decide to throw it or not
  }
  if (!response.Equals(null)) {
      // Capture request information such as the
      // status code, status message, and headers.
      tracker.ResponseCode = (int)response.StatusCode;
      tracker.StatusLine = response.ReasonPhrase;
      tracker.ResponseHeaderFields = response.Headers;
      tracker.ReportDone();
      return await response.Content.ReadAsStringAsync();
   }
   return null;
}
```

Leave Breadcrumbs

You can leave breadcrumbs to mark interesting events. For example, if your application crashes, the breadcrumbs you left with be displayed in the crash report and could provide context. You can also configure the breadcrumb to appear in sessions.

The following is the method signature for leaving breadcrumbs:

```
static void AppDynamics.Agent.Instrumentation.LeaveBreadcrumb(string breadcrumb, BreadcrumbVisibility mode)
```

You use the mode to set the visibility of the breadcrumb. The visibility defines where you will see the breadcrumb in the Controller UI. The value of `mode` can be one of the following:

- `BreadcrumbVisibility.CrashesOnly` – The breadcrumb will only appear in crash snapshots.
- `BreadcrumbVisibility.CrashesAndSessions` – The breadcrumb will appear in crash snapshots and sessions.

Thus, you would use the method below to set breadcrumbs that are only reported in crash reports:

```
using AppDynamics.Agent;
...
Instrumentation.LeaveBreadcrumb("GetUserInfo", BreadcrumbVisibility.CrashesOnly);
```

If you would like to see the breadcrumb in crash reports and sessions:

```
using AppDynamics.Agent;
...
Instrumentation.LeaveBreadcrumb("GetUserInfo", BreadcrumbVisibility.CrashesAndSessions);
```

Disable the Agent to Stop Sending User Data to the Collector

You can disable the agent to stop sending all data to the collector while the agent is initialized and running. For example, you can disable the agent if your app has an option for users to opt-out of monitoring for privacy reasons.

`shutdownAgent`

The `shutdownAgent` call stops outgoing data to the collector, and does not persist data on the device.

```
using AppDynamics.Agent;
...
Instrumentation.shutdownAgent();
```

- The call only stops the traffic out of the agent.
- Once the agent has been initialized, the call cannot be removed, and a license will have been consumed.
- If you want to make this state permanent for a device, add code in UserDefaults to save the state and use that flag to conditionally initialize the agent in your code.

`restartAgent`

To re-enable the agent and reverse `shutdownAgent`, use `restartAgent`.

```
using AppDynamics.Agent;
...
Instrumentation.restartAgent();
```

- This call will respect the server side calls that can remotely shutdown the agent in a similar way.
- The call is only in effect while the app is running.
- The call will be ignored if the agent has been remotely disabled.
- If the call is removed from memory and the app restarts, or the device is rebooted, the agent will be initialized as

Report Errors and Exceptions

You can report exceptions using the method `reportError` from the `Instrumentation` class. Reported exceptions will appear in session details.

You can also set one of the severity levels below for an issue. With the severity level, you can filter errors in the **Code Issues Dashboard** or **Code Issues Analyze**.

- `ErrorSeverityLevel.INFO`
- `ErrorSeverityLevel.WARNING`
- `ErrorSeverityLevel.CRITICAL`

The example below uses the API to report possible exceptions and sets the severity level to `ErrorSeverityLevel.CRITICAL` (critical) when writing to a file

```
using AppDynamics.Agent;
...
try {
    // possible exception //
}
catch (Exception e){
    Instrumentation.ReportError(exception, ErrorSeverityLevel.CRITICAL);
}
```

Disable the Reporting of Exceptions as Crashes

By default, the Xamarin Agent reports exceptions as a crash. You can enable or disable the all exceptions as crashes with the boolean property `enableExceptionReporting`. The default value is `true`.

```
using AppDynamics.Agent;
...
var config = AppDynamics.Agent.AgentConfiguration.Create(<EUM_APP_KEY>);
AppDynamics.Agent.Instrumentation.enableExceptionReporting = false;
AppDynamics.Agent.Instrumentation.InitWithConfiguration(config);
...
```

Report Aggregate Exceptions as Crashes

You can configure the Xamarin Agent to report aggregate exceptions (handled and unhandled) as crashes by setting the boolean property `EnableAggregateExceptionHandling` to `true`. When the property is set to `false`, only unhandled exceptions are reported. The default value is `false`.

The following code example configures the Xamarin Agent to report aggregate exceptions (handled and unhandled) as crashes.

```
using AppDynamics.Agent;
...
var config = AppDynamics.Agent.AgentConfiguration.Create(<EUM_APP_KEY>);
AppDynamics.Agent.Instrumentation.EnableAggregateExceptionHandling = true;
AppDynamics.Agent.Instrumentation.InitWithConfiguration(config);
...
```

Disable Crash Reporting

Crash reporting is enabled by default, but you can manually disable crash reporting through the instrumentation configuration. If you are using other crash reporting tools, you might disable crash reporting to minimize conflicts and optimize the crash report results.

You can disable crash reporting by configuring the instrumentation with the `crashReportingEnabled` property as shown in the following:

```
var config = AgentConfiguration.Create(<EUM_APP_KEY>);
config.CrashReportingEnabled = false;
Instrumentation.InitWithConfiguration(config);
```

Programmatically Control Sessions

By default, a mobile session ends after a period of user inactivity. For example, when a user opens your application, the session begins and only ends after the user stops using the app for a set period of time. When the user begins to use the application again, a new session begins.

Instead of having a period of inactivity to define the duration of a session, however, you can use the following API to programmatically control when sessions begin and end:

```
static void AppDynamics.Agent.Instrumentation.StartNextSession()
```

When you call the method `StartNextSession`, the current session ends and a new session begins. The API enables you to define and frame your sessions so that they align more closely with business goals and expected user flows. For example, you could use the API to define a session that tracks a purchase of a product or registers a new user.

Excessive use of this API will cause sessions to be throttled (excessive use is > 10 calls per minute per Xamarin Agent, but is subject to change). When not using the API, sessions will fall back to the default of ending after a period of user inactivity.

Example of a Programmatically Controlled Session

In the example below, the current session ends and a new one begins when an item is bought.

```
using AppDynamics.Agent;
...
public async Task BuySaleItemAsync(SaleItem item)
{
    try
    {
        bool buySucceeded = await this.MobileService.InvokeApiAsync<SaleItem, bool>("buy", item);
        if (buySucceeded)
        {
            await UserDialogs.Instance.AlertAsync("Thanks for buying this item");
            Instrumentation.StartNextSession();
        }
    }
    catch (Exception e)
    {
        Debug.WriteLine(@"Unexpected error {0}", e.Message);
    }
}
```

Start and End Session Frames

You can use the `ISessionFrame` API to create session frames that will appear in the session activity. Session frames provide context for what the user is doing during a session. With the API, you can improve the names of user screens and chronicle user flows within a business context.

Use Cases

The following are common use cases for the `ISessionFrame` API:

- One screen performs multiple functions and you want more granular tracking of the individual functions.
- A user flow spans multiple screens or user interactions. For example, you could use the API to create the session frames "Login", "Product Selection", and "Purchase" to chronicle the user flow for purchases.
- You want to capture dynamic information based on user interactions to name session frames, such as an order ID.

ISessionFrame API

The table below lists the two methods and one property you can use with session frames. In short, you start a session frame with `StartSessionFrame` an d then use the returned `ISessionFrame` object to rename and end the session frame.

Class	Method/Property	Description
`Instrument`	Method: `static ISessionFrame StartSessionFrame(string sessionFrameName)`	Use this to start and name your session frame. Naming session frames enables you to easily identify
`ISessionFra me`	Property: `string Name`	Rename the session frame name. You assign the updated session frame name with this `sionFrame`.
`ISessionFra me`	Method: `static void End()`	End the session frame. You call this method from the `ISessionFrame` objec

Session Frame Example

In the following example, the `ISessionFrame` API is used to track user activity during the checkout process.

```
using AppDynamics.Agent;
...

namespace ShoppingApp {
    public partial class ShoppingCart : ContentPage {
        private ISessionFrame sessionFrame;
        private string orderId;
        ...
        void checkoutCartButtonClicked(object sender, EventArgs e) {
            // The checkout starts when the user clicks the checkout button.
            // This may be after they have updated quantities of items in their cart, etc.
            sessionFrame = Instrumentation.StartSessionFrame("Checkout");
        }
        void confirmOrderButtonClicked(object sender, EventArgs e) {
            // Once they have confirmed payment info and shipping information, and they
            // are clicking the "Confirm" button to start the backend process of checking out,
            // we may know more information about the order itself, such as an order ID.
            sessionFrame.Name = $"Checkout: Order ID {this.orderId}";
        }
        void processOrderCompleted(object sender, EventArgs e) {
            // Once the order is processed, the user is done "checking out" so we end
            // the session frame.
            sessionFrame.End();
        }
        void checkoutCancelled(object sender, EventArgs e) {
            // If they cancel or go back, you'll want to end the session frame also, or else
            // it will be left open and appear to have never ended.
            sessionFrame.End();
        }
    }
}
```

Add User Data

You can set a key/value pair of strings to record important events or information. Below is the method signature for setting user data:

```
static void AppDynamics.Agent.Instrumentation.SetUserData(string key, string value)
```

For example, you might want to log the user ID when the method for logging in the user is called:

```
using AppDynamics.Agent;
...
void LogInUser(UserCredentials) {
   // Log in user
   ...
   // Set user data with the user name.
   Instrumentation.SetUserData("user_id", UserCredentials.ID);
}
```

This information is available in **Network Request Analyze** and is added to any crash snapshots that may be taken. Keys and values are limited to 2048 characters each.

You can also set user data with values of other types (long, boolean, double, DateTime) using the following methods:

- SetUserDataBoolean
- SetUserDataDate
- SetUserDataDouble
- SetUserDataLong

To remove user data, use the following methods:

- RemoveUserData
- RemoveUserDataBoolean
- RemoveUserDataDate
- RemoveUserDataDouble

- RemoveUserDataLong

Set the Logging Level (Optional)

You can set the logging level with the configuration `LoggingLevel` as part of the class AgentConfiguration.

You can set `LoggingLevel` to one of the levels listed in the table below.

Level	Description
`Off`	Agent will emit no messages at all.
`Error`	Only show errors and initial banner. This is the default.
`Warn`	Warning level messages and above.
`Info`	Information level messages and above May be useful to the developer.
`Debug`	Debug level messages and above. Useful for support personnel and developers.
`Verbose`	Verbose level messages and above. Use verbose logging only for troubleshooting. Be sure to disable for production.
`All`	All messages.

For example:

```
var config = AppDynamics.Agent.AgentConfiguration.Create("<#Your App Key#>");
config.LoggingLevel = AppDynamics.Agent.LoggingLevel.All;
AppDynamics.Agent.Instrumentation.InitWithConfiguration(config);
```

Transform URLs for Network Requests

Implement the Network Request Callback

The callback that modifies or ignore specific URLs should be assigned to the `Func` delegate below.

> The callback method `OnNetworkRequest` is synchronous, so we recommend that you return from the function quickly.

```
public Func<IHttpRequestTracker, bool> OnNetworkRequest { get; set; }
```

Transforming URLs

To transform URLs, the `OnNetworkRequest` method should:

1. Identify specific URLs using techniques such as regex or pattern matching.

   > This first step is optional because you can choose to transform the URLs of all network requests.

2. Modify the URL property of the `IHttpRequestTracker` object.
3. Assign a valid URL to the `url` property. Modifying other properties of the `IHttpRequestTracker` object will be ignored.
4. Return `true`.

For example:

```
public static bool NetworkRequestCallback(IHttpRequestTracker tracker)
{
    var maskUrl = new Uri("http://networkrequest-mask.com/");
    tracker.Uri = maskUrl;
    return true;
}
```

Transforming Sensitive URLs

You may want to identify and transform URLs that contain sensitive information.

For example:

```
public static bool NetworkRequestCallback(IHttpRequestTracker tracker)
{
    var urlString = tracker.Uri.ToString();
    if (urlString.Contains("accountInfo"))
    {
        tracker.Uri = new Uri("http://customer-account.com/");
    }
    return true;
}
```

Ignoring URLs

If the `onNetworkRequest` method returns `false`, the beacon is dropped. Generally, the process for ignoring beacons is:

1. Identify specific URLs using techniques such as regex or pattern matching.
2. Return `false`.

To ignore specific URLs, you would identify network requests that you didn't want to monitor and return `false` to ignore the network request.

For example:

```
public static bool NetworkRequestCallback(IHttpRequestTracker tracker)
{
    if (tracker.Uri.ToString().Contains("avatar"))
    {
        //ignore calls for avatars
        return false;
    }
    return true;
}
```

To ignore all network requests, implement the following:

```
public static bool NetworkRequestCallback(IHttpRequestTracker tracker)
{
    return false;
}
```

Register a Network Request Callback

To register a network request callback:

1. Define your own method to handle the callback:

   ```
   public static bool NetworkRequestCallback(IHttpRequestTracker tracker)
   {
       return true;
   }
   ```

2. Pass it during the `AgentConfiguration` initialization phase:

```
IAgentConfiguration config = AgentConfiguration.Create(appKey);
config.OnNetworkRequest += NetworkRequestCallback; //Register the Callback
Instrumentation.InitWithConfiguration(config);
```

3. Or you can use an anonymous function:

```
IAgentConfiguration config = AgentConfiguration.Create(appKey);
config.OnNetworkRequest += (IHttpRequestTracker tracker) =>
{
    return true;
};
Instrumentation.InitWithConfiguration(config);
```

Configure and Take Screenshots

Mobile screenshots are enabled by default, but you can configure the Controller UI to automatically take screenshots or use the Xamarin SDK to manually take a screenshot:

```
Instrumentation.TakeScreenshot();Disable Screenshots
```

The `takeScreenshot()` function limits screenshots to a maximum of 1 screenshot per 10 seconds.

Disable Screenshots

You can disable screenshots from the Controller UI or with the Xamarin SDK. To disable screenshots with the Xamarin SDK, set the property `ScreenshotsEnabled` of the `IAgentConfiguration` object to `false`:

```
IAgentConfiguration config = AgentConfiguration.Create(appKey); config.ScreenshotsEnabled = false;
Instrumentation.InitWithConfiguration(config);
```

Block and Unblock Screenshots

You can also use the Xamarin SDK to block screenshots from being taken during the execution of a code block. This just temporarily blocks screenshots from being taken until you unblock screenshots. This enables you to stop taking screenshots in situations where users are entering personal data, such as on login and account screens.

The Instrumentation class provides the methods `blockScreenshots()` and `unblockScreenshots()` to block and unblock screenshots.

If screenshots are disabled through the property `ScreenshotsEnabled` of the `IAgentConfiguration` object or through the Controller UI, these methods have no effect. You can also check the `Instrumentation.ScreenshotsBlocked` property to check if screenshots are being blocked.

```
public void LoginUser()
{
        if (!Instrumentation.ScreenshotsBlocked)
        {
                Instrumentation.BlockScreenshots();
        }
        LoginCredentials credentials = UserLogin.GetCredentials();


        if (credentials.Authorized)
        {
                RedirectToProfile(credentials.user);
                Instrumentation.UnblockScreenshots();
        }
}
```

Xamarin SDK Documentation

For the complete SDK API documentation, see the latest Xamarin SDK documentation or the previous versions listed below:

- https://sdkdocs.appdynamics.com/xamarin-sdk/21.5/html/
- https://sdkdocs.appdynamics.com/xamarin-sdk/21.2/html/
- https://sdkdocs.appdynamics.com/xamarin-sdk/20.11/html/
- https://sdkdocs.appdynamics.com/xamarin-sdk/20.10/html/
- https://sdkdocs.appdynamics.com/xamarin-sdk/20.4/html/
- https://sdkdocs.appdynamics.com/xamarin-sdk/20.3/html/

Instrument Cordova Applications

Related pages:

- Hybrid Application Support
- Customize the Cordova Instrumentation
- Update Cordova Agent Configurations

Follow the steps below to get your EUM App Key and instrument your Cordova-based apps.

For native mobile applications, follow either the iOS Instrumentation or Android Instrumentation instructions. The mobile agents will inject the JavaScript Agent and auto-instrument your mobile application including the auto-instrumentation of the WKWebView.

Check Requirements

To use the AppDynamics Cordova plugin, your hybrid application must be using the following:

- Cordova >= 8.0.0
- npm >= 3.10.10
- Node.js >= 6.11.3
- WKWebView for iOS applications

WkWebView Requirements

By default, the Cordova iOS platform command adds a version of `cordova-ios` that supports UIWebView. AppDynamics does not support UIWebView, so you must manually change this to WkWebView.

1. Run the following command:

   ```
   (ionic) cordova platform add ios
   ```

2. Verify the log output returns:

   ```
   Using WkWebView
   ```

3. If the log output returns `Using UIWebView`, you must manually change it to `Using UIWebView`.

Get Your Application Key

After you completed the **Getting Started Wizard**, you were given an EUM App Key. You will need this key when you modify the source code. In some cases, multiple mobile applications can share the same key.

If you have completed the **Getting Started Wizard**, but don't have your EUM App Key, see Get Your Application Key.

Install and Initialize the Cordova Plugin

The AppDynamics Cordova plugin is certified to work with the Ionic and PhoneGap frameworks. Thus, the installation instructions below are for Ionic and PhoneGap applications. See the plugin installation instructions for your Cordova-based framework.

Customize the Cordova Application Instrumentation

See Customize the Cordova Application showing how to use the Cordova SDK through examples and brief explanations.

Update Cordova Agent Configurations

To update the configurations set when you first added the Cordova plugin to your application, the suggested practice is to remove the values for configuration values and then add new configuration values.

See the following examples:

Change the App Key

To change the app key for your Ionic or PhoneGap applications:

1. Remove the old value for the variable `APP_KEY` with the `remove` command:

```
cordova plugin remove --variable APP_KEY="old-app-key" appd-plugin-eum-mobile
```

2. Add new value for the variable `APP_KEY` with the `add` command. Be sure to include the variables that you set when you first added the plugin (represented by `<variable_list>` below). For example, if you set `REACHABILITY_HOST=http://example.com` when you first added the plugin, you will need to reset the variable when running the `add` command again.

```
cordova plugin add --variable APP_KEY="new-app-key" <variable_list> appd-plugin-eum-mobile
```

Change the App Name

To change the name of your Ionic or PhoneGap applications:

1. Remove the old value for the variable `APP_NAME` with the `remove` command:

```
cordova plugin remove --variable APP_NAME="old-app-name" appd-plugin-eum-mobile
```

2. Add new value for the variable `APP_NAME` with the `add` command. Be sure to include the variables that you set when you first added the plugin (represented by `<variable_list>` below). For example, if you set `REACHABILITY_HOST=http://example.com/` when you first added the plugin, you will need to reset the variable when running the `add` command again.

```
cordova plugin add --variable APP_NAME="new-app-name" <variable_list> appd-plugin-eum-mobile
```

Customize the Cordova Instrumentation

Related pages:

- Instrument Cordova Applications
- Cordova SDK Documentation

Once you have instrumented your Cordova-based application with the Cordova Plugin, you can also use the APIs to customize the data for your app that appears in the Controller UI. The following sections show you how to use the Cordova Plugin SDK API to customize your instrumentation:

Using the Cordova Plugin SDK API

Syntax

To call SDK API methods, use the following syntax: `window.plugins.ADEUMMobilePlugin.<method>`

Arguments

The last two arguments for all of the SDK API methods should always be two functions. The first function should handle successful cases and the last method should handle failures.

For example:

```
window.plugins.ADEUMMobilePlugin.changeAppKey("<EUM_APP_KEY>",
    (success) => {
        this.showAlert("changeAppKey return: success");
    },
    (error) => {
        this.showAlert("changeAppKey error:" + error);
    }
);
```

Add Methods to Call SDK APIs

To use the SDK APIs, you are recommended to create class methods that call the SDK APIs as shown below.

```
export class HomePage {
    ...
    someMethod(event) {
         window.plugins.ADEUMMobilePlugin.<method>(<arg1>, <success_function>, <failure_function>);
    }
    ...
}
```

Example

Thus, for the `HomePage.js` file, your `HomePage` class could have the method `takeScreenshot` that calls the SDK API method `screenshot` as shown here.

```
export class HomePage {
    ...
    takeScreenshot(event) {
      // Call the Cordova plugin SDK methods
      window.plugins.ADEUMMobilePlugin.screenshot(
       (success) => {
        this.showAlert("crash return: success");
       },
      (error) => {
        this.showAlert("crash error:" + error);
      });
    }
    ...
}
```

Change the App Key

To change the EUM application key, you use the method `changeAppKey` with the parameters below.

Parameter Name	Data Type	Description
`appKey`	string	The EUM application key.
`success`	function	A user-defined function that is called when `changeAppKey` is successful.
`error`	function	A user-defined function that is called when `changeAppKey` fails.

For example, you could create a new method that takes a new app key and passed it to the SDK API method `changeAppKey`.

```
changeAppKey(event, newAppKey) {
    window.plugins.ADEUMMobilePlugin.changeAppKey(newAppKey,
        (success) => {
            this.showAlert("changeAppKey return: success");
        },
        (error) => {
            this.showAlert("changeAppKey error:" + error);
        }
    );
}
```

Collect Additional Types of Data

You can use methods available in the `ADEUMMobilePlugin` class to collect five additional types of data:

- Info points
- Custom timers
- Custom metrics
- User data
- Breadcrumbs

Info Points

Information points allow you to track how your own code is running. You can see how often a method is invoked, and how long it takes to run by calling `beginCall`. When the callbacks `success` or `error` are called, the call to track the info point ends.

- `beginCall(name, functionName, args, success, error)`

Parameters

The table below describes the parameters for the two methods:

Name	Type	Description
`name`	string	The name of the file or module where the info point is being recorded.
`functionName`	string	The function that is invoking `beginCall` to track an info point.

`success`	function	The user-defined callback for successful cases.
`error`	function	The user-defined callback for failed cases.

Example

For example, you can use create info points with something like the code below to determine how a method is invoked, and how long it takes to run:

```
beginCall(event) {
    window.plugins.ADEUMMobilePlugin.beginCall("home.ts", "callTrackerFunction", "event",
        (tracker) => {
            tracker.reportCallEndedWithReturnValue("Return from home.ts",
                (success) => { console.log("End call with return value success:" + success);},
                (error) => { console.log("End call with return value error:" + error); });
            },
            (error) => {
                console.log("Begin call error:" + error);
        }
    );
}
```

Custom Timers

You can create custom timers to time any arbitrary sequence of events within your code, even spanning multiple methods. You create the custom timers using the SDK API methods `startTimer` and `stopTimer`.

- `startTimerWithName(name, success, error)`
- `stopTimerWithName(name, success, error)`

Parameters

The two methods take the following parameters:

Name	Type	Description
`name`	string	The name of the custom timer. Allowed characters are [A-Za-z\s0-9]. Illegal characters are replaced by their ASCII hex value.
`success`	function	The user-defined callback for successful cases.
`error`	function	The user-defined callback for failed cases.

Example

For example, to track the time a user spends viewing a screen, the instrumentation could look like this:

```
startTimer(event) {
    window.plugins.ADEUMMobilePlugin.startTimerWithName(data.name,
        (success) => {
            this.showAlert("startTimerWithName return: success");
        },
        (error) => {
            this.showAlert("startTimerWithName error:" + error);
        }
    );
}
stopTimer(event) {
    window.plugins.ADEUMMobilePlugin.stopTimerWithName(data.name,
        (success) => {
            this.showAlert("stopTimerWithName return: success");
        },
        (error) => {
            this.showAlert("stopTimerWithName error:" + error);
        }
    );
}
```

Custom Metrics

You can also report custom metrics.

You create custom metrics with the `reportMetricWithName`:

- `reportMetricWithName(name, value, success, error)`

Parameters

The `reportMetricWithName` method takes the following parameters:

Name	Type	Description
`name`	string	The name of the custom metric. The metric names must consist of alphanumeric characters. Illegal characters are replaced by their ASCII hex value.
`value`	number	if value is not a whole number an error will be returned.
`success`	function	User-defined success callback.
`error`	function	User-defined error callback.

Example

For example, the following method could be used to report custom metrics:

```
reportMetric(event, data) {
    window.plugins.ADEUMMobilePlugin.reportMetricWithName(data.name, parseInt(data.value),
        (success) => {
            this.showAlert("reportMetricWithName : success");
        },
        (error) => {
            this.showAlert("reportMetricWithName error:" + error);
        }
    );
};
```

Breadcrumbs

Breadcrumbs allow you to situate a crash in the context of your user's experience. Set a breadcrumb when something interesting happens. If your application crashes at some point in the future, the breadcrumb will be displayed along with the crash report. Each crash report displays the most recent 99 breadcrumbs.

You create and leave breadcrumbs with the following SDK API method:

- `leaveBreadcrumb(breadcrumb, mode, success, error)`

Parameters

The method `leaveBreadcrumb` takes the following parameters:

Name	Type	Description
`breadcrumb`	string	The string to include in the crash report and sessions. Truncated at 2048 characters; empty values are ignored.
`mode`	number	The mode determining where the breadcrumb will be displayed: • 0 - for crashes only • 1 - for crashes and sessions. The mode defaults to crashes if the value is not parseable.
`success`	function	The user-defined callback for successful cases.
`error`	function	The user-defined callback for failed cases.

Example

This code example shows how to use the SDK API to leave a breadcrumb:

```
breadcrumb(mode) {
      window.plugins.ADEUMMobilePlugin.leaveBreadcrumb( "breadcrumb1", mode,
     (success) => {
        this.showAlert("leaveBreadcrumb return: success");
      },
      (error) => {
        this.showAlert("leaveBreadcrumb error:" + error);
    }
)
```

Add Custom User Data

You can set and later remove any string key/value pair you think might be useful with the following methods:

- `setUserData(key, value, success, error)`
- `removeUserData(key, success, error)`

Parameters

The following table describes the parameters:

Name	Type	Description
`key`	string	The key identifying the key-value pair.
`value`	string	The value associated with the key.
`success`	function	The user-defined callback for successful cases.
`error`	function	The user-defined callback for failed cases.

Example

The code example below shows how to set and remove user data with the SDK API:

```
setCustomData(event, data) {
    window.plugins.ADEUMMobilePlugin.setUserData(data.name, data.value,
        (success) => {
            this.showAlert("setUserData return: success");
        },
        (error) => {
            this.showAlert("setUserData error:" + error);
        }
    );
};
removeUserData(event, key) {
    window.plugins.ADEUMMobilePlugin.removeUserData(key,
        (success) => {
            this.showAlert("removeUserData return: success");
        },
        (error) => {
            this.showAlert("removeUserData error:" + error);
        }
    );
}
```

Programmatically Control Sessions

By default, a mobile session ends after a period of user inactivity. For example, when a user opens your application, the session begins and only ends after the user stops using the app for a set period of time. When the user begins to use the application again, a new session begins.

Instead of having a period of inactivity to define the duration of a session, however, you can use the following API to programmatically control when sessions begin and end.

```
startNextSession(success, error)
```

When you call the method `startNextSession` from `ADEUMMobilePlugin`, the current session ends and a new session begins. The API enables you to define and frame your sessions so that they align more closely with business goals and expected user flows. For example, you could use the API to define a session that tracks a purchase of a product or registers a new user.

Excessive use of this API will cause sessions to be throttled (excessive use is >10 calls per minute, but is subject to change). When not using the API, sessions will fall back to the default of ending after a period of user inactivity.

Example of a Programmatically Controlled Session

In the example below, the current session ends and a new one begins when the checkout is made.

```
completeCheckout() {
    if (this.validateAddress() && this.validatePayment()){
        let loader = this.loader.create({
            content: "Completing the checkout..."
        });
        loader.present();
        return this.order.create({
          shipping_address: this.address,
          billing_address: this.address,
          cart_id: Number(this.configuration.get('cart_id'))
        })
        ...
        window.plugins.ADEUMMobilePlugin.startNextSession(
           (success) => {
                          console.log("startNextSession return: success");
                        },
           (error) => {
                          console.log("startNextSession error:" + error);
                      }
       );
}
```

Start and End Session Frames

You can use Cordova Plugin to create session frames that will appear in the session activity. Session frames provide context for what the user is doing during a session. With the API, you can improve the names of user screens and chronicle user flows within a business context.

Use Cases

The following are common using session frames:

- One page performs multiple functions and you want more granular tracking of the individual functions.
- A user flow spans multiple pages or user interactions. For example, you could use the API to create the session frames "Login", "Product Selection", and "Purchase" to chronicle the user flow for purchases.
- You want to capture dynamic information based on user interactions to name session frames, such as an order ID.

SessionFrame API

The table below lists the three methods you can use with session frames. In short, you start a session frame with `startSessionFrame` and then use `updateName` and `end` to rename and end the session frame.

Method	Parameters	Description
`startSessionFrame(name, success, error)`	• `name` (string) - The name of the session frame. • `success` (function) - The user-defined success callback. • `error` (function) - The user-defined error callback.	Use this to start and name your session frame. You call this method from `window.plugins.ADEUMMobilePlugin`.

`updateName(name, success, error)`	• `name` (string) - The name of the session frame. • `success` (function) - The user-defined success callback. • `error` (function) - The user-defined error callback.	Rename the session frame name. You call this method from the `ADEUMMobilePlugin` object.
`end(success, error)`	• `success` (function) - The user-defined success callback. • `error` (function) - The user-defined error callback.	End the session frame. You call this method from the `ADEUMMobilePlugin` object.

Session Frame Example

In the following example, session frames are used to track user activity during the checkout process.

```
declare var window: any;
@Component({
    ...
})
export class OrderPage {
    sessionFrame: any;
    ...

    checkoutCartButtonClicked() {
        // The user starting to check out starts when the user clicks the checkout button
        // this may be after they have updated quantities of items in their cart, etc.
        window.plugins.ADEUMMobilePlugin.startSessionFrame("Checkout",
            (sessionFrame) => {
                // The returned object is saved to the class property 'sessionFrame' so
                // the SessionFrame API methods 'updateName' and 'end' can be called from it later.
                this.sessionFrame = sessionFrame;
            },
            (error) => {
                console.log("startSessionFrame call error:" + error);
             }
        );
    }
    confirmOrderButtonClicked() {
        // Once they have confirmed payment info and shipping information, and they
        // are clicking the "Confirm" button to start the backend process of checking out
        // we may know more information about the order itself, such as an Order ID.
        this.sessionFrame.updateName("Checkout: Order ID " + this.orderId,
            (success) => {
                console.log("Order has been placed and sessionFrame updated:" + this.orderId);
            },
            (error) => {
                console.log("Order has been placed but sessionFrame couldn't be updated because of the error "
+ error);
            }
        );
    }
    processOrderCompleted() {
        // Once the order is processed, the user is done "checking out" so we end
        // the session frame.
        this.sessionFrame.end(
            (success) => {
                console.log("Order was completed and sessionFrame ended: " + success);
            },
            (error) => {
                console.log("Order was completed but sessionFrame couldn't be ended because of: " + error);
            }
        );
    }
    checkoutCancelled() {
        // If they cancel or go back, you'll want to end the session frame also, or else
        // it will be left open and appear to have never ended.
        this.sessionFrame.end(
            (success) => {
                console.log("Order was cancelled and sessionFrame ended: " + success);
            },
            (error) => {
                console.log("Order was cancelled but sessionFrame couldn't be ended because of: " + error);
            }
        );
    }
} // end of export class OrderPage
```

Configure and Take Screenshots

Mobile screenshots are enabled by default. These screenshots will show up in the **Sessions Details** dialog.

You can configure the Controller UI to automatically take screenshots or use the Cordova plugin to manually take a screenshot as shown below:

```
screenshot(event) {
     // make a call to plugin
     console.log("screenshot click handler");
      window.plugins.ADEUMMobilePlugin.takeScreenshot(

     (success) => {
        this.showAlert("screenshot return: success");
      },
      (error) => {
        this.showAlert("screenshot error:" + error);
      });
}
```

This will capture everything, including personal information, so you must be cautious of when to take the screenshot.

Block and Unblock Screenshots

You can also block screenshots from being taken during the execution of a code block. This just temporarily blocks screenshots from being taken until you unblock screenshots. This enables you to stop taking screenshots in situations where users are entering personal data, such as on login and account screens.

```
logInScreen(event) {
    // Block Screen while getting user input
    window.plugins.ADEUMMobilePlugin.blockScreenshots(
        (success) => {
            console.log("blockScreenshots return: success");
            this.getUserInput();
            this.submitUserInput();
        },
        (error) => {
            console.log("blockScreenshots error:" + error);
         }
    );
    // Unblock screen and return to home page
    window.plugins.ADEUMMobilePlugin.unblockScreenshots(
        (success) => {
            console.log("unblockScreenshots return: success");
            this.homePage()
        },
        (error) => {
            console.log("unblockScreenshots error:" + error);
        }
    );
}
```

Enable Logging and Set Logging Level

You use the method `loggingLevel` to enable and set the logging level. You can set logging to one of the following levels:

Value	Logging Level	Description
0	None	No logs are displayed. **This level disables logging.**
1	Error	Only error messages are displayed. **This is the default logging level.**
2	Warn	Warning and error messages are displayed.
3	Info	Warning, errors, and developer-focused messages are displayed.
4	Debug	Errors, warnings, developer information, and debugging messages are displayed.
5	Verbose	Errors, warnings, developer information, debugging, and troubleshooting messages are displayed.

6	All	All the supported log messages are displayed.

You enable logging by setting the logging level in the instrumentation configuration. For example, in this example, you are enabling logging and setting the logging level to `Info`:

```
window.plugins.ADEUMMobilePlugin.initWithConfiguration(
    {
        "appKey": "<EUM_APP_KEY>",
        "loggingLevel": 3
     },
     (success) => {
        this.showAlert("initWithConfiguration return: success");
     },
     (error) => {
        this.showAlert("initWithConfiguration error:" + error);
     }
);
```

Cordova SDK Documentation

For the Cordova SDK API reference documentation, see the latest JSDoc documentation or the previous versions listed below:

- https://sdkdocs.appdynamics.com/cordova-plugin/21.5/
- https://sdkdocs.appdynamics.com/cordova-plugin/21.2/
- https://sdkdocs.appdynamics.com/cordova-plugin/20.11/
- https://sdkdocs.appdynamics.com/cordova-plugin/20.10/
- https://sdkdocs.appdynamics.com/cordova-plugin/20.7/

Instrument React Native Applications

Related pages:

- Hybrid Application Support
- Manually install the React Native Package
- Point to an On-Premises EUM Server (Optional)
- Customize the React Native Instrumentation

Follow these steps to get your EUM App Key and instrument your React Native application.

Requirements

- React Native >=0.60

  > The React Native Agent has been tested and certified to work with React Native >=0.60. React Native is in rapid development; thereby any minor update to React Native could potentially affect the functionality of the React Native Agent.

- (iOS only): Pod target and Xcode deployment target must both be >=11. Find the pod target version in the `podfile` located in `<your_react_native_app>/ios/`.

Get Your Application Key

From the Controller's **Getting Started Wizard**, create either an iOS or an Android application. You will be given an EUM App Key that you will use to instrument your application. In some cases, multiple mobile applications can share the same key.

If you have completed the **Getting Started Wizard**, but don't have your EUM App Key, see Get Your Application Key.

Eject Your React Native Application

If you created your React Native application with one of the commands in the following table, you will need to eject the application:

Command	Why Ejection Is Needed
`create-react-native-app`	This command enables you to quickly set up and run a Native React application with no configuration but does *not* provide the full development environment required by the React Native Agent.
`expo init`	Expo projects do not currently support the React Native Agent module.

If you created your app with `create-react-native-app`, use `npm` to eject the app. If you used `expo`, run the `expo` command below to eject the app.

From your project directory, run the command appropriate for your use case:

Install the React Native Agent

This module injects the native AppDynamics agents into your application and offers a JavaScript bridge to the `Instrumentation` management interface.

1. Install the React Native Agent:

   ```
   npm install --save @appdynamics/react-native-agent
   ```

2. Build the configuration to enable the build-time instrumentation of your application:

   ```
   node node_modules/@appdynamics/react-native-agent/bin/cli.js install
   ```

> The React Native Agent CLI assumes you've kept the default project structure created with the `react-native` CLI tools.

3. If the above commands failed, see manually install the React Native package. Otherwise, proceed to Add Required Libraries (iOS Only) and/or Add Permissions (Android Only).

Manually Install the React Native Package (Only If Required)

Most users do not have to manually install the package, but the command above may fail if you heavily modified your iOS and Android projects.

Follow these steps to manually install the React Native package:

Add Permissions (Android Only)

Add the following permissions to your app's `AndroidManifest.xml` (which should be located under `android/app/src/main/`).

```
  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.your.application">

    // Add the following:
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>

    // ...
</manifest>
```

> The React Native Agent tries to minimize its network footprint. Without this permission, the agent always assumes poor network conditions and some metrics may not get reported.

Instrument Your Application Code

In the `index.js` file of your React application:

1. At the top of the file near the other `import` statements, add the following line of code to import the Native React Agent:

   ```
   import { Instrumentation } from '@appdynamics/react-native-agent';
   ```

2. In the same file and in the global scope, initialize the instrumentation with the following, making sure to replace <EUM_APP_KEY> with a string containing your EUM App Key.

   ```
   Instrumentation.start({
       appKey: <EUM_APP_KEY>,
   });
   ```

3. Verify that your `index.js` file looks similar to the following:

```
import {AppRegistry} from 'react-native';
import App from './App';
// === Add the following to import the React Native Agent
import { Instrumentation } from '@appdynamics/react-native-agent';

// Initialize the instrumentation
Instrumentation.start({
    appKey: 'YOUR-APP-KEY',
});

// Create a component
const App = () => (
  // Other components
);

// Register the App component
AppRegistry.registerComponent('my-application', () => App);
```

4. (For older React Native setup builds) If you have `use_frameworks!` defined in your `ios/Podfile`, we recommend adding the following:

```
$ toggleADEUMRNStaticFramework = true
```

Build and Run Your React Native Applications (Optional)

1. Run the command to build and run the app for your platform.
2. From the Controller UI, verify that the instrumentation was successful.

Point to an On-Premises EUM Server (Optional)

To use an on-premises EUM Server, pass the URL to the on-premises EUM Server when you initialize the instrumentation with the EUM App Key from Get Your Application Key.

```
Instrumentation.start({
     appKey: <EUM_APP_KEY>,
     collectorURL: <COLLECTOR_URL>
});
```

Customize the React Native Instrumentation

The sections below describe how to use the React Native APIs to customize your instrumentation.

Using the React Native Agent APIs

After you have installed and initialized the React Native Agent, you instrument your application code by importing the class, interface, or enumeration and then calling API methods.

Import Syntax

To access the React Native Agent APIs, use an import statement at the top of the file. The following line imports the class `Instrumentation`:

```
import { Instrumentation } from '@appdynamics/react-native-agent';
```

API Method Syntax

Once the class, interface, or enumeration has been imported, call the API methods from the imported component. In this example, you set custom user data with `setUserDataBoolean` from the `Instrumentation` class.

```
Instrumentation.setUserDataBoolean("Boolean Key", true);
```

Change the App Key

You can use the Native React Agent API to dynamically change the EUM app key. You receive the EUM App Key when creating a mobile app in the Controller UI. See Set Up and Access Mobile RUM for information about getting the EUM App Key.

Class

The API to change app keys is available through the `Instrumentation` class.

```
import { Instrumentation } from '@appdynamics/react-native-agent';
```

Methods

The API consists of the following method:

- `changeAppKey(appKey)`

Method Parameters

The `changeAppKey` method takes the following parameter:

Parameter Name	Data Type	Description
`appKey`	string	The EUM App Key.

Examples

In this example, you create a method that takes a new app key and passes it to the API method `changeAppKey`.

```
import { Instrumentation } from '@appdynamics/react-native-agent';
...
private updateAppKey(newAppKey) {
    Instrumentation.changeAppKey(newAppKey);
}
```

Collect Additional Types of Data

The React Native APIs have methods that extend the application data types you can collect and aggregate with Mobile RUM. There are six basic kinds of extensions that you can create:

Type of Data	Description	Specifications	Where Data is Displayed
Info points	How often a method is invoked, and how long it takes to run.	• Data is numeric • Names must consist of alphanumeric characters and/or spaces	• Metric Browser • Custom Data • Network Request Snapshots • Mobile Sessions • Network Request Analyze
Custom timers	Any arbitrary sequence of events within your code timed, even spanning multiple methods.	• Data is numeric • Metric names must consist of alphanumeric characters and/or spaces	• Metric Browser • Custom Data
Custom metrics	Any integer-based data you wish to collect.	• Data is numeric • Metric names must consist of alphanumeric characters and/or spaces	• Metric Browser • Custom Data
User data	Any string key/value pair you think might be useful.	• Data can be any type • Metric names have no restrictions	• Network Request Snapshots • Mobile Sessions • Network Request Analyze • Crash Snapshots
Breadcrumbs	The context for a crash.	• Data can be any data type • Metric names have no restrictions	• Network Request Snapshots • Mobile Sessions • Network Request Analyze • Crash Snapshots
User interactions	Capture when users press buttons, click on lists, and select text.	• Data can be any data type • Metric names have no restrictions	• Network Request Snapshots • Mobile Sessions • Network Request Analyze

Report Info Points

Information points allow you to track how your own code is running. Unlike the other React Native APIs that require importing a module and using a method from that module, you add annotations to your code to report info points.

Class/Interface

The API is available through the `InfoPoint` function.

```
import { InfoPoint } from '@appdynamics/react-native-agent';
```

Methods

With the `@InfoPoint` annotation, you only need to provide a function to report info points. The `@InfoPoint` annotation has two signatures:

- `@InfoPoint`
- `@InfoPoint({className: string, fnName: string})`

Method Signatures

Methods	Parameters	Description
`@InfoPoint`	None	Automatically reports the info points of the method that is being annotated.
`@InfoPoint({className: string, fnName: string})`	Object • `className` - The name of the file or module where the info point is recorded. • `fnName` - The function that invokes `beginCall` to track an info point.	Manually specify the class and function names to annotate. For example: `@InfoPoint({ className: 'Automobiles', fnName: 'checkInventory' })`

Examples

The following examples show you how to use automatic and manual reporting of info points.

We recommended you use automatic reporting if your code will not be minified: it's simpler and reports the same information. If your code will be minified, however, you should manually report the class and function names to avoid reporting the useless class and function names created by minification.

Automatic Reporting

Add the annotation `@InfoPoint` to automatically create an info point for the method below it. Info point reports how the method is invoked and how long it takes to run.

```
import { InfoPoint } from '@appdynamics/react-native-agent';

class InfoPointsScreen extends Component {

  @InfoPoint
  public infoPointMethod(arg1, arg2, value) {
    console.log("Executing infoPointMethod!");
  }
}
```

Manually Reporting

To create an info point for a specific method in a class (the symbol names), pass an object to the `@InfoPoint` annotation that specifies the class and method name as shown below.

```
import { InfoPoint } from '@appdynamics/react-native-agent';

class InfoPointsScreen extends Component {

  @InfoPoint({ className: 'MyClass', fnName: 'infoPointMethod' })
  public infoPointMethod(arg1, arg2, value) {
    console.log("Executing infoPointMethod!");
  }
}
```

Set Custom Timers

You create custom timers to time any arbitrary sequence of events within your code, even spanning multiple methods.

Class/Interface

The Custom Timers API is accessible through the `Instrumentation` class:

```
import { Instrumentation } from '@appdynamics/react-native-agent';
```

Methods

You create the custom timers using the following React Native Agent API methods:

- `startTimerWithName(name)`
- `stopTimerWithName(name)`

Method Parameters

Both methods take the following parameter:

Name	Type	Description
`name`	string	The name of the custom timer. Allowed characters are [A-Za-z\s0-9]. Illegal characters are replaced by their ASCII hex value.

Examples

For example, to track the time a user spends viewing a screen, the instrumentation could look like this:

```
import { Instrumentation } from '@appdynamics/react-native-agent';
...
private startCustomTimer() {
  Instrumentation.startTimer("My timer");
}
private stopCustomTimer() {
   Instrumentation.stopTimer("My timer");
}
```

Create Custom Metrics

The React Native API enables you to report custom metrics. You specify the name of the custom metric that will appear in the Controller UI.

Class

The Custom Metrics API is accessible through the `Instrumentation` class:

```
import { Instrumentation } from '@appdynamics/react-native-agent';
```

Methods

You create custom metrics with the `reportMetric`:

- `reportMetric(name, value)`

Method Parameters

The `reportMetric` method takes the following parameters:

Name	Type	Requirements	Description
`name`	string	The metric names must consist of alphanumeric characters. Illegal characters are replaced by their ASCII hex value.	The name of the custom metric.
`value`	number	The value must be a whole number, otherwise, an error will be returned.	The numeric value associated with the custom metric.

Examples

For example, the following method could be used to report custom metrics:

```
import { Instrumentation } from '@appdynamics/react-native-agent';
...
private reportMetrics() {
    Instrumentation.reportMetric("Normal metric", 23);
    Instrumentation.reportMetric("Large metric", Number.MAX_SAFE_INTEGER + 1);
    Instrumentation.reportMetric("Small metric", Number.MIN_SAFE_INTEGER - 1);
}
```

Add Custom User Data

Use the Custom User Data API to set and remove key/value pairs of different data types. The key must be unique across your application.

Class

The Custom User Data API is accessible through the `Instrumentation` class:

```
import { Instrumentation } from '@appdynamics/react-native-agent';
```

Methods

The API provides the following methods for setting and removing custom user data:

Methods for Setting Custom User Data

- `setUserData(key, value)`
- `setUserDataBoolean(key, value)`
- `setUserDataDate(key, value)`
- `setUserDataDouble(key, value)`
- `setUserDataInteger(key, value)`

Methods for Removing Custom User Data

- `removeUserData(key)`
- `setUserDataBoolean(key)`
- `setUserDataDate(key)`
- `setUserDataDouble(key)`
- `setUserDataInteger(key)`

Method Parameters

The following table lists the parameters for the methods for setting custom user data.

Methods for Setting Custom User Data	Parameters	Data Type
`setUserData`	key	string
	value	string || null
`setUserDataBoolean`	key	string
	value	boolean
`setUserDataDate`	key	string
	value	Date object
`setUserDataDouble`	key	string
	value	number
`setUserDataInteger`	key	string
	value	number

The following table lists the parameters for the methods for removing custom user data.

Methods for Removing Custom User Data	Parameters	Data Type

`removeUserData`	key	string
`removeUserDataBoolean`		
`removeUserDataDate`		
`removeUserDataDouble`		
`removeUserDataInteger`		

The method `removeUserData` is called when setting custom data to the value `null`. For example, the method `Instrumentation.setUserData("name", null)` will call `Instrumentation.removeUserData("name")`.

Examples

The following code example shows how to set and remove user data with the Custom User Data API:

```
import { Instrumentation } from '@appdynamics/react-native-agent';
...
private setUserData() {
    Instrumentation.setUserData("userId", "AISJ1723871");
    Instrumentation.setUserDataBoolean("isVip", true);
    Instrumentation.setUserDataDate("purchaseDate", new Date(1234567890));
    Instrumentation.setUserDataDouble("monthlyVisits", 1.2345);
    Instrumentation.setUserDataInteger("totalPurchasedItems", 42);
}
private clearUserData() {
    Instrumentation.removeUserData("userId", null);
    Instrumentation.removeUserDataBoolean("isVip", null);
    Instrumentation.removeUserDataDate("purchaseDate", null);
    Instrumentation.removeUserDataDouble("monthlyVisits", null);
    Instrumentation.removeUserDataInteger("totalPurchasedItems", null);
}
```

Leave Breadcrumbs

Breadcrumbs enable you to situate a crash in the context of your user's experience. Set a breadcrumb when something interesting happens. If your application crashes at some point in the future, the breadcrumb will be displayed along with the crash report. Each crash report displays the most recent 99 breadcrumbs.

Class

The Breadcrumb API is accessible through the `Instrumentation` class:

```
import { Instrumentation } from '@appdynamics/react-native-agent';
```

Methods

You create and leave breadcrumbs with the following API method:

- `leaveBreadcrumb(breadcrumb, mode)`

Method Parameters

The method `leaveBreadcrumb` takes the following parameters:

Name	Type	Description
`breadcrumb`	string	The string to include in the crash report and sessions. Truncated at 2048 characters; empty values are ignored.

`mode`	Enumeration	The mode determining where the breadcrumb will be displayed. You can report crashes only or crashes and sessions with the following modes: • `BreadcrumbVisibility.CRASHES_ONLY` • `BreadcrumbVisibility.CRASHES_AND_SESSIONS` The default mode is `BreadcrumbVisibility.CRASHES_ONLY`. If the value is invalid, the default mode will be applied.

Examples

Basic Usage

The following example shows the syntax and usage of the Breadcrumb API.

```
import { Instrumentation } from '@appdynamics/react-native-agent';
...
private leaveBreadcrumb() {
    Instrumentation.leaveBreadcrumb("Drop a breadcrumb button implementation", BreadcrumbVisibility.
CRASHES_AND_SESSIONS);
}
```

Extended Use Case

Supposed your React Native app has a wizard that shows different steps of a process sequentially on the same screen. In the case of a crash or an "Application Not Responding" (ANR) error, you would want to know what step of the wizard caused the crash or ANR.

If your app had a wizard engine similar to the one below, you could leave a breadcrumb for each screen to track events and issues.

```
async function wizard(...screens) {
    let currentScreen = 0;
    while (true) {
        const screen = screens[currentScreen];
        if (screen == null) return;
        // Report the current screen with AppDynamics instrumentation
        Instrumentation.leaveBreadcrumb('wizard screen ' + screen.name);
        currentScreen += await screen.action();
    }
}
```

The wizard engine could be used for a checkout:

```
wizard(
    { name: 'review cart', action:reviewCartAction },
    { name: 'chose payment', action:chosePaymentAction },
    { name: 'chose address', action:choseAddressAction },
    { name: 'review order', action:reviewOrderAction },
    { name: 'checkout', action:checkoutAction }
)
```

If you receive an alert that your app is triggering a lot of ANRs, you can diagnose the problem through the mobile sessions where the ANRs occur. In the session, you will see the breadcrumbs associated with the ANRs. For example, you may discover that the breadcrumbs "review cart" and "chose payment" are associated with ANRs, but "chose address" is not. Further investigation of the payment screen could lead to the discovery that the process that is encrypting credit card numbers is running on the main thread and freezing the app.

Capture User Interaction

The React Native Agent can track certain UI events triggered by user interactions. Once user interactions have been captured, you can sort sessions by UI event and view UI events in the timeline of the session waterfall.

Supported User Interactions

You can capture when users do one or all of the following:

- press buttons
- select table cells
- select text fields

- select text views

Security and Privacy Concerns

The interaction capture mode is disabled by default for security and privacy reasons, as user interactions may contain sensitive information. Moreover, this potential security and privacy issue may be compounded if you enable both the capturing of UI interactions and screenshots.

Class

The Interaction Capture Mode API is accessible through the `InteractionCaptureMode` class, but you also need the `Instrumentation` class to start instrumentation with the configured user capture mode.

```
import { Instrumentation, InteractionCaptureMode } from '@appdynamics/react-native-agent';
```

Properties

The following properties are accessed through the `InteractionCaptureMode` class and used to configure the user interaction capture. For example: `InteractionCaptureMode.All`

Property	Description	Platform Support
`All`	Track all the user interactions.	iOS, Android
`ButtonPressed`	Track button presses.	iOS, Android
`ListViewItemsSelected`	Track "List Item" clicks and focus changes for `android.widget.AbsListView` and its subclasses.	Android
`None`	Disables the tracking of any user interactions.	iOS, Android
`TableCellSelected`	Track table cell selection.	iOS
`TextFieldSelected`	Track text field selection.	iOS, Android
`TextViewSelected`	Track text view selection.	iOS

Methods

The API methods take an array of user interaction capture modes and return an `InteractionCaptureMode` object that determines which user interactions will be captured. The methods do not mutate the object.

Method	Description
`with`	Combines the multiple user interaction capture modes.
`without`	Excludes the specified user interaction modes from being captured.

Examples

Set User Interaction Mode with API Properties

The following example configures the React Native Agent to capture all user interactions.

```
import { Instrumentation, InteractionCaptureMode } from '@appdynamics/react-native-agent';
...
Instrumentation.start({
     appKey: <EUM_APP_KEY>,
     // ...your other configurations

     // Capture all user interactions
     interactionCaptureMode: InteractionCaptureMode.All
 })
```

Combine User Interaction Modes

In the following example, the `with` method combines the `ButtonPressed` (button presses) and `TextFieldSelected` (text field selected) modes with the `None` mode, effectively just enabling those two user interaction modes.

```
import { Instrumentation, InteractionCaptureMode } from '@appdynamics/react-native-agent';
...
Instrumentation.start({
    appKey: <EUM_APP_KEY>,
    // ...your other configurations

    // Only enable "ButtonPressed" and "TextFieldSelected" interaction modes, and disable the rest.
    interactionCaptureMode: InteractionCaptureMode.None.with(
        InteractionCaptureMode.ButtonPressed,
        InteractionCaptureMode.TextFieldSelected
    )
})
```

Exclude User Interaction Modes

The following example excludes the user interaction modes `ButtonPressed` (button presses) and `TextFieldSelected` (text field selected) from the mode `InteractionCaptureMode.All` (all user interactions). Effectively, the React Native Agent would be configured to only capture `ListViewItemSelected` ("List Item" clicks), `TableCellSelected` (table cell selection), and `TextViewSelected` (text view selected) modes.

```
import { Instrumentation, InteractionCaptureMode } from '@appdynamics/react-native-agent';
...
Instrumentation.start({
    appKey: <EUM_APP_KEY>,
    // ...your other configurations

    // Include all capture modes except "ButtonPressed" and "TextFieldSelected".
    interactionCaptureMode: InteractionCaptureMode.All.without(
        InteractionCaptureMode.ButtonPressed,
        InteractionCaptureMode.TextFieldSelected
    )
})
```

Disable the Agent to Stop Sending User Data to the Collector

You can disable the agent to stop sending all data to the collector while the agent is initialized and running. For example, you can disable the agent if your app has an option for users to opt-out of monitoring for privacy reasons.

`shutdownAgent`

The `shutdownAgent` call stops outgoing data to the collector, and does not persist data on the device.

```
import { Instrumentation } from '@appdynamics/react-native-agent';
...
Instrumentation.shutdownAgent();
```

- The call only stops the traffic out of the agent.
- Once the agent has been initialized, the call cannot be removed, and a license will have been consumed.

`restartAgent`

To re-enable the agent and reverse `shutdownAgent`, use `restartAgent`.

```
import { Instrumentation } from '@appdynamics/react-native-agent';
...
Instrumentation.restartAgent();
```

- This call will respect the server side calls that can remotely shutdown the agent in a similar way.
- The call is only in effect while the app is running.
- The call will be ignored if the agent has been remotely disabled.
- If the call is removed from memory and the app restarts, or the device is rebooted, the agent will be initialized as normal.

Programmatically Control Sessions

By default, a mobile session ends after a period of user inactivity. For example, when a user opens your application, the session begins and only ends after the user stops using the app for a set period of time. When the user begins to use the application again, a new session begins. Instead of having a period of inactivity to define the duration of a session, however, you can use this API to programmatically control when sessions begin and end.

Class

The API is accessible through the `Instrumentation` class:

```
import { Instrumentation } from '@appdynamics/react-native-agent';
```

Methods

The API provides the following method for ending the current session and starting a new session:

Method	Description
`startNextSession()`	When you call the method `startNextSession` from `Instrumentation`, the current session ends and a new session begins. The API enables you to define and frame your sessions so that they align more closely with business goals and expected user flows. For example, you could use the API to define a session that tracks a purchase of a product or registers a new user.

Excessive use of this API will cause sessions to be throttled (excessive use is >10 calls per minute, but is subject to change). When not using the API, sessions will fall back to the default of ending after a period of user inactivity.

Examples

In the following example, the current session ends and a new one begins when the check out is made.

```
import { Instrumentation } from '@appdynamics/react-native-agent';
...
private checkoutCart(){
    if (currentCartItems!=null && currentCartItems.size()>0){
            CheckoutTask checkoutReq = new CheckoutTask();
                checkoutReq.execute(getEndpoint() + "cart/co");
                currentCartItemsMap.clear();
                convertItemsMaptoList();
        Instrumentation.startNextSession();
        } else {
            displayToast("There are no items in the cart");
        }
}
```

Start and End Session Frames

You can use the React Native Agent API to create session frames that appear in the session activity. Session frames provide context for what the user is doing during a session. With the API, you can improve the names of user screens and chronicle user flows within a business context.

Use Cases

The following are common use cases for session frames:

- One page performs multiple functions and you want more granular tracking of the individual functions.
- A user flow spans multiple pages or user interactions. For example, you could use the API to create the session frames "Login", "Product Selection", and "Purchase" to chronicle the user flow for purchases.
- You want to capture dynamic information based on user interactions to name session frames, such as an order ID.

Class/Interface

The SessionFrame API is accessible through the `Instrumentation` class.

```
import { Instrumentation } from '@appdynamics/react-native-agent';
```

To update the name and end a session frame, you would use the `SessionFrame` interface.

Methods

The following table lists the three methods you can use with session frames. In short, you start a session frame with `startSessionFrame` and then use `updateName` and `end` to rename and end the session frame.

Method	Description
`startSessionFrame(name)`	Starts the session frame. You call this method from `Instrumentation`, and it returns `SessionFrame` object.
`updateName(name)`	Updates the name of the session frame. You call this method from the `SessionFrame` object.

Method Parameters

The methods `startSessionFrame` and `updateName` takes the following parameter:

Parameters	Data Type	Description
name	string	The name of the session frame.

Session Frame Example

In the following example, session frames are used to track user activity during the checkout process.

```
let sessionFrame: SessionFrame | undefined;

private onCheckoutCartButtonClicked() {
    // The user starts the checkout by clicking the checkout button.
    // This may be after they have updated the quantities of items in the cart, etc.
    sessionFrame = Instrumentation.startSessionFrame("Checkout");
}
private onConfirmOrderButtonClicked() {
    // Once they have confirmed payment info and shipping information, and they
    // are clicking the "Confirm" button to start the backend process of checking out,
    // we may know more information about the order itself, such as an order ID.
    if (sessionFrame) {
        sessionFrame.updateName("Checkout: Order ID " + orderId);
    }
}
private onProcessOrderCompleted() {
    // Once the order is processed, the user is done "checking out", so we end the session frame.
    if (sessionFrame) {
            sessionFrame.end();
            sessionFrame = null;
    }
}
private onCheckoutCancled() {
  // If the user cancels or returns to the cart, you'll want to end the session frame also, or else it will be
    // left open and appear to have never ended.
    sessionFrame.end();
    sessionFrame = null;
  }
}
```

Configure and Take Screenshots

Mobile screenshots are enabled by default. These screenshots appear in the **Sessions Details** dialog.

Class

The Screenshots API is accessible through the `Instrumentation` class:

```
import { Instrumentation } from '@appdynamics/react-native-agent';
```

Methods

The Screenshot API provides the following methods:

Method	Return Value	Description
`takeScreenshot()`	None	Asynchronously takes a screenshot of the current screen. This will capture everything, including personal information, so you must be cautious of when to take the screenshot. These screenshots will show up in the Sessions screen for this user. The screenshots are taken on a background thread, compressed, and only non-redundant parts are uploaded, so it is safe to take many of these without impacting the performance of your application.
`screenshotsBlocked()`	Promise<boolean>	Returns a Boolean indicating whether screenshot capture is blocked.
`blockScreenshots()`	Promise<void>	Blocks screenshot capture and returns a Promise that resolves when screenshots are effectively blocked.
`unblockScreenshots()`	Promise<void>	Unblocks screenshot capture if it is currently blocked. Otherwise, this has no effect. If screenshots are disabled through `AgentConfiguration.screenshotsEnabled` or through the Controller UI, this method has no effect. If screenshots are set to manual mode in the Controller UI, this method unblocks for manual mode only.

Configure Screenshots

Disable Screenshots

You can disable screenshots from the Controller UI or with the React Native API. To disable screenshots, set the property `screenshotsEnabled` to `false` when initializing the Native React Agent.

```
Instrumentation.start({
    appKey: <EUM_APP_KEY>,
    screenshotsEnabled: false,
    ...
});
```

Set On-Prem Screenshot Service

If you have deployed an on-prem EUM Server, you will need to specify the URL to the EUM Server with the property `screenshotURL`.

```
Instrumentation.start({
    appKey: <EUM_APP_KEY>,
    screenshotURL: "https://<COLLECTOR_URL>:<PORT>",
    ...
});
```

Examples

Take Screenshots

You can configure the Controller UI to automatically take screenshots or use the React Native API to manually take a screenshot as shown below:

```
private loadShoppingCart() {
  // Load shopping cart
  this.setState({
      shoppingCart: this.state.cart
  });
  // Manually take screenshot
  Instrumentation.takeScreenshot();
}
```

This will capture everything, including personal information, so precaution is necessary when taking screenshots

Block and Unblock Screenshots

You can also block screenshots from being taken during the execution of a code block. This just temporarily blocks screenshots from being taken until you unblock screenshots. This enables you to stop taking screenshots in situations where users are entering personal data, such as on login and account screens.

```
private displayCustomerAccount() {
    // Check to see if screenshots are blocked
    if (! Instrumentation.screenshotsBlocked()) {
        // If screenshots aren't blocked, block them before showing customer details
        Instrumentation.blockScreenshots();
    }
    // Code to display customer details

    // After you're done, unblock screenshots
    Instrumentation.unblockScreenshots();
}
```

Enable Logging and Set Logging Level

You enable logging by setting the logging level in the instrumentation configuration. You are recommended to disable logging for production.

Class

The Logging Level API is accessible through the `LoggingLevel` class:

```
import { LoggingLevel } from '@appdynamics/react-native-agent';
```

Configuration Properties

You enable and set the logging level with the following configuration property:

- `loggingLevel`

Logging Levels

You can set logging to one of the following levels:

Enumeration	Logging Level	Description
`LoggingLevel.NONE`	None	No logs are displayed. **This level disables logging.**
`LoggingLevel.INFO`	Info	Warning, errors, and developer-focused messages are displayed.
`LoggingLevel.VERBOSE`	Verbose	Errors, warnings, developer information, debugging, and troubleshooting messages are displayed.

Examples

In the following example, logging is enabled and the logging level is set to `VERBOSE`:

```
private async start() {
        try {
            await Instrumentation.start({
                appKey: this.state.appKey,
                loggingLevel: LoggingLevel.VERBOSE,
                anrDetectionEnabled: true,
                interactionCaptureMode: InteractionCaptureMode.None.with(
                    InteractionCaptureMode.ButtonPressed,
                    InteractionCaptureMode.ListViewItemSelected,
                    InteractionCaptureMode.TableCellSelected,
                    InteractionCaptureMode.TextFieldSelected,
                    InteractionCaptureMode.TextViewSelected
                )
            }
        )
    }
}
```

Receive a Crash Report Summary for Native Crashes

You can use `CrashReportCallback` to receive a report for native crashes.

Class

The `CrashReportCallback` API is accessible through the `Instrumentation` class:

```
import { Instrumentation } from '@appdynamics/react-native-agent';
```

How to Use

1. Submit the `CrashReportCallback` object as the `crashReportCallback` native agent configuration option.

   ```
   Instrumentation.start({
         crashReportCallback: (summaries: CrashReportSummary[]) => {
             console.log(summaries);
         },
    });
   ```

2. On application restart, you will receive a `CrashReportSummary` for each crash.

   ```
   export type CrashReportSummary = {
     crashId: string;
     exceptionName: string;
     exceptionReason: string;
     signalName: string;
     signalCode: string;
   };
   ```

`CrashReportSummary` Properties

`CrashReportSummary` has the following properties:

Name	Type	Description
`crashId`	string	Uniquely defines the crash. Can be used as key to find the full crash report or look up the crash in the Controller UI.
`exceptionName`	string	The exception name, may be `null` if no exception occured.
`exceptionReason`	string	The exception reason, may be `null` if no exception occurred.
`signalName`	string	The exception signal name.

`signalCode`	string	The exception signal code.

Disable Crash Reporting

Crash reporting is enabled by default, but you can manually disable crash reporting through the instrumentation configuration. If you are using other crash reporting tools, you might disable crash reporting to minimize conflicts and optimize the crash report results.

You can disable crash reporting by configuring the instrumentation with the `crashReportingEnabled` property as shown in the following:

```
import { Instrumentation } from '@appdynamics/react-native-agent';

Instrumentation.start({
        appKey: <#EUM_APP_KEY#>,
        crashReportingEnabled: false
})
```

Log Silenced Errors

If you use React Native's `ErrorBoundary`, the errors are silently caught, but the agent will not receive and log them. If you want silenced crashes to be logged, we recommend you use `ErrorBoundary` provided in the AppDynamics React Native package:

```
import { ErrorBoundary } from '@appdynamics/react-native-agent'
```

Use the Agent with a Custom HTTP Library

The agent automatically detects network requests when the underlying implementation is handled by `NSURLConnection`/`NSURLSession` classes (iOS) or `HttpURLConnection`/`OkHttp` (Android). This covers most network requests, but in some cases, mobile applications use custom HTTP libraries.

To enable the agent to detect requests from a custom library, add request tracking code to your application manually with the `RequestTracker` class.

You must initialize the agent with a valid key using the `Instrumentation.start()` interface before using this method.

`RequestTracker` Properties

The following properties should be set on the RequestTracker object to describe the parameters of the custom call.

Property	Description	Code
`error`	Javascript error or `null` An error describing the failure to receive a response, if one occurred. If the request was successful, this should be left `null`.	`tracker.setError(e);`
`statusCode`	Number or `null` The status code of response, if one was received. If a response was received, this should be an an integer. If an error occurred and a response was not received, this should remain `null`.	`tracker.setResponseStatusCode(response.statusCode);`
`requestHeaders`	Dictionary or `null` A dictionary representing the keys and values from the client's request header.	`tracker.setRequestHeaders(headers);`
`responseHeaders`	Dictionary or `null` A dictionary representing the keys and values from the server's response header. If an error occurred and a response was not received, this should be `null`.	`tracker.setResponseHeaders(headers);`

Track a Custom HTTP Request

To add manual request tracking, you must specify when the request begins, when it ends and what the status of the response is.

1. Create a `RequestTracker` object immediately before sending the request:

```
import { Instrumentation } from '@appdynamics/react-native-agent';
...
const tracker = new RequestTracker({ "My_URL" });
```

"`MY_URL`" is the URL being called. This parameter must not be `null`.

2. (Optional) Add your custom headers to the tracker before sending the request:

```
 const headers = {
  'Content-Length': '5000',
         };
 tracker.setRequestHeaders(headers);
```

`requestHeaders`: A dictionary (or `null`) representing the keys and values from the request headers.

3. (Optional) Enable correlation between your request and server-side processing. Add specific headers to outgoing requests that the server-side agent can detect and return the headers obtained from the server-side agent in the response to make them available to the agent:

```
tracker.addServerCorrelationHeaders()
```

4. Immediately after receiving a response or an error, set properties on the tracker object based on whether the request finished successfully or ended in error:

```
try {
    const response = await customRequest(url);
    tracker.setResponseStatusCode(response.statusCode);
    tracker.setResponseHeaders(response.headers);
} catch (e) {
    tracker.setError(e);
}
```

6. Call `reportDone()`:

```
tracker.reportDone();
```


To track another request, instantiate a new tracker object again.

Complete example

```
const tracker = new requestTracker({
  url: <MY_URL>
});
```

```
const headers = {
  'Content-Length': '5000',
};
tracker.setRequestHeaders(headers);
tracker.addServerCorrelationHeaders();


try {
  const response = await customRequest(url);
  tracker.setResponseStatusCode(response.statusCode);
  tracker.setResponseHeaders(response.headers);
} catch (e) {
  tracker.setError(e);
} finally {
  tracker.reportDone();
}
```

React Native API Documentation

For the React Native Agent API reference documentation, see the latest React Native Agent API documentation or the previous versions:

- https://sdkdocs.appdynamics.com/react-native-agent/21.4/
- https://sdkdocs.appdynamics.com/react-native-agent/21.2/
- https://sdkdocs.appdynamics.com/react-native-agent/20.11/
- https://sdkdocs.appdynamics.com/react-native-agent/20.10/
- https://sdkdocs.appdynamics.com/react-native-agent/20.7/
- https://sdkdocs.appdynamics.com/react-native-agent/1/1.0/

Mobile SDK API Documentation

Related pages:

- Customize the iOS Instrumentation
- Customize the Android Instrumentation
- Customize the Xamarin Instrumentation
- Customize the Cordova Instrumentation
- Customize the React Native Instrumentation

This page provides links to the Mobile SDK API documentation:

iOS SDK Documentation

See the latest iOS SDK documentation or the previous versions listed below:

- https://sdkdocs.appdynamics.com/ios-sdk/21.5/html/
- https://sdkdocs.appdynamics.com/ios-sdk/21.2/html/
- https://sdkdocs.appdynamics.com/ios-sdk/20.12/html/
- https://sdkdocs.appdynamics.com/ios-sdk/20.10/html/
- https://sdkdocs.appdynamics.com/ios-sdk/20.8/html
- https://sdkdocs.appdynamics.com/ios-sdk/20.7/html
- https://sdkdocs.appdynamics.com/ios-sdk/2020.3/html

Android SDK Documentation

For the complete SDK API documentation, see the latest JavaDocs or the previous versions listed below:

- https://sdkdocs.appdynamics.com/android-sdk/21.5/
- https://sdkdocs.appdynamics.com/android-sdk/21.2/
- https://sdkdocs.appdynamics.com/android-sdk/20.11/
- https://sdkdocs.appdynamics.com/android-sdk/20.10/
- https://sdkdocs.appdynamics.com/android-sdk/20.7/
- https://sdkdocs.appdynamics.com/android-sdk/20.5/
- https://sdkdocs.appdynamics.com/android-sdk/20.3/

Xamarin SDK Documentation

For the complete SDK API documentation, see the latest Xamarin SDK documentation or the previous versions listed below:

- https://sdkdocs.appdynamics.com/xamarin-sdk/21.5/html/
- https://sdkdocs.appdynamics.com/xamarin-sdk/21.2/html/
- https://sdkdocs.appdynamics.com/xamarin-sdk/20.11/html/
- https://sdkdocs.appdynamics.com/xamarin-sdk/20.10/html/
- https://sdkdocs.appdynamics.com/xamarin-sdk/20.4/html/
- https://sdkdocs.appdynamics.com/xamarin-sdk/20.3/html/

Cordova SDK Documentation

For the Cordova SDK API reference documentation, see the latest JSDoc documentation or the previous versions listed below:

- https://sdkdocs.appdynamics.com/cordova-plugin/21.5/
- https://sdkdocs.appdynamics.com/cordova-plugin/21.2/
- https://sdkdocs.appdynamics.com/cordova-plugin/20.11/
- https://sdkdocs.appdynamics.com/cordova-plugin/20.10/
- https://sdkdocs.appdynamics.com/cordova-plugin/20.7/

React Native API Documentation

For the React Native Agent API reference documentation, see the latest React Native Agent API documentation or the previous versions:

- https://sdkdocs.appdynamics.com/react-native-agent/21.4/
- https://sdkdocs.appdynamics.com/react-native-agent/21.2/
- https://sdkdocs.appdynamics.com/react-native-agent/20.11/

- https://sdkdocs.appdynamics.com/react-native-agent/20.10/
- https://sdkdocs.appdynamics.com/react-native-agent/20.7/
- https://sdkdocs.appdynamics.com/react-native-agent/1/1.0/

Troubleshoot Mobile Applications

You can use Mobile RUM to investigate two different kinds of problems that can arise with your mobile applications:

- Slow network requests
- Mobile application crashes

Identify Your Slowest Network Request Types

To identify slow network requests:

1. Open the application in which you are interested.
2. In the left navigation bar, click **Network Requests**.
3. Select the Network Requests tab.
4. Click the top of the **Network Request Time (ms)** column, then toggle it to sort the network requests with the slowest ones at the top.
5. Skip over network requests that you expect to run for a long time or that have very little load (low Requests per Minute).
6. Select and double-click one of the slow network requests that you want to investigate.
7. In the network request dashboard, view the **Key Performance Indicators** at the top of the **Network Request Dashboard**. For example:

    - If the value for Network Request Time is large, the request or response body may be too large and is taking a while to transmit, or the data connection might be slow.
    - If the backend server is instrumented and the value for Total Server Time makes up a significant amount of the delay, scroll down to the **Related Business Transactions** section to investigate related business transactions on the server side.

Access Details of Individual Instances of Slow Requests

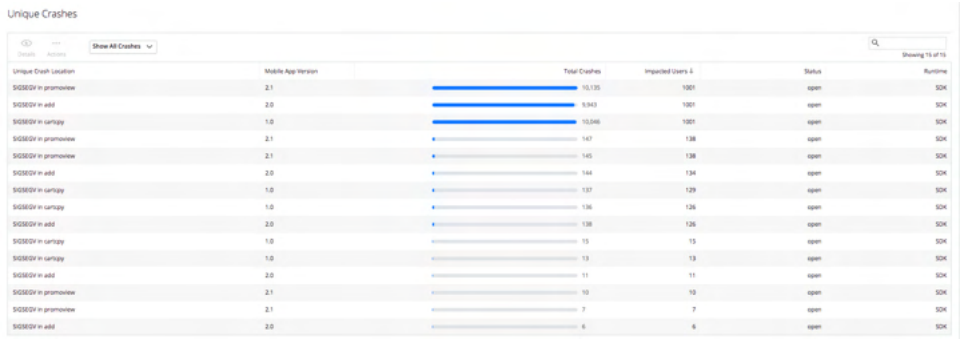
To investigate the cause of slow requests:

1. Select the Snapshots tab. The **Snapshots List** opens.
2. Click **Filters**.
3. In the **Network Request Names** dropdown under **Network in the Filters** panel, check the check box for the network request that you identified in Identify your slow network requests, then click **Search**.
   This restricts the list to snapshots for that network request only.
4. Click **Filters** again to close the filters panel.
5. In the list, click the top of the **Network Request Time (ms)** column, then toggle it to sort the network request snapshots with the slowest requests at the top.
6. Select and double-click one of the slow network requests.
   The network request snapshot displays the details of the slow request.
7. If this request is associated with a server-side application that is also instrumented, scroll down to see if transaction snapshots for this request are available on the server side.
   If transaction snapshots are available and if most of the time for this network request is spent on the server, click through to the related transaction snapshots to understand what is causing the slow performance on the server. See Transaction Snapshots.

Identify Applications that Crash the Most or Affect the Most Users

To troubleshoot mobile application crashes with crash dashboards and crash snapshots:

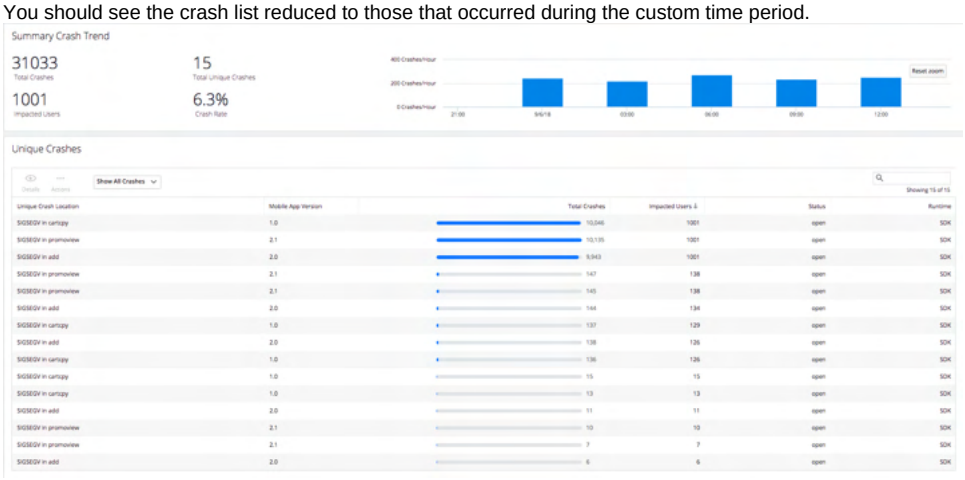
1. Open the application in which you are interested.
2. On the left navigation bar, click **Crashes**.
3. Select the Crash Dashboard tab.
4. Check the **Unique Crashes** list.
5. Sort either by **Total Crashes** or **Impacted Users**, depending on what you want to know. In the screenshot below, the list is sorted by impacted users.



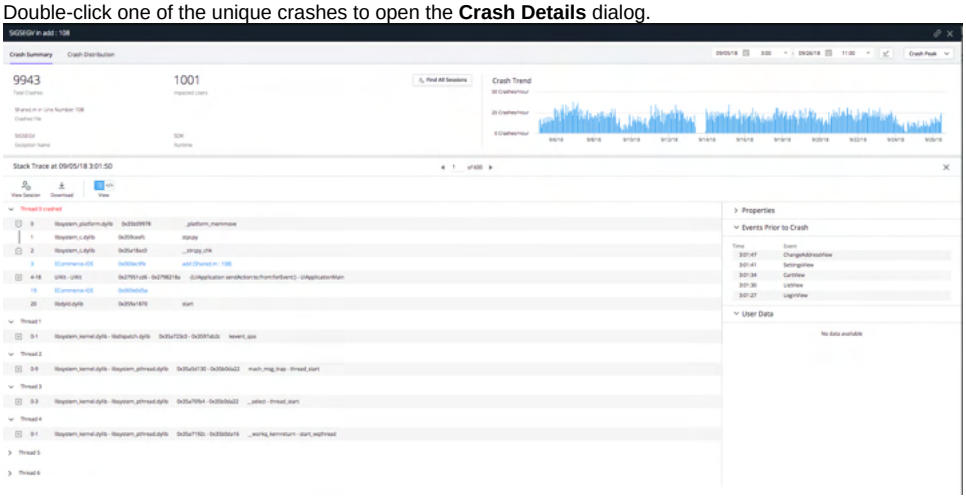
6. View the **Summary Crash Trend** graph to discover particularly problematic time periods.

7. To find crashes in the problematic timeframe, click **Analyze** and drag up on the timeline where the most crashes occurred.

8. You should see the crash list reduced to those that occurred during the custom time period.



9. Double-click one of the unique crashes to open the **Crash Details** dialog.



10. From here, you can view the stack trace of the crash snapshot, note the thread and function in which the crash occurred. For some crashes, the crashed line number is also available. Optional: Click **Download** to get a text version of the stack trace to send to your application development team.
11. To access a complete data set for crashes that belong to this crash group, click **Find Sessions**. This opens the Mobile Sessions view with a filter for this **Crash Group** already selected.

Get More Information about Mobile RUM

Use the following to find out more about Mobile RUM.

- Mobile RUM Metrics
- Mobile RUM Supported Environments
- Real User Monitoring Licenses
- Network Request Limits

Mobile RUM Metrics

AppDynamics displays key metrics for Mobile RUM on the various dashboards in the Mobile UI and in the **Metric Browser**.

> With the exception of App Crashes per Minute, the crash information displayed in the Controller UI is based on data stored in the Events Store and does not display in the **Metric Browser**.

Crashes

Crash measurements are created using the Analyze event store, and display in the **Crash Dashboard**.

- **Total Crashes:** total number of crashes over the selected time range
- **Total Unique Crashes:** total number of unique crash types over the selected time range
- **Impacted Users**: total number of unique users affected by a crash
- **Crashes by Usage Stats:** percentage of crashes based on device, OS version, carrier, and connection type
- **Crashes by Geo Location:** total number of crashes by origin over the selected time range

Crash Metrics

- **App Crashes per Minute:** arithmetic average number of crashes

HTTP Errors

An HTTP error occurs when an HTTP request is sent and a response is received, but the response status code indicates that an error occurred. These errors suggest that the network is working correctly, but a problem exists on the client side (4xx status codes) or the server side (5xx status codes) that prevented normal handling of the request.

- **HTTP Errors Per Minute**: arithmetic average of errors per minute for errors that return an HTTP response code between 400 and 599
- **HTTP Errors (total):** total number of HTTP errors over the selected time range (shown in the **Geo Dashboard**)

Mobile Device Metrics

See Mobile Device Metric Collection for more information.

Network Errors

A network error is any occurrence that prevents the HTTP request from being sent or the HTTP response from being received successfully.

- **Network Errors per Minute**: arithmetic average of network errors per minute
- **Network Errors (total)**: total number of network errors over the selected time range (shown in the **Geo Dashboard**)

Typical causes of network errors include:

- Host cannot be resolved
- Host refused connection
- Connection timed out
- Device is offline
- General connectivity problems

Network Requests

Network request metrics are reported for each platform and instrumented mobile application.

- **Network Request Time**: arithmetic average interval in milliseconds between the time that a mobile application initiates a request by calling the system API and the time that the system returns the response to the application
- **Requests per Minute**: arithmetic average number of HTTP requests per minute
- **Total Requests**: total of HTTP requests per minute over the selected time range (shown in the **Geo Dashboard**)
- **Request Content Length:** total number of bytes of data in the body of the request. The body is the part that comes after the blank line below the headers
- **Response Content Length:** total number of bytes of data in the body of the response

Mobile RUM Supported Environments

See the following for the list the supported environments for Android, iOS, Xamarin, and Cordova:

- Android Agent
- iOS Agent
- Xamarin Agent
- Cordova Plugin
- React Native Agent

Mobile RUM Support

Android Agent

Supported Environments	Name	Supported Version(s)
Operating System	Android	>= 2.3.3
Architecture	arm64-v8a, armeabi, armeabi-v7a, mips, mips64, x86, and x86_64	-
Frameworks	Ant	-
	Gradle	>= 0.6.3
	Maven	3.1.1
Crash Reporters	Mobile RUM does not officially support 3rd-party crash reporters.	N/A
HTTP Libraries	`HttpURLConnection/HttpsURLConnection`	-
	`HttpClient`	-
	`OkHttp`	2.2.0 - 3.9.0
	`ch.boye.httpclientandroidlib`	-
	`retrofit`	-
	`retrofit2`	-
	Other HTTP libraries can be added by using the agent SDK. See Customize the Android Instrumentation for more information.	-

Android Gradle Plugin Version	Minimum Android Agent Version
0.6.3	3.8.2.0
0.7.3 - 0.9.2	3.8.3.0
0.10.0 - 0.12.0	3.9.0.0
1.1.2 - 1.1.3	4.0.3.3
1.3.0 - 1.4.0	4.0.7
1.5.0 - 2.2.0	4.2.0*
2.3.0 - 3.4.0 (forces Build Tools 25.0.0 or higher)	4.2.9
>= 3.4.1	>= 20.7.1

Recommendations for Android Agent Versions

- If your project uses Android Gradle Plugin version < 3.4.1, use Android Agent 20.4.0.
- If your project uses Android Gradle Plugin version >= 3.4.1, use Android Agent >= 20.7.1.

*Android Agent Plugin versions before 4.2.9 are not compatible with Android Build Tools versions >=24 because of the Java 8 bytecode generated in class files.

iOS Agent

Supported Environments	Name	Supported Version(s)
Operating System	iOS	>= 9
Architecture	Apple 32-bit ARM, Apple 64-bit A7	-
Framework	XCode	>= 8
Crash Reporters	Mobile RUM does not officially support 3rd-party crash reporters.	N/A
Apple WatchKit Extension Environments	watchOS	1
HTTP Libraries	`NSURLConnection`	-
	`NSURLSession`	-
	Alamofire - Initialize the iOS Agent before initializing Alamofire to ensure that network requests are monitored. If you need to make a network request(s) prior to iOS Agent initialization, see Alamofire Guidelines.	-
	Other HTTP libraries can be added by using the agent SDK. See Customize the iOS Instrumentation for more information.	-

Xamarin Agent

Supported Environments	Name	Supported Version(s)
Operating System	Android	>= 2.3.3
	iOS	>= 9
Architecture	Xamarin.Android Supported CPU Architectures	-
	Apple 32-bit ARM, Apple 64-bit A7 - See 32/64-bit platform considerations for information regarding the correct architecture for your iOS version.	-
Framework	.NET - Mono	>= 4.8
Crash Reporters	Mobile RUM does not officially support 3rd-party crash reporters.	N/A
Apple WatchKit Extension Environments	Not supported	-
HTTP Libraries	N/A - The Xamarin Agent does not support automatic instrumentation for network requests made with any library. You will need to manually instrument HTTP network requests regardless of what library is used.	-

Cordova Plugin

Supported Environments	Name	Supported Version(s)
Operating System	Android	4.4 - 8.1
	iOS	>= 9
Architecture	Android: arm64-v8a, armeabi, armeabi-v7a, mips, mips64, x86, and x86_64	-
	iOS: Apple 32-bit ARM, Apple 64-bit A7	-
Framework	Apache Cordova	>= 7
Crash Reporters	Mobile RUM does not officially support 3rd-party crash reporters.	N/A
Apple WatchKit Extension Environments	Not supported	-

HTTP Libraries	Android: • `HttpURLConnection/HttpsURLConnection` • `HttpClient` • `OkHttp` • `ch.boye.httpclientandroidlib`	-
	iOS: • `NSURLConnection` • `NSURLSession`	-

React Native Agent

Supported Environments	Name	Supported Version(s)
Operating System	Android	>= 4.1 (with React Native Agent >= 20.10.1)
	iOS	>= 11 (with React Native Agent >= 20.10.1)
Architecture	Android: arm64-v8a, armeabi, armeabi-v7a, mips, mips64, x86, and x86_64	-
	iOS: Apple 64-bit A7 (with React Native Agent >= 20.10.1)	-
Framework	React Native	>= 0.60 (with React Native Agent >= 20.10.1) < 0.60 (with React Native Agent 20.7.0)
Crash Reporters	Mobile RUM does not officially support 3rd-party crash reporters.	N/A
Apple WatchKit Extension Environments	Not supported	-
HTTP Libraries	Android: • `HttpURLConnection/HttpsURLConnection` • `HttpClient` • `OkHttp` • `ch.boye.httpclientandroidlib`	-
	iOS: • `NSURLConnection` • `NSURLSession`	-

Alamofire Guidelines

If you need to use Alamofire to make a network request(s) prior to initializing the iOS Agent, you can use the following sample code as a guideline for making the necessary changes.

To ensure that Alamofire network requests made after iOS agent initialization will be instrumented, you can initialize a custom instance of `Session` (the principle class defined in Alamofire 5, renamed from previous Alamofire versions). This will create a new instance of `URLSession`, which can be instrumented by the iOS Agent. Any Alamofire network requests made prior to the agent initialization and creation of the Session instance cannot be instrumented by the agent.

- The order of operations in this code is very important. See the inline comments.
- This code assumes you are initially creating a customized instance of `Session` (which is common practice). If not, you can eliminate the first usage of customized `Session` and defer its creation until after the iOS agent initialization.
- You can test this code by going to https://www.google.com/?q=request%202, which is instrumented because it occurs after the replacement of `mySession` with `instrumentedSession`. The other two requests are not instrumented.

```
import UIKit
import Alamofire
import ADEUMInstrumentation

var mySession:Session?                                          // customized Session singleton

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

        func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {
                // Override point for customization after application launch.

                // create customized instance of AF Session
                let myConfiguration = URLSessionConfiguration.af.default
                mySession = Session(configuration: myConfiguration)

                let myUrl1 = URL(string: "https://www.google.com/?q=request%201")!
                let urlRequest1 = URLRequest(url: myUrl1)

                // the first network request prior to ADEumInstrumentation.initWith will not be instrumented
                mySession?.request(urlRequest1 as URLRequestConvertible).response {
                        response in debugPrint(response)

                        // initialize iOS agent
                        let config = ADEumAgentConfiguration(appKey: "XX-XXX-XXX-XXX")
                        config.loggingLevel = .all
                        ADEumInstrumentation.initWith(config)

                        // update our Session instance singleton, instrumentedSession, so that URLSession will
be instrumented
                        let myConfiguration = URLSessionConfiguration.af.default
                        let instrumentedSession = Session(configuration: myConfiguration)
                        // replace mySession with instrumentedSession
                        mySession = instrumentedSession

                        // subsequent requests use updated mySession from instrumentedSession, so should be
instrumented
                        let myUrl2 = URL(string: "https://www.google.com/?q=request%202")!
                        let urlRequest2 = URLRequest(url: myUrl2)
                        mySession?.request(urlRequest2 as URLRequestConvertible).response {
                                response in debugPrint(response)
                        }
                }
                // has instrumentedSession been created yet? Indeterminate. If not, this request won't be
instrumented
                let myUrl3 = URL(string: "https://www.google.com/?q=request%203")!
                let urlRequest3 = URLRequest(url: myUrl3)
                mySession?.request(urlRequest3 as URLRequestConvertible).response {
                        response in debugPrint(response)
                }
                return true
        }

}
```

Network Request Limits

The EUM Server (on-premises/SaaS) can process a maximum of 2000 network request types per mobile app group and 500 network requests per mobile application.

If the number of network requests for an application exceeds one of these limits, AppDynamics continues to monitor network requests that have already been discovered, but new network requests will not be discovered. Should a limit be exceeded, a warning appears on the network request list.

You can use these techniques to keep your usage under the two network request limits.

Remove Network Requests Without Load

Removing network request types that have been discovered but are not receiving any load is a good place to start to reduce the number of your registered requests.

> Deleting network requests does not prevent them from being re-discovered in the future if the request once again comes under load. To prevent specific network requests from ever being discovered, you must create exclude rules. See the section "Creating Mobile Exclude Rules" in Configure Mobile Network Request Naming.

Delete network requests with no load

1. In the **Network Requests** list, uncheck the **With Load** checkbox.
2. Refresh the list.
3. Find the requests that have no load. They are candidates for removal.
4. Select the requests that you want to delete.
5. In the **More Actions** dropdown, click **Delete Request(s)**.

Exclude Requests That Do Not Need to Be Monitored

The Mobile Agent may be detecting network requests types that are not interesting for you to monitor. You have two methods to exclude these types of network requests.

The first method is to add custom exclude rules from **Configuration > Mobile Request Naming, Thresholds, & Percentiles.** See Creating Mobile Exclude in Rules for explicit instructions.

Using custom exclude rules, network requests for new data that match the exclude rules will not be registered. The load on the existing network requests that match the exclude rule will drop to zero. To delete any network requests of the excluded type that have already been registered from the network request list, follow the procedure described in Delete network requests with no load.

The second method is to select an existing network request from the **Network Requests** list and then exclude it. This will change the app's exclude rules so that incoming data matching that request will not be tracked. In addition, the historical data for that request will be deleted.

To use the second method for the exclude rules that prevent a network request from being tracked and remove historical data for that network request:

1. From **Network Requests**, select the network request you want to exclude.
2. Click **Actions > Exclude Requests** to open the **Exclude Requests** dialog.
3. Click **Exclude**.

Group Network Requests of Similar Type

Review the default network request naming rule described in Configure Mobile Network Request Naming. It is possible that the default rule is generating many more network requests types than are desirable.

For example, perhaps your application loads images dynamically and stores them on your server with URLs like `http://myapp.com/image/image1234.jpg`. This would cause a separate network request to be generated for each image, which is probably not what you want. You could create a custom naming rule to group all the image URLs as a single network request. See Configure Mobile Network Request Naming.

After you have created custom rules to reduce the number of network requests detected, unregister the network requests for those are now covered by the custom rule, following the procedure described in Delete network requests with no load.

Mobile Agent Version and Deployment Support Matrix

Related pages:

- Mobile RUM Supported Environments

Using the AppDynamics Mobile Agents with the following recommended minimum Controller/EUM Server versions will ensure that all features for the specified releases will function correctly and that there will be no unexpected side effects.

Mobile Agent	Agent Version(s)	Minimum Controller/EUM Server Version
React Native Agent	1.0.x	4.5.8
iOS Agent	50.0.x	4.5.0
	50.1.x	4.5.8
Android Agent	5.0.x	4.5.0
	5.1.x	4.5.8
Xamarin Agent	>= 50.0.x	4.5.0
	50.1.x	4.5.8
Cordova Plugin	1.1.x – 1.7.x	4.5.0
	1.8.x	4.5.8

Post-release Support for the Mobile Agents

Although Mobile Agents may be compatible with future releases of the Controller and EUM Server, AppDynamics only provides support for Mobile Agent versions with the Controller and EUM Server for two years after the release of the agent.

Deployment Support for Mobile Agent Features

For deployment support for individual mobile features, see:

- iOS SDK Documentation
- Android SDK Documentation
- Xamarin SDK Documentation
- Cordova SDK Documentation
- React Native Agent Documentation

Mobile Agent Feature Support

This table lists the feature support for the Mobile Agents:

- Native Mobile Agents
- Hybrid Agents

For instrumentation instructions, see the custom instrumentation pages (Android, iOS, Xamarin, Cordova, React Native).

Native Mobile Agents

Feature	Mobile Agents	
	Android	**iOS**
Errors		
Crash Reporting	✓	✓
App Not Responding (ANR)	✓	✓
Error Reporting	✓	✓
Crash Reporting Callback	✓	✓
Network		
Automatic Network Request Capture	✓ **Supported Libraries** • `URLConnection` • `HttpClient` • `OkHttp` • `OkHttp3` • `HttpClientAndroidLib`	✓ **Supported Libraries** • `NSURLConnection` • `NSSession`
Manual Network Request Reporting	✓	✓
Business Transaction Correlation	✓	✓
Connection Transition Reporting	✓	✓
Network Request Callback	✓	✓
User Interaction		
Activity Tracking	✓ (Use the SessionFrame API)	✗
Root View Tracking	✗	✓ (Use the SessionFrame API)
UI Tracking	✓ **Captured User Interactions** • EditText • Button • List Selection	✓ **Captured User Interactions** • EditText • Button • List Selection
Fragment Tracking	✓ (Use the SessionFrame API)	✗

View Controller Tracking	✗	✓
Screenshots	✓	✓
Screenshots Touch Points	✓	✓
Custom Data		
Static Info Points (annotations in code)	✓	✗
Manual Info Points	✓	✓
Breadcrumbs	✓	✓
User Data	✓	✓
Custom Metrics	✓	✓
Session Frame API	✓	✓
Custom Timers	✓	✓
Programmatic Session Control API	✓	✓
Configuration		
Custom Collector Channel	✓	✓
Set App Key After Initialization	✓	✓
Hybrid Support (JavaScript Agent Support)		
Auto-injection of the JavaScript Agent into WebViews	✓	✓
Base Page Entry and Virtual Page as Mobile Network Requests	✓	✓
Ajax Calls as Mobile Network Requests	✓	✓

Hybrid Agents

Feature	Mobile Agents		
	Xamarin	Cordova	React Native
Errors			
Crash Reporting	✓	✗	✓
App Not Responding (ANR)	✗	✗	✓
Error Reporting	✗	✗	✓
Crash Reporting Callback	✗	✗	✗
Network			
Automatic Network Request Capture	✗	✓	✓ XHRs are made through the OkHttp3 and NSURLSession libraries.
Manual Network Request Reporting	✓	✓	✗
Business Transaction Correlation	✓	✗	✓
Connection Transition Reporting	✓	✗	✗
Network Request Callback	✗	✗	✗
User Interaction			
Activity Tracking	✓ (Use the ISessionFrame API)	✗	✗

Root View Tracking	✓ (Use the ISessionFrame API)	✗	✗
UI Tracking	✗	✗	✗
Fragment Tracking	✓ (Use the ISessionFrame API)	✗	✗
View Controller Tracking	✓ (Use the ISessionFrame API)	✗	✗
Screenshots	✗	✓	✓
Screenshots Touch Points	✗	✓	✓
Custom Data			
Static Info Points (annotations in code)	✗	✗	✓
Manual Info Points	✓	✓	✓
Breadcrumbs	✓	✓	✓
User Data	✓	✓	✓
Custom Metrics	✓	✓	✓
Session Frame API	✓	✓	✓
Custom Timers	✓	✓	✓
Programmatic Session Control API	✓	✓	✓
Configuration			
Custom Collector Channel	✗	✗	✗
Set App Key After Initialization	✓	✗	✓
Hybrid Support (JavaScript Agent Support)			
Auto-injection of the JavaScript Agent into WebViews	✗	✓	✗
Base Page Entry and Virtual Page as Mobile Network Requests	✗	✓	✗
Ajax Calls as Mobile Network Requests	✗	✓	✗

Hybrid Application Support

AppDynamics supports monitoring for hybrid applications based on the iOS/Android native SDK or any Cordova-based framework.

Hybrid Applications

Hybrid applications are web applications running inside an Android or iOS native shell. The web application can be in the form of an Android WebView, an iOS WKWebViews, or a Cordova web application that consists of HTML, CSS, JavaScript, and other resources just like a conventional web application.

Hybrid Use Cases

Mobile RUM supports these two use cases.

- Native Mobile applications that run web applications or web views. Examples of this are Android WebViews and iOS WKWebView.
- Applications written in Cordova-based frameworks, such as Ionic, that provide a native shell for running a web application and an interface for native mobile APIs.

Hybrid Application Instrumentation Requirements

For instrumenting hybrid apps, you are required to:

- Manually instrument the iFrames with the JavaScript Agent. For mobile native applications, the JavaScript Agent is injected into the top-level WebView or WKWebView instance, not in embedded web pages (iFrames).
- Manually report events for Cordova-based applications using the plugin JavaScript API.
- Use Android WebViews or iOS WKWebView for hybrid applications with embedded web views. See Instrument Cordova Applications.

Instrumentation Steps

To monitor your hybrid applications, you will need to set up and access Mobile RUM and follow the steps below based on your hybrid use case:

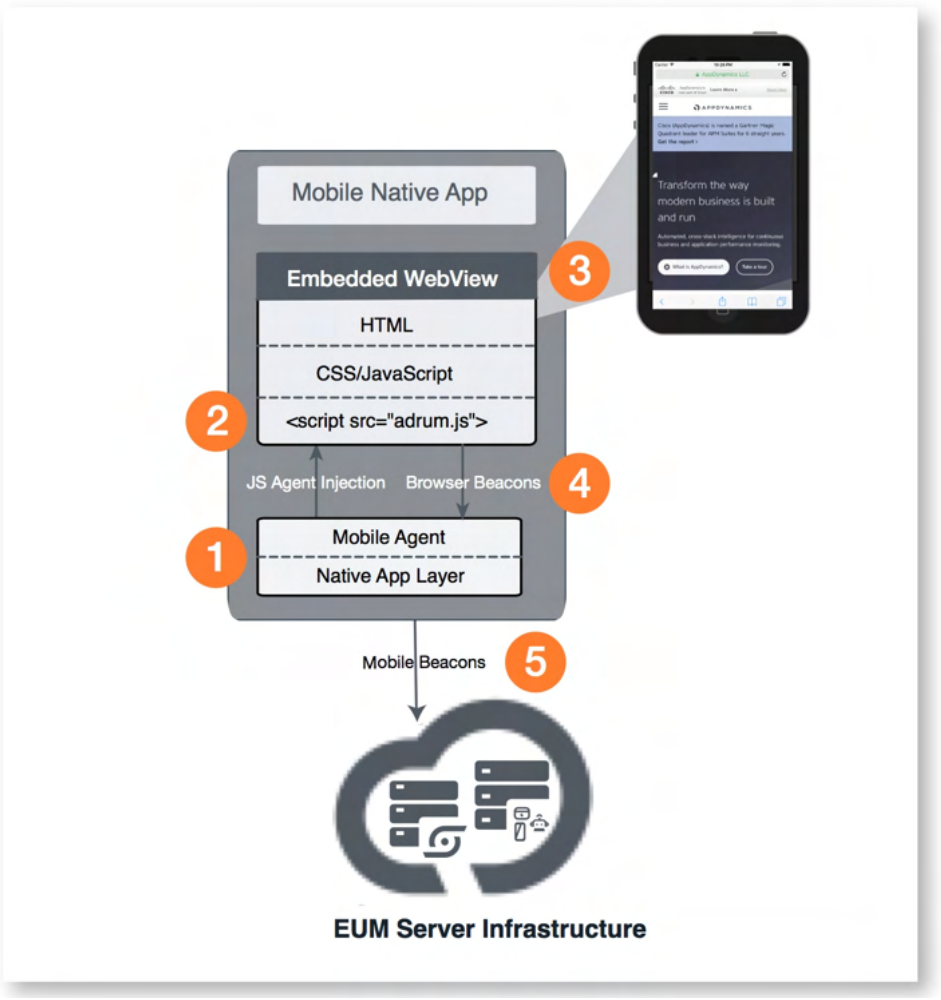
Native iOS Application	Native Android Application	Cordova-Based Applications
1. Instrument an iOS Application 2. Configure Hybrid Support for the iOS Application	1. Customize the Android Build 2. Instrument an Android Application 3. Configure Hybrid Support for Android Applications	1. Instrument a Cordova Application 2. Customize the Cordova Instrumentation

How it Works

For native Android/iOS applications, the Android Agent or iOS Agent will inject the JavaScript Agent into an Android WebView or iOS WKWebView. For Cordova-based applications, the AppDynamics plugin will inject the JavaScript Agent into the web application.

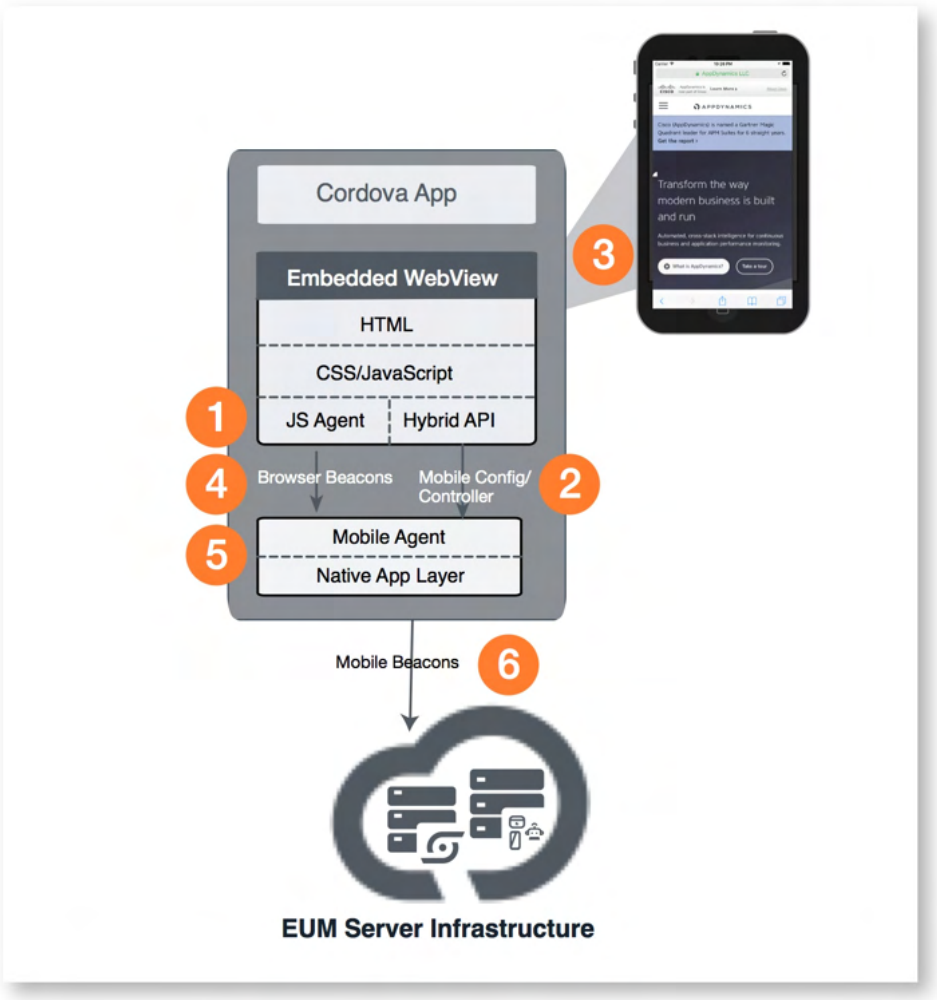
The diagrams below provide an overview of how Mobile RUM is used to monitor mobile native applications and Cordova-based applications.

Mobile Native Application Architecture



1. You instrument your mobile application with either the Android /iOS Agent.
2. When your mobile application instantiates a WebView or WKWebView instance, the Mobile Agent injects the JavaScript Agent into it.
3. The JavaScript Agent monitors the performance and WebView usage.
4. The JavaScript Agent reports the application activity by sending beacons to the Mobile Agent.
5. The Mobile Agent repackages the browser beacons into mobile beacons and sends the beacons to the EUM Server.

Cordova-Based Application Architecture



1. The AppDynamics Cordova plugin injects the JavaScript Agent into the Cordova WebView.
2. The Cordova-based application uses the plugin's JavaScript binding to configure the Mobile Agent.
3. The JavaScript Agent injected into the WebView monitors the application activity and performance.
4. The JavaScript Agent sends browser beacons to the Mobile Agent.
5. The Mobile Agent parses and converts the browser beacons into mobile beacons.
6. The Mobile Agent sends the mobile beacons to the EUM Server.

Mobile Agent and JavaScript Agent Compatibility

When hybrid app support is enabled, the Mobile Agents use an embedded JavaScript Agent to monitor mobile web pages. If you are already monitoring the same mobile web pages with a browser application, the JavaScript Agent version used to instrument the browser application may be different than the JavaScript Agent version used by the Mobile Agent. If the two versions of the JavaScript Agent are incompatible, the Mobile Agent will not be able to instrument web views or access event notifications from web views.

How to Ensure Compatibility

To ensure compatibility, browser applications should use JavaScript Agent version >= 4.5.4 and mobile applications should use Mobile Agent versions >= 4.5.5. When using compatible versions of the JavaScript Agents, the JavaScript Agent of the browser app will report beacons for a web page, and the embedded JavaScript Agent of the mobile app will be ignored and not report beacons because a web page cannot be instrumented with the JavaScript Agent multiple times.

> If you have disabled the Mobile Agent's hybrid app support, there will be no compatibility issues regardless of the JavaScript Agent version used by the browser application.

Troubleshooting Incompatible JavaScript Agent Versions

If you are using incompatible versions of the JavaScript Agents, do one of the following to correct the problem:

- Upgrade the JavaScript Agent that you are using to instrument the browser application.
- In the mobile instrumentation, disable the hybrid support (iOS/Android), so that the mobile web pages are only instrumented by the browser application instrumented with the older version of the JavaScript Agent.

Hybrid Application Network Requests

Network calls made in the context of the WebView for Cordova-based applications show up as Ajax calls from the JavaScript Agent.

iOS Data Collection Disclosure

With the iOS 14 release, Apple introduced a new data collection policy, available at https://developer.apple.com/app-store/user-privacy-and-data-use/ (the "Apple iOS 14 User Privacy and Data Use Policy"). This page describes how Apple's iOS 14 data use policy applies to AppDynamics.

Data Collection with Screenshots

When you use the Mobile Screenshots feature, the iOS Agent captures an image of the your user's screen and may send a copy of the image to AppDynamics' server for the Controller to visualize app usage. Such screenshots may capture anything your application displays, including any user data your user submits. It is your responsibility to review the data captured in the screenshots and determine where such captured data falls in Apple's categorization of types of disclosable data. See Data Collection Details.

Disable Mobile Screenshots

You can use `blockScreenshots()` to temporarily disable screenshots for any screen which may display such data. See Customize the iOS Instrumentation.

Data Collection Details

AppDynamics does not enable any targeted advertisements, data broker, or data combination as described in the Apple iOS 14 User Privacy and Data Use Policy. In addition, ID for Vendors (IDFV) as defined under such guidelines is not currently used in any AppDynamics Mobile features.

This table references https://developer.apple.com/app-store/app-privacy-details/ and describes how AppDynamics does, or does not collect data.

Apple Data Category	Apple Data Description	AppDynamics Data Collection
Contact Info	**Name** Such as first or last name	Subject to screen capture functionality
	Email Address Including but not limited to a hashed email address	
	Phone Number Including but not limited to a hashed phone number	
	Physical Address Such as home address, physical address, or mailing address	
	Other User Contact Info Any other information that can be used to contact the user outside the app	
Health and Fitness	**Health** Health and medical data, including but not limited to from the Clinical Health Records API, HealthKit API, MovementDisorderAPIs, or health-related human subject research or any other user provided health or medical data	Does not collect data
	Fitness Fitness and exercise data, including but not limited to the Motion and Fitness API	
Financial Info	**Payment Info** Such as form of payment, payment card number, or bank account number. If your app uses a payment service, the payment information is entered outside your app, and you as the developer never have access to the payment information, it is not collected and does not need to be disclosed.	Does not collect data
	Credit Info Such as credit score	
	Other Financial Info Such as salary, income, assets, debts, or any other financial information	
Location	**Precise Location** Information that describes the location of a user or device with the same or greater resolution as a latitude and longitude with three or more decimal places	Does not collect data
	Coarse Location Information that describes the location of a user or device with lower resolution than a latitude and longitude with three or more decimal places, such as approximate location services	

Sensitive Info	**Sensitive Info** Such as racial or ethnic data, sexual orientation, pregnancy or childbirth information, disability, religious or philosophical beliefs, trade union membership, political opinion, genetic information, or biometric data	Does not collect data
Contacts	**Contacts** Such as a list of contacts in the user's phone, address book, or social graph	Does not collect data
User Content	**Photos or Videos** The user's photos or videos	Subject to screen capture functionality
	Audio Data The user's voice or sound recordings	
	Gameplay Content Such as user-generated content in-game	
	Customer Support Data generated by the user during a customer support request	
	Other User Content Any other user-generated content	
Browsing History	**Browsing History** Information about content the user has viewed that is not part of the app, such as websites	Does not collect data
Search History	**Search History** Information about searches performed in the app	Does not collect data
Identifiers	**User ID** Such as screen name, handle, account ID, assigned user ID, customer number, or other user- or account-level ID that can be used to identify a particular user or account	Subject to screen capture functionality
	Device ID Such as the device's advertising identifier, or other device-level ID	Does not collect data
Purchases	**Purchase History** An account's or individual's purchases or purchase tendencies	Does not collect data
Usage Data	**Product Interaction** Such as app launches, taps, clicks, scrolling information, music listening data, video views, saved place in a game, video, or song, or other information about how the user interacts with the app	• Application life cycle events (start, relaunch, crash) • Views of transition events • Tabs • Clicks • Scrolling information • Activation/deactivation of text input fields
	Advertising Data Such as information about the advertisements the user has seen	Does not collect data
	Other Usage Data Any other data about user activity in the app	• Developer-specified breadcrumbs to track custom milestones
Diagnostics	**Crash Data** Such as crash logs	• Crash time • Crash stack traces
	Performance Data Such as launch time, hang rate, or energy use	• ANR (Application not Responding) with associated stack trace • Remaining storage, memory, and battery use
	Other Diagnostic Data Any other data collected for the purposes of measuring technical diagnostics related to the app	• Device type • OS version

Other Data	**Other Data Types** Any other data types not mentioned	• Network calls: duration, type (POST, GET, and so on), URL (parameters are stripped from URL prior to capture), content length, response content length, status codes.

Mobile Device Metric Collection

This page explains what mobile device metrics are collected, and how to enable collection and view device metrics in the Controller UI. To learn how Apple's iOS 14 data use policy applies to AppDynamics mobile device metric collection, see iOS Data Collection Disclosure.

Mobile agents can capture the following mobile device metrics:

- **Memory (RAM):** total memory of the device (i.e. 8GB total memory) and the resource consumption (i.e. 34% of device memory has been consumed)
- **Disk Space:** total storage capacity of the device (i.e. 64GB total storage capacity) and storage usage (i.e. 65% of device storage has been consumed)
- **Battery:** battery level of the device (i.e. include charging and lower power mode statuses)

For Android devices, the metrics reported in the UI might not exactly match the Android device metrics. Due to API limitations, the memory value that the Android Agent returns might not include segments on the device's reserved memory. For example, a device might have 64GB storage and 8GB memory, but the agent might report only 53GB storage and 6.5GB memory. This also varies based on device type and manufacturer.

Support

- Available for both SaaS and on-prem
- Controller 20.11.0 and later
- Android Agent 20.10.0 and later
- iOS Agent 20.10.0 and later

Enable and Configure Device Metric Collection

To configure device metric data collection in the Controller:

1. Go to a browser application.
2. In the left-hand panel, go to **Configuration** > **Mobile App Group Configuration** > **Settings**.
3. Under **Configure Mobile Device Metrics**, enable each option. It may take a few minutes for data to appear on the **Mobile App Dashboard**.
4. Set **Frequency of data collection** to how often device data is collected for the options you enabled. For example, if the frequency is set to 2, every 2 minutes the enabled device data is collected and shown in the dashboard widgets.

Warning

The maximum frequency is 2 minutes and collects the most data. However, sessions under 2 minutes or temporary metric spikes will not be captured.

Add Device Metric Widgets to the Mobile App Dashboard

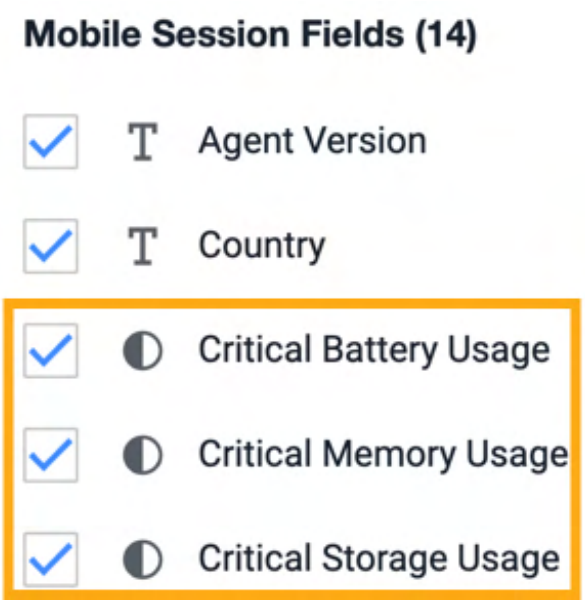
You can view device metric widgets on the **Mobile App Dashboard**. To add device metric widgets:

1. Go to the **Mobile App Dashboard**.
2. In the top-left corner, click **+**.
3. Add the widgets **Device Memory Capacity**, **Device Storage Capacity**, and **Device Battery Capacity**.
4. Click on a widget and view performance and sessions by battery, memory, and storage.

View Sessions with Device Metrics

On the **Sessions** dashboard, you can add columns to sort sessions for each device metric. To add device metric columns:

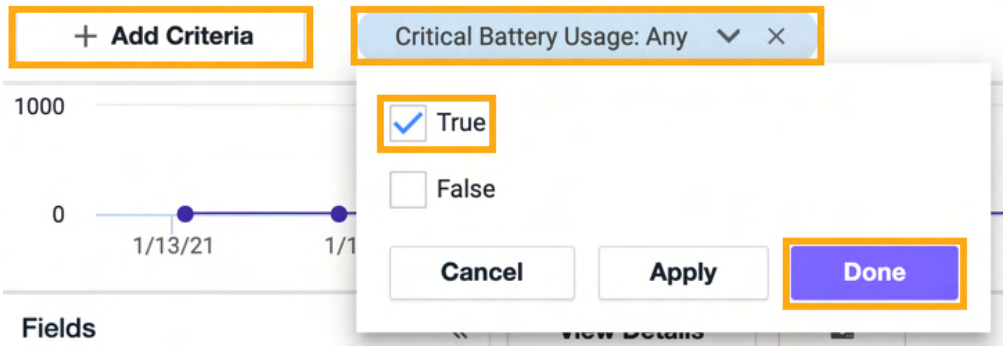
1. Go to **Mobile Session Fields**.
2. Check the fields **Critical Battery Usage**, **Critical Memory Usage**, and **Critical Storage Usage**.



Filter Sessions by Device Metric Values

You can filter sessions by critical device states you configured. For example, if you want to look at sessions with over 90% device storage used:

1. Go to **Sessions**.
2. Click **+ Add Criteria**.
3. Select "Critical Battery Usage: Any".
4. Check "True."

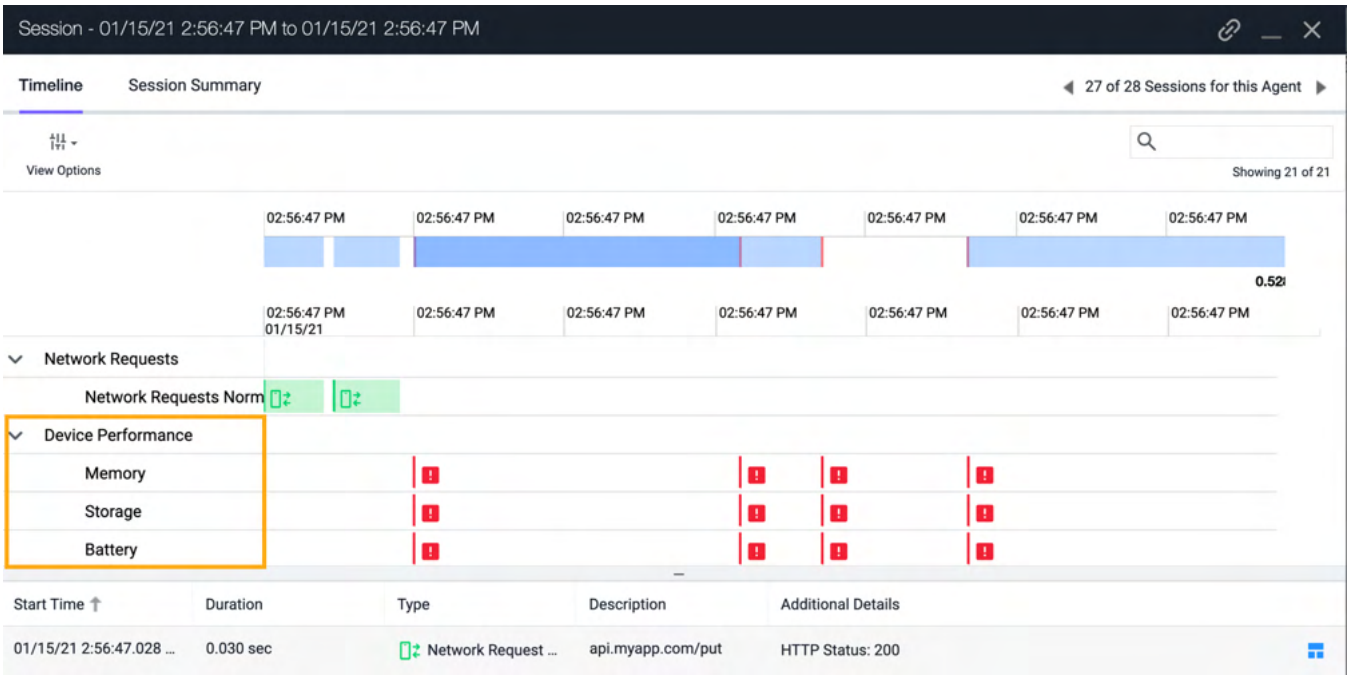


Set Critical Usage Thresholds

You can set the usage threshold to trigger a critical event. To configure what usage percentage triggers a critical event:

1. Go to **Configuration** > **Mobile App Group Configuration** > **Settings**.
2. Under **Configure Mobile Device Metrics**, enter a value for each device metric threshold. For example, if you want to capture battery usage below 50%, set **Enable Battery Usage and Capacity Metrics** to 50%. When that configured threshold is reached, the resource consumption (percentage value) is captured by the agent and shown in session data.

When you look at a mobile session, you can see the critical events you configured.


Session - 01/15/21 2:56:47 PM to 01/15/21 2:56:47 PM
Timeline
Session Summary
27 of 28 Sessions for this Agent
View Options
Showing 21 of 21
02:56:47 PM
01/15/21
Network Requests
Network Requests Norm
Device Performance
Memory
Storage
Battery
Start Time
Duration
Type
Description
Additional Details
01/15/21 2:56:47.028 ...
0.030 sec
Network Request ...
api.myapp.com/put
HTTP Status: 200

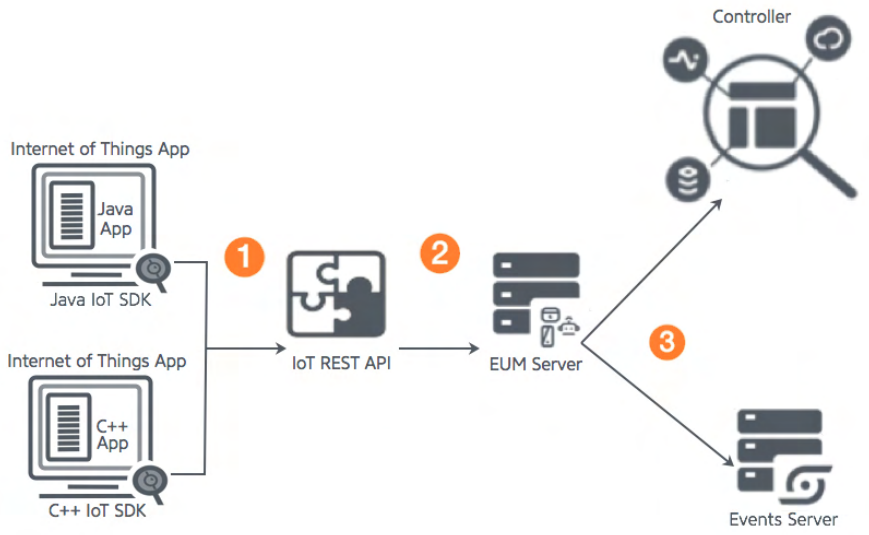
IoT Monitoring

Deployment Support

SaaS

AppDynamics IoT Monitoring enables you to track and understand the transactions of your IoT applications. Because IoT devices are diverse, both in terms of the platforms they use and their business function, AppDynamics has developed a REST API in addition to language SDKs to provide the maximum flexibility for reporting IoT data. This API can be used from any device that supports HTTPS and is connected to the Internet.

IoT Monitoring requires application developers to instrument their code. To make this process easier, AppDynamics has developed C/C++ and Java SDKs, so that developers using the platforms supporting these languages can leverage the features of the SDK instead of using the REST API. The IoT SDKs (1) use the REST APIs to report IoT Data to the EUM Server (2), where the data is aggregated and made available to the AppDynamics Controller and the Events Service (3) as shown in this diagram:



Benefits of IoT Monitoring

IoT Monitoring enables you to capture and diagnose network requests, errors, and custom application and domain-specific events for applications running on devices that impact business objectives.

With IoT Monitoring, you can monitor the following for devices:

- Availability
- Network Performance:
  - Slow interaction
  - Latency between the device and backend services
- Usage: User and system behavior and patterns
- Errors and Exceptions

IoT Monitoring Use Cases

IoT applications are typically embedded applications running on connected devices. This section focuses on how IoT Monitoring can collect and report data for the most common categories of IoT applications.

In the tabs below, you can view these IoT app categories and see how a typical example of each would use the three supported IoT Monitoring events to report data.

IoT SDKs

IoT Monitoring provides SDKs for C/C++ and Java languages. Some of the features of IoT SDKs are:

- APIs for reporting instrumentation data through custom events, error events, and network requests.
- Serialization of all events into JSON format before sending them to EUM Server
- API to plugin application HTTPS stack into SDK which will be used to communicate with EUM Server
- Business Transaction (BT) correlation
- Auto-disable when monitoring is turned off from the Controller UI

For languages other than C++ and Java, you can use the REST API to send custom events, error events, and network requests.

IoT Monitoring UI

IoT Monitoring also provides the following features through the Controller UI:

- Dashboards for monitoring devices, network requests, as well as errors and exceptions
- Predefined and custom widgets to visualize reported application data
- Analytics for device application data through AppDynamics Application Analytics
- Configuration for naming and excluding network requests

IoT Monitoring Workflow

The IoT SDKs and REST API enable you to report application data for a large number of different kinds of devices. This flexibility allows for a great deal of customization in the kind of the data you report, the quantity of data you report, and the timing of when the data is sent to the EUM Server.

To monitor customized data, you will need to better understand the data generated by your application and your device instrumentation. The goal of this page is to guide you through the process of understanding your application data, instrumenting your device application, and creating custom dashboards, so you can monitor device performance and activity.

We recommend that you follow this procedure as a guideline to monitor your devices:

1. Define your monitoring goals.
2. Identify what data needs to be captured.
3. Determine which events to use to report the data.
4. Obtain an EUM App Key.
5. Report events.
6. Monitor the availability, usage, and performance of your devices.
7. Build widgets.
8. Diagnose problems through filters.
9. Improve monitoring and troubleshoot.

Define Monitoring Goals

AppDynamics IoT Monitoring enables you to monitor the availability, performance, and usage of your device. You should devise a plan that prioritizes your monitoring goals, considering the needs of all stakeholders (developers, DevOps, business units), and lists the resources that need to be monitored. Also, you should consider who will be instrumenting the device application, who will be performing the monitoring, and the parties who should be notified when something goes wrong.

Identify Data to Capture

Next, you will need to analyze your device to identify what application data can be used to meet your monitoring goals. For example, if your device is a car, you would want to look at network requests to check device availability or report custom data for the wear and tear on its components or road conditions.

Determine Which Event Types to Use to Report Data

You report app data through the three types of events shown in the table below. Based on this information, map the most suitable event types to the data you identified and your monitoring goals.

Event Type	Monitoring Objective
Custom	Usage
	Business
Network Request	Availability
	Performance
	Usage

Error	Performance

Obtain an EUM App Key

Follow the instructions provided in Get an EUM App Key.

Report Events

You can use the IoT SDKs or REST API to capture and report key data points using one of the event types. If you are using a language other than C/C++ or Java, use the IoT REST API, which gives you the flexibility to report data from any platform.

Follow the getting started instructions to learn how to install the SDK and instrument your IoT application:

- Instrument Applications with the IoT C/C++ SDK
- Instrument Applications with the IoT Java SDK

Monitor Your IoT Application

To monitor your IoT applications, you will need to understand the function and purpose of the three IoT dashboards. See Monitor Applications with the IoT Dashboards for a brief introduction of each dashboard and usage instructions.

Build Widgets Based on Your Data Models

Each dashboard has a set of predefined widgets for basic performance and activity monitoring. You will need to create your own widgets, however, to optimize the monitoring capabilities of **IoT Monitoring**. Based on the events you reported, create custom widgets to monitor device activity and performance. See the section Build Custom Dashboard Widgets to learn how to build widgets with the **IoT Widget Builder Wizard**.

Diagnose Problems

When you do discover deviations from expected behavior, you can then add criteria to filter results. This enables you to narrow the results to certain devices or possible causes.

Troubleshoot Issues and Improve Monitoring

For performance issues, you can open the device details to analyze errors and slow network requests. The device details allow you to trace the error event you are interested in and even download the stack trace. The developer can then use this information to determine the root cause of the problem and devise a possible solution. Going back to our point-to-sale device example, you might discover that most of the failed payments were due to network request errors.

You can use the usage information to improve your monitoring or the devices themselves. For example, an inventory management application indicates that several stores sell a smaller quantity of a certain product. It turns out that this product is often out of stock at these stores. You could improve the monitoring of your IoT app by using custom events to report when a product is sold out and then creating custom widgets to let you know which stores need to restock.

IoT Analytics

All IoT data is processed and stored by the AppDynamics Platform Events Service. A separate product component, AppDynamics Analytics, has an event type called Connected Device Data. This event type is based on the same Events Service and uses the same data. Viewing this event type from the Analytics UI offers additional capabilities, including:

- Additional predefined widgets, such as the funnel widget
- ADQL for searching the data
- Creating custom widgets
- Manipulating multiple dashboard types
- Longer retention time for data storage

IoT Analytics does *not* require a license separate from the IoT Monitoring license.

To view Analytics data for connected devices, users need **Connected Devices Permissions** for the application (identified by a specific App Key).

IoT Licenses and Limits

Currently, AppDynamics only offers the **IoT PRO** license. To learn about the license entitlements and restrictions, contact your AppDynamics account representative.

Set Up and Access IoT Monitoring

This describes when you should use an IoT SDK or the IoT REST API, get an EUM App Key, and direct you to instructions for capturing and reporting data.

SDKs Versus REST API

The IoT SDKs use the REST API to transmit data to the EUM Server. By handling HTTP requests, serializing JSON, and managing events in memory, the SDKs make it easier to capture and report data, so you can focus on instrumenting your application. For platforms other than C++ or Java, you can take advantage of the wide variety of HTTP(S) and JSON libraries to use the IoT REST API to report data.

Get an EUM App Key

To get an EUM App Key, you need to either create an IoT app manually or use the **Getting Started Wizard**, which is recommended if you are a new user. You will need the EUM App Key to use the IoT REST API or IoT SDKs to report device information and events to the EUM Server. Multiple IoT applications can share the same key.

Create an IoT Application with the Getting Started Wizard

From the Controller:

> IoT applications created from existing applications cannot be deleted. You would need to delete the existing IoT application that was used to create the application.

Manually Create an IoT Application

From your Controller:

1. Click **User Experience** from the top navigation bar.
2. Select the Connected Devices tab.
3. Click **Get Started.**
4. From the **Create Application** dialog, select **Create an Application manually**.
5. Enter a name for your IoT app.
6. Click **OK**.
7. From the **Connected Devices** tab, select IoT app and click  to copy the App Key to your clipboard.

Report IoT Data

Based on the method you plan on reporting data, use your App Key and follow the instructions for one of these tutorials:

- Instrument Applications with the IoT C/C++ SDK
- Instrument Applications with the IoT Java SDK
- Instrument Applications with the IoT REST APIs

Correlate Business Transactions for IoT Monitoring

Related pages:

- Correlate Business Transactions for EUM Monitoring
- Business Transactions

You can correlate IoT network request events with business transactions. The correlation is made between beacons containing network request event information and instances of business transactions (transaction snapshots).

IoT Network Requests Correlation

This procedure outlines how IoT applications correlate network requests with business transactions:

1. The IoT application attaches AppDynamics HTTP headers to a network request to the app server agent. These AppDynamics HTTP headers instruct the app server agent to send back correlation headers.
2. The app server agent:
    - Sends HTTP response headers identifying the business transaction to the IoT application.
    - Aggregates backend metrics and sends them along with the business transaction identifiers to the Controller. This serves as the content for the transaction snapshot.
3. The IoT Agent sends business transaction identifiers (from the HTTP response headers) as part of IoT network request event to the EUM Server. This serves as the content for the network request.
4. The Controller fetches the events and business transaction identifiers from the EUM Server. These business transaction identifiers are used to correlate the network request with the transaction snapshots.

Correlate IoT Network Request Events with Business Transactions

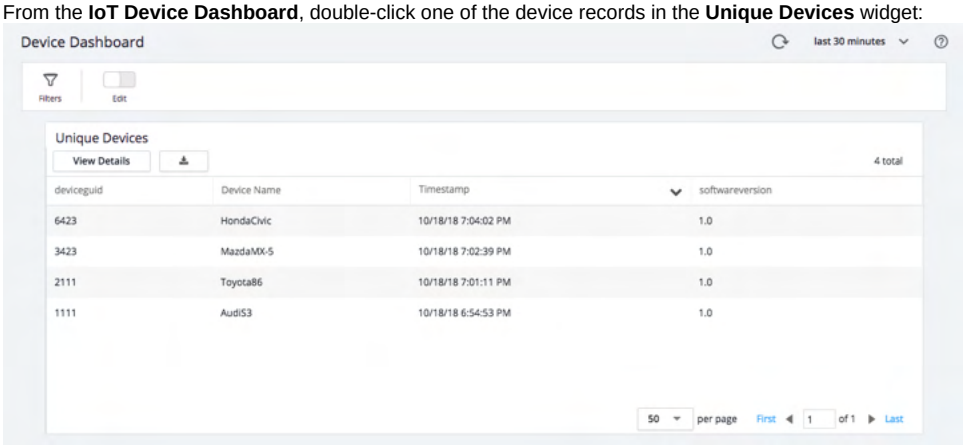
Use the IoT SDKs or the IoT REST API to send the returned response headers from the business application with beacons to report IoT network request events and correlate those events with the business transaction.

For correlation instructions, see "Correlate Business Transactions with Network Requests" for either the IoT C/C++ SDK, IoT Java SDK, or the IoT REST API.

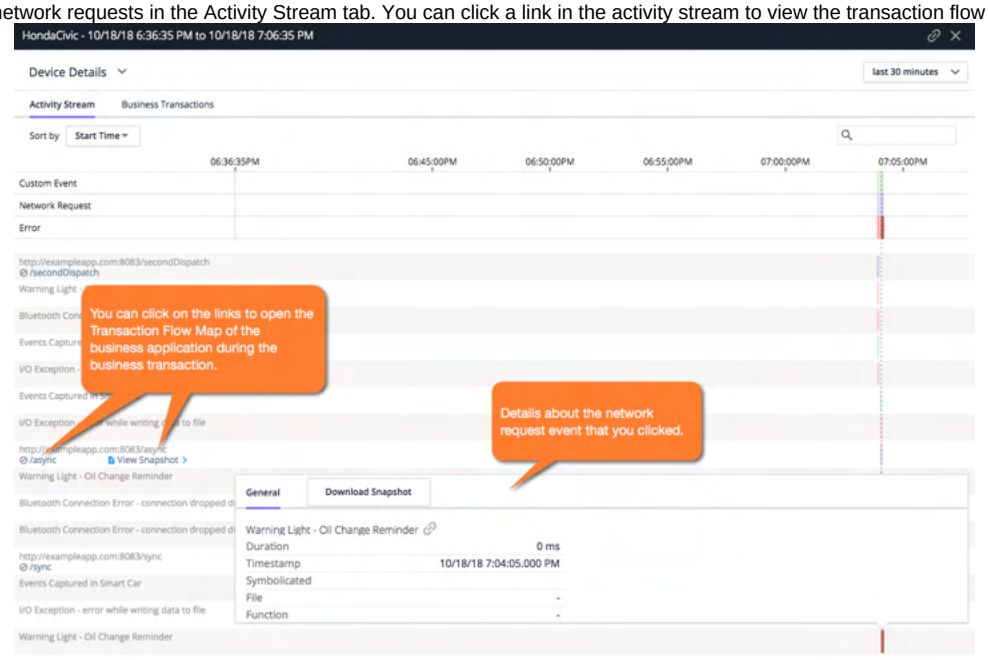
View Business Transactions Correlated with IoT Applications

There are several ways to navigate from a network request snapshot to its correlated business transaction. The following steps show you one possible way.

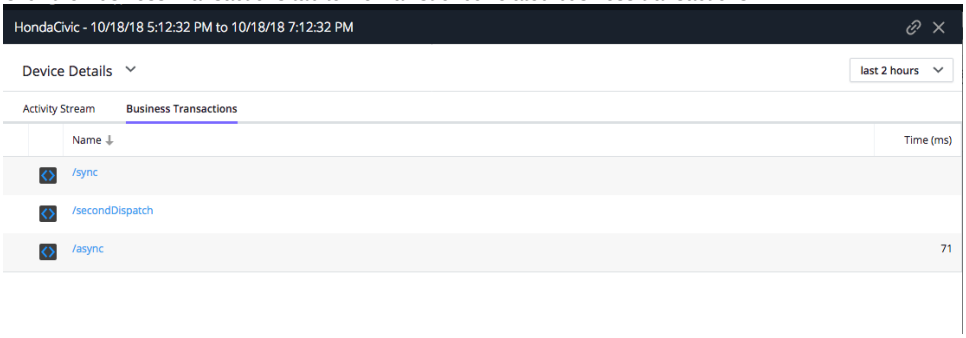
1. From the **IoT Device Dashboard**, double-click one of the device records in the **Unique Devices** widget:



2. You should now see a list of network requests in the Activity Stream tab. You can click a link in the activity stream to view the transaction flow map for a business transaction.



3. Click the Business Transactions tab to view a list of correlated business transactions.



4. You can then click a business transaction to view the **Transaction Flow Map** shown below**,** which provides a visual representation of the components and activities of the business application during the correlated business transaction.

Instrument Applications with the IoT C and C++ SDK

The IoT C++ SDK provides APIs to instrument C++ applications running on connected devices such as industrial or home gateways, points of sale, smart TVs or car infotainment systems. This getting started will show you how to install the C++ SDK and instrument your IoT application.

Follow these steps to get your EUM App Key and instrument your IoT C/C++ apps.

> If you have ANSI-C application or if your platform is not Linux x86, please contact your AppDynamics account representative.

Learn About the C/C++ SDK

You should know that the C++ SDK:

- Operates within the application thread and doesn't spawn any new threads.
- Keeps all the event data in memory and doesn't persist on disk.
- Provides an API to register for the network interface.
- Uses application's HTTPS stack to communicate with the EUM Server.
- Provides an API to fetch SDK log messages. Application developers have to manage logs by writing the log messages to `stderr` or to a log file.
- Uses the open-source json-c library that is statically linked.
- Makes synchronous blocking API calls that are not thread-safe. Application developers are responsible for making thread-safe calls.

Review the Requirements

Before you begin, make sure you meet these requirements:

- GNU C++ Compiler (g++) version 4.2 for 32/64-bit architectures
- Any Linux distribution based on glibc >= 2.20
- HTTPS stack for sending beacons to the EUM Cloud
- EUM App Key

Get the IoT C++ SDK

You can get the C++ SDK by cloning or downloading the IoT C++ SDK from GitHub. Follow the instructions given in Installation to build the IoT C++ SDK.

Upgrade the IoT C++ SDK

From the root directory of your clone of the IoT C++ SDK from GitHub:

- Update the repository: `$ git pull origin master`
- Follow the instructions given in Installation to rebuild the IoT C++ SDK.

Install the C++ SDK in Your Application

The C++ SDK is packaged as a tar zip file and contains the following:

- `include` - directory containing headers for the public API for use with the C++ SDK
- `lib` - the directory containing the shared object files for the C++ SDK

Add the SDK Headers

Copy or move the `include` directory, which contains the SDK header files into your application directory, and include it in your code to access the SDK APIs.

```
#include "appd_iot_interface.h"
  ....
{
```

Initialize the SDK

You must initialize the C++ SDK by providing the SDK and device configuration as input parameters and then calling the function `appd_iot_init_sdk` as shown below. The SDK configuration takes in parameters for the app key, log level, and the EUM Collector URL. The SDK uses the EUM Collector URL to send data to the EUM Server. The device configuration contains information to identify a unique device.

```
#include "appd_iot_interface.h"
  ....
{
  // Declare config variables for the SDK and device.
  appd_iot_sdk_config_t sdkcfg;
  appd_iot_device_config_t devcfg;
  appd_iot_init_to_zero(&sdkcfg, sizeof(sdkcfg));
  appd_iot_init_to_zero(&devcfg, sizeof(devcfg));

  // Set the initialization configurations for the SDK
  sdkcfg.appkey = "<EUM_APP_KEY>";

  // Set the device configurations
  devcfg.device_id = "1111";
  devcfg.device_type = "SmartCar";
  devcfg.device_name = "AudiS3";

  // Initialize the instrumentation
  appd_iot_init_sdk(sdkcfg, devcfg);
}
```

Register Network Interface

The SDK needs an HTTPS interface to send events to the EUM Server. The application developer has to provide callback functions for the SDK to execute the HTTPS request. Refer to Run the Sample Application for a sample network interface implementation using `libcurl`.

```
#include "appd_iot_interface.h"
  ....
{
  appd_iot_http_cb_t http_cb;

  //Callback function triggered by SDK to send http request and receive http response
  http_cb.http_req_send_cb = &your_network_interface_send_cb;

  //Callback function triggered by SDK to indicate completion of http response processing
  http_cb.http_resp_done_cb = &your_network_interface_resp_done_cb;


  //register http interface callbacks
  appd_iot_register_network_interface(http_cb);

  ...
}
```

Add and Send Events

To understand the different types of events, you will work with the sample smart car IoT application given in the sections below.

Custom Events

Custom event to capture technical stats of a "SmartCar".

```
#include "appd_iot_interface.h"
  ....

{
  appd_iot_custom_event_t custom_event;
  appd_iot_init_to_zero(&custom_event, sizeof(custom_event));

  custom_event.type = "SmartCar Stats";
  custom_event.summary = "Technical Stats of SmartCar";
  custom_event.timestamp_ms = ((int64_t)time(NULL) * 1000);
  custom_event.data = (appd_iot_data_t*)calloc(2, sizeof(appd_iot_data_t));
  appd_iot_data_set_integer(&custom_event.data[0], "Speed mph", 65);
  appd_iot_data_set_double(&custom_event.data[1], "Oil Temperature", 220);

  appd_iot_add_custom_event(custom_event);
  free(custom_event.data);

  ....

  appd_iot_send_all_events();
}
```

Network Request Events

Network Request event to capture the performance of an HTTPS call to get weather information.

```
#include "appd_iot_interface.h"
  ....

{

  appd_iot_network_request_event_t network_event;
  appd_iot_init_to_zero(&network_event, sizeof(network_event));

  network_event.url = "https://apdy.api/weather";
  network_event.resp_code = 202;
  network_event.duration_ms = 10;
  network_event.req_content_length = 300;
  network_event.req_content_length = 100;
  network_event.timestamp_ms = ((int64_t)time(NULL) * 1000);
  network_event.data = (appd_iot_data_t*)calloc(1, sizeof(appd_iot_data_t));
  appd_iot_data_set_string(&network_event.data[0], "city", "San Francisco");

  appd_iot_add_network_request_event(network_event);
  free(network_event.data);

  ....
  appd_iot_send_all_events();
}
```

Error Events

The Error event below is used to capture Bluetooth errors in the SmartCar app.

```
#include "appd_iot_interface.h"
  ....
{

  appd_iot_error_event_t error_event;
  appd_iot_init_to_zero(&error_event, sizeof(error_event));

  error_event.name = "Bluetooth Connection Error";
  error_event.message = "connection dropped due to bluetooth exception";
  error_event.severity = APPD_IOT_ERR_SEVERITY_CRITICAL;
  error_event.timestamp_ms = ((int64_t)time(NULL) * 1000);
  error_event.data = (appd_iot_data_t*)calloc(1, sizeof(appd_iot_data_t));
  appd_iot_data_set_integer(&error_event.data[0], "Bluetooth Error Code", 43);

  appd_iot_add_error_event(error_event);
  free(error_event.data);

  ....


  appd_iot_send_all_events();
}
```

Correlate Business Transactions with Network Requests (Optional)

To correlate business transactions (BTs) with network requests, you need to instrument a business application and enabled business transactions in the Controller UI. See Correlate Business Transactions for IoT Monitoring to learn more.

The steps below show you how to get the BT response headers and use them to correlate the BT with an IoT Network Request event.

1. Set the AppDynamics HTTP headers `ADRUM` and `ADRUM_1` as part of a network request to your business application.

```
/* Initialize all the data structures for the request and response. */
appd_iot_http_req_t http_req;
appd_iot_http_req_t http_resp;

/* Initialize the request and response. */
appd_iot_init_to_zero(&http_req, sizeof(http_req));
appd_iot_init_to_zero(&http_resp, sizeof(http_resp));

/* Provide the URL to your instrumented business app that is enabled for business transaction
correlation. */
http_req.url = "<url-to-your-business-app-enabled-for-bt>";


/* Add your other HTTP request parameters here:
...
*/
/* Call the SDK method to get the headers for ADRUM and ADRUM_1. */
const appd_iot_data_t* correlation_headers = appd_iot_get_server_correlation_headers();

for (size_t i = 0; i < APPD_IOT_NUM_SERVER_CORRELATION_HEADERS; i++)
{
  appd_iot_data_set_string(&http_req.headers[i], correlation_headers[i].key, correlation_headers[i].
strval);
}

/* Make the request, and assign the response to a variable. */
http_resp = http_curl_req_send_cb(&http_req);
```

2. The call will return response headers (i.e., `ADRUM_*`) that contain information for correlating the business transaction. If you were to print these BT response headers, you would see something like the following:

```
ADRUM_0: clientRequestGUID:0f5c7602-9b69-4e40-85a6-e0abf288accf
ADRUM_1: globalAccountName:eum-mobile_4debdbad-3f8e-4f6d-8faf-e5f5781ec0d7
ADRUM_2: btId:3867
ADRUM_3: serverSnapshotType:f
ADRUM_4: btDuration:829
```

3. Add these BT response headers to the network event that you send to the EUM Server:

```
/* Create a network event to report to the EUM Server. */
appd_iot_network_request_event_t network_event;
appd_iot_init_to_zero(&network_event, sizeof(appd_iot_network_request_event_t));

/* Add information about the network event that you want to report. */
network_event.url = "<url-to-your-business-app-enabled-for-bts>";
network_event.resp_code = http_resp->resp_code;

/* Assign the returned BT response headers from the call to the business app to the headers of the
request. */
network_event.resp_headers = http_resp->headers;

// Add the network event to beacon to send to the EUM Server. */
appd_iot_add_network_request_event(network_event);
appd_iot_send_all_events();
```

4. In the Controller UI, you should be able to view the correlated business transaction in the **Device Details** dialog.

Compile and Run Your App with the SDK Library File

1. Compile your program. For example, if the driver file is `main.cpp`:

```
$ g++ -c main.cpp -I<appd_iot_sdk_dir>/include
```

2. Create the binary with the object code of your application and linking AppDynamics IoT C++ SDK library.

   You can combine steps 1 and 2 into one step as below:

```
$ g++ main.cpp -o main -I<appd_iot_cpp_sdk_dir>/include -L<appd_iot_cpp_sdk_dir>/lib -lappdynamics_iot
```

3. Set the environment variable `DYLD_LIBRARY_PATH` to the `PATH` where the SDK library is installed. This will let dynamic linker know the directory to search for shared libraries.
4. Run your program. For example:

```
$ ./main
```

Verify the Instrumentation in the Controller UI

See Confirm the IoT Application Reported Data to the Controller to verify the instrumentation.

Customize the IoT C++ Instrumentation (Optional)

You can further customize the IoT C++ instrumentation using the IoT C++ SDK. See the latest IoT C++ SDK documentation or the previous versions listed below:

- https://sdkdocs.appdynamics.com/javadocs/iot-cpp-sdk/4.5/4.5.0/
- https://sdkdocs.appdynamics.com/javadocs/iot-cpp-sdk/4.5/4.5.1/
- https://sdkdocs.appdynamics.com/javadocs/iot-cpp-sdk/4.5/4.5.2/
- https://sdkdocs.appdynamics.com/javadocs/iot-cpp-sdk/4.5/4.5.4/

Run the Sample C++ Application

The sample C++ application sends sample data for Custom, Network Request, and Error events. The data mocks a smart car application, capturing usage information, network performance, and errors.

To run the sample app, follow the instructions given in Sample Application using IoT C++ SDK.

Instrument Applications with the IoT Java SDK

The IoT Java SDK can be included in IoT Java applications running on an edge device like gateways, points of sale, car infotainment centers. This getting started will show you how to install the Java SDK and instrument your IoT application.

Follow these steps to get your EUM App Key and instrument your IoT C/C++ apps.

This Java SDK differs from the AppDynamics Java Agent in that it is a very lightweight library specially designed for lower-end devices. A lot of flexibility is also built into the way event information can be extended and the instrumentation code configured as well as controlled.

Review the Requirements

Before you begin, make sure you meet these requirements:

- A device running one of the following version of Java Runtime:
    - Java SE 7
    - Java SE Embedded 7
    - Java SE 8
    - Java SE Embedded 8
- HTTPS interface to send beacons to the EUM Server
- EUM App Key

Get the IoT Java SDK

You can get the Java SDK by cloning or downloading the IoT Java SDK from GitHub. Follow the instructions given in Build the SDK to build the IoT Java SDK.

If you're using the IntelliJ IDE, add the file `lib/appd-iot-sdk.jar` to your project by following the instructions given in Working with module dependencies. Confirm that the JAR file shows up under **External Projects** in your IntelliJ project.

Upgrade the IoT Java SDK

From the root directory of your clone of the IoT Java SDK from GitHub:

- Update the repository: `$ git pull origin master`
- Follow the instructions given in Build the SDK to rebuild the IoT Java SDK.

Add the SDK Dependencies to the Gradle Configuration

Add the following to your `build.gradle` file:

```
dependencies {
   runtime group: 'org.slf4j', name: 'slf4j-api', version: '1.7.25'
   runtime group: 'com.google.guava', name:'guava', version:'18.0'
   runtime group: 'com.google.code.gson', name: 'gson', version: '2.8.0'
 }
```

Add the Instrumentation Code

Import the IoT SDK

In your application file, add the `import` statement that includes the Java IoT SDK:

```
import com.appdynamics.iot.Instrumentation;
```

Configure the IoT Java Agent

Configure the instrumentation by providing the EUM App Key and the URL to the EUM Collector. If the EUM Collector URL is not specified, the default SaaS Collector URL is used.

```
import com.appdynamics.iot.AgentConfiguration;
AgentConfiguration.Builder agentConfigBuilder = AgentConfiguration.builder();
AgentConfiguration agentConfig = agentConfigBuilder
        .withAppKey(<EUM_APP_KEY>)
        .build();
```

Set Device Info

You are required to set the name and ID for devices. The name should consist of a short string that identifies the type and model of the device, such as "EV Model 3" or "Thermostat Model Star7". The device ID must be a unique identifier for the device, such as a UUID, the VIN number of a car, or the MAC address of the device.

The example code below sets the device ID to a random UUID and the name to "Smart Shelf".

```
import java.util.UUID;
import com.appdynamics.iot.DeviceInfo;
...
DeviceInfo.Builder deviceInfoBuilder = DeviceInfo.builder("Smart Shelf P1", UUID.randomUUID().toString());
DeviceInfo deviceInfo = deviceInfoBuilder.withDeviceName("Smart Shelf").build();
```

Set Version Info

You can set the versions for the firmware, hardware, OS, and software as shown below.

```
import com.appdynamics.iot.VersionInfo;
...
VersionInfo.Builder versionInfoBuilder = VersionInfo.builder();
VersionInfo versionInfo = versionInfoBuilder
 .withFirmwareVersion("2.3.4")
 .withHardwareVersion("1.6.7")
 .withOsVersion("8.9.9")
 .withSoftwareVersion("3.1.1").build();
```

Initialize the Agent

To initialize the agent, pass the `AgentConfiguration` object, the `DeviceInfo` object, and the `VersionInfo` object to the `start` method:

```
Instrumentation.start(agentConfig, deviceInfo, versionInfo);
```

Build and Run the Application

Use your favorite Java IDE or CLI environment to build and run the application. Note that the AppDynamics IoT Java SDK needs to be the in build and runtime classpath.

For instructions on adding libraries to the classpath:

- IntelliJ IDE: Creating a library
- Eclipse: Classpath Variables
- Linux/Mac/Windows: PATH and CLASSPATH

For instructions to build and run the app:

- IntelliJ: Building and Running the Application
- Eclipse: Running your programs
- Gradle: Building Java Projects with Gradle

Add and Send Events

The following sections will show you how to create and send the supported events: Custom, Network Request, and Error.

Create a Basic Custom Event

A Custom event can be used to report any performance, device, or business logic data. It is the most general, configurable and flexible data type available.

The Custom event builder takes two required parameters.

- Event Type: A short human-readable description of the event, such as "FL Pressure Drop".
- Description: A string describing the event, such as "Front Left Tire Pressure Drop".

To make reporting this event meaningful, it is recommended that you provide a timestamp and at least one kind of datatype.

1. Create a basic custom event.

```
import com.appdynamics.iot.events.CustomEvent;
...
CustomEvent.Builder builder = CustomEvent.builder("FL Pressure Drop", "Front Left Tire Pressure Drop");
long eventStartTime = System.currentTimeMillis();
long duration = 6000;
builder.withTimestamp(eventStartTime).withDuration(duration);
builder.addLongProperty("PSI Drop", 37);
CustomEvent customEvent = builder.build();
```

   Additional information can be added to the `CustomEvent`. For details, see the `CustomEvent` class in the latest Java IoT SDK documentation.
2. Add the custom event to the instrumentation (this adds it to the in-memory buffer).

```
Instrumentation.addEvent(customEvent);
```

3. Send all the events to the EUM Server. This is a blocking call, so the application can send it on a separate thread as shown above.

```
Instrumentation.sendAllEvents();
```

Send a Network Event

1. Report a Network Request Event using the `HttpRequestTracker` class. This call automatically adds an event to the in-memory buffer, so you need to explicitly import the class.

```
import com.appdynamics.iot.HttpRequestTracker;
...
String url = "http://ip.jsontest.com/?callback=showMyIP";
// Add a Network Event
try {
    URL thisUrl = new URL(url);
    // [AppDynamics Instrumentation] Get a Tracker
    HttpURLConnection con = (HttpURLConnection) thisUrl.openConnection();
    final HttpRequestTracker tracker = Instrumentation.beginHttpRequest(thisUrl);
    con.setRequestMethod("POST");
    con.setRequestProperty("Accept-Language", "en-US,en;q=0.5");
    int responseCode = con.getResponseCode();
    con.setDoInput(true);
    con.setDoOutput(true);
    DataOutputStream wr = new DataOutputStream(con.getOutputStream());
    wr.flush();
    wr.close();
    System.out.println("Response Code :" + responseCode);
    // [AppDynamics Instrumentation] Retrieve the headers from the response
    Map<String, List<String>> headerFields = null;
    System.out.println("Sending 'POST' request to URL :" + url);
    BufferedReader in;
    String inputLine;
    new InputStreamReader(con.getErrorStream()));
    if (responseCode >= 200 && responseCode < 300) {
        in = new BufferedReader(new InputStreamReader(con.getInputStream()));
    } else {
        in = new BufferedReader(
    }
    StringBuffer response = new StringBuffer();
    if (headerFields != null && headerFields.size() > 0){
        while ((inputLine = in.readLine()) != null) {
           response.append(inputLine);
        }
        in.close();
        // [AppDynamics Instrumentation] Initiate adding NetworkRequestEvent
        if (responseCode >= 200 && responseCode < 300) {
            tracker.withResponseCode(responseCode).withError(response.toString()).reportDone();
                   .withResponseHeaderFields(headerFields)
                   .reportDone();
        } else {
            tracker.withResponseCode(responseCode).reportDone();
        }
    } else {
        tracker.withResponseCode(responseCode)
    }
// End: Add for AppDynamics Instrumentation - Initiate adding NetworkRequestEvent
} catch (MalformedURLException e) {
    e.printStackTrace();
} catch (Exception ex) {
    ex.printStackTrace();
}
```

2. Send all the events to the EUM Server. This is a blocking call. The application can send it on a separate thread. The recommendation is to batch a number of events together before calling the `sendAllEvents` method.

```
Instrumentation.sendAllEvents();
```

Send an Error Event

1. Report an Error Event using the API.

```
try {
    //Force creating an exception
    float f = (5 / 0);
} catch (Throwable t) {
    Instrumentation.addErrorEvent(t, Instrumentation.Severity.ALERT);
}
```

2. Send all the events to the EUM Server. This is a blocking call. The application can send it on a separate thread.

```
Instrumentation.sendAllEvents();
```

Verify the Instrumentation in the Controller UI

See Confirm the IoT Application Reported Data to the Controller to verify the instrumentation.

Correlate Business Transactions with Network Requests (Optional)

To correlate business transactions (BTs) with network requests, you need to have instrumented a business application and enabled business transactions in the Controller UI. See Correlate Business Transactions for IoT Monitoring to learn more.

The steps below show you how to get the BT response headers and use them to correlate the BT with an IoT Network Request event.

1. Make a network request that includes the AppDynamics HTTP request headers `ADRUM` and `ADRUM_1` to one of your business applications:

```
import com.appdynamics.iot.HttpRequestTracker;
...
// Create a network request to the business app.
String url = "<url_to_business_application>";
URL thisUrl = new URL(url);
[AppDynamics Instrumentation] Get a Tracker
HttpURLConnection con = (HttpURLConnection) thisUrl.openConnection();
final HttpRequestTracker tracker = Instrumentation.beginHttpRequest(thisUrl);
con.setRequestMethod("POST"); // Some HTTP method: GET, POST, PUT...

// Add the AppDynamics HTTP headers ADRUM and ADRUM_1 to the request.
con.setRequestProperty("ADRUM", "isAjax:true");
con.setRequestProperty("ADRUM_1", "isMobile:true");

// Make the request to your business app.
con.setDoInput(true);
```

2. The call will return response headers that contain information for correlating business transactions. If you were to print these BT response headers, you would see something like the following:

```
{
  ADRUM_1=[globalAccountName:customer1_78203698-278e-428f-8726-bb381219c6cb],
  null=[HTTP/1.1 200 OK],
  ADRUM_0=[clientRequestGUID:2ff45113-6746-4c94-b6d0-4af26055613c],
  ADRUM_3=[btERT:269],
  ADRUM_2=[btId:4423],
  Server=[Jetty(9.4.z-SNAPSHOT)],
  ADRUM_5=[btDuration:327],
  ADRUM_4=[serverSnapshotType:f],
  Content-Length=[514],
}
```

3. Send a beacon containing the BT response headers to the EUM Server:

```
// Fetch the response headers, which will include the BT headers (ADRUM_0, ADRUM_1, ...).
Map<String, List<String>> headerFields = con.getHeaderFields();

// Add the BT response headers to the request body of the Network Request event.
// that you're reporting.
tracker.withResponseCode(responseCode).withError(response.toString()).reportDone();
                   .withResponseHeaderFields(headerFields)
                   .reportDone();

// Report the Network Request event to the EUM Server.
Instrumentation.sendAllEvents();
```

4. In the Controller UI, you should be able to view the correlated business transaction in the **Device Details** dialog.

Enable Logging for the SDK (Optional)

The IoT Java SDK uses the Simple Logging Facade for Java (SLF4J) as the logging framework. You can use your favorite logging engine that is compatible with SLF4J.

If no binding is found on the classpath, then SLF4J will default to a no-operation implementation and display console messages like the following:

```
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation //
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
```

To use the `java.util.logging` engine, add the following line to the `build.gradle` file:

```
dependencies {
   ....
   runtime group: 'org.slf4j', name: 'slf4j-jdk14', version: '1.7.25'
   ....
 }
```

To see all the debug messages from the library, append the following line to the bottom of the file `/Library/Java/JavaVirtualMachines/<your-jdk-version>/Contents/Home/jre/lib/logging.properties`:

```
com.appdynamics.iot.level = FINEST
```

Customize the IoT Java Instrumentation (Optional)

You can further customize the IoT Java instrumentation using the IoT Java SDK. See the latest IoT Java SDK documentation or the previous versions listed below:

- https://sdkdocs.appdynamics.com/javadocs/iot-java-sdk/4.5/4.5.0/
- https://sdkdocs.appdynamics.com/javadocs/iot-java-sdk/4.5/4.5.1/
- https://sdkdocs.appdynamics.com/javadocs/iot-java-sdk/4.5/4.5.2/
- https://sdkdocs.appdynamics.com/javadocs/iot-java-sdk/4.5/4.5.4/

Run the Sample Java Application

The sample Java application sends sample data for Custom, Network Request, and Error events. The data mocks a smart car application, capturing usage information, network performance, and errors.

To run the sample app, follow the Getting Started instruction given in the iot-java-sdk GitHub repository.

Troubleshooting the IoT Java SDK

This section provides instructions for debugging common issues.

Unable to Link the IoT Java Agent

If you are getting the following error when trying to link the IoT Java Agent, it's because of a dependency on `log4j`.

```
loader constraint violation: when resolving method "org.slf4j.impl.StaticLoggerBinder.getLoggerFactory()Lorg
/slf4j/ILoggerFactory;" the class loader (instance of com/intellij/ide/plugins/cl/PluginClassLoader) of the
current class, org/slf4j/LoggerFactory, and the class loader (instance of com/intellij/util/lang
/UrlClassLoader) for the method's defining class, org/slf4j/impl/StaticLoggerBinder, have different Class
objects for the type org/slf4j/ILoggerFactory used in the signature
java.lang.LinkageError: loader constraint violation: when resolving method "org.slf4j.impl.StaticLoggerBinder.
getLoggerFactory()Lorg/slf4j/ILoggerFactory;" the class loader (instance of com/intellij/ide/plugins/cl
/PluginClassLoader) of the current class, org/slf4j/LoggerFactory, and the class loader (instance of com
/intellij/util/lang/UrlClassLoader) for the method's defining class, org/slf4j/impl/StaticLoggerBinder, have
different Class objects for the type org/slf4j/ILoggerFactory used in the signature
        at org.slf4j.LoggerFactory.getILoggerFactory(LoggerFactory.java:273)
        at org.slf4j.LoggerFactory.getLogger(LoggerFactory.java:241)
        at org.slf4j.LoggerFactory.getLogger(LoggerFactory.java:254)
        at com.appdynamics.iot.Instrumentation.<clinit>(Instrumentation.java:39)
        ...
```

To correct the problem, you need to remove the dependency that you added to enable logging. Thus, *remove* the line specifying the group `org.slf4j` shown below from `dependencies`:

```
dependencies {
   ....
   runtime group: 'org.slf4j', name: 'slf4j-jdk14', version: '1.7.25'
   ....
}
```

Instrument Applications with the IoT REST APIs

The IoT REST APIs enable you to report instrumentation data directly to the EUM Server. You can use any platform or language that has support for HTTPS requests and JSON.

This page describes how to create a JSON request body, form the resource URI, and make an HTTPS request to the IoT REST APIs to report instrumentation data for the three supported events.

Follow these steps to get your EUM App Key and use the IoT REST API:

Review the Requirements

Before you begin, make sure you meet these requirements:

- Get an EUM App Key
- Platform/language that supports HTTPS requests
- JSON support

Form the IoT REST URLs

To form the IoT Monitoring REST resource URL, you will need to know the IoT REST API base URL and port as well as your App Key.

IoT REST API Base URL

The IoT REST API base URL depends on your Controller location. See the IoT REST API row on SaaS Domains and IP Ranges. For example, the IoT base URL for the Americas region would be:

```
https://iot-col.eum-appdynamics.com/eumcollector/iot/v1
```

After creating your IoT Application, use your App Key to test your IoT endpoint and look for a HTTP 200 response. For example, for the Americas region, run the following command:

```
curl -I https://iot-col.eum-appdynamics.com/eumcollector/iot/v1/application/<APPKEY>/enabled
```

IoT Endpoints

With your App Key, you can form the IoT resource endpoints. See the Summary of the IoT endpoints for the list of supported resource endpoints and their descriptions.

Create the JSON Request Body

You report device information and events in a JSON request body. The JSON includes an array of beacon objects, with each beacon object containing device data and events. The array enables you to transmit data from multiple devices in one request to the EUM Server. You can transmit up to 200 beacons per request.

Save the JSON below to a file (e.g., `testBeacon.json`) and replace the values for the timestamp properties with integers representing the UNIX Epoch time in milliseconds. The JSON contains the three supported events `customEvents`, `networkRequestEvents`, and `errorEvents` for a smart thermostat. In the next two steps, you will validate and send the JSON as a beacon to the IoT REST API.

Make sure you save timestamps in milliseconds, and not seconds.

```
[
 {
  "deviceInfo": {
   "deviceType": "Thermostat",
   "deviceId": "4e75d70d-a3f9-474b-bacf-0f4a57fa944c"
  },
  "versionInfo": {
    "hardwareVersion": "Board Rev. 13A",
    "firmwareVersion": "123.5.31",
    "softwareVersion": "9.1.3",
    "operatingSystemVersion": "Linux 13.4"
  },
  "customEvents": [
    {
      "timestamp": <UNIX_Epoch_time_in_milliseconds>,
      "eventType": "Temperature Reading",
      "eventSummary": "Temperature: 25° c",
      "doubleProperties": {
        "celsius": 25.0
      }
    }
  ],
  "networkRequestEvents": [
   {
    "timestamp": <UNIX_Epoch_time_in_milliseconds>,
    "duration": 245,
    "url": "https://api.company.com/v1/temperature",
    "statusCode": 200,
    "requestContentLength": 32,
    "responseContentLength": 0,
    "doubleProperties": {
     "reportedTemperature": 25.0
    }
   }
  ],
  "errorEvents": [
   {
    "timestamp": <UNIX_Epoch_time_in_milliseconds>,
    "name": "SQLException",
    "message": "open() failed because db is locked"
   }
  ]
 }
]
```

Transmit the Beacon Data

To send the beacon, you post the JSON request body to the `/beacons` endpoint. Again, in this cURL example, use the JSON you saved to the file `testBeacon.json` and replace <appKey> with your EUM App Key:

```
curl -v -X POST -d '@testBeacon.json' https://iot-col.eum-appdynamics.com/eumcollector/iot/v1/application
/<appKey>/beacons
```

If the beacons were transmitted successfully, the IoT REST API will return the HTTP Status Code 202:

```
< HTTP/1.1 202 Accepted
```

Verify the Instrumentation in the Controller UI

See Confirm the IoT Application Reported Data to the Controller to verify the instrumentation.

Correlate Business Transactions with Network Requests (Optional)

To correlate business transactions (BTs) with network requests, you need to have instrumented a business application and enabled business transactions in the Controller UI. See Correlate Business Transactions for IoT Monitoring to learn more.

The steps below show you how to get the BT response headers and use them to correlate the BT with an IoT Network Request event.

1. Make a network request that includes the AppDynamics HTTP request headers `ADRUM` and `ADRUM_1` to one of your business applications:

```
curl -H "ADRUM: isAjax:true" -H "ADRUM_1: isMobile:true" -H "Accept: application/json" -H "Content-Type:
application/xml" -H "Content-Length: 0" -X GET http://<url_to_business_app>
```

2. The business application will return response headers that contain information for correlating business transactions. If you were to print these BT response headers, you would see something like the following:

```
ADRUM_0: clientRequestGUID:a27ce4da-d4e6-4bf5-bbca-9b1751aa44a4
ADRUM_1: globalAccountName:customer1_78203698-278e-428f-8726-bb381219c6cb
ADRUM_2: btId:4423
ADRUM_3: btERT:267
ADRUM_4: btDuration:368
Content-Length: 469
Server: Jetty(9.4.z-SNAPSHOT)
```

3. Create a beacon file `btCorrelation.json` with the returned BT response headers (only those headers that include `ADRUM_*`). You assign the returned `ADRUM_*` response headers from the network event request to the business application to the object `responseHeaders` in the beacon as shown below.

```
[
      {
         'deviceInfo':{
            'deviceId':'1111',
            'deviceName':'AudiS3',
            'deviceType':'SmartCar'
         },
         'versionInfo':{
            'hardwareVersion':'1.0',
            'firmwareVersion':'1.0',
            'softwareVersion':'1.0',
            'operatingSystemVersion':'1.0'
         },
         'networkRequestEvents':[
            {
               'url':'<url_to_business_app>',
               'statusCode':200,
               'responseHeaders':{
                  'ADRUM_0':[
                     '<value_returned_from_business_app>'
                  ],
                  'ADRUM_1':[
                     '<value_returned_from_business_app>'
                  ],
                  'ADRUM_2':[
                     '<value_returned_from_business_app>'
                  ],
                  'ADRUM_3':[
                     '<value_returned_from_business_app>'
                  ]
               },
               'timestamp':1525226857000,
               'duration':0,
               'requestContentLength':0,
               'responseContentLength':457
            }
         ]
      }
]
```

4. Send the beacon containing the BT headers to the EUM Server with a cURL command similar to the one here:

```
curl -I -H "Content-Type: application/json" -H "Accept: application/json" -X POST -d @btCorrelation.json
-H https://iot-col.eum-appdynamics.com/eumcollector/iot/v1/application/<appKey>/beacons
```

5. For a successful call, the response headers should be similar to the following:

```
HTTP/1.1 202 Accepted
Cache-Control: private, no-cache, no-store, must-revalidate, max-age=0, proxy-revalidate, s-maxage=0
Expires: 0
Pragma: no-cache
Vary: *
Transfer-Encoding: chunked
Via: 1.1 sjc12-dmz-wsa-5.cisco.com:80 (Cisco-WSA/X)
Connection: keep-alive
```

6. In the Controller UI, you should be able to view the correlated business transaction in the **Device Details** dialog.

Customize the IoT REST API Instrumentation (Optional)

You can further customize the IoT instrumentation using the IoT REST API. See the latest IoT REST API documentation or the previous versions listed below:

- https://sdkdocs.appdynamics.com/javadocs/iot-rest-api/4.5/4.5.0/
- https://sdkdocs.appdynamics.com/javadocs/iot-rest-api/4.5/4.5.1/
- https://sdkdocs.appdynamics.com/javadocs/iot-rest-api/4.5/4.5.2/
- https://sdkdocs.appdynamics.com/javadocs/iot-rest-api/4.5/4.5.4/

Run the Sample Python Application

The sample Python application uses the IoT REST API to send sample data for Custom, Network Request, and Error events. The Network Request events include correlated business transactions. The data mocks a smart car application, capturing usage information, network performance, and errors.

To run the sample app, follow the instructions given in the GitHub repository iot-rest-api-sample-apps.

Troubleshoot the IoT REST API Instrumentation

The sections below provide instructions for troubleshooting your IoT REST API Instrumentation.

Verify Your IoT App Has Been Enabled

Using your App Key, verify that your IoT app has been enabled:

```
curl -v -X GET https://iot-col.eum-appdynamics.com/eumcollector/iot/v1/application/<appKey>/enabled
```

If your App Key has been enabled, you should get the following response:

```
< HTTP/1.1 200 OK
< Cache-Control: private, no-cache, no-store, must-revalidate, max-age=0, proxy-revalidate, s-maxage=0
< Date: Sat, 19 Aug 2017 01:20:39 GMT
< Expires: 0
< Pragma: no-cache
< Vary: *
< Content-Length: 0
< Connection: keep-alive
```

If the App Key does not exist:

```
< HTTP/1.1 403 Forbidden
```

Validate Beacons

You can use the validate beacon endpoint (`/validate-beacons`) to verify that the beacon's JSON request body complies with the REST API schema.

> You are not required or recommended to validate beacons before transmitting them. You should only use this endpoint in development for testing and troubleshooting.

In this cURL example, you are verifying that the JSON given in the file `testBeacon.json` is valid. Replace `<appKey>` with your EUM App Key.

```
curl -v -X POST -d '@testBeacon.json' https://iot-col.eum-appdynamics.com/eumcollector/iot/v1/application
/<appKey>/validate-beacons
```

If the JSON request body containing the beacon data is valid, the IoT Monitoring REST API will return the HTTP Status 200:

```
HTTP/1.1 200 OK
```

If the JSON request body is invalid, the IoT REST API will return the HTTP Status 422 and a response body with the description of the error message.

```
< HTTP/1.1 422 Unprocessable Entity
```

Verify Timestamps

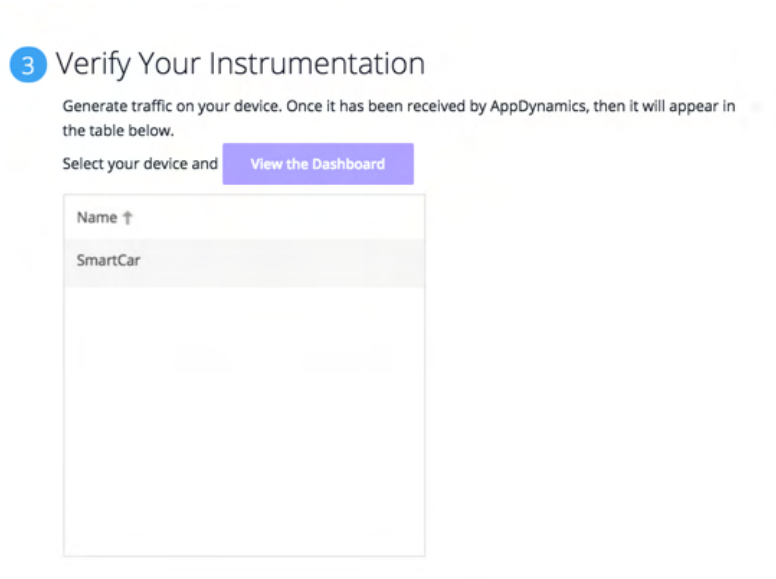
When you create the JSON body and replace the values for the timestamp properties, make sure the timestamps are in milliseconds, not seconds.

Confirm the IoT Application Reported Data to the Controller

Related pages:

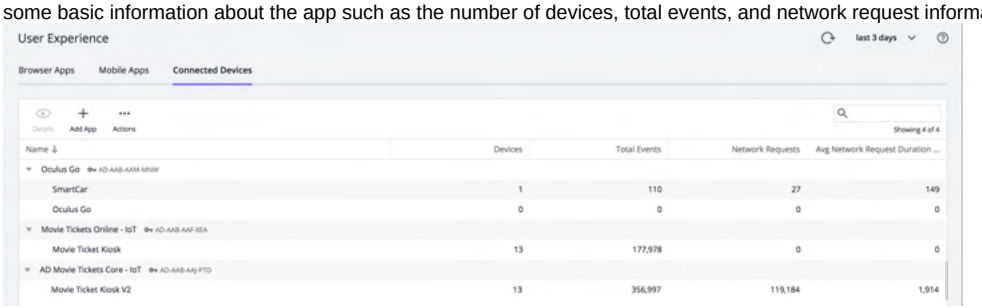
- Instrument Applications with the IoT C/C++ SDK
- Instrument Applications with the IoT Java SDK
- Instrument Applications with the IoT REST APIs

From the **Getting Started Wizard**, you will see your device listed in the table after the IoT application has reported data to the Controller.



If you left the **Getting Started Wizard**, you can always verify the IoT application has been enabled and reported data by doing the following:

1. In the Controller UI, open **User Experience > Connected Devices**.
2. Check the list of registered connected device applications to verify that the application is registered with the Controller. You can also use view some basic information about the app such as the number of devices, total events, and network request information.



3. Start monitoring your application. See Configure IoT Application Monitoring.

Monitor Applications with the IoT Dashboards

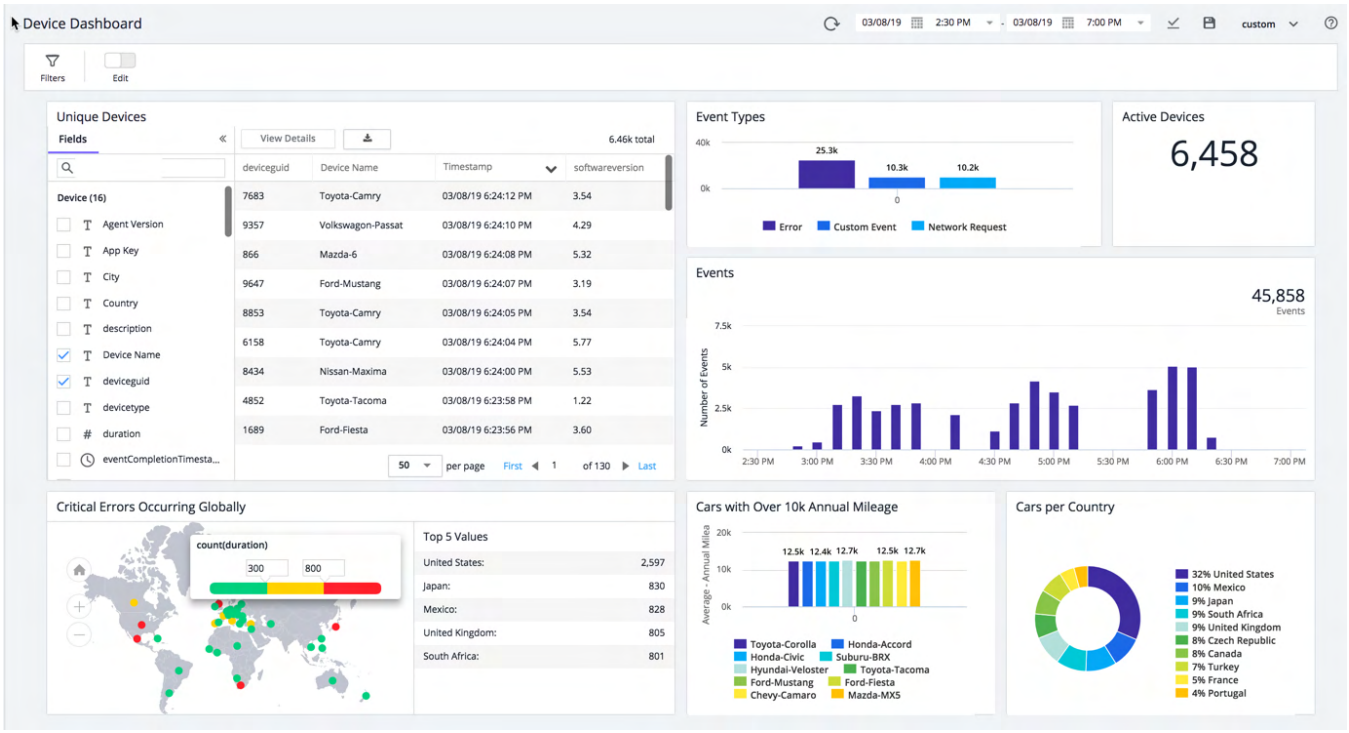
IoT Monitoring provides you with three dashboards to view and analyze the performance and usage of your application. Each dashboard provides you with a different aspect of your application's performance. Within each dashboard, you can view data visualized as widgets or detailed records or create custom widgets based on your chosen parameters.

The page introduces the features, functionality, and goal of each of these dashboards:

Device Dashboard

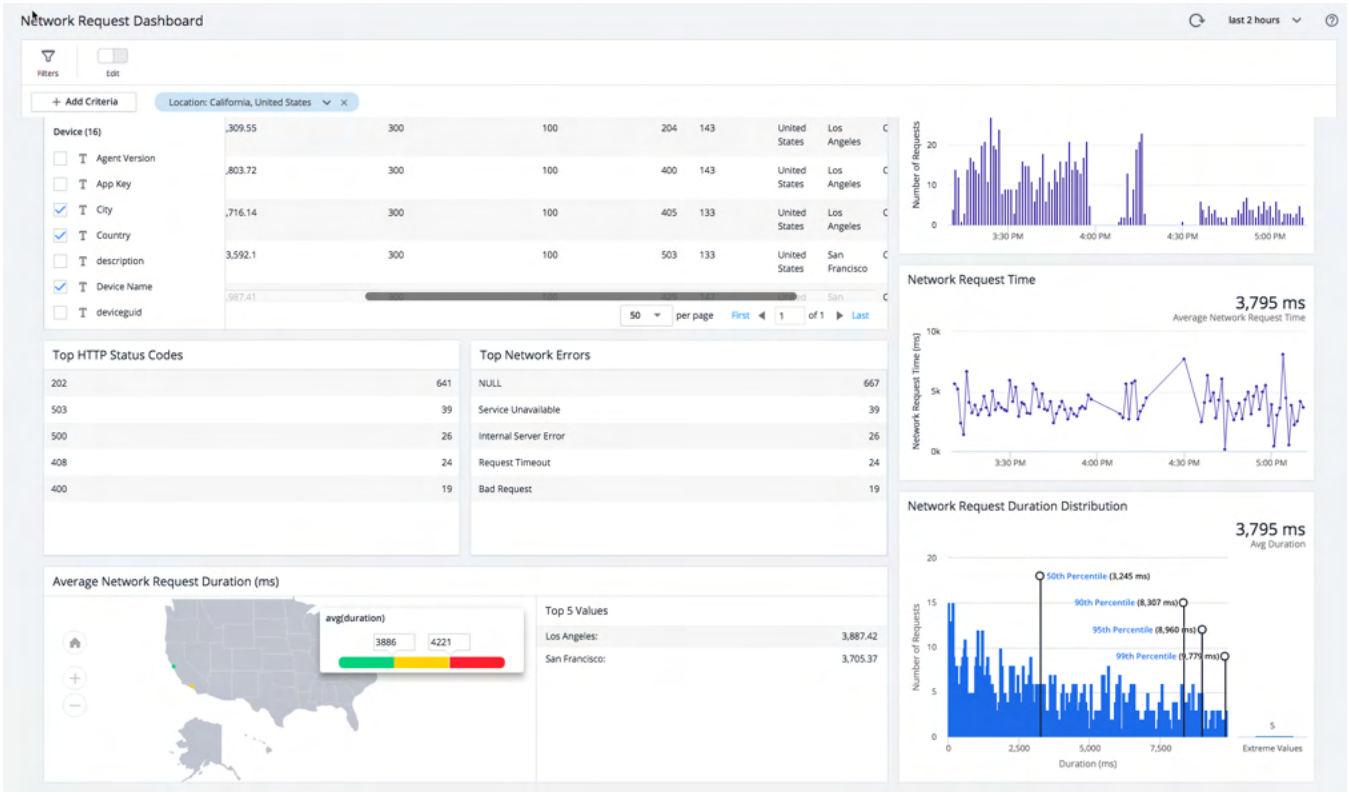
The **Device Dashboard** enables you to monitor your device status, custom information, and custom device application information. You can view summary activity or use filters to drill down to view metrics for a specific device.

Using the **Device Dashboard**, DevOps can ensure that devices are up and running, product managers and business stakeholders can examine device activity and trends, and developers can drill down into results to identify issues affecting single or multiple devices.



Network Request Dashboard

The **Network Request Dashboard** enables you to view outgoing network requests, network request details, and analyze network requests.

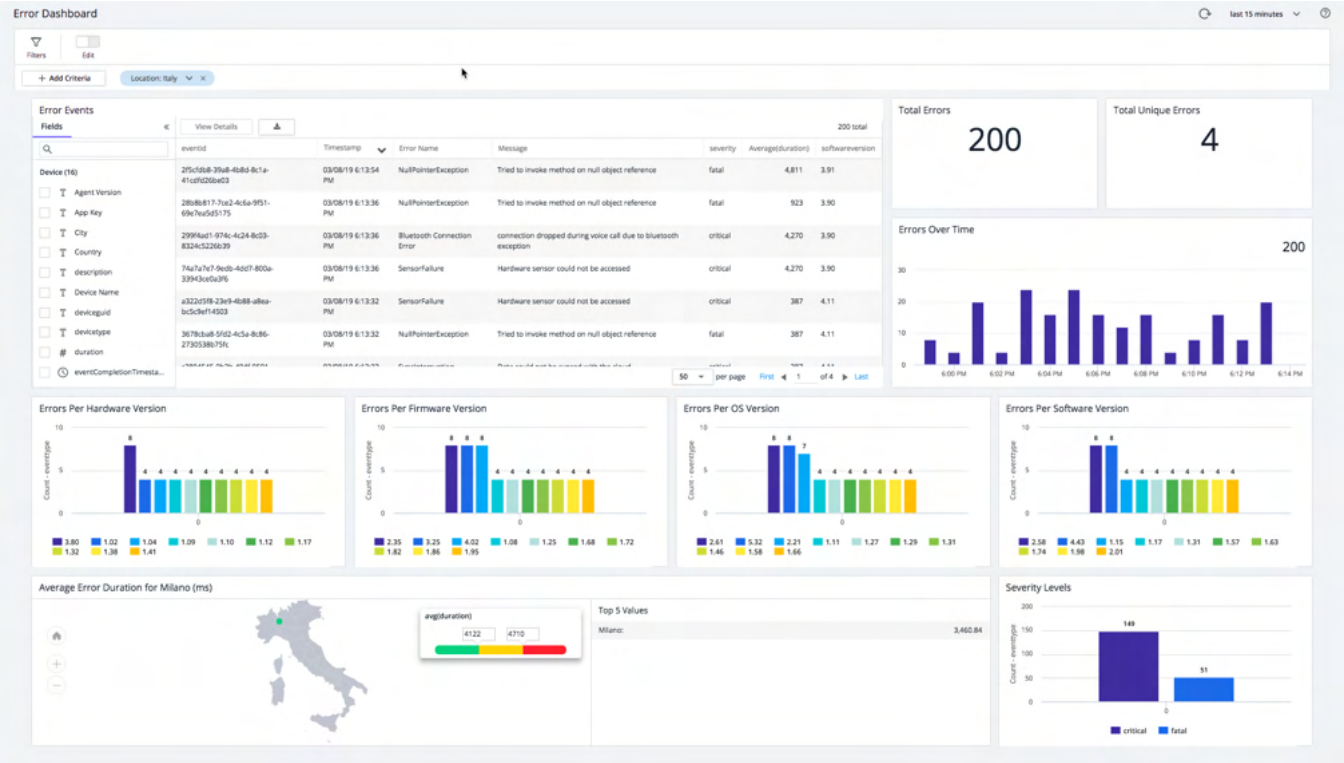


With the **Network Requests Dashboard**, you can:

- Check the availability of a device or service
- Discover slow network requests
- View failed network requests
- Analyze and sort network requests by criteria such as status code, app version, and so on
- Monitor network traffic over time

Error Dashboard

The **Error Dashboard** enables you to view crashes, exceptions, and custom errors, both fatal and non-fatal. DevOps can monitor activity list and notify developers of error events. The developers can select the error event to view details and download the stack trace for debugging.



The stack trace will only be available if it is reported with the IoT SDKs or through the IoT REST API.

With the **Error Dashboard**, you can:

- Discover new errors
- Assess the scope and impact of errors, such as how many devices are affected, what type of devices are affected, and how seriously affected are the devices.
- Monitor error patterns
- Download stack traces of errors to debug code
- Discover broken devices (failed hardware)

How to Use IoT Dashboards

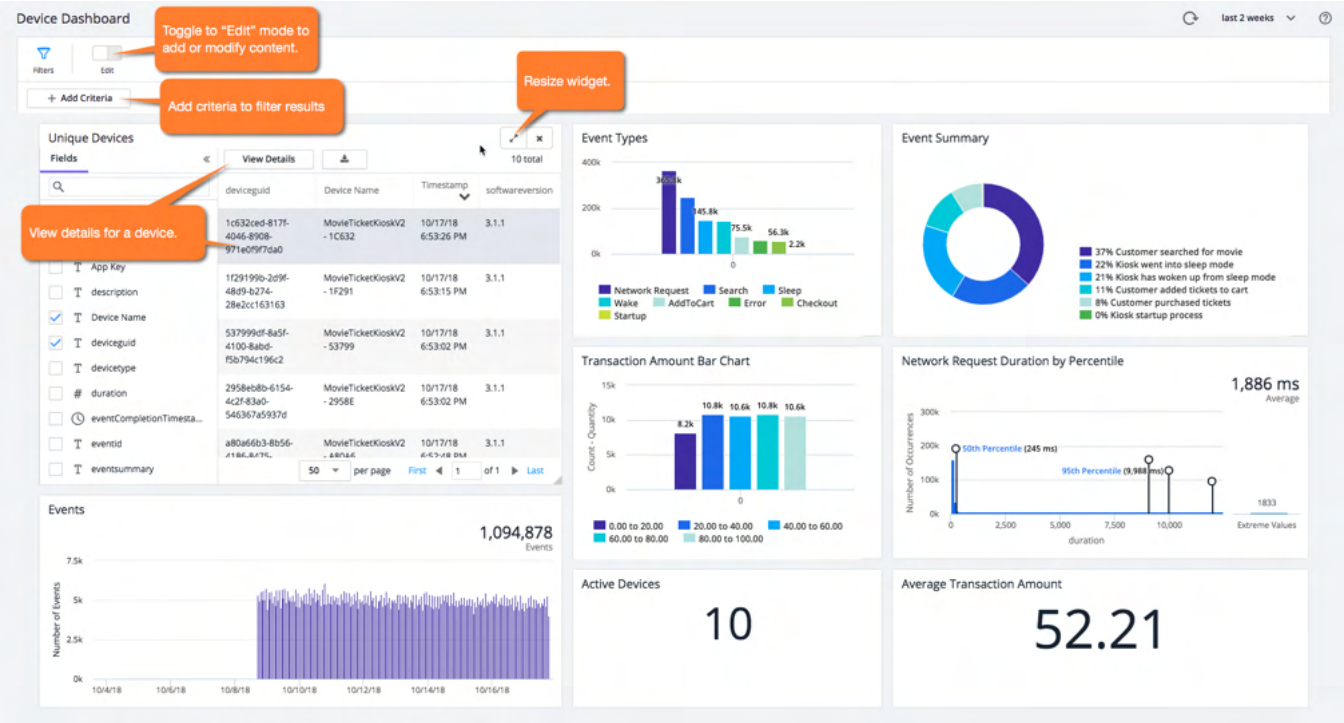
Although each dashboard presents different information, they all have the same UI functionality and features. This page provides an overview of the dashboard functionality to help you get started monitoring your devices.

Access the IoT Dashboards

1. Open the application you want.
2. Select one of the dashboards to view:
    - **Devices**
    - **Network Requests**
    - **Errors**

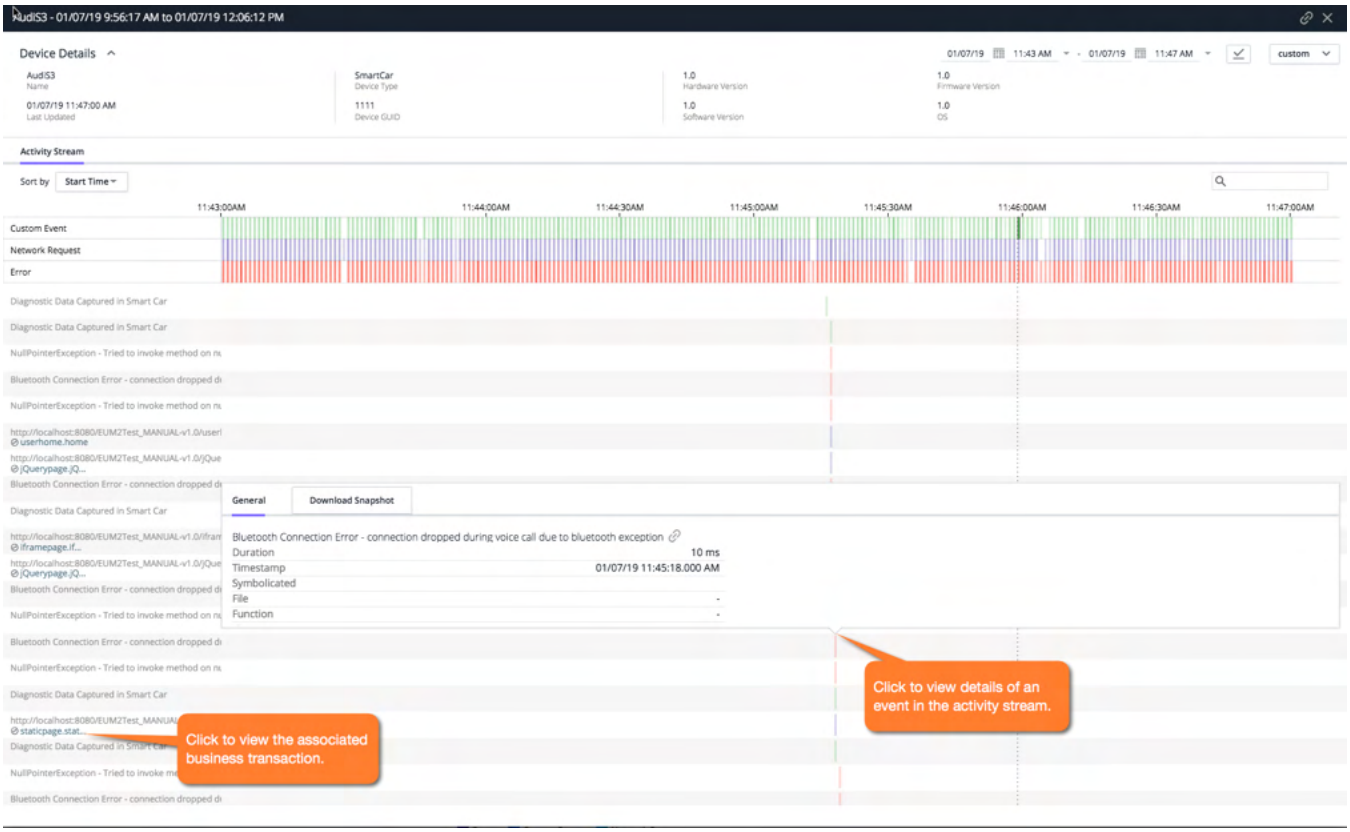
View Dashboard Summaries

From each dashboard, you can view aggregated data in predefined and custom widgets. Each dashboard is in view mode by default, so data is read-only, although you can set filters, view details, and resize widgets.



View Details

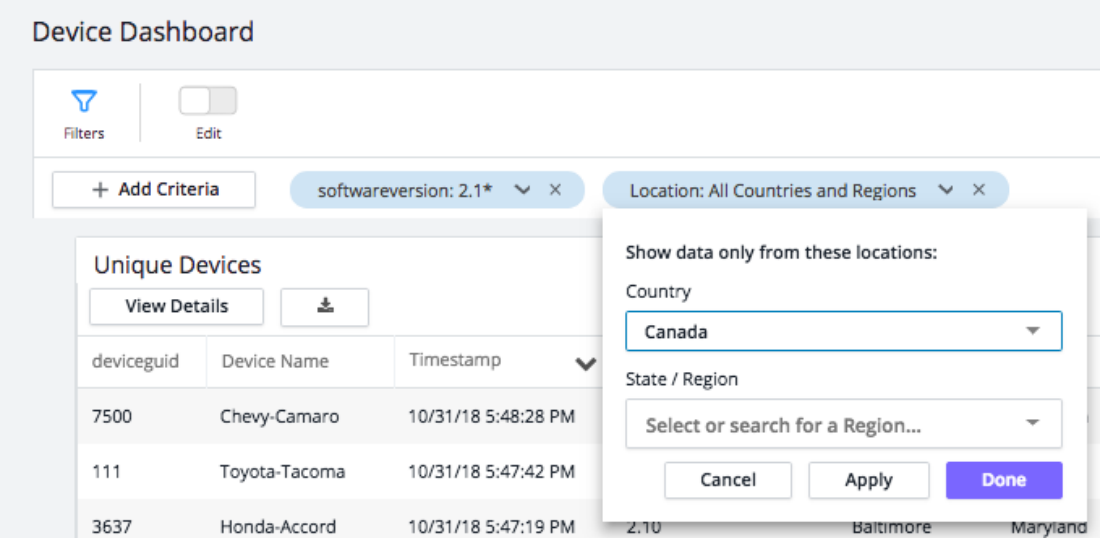
To view details, you select a row in one of the tables and click the **Details** icon. The **Device Details** dialog displays. You can view when the three types of events (Custom Event, Network Request, and Error) occurred on the timeline, or by duration. You can also click on the event bar in the waterfall for more details.



Filter Results

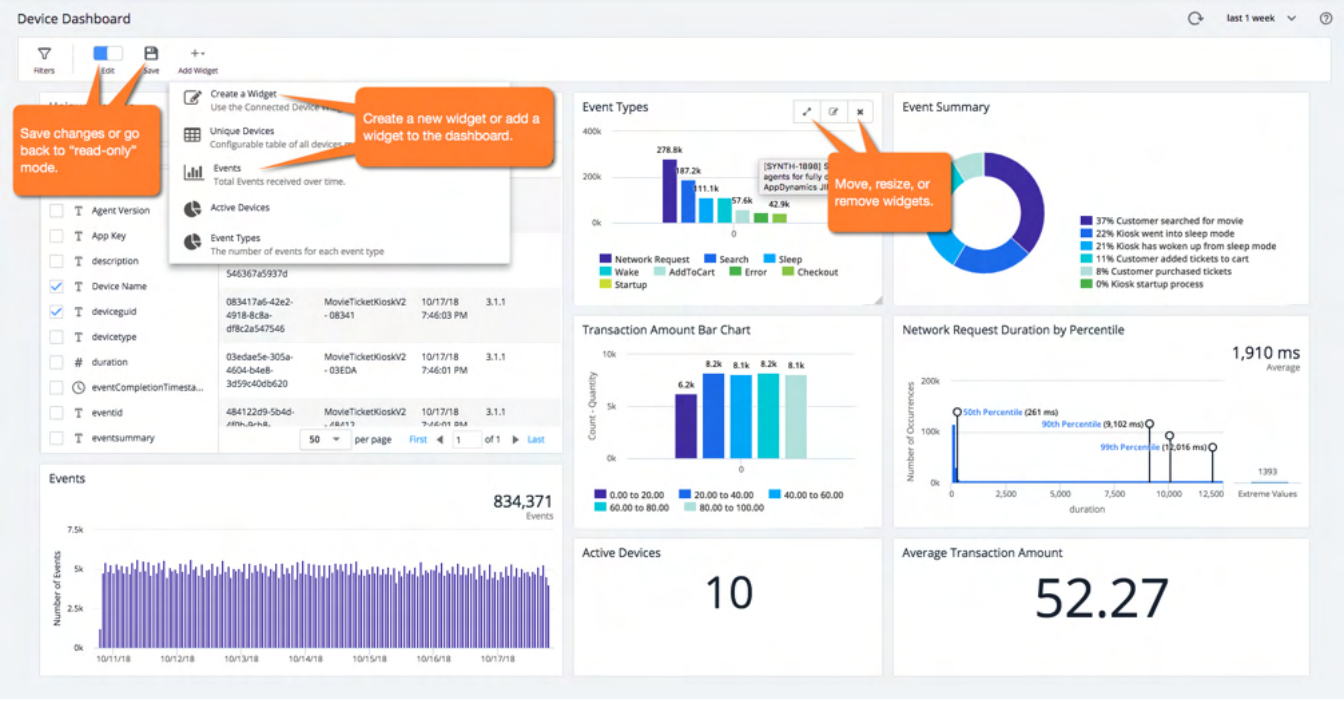
You can filter results by adding criteria. This enables you to focus on factors affecting availability, usage, and performance.

In this screenshot, the criteria **softwareversion** and **Country** are added, which could potentially home in on performance issues caused by a particular software version in a particular country.



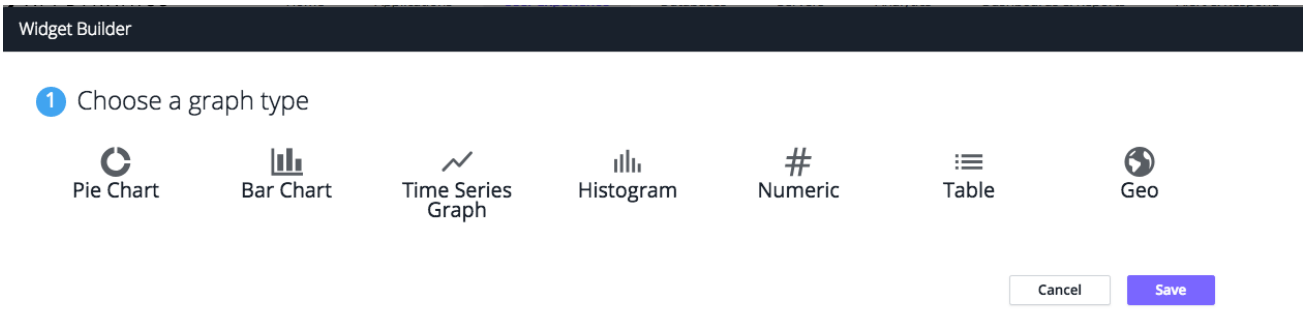
Modify Content and Presentation

To modify the dashboard, you need to be in **Edit** mode. You simply change the toggle to **Edit**, and then you'll be able to add, move, resize, remove, and even create widgets.



Build Custom Dashboard Widgets

Each dashboard has predefined widgets for basic monitoring, but because there are so many different devices and platforms, you will need to create custom widgets with the **Widget Builder**, which guides you through the creation of a widget. This page provides guidance for selecting graph types and using filters when you create widgets.



How to Access the Widget Builder

1. Select a dashboard.
2. Change to **Edit** mode.
3. From the toolbar, click **Add Widget**.
4. From the dropdown, click **Create a Widget**.

Choose the Best Graph Type

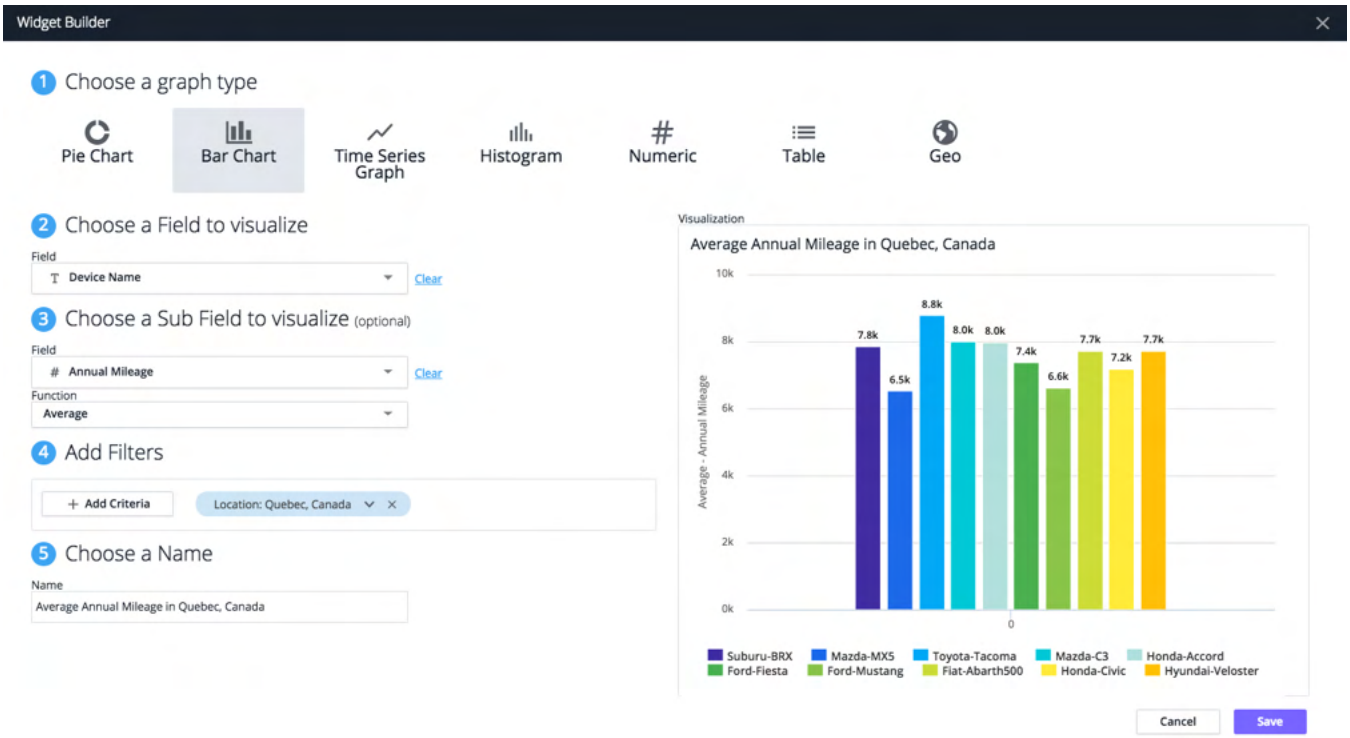
The **Widget Builder** allows you to choose from one of six types of graphs, you should try to use the best graph for your data set. The following is a guideline for choosing the graph type:

Graph Type	Graph Properties	Use Case(s)
Pie Chart	• Field • Subfield • Filters • Name	Use pie charts to compare parts of a whole. Examples: • For a retail device, the percentage of sales for each store. • For a fitness device, the percentage of different types of activities such as running, bicycling, walking.
Bar Chart	• Field • Subfield • Filters • Name	Use bar charts to compare different items within the same category. Examples: • For an inventory device, the number of items per store. • For a smart car, the mileage of different cars.
Time Series	• Field • Filters • Name	Use a time series to measure events over time. Examples: • network errors • sales
Histogram	• Field • Filters • Name	Use histograms to analyze and understand the distribution of large sets of numerical data. Examples: • The distribution of network request duration. • For a smart home device, the distribution of temperatures.
Number	• Field • Filters • Name	Use a number graph to highlight an important data point. Examples: • active devices • sales total • network errors

Table	• Field • Filters • Name	Use a table to view detailed data for a list of similar items. Examples: • customers • network requests • devices
Geo	• Field • Filters • Name	Use the Geo widget to analyze and understand performance across geographic areas. Examples: • Sales in different countries, regions, and cities • Network request durations across countries, regions, and cities • Device errors in different countries, regions, and cities

How to Use Filters

Filters enable you to narrow your results to meaningful information. For example, suppose you wanted to see the total number of items sold at each store, but you were particularly interested in those more expensive items. You could choose a bar graph and use the field **Device Name** and the subfield **Annual Mileage** with the filter **Location: Quebec, Canada** to view the average annual mileage of cars in Quebec, Canada, as shown in the example widget:



Configure IoT Application Monitoring

In addition to enabling and disabling **IoT Monitoring**, you can also configure the display names of network requests and exclude network requests matching given criteria from being monitored.

You can:

- Use the AppDynamics default naming rule, which you can leave as is or modify.
- Disable/modify the default naming configuration.
- Create custom include rules to override the default convention.
- Create custom exclude rules to exclude from monitoring network requests that meet certain criteria.

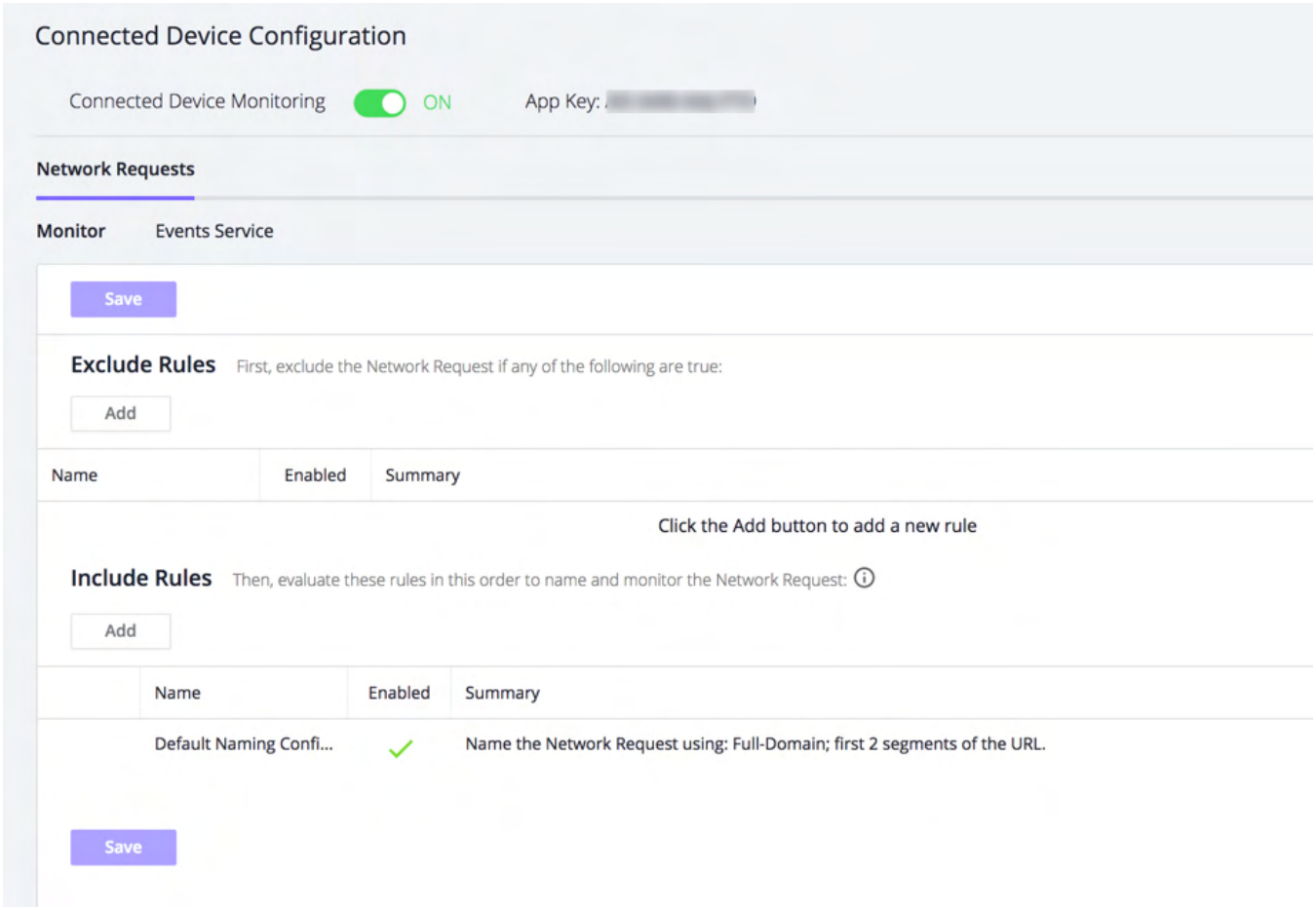
Access the IoT App Configuration

To access connected device configuration:

1. Open the IoT application you want.
2. From the left-hand navigation menu, click **Configuration**.

Enable/Disable IoT Monitoring

From the **Configuration** page, toggle the **Connected Device Monitoring** switch to **ON** to enable monitoring or **OFF** to disable monitoring.



Name IoT Network Requests

The following sections show you how to modify the default naming configuration for network requests and create include naming rules for network requests.

Access Network Requests Rules

From the **Configuration** page, click the Monitor tab if it's not selected already.

Default Network Request Naming Configuration

By default, AppDynamics names network requests using:

- Hostname
- First two segments of the URL

For example, if an application makes this HTTP request: `http://myapp.com/friends/profiles/12345`

The default name that is displayed in the Controller UI for that request is: `myapp.com/friends/profiles`

If this is adequate for your needs, you can leave the default as is. The naming rules you configure here apply to all the IoT applications that are in the same IoT App Group.

Modify the Default Naming Configuration Rule

You may want to configure a different default rule for naming your network requests to help you visualize the parts of your application more clearly. The task is similar to configuring naming rules for business transactions on the server side. Try to group logically related requests together while keeping unrelated requests in separate groups.

- If the default hostname and first two segments of the URL for all your requests are identical, you might want to name the requests based on the last segments or a selection of non-contiguous segments of the URL to distinguish among requests in the network requests list.
- You can also name the requests based on query parameters. For example, if the request passes an order number, you could specify that the value of the `order-number` query parameter is used in the network request name.
- You can also base the name on a regular expression run on the URL. AppDynamics uses the Java libraries for regular expressions. See:
  - Tutorial: http://download.oracle.com/javase/tutorial/essential/regex/index.html
  - Javadoc: http://download.oracle.com/javase/1.5.0/docs/api/java/util/regex/Pattern.html

Modify the Default Network Request Naming Rule

The default configuration covers how all your requests are named if you do not customize them further.

1. From the Network Request tab, scroll down to the **Include Rules** section.
2. Double-click **Default Naming Configuration**.
3. In the **Include Rule** dialog, select the elements you want to use for your default network request naming.
4. Click **OK**.
5. Click **Save**.

Create IoT Network Request Include Rules

By default, the same request naming rule is applied to every URL that your application requests. If you want to apply different naming rules to different URLs, create include rules.

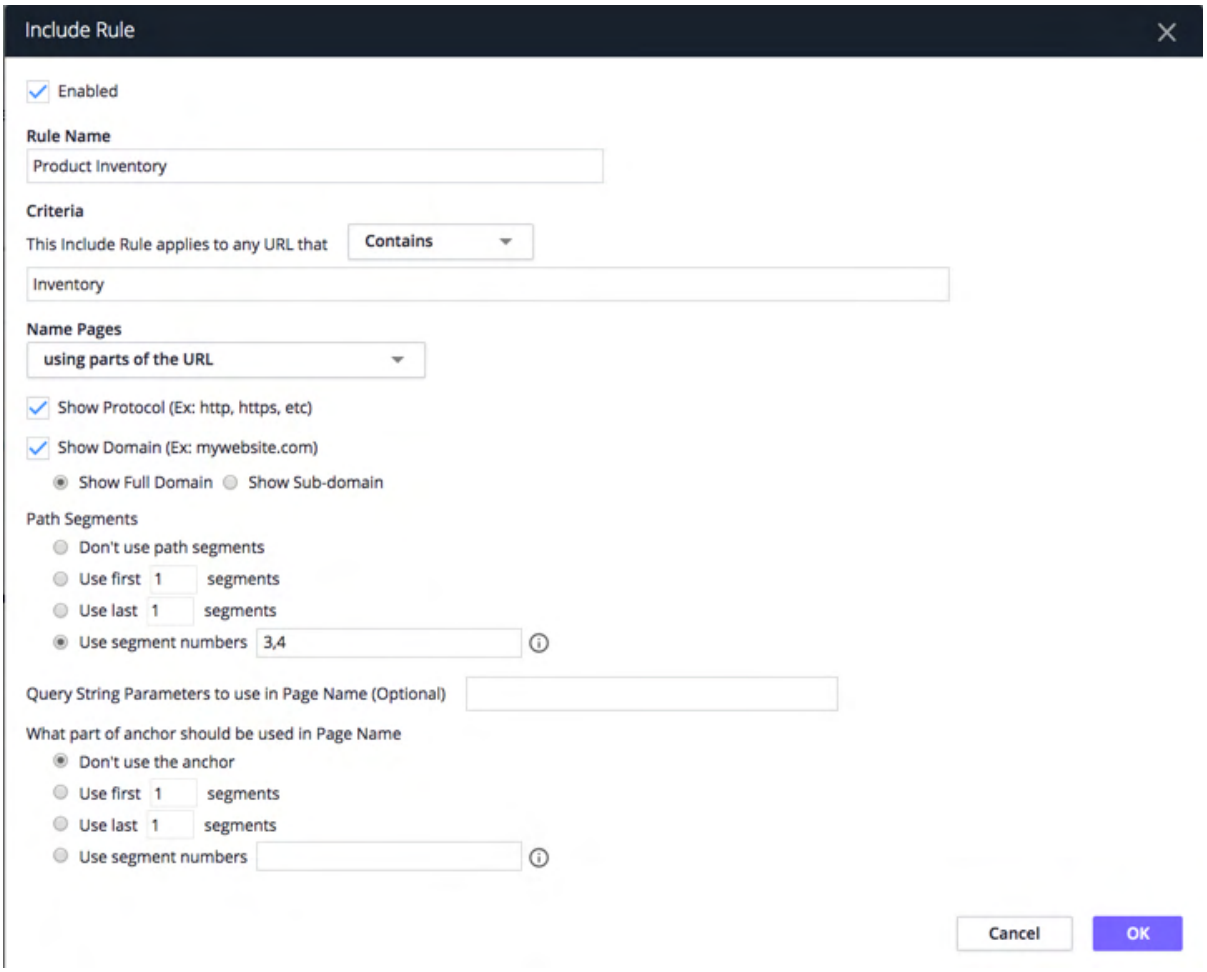
For example, if some requests call your own in-house server and others call out to a third-party API, you may want to see all the third-party API calls as one network request and use the default naming rules for the calls to your own server. You would create a custom naming rule that matches the third party calls and uses only the host in the default rule name or perhaps also include certain query parameters.

Creating an Include Rule

1. From the **Network Request** tab, scroll down to the **Include Rules** section.
2. Click **Add**.
3. In the **Include Rule** dialog, enter a name for the custom rule that you are creating.
4. Check the **Enabled** checkbox to enable the rule.
5. Select the checkboxes and radio buttons and enter the match criteria for AppDynamics to use to name network requests.
6. Click **OK.**

Sample Include Rule

The following rule creates a custom match rule for requests in which the URL contains "inventory". This rule uses the protocol, the subdomain and the third and fourth segments of the URL in the network request name.



You can temporarily cancel the application of a custom naming rule by clearing the **Enabled** checkbox in the custom rule configuration. In this case, the default naming rule is applied to requests that would have been named by the disabled custom rule. To remove the rule permanently, select the custom rule in the **Custom Naming Rules** list and click the **Delete** icon.

Exclude IoT Network Requests

If there are certain types of requests that you do not want to monitor, create custom exclude rules for them based on the URL and/or the application name. Excluded network requests are not reported or counted toward the network request limit of 500 requests per controller application.

Create an Exclude Rule

1. From the Network Request tab, scroll down to the **Exclude Rules** section.
2. Click **Add**.
3. In the **Exclude Rule** dialog, enter a name for the exclude rule that you are creating.
4. Check the **Enabled** checkbox to enable the rule.
5. Select the check boxes and radio buttons and enter the match criteria for AppDynamics to use to name network requests.
6. Click **OK.**

You can temporarily cancel the application of an exclude rule by clearing the **Enabled** checkbox in the exclude rule configuration. To remove the rule permanently, select the exclude rule in the **Exclude Rules** list and click the **Delete** icon.

Change Priority of Rules

Rules are evaluated in the order that they appear in the include or exclude list. You can change the priority of the rules by dragging and dropping rules towards the top (higher priority) or towards the bottom of the list (lower priority). Custom rules are always evaluated before the default naming rule, beginning with the custom rule that has the highest priority.

Your IoT app may make various kinds of network requests, and not all of them may be equally important to monitor in detail. For example, any requests to Google Analytics that your app may make are useful but probably aren't as important to analyze as the requests it makes to your backend.

To manage the impact on your overall Events Service usage, you can create rules which specify which of these network requests should be sent on to the Event Service, either by excluding a request entirely, including a particular request or a sample of that request types by percentage, or by simply allowing the request to be sent on.

In general, the behavior follows this pattern:

- If no rules are specified, data on all network requests are sent on.
- If exclude rules are specified, and a network request satisfies a rule, that data is *not* sent on.
- If include rules are specified, any network request that satisfies a rule is sent on, based on sampling defined by the percentage indicated in the rule.
- If both include and exclude rules are specified, a network request that satisfies an include rule but does *not* satisfy an exclude rule is sent on.

IoT Custom Geo Mapping

By default, IoT Monitoring uses public geographic databases to resolve external IP addresses to geolocations. You can instead use IoT custom geo mapping to map internal IP addresses to geolocations for your IoT applications.

With custom geo mapping, you can monitor the performance of devices running in Virtual Private Clouds (VPCs) with internal addresses distributed across different geographies.

Methods to Map IP Addresses

You can map one IP address, a range of IP addresses, or multiple IP addresses with a subnet mask to one geolocation, giving you a wide range of control over how to map internal IP addresses to geographic locations.

Scope of Custom Geo Mapping

You can only use custom geo mapping in SaaS Controllers. You can provide one custom geo mapping for each app key. Each app key can include multiple applications, enabling them to use the custom geo mapping.

Geo Mapping File

The geo-mapping file is an XML file used to map IP addresses to geolocations. You upload this file to the Controller for your IoT application.

File Specification

The geo-mapping file size is limited to 1 GB, but there is no limit to the number of mapped IP addresses.

Default Mapping

If you upload a custom geo-mapping file, IoT Monitoring will only resolve the IP addresses based on your mapped IP addresses and locations. Thus, if you are going to use a custom geo-mapping file, you should configure the default to cover the entire IP range.

IP Addresses and Ranges

You can only specify IPv4 addresses in the custom geo-mapping file.

When using IP address ranges, you must list the addresses in ascending order. For example, if your geo-mapping file specifies the two ranges 1.1.1.1–2.2.2.2 and 2.2.2.2–3.3.3.3, you must list the first range before the second so that they are in ascending order.

Example Geo Mapping Files

The example geo-mapping files below show you the scope of mapping geolocations and provide a general use case for each.

Geo Mapping File XML Schema

You can download the XML schema file `custom-geo-mapping.xsd` below to validate your geo-mapping XML file.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="mappings" type="config"/>
  <xs:element name="ip-range" type="explicitIpAddressRange"/>
  <xs:element name="subnet" type="subnetIpAddressRange"/>
  <xs:complexType name="config">
    <xs:sequence>
      <xs:element name="mapping" type="mapping" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="default" type="geoLocation" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="mapping">
    <xs:sequence>
      <xs:choice>
        <xs:element ref="ip-range"/>
        <xs:element ref="subnet"/>
      </xs:choice>
      <xs:element name="location" type="geoLocation" minOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="geoLocation">
    <xs:attribute name="country" type="xs:string" use="required"/>
    <xs:attribute name="region" type="xs:string" use="required"/>
    <xs:attribute name="city" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:complexType name="explicitIpAddressRange">
      <xs:attribute name="from" type="xs:string" use="required"/>
      <xs:attribute name="to" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:complexType name="subnetIpAddressRange">
      <xs:attribute name="from" type="xs:string" use="required"/>
      <xs:attribute name="mask" type="xs:string" use="required"/>
  </xs:complexType>
</xs:schema>
```

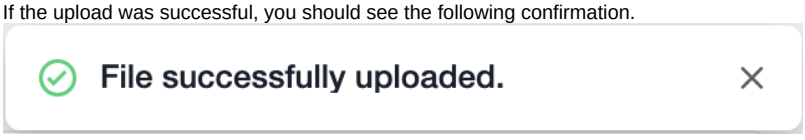
Upload Geo Mapping Files

From your SaaS Controller, you can perform the following actions with your custom geo-mapping file:

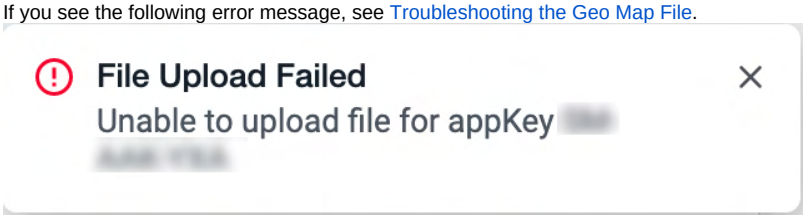
- Upload New Geo Mapping File
- Overwrite Existing Geo Map File
- Reset Geo Map File

Upload New Geo Mapping File

1. Navigate to the **Configuration > Custom GeoMap**.
2. From the **Upload Custom GeoMap file** dialog, click **Choose File**.
3. Select a geo-mapping XML file.
4. Click **Upload**.
5. If the upload was successful, you should see the following confirmation.



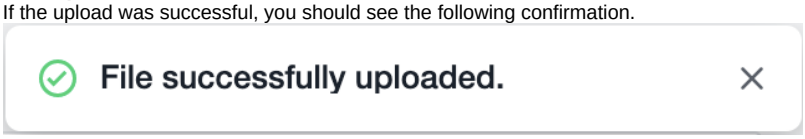
6. If you see the following error message, see Troubleshooting the Geo Map File.



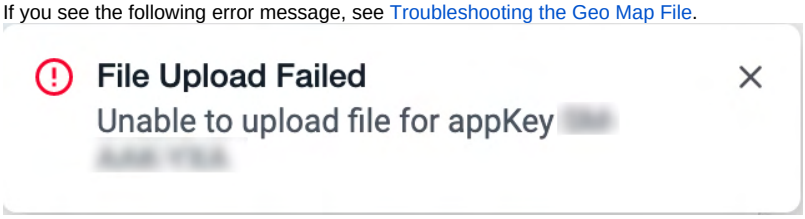
After you have uploaded the geo-mapping file, it may take some time for IoT Monitoring to apply the new custom mapping to your applications.

Overwrite Existing Geo Map File

1. Navigate to the **Configuration > Custom GeoMap**.
2. From the **Upload Custom GeoMap file** dialog, click **Overwrite Current Mapping**.
3. Select a geo-mapping XML file.
4. Click **Upload**.
5. If the upload was successful, you should see the following confirmation.



6. If you see the following error message, see Troubleshooting the Geo Map File.



Reset Geo Map File

1. Navigate to the **Configuration > Custom GeoMap**.
2. From the **Upload Custom GeoMap file** dialog, click **Reset to Default**.
3. From the **Reset Confirmation** dialog, click **Reset**.

View Custom Geo Locations

You can add the **Geo** widget to one or more of the available dashboards to view the geolocation information of your devices. The Geo widget will display geo locations resolved with the public geographic databases or a custom IP address mapping. To view the geolocations from the **Device Dashboard**:

1. From your IoT application, navigate to **Devices**.
2. Scroll down to the **Geo** widget.
3. (Optional) Build a custom **Geo** widget. You can filter the devices by geodata such as city, region, or country.

In addition, you can filter data in the other widgets by geodata based on your geo-mapping file.

Troubleshooting the Geo Map File

Upload Issues

If you get an error message trying to upload the custom geo-mapping file, check the following:

- XML validity: Use an XML validation tool to determine if your XML of your geo-mapping file is valid.
- XML schema: Confirm that your custom geo-mapping file conforms to the XML schema.
- Invalid IP addresses: Confirm your IP addresses are valid.
- File size: Confirm that the custom geo-mapping file is less than 1 GB.

Incorrect Geo Mapping

If IP addresses are not being resolved to the correct geolocations, confirm that the IP addresses and geo-locations in your geo-mapping file are correct.

Database Visibility

Database Visibility in AppDynamics provides end-to-end visibility on the performance of your database, helps you troubleshoot problems such as slow response times and excessive load, and provides metrics on database activities, such as:

- SQL statements or stored procedures that are consuming most of the system resources
- Statistics on procedures, SQL statements, and SQL query plans
- Time spent on fetching, sorting, or waiting on a lock
- Activity from the previous day, week, or month

Installation and Administration

- Install the Database Agent
- Start and Stop the Database Agent
- Upgrade the Database Agent

Configuration

- Configure the Database Agent
- Add Database Collectors

Using Database Visibility

- Overview of Database Visibility
- Monitor Database Performance
- Monitor Database Server Hardware

Reference

- Database Agent Configuration Properties
- Database Agent Events Reference
- Database Monitoring Metrics

Overview of Database Visibility

Related pages:

- Add Database Collectors
- Database Visibility

Database Visibility provides metrics on the performance of your database and helps troubleshoot performance-related issues.

Components of Database Visibility

AppDynamics Database Visibility consists of four main components:

- Database Agent
- Collector
- Controller
- Events Service (on-premises only)

Database Agent

The AppDynamics Database Agent is a standalone Java program that collects performance metrics about your database instances and database servers. You can deploy the Database Agent on any machine running Java 1.8 or higher. The machine must have network access to the AppDynamics Controller and the database instance that you want to be monitored.

A database agent running on a typical machine with 16 GB of memory can monitor about 25 databases. On larger machines, a database agent can monitor up to 200 databases. To monitor more than 100 databases, increase the initially allocated heap size for the Database Agent JVM. See Database Visibility Supported Environments

Collector

The Database Agent Collector is the process that runs within the Database Agent to collect performance metrics about your database instances and database servers. One collector collects metrics for one database instance. Multiple collectors can run in one Database Agent.

AppDynamics Database Visibility detects when a database back end has matching credentials with a database server being monitored by a collector. Database Visibility automatically associates the back end with the collector, so you can view its performance with the Application Flow Map, Tier Flow Map, or Node Flow Map.

Controller

The Controller is the central interface where you can see all your database instances and database server performance metrics.

The following types of information are sent to the Controller:

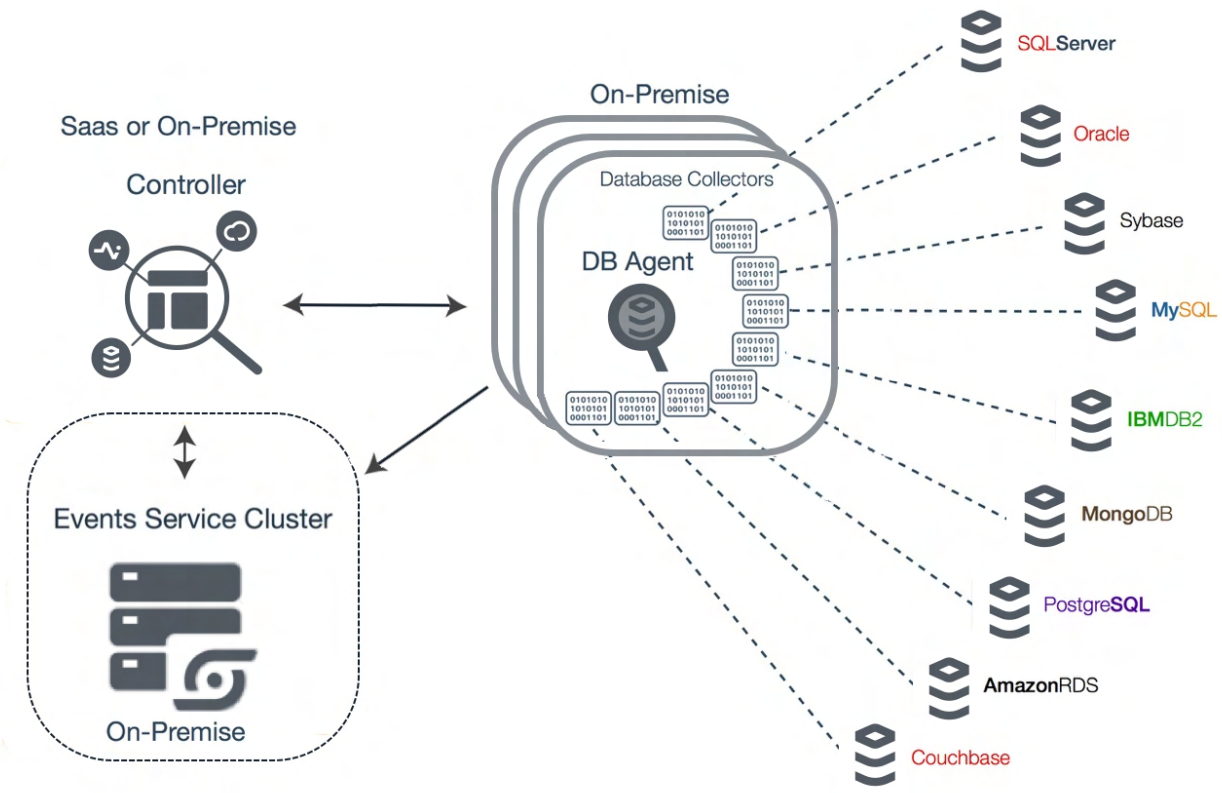
- Database-level metrics, such as the number of queries processed and other database statistics
- Names and attributes of all sessions, clients, queries, and other objects on the monitored system

Events Service

Event service stores high volumes of metric data. If you are using the on-premises version of Database Visibility, you must install the Events Service.

The following types of information are sent to the Events Service:

- Time that each query spends at each wait state
- Individual query statistics for databases that support it
- Information about individual execution plans in databases that support it


Saas or On-Premise
Controller
Events Service Cluster
On-Premise
On-Premise
Database Collectors
DB Agent
SQLServer
Oracle
Sybase
MySQL
IBMDB2
MongoDB
PostgreSQL
AmazonRDS
Couchbase

Database Visibility Supported Environments

Related pages:

- Add Database Collectors
- Database Visibility

This page describes the application environments and versions supported by the Database Visibility Agent.

Database Visibility Support

Once Database Visibility is available, you can create collectors that run on the Database Agent to monitor any of these systems:

Database	Supported through Amazon RDS	Version
Apache Cassandra		>= 3.11.4
Datastax Enterprise (DSE) Cassandra		>= 6.7.3
Couchbase		>= 4.5
IBM DB2 LUW		9.x, 10.x, 11.x
MongoDB, MongoDB cluster		>= 2.6
MySQL	Yes	All versions including MySQL Version 8.0, Percona, MariaDB, and Aurora
Microsoft SQL Server	Yes	2005, 2008, 2012, 2014, 2016, 2017, 2019, and SQL Azure
Microsoft SQL Server on Linux		SQL Server on Linux is currently available as a public preview and is not recommended for production use. Database Visibility works well with this preview release, but monitoring results may vary until a stable version of SQL Server on Linux is available.
Oracle, Oracle RAC	Yes	10g (>= 10.2), 11g, 12c, 18c, and 19c
PostgreSQL	Yes	All the versions including Azure Database for PostgreSQL
Sybase ASE		>= 15
Sybase IQ		All versions

Operating System	Version
Windows	64-bit From 20.5 release, 32-bit is unsupported.
Linux	32-bit and 64-bit
Solaris	All versions
AIX	>= 6.1
Amazon RDS	All versions

To avoid metric value errors, use a 64-bit JRE with the 64-bit operating system.

Recommended

If you use a third-party application along with Database Visibility to monitor Sybase, the data displayed in the controller may not display correctly. For accurate metrics, do not use both a third-party application and Database Visibility to monitor Sybase.

Database Visibility System Requirements

This page describes the hardware and software requirements using Database Visibility.

Hardware Requirements

Hardware requirements vary depending on database activity. If your database activity increases, you may need to adjust your hardware configuration.

The machine running the Database Agent should meet the following hardware requirements:

- 1 GB of heap space and an additional 512 MB of heap space for each monitored database instance. For less busy databases, you may reduce the heap space to 256 MB per monitored database instance.
- 2 GHz or higher CPU.

> **Database Instance**
>
> A Database instance can be a node in the Oracle RAC, MongoDB, Couchbase cluster, standalone-collector, or a sub-collector.

This table shows sample calculations for heap space allocation:

Number of Database Instances Monitored	Heap Space Allocation
5	(5 x 512 MB) + 1024 MB = 3,584 MB
20	(20 x 512 MB) + 1024 MB = 11,264 MB
100	(100 x 512 MB) + 1024 MB = 52,224 MB

AppDynamics Controller Sizing Requirements

The Controller database should meet the following hardware requirements:

- 500 MB of disk space per collector per day
- 500 MB of disk space for the Events Service per day. By default, the Events Service retains data for 10 days.

See Controller System Requirements.

> **Note**
>
> The Database Agent requires the Events Service. Start Event Service before you start the Database Agent.

Software Requirements

- The Database Agent runs on a Java Virtual Machine. You must have Java >= 1.8.
- The operating systems Linux and Windows are supported.

Network Requirements

- The machine on which the database is running or the machine you want to monitor must be accessible from the machine where the Database Agent is installed and running. This machine must have a network connection, internet, or intranet.
- If your databases are behind a firewall, you must configure the firewall to permit the machine running the Database Agent program access to the databases. The database listener port (and optionally the SSH or WMI port) must be open.
- The network bandwidth used between the agent and the controller is approximately 300 KB per minute per collector for a large database with 200 clients using 50 schemas, processing about 10,000 queries a minute. The actual numbers depend on the type of database server, the number of individual schemas on the server, and the number of unique queries executed daily, and therefore varies.

Administer the Database Agent

Related pages:

- Database Visibility
- Database Agent Configuration Properties
- Add Database Collectors
- Start the Database Agent Automatically on Linux
- Start the Database Agent Automatically on Windows

The Database Agent is a standalone Java program that collects performance metrics about your database instances and database servers. You can view these performance metrics in the Metric Browser of the AppDynamics Controller UI.

To start monitoring your database, you must first install the Database Agent. You may want to install multiple agents for the following reasons:

- You have databases on multiple networks that are not accessible by one machine.
- You have multiple SQL Server instances that each require different credentials, and you want to connect to them through Windows Authentication.
- You want one or more agents to back up a primary agent. Backup agents ensure that your database instances are continually monitored during agent failure or machine downtime.

Prepare to Install the Database Agent

Related pages:

- Database Visibility
- Database Agent Configuration Properties
- Add Database Collectors
- Start the Database Agent Automatically on Linux
- Start the Database Agent Automatically on Windows

Database Agent Installation Requirements

Ensure that your environment meets these requirements:

- Java >= 1.8
- The machine that you install the agent on has network access to the databases that you want to monitor.

Controller Communication Information

If you are installing using the Database Agent downloaded from the **Getting Started Wizard**, the Controller communications information is already configured for you in the agent `controller-info.xml` file.

To complete the **Getting Started Wizard - Databases**, you need:

- Controller Host Name
- Controller Port Number

If you are installing using the agent installation zip file downloaded from the AppDynamics portal, you need:

- Controller Host Name
- Controller Port Number
- Account Access Key

See Agent-to-Controller Connections.

Install the Database Agent

Related pages:

- Database Visibility
- Database Agent Configuration Properties
- Add Database Collectors
- Enable SSL and SSH for Database Agent Communications
- Start the Database Agent Automatically on Linux
- Start the Database Agent Automatically on Windows

This page describes how to manually install the Database Agent. You can alternatively follow the steps in the **Download & Install Wizard**, which you can access from the Home page in the Controller UI. The wizard simplifies the install process.

To avoid permission issues, you should install the Database Agent as the same user who will run the agent, or as an administrator on the host machine. To run the Database Agent, you must have write privileges to the logging output directory and to the `conf` directory, which are located in the agent installation directory.

Install the Agent Software

1. Download the version of the installation package that is appropriate for your OS environment from the AppDynamics Download Center (http://download.appdynamics.com).
2. Extract the zip file to the destination directory. Do not use spaces in the agent installation destination directory.
3. Log on as an administrator to the machine where you will be installing the Database Agent.

Windows

Double-click the `dbagent-x.x.x.zip` file and extract the files to `<db_agent_home>`. If necessary, you can unblock the zip file before you extract it as follows: Right-click on the zip file, select **Properties** and choose **unblock**.

There are two versions of the database agent: one that is 32-bit and one that is 64-bit *(as of 4.5.2)*. Choose a version based on your OS requirements.

Linux

Enter the following on the command line:

```
unzip dbagent-x.x.x.zip -d <db_agent_home>
```

Running Multiple Database Agents

You can have multiple Database Agents concurrently running the agent jar in the `<db_agent_home>` directory on the same machine. Some system properties may be required depending on how you'll be using the agents.

Database Agent Licensing (On-premises Only)

Obtain a `license.lic` file with Database Monitoring licensing from your sales or support representative and put the license file in the directory where you installed the Controller. After placing the license in the directory, the Controller may take a minute or two to detect the new license. Restarting the Controller forces it to detect new licenses.

Configure the Agent

You can configure properties for the Controller host name, port number, and account access key using either the `<db_agent_home>/conf/controller-info.xml` file or by adding system properties to the Database Agent startup script.

Controller Host Name

Configure using controller-info.xml: `<controller_host>`
Configure using System Properties: `-Dappdynamics.controller.hostName`
Required: Yes
Default: None

Controller Port

Configure using controller-info.xml: `<controller_port>`
Configure using System Properties: `-Dappdynamics.controller.port`
Required: Yes
Default: For On-premises Controller installations: Port 8090 for HTTP and 8181 for HTTPS communication.
For SaaS Controller service: Port 80 for HTTP and port 443 for HTTPS communication.

Account Access Key

Configure using controller-info.xml: `<account_access_key>`
Configure using System Properties: `-Dappdynamics.agent.accountAccessKey`
Required: Yes
Default: None

Optional Configurations:

- Configure the agent to use SSL, see Enable SSL for Communicating with the Controller.
- Configure the agent to use Proxy Settings, see Proxy Settings for the Controller.
- Configure the agent to run automatically when the Machine starts on Linux or Windows.
- Configure the agent to uniquely identify itself to the Controller, such as when you require multiple agents. See Multiple Agent Environment Properties.
- Configure the agent to act as a backup to another Database Agent. See Multiple Agent Environment Properties.
- Configure the logging level of the Database Agent running on the agent JVM. The attached log file shows an example of the agent log file when the agent logging is set to INFO level. This is the default. The log files are in `<db_agent_home>\logs`.

Start the Controller Events Service (On-premises Only)

In an on-premises environment, for monitoring the database you must start the controller events service before starting Database Agent.

Linux

```
bin/platform-admin.sh start-events-service
```

Windows

```
bin\platform-admin.exe start-events-service
```

Launch the Database Agent using System Properties

The following assumes that all the necessary parameters have been specified in the `controller-info.xml`.

Method 1: Launch Agent with Start Script

You can use a script to start the agent. This is the recommended method.

Linux

```
./start-dbagent -Xms<min_heap_size> -Xmx<max_heap_size> &
```

Windows

```
start-dbagent.bat -Xms<min_heap_size> -Xmx<max_heap_size>
```

Alternatively, you can launch the agent as a Windows Service *(as of 4.5.4)*. See Install the Database Agent as a Windows Service.

Method 2: Launch Agent with Java Command

Alternatively, you can launch the agent with these Java commands:

Linux

```
java -XX:+HeapDumpOnOutOfMemoryError -XX:OnOutOfMemoryError="kill -9 %p" -jar db-agent.jar
```

Windows

```
C:\java -Djava.library.path="<db_agent_home>\auth\x64" -Ddbagent.name="Scarborough Network Database Agent" -jar
<db_agent_home>\db-agent.jar
```

System Properties

Database Agent Name

Configure using System Properties: `-Ddbagent.name=<db_agent_name>`
Type: ASCII string, including spaces. If `<db_agent_name>` contains spaces, you must enclose the entire name in double quotes (" ").
Required: Yes, when you have multiple agents reporting to the same Controller.
Default: Default Database Agent

Java Library Path

Configure using System Properties: `-Djava.library.path=<db_agent_home>\auth\x64`
Type: ASCII string, including spaces. If `<db_agent_home>` contains spaces, you must enclose the entire name in double quotes (" ").
Required: Yes, for Windows only
Specify:
For 64-bit systems: <db_agent_home>\auth\x64
For 32-bit systems: <db_agent_home>\auth\x86

Increase the JVM Memory

To monitor multiple databases, you may need to increase the JVM memory allocation size. Increased activity on the databases you are monitoring results in increased memory usage.

Use one of the following commands to start the agent, and to initially allocate 1536 MB to the agent instead of the default of 64 MB.

For Windows 64-bit

```
C:\java -Xmx1536m -Djava.library.path="<db_agent_home>\auth\x64" -jar <db_agent_home>\db-agent.jar
```

For Linux

```
% java -Xmx1536m -jar <db_agent_home>/db-agent.jar
```

Verify the Database Agent Installation

Related pages:

- Database Agent Configuration Properties
- Agent and Controller Compatibility

Review the Agent Logs

After a successful install, your agent logs, located at `<db_agent_home>/logs`, should contain this message:

```
Started AppDynamics Database Agent Successfully
```

If the agent log file is not present, the Database Agent may not be accessing the Database Agent command properties. To troubleshoot, check the application server log file where STDOUT is logged. It will have the fallback log messages, useful for troubleshooting the agent.

Verify that the Agent is Reporting to the Controller

1. Click **Settings > AppDynamics Agents**.
2. Click the Database Agents tab.

A list for each Database Agent reporting to the Controller should display. An agent can have one of the following statuses:

- Active: The agent is running
- Passive: The agent is used as a backup for active agents

If you don't see the Database Agent, check your `controller-info.xml` properties to ensure they have specified the correct host properties.

Verify that the Agent is Running

Use the following command to verify that the agent process is running:

Linux:

```
ps -ef | grep db-agent
```

Windows:

1. Open a command-line console.
2. Start the **Task Manager** and click the Processes tab.
3. The agent process should be running. If it is not running, then stop and restart the agent.

Upgrade the Database Agent

Related pages:

- Database Agent Configuration Properties
- Agent and Controller Compatibility

Important Upgrade Notes

- Before upgrading, ensure that you have all the required user permissions as mentioned in each database collector configuration.
- If you are upgrading the Controller and agents, first upgrade the Controller and then upgrade the Database Agents.
- Shut down the Database Agent process before you install the new agent. All Database Agents that are running from the same install location need to be shut down when updating that install location.
- Download the installation package that is appropriate for your OS environment from the AppDynamics Download Center (https://download.appdynamics.com/download/).
- Back up the `<db_agent_home>` directory so you can revert to the previous installation if required. To maintain the same configuration information, you will also need the `<db_agent_home>\conf\controller-info.xml` file.
- If both the old and new agents are >= 4.2, you can keep the old agent running while installing the new agent, allowing you to upgrade the agent with no downtime. Once you see the new agent showing up as "ACTIVE" in the agents page in the Controller, stop the old agent.

Stop the Agent

Stop the agent as described for your specific installation in Start and Stop the Database Agent.

Back up the Existing Agent Directory

Make a copy of the existing agent directory, `<db_agent_home>`. Backing up allows you to revert to the previous agent installation if you need to. You can also copy over the `controller-info.xml` configuration file to the new installation to ensure the agent configuration is maintained.

Install the Agent

Install the Database Agent as described for your specific installation in Administer the Database Agent.

Copy the controller-info.xml File

To ensure the agent configuration is maintained, copy the `<backup_db_agent_home>\conf\controller-info.xml` file to the new installation directory, `<db_agent_home>\conf`.

Start the New Agent

See Start and Stop the Database Agent.

Verify the Database Agent Installation

See Verify the Database Agent Installation.

Uninstall the Database Agent

Related pages:

- Administer the Database Agent
- Database Agent Configuration Properties
- Manage App Agents

To uninstall the database agent:

1. Shut down the JVM that the database agent runs on.
2. Delete the installation directory.

If you have the agent installed the agent as a service, you must also stop the Database Agent service.

To prevent a Database Agent from connecting to the Controller, ensure that it doesn't start up. This frees the license associated with the agent in the Controller and makes it available for use by another Database Agent.

Configure the Database Agent

Once you have installed and launched the Database Agent, you can further customize the agent.

- Database Agent Configuration Properties
- Enable SSL and SSH for Database Agent Communications
- Enable Snapshot Correlation for Oracle
- Increasing JVM Memory
- Database Agent Logging

Database Agent Configuration Properties

This page describes how configure a Database Agent.

Related pages:

- Administer the Database Agent

Where to Configure Database Agent Properties

You can configure agent properties in the following locations:

- The `controller-info.xml` file located in the `<db_agent_home>/conf` directory
- The system properties (`-D options`) section in the JVM start-up command. The system properties override the settings in the `controller-info.xml` file. System properties are case-sensitive.

  ```
  java -XX:+HeapDumpOnOutOfMemoryError -XX:OnOutOfMemoryError="kill -9 %p" -jar db-agent.jar
  ```

  Alternatively, you can use a script to start the agent.

Windows	`start-dbagent.bat -Xms<min_heap_size> -Xmx<max_heap_size>`
Linux	`./start-dbagent -Xms<min_heap_size> -Xmx<max_heap_size> &`

Example Database Agent controller-info.xml File

```
        <?xml version="1.0" encoding="UTF-8"?>
<controller-info>

        <controller-host>192.10.10.10</controller-host>
        <controller-port>8090</controller-port>
    <account-access-key>165e65645-95c1-40e3-9576-6a1424de9625</account-access-key>
        <!-- The following attribute enables or disables SSL communications between the agent and the
Controller.-->
        <controller-ssl-enabled>false</controller-ssl-enabled>

        <!-- The following account-related parameters are necessary only for SaaS installations-->
        <!--account-name></account-name-->

</controller-info>
```

Example Startup Configuration Using System Properties

A bash example. Note that the system properties are case-sensitive.

```
-Dappdynamics.controller.hostName=192.168.1.20 -Dappdynamics.controller.port=8090
```

Database Agent Properties

This section describes the Database Agent configuration properties, including their `controller-info-xml` elements and their system property options.

Required System Properties

Path to the Agent jar File

Description: Provides the absolute path to the jar file.

System Property: `-jar`

Value: `<db_agent_home>db-agent-jar`

Type: ASCII string, including spaces. If `<db_agent_name>` contains spaces, you must enclose the entire name in double quotes (" ").

Required: Yes

Example: `-jar="D:\AppDynamics\Database Agent\db-agent.jar"`

Path to the Java Library

Description: Provides the absolute path to `sqlijdbc_auth.dll`.

System Property: `-Djava.library.path`

Value:

- For 64-bit systems: `<db_agent_home>\auth\x64`
- For 32-bit systems: `<db_agent_home>\auth\x32`

Type: ASCII string, including spaces. If `<db_agent_home>` contains spaces, you must enclose the entire name in double quotes (" ").

Required: Recommended. Required for SQL Server Windows Authentication on Windows 64-bit systems.

Example: `-Djava.library.path="D:\AppDynamics\Database Agent\auth\x64"`

Optional System Properties

Disable Retry on Auth Failure

Description: Stops retrying to establish JDBC connection when JDBC authentication fails.

System Property: `-Dretry.on.auth.failure=false`

Default: The system property is set to true.

Type: String

Supported Locales for Parsing Numbers

Description: Specifies which locale to use for parsing numbers. See Java 7 Supported Locales or Java 8 Supported Locales for valid values. Use the values for the Java version that you are running in your environment.

System Property: `-Ddbagent.language`

Default: The system property is set to `en-US` or `en_US`.

Type: String

Agent-Controller Communication Properties

Controller Host Property

Description: This is the host name or the IP address of the AppDynamics Controller, for example, 192.168.1.22 or myhost or myhost.abc.com. This is the same host that you use to access the AppDynamics Controller UI.

Element in controller-info.xml: `<controller-host>`

System Property: `-Dappdynamics.controller.hostName`

Environment Variable: `APPDYNAMICS_CONTROLLER_HOST_NAME`

Type: String

Default: None

Required: Yes

Controller Port Property

Description: This is the HTTP(S) port of the AppDynamics Controller. This is the same port that you use to access the AppDynamics browser-based user interface. If the Controller SSL Enabled property is set to true, specify the HTTPS port of the Controller; otherwise, specify the HTTP port. See **Controller SSL Enabled Property**.

Element in controller-info.xml: `<controller-port>`

System Property: `-Dappdynamics.controller.port`

Environment Variable: `APPDYNAMICS_CONTROLLER_PORT`

Type: Positive Integer

Default: For on premises installations, port 8090 for HTTP and port 8181 for HTTPS are the default ports the Controller listens to.
For the SaaS Controller Service, port 80 for HTTP and port 443 for HTTPS are the default ports the Controller listens to.

Required: Yes

Account Access Key Property

Description: This is the account access key used to authenticate with the Controller.

Element in controller-info.xml: `<account-access-key>`

System Property: `-Dappdynamics.agent.accountAccessKey`

Environment Variable: `APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY`

Type: String

Default: None

Required: Yes. Earlier than 4.1, this property was required only for SaaS and multi-tenant Controllers. The account access key property is now required to authenticate all agent to Controller communications.

Example: `-Dappdynamics.agent.accountAccessKey=165e65645-95c1-40e3-9576-6a1424de9625`

Multiple Agent Environment Properties

The following properties are useful when you are configuring your system to include more than one Database Agent.

Database Agent Name Property

Description: This property uniquely identifies the Database Agent to the Controller.

System Property: `-Ddbagent.name=<db_agent_name>`

Type: ASCII string, including spaces. If `<agent_name>` contains spaces, you must enclose the entire name in double quotes (" ").

Default: Default Database Agent

Required: Required when you have more than one Database Agent reporting to the Controller.

You may want to run multiple **Database Agents** under the following conditions:

- When you have databases spread across multiple networks, and one machine cannot access all the databases on all the networks. In this case, you can have a uniquely named Database Agent in each network that monitors the databases only visible on that network.
- If you have multiple SQL Server instances that each require different credentials, and you want to connect to them via Windows Authentication. When using Windows Authentication, the Database Agent will use the credentials of the currently logged in user.
- When you want one or more agents to back up a primary agent. The database agent names for your backup agents must match the database agent name of your primary agent. The last agent you launch will be the primary agent.

Example: `-Ddbagent.name="Scarborough Network Database Agent"`

Unique Host ID

Description: The Unique Host ID logically partitions a single physical host or virtual machine such that it appears to the Controller that the application is running on different machines. Set the value to a string that is unique across the entire managed infrastructure. The string must not contain any spaces. If you have a machine agent that is running on the same host as that of the database agent, then this property value must be different than that of the machine agent. If you are running multiple database agents from the same host with the same name, then this property value must be different for those agents.

System Property: `-Dappdynamics.agent.uniqueHostId`

Type: String

Default: None

Environment Variable: `APPDYNAMICS_AGENT_UNIQUE_HOST_ID`

Multi-Tenant Mode Properties

If the AppDynamics Controller is running in multi-tenant mode or if you are using the AppDynamics SaaS Controller, specify the account name and account access key for this agent to authenticate with the Controller.

If the Controller is running in single-tenant mode (the default) there is no need to configure these values.

Account Name Property

Description: This is the account name used to authenticate with the Controller.
If you are using the AppDynamics SaaS Controller, the Account Name is in the Welcome email AppDynamics sent to you.

Element in controller-info.xml: `<account-name>`

System Property: `-Dappdynamics.agent.accountName`

Environment Variable: `APPDYNAMICS_AGENT_ACCOUNT_NAME`

Type: String

Default: None

Required:

Yes for AppDynamics SaaS Controller and other multi-tenant users

No for single-tenant users

Proxy Properties for the Controller

These properties route data to the Controller through a proxy.

Proxy Host Property

Description: This is the proxy host name or IP address.

Element in controller-info.xml: Not applicable

System Property: `-Dappdynamics.http.proxyHost`

Type: String

Default: None

Required No

Proxy Port Property

Description: This is the proxy HTTP(S) port.

Element in controller-info.xml: Not applicable

System Property: `-Dappdynamics.http.proxyPort`

Type: Positive Integer

Default: None

Required: No

Proxy User Name

Description: The name of the user that is authenticated by the proxy host.

Element in controller-info.xml: Not applicable

System Property: `-Dappdynamics.http.proxyUser`

Type: String

Default: None

Required: No

Proxy Password

Description: The absolute path to the file containing the password of the user that is authenticated by the proxy host. The password must be the first line of the file.

If Use Encrypted Credentials is false, enter the password in plain text. If Use Encrypted Credentials is true, encrypt the password. See Encrypt Agent Credentials.

Element in controller-info.xml: Not applicable

System Property: `-Dappdynamics.http.proxyPasswordFile`

Type: `String`

Default: None

Required: No

Other Properties

Controller SSL Enabled Property

Description: This property specifies whether the agent should use SSL (HTTPS) to connect to the Controller. If SSL Enabled is true, set the Controller Port property to the HTTPS port of the Controller. See **Controller Port Property**.

Element in controller-info.xml: `<controller-ssl-enabled>`

System Property: `-Dappdynamics.controller.ssl.enabled`

Environment Variable: `APPDYNAMICS_CONTROLLER_SSL_ENABLED`

Type: Boolean

Default: False

Required: No

Enable SSL and SSH for Database Agent Communications

Enable SSL for the Database Agent

This page describes how to configure the Database Agent to connect to the Controller using SSL. It assumes that you use a SaaS Controller or have configured the on-premises Controller to use SSL.

The Database Agent supports extending and enforcing the SSL trust chain when in SSL mode.

Requirements

Gather this information:

- The Controller SSL port.
  - For SaaS Controllers the SSL port is 443.
  - For on-premises Controllers the default SSL port is 8181, but you may configure the Controller to listen for SSL on another port.
- The signature method for the Controller's SSL certificate:
  - A publicly known certificate authority (CA) signed the certificate. This applies for DigiCert, Verisign, Thawte, and other commercial CAs.
  - A CA internal to your organization signed the certificate. Some companies maintain internal certificate authorities to manage trust and encryption within their domain.
  - The Controller uses a self-signed certificate.

Establish Trust for the Controller's SSL Certificate

To establish trust between the Database Agent and the AppDynamics Controller, you must create an agent truststore that contains the root certificate for the authority that signed the Controller's certificate.

> If you secured your on-premises Controller with a self-signed certificate, see Keystore Certificate Extractor Utility for instructions to create the agent keystore.

1. Obtain one of the following root certificates:
   - DigiCert Global Root CA for the AppDynamics SaaS Controller
   - Root certificate for the publicly known certificate authority (CA) that signed the certificate for your on-premises Controller
   - Root certificate for the internal CA that signed the Controller certificate for your on-premises Controller
2. Run the Java keytool command to create the Database Agent truststore:

   ```
   keytool -import -alias rootCA -file <root_certificate_file_name> -keystore cacerts.jks -storepass
   <truststore_password>
   ```

   For example:

   ```
   keytool -import -alias rootCA -file /usr/home/appdynamics/DigicertGlobalRootCA.pem -keystore cacerts.jks
   -storepass MySecurePassword
   ```

   > Make note of the truststore password, you need it to configure the Database Agent.

3. Install the agent truststore to the Database Agent configuration directory:

   ```
   <db_agent_home>/conf/
   ```

Secure the Database Agent Truststore

AppDynamics recommends you take the following security measures to prevent tampering with the Database Agent truststore:

- Secure the truststore file through file system permissions:
  - Make the Database Agent truststore readable by any user.
  - Make the truststore owned by a privileged user.
  - Make the truststore writable only by the specified privileged user.
- Secure the Database Agent configuration files so that they are only readable by the agent runtime user and only writable by a privileged user:
  - Global configuration file: `<db_agent_home>/conf/controller-info.xml`.

Configure SSL System Properties for the Database Agent

1. Configure the following system properties in the versioned `controller-info.xml`: `<db_agent_home>/<version_number>/conf/controller-info.xml`. See "SSL Configuration Properties" on Database Agent Configuration Properties for full details on each property.
    - **Controller Port**: the SSL port for the Controller. 443 for AppDynamics SaaS.

      ```
      <controller-port>443</controller-port>
      ```
    - **Controller SSL Enabled:** true.

      ```
      <controller-ssl-enabled>true</controller-ssl-enabled>
      ```
    - **Controller Keystore Password:** the plain text password for the Database Agent truststore.

      ```
      <controller-keystore-password>MySecurePassword</controller-keystore-password>
      ```
    - **Controller Keystore Filename:** path of the Database Agent truststore relative to `<db_agent_home>/<version>/conf`. Required if you use a truststore other than the default `<db_agent_home>/<version_number>/conf/cacerts.jks`.

      ```
      <controller-keystore-filename>../../conf/cacerts.jks</controller-keystore-filename>
      ```

      > You can specify the Controller port and enable SSL for the Controller in the JVM startup script, but you must specify the truststore password and filename in the `controller-info.xml` file.

2. Save your change to the `controller-info.xml` file and restart the Database Agent.

Sample SSL `controller-info.xml` Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<controller-info>
        <controller-host>mycompany.saas.appdynamics.com</controller-host>
        <controller-port>443</controller-port>
        <controller-ssl-enabled>true</controller-ssl-enabled>
        <controller-keystore-password>MySecurePassword</controller-keystore-password>
        <controller-keystore-filename>../../conf/cacerts.jks</controller-keystore-filename>
        ...
</controller-info>
```

Keystore Certificate Extractor Utility

The Keystore Certificate Extractor Utility exports certificates from the Controller's Java keystore for the Database Agent truststore. It installs to the following location:

```
<agent_home>/utils/keystorereader/kr.jar
```

To avoid copying the Controller keystore to a Database Agent machine, you can run this utility from the Controller server. Access the agent distribution on the Controller at the following location:

1. Run the Keystore Certificate Extractor Utility from the Controller:

   ```
   % /<full path to application JRE>/bin/java -jar <controller_home>/appserver/glassfish/domains/domain1/appagent/<controller_version>/utils/keystorereader/kr.jar
   ```

2. Enter the following when prompted:
    - The full path to the Controller's keystore:

      ```
      Enter input keystore: <controller_home>/appserver/glassfish/domains/domain1/config/keystore.jks
      ```

    - The truststore output file name. By default the Database Agent looks for `cacerts.jks`.

      ```
      Enter output agent truststore file name: <controller_home>/appserver/glassfish/domains/domain1/config/keystore.jks
      ```

      The password for the Controller's certificate, which defaults to "changeit". If you don't include a password, the extractor applies the password "changeit" to the output truststore.

- **Example command to execute kr.jar**

```
/<full path to application JRE>/bin/java -jar kr.jar <controller_home>/appserver/glassfish/domains
/domain1/config/keystore.jks cacerts.jks <controller_certificate_password>
```

3. Install the agent trust store to the agent configuration directory:

```
<db_agent_home>/conf/
```

Enable SSH for the Database Agent

> **For Linux hosts only**
>
> Applies only when Database Agent is running on a Linux host.

When the Database Agent is running on Linux and you want to monitor hardware, except to monitor the local host, authentication is required and a password is passed between the Database Agent and the database server.

> **SSH port option**
>
> The SSH port option does not appear unless the Database Agent is running on Linux.

1. On the agent machine, generate the rsa or dsa key as follows:

Generate rsa key

```
% ssh-keygen -b 1024 -f id_rsa -t rsa
```

or

Generate dsa key

```
% ssh-keygen -b 1024 -f id_dsa -t dsa
```

This will create a rsa or dsa 1024-bit key and put the keys into `/home/<user_name>/.ssh/id_rsa` and `/home/<user_name>/.ssh/id_rsa.pub` or `/home/<user_name>/.ssh/id_dsa'` and `/home/<user_name>/.ssh/ida_dsa.pub'` files. Do not change the names of these files.

2. Copy the private key, `/home/<user_name>/.ssh/id_rsa` or `/home/<user_name>/.ssh/id_id_dsa` into the `<db-agent_home>/keys` directory.
3. On both the monitored machine and the agent machine, verify that you have the correct permissions on the `.ssh` directory, or set them as follows:

.ssh directory permissions

```
% cd /home/<user_name>
% chmod 755 .ssh
```

4. On the monitored machine, verify that you have a `/home/<user_name>/.ssh/authorized_keys` file. If you do not have this file, create the a `uthorized_keys` file on the monitored machine as follows:

Create authorized_keys file

```
% cd /home/<user_name>/.ssh
% touch .ssh/authorized_keys
```

5. Verify that you have the correct permissions to the `/home/<user_name>/.ssh/authorized_keys` file, or change the permissions as follows:

authorized_keys file permissions

```
% cd /home/<user_name>/.ssh
% chmod 644 authorized_keys
```

6. Append (do not copy) the file 'id_rsa.pub' or the file 'id_dsa.pub' to the file, /home/<user_name>/.ssh/authorized_keys, such as follows:

Append rsa public key to authorized key

```
% echo /home/<user_name>/.ssh/id_rsa.pub >> /home/<user_name>/.ssh/authorized_keys
```

or

Append dsa public key to authorized key

```
% echo /home/<user_name>/.ssh/id_dsa.pub >> /home/<user_name>/.ssh/authorized_keys
```

7. The SSH port of the database Collector is set to 22 by default. You can change it by navigating to the Monitoring Hardware section of the Collector configuration dialog. In the dialog, set the SSH port to the port your require.
8. Save your change to the Collector configuration and restart the Database Agent.

Enable SSH via PEM certificate

For Linux, AppDynamics also supports certificate-based authentication via Privacy Enhanced Mail (PEM). To implement certificate-based authentication:

1. Enable the **Use certificate** option in the Monitoring hardware section of the Collector configuration dialog.
2. Copy the PEM file to the <db_agent_home>/keys directory. Note, if the home/<user_home>/.ssh directory exists, the agent will use the certificate found there.
3. Restart the agent.

Monitor SSL-enabled PostgreSQL on Amazon RDS

To monitor a PostgreSQL instance that uses SSL connections, complete the following steps.

1. Create the `.postgresql` directory under the system home folder, then create a file, `root.crt` in the directory.
   `/home/<name>/.postgresql/root.crt`
2. Download PEM file from Amazon and copy to a local directory.
3. Convert the PEM file to a DER file using the following `openssl` command:

   ```
   openssl x509 -outform der -in rds-combined-ca-bundle.pem -out rds-combined-ca-bundle.der
   ```

4. Add the certificate to the Java keystore using the following command:

   ```
   sudo keytool -import -noprompt -trustcacerts -alias AmazonRDS -file rds-combined-ca-bundle.der -keystore
   $JAVA_HOME/jre/lib/security/cacerts -storepass changeit
   ```

   You can verify that the certificate was added by running the following command:

   ```
   keytool -list -keystore $JAVA_HOME/jre/lib/security/cacerts -storepass changeit
   ```

5. Re-start the dbagent process to register the certificate you added.
6. In the Controller, create a new collector for PostgreSQL.
   a. In the **Custom JDBC Connection String** field, enter the following JDBC string:
      `jdbc:postgresql://<RDS-Hostname>:<RDS-Port>/postgres?ssl=true`
   b. In the **ADVANCED OPTIONS** section, select **Connection Properties** and then specify these property details:

Property Name	Property Value
`sslrootcert`	The location of the downloaded PEM file
`sslmode`	`verify-full`

Monitor SSL-enabled MySQL on Amazon RDS

To monitor a MySQL instance that uses SSL connections, enable MySQL in SSL mode, consulting the online documentation appropriate to your deployment.

Download the files shown below as they are required to complete this procedure:

- `ca.pem`
- `server-cert.pem`
- `server-key.pem`
- `client-cert.pem`
- `client-cert.key`

Run the following commands at the command line. Substitute URLs and other information from your deployment for the placeholders in the examples.

1. Test your local SSL connection:

```
mysql -h ec2-11-111-111-11.us-west-2.compute.amazonaws.com -u Testssl --ssl-ca=/etc/certs/ca.pem --ssl-
cert=/etc/certs/server-cert.pem --ssl-key=/etc/certs/server-key.pem -p
```

2. Verify the remote connection:

```
mysql -h ec2-11-111-111-11.us-west-2.compute.amazonaws.com -u Testssl --ssl-ca=/home/appdynamics/cert/ca.
pem --ssl-cert=/home/appdynamics/cert/client-cert.pem --ssl-key=/home/appdynamics/cert/client-key.pem -p
```

3. Import the `ca.pem` file to the default truststore:

```
sudo keytool -importcert -alias MySQLCACert -file ca.pem -keystore $JAVA_HOME/jre/lib/security/cacerts -
storepass changeit
```

   - Do not forget the password, as you need it in step 7

4. Convert the client key and certificate files to a PKCS #12 archive:

```
openssl pkcs12 -export -in client-cert.pem -inkey client-key.pem -name "mysqlclient" -passout pass:
changeit -out client-keystore.p12
```

5. Import the client key and certificate into a Java keystore:

```
sudo keytool -importkeystore -srckeystore client-keystore.p12 -srcstoretype pkcs12 -srcstorepass
changeit -destkeystore $JAVA_HOME/jre/lib/security/cacerts -deststoretype JKS -deststorepass changeit
```

6. Verify that the certificate was added:

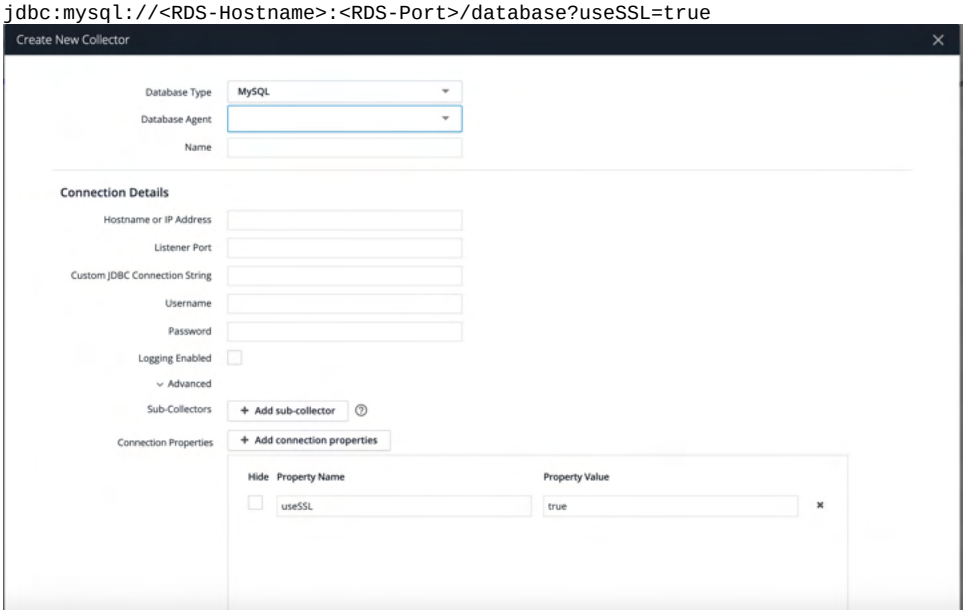
```
keytool -list -keystore $JAVA_HOME/jre/lib/security/cacerts -storepass mypassword
```

7. Start the db-agent after ensuring that the following properties are added:

```
/<full path to application JRE>/bin/java -jar -Djavax.net.ssl.trustStore=$JAVA_HOME/jre/lib/security
/cacerts -Djavax.net.ssl.trustStorePassword=changeit -Djavax.net.ssl.keyStore=$JAVA_HOME/jre/lib/security
/cacerts -Djavax.net.ssl.keyStorePassword=changeit db-agent.jar
```

8. In the Controller, create a new collector for MySQL
   - In the Configuration panel, add the connection property with Property Name "*useSSL*" and Property Value "*true*"
   - Alternatively, you can also use the following custom connection string:

```
jdbc:mysql://<RDS-Hostname>:<RDS-Port>/database?useSSL=true
```



Monitor SSL-enabled MongoDB

You can configure your MongoDB instance to use SSL connections.

1. Create a keystore on the database agent host.
2. Import the SSL certificates to this keystore using the following command:

```
keytool -importcert -trustcacerts -file <mongodb certificate location> -keystore <keystore e.g. path
/mongo.jks> -storepass <keystore password> -alias <alias>
```

3. Start the database agent with the following flags:

```
java -Ddbagent.name=<DBAgent Name> -Djavax.net.ssl.trustStore=<keystore e.g. path/mongo.jks> -Djavax.net.
ssl.trustStorePassword=<keystore password> -jar db-agent.jar
```

4. Create the mongodb collector with custom URL:

```
mongodb://<mongodb-host>:27017/?ssl=true
```

Enable Snapshot Correlation for Oracle

For the Oracle Collector AppDynamics Database Visibility to collect database queries, sessions, clients, and schemas information that occurred during snapshots captured by the Java Agent, you need to set the `jdbc-dbcam-integration-enabled` App Agent property for the Java Agent.

For information on the `jdbc-dbcam-integration-enabled` property, see App Agent Node Properties.

For information on snapshot correlation between the Oracle database Collector and a Java business transaction, see Access Database Visibility from Application Monitoring Views

Increasing JVM Memory

To monitor multiple databases, you may need to increase the JVM memory allocation size. Increased activity on the databases you are monitoring results in increased memory usage.

Use one of the following commands to start the agent, and to initially allocate 1536 MB to the agent instead of the default of 64 MB.

For Windows 64-bit

```
C:\java -Xmx1536m -Djava.library.path="<db_agent_home>\auth\x64" -jar <db_agent_home>\db-agent.jar
```

For Linux

```
% java -Xmx1536m -jar <db_agent_home>/db-agent.jar
```

Database Agent Logging

By default, the Database Agent writes log files to the */log* directory.

Configure Agent Logging

You can configure the agent to generate log files.

1. From the agent home directory, navigate to log directory.
2. Open `log4j.xml`.
3. Find `logger name="com"`. Inside this tag, set `level value` to `debug`, as shown below:

```
<logger name="com">
   <level value="debug"/>
</logger>
```

4. Let the agent run for 10 minutes to generate some logs.
5. From the agent home directory, navigate to the */log* directory to view your logs.

Start and Stop the Database Agent

Start the Agent

If you are using an on-premises version of Database Visibility, you must start the Events Service before you start the Database Agent. Before starting the agent, you must configure the properties in this file. You must specify these properties:

- Controller Host Property
- Controller Port Property
- Account Access Key Property

Start the Agent in a Command Shell

The following assumes that all the necessary parameters have been specified in the `controller-info.xml`. The `controller-info.xml` file describes the properties of agent to Controller communications. Note that `<db_agent_home>` is the absolute path to the location where the Database Agent is installed. Setting the heap size is recommended as shown below. See Database Visibility System Requirements.

Windows

```
cd <db_agent_home>
start-dbagent.bat -Xms<min_heap_size> -Xmx<max_heap_size>
```

Linux

```
cd <db_agent_home>
./start-dbagent -Xms<min_heap_size> -Xmx<max_heap_size> &
```

Start the Agent as a Service

See Install the Database Agent as a Windows Service, Start the Database Agent Automatically on Windows, and Start the Database Agent Automatically on Linux.

Stop the Agent

Stop the Agent in a Command Shell

If the Database Agent process is running in the background, you can stop it by simply entering the kill command with the process ID as the argument. If it is running in the foreground in a console, you can press Ctrl+C to shut down the agent.

Stop the Agent Service in Windows

In the Windows Services application, select **AppDynamics Database Agent** and click **Stop**.

Start the Database Agent Automatically on Linux

Related pages:

- Install the Database Agent
- Database Agent Configuration Properties

To enable the Database Agent to start automatically whenever the machine starts up:

1. Create an initialization script that starts the agent, as in the sample initialization script that is an attachment to this page.

   In your script, set the `JAVA` and `AGENT_HOME` values to the paths for your system. Also, configure agent options and heap size, as needed.

2. Enable execution permissions for the script. For example, given an initialization script named `db-agent`, enter:

   ```
   sudo chmod 775 db-agent
   ```

3. Place the script in the initialization directory on your system, typically `/etc/init.d`. Alternatively, create a symbolic link to the script from the `init.d` directory to the script if you want to keep it in another location.
4. Add the script as a service as follows:

   - For Red Hat and most Linux Operating Systems, run these commands, replacing `db-agent` with the name of your file or symbolic link:

     ```
     chkconfig --add db-agent
     chkconfig --level 2345 db-agent on
     ```

   - For Ubuntu and Debian Operating Systems, run this command, replacing `db-agent` with the name of your file or symbolic link:

     ```
     update-rc.d -f db-agent start 99 2 3 4 5 .
     ```

     In the command:

     - `start` is the argument given to the script (start, stop).
     - 99 is the start order of the script (1 = first one, 99 = last one)
     - 2 3 4 5 are the runlevels to start
     - Be sure to include the dot (.) at the end of the command.

The Database Agent now starts automatically upon machine startup. When setting the agent to automatically start up using Task Scheduler when the machine starts or restarts, ensure the agent runs with the highest privileges available.

Start the Database Agent Automatically on Windows

Related pages:

- Administer the Database Agent
- Database Agent Configuration Properties
- Database Visibility

Most Windows server software runs as a service when the machine boots up. You can create a Windows service to run the Database Agent by executing a script bundled with the product.

Install the Database Agent as a Windows Service

To create the Database Agent as a Windows service, run the following command as an administrator:

```
cscript <db_agent_home>\InstallService.vbs <jvm_options>
```

<db_agent_home> is the path to the install directory.

A Windows service named `Appdynamics Database Agent` is created.

`SilentInstall` and `serviceName` are optional parameters, which can be used along with the preceding command. The following syntax shows how to add these parameters:

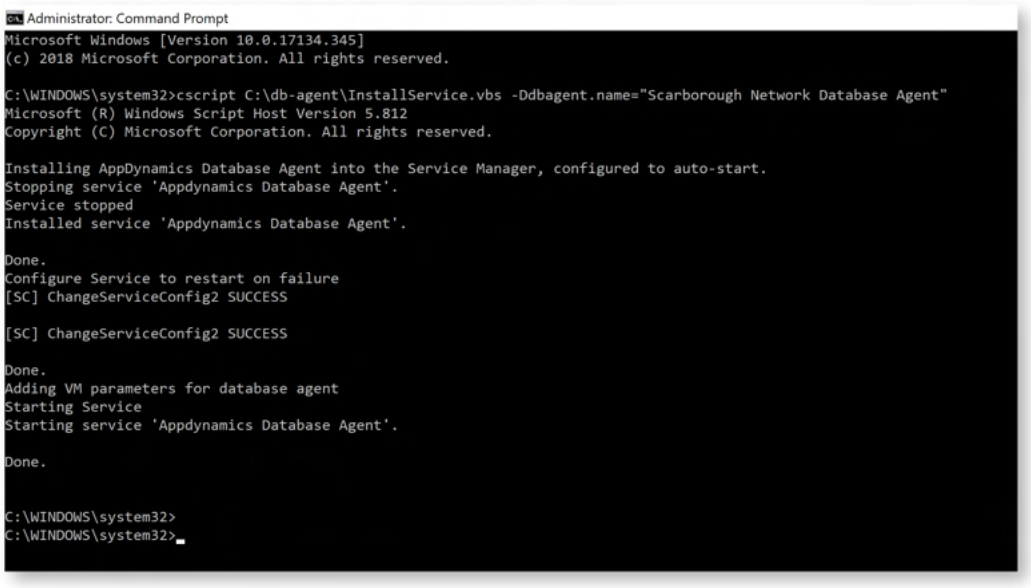
```
cscript <db_agent_home>\InstallService.vbs SilentInstall serviceName=<name_of_the_service> <jvm_options>
```

<db_agent_home> is the path to install directory, and <name_of_service> is the name that you want to use for renaming the service.

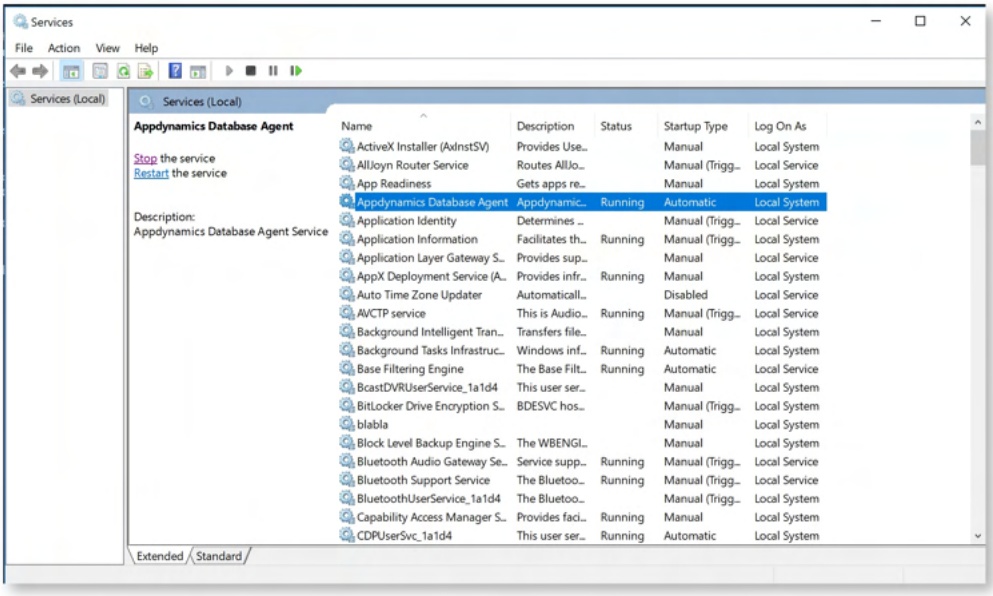
You can use `SilentInstall` to create the Database Agent as a Windows service in the silent mode where the script skips user input for installation.

For database Agent>= 20.5, you can add a service with a different name or change the name of the service by using the `serviceName` parameter. If you do not specify any value for this parameter, the service name is set to `Appdynamics Database Agent` by default.

> To use different Windows service name for different Database Agents, ensure that you run each Database Agent as a service from different install directories.



The screenshot below shows the Windows Service Control Manager.

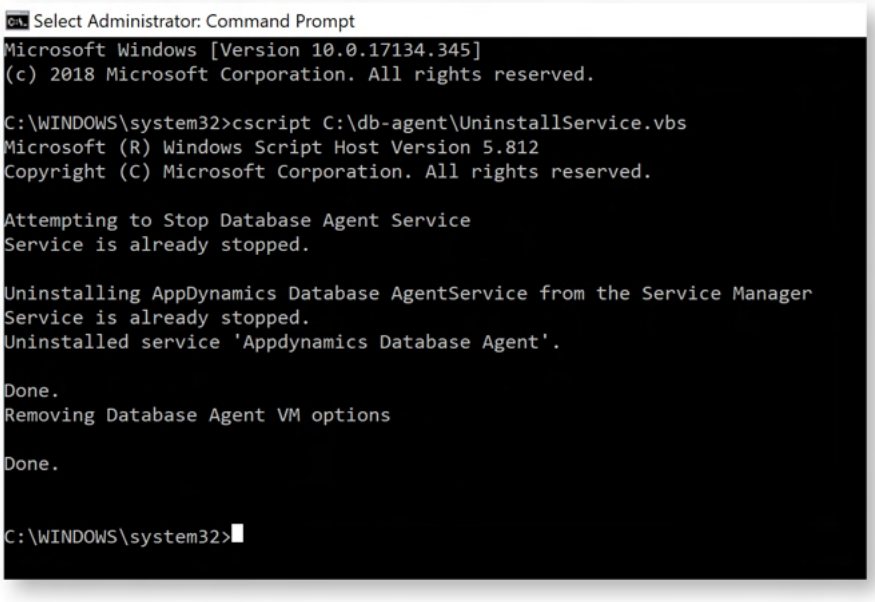


If you choose to use Windows authentication for Microsoft SQL Server, you must change your login credentials and restart the database agent service.

Uninstall the Database Agent as a Service

To uninstall the database agent service, run the following command as an administrator:

```
cscript <db_agent_home>\UninstallService.vbs
```



Running the Agent on Windows Using the Task Scheduler

It is recommended to run the Database Agent as a Windows service. However, if you choose to run the agent using Task Scheduler, ensure the agent runs with the highest privileges available. When using Task Scheduler, if the host is restarted, the agent will not start automatically.

Add Database Licenses

Related pages:

- Add Database Collectors
- License Management in the Controller

The Controller requires a license for the total number of databases to monitor concurrently. For Database Agent license details, see License Entitlements and Restrictions.

Add New Licenses

To add a new license, see Apply or Update a License File.

Manage Licenses in the Controller

To view and manage license consumption in the Controller, see License Management in the Controller.

Transfer a License Between Databases

A database license is allocated to the first agents that register with the Controller up to the licensed limit.

To transfer a license for the Database Agent to another database, you simply remove the Collector for the old database and then add a new Collector for the database you want to monitor instead. See Add Database Collectors.

If you disable a collector, it still consumes a license.

Configure Controller Settings for Monitoring Database

You can configure parameters for the controller, such as the number of events sent to the Events Service at a given time.

To change the Controller settings for Database Agents:

1. Log in to the Administration Console.
2. Click **Controller Settings**.
3. Change the settings as needed and **Save**.

Controller Settings

Property Name	Property Overview	Default	Minimum Value
`dbmon.event.publisher.esclient.batch.size`	Batch size for each Event Service client used by Database Monitoring.	1500	200
`dbmon.event.publisher.queue.size`	Maximum measurement queue size maintained by Database Monitoring.	100000	100000
`dbmon.config.data.retentionperiod`	Number of days to retain database monitoring data (queries, clients, users, sessions, schemas) excluding the current day. Data reported before the retention period might be unrecoverable, even for a momentary update of this property.	10	1
`dbmon.config.max.custommetric`	Number of custom metrics that can be created per collector.	40	0

Configure the Agent Settings for Monitoring Database

This page provides information about the different agent properties that are used for database monitoring. For configuring the basic agent properties, see Database Agent Configuration Properties.

You can update the following properties using the `controller-info.xml` file located in the `<db_agent_home>/conf` directory.

The following table includes the property names that can be used for any specific database or for all the databases along with the purpose of the properties.

Agent Properties for Monitoring Relational Databases

Property Name	Database	Purpose	Default Value
`-Ddbagent.disable.sybase.ase.system.monitoring`	Sybase	This property specifies whether the agent should disable Sybase. You may want to disable Sybase monitoring with Database Visibility if you are already using other tools to monitor Sybase. Refer to Sybase Database Permissions for information on configuring Sybase permissions. If an agent is run with the `-Ddbagent.disable.sybase.ase.system.monitoring` flag, Database Visibility will stop executing `sp_sysmon` for Sybase databases. As a result, the following metrics might not show reliable data: • Calls per Minute (KPI) • All metrics under Server Statistic in the Metric Browser • The Load value on the Sybase dashboard	false
`-Ddbagent.sampling.interval`	All relational database ⓘ This property can be used for MongoDB and Couchbase.	This is the interval at which queries are sampled. The interval can be 1s, 2s, 5s, 10s, 20s, or 30s. For example, if you have configured the property value as 2, **Elapsed Time** is displayed for 2s, 4s, 6s, and so on (in the multiples of 2).	1

Agent properties for monitoring Cassandra

Property Name	Purpose	Default Value
`dbagent.cassandra.sampling.interval.seconds`	The interval at which queries are sampled. According to the sampling interval (between 1 and 60), the sampling threshold is set (between 300 ms and 1000 ms).	10
`dbagent.cassandra.max.query.execution.time.seconds`	Maximum query execution time. Queries executing for more than this duration are not be sampled.	300
`dbagent.cassandra.query.sampling.limit`	The maximum number of queries to sample in a single sampling period. If this value is 0, query sampling is disabled.	100,000
`dbagent.cassandra.dse.skip.slow.query.table.for.sampling`	Whether to use sampling from query traces instead of the node_slow_log table for DSE Cassandra.	false
`dbagent.cassandra.apache.sampling.threshold.in.milliseconds`	Defines the sampling threshold for Apache Cassandra. In other words, queries taking more than this threshold to execute, are sampled by the database agent. ⓘ This property sets a fixed sampling threshold. To use a dynamic sampling threshold based on the sampling interval set a negative value to this property.	-1

Add Database Collectors

To enable the Database Agent to collect data, you must add and configure a Database Collector for each database that you want to monitor. The Database Collector is the process that runs within the Database Agent to collect performance metrics about your database instances and database servers.

Required Database Permissions

You must have the required permissions to monitor your databases. See Database Visibility Supported Environments.

AppDynamics Permissions

To add, edit, and create collectors, users need a role with the proper permissions:

- Can Create Collectors
- Can Edit All Collectors
- Can Delete All Collectors

See Database Permissions.

Add a Database Collector

1. Click the Configuration tab in the left panel menu.
2. Click the **Collectors** option.
3. In the Collectors panel, click **Add** (**+**).
4. Complete each field of the **Create New Collector** dialog
5. If you want to add a sub-collector to be monitored, click **Advanced** >> **+Add sub-collector** to enter host and port details.

   > Sub-collectors are applicable only to relational database types.

6. Click **OK**. Your database administrator can provide you with the necessary details.

After you have added a collector, you can configure the collector for your database. You can also link a database on the application flow maps to a database instance monitored by Database Visibility.

Hardware Monitoring

For information on the fields to complete in order for the Database Agent to monitor the server hardware in addition to the database, see Configure the Database Agent to Monitor Server Hardware.

Verify Collector Setup

Once the Collector is up and running, in just a short time you can start viewing the historical activity data.

The Collector configuration window now has a collector icon you can click to edit the details of the collector if required. The Collector will also appear in the list of Databases shown in the left navigation menu. It might take a few minutes before the Collector and its metrics are reported.

From the left navigation menu, click **Databases** to see a high-level view of the activity of all the configured Collectors.

Click the name of the database to see more details of the metrics AppDynamics Database Monitoring has captured.

For information on using and interpreting the Collector windows, see Monitor Databases and Database Servers.

Edit a Database Collector

From the Collectors panel, you can edit any of the details of the collector except the type of database platform to monitor.

For more information on collector fields, see the collector configuration page for your database.

- Configure IBM DB2 Collectors
- Configure Microsoft SQL Server Collectors
- Configure Microsoft Azure Collectors
- Configure MongoDB Collectors
- Configure MySQL Collectors
- Configure Oracle Collectors
- Configure PostgreSQL Collectors
- Configure Sybase Collectors
- Configure Sybase IQ Collectors

Delete a Database Collector

From the Collectors panel, you can delete a database Collector.

Export Database Collector Data

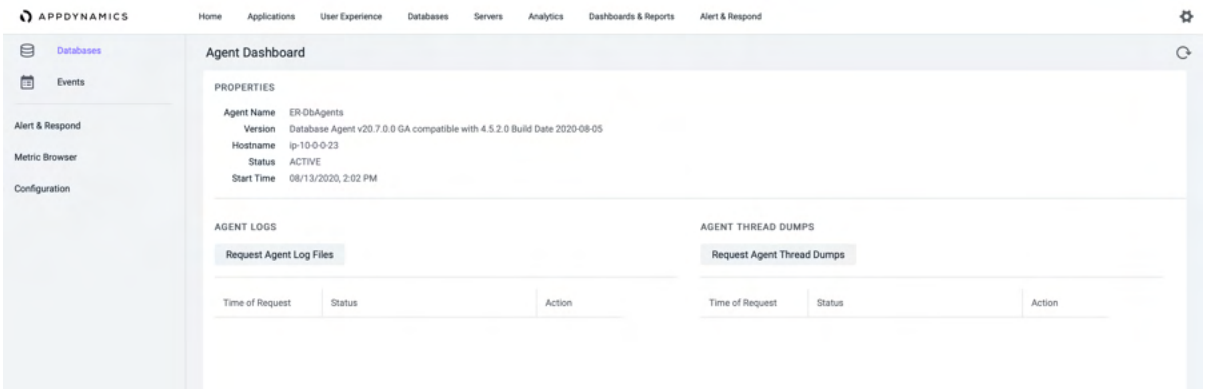
From the Collectors panel, you can click on **Actions** > **Export Data** to generate a CSV file containing the following data for each collector:

- Collector name
- Host name
- Port number
- Username
- Database type
- Agent name

View Database Agent Properties and Logs

You can view the database agent properties along with the agent logs and thread dumps by accessing **Agent Dashboard**. These logs are required for deb ugging and troubleshooting the issues.

To access **Agent Dashboard**, click the agent name under **Agent Name** on the **Collectors** page.



You can click **Request Agent Log files** to either download logs from a specific logger by setting a specific log level for a specific duration or download all logs.

You can click **Request Agent Thread Dumps** to download the required number of thread dumps.

See Agent Log Files and Request Agent Log Files.

Troubleshooting Collector Problems

Collectors that have not been configured correctly, or that cannot connect to the database for any reason, will show an error on the **Databases** overview page and individual database dashboards. Hovering over the error icon displays the potential reason for the error.

If your Collector isn't reporting any metrics after a few minutes, and you know the database is up and running with activity, check the **Events** panel. Agent Diagnostic Events can appear if the password is incorrect or communication errors have occurred. The message summaries on the **Events** panel can help you diagnose and troubleshoot Collector problems.

Check the collector configuration to ensure all the values you entered are correct.

Ensure that your Database Agent has network connection to the databases you want to monitor along with the required permissions. See Database Monitoring Requirements and Supported Environments.

Configure Cassandra Collectors

This page provides configuration details for Cassandra collectors.

Prerequisites

To monitor Cassandra with Database Visibility, you must be running version:

- Apache Cassandra >= 3.11.4
- Datastax Enterprise (DSE) Cassandra >= 6.7.3

Connection Details

	Field	Description
Create New Collector	Database Type	The Database type that you want to monitor.
	Agent	The Database Agent that manages the collector.
	Collector Name	The name you want to identify the collector by.
Connection Details	Hostname or IP Address	The hostname or IP address of the machine that your database is running on.
	Listener Port	The TCP/IP address of the port on which your database communicates with the Database Agent.
	JMX Port	The port to remotely connect through JMX (optional for DSE Cassandra). See JMX Configurations.
	JMX Username	The name of the JMX user who is connecting to and monitoring the database using the Database Agent. See JMX configurations.
	JMX Password	The password of the JMX user who is connecting to and monitoring the database through the Database Agent.
Username and Password	Username	The name of the user who is connecting to and monitoring the database using the Database Agent. The user must have the permissions described in: • User Permissions for Apache Cassandra • User Permissions for DSE Cassandra
	Password	The password of the user who is connecting to and monitoring the database through the Database Agent.
Advanced Options	SSL Connection	Click to enable SSL Connection: • Truststore Location: Location of the certificate on the DB Agent host. • Truststore Type: Type of SSL Connection. There are two `truststore` types: **PKCS12** (default) **SSO**: Enables auto-login. If you use SSO, you only need to provide the `truststore` location and `truststore` type • Truststore Password: Password for SSL Connection. If you also use client certificate authentication, then click the **Enable SSL Client Authentication** box. • Keystore Location: location of the certificate on the DB Agent host • Keystore Type: type of SSL Connection • Keystore Password: password for SSL Connection
	Monitor Operating Systems	See Configure the Database Agent to Monitor Server Hardware.

Cassandra Configurations

These are the required configurations:

- **Query Capture Configurations**
  - Apache Cassandra
  - DSE Cassandra
- **JMX Configurations for Apache/DSE Cassandra**

a. Open `cassandra-env.sh` file.
b. To enable JMX for:
    i. local connection, under the if *["$LOCAL_JMX" = "yes"];* then statement, set:

```
JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.authenticate=true"
```

    ii. for remote connection, under the corresponding else statement, set:

```
JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.authenticate=true"
```

c. Set the path to the credentials file, `jmxremote.password`:

```
JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.password.file=PATH TO FILE"
```

d. In the `jmxremote.password` file, set JMX credentials to be the same as that of the database user credentials to be specified in the collector configuration:

```
username password
```

You may choose to give ReadOnly JMX access to the database monitoring user by defining in the `jmxremote.access` file as mentioned in Enabling JMX authentication and authorization.

Database Column Families Panel

To see the Top N Queries run on a column family, double-click or single-click the column family, and then click **View Top Queries**. For more details about a query, double-click a query to open the **Query Details** panel.

Access the Column Lists Panel

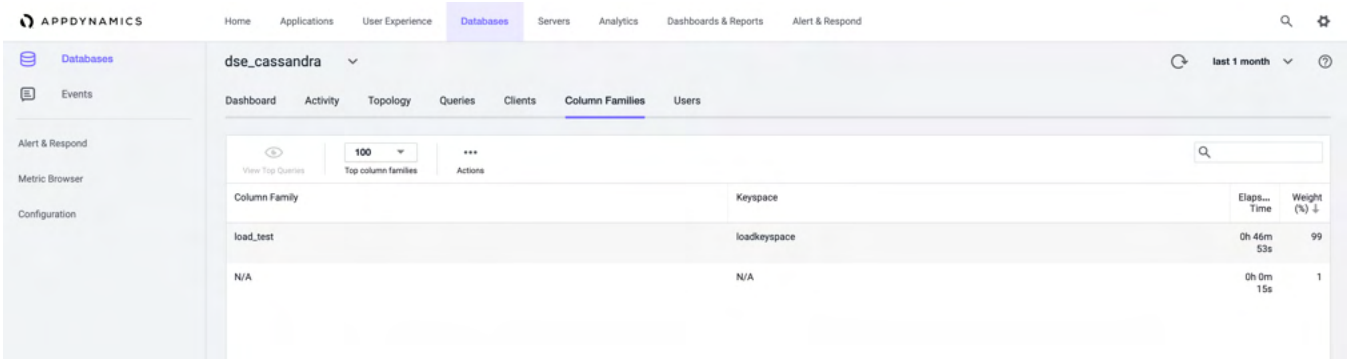
To access the **Column Lists** panel:

1. Click the name of the database from the list.
2. Click the Column Families tab. You may have to click the expand button (**>>**) to see the tab. Use the **Top column families** dropdown to view information for the top 10, 50, 100 or 200 column families sorted by total time spent in the database.

Features of the Column Families Panel

On the **Database Column Families** panel, you can see this information:

- **Column Family**: The names of the column families on which queries have been executed in this database.
- **Keyspace**: The names of the keyspaces on which queries have been executed in this database.
- **Elapsed Time**: Duration of time queries executed in the column family.
- **Weight (%)**: Time spent on executing queries on column families as a percentage of the total time spent by all column families.



Click **Actions** to export the data on this panel in a `.csv` formatted file that is automatically downloaded to your specified downloads directory.

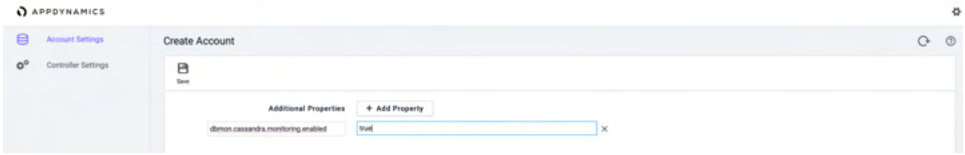
Enable Cassandra Monitoring

Before you can enable Cassandra monitoring, you must meet these requirements:

- Controller >= 4.5.17
- Database Agent >= 4.5.16
- Set the account level property to "`dbmon.cassandra.monitoring.enabled`":

To enable Cassandra monitoring :

1. Go to `<controller_url>/admin.jsp`, then click **Account Settings**. A list of accounts display.
2. Select an account from the list of options to enable Cassandra monitoring, then click **Edit**.
3. Click **+ Add Property**. Two blank fields display.
4. Enter `dbmon.cassandra.monitoring.enabled` and `true` in the first and second blank fields, respectively.



5. Select **Save**.

Query Capture Configurations for Apache Cassandra

The Database Agent captures only traced queries. To enable query tracing:

1. Turn on tracing in the CQLSH prompt
2. Enable tracing for the statement/prepared statement in the client application
3. Enable probabilistic tracing

Add this configuration to the `cassandra.yaml` file:

```
# TTL for different trace types used during logging of the repair process.
tracetype_query_ttl: 300
```

This reduces the persistence of entries in `system_traces keyspace` (sessions and events) from 24 hours to 30 minutes. You may allow longer retention based on the number of queries being traced.

User Permissions for Cassandra

These are the user permissions to configure Cassandra:

```
CREATE ROLE appdynamics_role;
GRANT SELECT ON KEYSPACE system_traces TO appdynamics_role;
GRANT appdynamics_role TO appdynamics;
```

Query Capture Configurations for DSE Cassandra

In the `dse.yaml` file, configure (or uncomment and edit) these parameters:

```
cql_slow_log_options:
    enabled: true
    #  When t > 1, log queries taking longer than t milliseconds.
    #      0 <= t <= 1,  log queries above t percentile
    threshold: 2000
    # Initial number of queries before percentile filter becomes active
    minimum_samples: 100
    ttl_seconds: 259200
     # Keeps slow queries in-memory only and doesn't write data to the database.
    # WARNING - if this is set to 'false' then set threshold >= 2000, otherwise there will be a
    # high load on the database.
    skip_writing_to_db: false
    # The number of slow queries to keep in-memory
    num_slowest_queries: 5
```

You can adjust all of these parameters, except "`enabled`" and "`skip_writing_to_db`", per the monitoring requirements.

User Permissions for DSE Cassandra

```
CREATE ROLE appdynamics_role;
GRANT SELECT ON KEYSPACE system_traces TO appdynamics_role;
GRANT SELECT ON TABLE dse_perf.node_slow_log TO appdynamics_role;
GRANT appdynamics_role TO appdynamics;
```

Configure Couchbase Collectors

To monitor Couchbase with Database Visibility, you must have:

- Couchbase >= 4.5
- Database Agent >= 4.5

Connection Details

Section	Field	Description
Create New Collector	Database Type	The database type that you want to monitor.
	Agent	The Database Agent that manages the collector.
	Collector Name	The name you want to identify the collector by.
Connection Details	Hostname or IP Address	The hostname or IP address of the machine that your database is running on.
	Listener Port	The TCP/IP address of the port on which your database communicates with the Database Agent
Username and Password	Username	The name of the user who is connecting to and monitoring the database through the Database Agent. The user should have the permissions described in User Permissions for Couchbase.
	Password	The password of the user who is connecting to and monitoring the database through the Database Agent.
Advanced Options	Exclude Databases	The databases that you want to exclude, separated by commas.
	Monitor Operating Systems	See this page for more details, Configure the Database Agent to Monitor Server Hardware.

User Permissions for Couchbase

The monitoring user must have Full Administrator or Read-only Administrator privileges. Alternatively, if you do not want the monitoring user to have Administrator privileges, you can assign a combination of Data Monitoring and Query System Catalog privileges (available for Couchbase >= 5).

Configure IBM DB2 Collectors

This page describes configuration details for IBM DB2 collectors.

> **Prerequisites**
>
> To monitor IBM DB2 with Database Visibility, you must be running DB2 >= 9.x.

Connection Details

Section	Field	Description
Create New Collector	Database Type	The database type that you want to monitor.
	Agent	The Database Agent that manages the collector.
	Collector Name	The name you want to identify the collector by.
Connection Details	Hostname or IP Address	The hostname or IP address of the machine that your database is running on.
	Database	The name of the database instance that you want to monitor. If the database instance hosts multiple databases, ensure to create separate collector for each database.
	Listener Port	The TCP/IP address of the port on which your database communicates with the Database Agent.
	Custom JDBC Connection String	The JDBC connection string generated by the database agent, for example, `jdbc:db2://`. You can also specify a custom connection string, which is useful for setting custom authentication options. If you are using the connection string for Kerberos authentication, ensure to select the **Kerberos** option under **Advanced Options**.
Username and Password	Username	The name of the user who is connecting to and monitoring the database through the Database Agent. The user should have the permissions described in User Permissions for IBM DB2 LUW.
	Password	The password of the user who is connecting to and monitoring the database through the Database Agent.
	Cyberark	Click to enable CyberArk for database username and password. When CyberArk is enabled, information about Application, Safe, Folder, and Object is required to fetch the username and password for your database. To use CyberArk with Database Visibility, you must download the `JavaPasswordSDK.jar` file from the CyberArk web site and rename the file to `cyberark-sdk-9.5.jar`. Then, you must copy the JAR file to the lib directory of the database agent zip file.
Advanced Options	Sub-Collectors	Click to monitor multiple database instances in a consolidated view, and aggregate metrics of multiple databases. To monitor a custom cluster, you can add additional hostname or IP address, and port details for each sub-collector. You can add up to a total of 29 sub-collectors. Thereby, 30 databases can be monitored in a custom cluster. In addition to the licenses consumed by the main collector, each sub-collector consumes one or more licenses, depending on the database type. **Note** • All connection parameters other than the hostname or IP address, and port details of the sub-collector are the same as the main collector. If you want to specify different parameters for the sub–collectors, while creating or editing the collector configuration, you can do that only through the Create Collector API. • You cannot convert a custom cluster collector to a standalone collector. If you want to monitor a standalone database, delete the entire custom cluster collector and create a fresh standalone collector. However, note that deleting the custom cluster collector will delete all its historical data.

	Connection Properties	Click to add a new JDBC connection property or edit an existing property for relational databases. For monitoring the database using Kerberos authentication, follow the instructions mentioned at Monitor IBM DB2 Databases Using Kerberos Authentication.
	Kerberos	Click to use Kerberos authentication to manage the database user and host user passwords.
	Monitor Operating System	See Configure the Database Agent to Monitor Server Hardware.

User Permissions for IBM DB2 LUW

The monitoring user needs SYSMON authority and connection privileges to monitor. This user must be a part of the sysmon_group.

To generate an execution plan, you must create the associated EXPLAIN tables in the schema.

DB2 >= 9.7

For complete Database Visibility functionality, the following monitoring switches of the DB2 server need to be enabled: "TIMESTAMP".

To enable the "TIMESTAMP" monitoring switch, enter:

```
update dbm cfg using dft_mon_timestamp on;
update db cfg using mon_act_metrics BASE
```

Privileges

```
grant select on SYSIBMADM.MON_CURRENT_SQL to user DBMon_Agent_User
grant select on SYSIBMADM.MON_LOCKWAITS to user DBMon_Agent_User
grant execute on function SYSPROC.MON_GET_CONNECTION to user DBMon_Agent_User
grant execute on function SYSPROC.MON_GET_PKG_CACHE_STMT to user DBMon_Agent_User
(version 10.5 and above) grant execute on function SYSPROC.MON_GET_TRANSACTION_LOG to DBAgent_User
(version 10.5 and above) grant execute on function SYSPROC.MON_GET_DATABASE to DBAgent_User
```

Replace `DBMon_Agent_User` with the user name under which you run the Database Visibility Agent.

DB2 9.5

For complete Database Visibility functionality, the following monitoring switches of the DB2 server need to be enabled: "STATEMENT", and "TIMESTAMP".

To enable these monitoring switches, enter:

```
update dbm cfg using dft_mon_stmt on;
update dbm cfg using dft_mon_timestamp on;
```

Privileges

```
grant select on SYSIBMADM.SNAPSTMT to user DBMon_Agent_User
grant select on SYSIBMADM.SNAPAPPL_INFO to user DBMon_Agent_User
grant select on table SYSIBMADM.ENV_PROD_INFO to user DBMon_Agent_User
```

where `DBMon_Agent_User` is the user name under which you run the Database Visibility Agent.

User Permissions When restrict_access is Set to YES

If your database has the `restrict_access` parameter set to YES, you must grant these privileges:

```
grant select on SYSIBMADM.MON_CURRENT_SQL to user DBMon_Agent_User;
grant select on SYSIBMADM.MON_LOCKWAITS to user DBMon_Agent_User;
grant execute on function SYSPROC.MON_GET_CONNECTION to user DBMon_Agent_User;
grant select on SYSIBMADM.SNAPAPPL_INFO to user DBMon_Agent_User;
grant EXECUTE on function SYSPROC.MON_GET_PKG_CACHE_STMT to user DBMon_Agent_User;
grant execute on function SYSPROC.MON_GET_TRANSACTION_LOG to user DBMon_Agent_User;
grant EXECUTE on package NULLID.SQLC2K26 to user DBMon_Agent_User;
grant select on SYSIBM.SYSDUMMY1 to user DBMon_Agent_User;
grant select on SYSIBMADM.ENV_PROD_INFO to user DBMon_Agent_User;
grant select on SYSIBMADM.ENV_SYS_RESOURCES to user DBMon_Agent_User;
grant execute on function SYSPROC.SNAP_GET_STMT(varchar(),Integer) to user DBMon_Agent_User;
grant select on SYSCAT.STATEMENTS to user DBMon_Agent_User;
grant select on SYSIBMADM.DBCFG to user DBMon_Agent_User;
grant execute on function SYSPROC.SNAP_GET_DB(varchar(),Integer) to user DBMon_Agent_User;
grant EXECUTE on package NULLID.SYSSH200 to user DBMon_Agent_User;
grant select on SYSIBMADM.ENV_SYS_RESOURCESto user DBMon_Agent_User;
grant select on SYSCAT.DBAUTH to user DBMon_Agent_User;
grant execute on function SYSPROC.SNAP_GET_DBM(Integer) to user DBMon_Agent_User;
grant select on syscat.schemata to user DBMon_Agent_User;
```

Replace `DBMon_Agent_User` with the user name under which you run the Database Visibility Agent.

Generate Execution Plans

To generate an execution plan in DB2, the monitoring user ID must have access to the explain_* tables.

1. Create the explain tables one of theses methods:

    - Call the `SYSPROC.SYSINSTALLOBJECTS procedure:{{}}`

      ```
      {{db2 CONNECT TO database-name
      db2 }}

      {{CALL SYSPROC.SYSINSTALLOBJECTS('EXPLAIN', 'C',
      CAST (NULL AS VARCHAR(128)), CAST (NULL AS VARCHAR(128)))}}
      ```

This call creates the explain tables under SYSTOOLS schema. To create them under a different schema, specify a schema name as the last parameter in the call.

    - Run the `EXPLAIN.DDL command file:{{}}`

      ```
      {{db2 CONNECT TO database-name
      db2 -tf EXPLAIN.DDL}}

      {{}}
      ```

Explain plans will not function properly unless the monitoring user ID is granted SELECT privilege on every table being accessed in the SQL as well as the necessary explain_* tables.

Monitor IBM DB2 Databases Using Kerberos Authentication

For the Database Agent to connect to and monitor the DB2 databases using Kerberos authentication, these settings are required:

- Start the Database Agent using the `Djava.security.krb5.conf="<path of the Kerberos configuration file>"`:

  ```
  nohup java -Djavax.security.krb5.conf="<path to Kerberos configuration file>"
  ```

- In the **CONNECTION DETAILS** section of the Collector configuration dialog, specify **Hostname or IP Address**, **Database**, **Listener Port**, and **Custom JDBC Connection String** with Kerberos details:

  ```
  jdbc:DB2:Tds:<hostname>:<portnum>?<Kerberos connection details>
  ```

- In the **ADVANCED OPTIONS** section of the Collector configuration dialog, ensure to provide these details:

  - Select **Kerberos**.
  - Select **Connection Properties**, then specify these JDBC connection properties:

Property Name	Property Value
securityMechanism	11
kerberosServerPrincipal	<Specify the name of the Kerberos Principal>

Configure Microsoft SQL Server Collectors

This page provides configuration details for Microsoft SQL Server collectors.

Prerequisites

To monitor Microsoft SQL Server with Database Visibility, you must be running the 2005 version or newer.

To configure an Azure SQL Managed instance, follow the configuration procedures of a Microsoft SQL Server Collector described in this topic.

Connection Details

Section	Field	Description
Create New Collector	Database Type	The database type that you want to monitor.
	Agent	The Database Agent that manages the collector.
	Collector Name	The name you want to identify the collector by.
Connection Details	Hostname or IP Address	The hostname or IP address of the machine that your database is running on. To monitor a cluster, specify the hostname or IP address of the listener of the cluster. See Monitor the Microsoft SQL Server Cluster to enable monitoring the MSSQL cluster.
	Listener Port	The TCP/IP address of the port on which your database communicates with the Database Agent. If you are using a Microsoft SQL Server cluster, specify the listener port of the cluster. To enable monitoring of all the nodes in a cluster, you must enable the monitoring of the cluster. This allows the agent to automatically discover all the nodes present in a cluster. See Monitor the Microsoft SQL Server Cluster
	Windows Authentication	Click to enable Windows authentication when connecting to the database.
	Custom JDBC Connection String	The JDBC connection string generated by the database agent, for example, `jdbc:sqlserver://`. You can also specify a custom connection string, which is useful for setting custom authentication options.
Username and Password	Username	The name of the user who is connecting to and monitoring the database through the Database Agent. The user should have the permissions described in User Permissions for Microsoft SQL Server.
	Password	The password of the user who is connecting to and monitoring the database through the Database Agent.
	CyberArk	Click to enable CyberArk for database username and password. When CyberArk is enabled, information about Application, Safe, Folder, and Object is required to fetch the username and password for your database. To use CyberArk with Database Visibility, you must download the `JavaPasswordSDK.jar` file from the CyberArk web site and rename the file to `cyberark-sdk-9.5.jar`. Then, you must copy the JAR file to the `lib` directory of the database agent zip file.

Advanced Options	Sub-Collectors	Click to monitor multiple database instances in a consolidated view, and aggregate metrics of multiple databases. To monitor a custom cluster, you can add additional hostname or IP address, and port details for each sub-collector. You can add up to a total of 29 sub-collectors. Thereby, 30 databases can be monitored in a custom cluster. In addition to the licenses consumed by the main collector, each sub-collector consumes one or more licenses, depending on the database type. **Note** • All connection parameters other than the hostname or IP address, and port details of the sub-collector are the same as the main collector. If you want to specify different parameters for the sub–collectors, while creating or editing the collector configuration, you can do that only via the Create Collector API. • You cannot convert a custom cluster collector to a standalone collector. If you want to monitor a standalone database, delete the entire custom cluster collector and create a fresh standalone collector. However, note that deleting the custom cluster collector will delete all its historical data.
	Connection Properties	Click to add a new JDBC connection property or edit an existing property for relational databases.
	Exclude Databases	The databases that you want to exclude, separated by commas.
	Monitor Operating System	See this page for more details, Configure the Database Agent to Monitor Server Hardware.

Note

To handle High Availability (Multi-AZ) for Amazon RDS, set the databaseName JDBC connection string property to a user database.

Monitor Microsoft SQL Server

You can monitor the SQL Server either using an existing user account or by creating a new user account, with relevant privileges.

Monitor the Microsoft SQL Server Cluster

You can monitor all the nodes in the availability group of an MSSQL database server 2012 or later.

To enable monitoring of all the nodes, you must enable the `dbagent.mssql.cluster.discovery.enabled` property either at the Controller level or at the agent level.

This property is disabled by default.

Authentication Methods

You can monitor the SQL server using either of the following authentication methods:

- Windows authenticated account (if the Database Agent is running on Windows)
- SQL Server authenticated account (if the Database Agent is running on Windows or Linux)

Before you Begin

To connect to the SQL Server database using a Windows authenticated account, perform the following:

Select **Windows Authentication** checkbox when creating the collector using "Create New Collector" dialog.

Note

Do not specify the username and password when updating database connection details.

Specify the path to the Database Agent authentication library as follows:

Version Details	Path

Windows 64-bit	`java -Djava.library.path="C:\dbagent_install_dir\auth\x64" -jar db-agent.jar`
Windows 32-bit	`java -Djava.library.path="C:\dbagent_install_dir\auth\x86" -jar db-agent.jar`

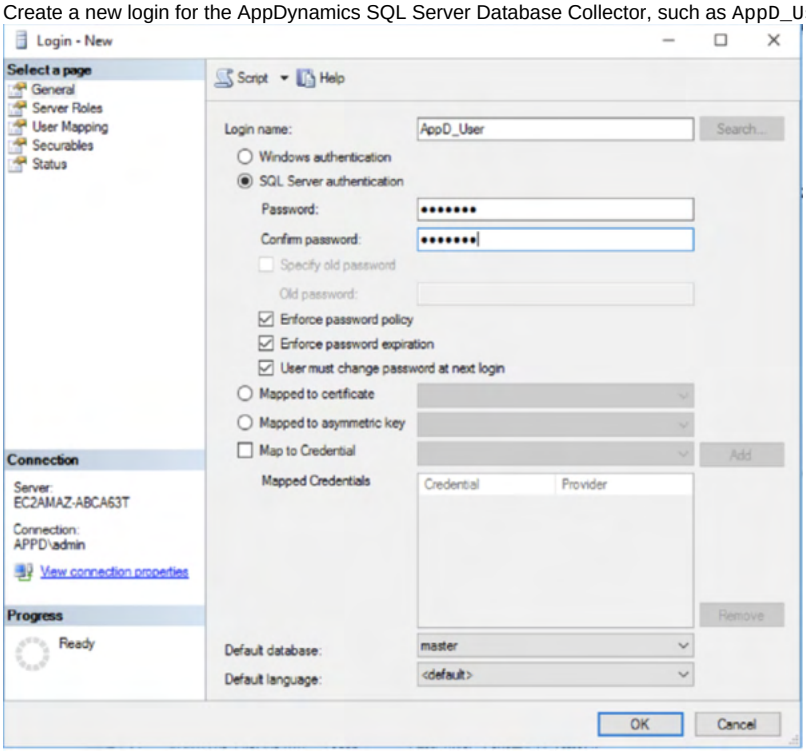
Ensure that the Windows account user has appropriate privileges to authenticate the database server and can start the database agent.

Change the logon credentials of the service to the Windows account with SQL Server access, if using a Windows service to run the Database Agent.

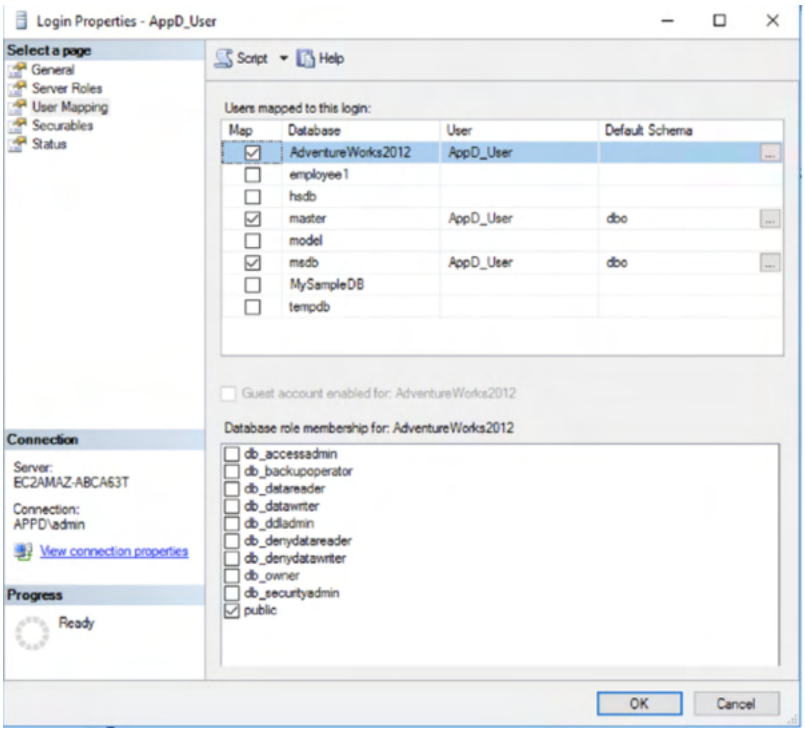
Server Level Permissions for SQL Server Logon

To create a new SQL Server user (with minimum permissions required to monitor), perform the following steps:

1. Create a new login for the AppDynamics SQL Server Database Collector, such as `AppD_User`, using SQL Server Management Studio (SSMS).



2. To map the new user to "master" and "msdb" databases, click "master" and "msdb" under **Users mapped to this login**.



> User mapping to "master" and "msdb" are mandatory for monitoring. To view object information in the Object Browser screens, additional mapping to other databases is required.

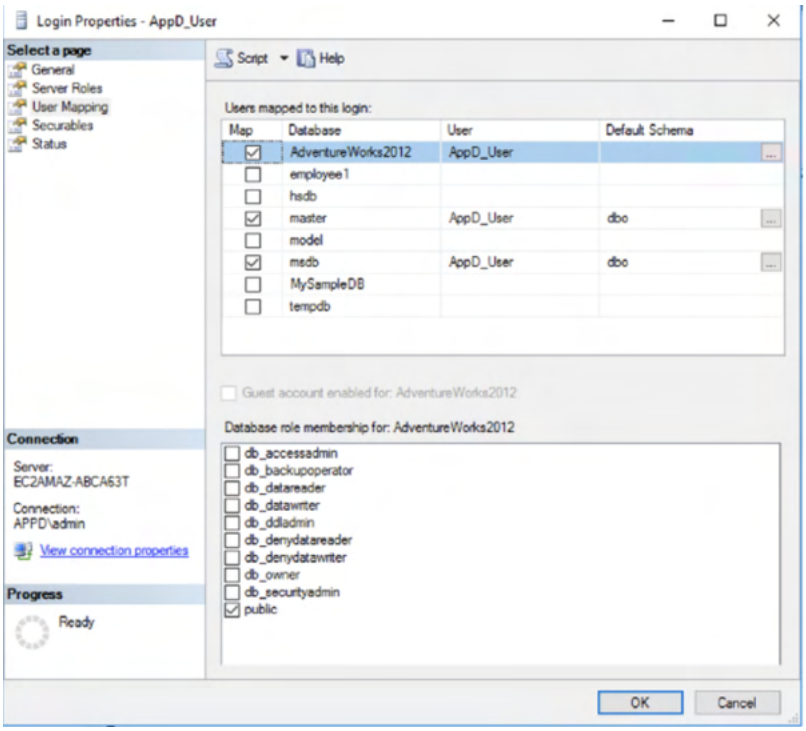
3. After creating the login, grant the following privileges to the user by substituting `AppD_User` with the name you specified on the Login - New window:

```
use master
GRANT VIEW ANY DATABASE TO AppD_User;
GRANT VIEW ANY definition to AppD_User;
GRANT VIEW server state to AppD_User;
GRANT SELECT ON [sys].[master_files] TO AppD_User;
```

Optional Object Permissions for SQL Server

The following object permissions are required for optional screens within the Database (DB) Visibility user interface:

Screen	Object Permissions
Object Browser > Users	`GRANT execute on sp_helplogins to AppD_User;` **Note** securityAdmin role is required
Object Browser > Storage	To view object storage metadata, user mapping to other databases of interest is required. **Note** public role is required

Object Browser > Job Status	``` use msdb GRANT SELECT on dbo.sysjobsteps TO AppD_User; GRANT SELECT on dbo.sysjobs TO AppD_User; GRANT SELECT on dbo.sysjobhistory TO AppD_User; ```
Object Browser > Error Log	``` GRANT execute on sp_readErrorLog to AppD_User; ``` **Note** securityAdmin role required
Object Browser > Database	To view object storage metadata, user mapping to other databases of interest is required e.g. Table/View metadata. **Note** public role required. Following image shows how users are mapped to AdventureWorks2012 database: 

Microsoft SQL Server on AWS RDS Permissions

The user account used for monitoring can be a Windows authenticated account (if the Database Agent is running on Windows) or SQL Server authenticated (if AppDynamics Database Visibility is running on Windows or Linux).

Minimum Permissions Required for SQL Server Logon

You can use the procedure to create a SQL Server user with the minimum permissions required.

Use the following to create a SQL Server logon user that provides the minimal level of permissions required in order to gain full AppDynamics Database Visibility/SQL Server functionality.

1. Using SQL Server Management Studio, create a new login for the AppDynamics SQL Server Database Collector, such as `DBMon_Agent_User`.
2. From the User Mapping tab, map the new user to the master and msdb databases.

   Viewing Object Information

   To view object information on the **Database > Objects Browser**, map the monitoring user to the databases of interest.

3. Once you have created the login, give the following privileges to the user, substituting `DBMon_Agent_User` with the name you specified on the **Login - New** panel:

You can execute the following as a batch from a query window in Management Studio. The example shows grants to `DBMon_Agent_User`; remember to change this if you have set up a different login.

```
use master
ALTER SERVER ROLE processadmin ADD MEMBER DBMon_Agent_User;
GRANT VIEW ANY DATABASE TO DBMon_Agent_User;
GRANT VIEW ANY definition to DBMon_Agent_User;
GRANT VIEW server state to DBMon_Agent_User;
```

where `DBMon_Agent_User` is the name of the SQL Server user account specified in **Create New Collector, Connection Details, Username** field.
To generate an execution plan on AWS RDS, you need additional permissions. The SHOWPLAN permission must be provided explicitly for each database:

```
USE <DB NAME>
go
GRANT SHOWPLAN to DBMon_Agent_User;
go
```

Configure Microsoft Azure Collectors

To monitor Microsoft Azure with Database Visibility, you must run Microsoft Azure >= 2008.

> To configure an Azure SQL Managed Instance, follow the configuration procedures of a Microsoft SQL Server Collector. See Configure Microsoft SQL Server Collectors.
>
> Procedures in this page are specific to an Azure SQL Database collector and **not** applicable for an Azure SQL Managed Instance.

Connection Details

Section	Field	Description
Create New Collector	Database Type	The database type that you want to monitor.
	Agent	The Database Agent that manages the collector.
	Collector Name	The name you want to identify the collector by.
Connection Details	Hostname or IP Address	The hostname or IP address of the machine that your database is running on.
	Database	The name of the database instance that you want to monitor.
	Listener Port	The TCP/IP address of the port on which your database communicates with the Database Agent
	Custom JDBC Connection String	The JDBC connection string generated by the database agent. You can also specify a custom connection string, which is useful for setting custom authentication options.
Username and Password	Username	The name of the user who is connecting to and monitoring the database through the Database Agent. The user should have the permissions described in User Permissions for Microsoft Azure.
	Password	The password of the user who is connecting to and monitoring the database through the Database Agent.
	CyberArk	Click to enable CyberArk for database username and password. When CyberArk is enabled, information about Application, Safe, Folder, and Object is required to fetch the username and password for your database. To use CyberArk with Database Visibility, you must download the `JavaPasswordSDK.jar` file from the CyberArk web site and rename the file to `cyberark-sdk-9.5.jar`. Then, you must copy the JAR file to the `lib` directory of the database agent zip file.
Advanced Options	Sub-Collectors	Click to monitor multiple database instances in a consolidated view, and aggregate metrics of multiple databases. To monitor a custom cluster, you can add additional hostname or IP address, and port details for each sub-collector. You can add up to a total of 29 sub-collectors. Thereby, 30 databases can be monitored in a custom cluster. In addition to the licenses consumed by the main collector, each sub-collector consumes one or more licenses, depending on the database type. **Note** • All connection parameters other than the hostname or IP address, and port details of the sub-collector are the same as the main collector. If you want to specify different parameters for the sub–collectors, while creating or editing the collector configuration, you can do that only via the Create Collector API. • You cannot convert a custom cluster collector to a standalone collector. If you want to monitor a standalone database, delete the entire custom cluster collector and create a fresh standalone collector. However, note that deleting the custom cluster collector will delete all its historical data.
	Connection Properties	Click to add a new JDBC connection property or edit an existing property for relational databases.

	Monitor Operating System	See this page for more details, Configure the Database Agent to Monitor Server Hardware.

User Permissions for Microsoft Azure

The user account used for monitoring can be a Windows authenticated account (if the Database Agent is running on Windows) or an SQL Server authenticated account (if AppDynamics Database Visibility is running on Windows or Linux).

Minimum Permissions Required for SQL Server Logon

You can create an SQL Server authenticated login or user with minimal level of permissions. To create a SQL Server login, you must be connected to the primary database from an admin account through SQL Server Management Studio (SSMS) or SQL Editor.

1. Run the command given below to create a login. Specify a secure password in the command.

   ```
   CREATE LOGIN DBMon_Agent_User WITH PASSWORD = 'Password123'
   ```

2. Run the following command in your Azure SQL database to create a user account for the newly created login:

   ```
   CREATE USER DBMon_Agent_User FOR LOGIN DBMon_Agent_User WITH DEFAULT_SCHEMA = dbo
   ```

3. While connected to your Azure SQL database, run the command given below to grant the pre-requisite roles and privileges:

   ```
   grant VIEW DATABASE STATE to DBMon_Agent_User
   ```

Configure MongoDB Collectors

To monitor MongoDB with Database Visibility, you must run MongoDB >= 2.2.

If you are configuring a collector for a MongoDB cluster, you only need to configure one collector for the entire cluster. You can choose any node in the cluster to connect to, and the entire cluster will automatically be detected.

Connection Details

Section	Field	Description
Create New Collector	Database Type	The database type that you want to monitor.
	Agent	The Database Agent that manages the collector.
	Collector Name	The name you want to identify the collector by.
Connection Details	Hostname or IP Address	The hostname or IP address of the machine that your database is running on.
	Listener Port	The TCP/IP address of the port on which your database communicates with the Database Agent
	Custom Connection String	The connection string generated by the database agent. You can also specify a custom connection string, which is useful for setting custom authentication options. Ensure that you do not specify the username and password in the custom connection string because it can cause a security risk.
	SRV Record	Enable this field to use the SRV record, where the connection string includes `mongodb+srv`.
Username and Password	Username	The name of the user who is connecting to and monitoring the database through the Database Agent. The user should have the permissions described in User Permissions for MongoDB.
	Password	The password of the user who is connecting to and monitoring the database through the Database Agent.
	CyberArk	Click to enable CyberArk for database username and password. When CyberArk is enabled, information about Application, Safe, Folder, and Object is required to fetch the username and password for your database. To use CyberArk with Database Visibility, you must download the `JavaPasswordSDK.jar` file from the CyberArk web site and rename the file to `cyberark-sdk-9.5.jar`. Then, you must copy the JAR file to the `lib` directory of the database agent zip file.
Advanced Options	Exclude Databases	The databases that you want to exclude, separated by commas.
	Monitor Operating System	See Configure the Database Agent to Monitor Server Hardware.

When configuring the collector, ensure that you encode the username and password that uses special characters.

User Permissions for MongoDB

For MongoDB < 2.6.*x*, the readAnyDatabase and ClusterMonitor built-in roles are required to monitor using AppDynamics Database Visibility. For MongoDB shared clusters, the monitoring user must have access to all shards.

For MongoDB >= 2.6, the `clusterMonitor` built-in role in addition to read is required.

If you choose to create a new user to monitor MongoDB, the user must be created in the admin database.

You can configure user roles as shown in the sample query below:

```
use admin
db.createUser({ user: "tanujaAdmin",
pwd: "tanuja123",
    roles: [
        { role: "clusterMonitor", db: "admin" },
          { role: "read", db: "admin" },
    ]
})
```

Configure MySQL Collectors

To monitor MySQL with Database Visibility, you must run MySQL >= 2.2.

Connection Details

Section	Field	Description
Create New Collector	Database Type	The database type that you want to monitor.
	Agent	The Database Agent that manages the collector.
	Collector Name	The name you want to identify the collector by.
Connection Details	Hostname or IP Address	The hostname or IP address of the machine that your database is running on.
	Listener Port	The TCP/IP address of the port on which your database communicates with the Database Agent
	Custom JDBC Connection String	The JDBC connection string generated by the database agent, for example, `jdbc:mysql://`. You can also specify a custom connection string, which is useful for setting custom authentication options.
Username and Password	Username	The name of the user who is connecting to and monitoring the database through the Database Agent. The user should have the permissions described in User Permissions for MySQL.
	Password	The password of the user who is connecting to and monitoring the database through the Database Agent.
	CyberArk	Click to enable CyberArk for database username and password. When CyberArk is enabled, information about Application, Safe, Folder, and Object is required to fetch the username and password for your database. To use CyberArk with Database Visibility, you must download the JavaPasswordSDK.jar file from the CyberArk web site and rename the file to cyberark-sdk-9.5.jar. Then, you must copy the JAR file to the lib directory of the database agent zip file.
Advanced Options	Sub-Collectors	Click to monitor multiple database instances in a consolidated view, and aggregate metrics of multiple databases. To monitor a custom cluster, you can add the additional hostname or IP address, and port details for each sub-collector. You can add up to a total of 29 sub-collectors. Thereby, 30 databases can be monitored in a custom cluster. In addition to the licenses consumed by the main collector, each sub-collector consumes one or more licenses, depending on the database type. **Note** • All connection parameters other than the hostname or IP address, and port details of the sub-collector are the same as the main collector. If you want to specify different parameters for the sub–collectors, while creating or editing the collector configuration, you can do that only via the Create Collector API. • You cannot convert a custom cluster collector to a standalone collector. If you want to monitor a standalone database, delete the entire custom cluster collector and create a fresh standalone collector. However, note that deleting the custom cluster collector will delete all its historical data.
	Connection Properties	Click to add a new JDBC connection property or edit an existing property for relational databases.
	Exclude Databases	The databases that you want to exclude, separated by commas.
	Monitor Operating System	See Configure the Database Agent to Monitor Server Hardware.

User Permissions for MySQL

The MySQL user the Database Agent uses to monitor the MySQL database must have "SELECT", "PROCESS", and "SHOW DATABASES" privileges on all databases.

The user must also have the "REPLICATION CLIENT" privilege to collect these metrics:

- Slave_io_running
- Slave_sql_running
- Seconds_behind_master
- SQL_Delay

If you do not have a suitable existing user, you can use the command below to create a new user.

```
GRANT SELECT,PROCESS,SHOW DATABASES on *.* to 'DBMon_Agent_User'@'host' identified by 'password';
GRANT REPLICATION CLIENT ON *.* to 'DBMon_Agent_User'@'host';
FLUSH privileges;
```

Substitute `DBMon_Agent_User` with the username under which you run the Database Visibility Agent. Substitute `host` with the hostname or IP address of the machine running the AppDynamics Database Agent, and substitute `password` with a secure password.

Set the `max_allowed_packet` parameter to 1073741824 on the server.

Troubleshooting MySQL Permissions

View MySQL Error Logs

To view the error log file, these conditions must be met:

- `secure_file_priv` is either empty or the name of the folder containing the error log file. You must restart MySQL for changes in the `secure_file_priv` variable to take effect.
- The error log file is readable to all users.
- `max_allowed_packet` is larger than the error log file size.
- The user has FILE privileges.

Configure Oracle Collectors

To monitor Oracle with Database Visibility, you must be running Oracle 10g or newer.

If you are configuring a collector for Oracle RAC, you only need to configure one collector for the entire cluster. You can choose any node in the cluster to connect to, and the entire cluster will automatically be detected.

If you are using Oracle 12c as PDB, the query for system metrics is available from Oracle Database 12c Release 2 (12.2.0.1) onwards. See the Oracle documentation for more information.

Connection Details

Section	Field	Description
Create New Collector	Database Type	The database type that you want to monitor.
	Agent	The Database Agent that manages the collector.
	Collector Name	The name you want to identify the collector by.
Connection Details	Hostname or IP Address	The hostname or IP address of the machine that your database is running on.
	Listener Port	The TCP/IP address of the port on which your database communicates with the Database Agent
	SID or Service Name	The SID or service name of the Oracle instance you want to monitor. For CDB (container database) monitoring, provide the CDB service name and for PDB (pluggable database) monitoring, provide the PDB service name. For PDB (pluggable database) monitoring, the service name must be present in all the nodes that have PDB so that all the nodes get monitored.
	Custom JDBC Connection String	The JDBC connection string generated by the database agent, for example, `jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=`. You can also specify a custom connection string, which is useful for setting custom authentication options. When using connection string for Kerberos, ensure that you select the **LDAP/Kerberos** option under **Advanced Options**.
Username and Password	Username	The name of the user who is connecting to and monitoring the database through the Database Agent. The user should have the permissions described in User Permissions for Oracle.
	Password	The password of the user who is connecting to and monitoring the database through the Database Agent.
	Cyberark	Click to enable CyberArk for database username and password. When CyberArk is enabled, information about Application, Safe, Folder, and Object is required to fetch the username and password for your database. To use CyberArk with Database Visibility, you must download the JavaPasswordSDK.jar file from the CyberArk web site and rename the file to cyberark-sdk-9.5.jar. Then, you must copy the JAR file to the lib directory of the database agent zip file.
Advanced Options	Sub-Collectors	Click to monitor multiple database instances in a consolidated view, and aggregate metrics of multiple databases. To monitor a custom cluster, you can add additional hostname or IP address, and port details for each sub-collector. You can add up to a total of 29 sub-collectors. Thereby, 30 databases can be monitored in a custom cluster. In addition to the licenses consumed by the main collector, each sub-collector consumes one or more licenses, depending on the database type. **Note** • All connection parameters other than the hostname or IP address, and port details of the sub-collector are the same as the main collector. If you want to specify different parameters for the sub–collectors, while creating or editing the collector configuration, you can do that only via the Create Collector API. • You cannot convert a custom cluster collector to a standalone collector. If you want to monitor a standalone database, delete the entire custom cluster collector and create a fresh standalone collector. However, note that deleting the custom cluster collector will delete all its historical data.

Connection Properties	Click to add a new JDBC connection property or edit an existing property for relational databases. For monitoring the database using Kerberos authentication, follow the instructions mentioned at Monitor Oracle Databases Using Kerberos Authentication.
Exclude Schemas	The schemas that you want to exclude, separated by commas.
SSL Connection	Click to enable the SSL connection. You can use SSL Connection for authentication and signing credentials. • Truststore Location: location of the SSL certificate that is stored on the DB Agent host • Truststore Type: type of truststore used for SSL certificate. There are two truststore types: **PKCS12** (default) **SSO**: enables auto-login. If you use SSO, you only need to provide the truststore location and truststore type. • Truststore Password: password for SSL certificate. If the SSL_CLIENT_AUTHENTICATION parameter is set to true in listener.ora and sqlnet.ora, then click the Enable SSL Client Authentication box. • Keystore Location: location of the SSL certificate on the DB Agent host • Keystore Type: type of SSL certificate • Keystore Password: password of SSL certificate
LDAP/Kerberos	Use LDAP/Kerberos authentication to manage the database user and the host user passwords.
Monitor Operating System	See this page for more details, Configure the Database Agent to Monitor Server Hardware.

User Permissions for Oracle

For versions of Oracle 10g and later

The following permissions are required for the Oracle user:

- CREATE SESSION
- SELECT_CATALOG_ROLE

To create a user with these permissions, you can run the following SQL. In this SQL, change "password" to a safe and secure password, and change the tablespace names, "users" and "temp" to those available in your Oracle instance.

```
CREATE USER DBMon_Agent_User IDENTIFIED BY password
default tablespace users
temporary tablespace temp;

GRANT CREATE SESSION, SELECT_CATALOG_ROLE TO DBMon_Agent_User;
```

where DBMon_Agent_User is the user name under which you run the Database Visibility Agent.

For versions of Oracle 12c and later with Multitenant Container Database Option enabled

The following permissions are required for the Oracle user:

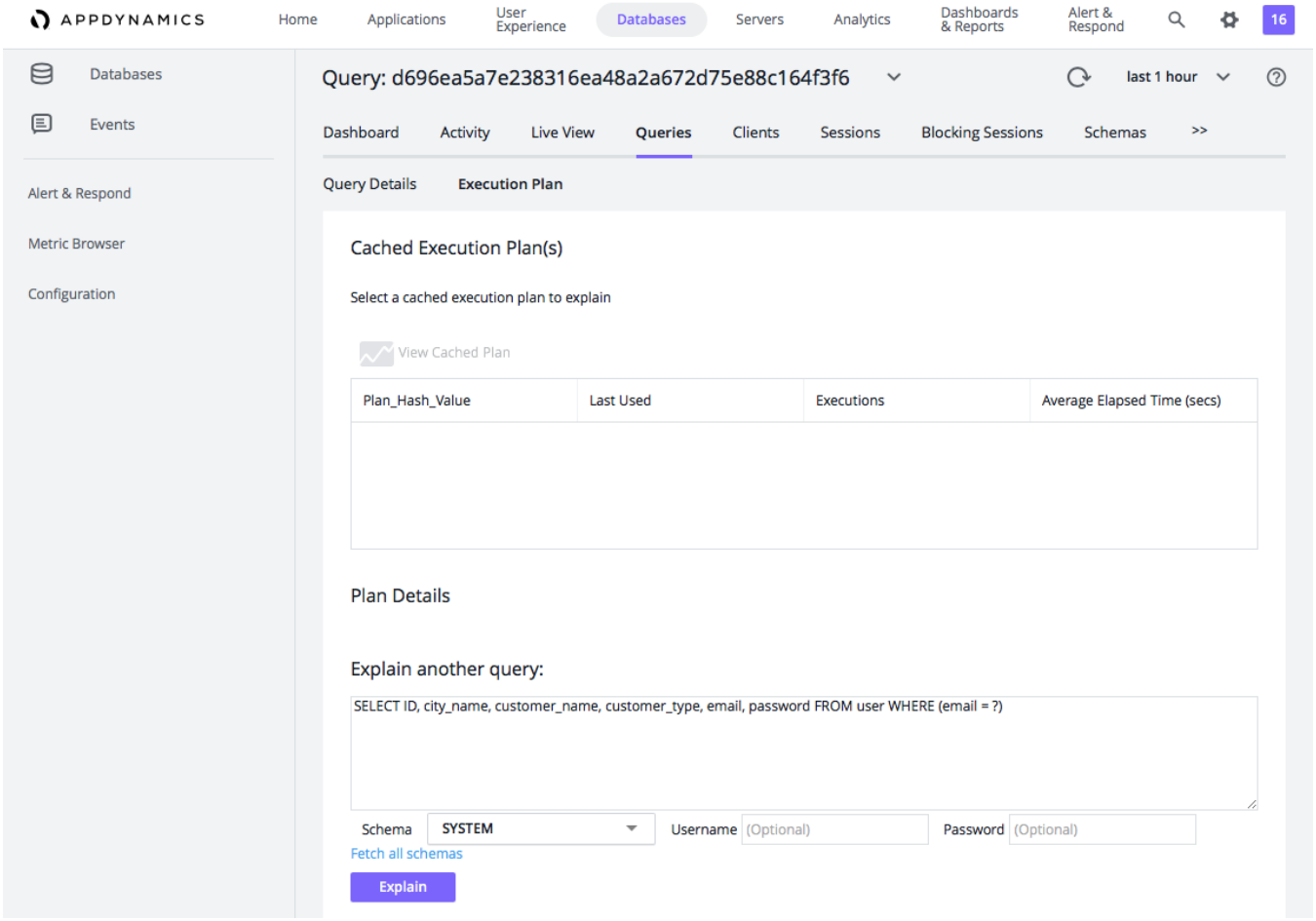
- CREATE SESSION
- SELECT_CATALOG_ROLE

To create a user with these permissions, you can run the following SQL. In this SQL, change "password" to a safe and secure password, and change the tablespace names, "users" and "temp" to those available in your Oracle instance.

```
CREATE USER C##DBMon_Agent_User IDENTIFIED BY password default tablespace users temporary tablespace temp
CONTAINER=ALL;
GRANT CREATE SESSION, SELECT_CATALOG_ROLE TO C##DBMon_Agent_User CONTAINER=ALL;
ALTER USER C##DBMon_Agent_User QUOTA 100M ON USERS;
alter user C##DBMON_AGENT_USER set container_data=all container=current;
```

Permissions Required for Oracle Explain Plans

To generate an execution plan for a specified query, paste the query text into the **Explain another query** boxy. You must provide a plan table in the AppDynamics monitoring user's schema. You can find a sample output table named PLAN_TABLE in the UTLXPLAN.SQL script. To update or view the plan table, you need INSERT and SELECT privileges. To explain a query, you need access privileges for the tables and views included in that query.



AppDynamics Database Visibility can generate explain plans within its SQL drill-down window. To enable this functionality, you must have a plan table accessible to the AppDynamics Database Visibility schema user. You can create this plan table with the following command from sqlplus when logged on as the AppDynamics Database Visibility user:

Windows:

```
@?\rdbms\admin\utlxplan.sql
```

Linux:

```
@?/rdbms/admin/utlxplan.sql
```

Permissions for individual views and tables

You can grant permission to individual views and tables even if the SELECT_CATALOG_ROLE permission is not available to your organization.

1. Execute permissions on dbms_application_info.set_module.
2. Select permissions on the following privileges:
   `dba_data_files`
   `dba_ind_columns`
   `dba_indexes`
   `dba_objects`
   `dba_procedures`
   `dba_segments`
   `dba_sequences`

```
dba_synonyms
dba_tab_columns
dba_tables
dba_users
dba_views
gv$instance
gv$containers
gv$parameter
v$archive_dest
v$archive_dest_status
v$database
v$datafile
v$event_name
v$instance
v$license
v$log
v$osstat
v$parameter
v$process
v$session
v$sesstat
v$session_wait
v$sga
v$sql
v$sql_plan
v$sqlstats
v$sqltext
v$statname
v$sysmetric
v$sysstat
v$system_event
```

When you are granting access to v$ views, you must grant SELECT on the underlying objects, which are named using the v_$ format, e.g. `GRANT SELECT on v_$archive_dest`.

Configure Oracle RAC

Related pages:

- Database Topology Window

If you are configuring a collector for Oracle RAC, you only need to configure one collector for the entire cluster. You can choose any node in the cluster to connect to, and the entire cluster is automatically detected.

If you have sub-collectors, it will disable automatic cluster discovery.

When the Database Visibility collecter connects to the hostname or IP address defined in the collector configuration, it auto-discovers the RAC instance. Once a successful connection is made to a node within the RAC, then a query on GV$INSTANCE will be used to return the details of the other member nodes and to populate the Topology Window.

When monitoring Oracle RAC, the Controller displays the aggregate data for the entire cluster.

User Permissions for Oracle RAC

When you upgrade to 4.3, you may encounter the following error: Network access denied by access control list (ACL). You can resolve the error by enabling ACLs for the monitoring user.

```
begin
dbms_network_acl_admin.create_acl (
acl             => 'Resolve_Access.xml',
description     => 'Resolve Network Access using UTL_INADDR',
principal       => '<DBMON USER>',
is_grant        => TRUE,
privilege       => 'resolve',
start_date      => null,
end_date        => null
);
commit;
end;
/

begin
dbms_network_acl_admin.assign_acl (
acl           => 'Resolve_Access.xml',
host          => '*',
lower_port    => null,
upper_port    => null);
commit;
end;
/
```

Monitor Oracle Databases Using Kerberos Authentication

For the Database Agent to connect to and monitor the Oracle databases using Kerberos authentication, these settings are required:

- Start the Database Agent using the property, `java.security.krb5.conf`:

  ```
  nohup java -Djavax.security.krb5.conf="<path to Kerberos configuration file>"
  ```

- In the **CONNECTION DETAILS** section of the Collector configuration dialog, specify the JDBC connection string:

  ```
  jdbc:Oracle:Tds:<hostname>:<portnum>?<Kerberos connection details>
  ```

- In the **ADVANCED OPTIONS** section of the Collector configuration dialog, ensure to provide these details:

  - Select **LDAP/Kerberos**.
  - Select **Connection Properties**, then specify these JDBC connection properties:

Property Name	Property Value
oracle.net.authentication_services	(KERBEROS5)
oracle.net.kerberos5_mutual_authentication	true
oracle.net.kerberos5_cc_name	"<*cache_file_path*>" For example: `"I:\\Kerberos\\krb5ccFile"`

Configure PostgreSQL Collectors

You can monitor any version of PostgreSQL with Database Visibility.

Connection Details

Section	Field	Description
Create New Collector	Database Type	The database type that you want to monitor.
	Agent	The Database Agent that manages the collector.
	Collector Name	The name you want to identify the collector by.
Connection Details	Hostname or IP Address	The hostname or IP address of the machine that your database is running on.
	Listener Port	The TCP/IP address of the port on which your database communicates with the Database Agent.
	Custom JDBC Connection String	The JDBC connection string generated by the database agent, for example, `dbc:postgresql://`. You can also specify a custom connection string, which is useful for setting custom authentication options.
Username and Password	Username	The name of the user who is connecting to and monitoring the database through the Database Agent. The user should have the permissions described in User Permissions for PostgreSQL.
	Password	The password of the user who is connecting to and monitoring the database through the Database Agent.
	CyberArk	Click to enable CyberArk for database username and password. When CyberArk is enabled, information about Application, Safe, Folder, and Object is required to fetch the username and password for your database. To use CyberArk with Database Visibility, you must download the `JavaPasswordSDK.jar` file from the CyberArk web site and rename the file to cyberark-sdk-9.5.jar. Then, you must copy the JAR file to the lib directory of the database agent zip file.
Advanced Options	Sub-Collectors	Click to monitor multiple database instances in a consolidated view, and aggregate metrics of multiple databases. To monitor a custom cluster, you can add additional hostname or IP address, and port details for each sub-collector. You can add up to a total of 29 sub-collectors. Thereby, 30 databases can be monitored in a custom cluster. In addition to the licenses consumed by the main collector, each sub-collector consumes one or more licenses, depending on the database type. • All connection parameters other than the hostname or IP address, and port details of the sub-collector are the same as the main collector. If you want to specify different parameters for the sub–collectors, while creating or editing the collector configuration, you can do that only via the Create Collector API. • You cannot convert a custom cluster collector to a standalone collector. If you want to monitor a standalone database, delete the entire custom cluster collector and create a fresh standalone collector. However, note that deleting the custom cluster collector will delete all its historical data.
	Connection Properties	Click to add a new JDBC connection property or edit an existing property for relational databases.
	EnterpriseDB	Click if your PostrgreSQL database installation is an EnterpriseDB distribution.
	Exclude Databases	The databases that you want to exclude, separated by commas.

	Monitor Operating System	See Configure the Database Agent to Monitor Server Hardware.

User Permissions for PostgreSQL

The monitoring user must either have a role of superuser or be granted access to the pg_stat_activity table using the technique described below.

The following script allows you to create a SECURITY DEFINER function that is owned by the superuser, and runs the query you want, thus allowing non-superusers to see the contents of `pg_stat_activity` by calling the `get_sa()` function and pg_stat_statements by calling the get_querystats( ) function respectively. The script must be executed by the superuser.

> Ensure that you create the `pg_stat_statements` extension by using the command, `create extension pg_stat_statements`.

```
CREATE FUNCTION get_sa()
RETURNS SETOF pg_stat_activity LANGUAGE sql AS
$$ SELECT * FROM pg_catalog.pg_stat_activity; $$
VOLATILE
SECURITY DEFINER;

CREATE VIEW pg_stat_activity_allusers AS SELECT * FROM get_sa();
GRANT SELECT ON pg_stat_activity_allusers TO public;
```

```
CREATE FUNCTION get_querystats()
RETURNS SETOF pg_stat_statements LANGUAGE sql         AS
$$ SELECT * FROM pg_stat_statements; $$
VOLATILE
SECURITY DEFINER;
CREATE VIEW pg_stat_statements_allusers AS SELECT * FROM get_querystats();
GRANT SELECT ON pg_stat_statements_allusers TO public;
```

The monitoring user must also be able to connect remotely to the PostgreSQL instance from the AppDynamics for Databases machine.

Configure Sybase Collectors

To monitor Sybase with Database Visibility, you must run Sybase >= 15.

Connection Details

Section	Field	Description
Create New Collector	Database Type	The database type that you want to monitor.
	Agent	The Database Agent that manages the collector.
	Collector Name	The name you want to identify the collector by.
Connection Details	Hostname or IP Address	The hostname or IP address of the machine that your database is running on.
	Listener Port	The TCP/IP address of the port on which your database communicates with the Database Agent.
	Custom JDBC Connection String	The JDBC connection string generated by the database agent, for example, `jdbc:sybase:`. You can also specify a custom connection string, which is useful for setting custom authentication options.
Username and Password	Username	The name of the user who is connecting to and monitoring the database through the Database Agent. The user should have the permissions described in User Permissions for Sybase.
	Password	The password of the user who is connecting to and monitoring the database through the Database Agent.
	CyberArk	Click to enable CyberArk for database username and password. When CyberArk is enabled, information about Application, Safe, Folder, and Object is required to fetch the username and password for your database. To use CyberArk with Database Visibility, you must download the JavaPasswordSDK.jar file from the CyberArk web site and rename the file to cyberark-sdk-9.5.jar. Then, you must copy the JAR file to the lib directory of the database agent zip file.
Advanced Options	Sub-Collectors	Click to monitor multiple database instances in a consolidated view, and aggregate metrics of multiple databases. To monitor a custom cluster, you can add additional hostname or IP address, and port details for each sub-collector. You can add up to a total of 29 sub-collectors. Thereby, 30 databases can be monitored in a custom cluster. In addition to the licenses consumed by the main collector, each sub-collector consumes one or more licenses, depending on the database type. • All connection parameters other than the hostname or IP address, and port details of the sub-collector are the same as the main collector. If you want to specify different parameters for the sub–collectors, while creating or editing the collector configuration, you can do that only via the Create Collector API. • You cannot convert a custom cluster collector to a standalone collector. If you want to monitor a standalone database, delete the entire custom cluster collector and create a fresh standalone collector. However, note that deleting the custom cluster collector will delete all its historical data.
	Connection Properties	Click to add a new JDBC connection property or edit an existing property for relational databases.
	Exclude Databases	The databases that you want to exclude, separated by commas.
	Monitor Operating System	See Configure the Database Agent to Monitor Server Hardware.

User Permissions for Sybase

For complete AppDynamics Database Visibility functionality, the monitoring user requires the permissions listed in the table below.

Permission type	Permission
Role permission	• sa_role • mon_role
Select permission	• master.dbo.monWaitEventInfo • master.dbo.sysconfigures • master.dbo.sysmonitors • master.dbo.monProcess • master.dbo.monProcessLookup • master.dbo.monProcessSQLText • master.dbo.monProcessProcedures
Execute permission	• sp_sysmon

To create a new dedicated user for AppDynamics Database Visibility, you can use following sample user creation script. Before running the script, change "password" to a more secure value.

```
exec  sp_addlogin  'DBMon_Agent_User', 'password', @defdb='master', @deflanguage='us_english',
@fullname='DBMon_Agent_User monitoring account', @auth_mech = 'ANY'
go
exec  sp_locklogin  'DBMon_Agent_User', 'unlock'
go
exec  sp_role 'grant', 'mon_role', 'DBMon_Agent_User'
go
```

where DBMon_Agent_User is the user name under which you run the Database Visibility Agent.

Also, the following configuration parameters must be set to 1 (true) to monitor the Sybase ASE database with AppDynamics Database Visibility: "enable monitoring", "wait event timing", "SQL batch capture", and "object lockwait timing". You should also set "max SQL text monitored" to at least 8192 (8kB).

Here is an example of the commands required to configure these settings:

```
sp_configure "enable monitoring", 1
go
sp_configure "wait event timing", 1
go
sp_configure "SQL batch capture", 1
go
sp_configure "object lockwait timing", 1
go
sp_configure "max SQL text monitored", 8192
go
```

If the value for "max SQL text monitored" was previously less than 4096, then increasing this setting will require that you restart the Sybase ASE instance.

To monitor Sybase >= 15.7 without the sa_role permission, run these commands:

```
use sybsystemprocs
grant execute on sp_sysmon to mon_role
```

If you choose to monitor Sybase using sp_sysmon, you may encounter the following errors:

- Thread utilization is incorrectly reported. View the official report here.
- Timeslice error in mmap64 or mda_flush_iostats. View the official report here.

Configure Sybase IQ Collectors

You can monitor any version of Sybase IQ with Database Visibility.

Connection Details

Section	Field	Description
Create New Collector	Database Type	The database type that you want to monitor.
	Agent	The Database Agent that manages the collector.
	Collector Name	The name you want to identify the collector by.
Connection Details	Hostname or IP Address	The hostname or IP address of the machine that your database is running on.
	Listener Port	The TCP/IP address of the port on which your database communicates with the Database Agent.
	Custom JDBC Connection String	The JDBC connection string generated by the database agent, for example, `jdbc/sybaseiq:`. You can also specify a custom connection string, which is useful for setting custom authentication options.
Username and Password	Username	The name of the user who is connecting to and monitoring the database through the Database Agent. The user should have the permissions described in User Permissions for Sybase IQ.
	Password	The password of the user who is connecting to and monitoring the database through the Database Agent.
	CyberArk	Click to enable CyberArk for database username and password. When CyberArk is enabled, information about Application, Safe, Folder, and Object is required to fetch the username and password for your database. To use CyberArk with Database Visibility, you must download the JavaPasswordSDK.jar file from the CyberArk web site and rename the file to cyberark-sdk-9.5.jar. Then, you must copy the JAR file to the lib directory of the database agent zip file.
Advanced Options	Sub-Collectors	Click to monitor multiple database instances in a consolidated view, and aggregate metrics of multiple databases. To monitor a custom cluster, you can add additional hostname or IP address, and port details for each sub-collector. You can add up to a total of 29 sub-collectors. Thereby, 30 databases can be monitored in a custom cluster. In addition to the licenses consumed by the main collector, each sub-collector consumes one or more licenses, depending on the database type. • All connection parameters other than the hostname or IP address, and port details of the sub-collector are the same as the main collector. If you want to specify different parameters for the sub–collectors, while creating or editing the collector configuration, you can do that only via the Create Collector API. • You cannot convert a custom cluster collector to a standalone collector. If you want to monitor a standalone database, delete the entire custom cluster collector and create a fresh standalone collector. However, note that deleting the custom cluster collector will delete all its historical data.
	Connection Properties	Click to add a new JDBC connection property or edit an existing property for relational databases.
	Exclude Databases	The databases that you want to exclude, separated by commas.
	Monitor Operating System	See Configure the Database Agent to Monitor Server Hardware.

User Permissions for Sybase IQ

To monitor with a non-DBA account, you need:

- User account with SERVER OPERATOR system privilege.
- Execution privileges on several procedures which can be granted as follows:

```
grant execute on sp_iqconnection to DBMon_Agent_User
grant execute on sp_iqtransaction to DBMon_Agent_User
grant execute on sp_iqcontext to DBMon_Agent_User
grant execute on sp_iqstatus to DBMon_Agent_User
grant execute on sp_iqcolumn to DBMon_Agent_User
grant execute on sp_iqindex to DBMon_Agent_User
```

  where DBMon_Agent_User is the user name under which you run the Database Visibility Agent.
- Grant DBA authority to DBMon_Agent_User:

```
grant DBA to DBMon_Agent_User
```

Monitor Sybase Databases Using Kerberos Authentication

For the Database Agent to connect to and monitor Sybase databases using Kerberos authentication, these settings are required:

- In the Connection Details section of the **Collector** configuration dialog, specify the JDBC connection string:

```
jdbc:sybase:Tds:<hostname>:<portnum>?REQUEST_KERBEROS_SESSION=true&SERVICE_PRINCIPAL_NAME=<myserver>
```

- Start the Database Agent using the `-Djavax.security.auth.useSubjectCredsOnly=false` option:

```
nohup java -Djavax.security.auth.useSubjectCredsOnly=false /<db_agent_home>/db-agent.jar &
```

Configure the Database Agent to Monitor Server Hardware

Related pages:

- Monitor Database Server Hardware

In addition to monitoring the database server, the Database Agent can also monitor the hardware that hosts the database server. To configure the Database Agent to monitor the database server hardware, complete the **Hardware Monitoring** section of the Collector configuration dialog.

Click **Databases > Configuration > Collectors** and then choose an existing collector to monitor the hardware for that database server and click **Edit**, or click **Add** to set up a new collector to monitor a different database server and the hardware hosting that server.

Complete the Hardware Monitoring Section

The following describes the fields in the **Collector Configuration** dialog that you must complete if you want to monitor the database server host hardware, in addition to the database server.

> For Oracle database, the **CPU** and the **Memory** metrics are monitored even when hardware monitoring is disabled.

Monitor Operating System: Select if you want CPU consumption metrics collected from the monitored host.

Operating System: Specify the operating system of the monitored host: Windows, Linux, Solaris, AIX, or RDS.

Use Local WMI: Check this box if you want to monitor the machine the Database agent is running on, i.e. localhost. When selected, the authorization fields are not used because no authorization is required.

Domain: For Windows only, specify the name of the domain in which the hardware resides.

SSH Port: For Linux, AIX, and Solaris only, specify the Secure Shell (SSH) port number the Controller should use for encrypted communications with the monitored host. The default port number of 22 will be used if you do not specify a different port number here.

Use certificate: For Linux, AIX, and Solaris only, AppDynamics also supports certificate-based authentication through Privacy Enhanced Mail (PEM). To implement certificate-based authentication, enable the **Use certificate** option and copy the PEM file to the `<db_agent_home>\keys` directory. Note, if the `$HOME/.ssh` directory exists, the agent will use the certificate found there. This option appears only if the agent is running on a machine running Linux, AIX or Solaris.

Username: Specify the name of the user the Database Agent uses to log on to the monitored host. To collect OS metrics from a Windows host, the configured user (or Collector Service user if using Windows Authentication) must be able to establish a WMI connection to the target host and collect Windows Performance Counters.

Password: Specify the password of the user the Database Agent uses to log on to the monitored host. The number of echo characters shown in the password text field should not be interpreted to imply the number of characters stored for the (encrypted) user password.

> Windows hardware monitoring is not supported by Database Agent installed on a Linux system. However, Database Visibility can support hardware metrics for SQL server through SQL query collection.

Resolve Metric Reporter Type Mismatch Problem

Switch from the Java Hardware Monitor to the Hardware Monitor if you see messages similar to the following in the logs:

```
[Worker-7] 12 Sep 2014 07:27:49,449 WARN MonitorOutputHandler - Metric Reporter type mismatch for metric
[Hardware Resources|Network|Incoming packets/sec]
com.singularity.ee.agent.commonservices.metricgeneration.metrics.MetricReporterTypeMismatchException:
Aggregator of OBSERVATION already exists for metric Metric Identifier[Hardware Resources|Network|Incoming
packets/sec] ID[0]
```

Required Monitored Host Permissions

Database Agent User Permissions

Database Visibility collects the stats from common commands like `vmstat/iostat` or gathering metrics from the file system such as `/proc` and as such the user that runs the Database Agent requires no special permissions, just the ability to run those common commands and write to files in the Database Agent directory.

Database Agent Collector Authentication

When monitoring the database host, Database Visibility must log in to the host system to gather system information. There are three ways to authenticate the Database Collector to access the monitored host:

- Specify a username and password in the Collector configuration dialog
- Place a PEM file or an id_rsa file in the `<agent home>/keys` directory
- If the Database Agent is running on LINUX, Solaris or AIX, place SSH keys in the `<home>.ssh` directory of the user running the agent. You can create an SSH key using the same procedure as described for the Controller. See Configure a Controller SSH Key.

Required SSH Permissions

When monitoring the database host, Database Visibility uses SSH on Linux and Windows systems to gather system information. SSH access can be through either an authenticated username/password or a private key.

You can create an SSH key using the same procedure as described for the Controller. See Configure a Controller SSH Key.

Configure WMI Permissions and Security

Related pages:

- WMI and the Database Agent on the AppDynamics Community

To monitor Windows-based machine hardware with AppDynamics Database Visibility, AppDynamics uses Windows Management Instrumentation (WMI) to remotely gather the metrics. WMI is often complicated to troubleshoot when the Database Agent is running on a Linux or Unix-like machine.

This page describes requirements for the target machine configuration that can help you avoid some problems and pitfalls. It also provides some additional considerations regarding using WMI to monitor a SQL Server database agent and preventing unauthorized remote access to WMI.

> **Named Windows Account:**
>
> The user specified in the collector configuration that the AppDynamics Database Agent uses to connect to the target machine is referred to as <named Windows account>.

The following are required when the Database Agent is hosted on AIX, Linux or Solaris platforms to monitor Windows >= 7 systems.

- Ensure that the named Windows account is a member of the local Administrators group.
- Ensure User Account Meets Minimum Security Requirements When Using WMI
- Enable Remote Registry Access
- Grant access to WBEM scripting locator

The following are required when the Database Agent is hosted on AIX, Linux, or Solaris platforms to monitor Windows >= 2012 systems.

- Grant full control permissions to select registry keys

The following is required when the Database Agent is hosted on Windows.

- Ensure that the named Windows account is a member of the local Administrators group.

Requirements to Monitor Windows >= 7 Systems (Agent Running on Unix-like Platform)

The following are required when the Database Agent is hosted on AIX, Linux, or Solaris platforms to monitor Windows >= 7 systems.

Ensure User Account Meets Minimum Security Requirements When Using WMI

Enable Security Options for Windows Systems are Part of a Domain

Ensure the named Windows account has the correct permissions for WMI Control.

1. Run the `wmimgmt.msc` program.
2. Right-click the **WMI Control** icon on the left and click **Properties**.
3. Click **Security.**
4. Click the root node of the tree, and click **Security**.
5. Ensure that the named user account running the Database Agent has the relevant permissions.

> The minimum permissions that your remote Windows account needs for the Database Agent are:
>
> - Execute Methods
> - Enable Account
> - Remote Enable
>
> If the named Windows account does not have all of these permissions, you may receive an access denied error or the following errors:
>
> ```
> Error=800706BA The RPC server is unavailable. SWbemLocator
> ```
>
> or
>
> ```
> Error=80070005 Access is denied SWbemLocator
> ```

Enable Classic Security Options for Local (Non-Domain) Windows Systems

Applies to Windows computers that are not part of a domain.

1. Open the **Control** panel, and go to **Administrative Tools** > **Local Security Policy**. The **Local Security Settings** panel appears.
2. Go to **Local Policies** > **Security Options.**
3. Change the value of **Network access: Sharing and security model for local accounts** to **Classic**.

Enable Remote Registry Access

The Remote Registry service must be running on the target machine. If the Remote register service is off, you will receive the following error:

```
Message not found for errorCode: 0xC0000034
```

or

```
Access is denied
```

By default Windows >= 7 systems will still deny remote access to the registry, even if the Remote Registry service is started.

To test this:

1. Attempt to access the replica registry using Regedit on another machine.
2. If you get an error similar to `Access is denied`, run PowerShell as an administrator on the replica, and execute `Enable-PSRemoting`.
3. Restart the machine and try launching the replica again.

Grant Access to WBEM Scripting Locator

The Database Agent requires full access to the WBEM Scripting Locator. On the target system, allow full access to the WBEM Scripting Locator as follows:

1. As an Administrator on the target machine, launch Regedit.
2. Locate the registry key:
   `76A64158-CB41-11D1-8B02-00600806D9B6` in `HKEY_CLASSES_ROOT\CLSID`
3. Right-click the key and click **Permissions**.
4. Click **Advanced**.
5. Click **Owner** and change the owner to the Administrators group. Click **Apply**.
6. Click **Permissions** and change the permissions for the Administrators group to **Full Control**. Click **Apply**.
7. Close Regedit.
8. Restart the Remote Registry Service, using **Administrative Tools** > **Services**.

Configure the Firewall

WMI uses RPC which listens on port 135 but then allocates a dynamic port for subsequent communication. Configure your Firewall to always allow the TCP port 135 exception and follow the dynamic RPC ports. If there is a problem with the firewall, port 135 then you will probably see this error:

```
ERROR: Message not found for errorCode: 0xC0000001
```

See How to configure RPC dynamic port allocation to work with firewalls.

Additional Requirements to Monitor Windows >= 2012 Systems (Agent Running on Unix-like Platform)

In addition to the requirements described in Requirements to Monitor Windows >=7 Systems, the following are also required when the Database Agent is hosted on AIX, Linux, or Solaris platforms to monitor Windows >= 2012 systems.

Grant Full Control Permissions to Select Registry Keys

For the Database Agent running on AIX, Linux, or Solaris to monitor Windows >= 2012 (64-bit) systems, complete the following changes on the target system.

1. As an Administrator on the target machine, launch Regedit.
2. Change the permissions for both of these registry keys to **Full Control**:
   `72C24DD5-D70A-438B-8A42-98424B88AFB8` in `HKEY_LOCAL_MACHINE\SOFTWARE\Classes\Wow6432Node\CLSID`
   `76A64158-CB41-11D1-8B02-00600806D9B6` in `HKEY_CLASSES_ROOT\CLSID`
3. Find this registry key:
   `72C24DD5-D70A-438B-8A42-98424B88AFB8` in `HKEY_LOCAL_MACHINE\SOFTWARE\Classes\Wow6432Node\CLSID`
4. Right-click and click **Permissions**.
5. Change the owner to the Administrators group.
6. Change the permissions for the Administrators group to **Full Control**.
7. Change owner back to **TrustedInstaller**. User is <NT Service\Trusted Installer> on the local machine.
8. Repeat steps 4 to 6 above for this registry key:
   `76A64158-CB41-11D1-8B02-00600806D9B6` in `HKEY_CLASSES_ROOT\CLSID`.
9. Close Regedit.
10. Restart the Remote Registry service, using **Administrative Tools** > **Services**.

General Considerations for all Platforms

This information applies to the Database Agent running on Windows systems.

Use Windows Authentication for Microsoft SQL Server

To use Windows Authentication for the Database Agent to connect to a Microsoft SQL Server database instance, you must use a command similar to the following to start the Database Agent; specifying the path to the Database Agent authentication library.

Windows 64-bit

```
java -Djava.library.path="C:\dbagent404\auth\x64" -jar db-agent.jar
```

Windows 32-bit

```
java -Djava.library.path="C:\dbagent404\auth\x86" -jar db-agent.jar
```

The Windows account used to start the Database Agent must be a Windows user who can authenticate with the database server.

Monitor Databases and Database Servers

Related pages:

- Overview of Database Visibility
- Database Visibility
- Create and Manage Tenant Users
- Top 6 Database Performance Metrics to Monitor in Enterprise Applications

This page describes how to use the features of AppDynamics to monitor and troubleshoot your database environments.

- Access Database Visibility from Application Monitoring Views
- Discover Normal Database and Server Activity
- Monitor Database Performance
- Monitor Database Server Hardware
- Database Health Rules and Alerts
- Database Monitoring Metrics
- Database Agent Events Reference
- Wait State Filtering
- Configure Custom Metrics
- Configure Query Literals Security

For roles and permissions required to view Database Monitoring windows, see Create and Manage Custom Roles.

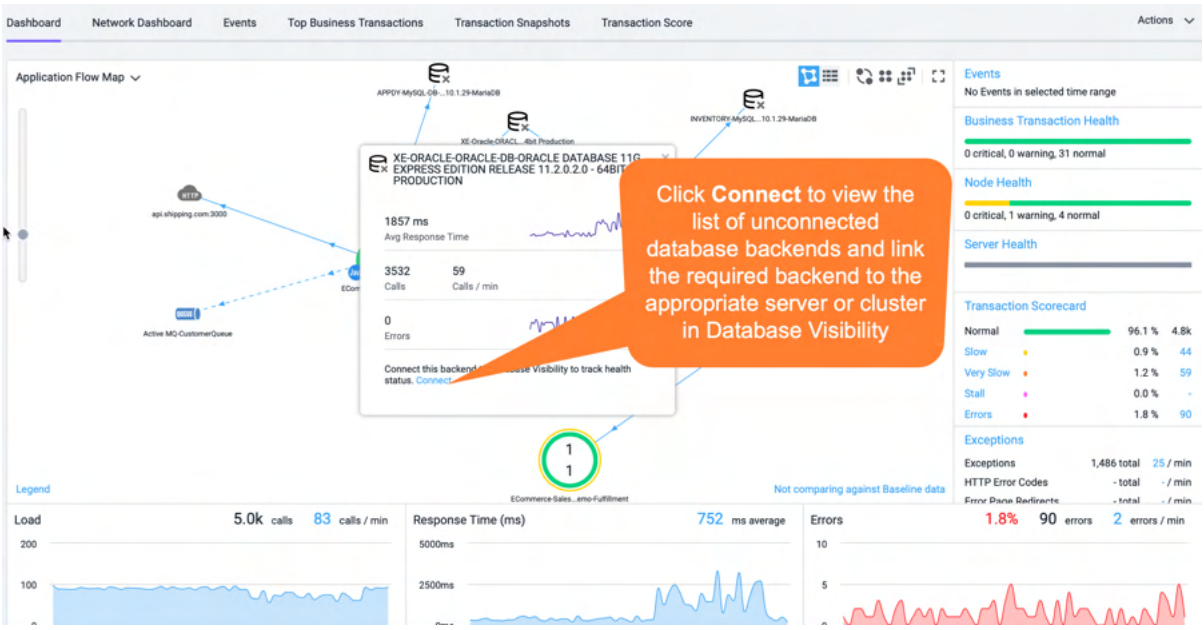
Access Database Visibility from Application Monitoring Views

Related pages:

- Monitor Database Performance
- App Agent Node Properties

To view the application dashboard of a specific application, click the application on the **Home** page. On the application dashboard, you can view the following:

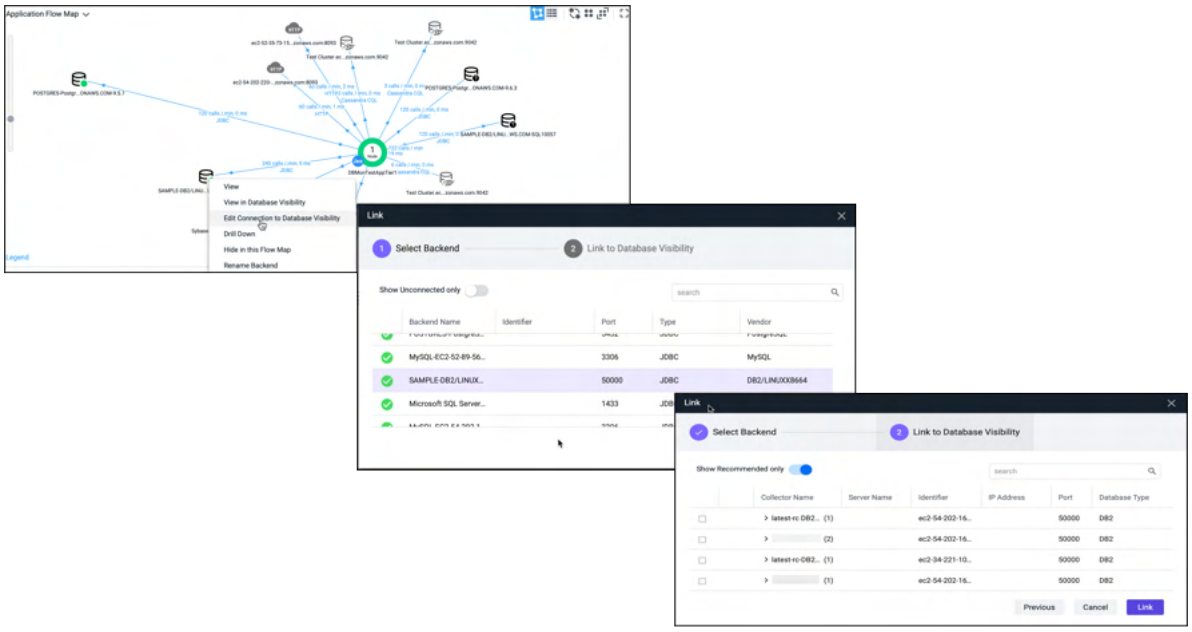
- Unconnected Backends: If there are any unconnected backend database servers, the flowmap displays the unconnected database server icon. You can click the icon, then click **Connect** to connect the backend database to the appropriate server or cluster in **Database Visibility**.
- Status of the Database: There are different icons to display different states of the database based on the health rule violation. For information about the different icons, see Database Icons.
- Quick View of **Database Visibility**: If you require to view some of the main metrics without going to the **Database Visibility** UI, you can click the database icon and the quick view of the metrics is displayed. For detailed information, click **View in Database Visibility.**



View Database Activity in Database Visibility

Once you have configured a database Collector in AppDynamics Monitoring, you can access Database Visibility either from the AppDynamics Home page or you can link to the Database Visibility database instance Dashboard by right-clicking the database and selecting **View in Database Visibility** on the Application Flow Map, Tier Flow Map or Node Flow Map.

Database backends with the same hostname, port number, and database type as a database server already configured in a database Collector are automatically matched with the Collector, and drilldowns from the Application Flow Map, Tier Flow Map or Node Flow Map to Database Visibility are enabled.



The AppDynamics Database Visibility section in the Application Database Dashboard displays which database Collector is associated with the database. If you require to change this association, right-click the database and then click **Edit Connection to Database Visibility** and choose the appropriate server or cluster that is configured in **Database Visibility**.

View Business Transaction Snapshot Correlated Database Details

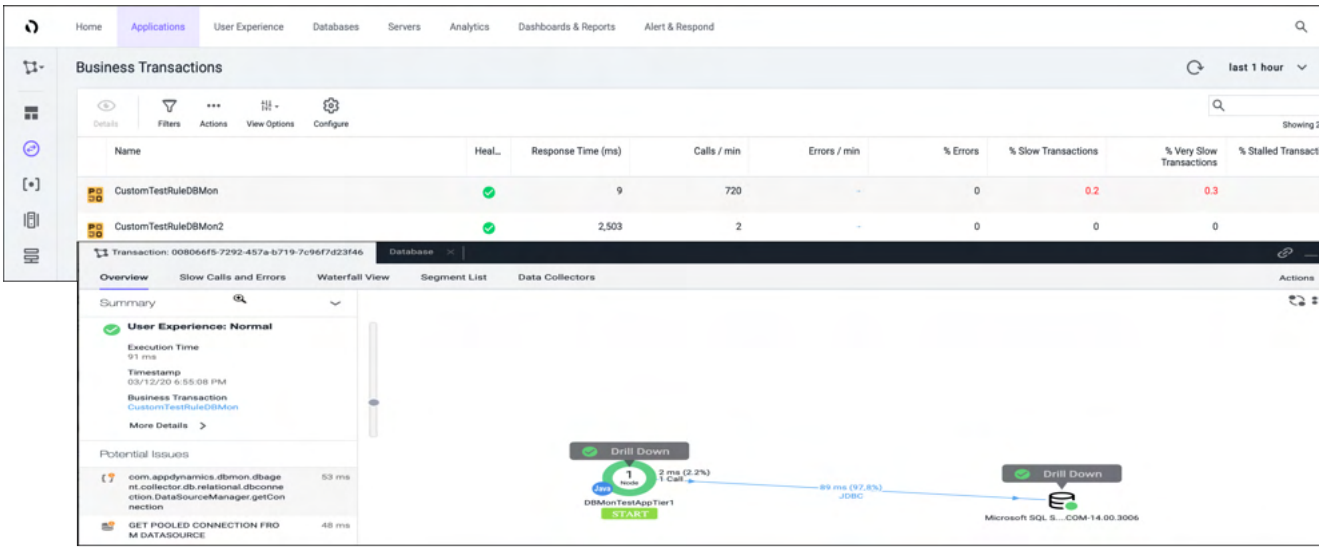
Available for Java, .NET, and PHP applications and their relational database backends, snapshot correlation shows the details of queries executed by the business transaction. It also shows the clients, sessions, and schemas in which those queries were executed.

View Database Details

1. On the Business Transactions page, double click any business transaction to view the queries that are executed in that transaction.
2. Click the DB Queries tab. On this tab, you can view the queries that were issued by this business transaction.

View Correlated Database Details

1. From a list of business transactions with snapshots, choose a long-running business transaction that accesses the database, one that takes a few seconds or more. These are more likely to have captured database details.
2. Double-click **Drill Down** above the database icon. The correlated database details window appears.
3. Click the Queries, Clients, Sessions, and Schema tabs to view details of database activity that occurred around the time when the snapshot was captured. For relational databases, these details reflect the database activity from the transaction snapshot. For non-relational databases, these details reflect the database activity that occurred around the time of the transaction snapshot.



For Oracle backends only: If you want to list only the queries executed by the Transaction Snapshot in the Transaction Snapshot window, you can enable the `jdbc-dbcam-integration-enabled` node property for the Java Agent.

Discover Normal Database and Server Activity

Related pages:

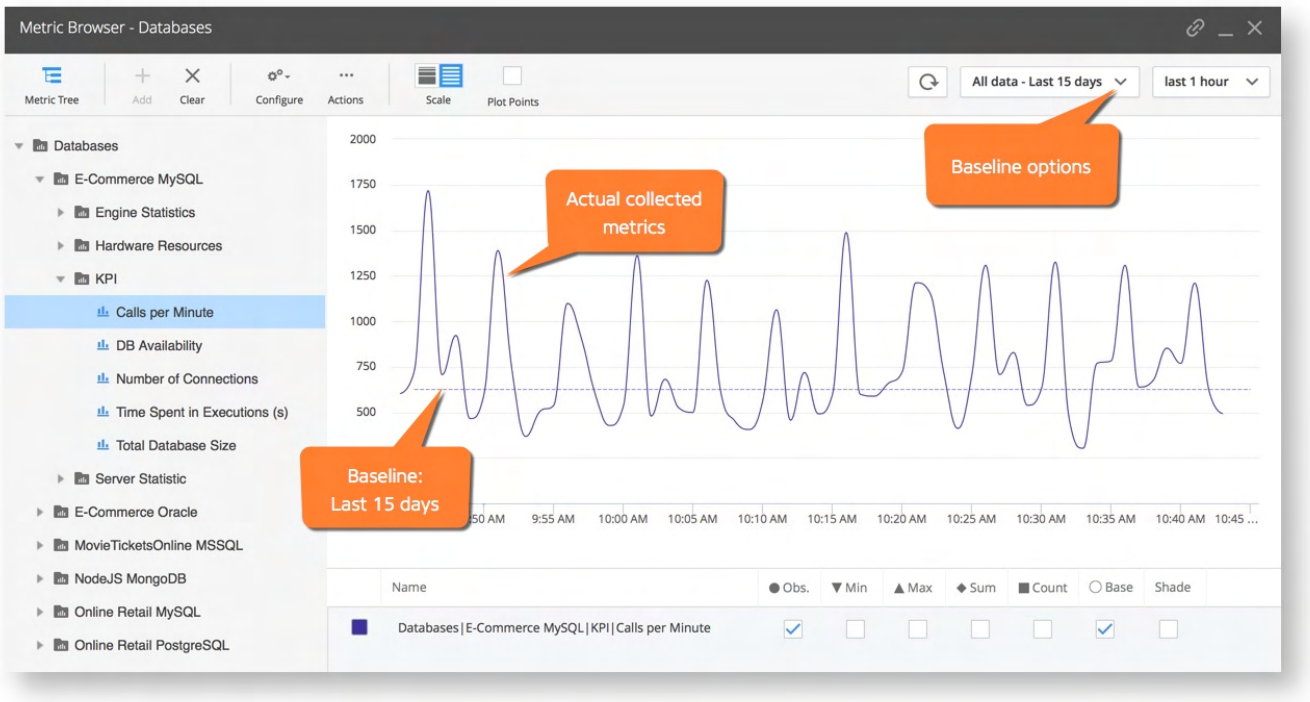
- Dynamic Baselines
- Health Rules

AppDynamics Database Visibility automatically learns to detect performance anomalies using baselines that are specific to your database and database server environments.

AppDynamics creates baselines by collecting metrics from your monitored databases and servers over defined periods of time. This establishes what is normal for your environment. You can also create your own baselines.

View Performance Metrics on the Metric Browser

On the Metric Browser for Database Visibility, you can visualize performance metrics and see how they deviate from expected behaviors established by the baseline.



Monitor Database Performance

Related pages:

- Access Database Visibility from Application Monitoring Views
- Monitor Databases and Database Servers

The subtabs for each database provide detailed information about your database. This page describes the various panels for the database instance and how to interpret the displayed information.

From most of these subtabs, you can drill down to the Queries view.

- View Overall Database and Server Performance
- Database Dashboard
- Database Activity Window
- Database Topology Window
- Database Live View Window
- Database Queries Window
- Database Procedures Window
- Database Clients Window
- Database Sessions Window
- Database Blocking Sessions Window
- Database Schemas and Databases Windows
- Database Modules Window
- Database Programs Window
- Database Containers Window
- Database Users Window
- Database Buckets Window
- Database Business Transactions Window
- Database Applications Window
- Database Object Browser Window
- Database Custom Metrics Window

Important information

Garbage collection affects all monitored entities, including database queries, sessions, clients, and schemas. Entities that are garbage collected will be displayed as unavailable.

View Overall Database and Server Performance

Related pages:

- Access Database Visibility from Application Monitoring Views

Database Visibility provides overall views of your databases that show key performance indicators.

Access the Databases Window

From AppDynamics Home, click **Databases**.

Features of the Databases Window

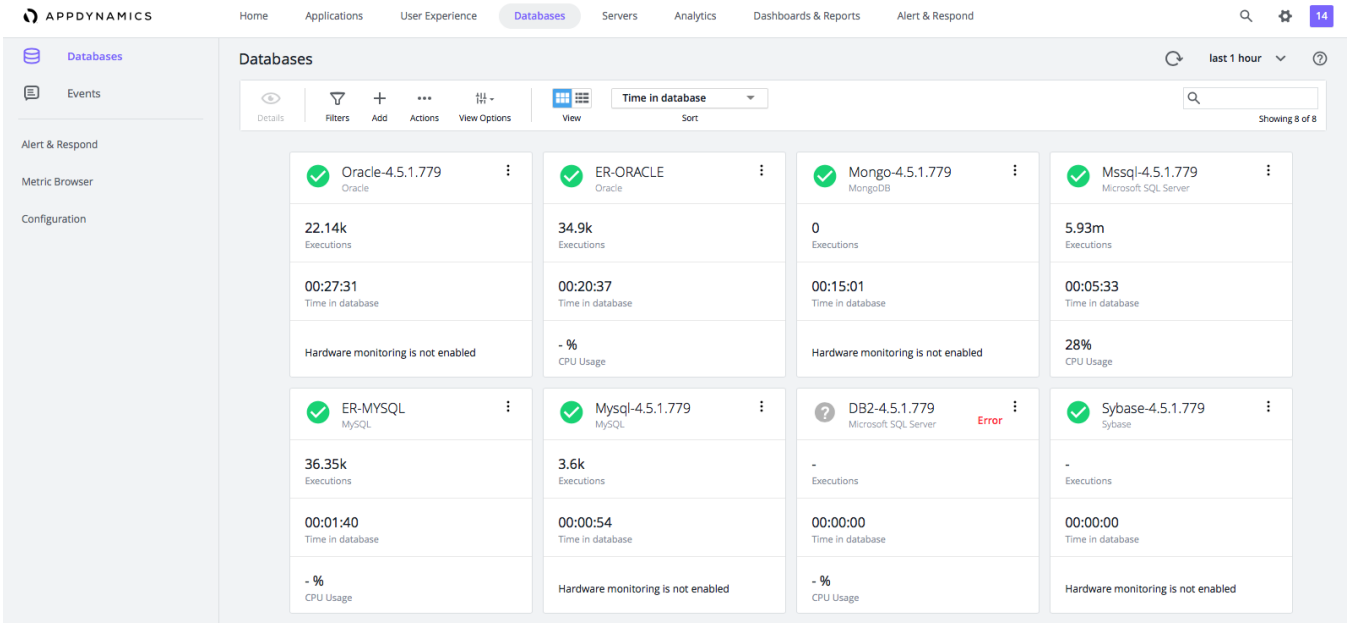
On the views of the Databases window you can:

- Show only databases meeting certain criteria by entering search criteria in the search box on the top right of the page.
- Click **Filters** to show only databases meeting certain search criteria, such as criteria that describes Health, Load, Time in database or Type.
- Click **+** to add a new database Collector.
- Click **Actions** to export the data on this window in a .csv formatted file that is automatically downloaded to your specified downloads directory.
- Click **View** to switch between card and list view.
- See the overall performance of your databases and if there are any critical issues with them. The server health or status indicates whether criteria has been met for normal, warning or critical health rules. The green checkmark indicates that the database is operating within normal conditions, the red exclamation mark indicates that at least one critical health rule has been violated.
- Click the name of the database to switch to the Database Dashboard for that specific database.

Databases Card View

From the Databases card view, you can view:

- The configured collectors along with the database type and the number of nodes used in the database cluster.
- The **Total Executions** graph that provides insight into the total number of calls (transactions for PostgreSQL databases) monitored for the database over the selected time period.
- The **Time in Database** graph that indicates the total time spent executing those calls during the selected time period. This metric is expressed in hours, minutes, and seconds.
- **CPU Usage** metrics that when enabled shows the percentage of CPU capacity consumed by the database.



Databases List View

From the Databases list view, you can:

- View the number of database nodes under the **Nodes** column.
- Click **View Options** to turn the spark charts on and off.
- Get to the databases you want to see quickly if you are monitoring many databases.
- Click on any point in the spark charts to see a metric for that point in time. For example, clicking on a point in the calls Trend shows how many calls were monitored at that time.
- Click on a column name to sort the list on that key.

Database Dashboard

Related pages:

- Access Database Visibility from Application Monitoring Views
- View Overall Database and Server Performance
- Discover Normal Database and Server Activity
- Database Health Rules and Alerts
- Monitor Database Performance

The dashboard for each individual database provides detailed information about your database.

Permissions

To view database collectors, users need a role with the **Can View All Collectors** permission.

Access the Database Dashboard

You can access the Database Dashboard in one of these ways:

- In AppDynamics Home, on the **Databases** card, click the name of the database for which you want to see the Database Dashboard.
- In the Databases overview panel, click the name of the database for which you want to see the Database Dashboard.

Features of the Database Dashboard

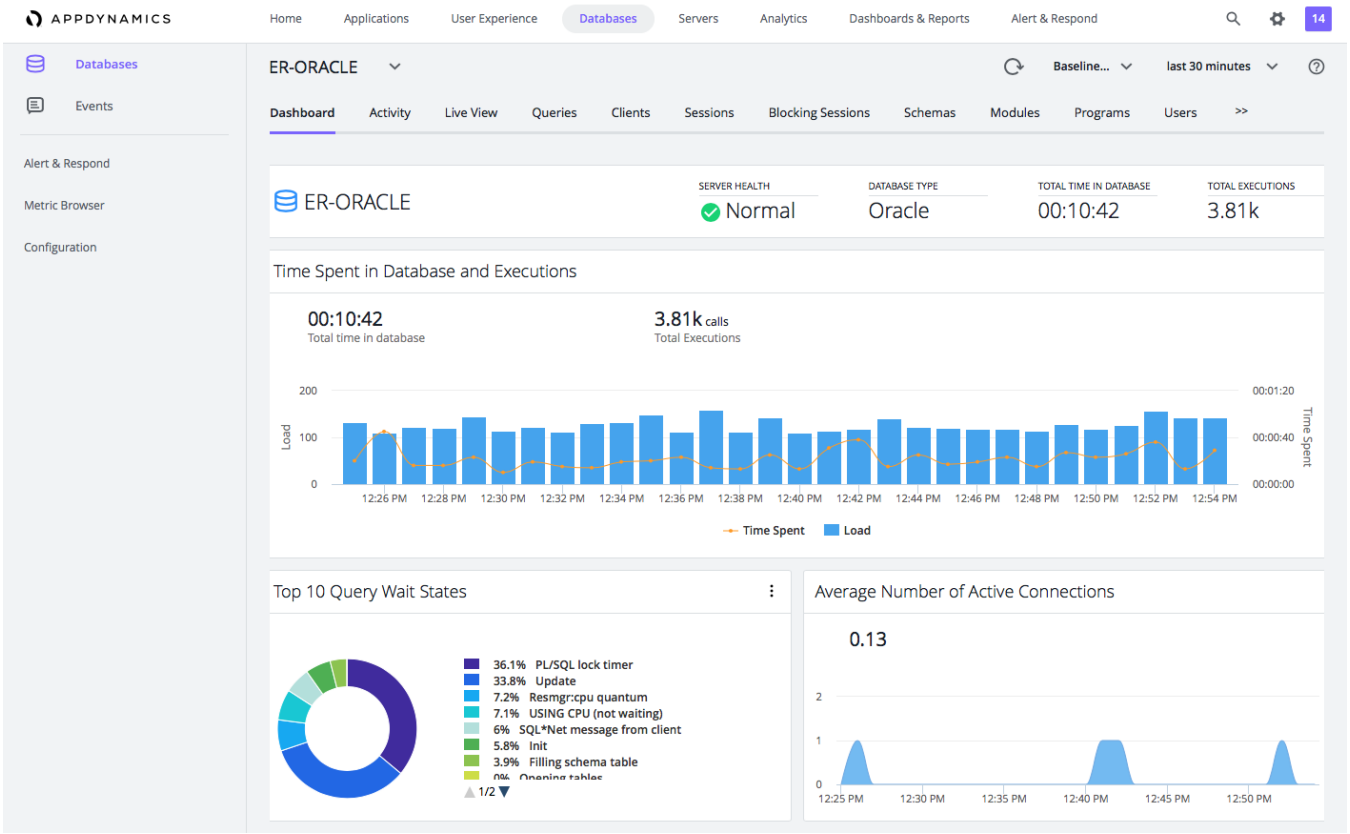
On the Database Dashboard you can:

- Click the dropdown arrow next to the database Collector name at the top of the page to choose a database that you want to see the dashboard for.
- (From 21.5 onwards) For a clustered database, click the down-arrow below the collector name to navigate between different nodes of the cluster. By default, **All Nodes** is selected.
- Click the down-arrow next to the clustered database Collector name to navigate between different nodes of the cluster.
- Click the status icon below **SERVERS** in the top section of the dashboard to go directly to **Events** where you can see any recent events.

On the Database Dashboard, you can view:

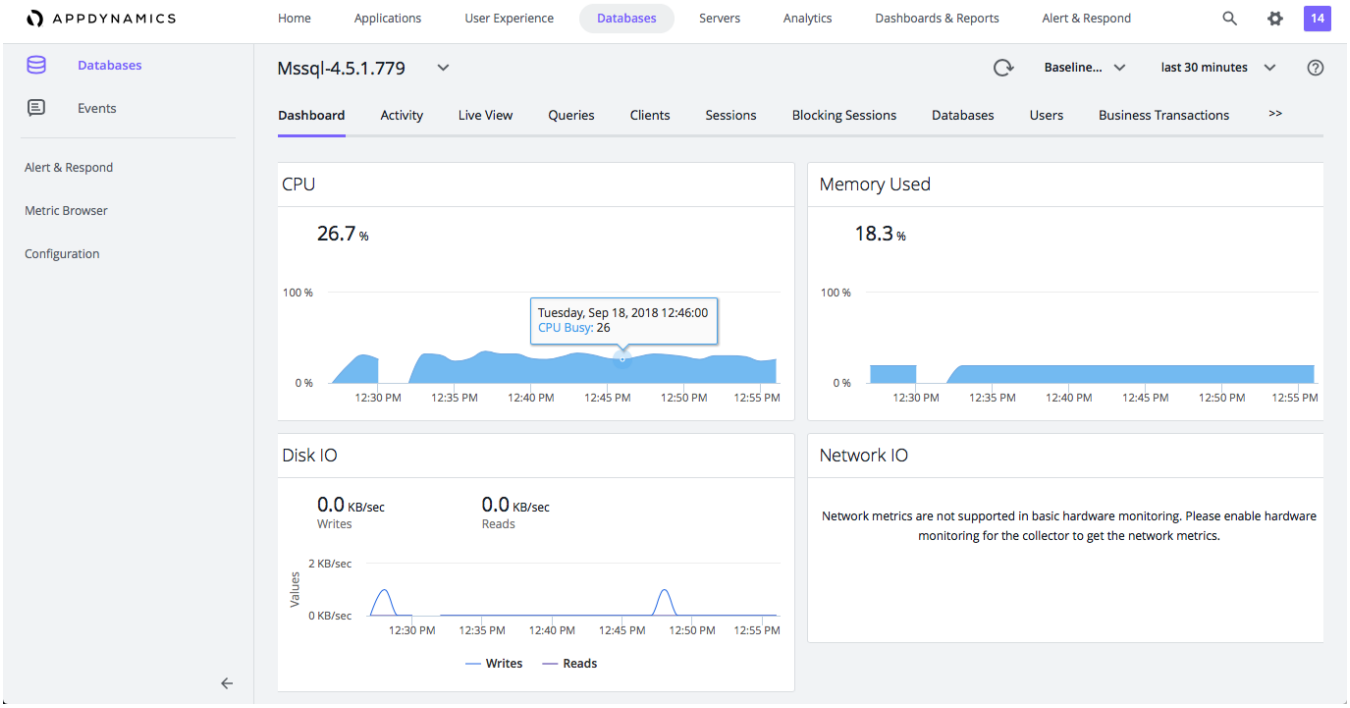
- **Server Health**: Server Health at the top of this panel indicates the extent to which health rules are being violated:
  - Green - Healthy server
  - Yellow/orange - server with warning-level violations
  - Red - server with critical-level violations
- **Type**: The database type.
- **Time Spent in Database and Executions**:
  - Load - At a glance, you can see the total number of calls (transactions for PostgreSQL databases) during the specified time period and the number of calls for any point in time.
  - Time spent in Database - The total time spent executing SQL statements during the specified time period.
  - Changes - Changes made to the database configuration parameters.
  - Max CPU - It displays the CPU cores present in the database server and is disabled by default. The CPU core count is collected using SQL (for Oracle, DB2, and MSSQL) and Hardware monitoring (if enabled for MySQL, PostgreSQL and Sybase).
- **Top 10 SQL Wait States (not available for Couchbase)**: Activities that contribute to the time it takes the database to service the request. The wait states consuming the most time may point to performance bottlenecks. For example, a db file sequential read wait state may be caused by segment header contention on indexes or by disk contention. See your database platform documentation for descriptions of the SQL wait states. For example:
  - Wait State - CHECKPOINT_QUEUE
  - Description - Used by background worker that waits on events on queue to process checkpoint requests.
  - Recommended Action - You should be able to safely ignore this action because it indicates that the checkpoint background worker is waiting for work to do. If you thought you had issues with checkpoints not working or log truncation, you might see if this worker ever "wakes up". Expect higher wait times as this will only wake up when there is work to do.
- **Top 10 Phases (Couchbase only)**: The phases that have processed the most documents.
- **Average number of active connections**: The average number of sessions that are actively running a query during the selected time period.

> **Accessing the Comparison Report**
>
> Click on any data point to view the time comparison report, which shows query run times and wait states 15 minutes before, and 15 minutes after the selected time.



The **CPU**, **Memory**, **Disk I/O**, and **Network I/O** graphs display when the Database Agent has been configured to also monitor the database host hardware. See, "Configure the Database Agent to Monitor Server Hardware" on Add Database Collectors.

- **CPU**: The CPU graph shows the relative percentages of CPU processing time used for handling system and users processes.
- **Memory**: The Memory graph shows the percentage of total memory in use at any point in time.
- **Disk I/O**: The Disk I/O graph shows disk usage, the volume of data read and written.
- **Network I/O**: The Network I/O graph shows network activity, the volume of data sent and received.



Accessing the Metric Browser

To see more information about a specific metric, double click any point on the graph and the Metric Browser opens displaying that metric. You can then hover over a point on the graph in the Metric Browser for more information about the metric. This feature is available for all the graphs except the SQL wait states graph.

Database Activity Window

Related pages:

- Access Database Visibility from Application Monitoring Views
- Database Visibility
- Monitor Databases and Database Servers
- Monitor Database Performance

The following describes the reports available in Database Visibility on the Database Activity Window. The reports available depend on the database platform being monitored. The reports are listed in alphabetical order and indicate which database platform supports them.

Access the Database Activity Window

To access the Database Activity window

1. To view a database's activity, click the name of the database.
2. Click the **Activity** tab.

Features of the Database Activity Windows

To view the database reports of different databases, click the down arrow next to the database name at the top of the page. From the list, select the database you want to view or search for the database in the search bar. Click the refresh icon to show only databases that meet that search criterion.

The activity reports available are described below. The reports available are database-dependent. The reports are listed in alphabetical order and indicate which database platform supports them. The reports will only show 10 days of data, which is the Events Service limit.

Wait State

Description: This report displays time-series data on Wait Events (states) within the database. Each distinct wait is color-coded, and the Y-axis displays time in seconds. This report also displays data in a table and highlights the time spent in each wait state for each SQL statement.

Relevance: The wait states consuming the most time may point to performance bottlenecks. For example, db file sequential reads may be caused by segment header contention on indexes or by disk contention.

Platform: IBM DB2, Microsoft SQL Server, MySQL, Oracle, PostgreSQL, Sybase, and Sybase IQ

Max CPU - It displays the CPU cores present in the database server and is disabled by default. The CPU core count is collected using SQL (for Oracle, DB2, and MSSQL) and Hardware monitoring (if enabled for MySQL, PostgreSQL and Sybase).

Phase

Description: This report displays the number of documents processed during a phase of query execution.

Relevance: The phase processing the most documents may point to performance bottlenecks.

Platform: Couchbase

Top Activity

Description: Display the top time in database SQL statements in a time-series view. This report also displays data in a table and highlights the time spent in the database for each of 10 top SQL statements. Only queries reported by newer agents are grouped; the queries reported by older agents may not appear upon grouping.

Relevance: Use this report to see which SQL statements are using the most database time. This helps to determine the impact of specific SQL statements on overall system performance allowing you to focus your tuning efforts on the statements that have the most impact on database performance.

Platform: IBM DB2, Microsoft SQL Server, MySQL, Oracle, PostgreSQL, Sybase, Sybase IQ, and Couchbase

Time Comparison

Description: This report allows compares the performance of two databases during the same time period based on a specific statistic type. You can click Group Similar to group together queries with the same number of parameters. Only queries reported by newer agents are grouped; the queries reported by older agents may not appear upon grouping.

Relevance: You may want to compare the performance of your development and production databases before and after you tune the SQL queries or add an index or join. This report can help you determine the effectiveness of any performance tuning procedures you have implemented.

Platform: IBM DB2, Microsoft SQL Server, MySQL, Oracle, PostgreSQL, Sybase, and Sybase IQ

I/O

Description: This report gives information on physical I/O performed by the database instance.

The SQL Server and PostgreSQL I/O reports include the following metrics:

- Time spent in the database
- Query throughput
- CPU usage (if hardware monitoring is enabled)
- Read and Write I/O. This metric is only available for SQL Server.
- Per file statistics (if the agent is run with the `dbagent.mssql.datafile.statistics` property). This metric is only available for SQL Server.
- Per database statistics. This metric is only available for PostgreSQL.

Relevance: Your physical disk I/O may be affecting database performance.
Poor response times may mean one of the following:

- You're doing too much physical I/O and need to adjust your I/O capacity
- You're scanning tables or indexes when you should be doing seeks
- Your database tables are missing indexes
- Your SQL needs to be tuned

Platform: Microsoft SQL Server, Oracle, PostgreSQL

Top Query

Description: This report displays the top 10 SQL statements for the specified statistic in a time-series view.

You can filter queries by command type using the **Command Type** dropdown menu and by statistic type using the **Statistic Type** dropdown menu.

With the Controller version 21.2.0 and later, some of the unused statistic types under the **Statistic Type** dropdown menu are removed from the list. You can filter queries using the most common statistic types.

Relevance: Use this report to see which SQL statements are using the most database resources. This helps to determine the impact of specific SQL statements on overall system performance allowing you to focus your tuning efforts on the statements that have the most impact on database performance. You can choose one of many statistics to base the report on.

Platform: IBM DB2, Microsoft SQL Server, Oracle

Query Wait State

Description: This report displays the wait times for all queries.

Relevance: Use this report to see how much time queries are spending in different wait states.

Platform: IBM DB2, Microsoft SQL Server, MongoDB, MySQL, Oracle, PostgreSQL, Sybase, and Sybase IQ

Parameter Changes

Description: This report displays changes to the database configuration parameters. You can select a parameter change and click one of the following options:

- **View Event Details**: Displays the parameter change event details, such as comments associated with the parameter change.
- **View Comparison Report**: Compares the query run times and wait states 15 minutes before and 15 minutes after the database parameter was changed. The comparison report is not available for clusters.

Relevance: Use this report to keep track of the changes you make to the database configuration parameters.

Platform: IBM DB2, Microsoft SQL Server, Microsoft Azure, MySQL, Oracle, PostgreSQL, Sybase ASE, and Sybase IQ

BT Activity

Description: This report displays the top (Java or PHP) business transactions that make calls to the database.

Relevance: Use this report to see which business transactions are affected by the database.

Platform: IBM DB2, Microsoft SQL Server, Microsoft Azure, MySQL, Oracle, PostgreSQL, Sybase ASE, and Sybase IQ

Database Topology Window

The Topology view displays current activity for the routing service and shards of a MongoDB database, Oracle RAC server or a custom cluster. The Topology view shows key performance metrics for the routing service and shards or nodes of the database.

Access the Database Topology Window

To access the Database Topology window:

1. To view a database's topology, click the name of the database.
2. Click the Topology tab.

Database Topology Window Features

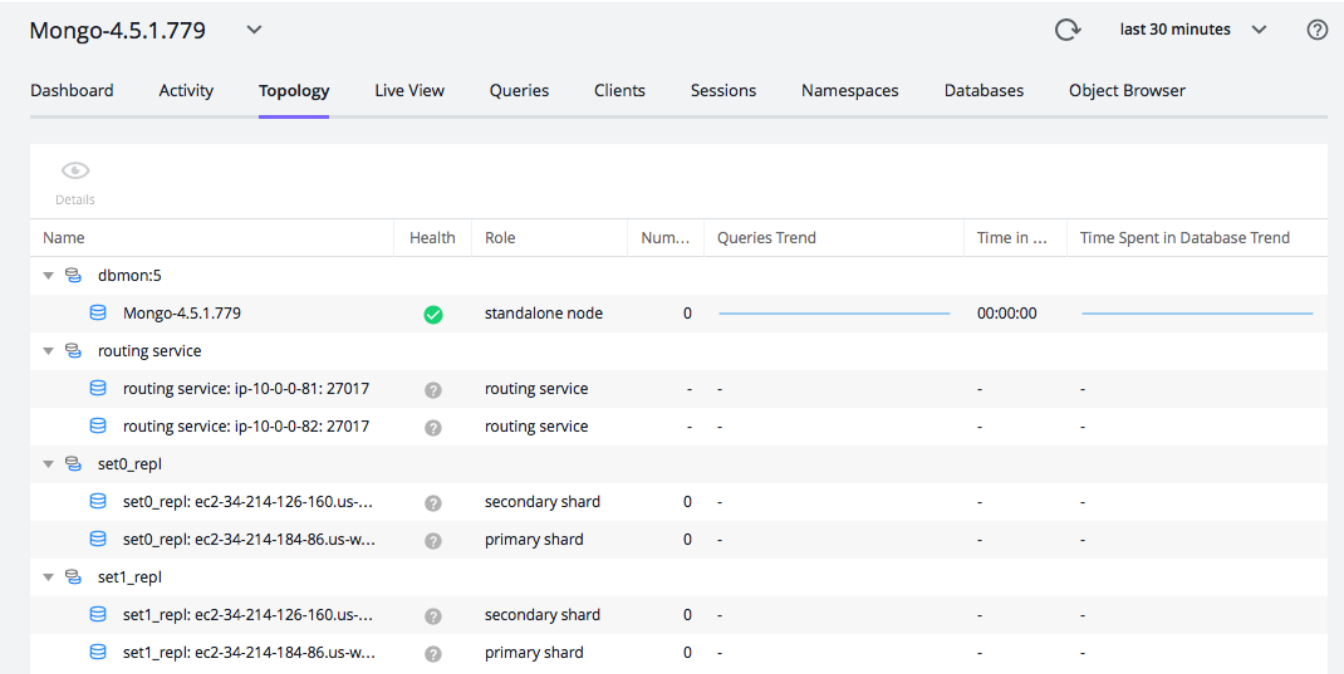
From the Database Topology window, you can:

- Select the **Auto Refresh** box. From the **Every** list, you can choose how often the system updates the live view.
- Click the dropdown arrow next to the database Collector name at the top of the page to choose a node or cluster that you want to see the metric views for. You can either select the database Collector name from the list or search for the database Collector by entering text in the search bar and then clicking the refresh icon to show only database Collectors that meet those search criteria.
- Hover over the sections of the trends to see their details.
- Click a column title on the **Topology** name list to sort the topologies using that column as the sort key.

From the Topology window, you can view:

- **Health**: Indicates whether any health rules for the routing service or shards of the replica set or nodes have been violated.
- **Role**: The role performed by the shard. The routing service routes the queries to the shards to balance the load. The members or shards of the replica sets can have either primary or secondary shard role.
- **Num Queries**: The number of queries handled.
- **Queries Trend**: The trend of queries handled over time.
- **Time in Database**: The time required for the database to respond to the queries.
- **Time Spent in Database Trend**: The trend of time spent in the database over time.

> If you change the port number for a monitored database node, the Topology window shows a new node with the new port number, as well as the original node with the old port number. The original node remains in the Topology window for the duration of the retention period that you specify in Controller Settings.



Database Live View Window

Related pages:

- Access Database Visibility from Application Monitoring Views
- Database Visibility
- Monitor Databases and Database Servers
- Monitor Database Performance

The Live View displays current activity for the database server. The live view shows key performance metrics for the last time period, which can be refreshed in a specified time interval between 10 seconds and 5 minutes.

Access the Database Live Window

To access the Database Live window:

1. To view a database's current performance, click the name of the database.
2. Click the **Live View t**ab.
3. For clustered databases, select the node to view its details.

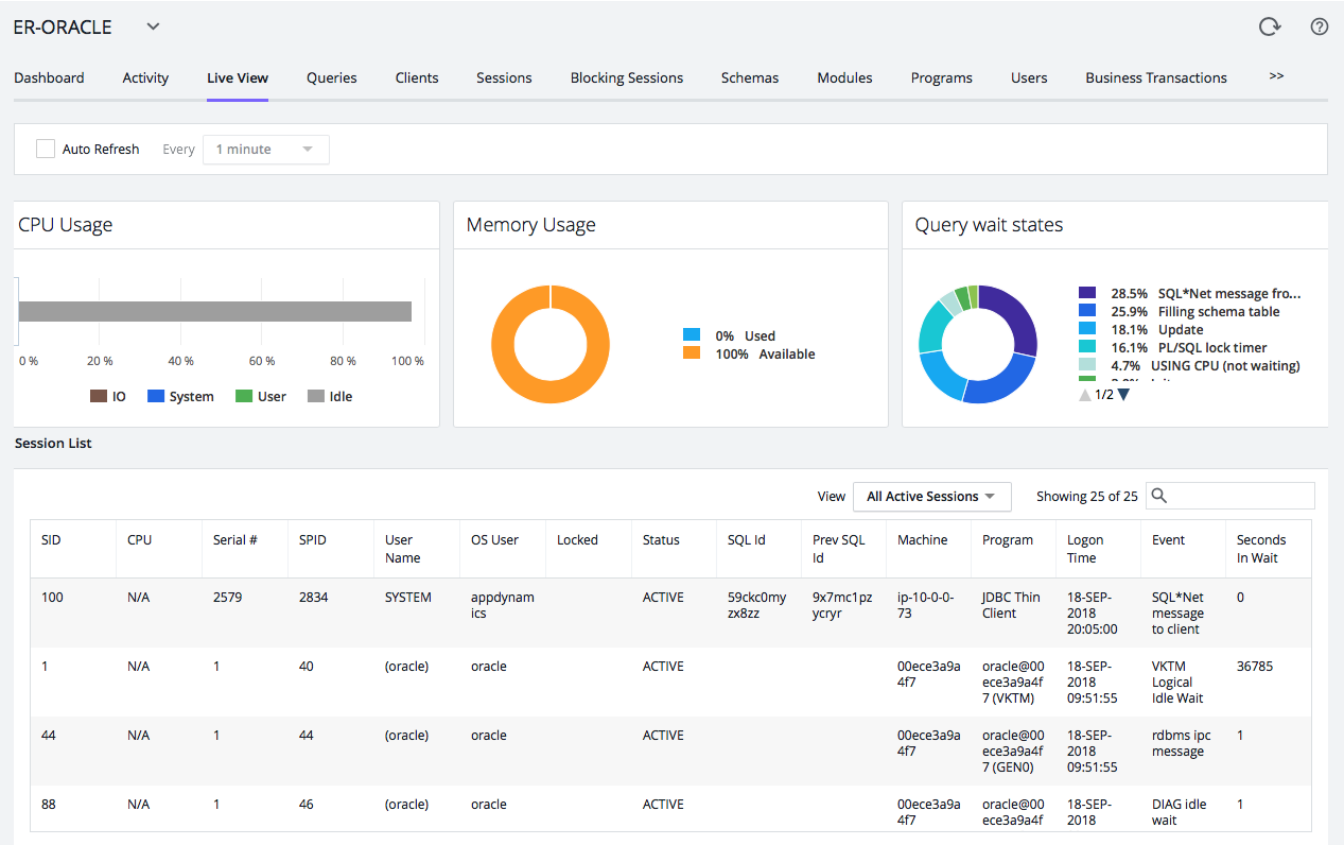
Database Live Window Features

From the Database Live window, you can:

- Select the **Auto Refresh** box and then from the **Every** list, choose how often you want the system to update the live view.
- Click the down arrow next to the database collector name at the top of the page to choose to view the live view of a different database Collector by either selecting the database Collector name from the list or by searching for the database collector by entering text in the search bar and then clicking the refresh icon to show only database collectors that meet that search criteria.
- Hover over the sections of the charts to see their details.
- Click the **View** list to view **All Active Sessions** or **All Sessions**.
- Click a column title on the **Session List** to sort the Session List using the clicked column as the sort key.

From the Database Live window, you can view:

- **CPU Usage**: If you have enabled Hardware Monitoring for the server the chart shows the percentage of available CPU resources consumed by users and the system and a break down of how the CPU is being used.
    - IO - Disk input and output.
    - System - Database operating system activities.
    - User - User interaction with the database.
- **Memory Usage**: Displays how much memory is in use and how much is available.
- **Query wait states**: Activities that contribute to the time it takes the database to service the request. The wait states consuming the most time may point to performance bottlenecks. For example, database file sequential reads may be caused by segment header contention on indexes or by disk contention.
- **Session List**: A table describing database usage instances and their properties. The database session is the application interaction with the database, a new session is started for each request. The columns displayed are database dependent.
- **Blocking Tree**: A tree showing all the sessions causing blocks. Nested under each blocking session are the sessions being blocked and the corresponding queries and blocked objects.



For SQL Server, the **CPU Usage** and **Memory Usage** statistics are displayed in Live View even when hardware monitoring is disabled.

In the Live View for database clusters, you must select a node from the dropdown menu to view that node's activity.

Database Queries Window

Related pages:

- Access Database Visibility from Application Monitoring Views
- Add Database Collectors
- Database Visibility
- Monitor Databases and Database Servers
- Monitor Database Performance

The Queries window displays the SQL statements and stored procedures that consume the most time in the database. You can compare the query weights to other metrics such as SQL wait times to determine SQL that requires tuning.

Access the Database Queries Window

To access the Database Queries window:

1. To view database queries, click the name of the database.
2. Click the Queries tab.

Database Queries Window Features

From the Database Queries window, you can:

- View the top N queries. These are the queries that consumed the greatest amount of database time to complete.
- Choose to display the top 5, 10, 100 or 200 queries.
- Click the name of a column to sort the **Query** list using that key.
- Click **Filter by Wait States** to choose wait states to filter the **Query** list by. The filtered list only displays queries that caused the selected wait states.
- Check the **Group Similar** box to group together queries with the same syntax. Queries that share the same syntax but different 'IN' clause parameters are still grouped together. Query grouping is case-insensitive. Double-click a query group to dive into the Group Details for more information about that query group. Only queries reported by newer agents are grouped; the queries reported by older agents may not appear upon grouping.
- Search for a specific query, by entering text in the search bar that may appear in the Query. This is useful if you found a slow query in the Slow Database calls window of AppDynamics Application monitoring.
- Double-click a query or select a query and click **View Query Details** to dive into the **Query Details** for more detailed information about that specific query. If you see a message saying that query details are unavailable, you may need to update your permissions or extend your query retention period.
- Click the down arrow next to the database Collector name at the top of the page to choose to view the database queries of a different database Collector by either selecting the database Collector from the list or by searching for the database Collector by entering text in the search bar and then clicking the refresh icon to show only database Collectors that meet that search criteria.
- Click **Actions** to:
    - Export the data on this window in a .csv formatted file that is automatically downloaded to your specified downloads directory.
    - Rename the selected query so that it is easier to identify. Note that the query name is associated with the query itself. Thus, when a query's retention period expires, both the query and the query name disappear.

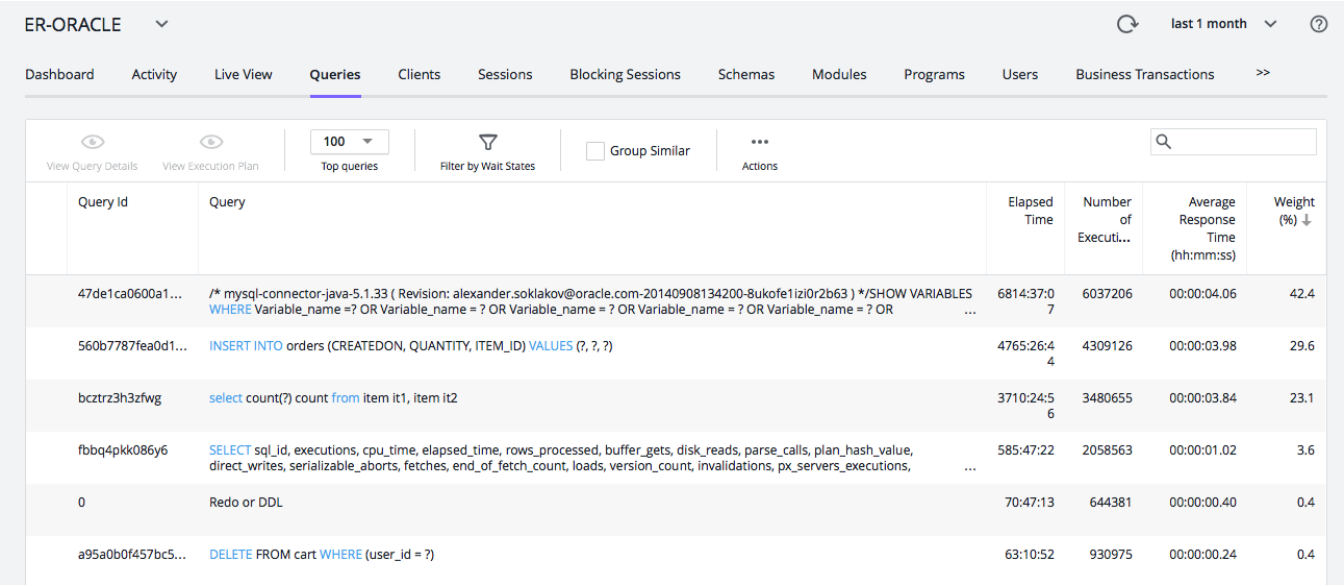
From the Database Queries window, you can view:

- **Query Id**: This is the unique ID assigned to the query internally by the database.
- **Query**: The text of the query, custom query name, or, in the case of a stored procedure invocation, the name of the stored procedure. If the user does not have the ANY permission to view the stored procedure name, the definition of the stored procedure is shown instead of the stored procedure name.

For SQL Server, Azure, Oracle, PostgreSQL and DB2 the metrics Number of Executions, Average Response TIme and Elapsed Time are inferred from the query index. When you are viewing the queries with Group Similar checked, the statistics reflect only the queries that have captured query statistics. Queries that do not have query statistics are excluded from the statistic values.

For databases such as MySQL and MongoDB which do not have a query index, no data will appear in the Number of Executions and Average Response Time columns. However, Elapsed Time can be inferred for MySQL and MongoDB databases using the wait state index. Elapsed Time can also be inferred using the wait state index when query statistics for a query run on an SQL Server, Azure, Oracle, PostgreSQL or DB2 database are otherwise not available. When query statistics are not available for a query run on an SQL Server, Azure, Oracle, PostgreSQL or DB2 database, the Number of Executions and Average Response Time statistics cannot be reported and are marked with a hyphen (-). This means that the database did not provide statistics for this query in the selected time range.

- **Elapsed Time**: The total time consumed by all executions of this query. Query statistics such as Number of Executions, Average Response Time and Elapsed time are available for databases supporting the query index, such as SQL Server, Azure, Oracle, PostgreSQL and DB2. For databases where the query index is not available, the elapsed time is inferred from the wait state index.
  By default, the query sampling frequency is once every second. If a query takes less than one second and happens to occur in the time between sampling instances, its elapsed time is reported as 0 seconds.
  You can configure the sampling frequency by adding the `-Ddbagent.sampling.interval= <value>` property for Database Agent. For information about adding the system properties, see Database Agent Configuration Properties.
- **Number of Executions**: (supported for SQL Server, Azure, Oracle, DB2, PostgreSQL) The number of times the query ran during the specified time period. If you are monitoring Greenplum, the number of executions appears as 0.
- **Average Response Time (hh:mm:ss)**: (supported for SQL Server, Azure, Oracle, DB2, PostgreSQL) The average time required during the specified time period to respond to the query.
- **Weight (%)**: The percentage of the total time consumed by the query.



Once you have identified the statements that are consuming the most resources, you can dig down deeper for details that can help you tune the SQL statement. For in-depth query details, click a SQL statement and then click **View Query Details**. See Database Query Details Window.

Database Query Details Window

Related pages:

- Access Database Visibility from Application Monitoring Views
- Add Database Collectors
- Database Visibility
- Monitor Databases and Database Servers
- Monitor Database Performance

Once you have identified the statements on the Database Queries window that are spending the most amount of time in the database, you can dig down deeper for details that can help you tune those SQL statements. The database instance Query Details window displays details about the query selected on the Database Queries window.

Access the Database Query Details Window

To access the Database Query Details window:

1. To view a database's query details, click the name of the database.
2. Click the Queries tab.
3. Select a query and click **View Query Details**.

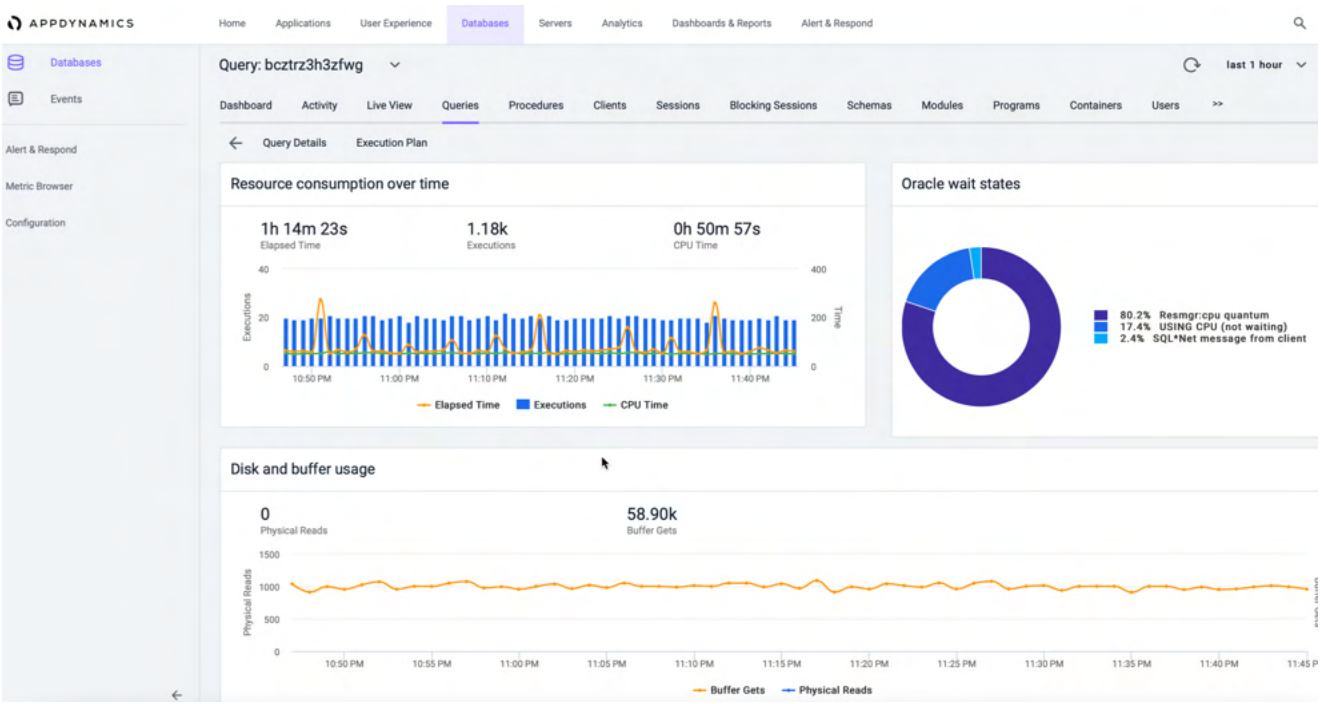
Database Query Details Window Features

From the Database Query Details window, you can:

- Click **Execution Plan** to view an explanation plan for the query you selected on the Queries window if it was a SELECT query.
- Click on the graphs to see metrics for that specific point in time.
- Click the down arrow next to the database Collector name at the top of the page to choose to view the query details of a different database Collector.
- (From 21.5 onwards) Click the back arrow () icon to return to the top queries view in the **Database Queries** window.

From the **Database Query Details** window, you can view:

- **Resource consumption over time**: For SQL Server, Azure, Oracle, and DB2, shows the amount of time the query spent in the database using resources, the number of executions, and the amount of CPU time consumed. Query statistics are only available for the aforementioned database platforms.
  The resource consumption card is also available for query groups. If all queries in the group have query metrics (for example, elapsed time and number of executions), then the aggregate of the group's metrics are shown. If none of the queries in the group have query metrics, then the aggregate elapsed time of the queries in the group, taken from the wait state index, is shown. If some queries in the group have query metrics and others do not, then the aggregate value of the available query metrics is shown.
- **Business Transactions Executing Similar Queries**: Displays the Java or PHP business transactions that execute queries similar to this query.
- **Disk and buffer usage**: Displays data when hardware monitoring is enabled for this database collector. This graph shows at a glance how effectively the database buffer is used. SQL that runs frequently is best kept in the buffer as physical disk reads are much more time-consuming. You may need to optimize your buffer manager if you see that the ratio of physical reads to buffer gets is too high.
- **Clients**: Shows the machines that executed the selected SQL statement and the percentage of the total time required to execute the statement performed by each machine. The table also shows the applications, nodes, and tiers of the machines.
- **<database_type> wait states**: Activities that contribute to the time it takes the database to service the selected SQL statement. The wait states consuming the most time may point to performance bottlenecks. For example, a db file sequential read wait state may be caused by segment header contention on indexes or by disk contention. See your database platform documentation for descriptions of the SQL wait states and recommended actions.
- **Query Active in Database/Schema**: Shows the schemas that have been accessed by this SQL.
- **Query ID (Oracle) SQL Handle (SQL Server)**: A unique ID that allows the database server to more quickly locate this SQL statement in the cache.
- **Query**: The entire syntax of the selected SQL statement. You can click the pencil icon in the top right corner of the Query card to edit the query name so that it is easy to identify.
- **Queries in Group (only in Group Details view)**: The top 100 queries in the query group
- **Users**: The users that executed this query.
- **Programs/Applications**: The program/application from which this query was executed.
- **Procedures**: The list of procedures for the queries that you monitor.
- **Containers:** The containers on which the query runs.



To see how the SQL was executed internally within the database, click the Execution Plan tab. The statement execution plan is the sequence of operations the database performs to run the statement.

Oracle Explain Plan Limitations

For Oracle, AppDynamics Database Visibility displays execution plans for only SELECT, UPDATE, INSERT, and DELETE statements because Database Visibility relies on the Oracle EXPLAIN PLAN statement to obtain the execution plans and the Oracle EXPLAIN PLAN statement only provides the Oracle optimizer's execution plans for SELECT, UPDATE, INSERT, and DELETE statements. See also, User Permissions for Oracle.

If you have a poorly performing piece of SQL that you are trying to tune, the obvious place to start is to look at the most costly step of the execution plan.

Tuning SQL is a vast topic, but a couple of things to look out for include:

- Index or table scans: May indicate a need for better or additional indexes.
- Bookmark Lookups: Consider changing the current clustered index, using a covering index and limiting the number of columns in the SELECT statement.
- Filter: Remove any functions in the WHERE clause, don't include views in your Transact-SQL code, may need additional indexes.
- Sort: Does the data really need to be sorted? Can an index be used to avoid sorting? Can sorting be done at the client more efficiently?

Database Query Execution Plan Window

Related pages:

- Access Database Visibility from Application Monitoring Views
- Add Database Collectors
- Database Visibility
- Monitor Databases and Database Servers
- Monitor Database Performance

The Database Query Execution Plan window can help you to determine the most efficient execution plan for your queries. Once you've discovered a potentially problematic query, you can run the EXPLAIN PLAN statement to check the execution plan that the database created. A query's execution plan reveals whether the query is optimizing its use of indexes and executing efficiently. This information is useful for troubleshooting queries that are executing slowly.

Access the Database Query Execution Plan Window

To access the Database Query Execution Plan window:

1. To view a query execution plan, click the name of the database.
2. Click the Queries tab.
3. Click a **SELECT** statement to examine and click **View Query Details**.
4. Click the Execution Plan tab.

Database Query Execution Plan Window Features

From the Database Query Execution Plan window, you can:

- Click **Explain** to view the execution plan for a query.
- Click **Schema** to choose a different schema and then explain the execution plan based on that schema. If you want to view the execution plan for a query that belongs to a schema outside the scope of your database account permissions, you can enter a username and password for a different database account that has access to the schema.
- If you want to improve a suboptimal query, you can modify the query and paste it into the **Explain another query** box. Click the **Explain** button to generate the execution plan for your modified query and determine whether the modified query is more efficient.
- Click the down arrow next to the query name at the top of the page to choose to view queries and query execution plans of a different database Collector. You can either select the database Collector name from the list or search for the database Collector in the search bar. Click the refresh icon to show only database Collectors that meet your search criteria.

If you are using MSSQL, you can perform these steps to view the costly operations (the amount of work such as CPU usage, I/O, and memory required to complete the operation) in the execution plan along with the warnings (if any) in the plan:

1. Under **Query Plan Handle(s)**, select a cached execution plan.
2. Click **View Cached Plan.**
   You can view the following details:
   Top Plan Operations:

Field Name	Description
Operation	The object scan (table or index) operation in the execution plan
Database	The database name of the object
Schema	The schema of the object in the database
Object Name	The name of the object (table or index) that is scanned in the operation
Estimated I/O Cost	The estimated cost of the input/output (`I/O`) to perform the operation
Estimated CPU Cost	The estimated CPU cost to perform the operation
Estimated number of rows	The estimated number of rows in the object

Warnings: This table includes the warnings in the execution plan related to any issue such as Typecast. This table helps you in understanding the potential performance-degrading activities that happen during the query execution. This table includes these fields:

Issue: The issue description for the warning.

Expression: The query expression used in the execution plan that correlates to the issue.

If you are using Oracle, SQL Server or Azure you can view the execution plan in one of the following ways:

- Select a cached execution plan and click **View Cached Plan** to view the plan details.

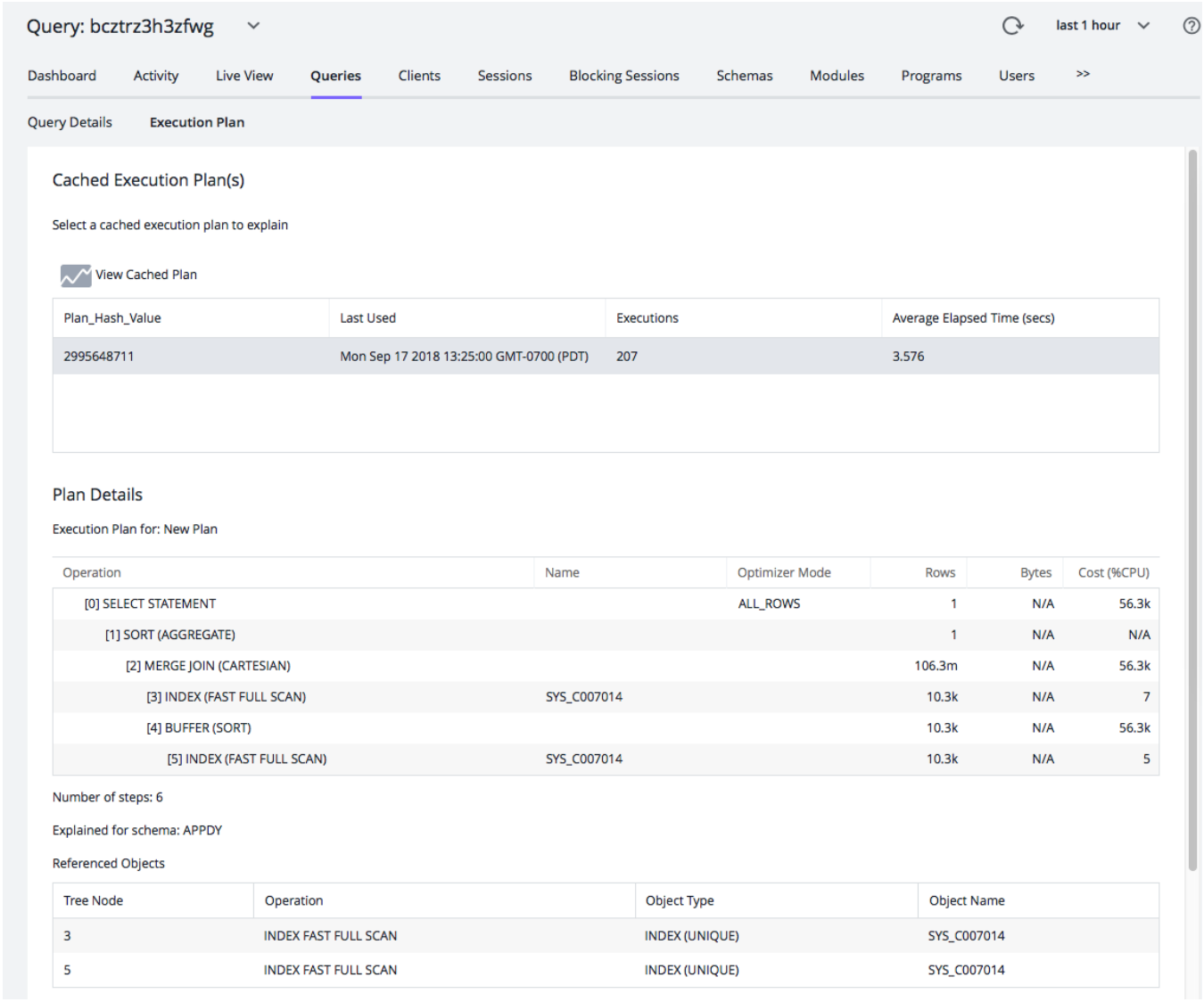
- Double-click a cached execution plan.

If you are using MySQL, you can also:

- Choose to explain the query on another schema. In **Plan Details**, from the list on the right, choose the schema name and then click **Explain**.
- Copy the text in the Parsed SQL output box and paste it into the **Explain another query** box where you can edit it and then explain the edited version of the query.

From the Database Query Execution Plan window, you can view:

- **Cached Execution Plan(s)**: identifying details of the cached execution plan, which vary from one database to another. For SQL Server, the execution plan is rendered as a diagram.
- **Plan Details**: execution plan details, showing the step by step procedure the database followed to process the selected query.
- **Referenced Objects**: the database objects accessed by the execution plan.



Database Procedures Window

The Database **Procedures** window is available for Oracle Server.

The Database **Procedures** window shows you the names of the Top N procedures on the database instance based on time. To see the Top N Queries that are executed in a procedure, click a procedure name and then click **View Top Queries**. From the **Procedures** top queries page, to see more details about a query you can double-click a query to open the Query Details window.

Access the Database Procedures Window

To access the Database Procedures window:

1. To view a database's procedure, click the name of the database.
2. Click the Procedures tab.

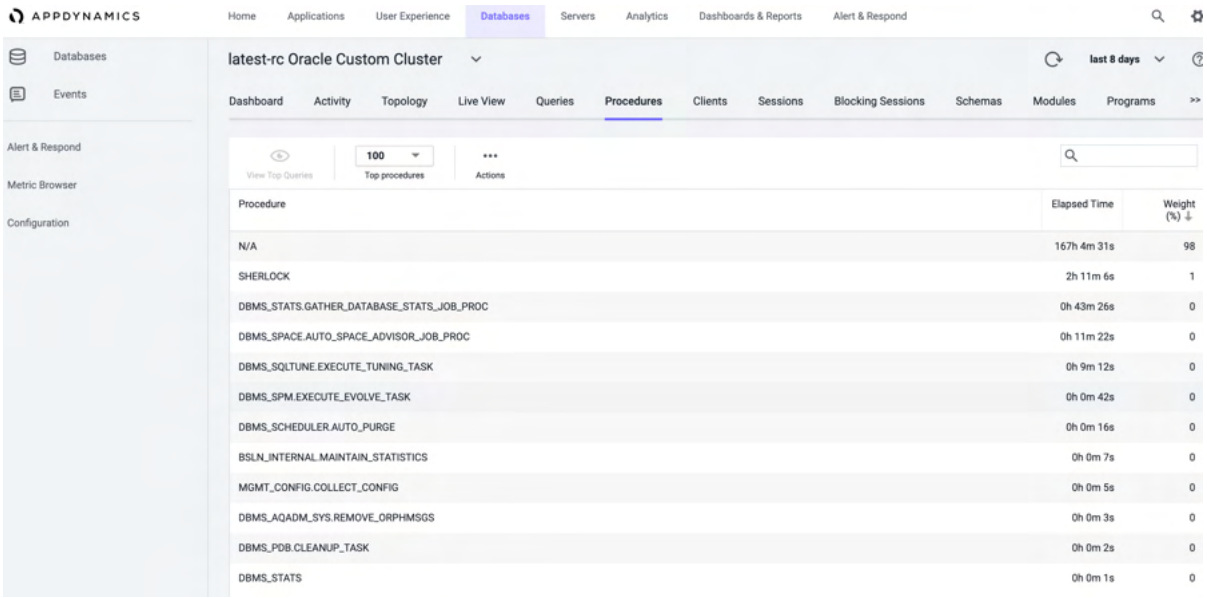
Database Procedures Window Features

From the Database Procedures window, you can:

- Choose to view information for the top 10, 50, 100 or 200 procedures that most frequently access the database server.
- Double click the name of a procedure to view the top queries for one procedure in particular, or select the procedure and then click **View Top Queries**.
- Click the down arrow next to the database collector name at the top of the page to choose to view the procedures of a different Oracle database collector by either selecting the database collector from the list or by searching for the database collector by entering text in the search bar and then clicking the refresh icon to show only database collectors that meet that search criteria.
- Click **Actions** to export the data on this window in a .csv formatted file that is automatically downloaded to your specified downloads directory.

From the Database **Procedures** window, you can view:

- **Procedure**: The name of the procedure run by an active session in the database.
- **Elapsed Time**: The time taken by the procedure.
- **Weight (%)**: The percentage of time the procedure was consuming the database resources in comparison with the usage percentage of other procedures.



Database Clients Window

Related pages:

- Access Database Visibility from Application Monitoring Views
- Database Queries Window
- Database Visibility
- Monitor Databases and Database Servers
- Monitor Database Performance

The Database Clients window shows you the hostname or IP addresses of the Top N clients using the database. A database client is any host that accesses the database instance. To see the Top N Queries run by the client, click a client name and then click **View Top Queries**. From the client top queries page, to see more details about a query you can double-click a query to open the Query Details window.

Access the Database Clients Window

To access the Database Clients window:

1. To view a database's clients, click the name of the database.
2. Click the Clients tab.

Database Clients Window Features

From the Database Clients window, you can:

- Choose to view information for the top 10, 50, 100 or 200 clients sorted by total time spent in the database.
- Choose to view information for one client in particular.
- Click the down arrow next to the database name at the top of the page to choose to view the database clients of a different database by either selecting the database from the list or by searching for the database by entering text in the search bar and then clicking the refresh icon to show only databases that meet that search criteria.
- Click **Actions** to export the data on this window in a .csv formatted file that is automatically downloaded to your specified downloads directory.

From the Database Clients window, you can view:

- **Client**: The names of the machines that have connected to the database.
- **Elapsed Time**: The total time spent by the client executing queries.
- **Weight (%)**: The percentage of time the client was connected to the database in comparison with connection times of other clients.

Database Sessions Window

Related pages:

- Access Database Visibility from Application Monitoring Views
- Database Visibility
- Monitor Databases and Database Servers
- Monitor Database Performance

The **Sessions** window shows you the Session ID of the Top N sessions using the database sorted by time spent. To see the Top N SQL run by a session, click the **Session ID** and then click **View Top Queries**. From the Client top queries page, to see more details about a query, double-click a query to open the Query Details window and from there see the Execution Plan of the query.

Access the Database Sessions Window

To access the Database Sessions window:

1. To view a database's sessions, click the name of the database.
2. Click the Sessions tab.

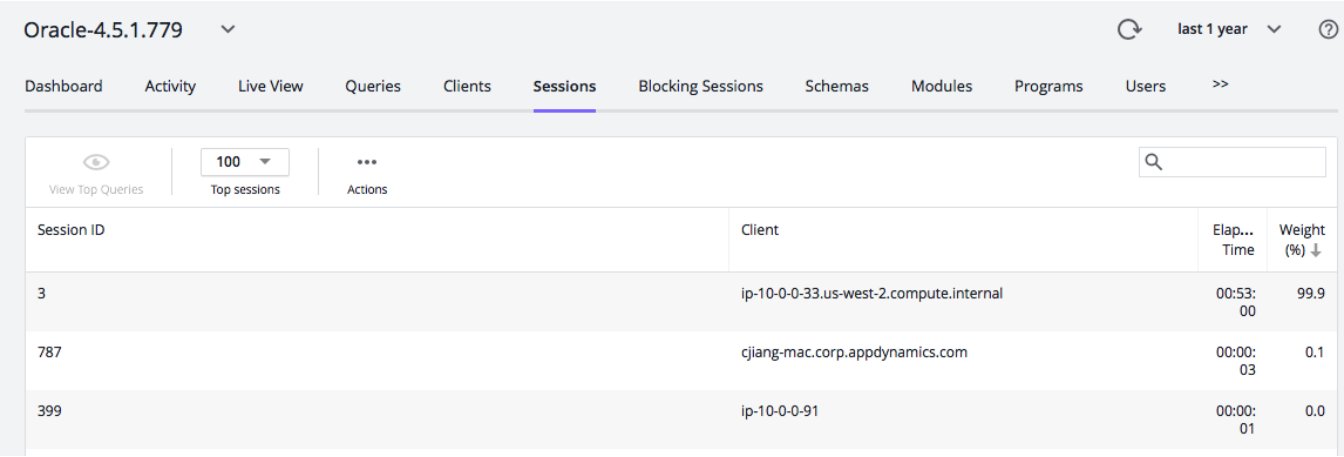
Database Sessions Window Features

From the Database Sessions window, you can:

- Choose to view information for the top 10, 50, 100 or 200 sessions consuming database resources.
- Double click a **Session ID** to view the queries run in that particular session.
- Click the down arrow next to the database name at the top of the page to choose to view the database sessions of a different database by either selecting the database from the list or by searching for the database by entering text in the search bar and then clicking the refresh icon to show only databases that meet that search criteria.
- Click **Actions** to export the data on this window in a .csv formatted file that is automatically downloaded to your specified downloads directory.

From the Database Sessions window, you can view:

- **Session ID**: the Session ID of each database instance usage.
- **Weight (%)**: the percentage of time that instance was using the database in comparison with the database usage of other sessions.



Database Blocking Sessions Window

The blocking tree displays a tree view of sessions that are blocking other sessions for Oracle, SQL Server, Sybase ASE, and DB2. You can use the blocking tree to determine the cause of deadlocks.

Access the Database Blocking Sessions Window

To access the Database Topology window:

1. To view a database's blocking sessions, click the name of the database.
2. Click the Blocking Sessions tab.

Database Blocking Tree Window Features

From the Database Blocking Tree window you can expand the tree of a blocking session to see the sessions it is blocking.

From the Database Sessions window, you can view:

- **Session ID**: the Session ID of the sessions causing the block, and the sessions being blocked
- **Client**: the names of the machines where the session is blocked
- **User**: the user of the blocked session
- **Query**: the query that is blocked or causing the block
- **Blocked Object**: the object that is blocked
- **First occurrence of block**: the time that the block first occurred in the specified time range
- **Block Duration**: the amount of time that the session was blocked or causing the block

Database Schemas and Databases Windows

The Database Schemas window is available only for Oracle database servers.

The Database Databases window is available for other database servers.

The database Schemas or Databases window shows you the names of the Top N busiest schemas or databases on the database server. To see the Top N Queries run on a specific Database or Schema, click a name and then click **View Top Queries**. From the Schema or Database top queries page, to see more details about a query you can double-click a query to open the Query Details window.

Access the Database Schemas or Databases Window

To access the Database Schemas or Database window:

1. From the AppDynamics Home page, on the **Databases** card, click the name of the database that you want to view the schemas or databases for.
2. Depending on the platform, click either the Schemas or Databases tab.

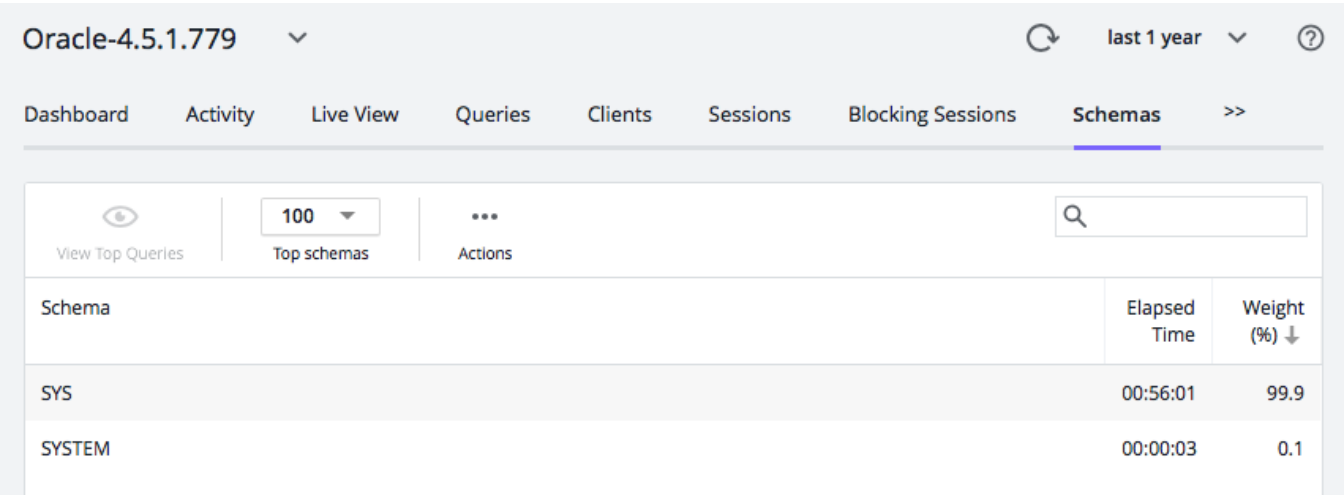
Database Schemas or Databases Window Features

From the Database Schemas and Databases window you can:

- Choose to view information for the top 10, 50, 100 or 200 schemas or databases containing the most time spent executing queries by the database server.
- Double click the name of a schema or database to view the top queries for one schema database in particular, or select the schema or database and click **View Top Queries**.
- Click the down arrow next to the Database Collector name at the top of the page to choose to view the database schemas or databases of a different database Collector by either selecting the database Collector from the list or by searching for the database Collector by entering text in the search bar and then clicking the refresh icon to show only database Collectors that meet that search criteria.
- Click **Actions** to export the data on this window in a .csv formatted file that is automatically downloaded to your specified downloads directory.

From the Database Schemas or Databases window, you can view:

- **Schema or database**: The names of the schemas or database on the database server.
- **Weight (%)**: The percentage of time the schema or database was used by the database server in comparison with the usage percentage of other schemas or databases.



Database Modules Window

The Database Modules window is available only for Oracle database servers. The **Module** field is often identical to the **Program** field unless it has been programmatically set with a call to DBMS_APPLICATION_INFO. Some packaged applications, such as Oracle E-Business Suite, automatically set the **Module** field with relevant identifiers.

The Database Modules window shows you the names of the Top N busiest modules on the Oracle database server. To see the Top N Queries run on the database by a module, click a module name and then click **View Top Queries**. From the Modules top queries page, to see more details about a query you can double-click a query to open the Query Details window.

Access the Database Modules Window

To access the Database Modules window:

1. To view a database's modules, click the name of the database.
2. Click the Modules tab.

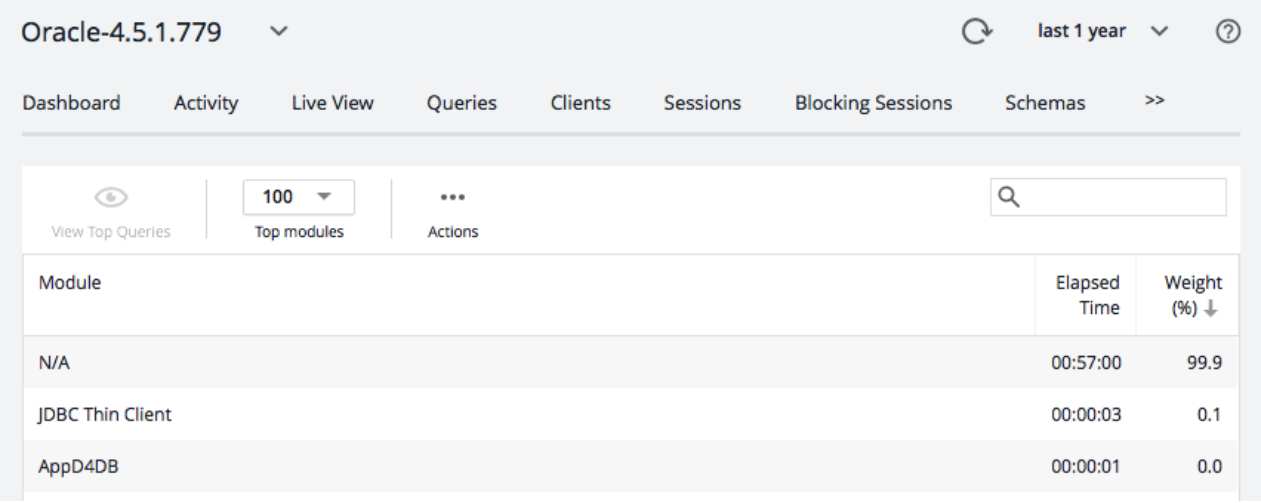
Database Modules Window Features

From the Database Modules window, you can:

- Choose to view information for the top 10, 50, 100 or 200 modules most frequently accessing the database server.
- Double click the name of a module to view the top queries for one module in particular, or select the module and click **View Top Queries**.
- Click the down arrow next to the database collector name at the top of the page to choose to view the modules of a different Oracle database collector by either selecting the database collector from the list or by searching for the database collector by entering text in the search bar and then clicking the refresh icon to show only database collectors that meet that search criteria.
- Click **Actions** to export the data on this window in a .csv formatted file that is automatically downloaded to your specified downloads directory.

From the Database Modules window, you can view:

- **Module**: The name of the module run by the active sessions in the database.
- **Weight (%)**: The percentage of time the module was consuming the database resources in comparison with the usage percentage of other modules.



Database Programs Window

The Database Programs window is available for Oracle and SQL Server.

The Database Programs window shows you the names of the Top N programs on the database instance based on time. To see the Top N Queries run on the database by a program, click a program name and then click **View Top Queries**. From the Programs top queries page, to see more details about a query you can double-click a query to open the Query Details window.

Access the Database Programs Window

To access the Database Programs window:

1. To view a database's programs, click the name of the database.
2. Click the Programs tab.

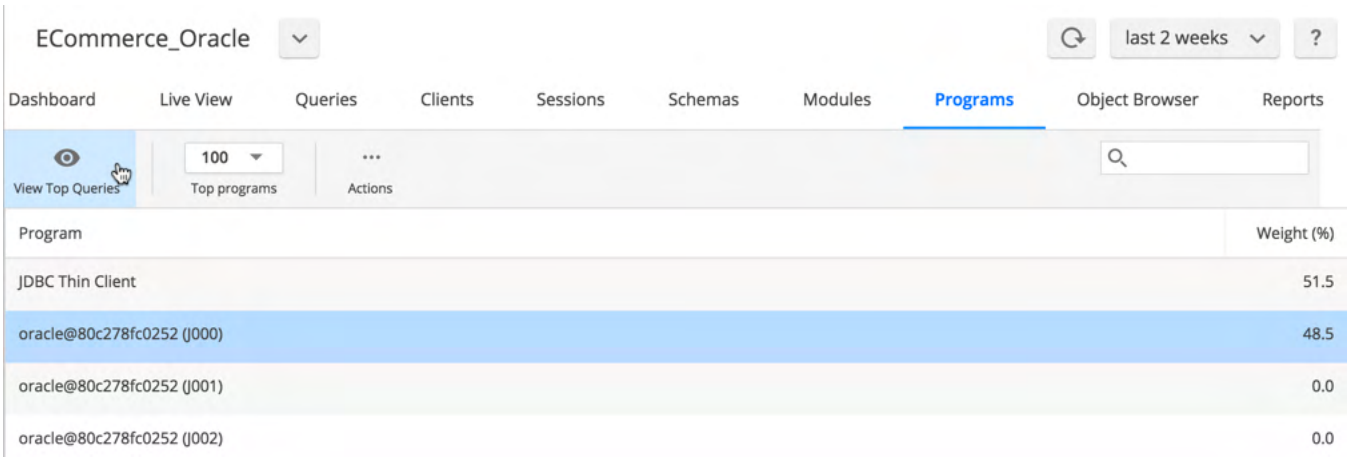
Database Programs Window Features

From the Database Programs window, you can:

- Choose to view information for the top 10, 50, 100 or 200 programs most frequently accessing the database server.
- Double click the name of a program to view the top queries for one program in particular, or select the program and click **View Top Queries**.
- Click the down arrow next to the database collector name at the top of the page to choose to view the programs of a different Oracle database collector by either selecting the database collector from the list or by searching for the database collector by entering text in the search bar and then clicking the refresh icon to show only database collectors that meet that search criteria.
- Click **Actions** to export the data on this window in a .csv formatted file that is automatically downloaded to your specified downloads directory.

From the Database Programs window, you can view:

- **Program**: The name of the program run by an active session in the database.
- **Weight (%)**: The percentage of time the program was consuming the database resources in comparison with the usage percentage of other programs.



Database Containers Window

The Database **Containers** window is available for Oracle Server.

The **Containers** window displays the names of the Top N containers on the database instance based on time. To see the top N queries run on the container, click a container name and then click **View Top Queries**. From the **Containers** top queries page, to see more details about a query you can double-click a query to open the Query Details window.

Access the Database Containers Window

To access the Database Containers window:

1. To view a database's containers, click the name of the database.
2. Click the Containers tab.

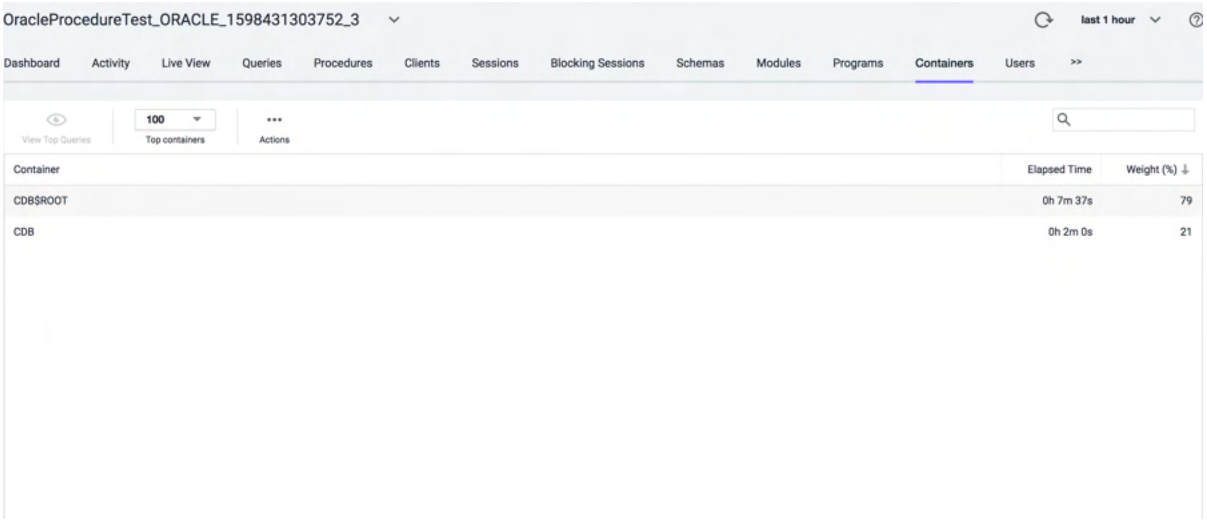
Database Containers Window Features

From the Database Containers window, you can:

- Choose to view information for the top 10, 50, 100 or 200 containers most frequently accessing the database server.
- Double click the name of a container to view the top queries for one container in particular, or select the container and click **View Top Queries**.
- Click the down arrow next to the database collector name at the top of the page to choose to view the containers of a different Oracle database collector by either selecting the database collector from the list or by searching for the database collector by entering text in the search bar and then clicking the refresh icon to show only database collectors that meet that search criteria.
- Click **Actions** to export the data on this window in a .csv formatted file that is automatically downloaded to your specified downloads directory.

From the Database Containers window, you can view:

- **Container**: The name of the container where queries run in the database.
- **Elapsed Time**: The time taken by the queries to run on the container.
- **Weight (%)**: The percentage of time the Container was consuming the database resources in comparison with the usage percentage of other Containers.



Database Users Window

The Database **Users** window shows you the users using the database. To see the Top N Queries run by a user, double-click the User name, or single-click the User name and then click **View Top Queries**. From the User top queries page, to see more details about a query you can double-click a query to open the Query Details window.

Access the Database Users Window

To access the Database Users window:

1. To view a database's users, click the name of the database.
2. Click the Users tab. You may need to click the expand button (**>>**) to see the Users tab.

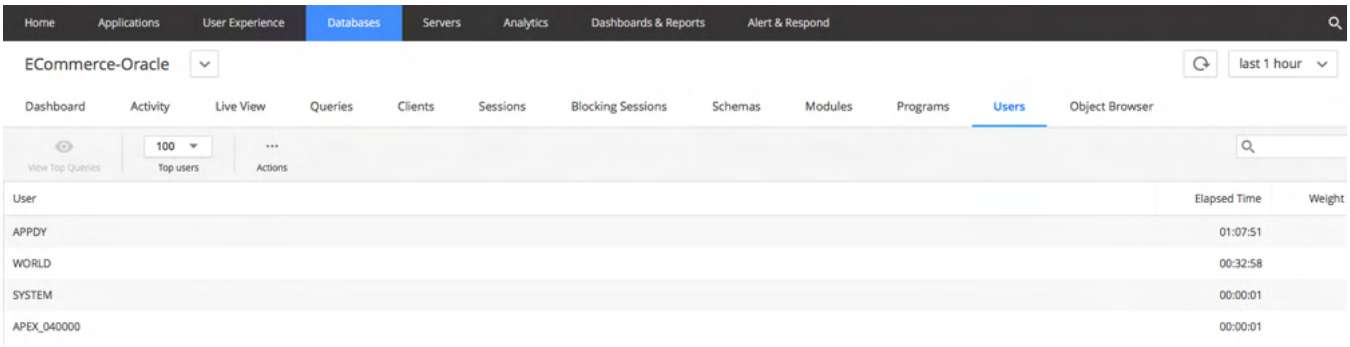
Database Users Window Features

From the Database Users window, you can:

- Use the **Top users** dropdown menu to view information for the top 10, 50, 100 or 200 users sorted by total time spent in the database.
- Click a user to view information for about that user.
- Click the down arrow next to the database name at the top of the page to choose to view the database users of a different database. You can either select the database from the list or search for the database by entering text in the search bar and clicking the refresh icon to show only databases that meet that search criterion.
- Search for the database by entering text in the search bar and then clicking the refresh icon to show only databases that meet that search criterion.
- Click **Actions** to export the data on this window in a .csv formatted file that is automatically downloaded to your specified downloads directory.

From the Database Users window, you can view:

- **User**: The names of the user accounts that have connected to the database.
- **Elapsed Time**: Duration of time that the user spent executing queries.
- **Weight (%)**: Time the user spent executing queries as a percentage of the total time spent by all users.



Database Buckets Window

The **Buckets** window is available only for Couchbase collector.

The **Buckets** window shows you the names of the top 200 buckets sorted in descending order based on operations. To see the details of a specific bucket, click a bucket name, and then click **View Bucket Details**.

Access the Buckets Window

1. From the AppDynamics Home page, on the **Databases** card, click the name of the Couchbase collector that you want to view the buckets for.
2. Click the Buckets tab.

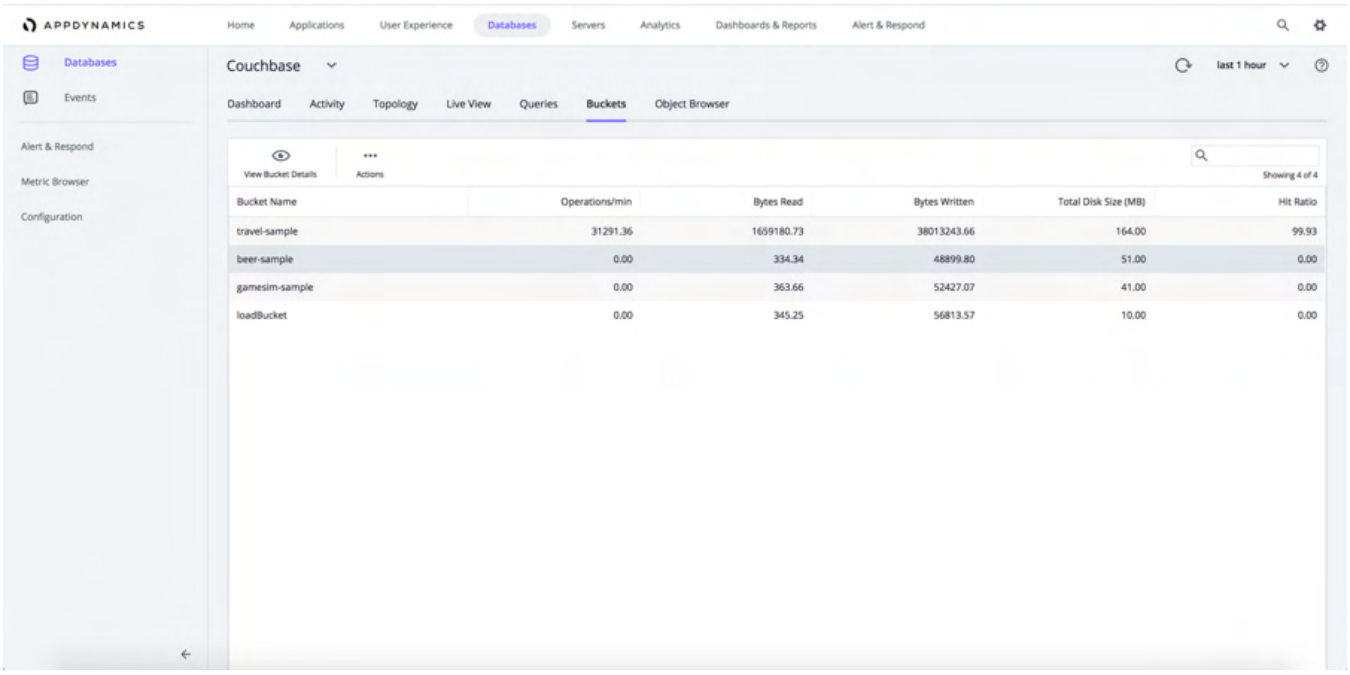
Buckets Window Features

From the Buckets window, you can:

- Double click the name of the bucket to view the bucket statistics for the selected time range.
- Click the down arrow next to the Couchbase collector name at the top of the page to choose to view the buckets of a different Couchbase collector by either selecting the Couchbase collector from the list or by searching for the Couchbase collector by entering text in the search bar and then clicking the refresh icon to show only database collectors that meet that search criteria.
- Click **Actions** to export the data on this window in a `.csv` formatted file that is automatically downloaded to your specified downloads directory.

From the Buckets window, you can view statistics for a bucket:

Operations (`ops`), Bytes Read (`bytes_read`), Bytes Written (`bytes_written`), Total Disk Size, in MB (`couch_total_disk_size`), Hit Ratio (`hit_ratio`)



Database Business Transactions Window

The Database **Business Transactions** window shows you the top Java or PHP business transactions, ranked by their cumulative execution time on the database. To see the Top N SQL run by a business transaction, click the business transaction and then click **View Top Queries**. From the Business Transaction top queries page, you can double-click a query to open the Query Details window.

Supported Platforms

AppDynamics supports Java as an application framework. Relational databases are supported as backends that need to be mapped on the Java APM to a monitored instance in database visibility. For more information, see Linking Database with application backend.

Access the Database Business Transactions Window

To access the Top Business Transactions window:

1. To view a database's business transactions, click the name of the database.
2. Click the Business Transactions tab.

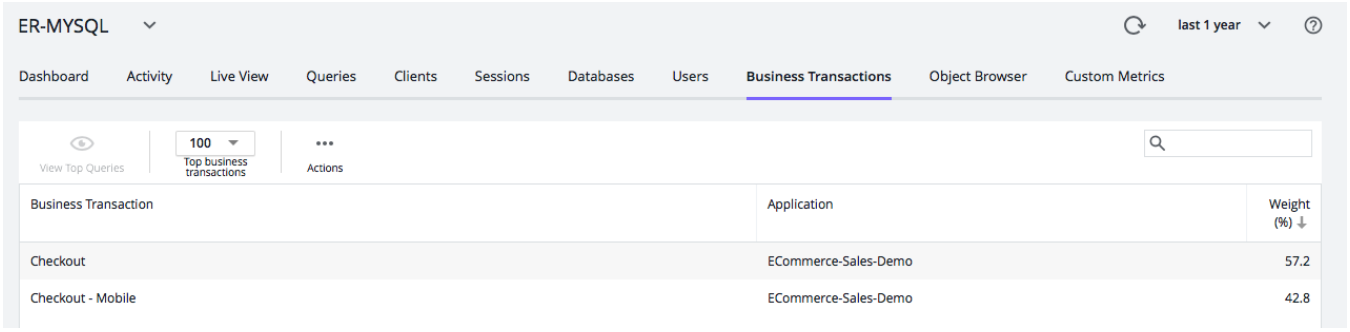
Database Business Transactions Window Features

From the Database Business Transactions window, you can:

- Choose to view information for the top 10, 50, 100 or 200 business transactions executing queries on the database.
- Double-click a business transaction to view the queries run in that particular transaction.
- Click the down arrow next to the database name at the top of the page to choose to view the business transactions that access a different database by either selecting the database from the list or by searching for the database by entering text in the search bar and then clicking the refresh icon to show only databases that meet that search criteria.
- Click **Actions** to export the data on this window in a .csv formatted file that is automatically downloaded to your specified downloads directory.

From the Database Business Transactions window, you can view:

- **Business Transaction**: the business transactions that run the most number of queries on the database.
- **Application**: the application that each business transaction is executing in.
- **Weight (%)**: the percentage of time that business transaction took to execute compared to other business transactions.



Database Applications Window

The Database **Applications** window is available for Postgres, Sybase ASE, Sybase IQ, and DB2.

The **Applications** window shows you the names of the Top N Applications on the database instance based on time. To see the Top N Queries run on the database by an application, click the name of an application and then click **View Top Queries**. From the Applications top queries page, to see more details about a query you can double-click a query to open the Query Details window.

Access the Database Applications Window

To access the Database Applications window:

1. To view a database's applications, click the name of the database.
2. Click the Applications tab.

Database Applications Window Features

From the Database Applications window, you can:

- Choose to view information for the top 10, 50, 100 or 200 applications most frequently accessing the database server.
- Double click the name of an application to view the top queries for one application in particular, or select the application and click **View Top Queries**.
- Click the down arrow next to the database collector name at the top of the page to choose to view the applications of a different MongoDB database collector by either selecting the database collector from the list or by searching for the database collector by entering text in the search bar and then clicking the refresh icon to show only database collectors that meet that search criteria.
- Click **Actions** to export the data on this window in a `.csv` formatted file that is automatically downloaded to your specified downloads directory.

From the Database Applications window, you can view:

- **Applications**: The name of the application run by an active session in the database.
- **Weight (%)**: The percentage of time the application was consuming the database resources in comparison with the usage percentage of other applications.

Database Object Browser Window

Related pages:

- Access Database Visibility from Application Monitoring Views
- Database Visibility
- Monitor Databases and Database Servers
- Monitor Database Performance

When you detect a problem with database performance, you may be able to correct the problem by adding an index, refusing access to certain users, or creating a new index on a table. You can browse through the database objects to find ways to tune the database, including the indexes that are currently available on the various tables. The contents of the **Database Object Browser** are updated when you access them or click **Refresh**.

> The objects available for browsing are database dependent. For clustered databases, select the node to browse its objects.

Access the Database Objects Window

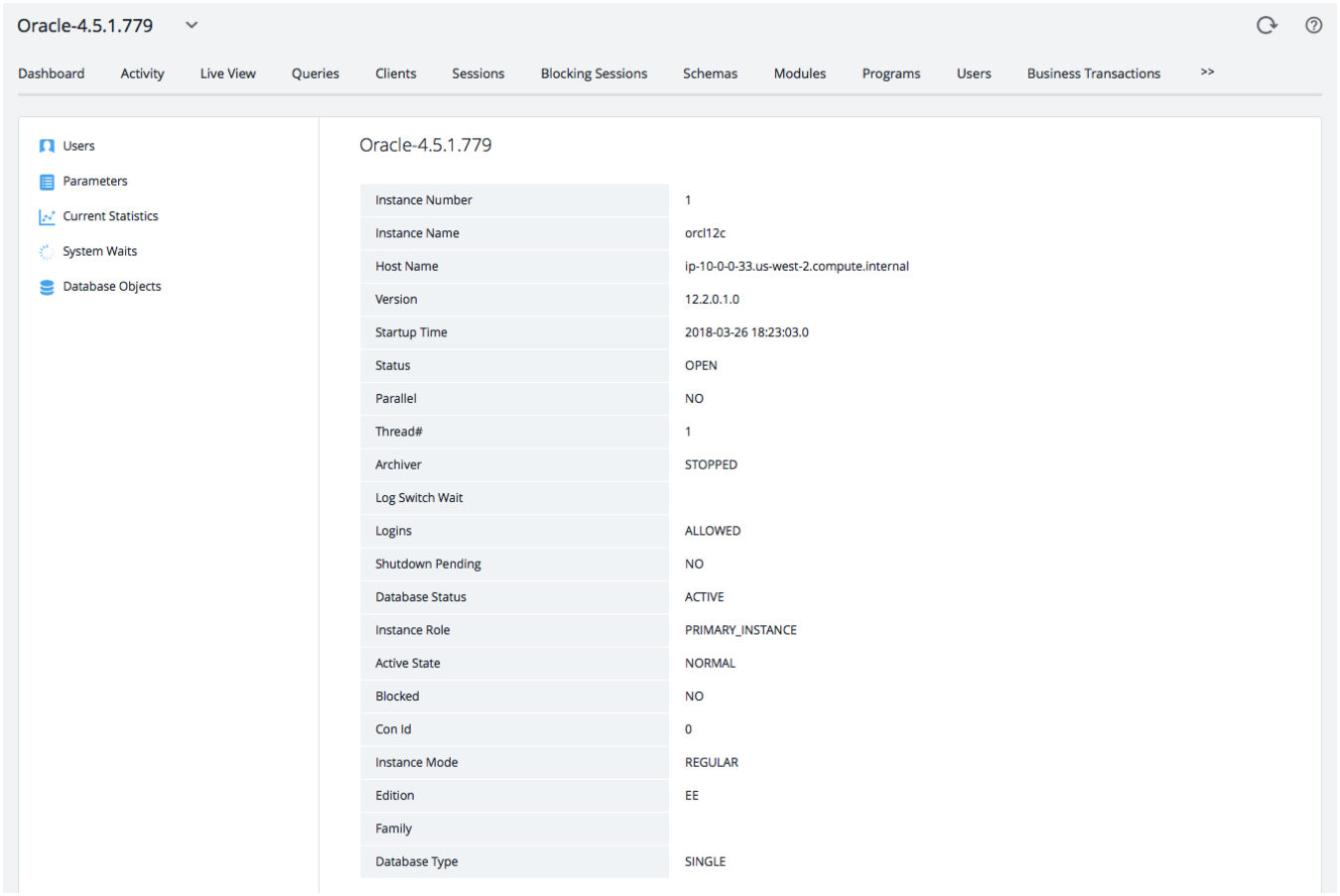
To access the Database Objects window:

1. To view a database's objects, click the name of the database.
2. Click the Objects tab.

Database Objects Window Features

From the **Database Objects** window you can:

- Click an object type in the tree to see more information about it.
- For Oracle, search for a specific database item.
- Click the down arrow next to the database name at the top of the page to choose to view the database objects of a different database by either selecting the database from the list or by searching for the database by entering text in the search bar and then clicking the refresh icon to show only databases that meet that search criteria.



The following are some examples of browsable content.

- Database uptime and version for all platforms
- **Users**: who has access to the database for all platforms (supported for Oracle, MySQL)
- **Job Status**: status of SQL agent job (supported for SQL Server)
- **Parameters**: database initialization parameters (supported for Oracle)
- **Variables**: system variables indicating how the database was configured.
- **Current Statistics**: the current value of statistics collected. The various statistics available are database dependent and are a superset of those shown in the **Database Monitoring - Metric Browser** page.
- **System Waits**: SQL waits experienced by the database
- **Database Objects**: search for specific objects, and browse through database schemas, tables, columns, and indices.

Database Custom Metrics Window

The **Custom Metrics** window displays metrics for the custom queries that you specified for Database Visibility to monitor. See Configure Custom Metrics.

Access the Database Custom Metrics Window

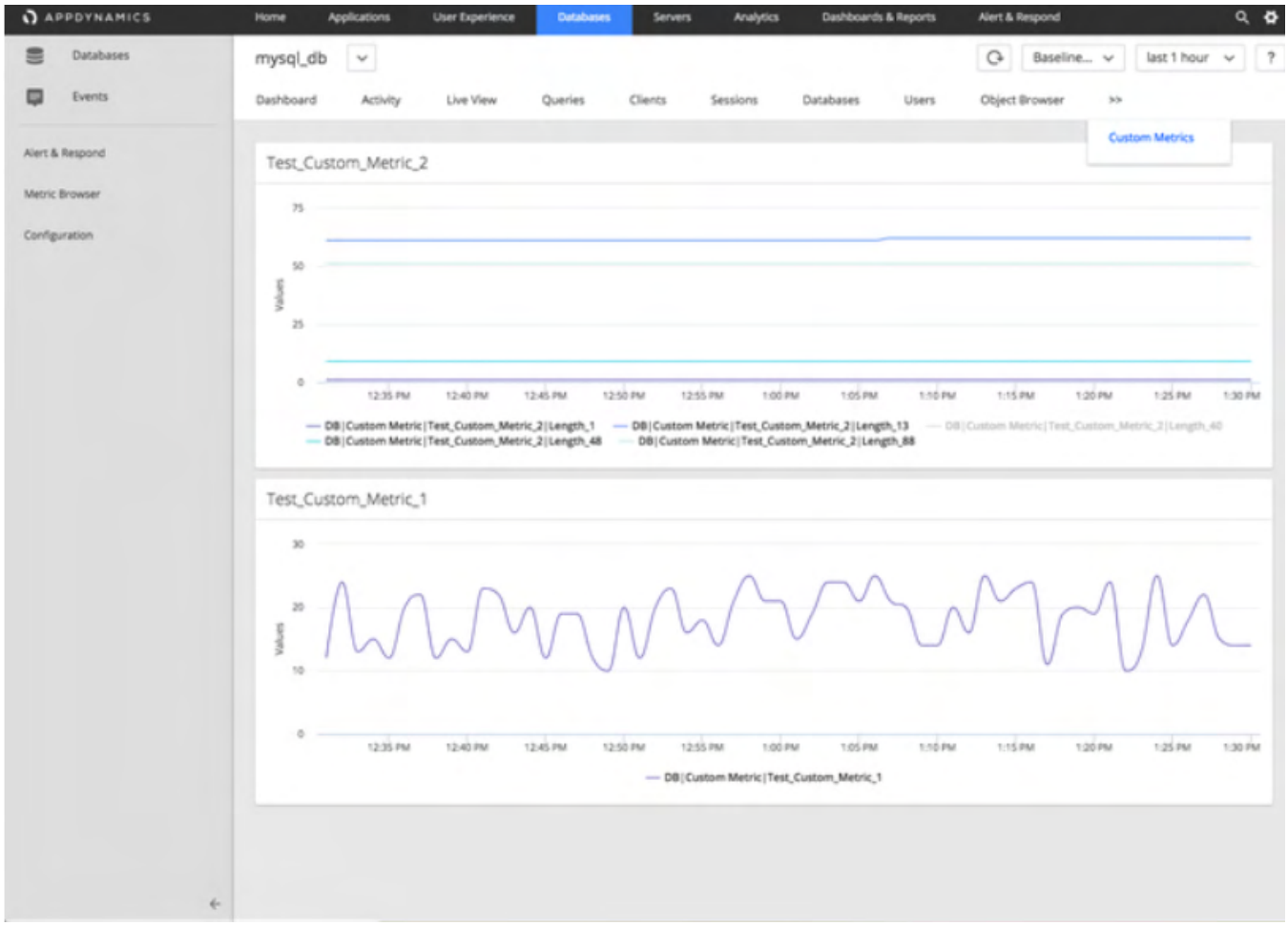
To access the Database **Custom Metrics** window:

1. To view a database's objects, click the name of the database.
2. Click the Custom Metrics tab.

Database Custom Metrics Window Features

From the Database Custom Metrics window, you can:

- View the values for each row of the custom query output. For example, if the query returns four rows, then the graph displays four lines.
- Double-click the graph to view the custom metric in the **Metric Browser**.



Monitor Database Server Hardware

Access the Database Server Hardware Metrics Window

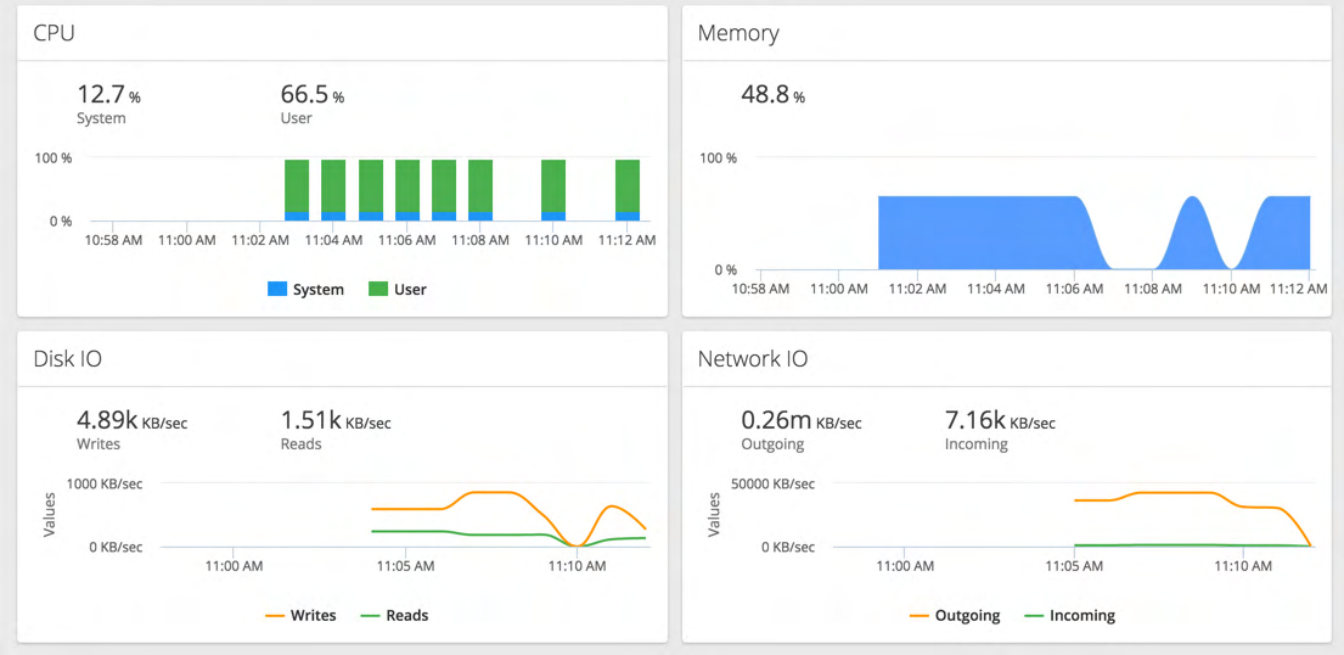
Hardware metrics are reported on:

- Database Dashboard window
- Database Live window
- Metric Browser

Hardware Metrics Graphs Features

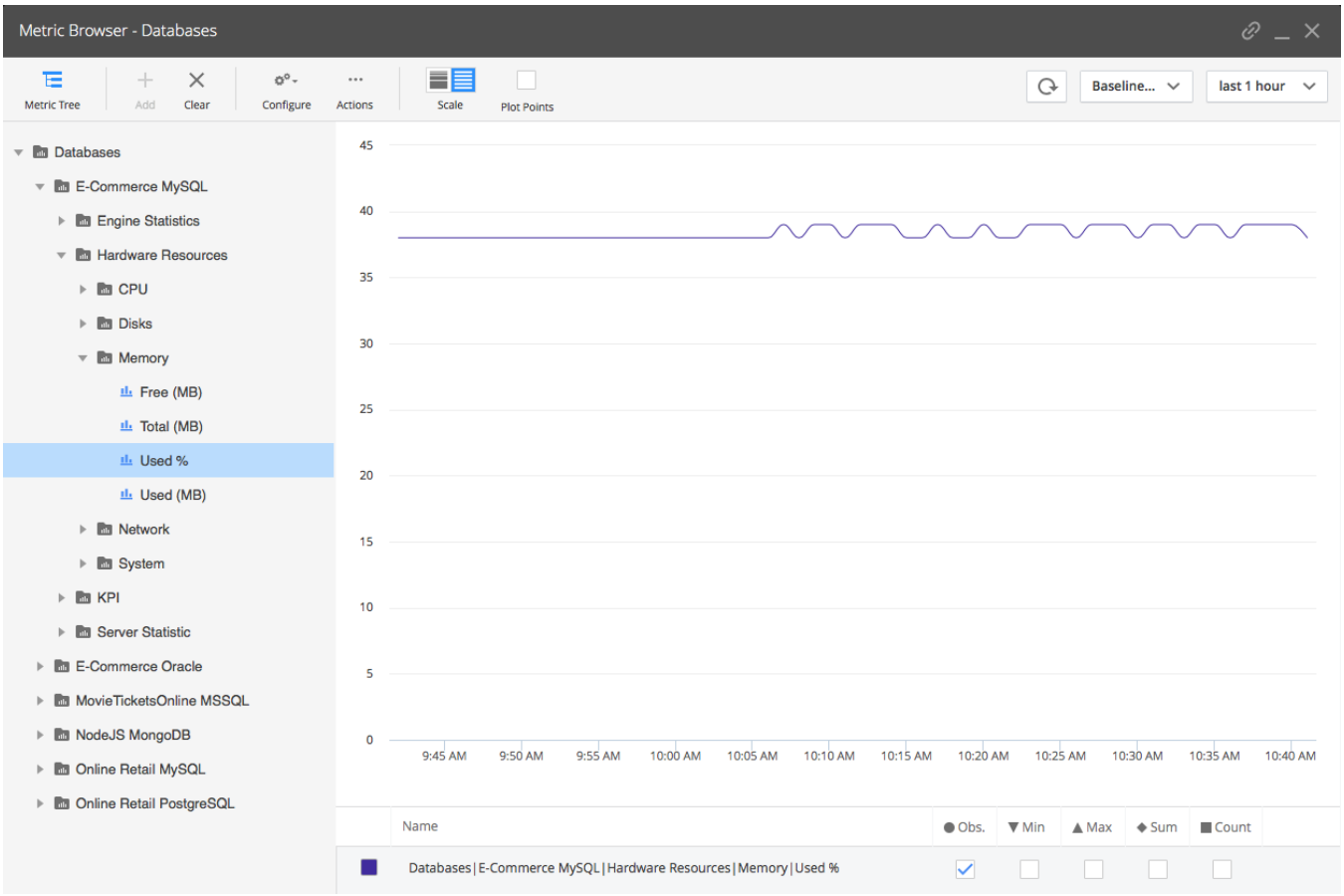
On the hardware metrics graphs, you can:

- Click anywhere on the graphs to get details about activity at that point in time.
- Double-click KPI points in the graph to display the Metric Browser showing the selected metric. Metrics are collected by the agent and reported once a minute.



Compare Server and Database Metrics

You can view and compare hardware and database metrics in the Metric Browser. This can help you visualize the effect of server performance on database performance.


Metric Browser - Databases
Metric Tree
Add
Clear
Configure
Actions
Scale
Plot Points
Baseline...
last 1 hour
Databases
E-Commerce MySQL
Engine Statistics
Hardware Resources
CPU
Disks
Memory
Free (MB)
Total (MB)
Used %
Used (MB)
Network
System
KPI
Server Statistic
E-Commerce Oracle
MovieTicketsOnline MSSQL
NodeJS MongoDB
Online Retail MySQL
Online Retail PostgreSQL
45
40
35
30
25
20
15
10
5
0
9:45 AM
9:50 AM
9:55 AM
10:00 AM
10:05 AM
10:10 AM
10:15 AM
10:20 AM
10:25 AM
10:30 AM
10:35 AM
10:40 AM
Name
Obs.
Min
Max
Sum
Count
Databases | E-Commerce MySQL | Hardware Resources | Memory | Used %

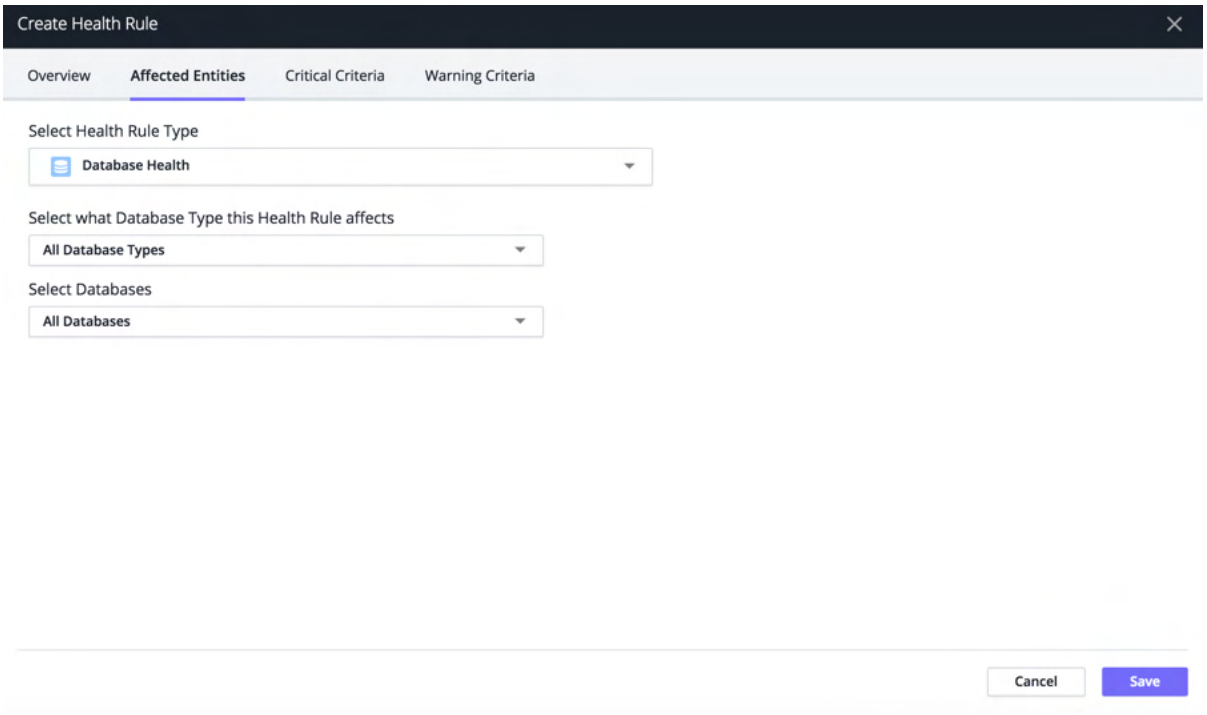
Database Health Rules and Alerts

You can configure AppDynamics Database Visibility to alert you when certain conditions are met or exceeded for monitored databases, operating systems, and server hardware. Use the **Getting Started Wizard** to guide you through the process if you are a new AppDynamics Pro user.

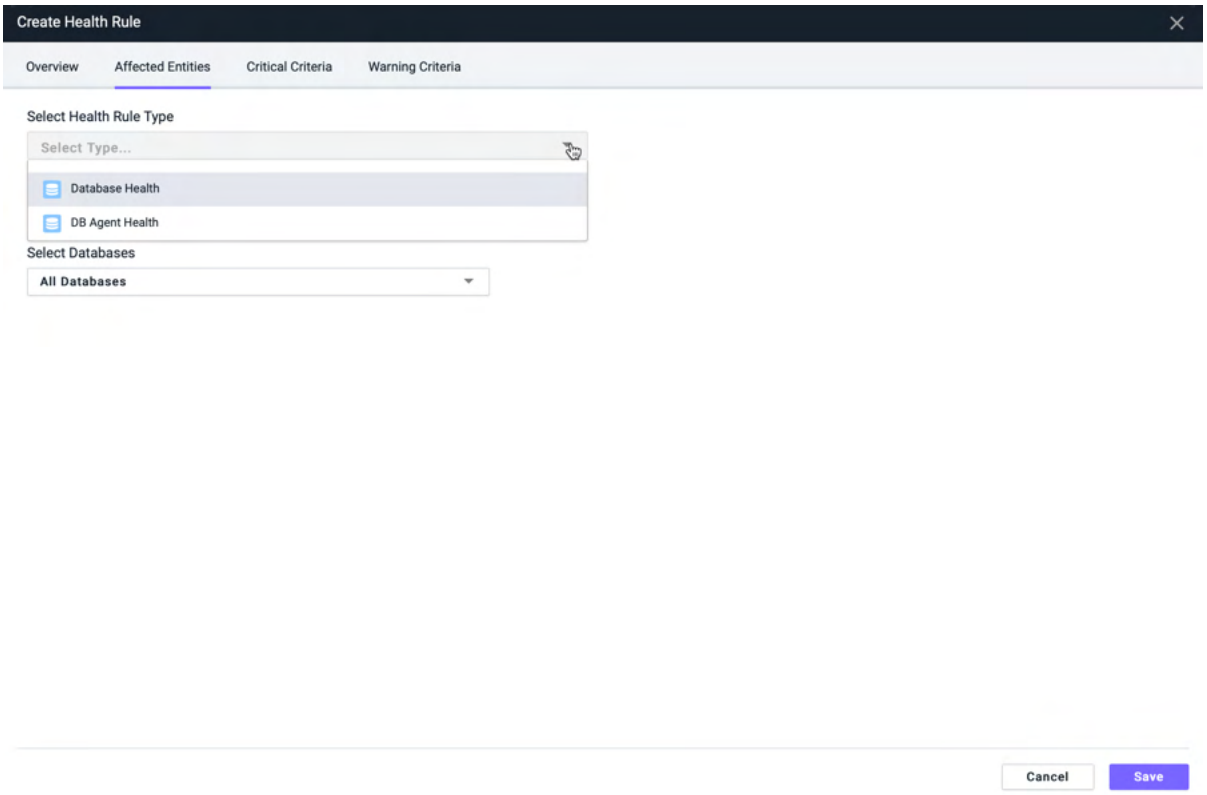
You configure Health Rules, Actions, Policies, and Email Digests for monitoring databases almost exactly as you would configure these for monitored applications.

To view the health rules for your databases, click **Alert & Respond > Health Rules** and select **Databases** from the dropdown at the top. Use the Affected Entities tab of the **Create Health Rule** dialog to specify the **Database Health** rules. See Alert and Respond.

In the Affected Entities tab, you can apply a database health rule to all databases, all databases of a specific database type, or to select instances of a specific database type being monitored. You can select the metrics on which to base the health rule. The health rule is violated when specified critical or warning conditions are met.



Similarly, you can create the health rule for Database Agent (**DB Agent Health)** and the Affected Entities tab of the **Create Health Rule** dialog allows specifying the **DB Agent Health** rule. You can apply a Database Agent (**DB Agent Health**) rule to all the Database Agents or specific Database Agents. You can select the required metrics on which to base the health rule. The health rule is violated when specified critical or warning conditions are met.



Each Health Rule is only evaluated for the subset of databases or database that you currently have permissions for. If your permissions have changed since you created the Health Rule, then your current permissions are applied to the Health Rule evaluation.

Database Policy Actions are limited to email or SMS message notifications, and custom actions that have been uploaded to the Controller.

Once you have defined Health Rules and Actions, you can create a Policy based on a Health Rule to send you an email or an SMS message when a Health Rule is violated.

Database Monitoring Metrics

Related page:

Metric Browser
This document describes metrics collected by the Database Agent. It shows the path to the metric from the main AppDynamics menu down to the relevant branch of the Metric Browser tree. Some of these metrics are displayed in the Database Visibility user interface and so are also documented with the name of the window, column, section, and metric name.

The Metric Browser window name uses the following format:

- Metric Browser - *application_name*: When you access the Metric Browser from Applications.
- Metric Browser - Database Monitoring: When you access the Metric Browser from Infrastructure.

For most types of metrics in the browser, you can click any of the points in the graph to view more information about the metric observed at that point in time. The information shown includes the metric identifier, date and time of the observation, along with any of the following values relevant to the metric:

- **Obs** (observed value): the average of all data points seen for that interval. For a cluster or a time rollup, this represents the weighted average across nodes or over time.
- **Min**: the minimum data point value seen for that interval
- **Max**: the maximum data point value seen for that interval
- **Sum**: the sum of all data point values seen for that interval.
- **Count**: number of observations aggregated in that one point. For example, a count of 5 indicates that there were 5 1-minute data points aggregated into one point.

Key Performance Indicators (KPI)

These metrics are available for all database platforms supported.

Calls per Minute: The number of SQL calls to the database per minute.

DB Availability: The times when the database is available (has an active connection). If the database is not available, nothing is reported.

Number of Connections: The number of connections established with the database at any point during the selected time period. A connection is a session established between a database client and a server. Since the value displayed is the average number of connections over a time range, it is common to see a very low number or 0 for this metric.

Time Spent in Execution (s): The current amount of time the database spent executing SQL statements.

Total Database Size (PostgreSQL only): The amount of disk space (in MB) that the database is using.

Total Lock Time: Total time the database was in lock state in seconds.

Cassandra Metrics

Metrics	Description
Average Read|Write latency (microseconds)	Total Read|Write divided Number of Reads|Writes
Commands Completed(Only For DSE Cassandra)	Total read repair commands completed since startup.
Commands Pending(Only For DSE Cassandra)	A current number of the read repair commands pending.
Compaction Tasks Completed	The number of completed compactions since server start or restart.
Compaction Tasks Pending	The estimated number of compactions remaining to perform.
Disk Load (bytes)	The size (in bytes) of the on-disk data size the node manages.
Dropped Mutations	A dropped Write attempt on a node
Exceptions	The number of internal exceptions caught. Under normal exceptions, this should be zero.
GC Collection Count	The total number of collections that have occurred.
GC Collection Time (ms)	The time (in milliseconds) that Cassandra JVM takes to perform the garbage collection.
Heap Memory Used (bytes)	The heap Memory Used by JVM in bytes.
Key Cache Hit Percentage	The percentage of Key cache(Cache for partition to sstable offsets) hit rate.
Max Partition Size	The size of the largest compacted partition (in bytes).

Memtable Live Data Size (bytes)	The disk space used by SSTables that belongs to this table (in bytes).
Messaging Latency	The latency for inter-node communication.
Non Heap Memory Used (bytes)	The memory used by JVM other than heap memory (in bytes).
Number of Reads|Writes	The number of the `Reads|Write` requests.
Prepared Statements Executed	A number of Prepared Statements executed.
Process CPU Load Percentage	The Percentage of the `recent cpu usage` for the Java Virtual Machine process.
Read Failures	The number of failed `Read` operations
Read Repair Attempted(Only For DSE Cassandra)	The read repairs attempted since startup.
Read Repaired Background(Only For DSE Cassandra)	A number of the read repairs performed asynchronously since startup.
Read Repaired Blocking(Only For DSE Cassandra)	A number of the read repairs performed synchronously since startup.
Read Timeout Exceptions	The number of timeouts encountered for the read request.
Read Unavailability Exceptions	The number of unavailable exceptions encountered for the read request.
Read|Write Throughput	Derived from Number of Reads|Writes metrics over 1 minute.
Regular Statements Executed	A number of nonprepared statements executed.
Repair Age Percentage	The percentage of the table data that is repaired on disk.
Responses Completed(Only For DSE Cassandra)	The current read repairs completed count.
Responses Pending(Only For DSE Cassandra)	The current read repair responses pending count.
System CPU Load Percentage	The Percentage of the `recent cpu usage` for the whole system.
ThreadPools|Request|Active Tasks|Authz Stage	The number of tasks being actively worked on by AuthzStage thread pool.
ThreadPools|Request|Active Tasks|Read Repair Stage	The number of tasks being actively worked on by Read Repair Stage thread pool.
ThreadPools|Request|Active Tasks|Request Response Stage	The number of tasks being actively worked on by Request Response Stage thread pool.
ThreadPools|Internal|Active Tasks|Anti Entropy Stage	The number of tasks being actively worked on by Anti Entropy Stage thread pool.
ThreadPools|Internal|Active Tasks|Background IO Stage	The number of tasks being actively worked on by Background IO Stage thread pool.
ThreadPools|Internal|Active Tasks|Gossip Stage	The number of tasks being actively worked on by Gossip Stage thread pool.
ThreadPools|Internal|Active Tasks|Internal Response Stage	The number of tasks being actively worked on by Internal Response Stage thread pool.
ThreadPools|Internal|Active Tasks|Migration Stage	The number of tasks being actively worked on by Migration Stage thread pool.
ThreadPools|Internal|Active Tasks|Misc Stage	The number of tasks being actively worked on by Misc Stage thread pool.
ThreadPools|Request|Pending Tasks|Authz Stage	The number of queued tasks queued up by AuthzStage thread pool.
ThreadPools|Request|Pending Tasks|Read Repair Stage	The number of queued tasks queued up by the Read Repair Stage thread pool.
ThreadPools|Request|Pending Tasks|Request Response Stage	The number of queued tasks queued up by Request Response Stage thread pool.
ThreadPools|Internal|Pending Tasks|Anti Entropy Stage	The number of queued tasks queued up by Anti Entropy Stage thread pool.
ThreadPools|Internal|Pending Tasks|Background IO Stage	The number of queued tasks queued up by Background IO Stage thread pool.
ThreadPools|Internal|Pending Tasks|Gossip Stage	The number of queued tasks queued up by Gossip Stage thread pool.
ThreadPools|Internal|Pending Tasks|Internal Response Stage	The number of queued tasks queued up by Internal Response Stage thread pool.
ThreadPools|Internal|Pending Tasks|Migration Stage	The number of queued tasks queued up by Migration Stage thread pool.
ThreadPools|Internal|Pending Tasks|Misc Stage	The number of queued tasks queued up by Misc Stage thread pool.
ThreadPools|Request|Currently Blocked Tasks|Authz Stage	The number of tasks that were blocked due to queue saturation on AuthzStage thread pool.

ThreadPools|Request|Currently Blocked Tasks|Read Repair Stage	The number of tasks that were blocked due to queue saturation on Read Repair Stage thread pool.
ThreadPools|Request|Currently Blocked Tasks|Request Response Stage	The number of tasks that were blocked due to queue saturation on Request Response Stage thread pool.
ThreadPools|Internal|Currently Blocked Tasks|Anti Entropy Stage	The number of tasks that were blocked due to queue saturation on Anti Entropy Stage thread pool.
ThreadPools|Internal|Currently Blocked Tasks|Background IO Stage	The number of tasks that were blocked due to queue saturation on Background IO Stage thread pool.
ThreadPools|Internal|Currently Blocked Tasks|Gossip Stage	The number of tasks that were blocked due to queue saturation on Gossip Stage thread pool.
ThreadPools|Internal|Currently Blocked Tasks|Internal Response Stage	The number of tasks that were blocked due to queue saturation on Internal Response Stage thread pool.
ThreadPools|Internal|Currently Blocked Tasks|Migration Stage";	The number of tasks that were blocked due to queue saturation on Migration Stage thread pool.
ThreadPools|Internal|Currently Blocked Tasks|Misc Stage";	The number of tasks that were blocked due to queue saturation on Misc Stage thread pool.
Total Read|Write latency (microseconds)	Total Read|Write Latency
Write Failures	The number of failed `Write` operations
Write Timeout Exceptions	Number of timeouts encountered for write request.
Write Unavailability Exceptions	Number of unavailable exceptions encounteredfor write request.

Couchbase Metrics

Gc_num: Number of objects garbage collected
Gc_pause_time: Garbage collection pause time
Gc_pause_percent: Garbage collection pause percentage

Memory_usage: Memory (in GB) currently used by Couchbase
Memory_total: Memory (in GB) used by Couchbase over the total period of time
Memory_system: Memory (in GB) used by the system.

Cpu_user_percent: Percentage of CPU used by the user
Cpu_sys_percent: Percentage of CPU used by the system

Request_completed_count: Number of requests completed
Request_active_count: Number of active requests
Request_per_sec_1min: Requests per second in the last minute
Request_per_sec_5min: Requests per second in the last 5 minutes
Request_per_sec_15min: Requests per second in the last 15 minutes
Request_time_mean: Average request time
Request_time_median: Median request time
Request_time_80: 80th percentile request time
Request_time_95: 95th percentile request time
Request_time_99: 99th percentile request time
Request_prepared: Number of requests prepared

cpu_utilization_rate: Rate of CPU utilization
swap_total: Total amount of swap available
swap_used: Amount of swap used
mem_total: Total available memory
mem_free: Total amount of free memory

cmd_get: Compare and Swap gets
couch_docs_actual_disk_size: The physical memory used in the node
couch_docs_data_size: Data size of couch documents associated with a node
couch_spatial_data_size: Size of object data for spatial views
couch_spatial_disk_size: Amount of disk space occupied by spatial views
couch_views_actual_disk_size: Amount of disk space occupied by Couch views
couch_views_data_size: Size of object data for Couch views
curr_items: Number of active items in memory
curr_items_tot: Total number of items
ep_bg_fetched: Disk reads per second
get_hits: Number of get hits
mem_used: Engine's total memory usage
ops: Spatial operations
vb_replica_curr_items: Number of in memory items
operations per second: The total amount of operations per second (including XDCR) to this bucket (measured from `cmd_lookup` + `cmd_set` + `incr_misses + incr_hits` + `decr_misses` + `decr_hits` + `delete_misses` + `delete_hits` + `ep_num_ops_del_meta` + `ep_num_ops_get_meta` + `ep_num_ops_set_meta`).
active docs resident %: The percentage of active items cached in RAM in this bucket (measured from `vb_active_resident_items_ratio`).
gets per sec: The number of reads (get operations) per second from this bucket (measured from `cmd_get`).
sets per sec: The number of writes (set operations) per second to this bucket (measured from `cmd_set`).
deletes per sec: The number of delete operations per second for this bucket (measured from `delete_hits`).
disk creates per sec: The number of new items created on disk per second for this bucket (measured from `vb_active_ops_create` + `vb_replica_ops_create` + `vb_pending_ops_create`).
disk updates per sec: The number of items updated on disk per second for this bucket (measured from `vb_active_ops_update` + `vb_replica_ops_update` + `vb_pending_ops_update`).
disk reads per sec: The number of reads per second from disk for this bucket (measured from `ep_bg_fetched`).
disk read failures: The number of disk read failures (measured from `ep_data_read_failed`).
disk write failures: The number of disk write failures (measured from `ep_data_write_failed`).
temp OOM per sec: The number of back-offs sent per second to client SDKs due to the out-of-memory situations from this bucket (measured from `ep_tmp_oom_errors`).
cache miss ratio: The percentage of reads per second to this bucket from disk as opposed to RAM (measured from `ep_bg_fetches` / `cmd_lookup` * 100).
disk queue fill rate: The total number of items per second being put on the disk queue in this bucket (measured from `ep_diskqueue_fill`).
disk queue drain rate: The total number of items per second being written to disk in this bucket (measured from `ep_diskqueue_drain`).
disk write queue: The number of items waiting to be written to disk in this bucket (measured from ep_queue_size+ep_flusher_todo).

DB2 Server Metrics

ACTIVE_SORTS: The number of sorts in the database that currently have a sort heap allocated.

AGENTS_TOP: At the database level, it is the maximum number of agents for all applications.

APPLS_CUR_CONS: The number of applications that are currently connected to the database.

APPLS_IN_DB2: The number of applications that are currently connected to the database, and for which the database manager is currently processing a request.

BINDS_PRECOMPILES: The number of binds and pre-compiles attempted. You can use this element to gain insight into the current level of activity within the database manager. This value does not include the count of int_auto_rebinds, but it does include binds that occur as a result of the REBIND PACKAGE command.

CONNECTIONS_TOP: The highest number of simultaneous connections to the database since the database was activated.

COORD_AGENTS_TOP: The highest number of coordinating agents. If the peak number of coordinating agents represents too high a workload for this node, you can reduce this upper boundary by changing the max_coordagents configuration parameter.

DDL_SQL_STMTS: The number of SQL Data Definition Language (DDL) statements that were executed.

DEADLOCKS: The number of deadlocks that have occurred.

DIRECT_READ_REQS: Use the following formula to calculate the average number of sectors that are read by a direct read: direct_reads / direct_read_reqs.

DIRECT_READ_TIME: Time spent doing direct read operations.

DIRECT_READS: Direct reads are performed in units, the smallest being a 512-byte sector. They are used when: Reading LONG VARCHAR columns, Reading LOB (large object) columns, Performing a backup.

DIRECT_WRITE_REQS: Use the following formula to calculate the average number of sectors that are written by a direct write: direct_writes / direct_write_reqs.

DIRECT_WRITE_TIME: Time spent doing direct write operations.

DIRECT_WRITES: Direct writes are performed in units, the smallest being a 512-byte sector. They are used when: Writing LONG VARCHAR columns, Writing LOB (large object) columns, Performing a restore, Performing a load, Allocating new extents for SMS table space if MPFA is enabled (which is the default).

DYNAMIC_SQL_STMTS: The number of dynamic SQL statements that were attempted.

ELAPSED_EXEC_TIME_MS: The total time (in milliseconds) required to execute all the statements for a particular application during the specified time period.

ELAPSED_EXEC_TIME_S: The total time (in seconds) required to execute all the statements for a particular application during the specified time period.

FAILED_SQL_STMTS: The number of SQL statements that were attempted, but failed.

INT_AUTO_REBINDS: The number of commits initiated internally by the database manager.

INT_COMMITS: The number of commits initiated internally by the database manager.

INT_DEADLOCK_ROLLBACKS: The number of forced rollbacks initiated by the database manager due to a deadlock. A rollback is performed on the current unit of work in an application selected by the database manager to resolve the deadlock.

INT_ROLLBACKS: The number of rollbacks initiated internally by the database manager.

INT_ROWS_DELETED: The number of rows deleted from the database as a result of internal activity.

INT_ROWS_INSERTED: The number of rows inserted into the database as a result of internal activity caused by triggers.

INT_ROWS_UPDATED: The number of rows updated from the database as a result of internal activity.

LOCK_ESCALS: The number of times that locks have been escalated from several row locks to a table lock.

LOCK_LIST_IN_USE: The total number of bytes of lock list memory in use.

LOCK_TIMEOUTS: The number of times that a request to lock an object timed-out instead of being granted.

LOCK_WAIT_TIME: The total elapsed time (in milliseconds) spent waiting for locks.

LOCK_WAITS: The number of times that applications or connections waited for locks.

LOCKS_HELD: The number of locks currently held.

LOCKS_WAITING: The number of agents waiting on a lock.

LOG_HELD_BY_DIRTY_PAGES: The amount of log (in bytes) corresponding to the difference between the oldest dirty page in the database and the top of the active log.

LOG_READ_TIME_NS: The total elapsed time spent by the logger reading log data from the disk. For event monitors that write to tables, the value of this element is given in microseconds by using the BIGINT data type.

LOG_READ_TIME_S: At the database level, this is the number of subagents for all applications.

LOG_READS: The number of log pages read from disk by the logger.

LOG_TO_REDO_FOR_RECOVERY: The size of the log (in bytes) that will have to be redone for crash recovery.

LOG_WRITE_TIME_NS: The total elapsed time spent by the logger writing log data to the disk. For event monitors that write to tables, the value of this element is given in microseconds by using the BIGINT data type.

LOG_WRITE_TIME_S: At the database level, this is the number of subagents for all applications.

LOG_WRITES: The number of log pages written to disk by the logger.

NUM_ASSOC_AGENTS: At the database level, this is the number of subagents for all applications.

NUM_INDOUBT_TRANS: The number of outstanding indoubt transactions in the database. Indoubt transactions hold log space for uncommitted transactions, which can cause the logs to become full. When the logs are full, further transactions cannot be completed. The resolution of this problem involves a manual process of heuristically resolving the indoubt transactions. This monitor element provides a count of the number of currently outstanding indoubt transactions that must be heuristically resolved.

NUM_LOG_BUFFER_FULL: The number of times agents had to wait for log data to write to disk while copying log records into the log buffer.

NUM_LOG_DATA_FOUND_IN_BUFFER: The number of times log data was read from buffer instead of from disk, which is slower.

NUM_LOG_PART_PAGE_IO: Number of I/O requests issued by the logger for writing partial log data to disk. To determine if the current disk is adequate for logging, use this metric in conjunction with log_writes, log_write_time, and num_log_write_io.

NUM_LOG_READ_IO: Number of I/O requests issued by the logger to read log data from disk. To determine if the current disk is adequate for logging, use this metric in conjunction with log_reads and log_read_time.

NUM_LOG_WRITE_IO: Number of I/O requests issued by the logger to write log data to disk. To determine if the current disk is adequate for logging, use this metric in conjunction with log_writes and log_write_time.

POOL_ASYNC_DATA_READ_REQS: Number of asynchronous read requests by the prefetcher to the operating system. These requests are usually large block I/Os of multiple pages.

POOL_ASYNC_DATA_READS: Number of data pages read in from the table space physical containers by asynchronous engine dispatchable units for all types of table spaces. To determine the number of physical read that were performed synchronously, use this metric along with the pool_data_p_reads metrics.

POOL_ASYNC_DATA_WRITES: Number of times a buffer pool data page was physically written to disk by either an asynchronous page cleaner or prefetcher. To determine how well the buffer page cleaners are performing, use this metric in conjunction with pool_data_writes and pool_async_data_writes.

POOL_ASYNC_INDEX_READS: Number of index pages read in from the physical table space containers by asynchronous engine dispatchable units for all types of table spaces. To determine how well the prefetchers are working, compare the ratio of asynchronous reads to total physical reads. Use this information to tune the num_ioservers configuration parameter.

POOL_ASYNC_INDEX_WRITES: Number of times a buffer pool index page was physically written to disk by either an asynchronous page cleaner or prefetcher.

POOL_ASYNC_READ_TIME: Total number of milliseconds spent reading in data and index pages from physical table space containers by asynchronous engine dispatchable units for all types of table spaces. Use this metric to analyze the I/O work being performed.

POOL_ASYNC_WRITE_TIME: Total number of milliseconds spent writing data or index pages from the buffer pool to disk by database manager page cleaners. Use this metric to analyze the I/O work being performed.

POOL_DATA_FROM_ESTORE: Number of buffer pool data pages read from the extended storage monitor.

POOL_DATA_L_READS: Number of data pages which have been requested from the buffer pool (logical) for regular and large table spaces. The functionality to record buffer pool information at the statement level is supported for API and CLP snapshot requests.

POOL_DATA_P_READS: Number of data pages read in from the table space containers (physical) for regular and large table spaces. The functionality to record buffer pool information at the statement 7 level is supported for API and CLP snapshot requests.

POOL_DATA_WRITES: Number of times a buffer pool data page was physically written to disk.

POOL_DRTY_PG_STEAL_CLNS: Number of times a page cleaner was invoked because a synchronous write was needed during the victim buffer replacement for the database.

POOL_DRTY_PG_THRSH_CLNS: Number of times a page cleaner was invoked because a buffer pool had reached the dirty page threshold criterion for the database.

POOL_INDEX_L_READS: Number of index pages which have been requested from the buffer pool (logical) for 9 regular and large table spaces. The functionality to record buffer pool information at the statement level is supported for API and CLP snapshot requests.

POOL_INDEX_P_READS: Number of index pages read in from the table space containers (physical) for regular and large table spaces. The functionality to record buffer pool information at the statement level is supported for API and CLP snapshot requests.

POOL_INDEX_WRITES: Number of times a buffer pool index page was physically written to disk.

POOL_LSN_GAP_CLNS: Number of times a page cleaner was invoked because the logging space used had reached a predefined criterion for the database.

POOL_NO_VICTIM_BUFFER: Number of times an agent did not have a preselected victim buffer available.

POOL_READ_TIME: Number of milliseconds spent reading in data and index pages from the physical table space for all types of table spaces.

POOL_TEMP_DATA_L_READS: Number of data pages which have been requested from the logical buffer pool for temporary table spaces.

POOL_TEMP_DATA_P_READS: Number of data pages read in from the physical table space containers for temporary table spaces.

POOL_TEMP_INDEX_L_READS: Number of index pages which were requested from the logical buffer pool for temporary table spaces.

POOL_TEMP_INDEX_P_READS: Number of index pages read in from the physical table space containers for temporary table spaces.

POOL_WRITE_TIME: Number of milliseconds spent physically writing data or index pages from the buffer pool to disk.

PREFETCH_WAIT_TIME: Number of milliseconds spent waiting for an I/O prefetcher server to finish loading pages into the buffer pool.

ROWS_DELETED: Number of row deletions attempted.

ROWS_INSERTED: Number of row insertions attempted.

ROWS_READ: Number of rows read from tables.

ROWS_SELECTED: Number of rows that have been selected and returned to the application.

ROWS_UPDATED: Number of row updates attempted.

SEC_LOG_USED_TOP: The maximum number of bytes of secondary log space used.

SEC_LOGS_ALLOCATED: Number of secondary log files currently being used for the database.

SELECT_SQL_STMTS: The number of SQL SELECT statements that were executed.

SORT_HEAP_ALLOCATED: The number of allocated pages of sort heap space for all sorts at the level chosen and at the current time.

SORT_OVERFLOWS: The number of sorts that ran out of sort heap and may have required disk space for temporary storage.

SORT_SHRHEAP_ALLOCATED: The total amount of shared sort memory allocated in the database.

SORT_SHRHEAP_TOP: Database-wide shared sort memory high-water mark in 4k pages.

STATIC_SQL_STMTS: The number of static SQL statements that were attempted.

TOT_LOG_USED_TOP: The maximum number of bytes used for log space.

TOTAL_APP_COMMITS: Total number of commit statements issued by the client application.

TOTAL_APP_ROLLBACKS: Total number of rollback statements issued by the client application.

TOTAL_CONS: Number of newly opened connections to the database.

TOTAL_LOG_AVAILABLE: Number of bytes of active log space in the database that is not being used by uncommitted transactions.

TOTAL_LOG_USED: Number of bytes of active log space currently used in the database.

TOTAL_SEC_CONS: The number of connections made by a subagent to the database at the node.

TOTAL_SORT_TIME: The total elapsed time (in milliseconds) for all sorts that have been executed.

TOTAL_LOCK_TIME: The total time for which database is in locked state.

TOTAL_SORTS: The number of sorts that have been executed.

TOT_LOG_USED_TOP: The maximum amount of total log space used (in bytes).

UID_SQL_STMTS: Number of SQL UPDATE, INSERT, and DELETE statements that were executed.

UNREAD_PREFETCH_PAGES: Number of pages that the prefetcher read in that were never used.

MongoDB Server Metrics

asserts_msg: The number of message asserts. These are internal server errors that have a well defined text string. Stack traces are logged for these.

asserts_regular: The number of regular asserts raised since this process started.

asserts_user: The number of user asserts. These are errors that can be generated by a user such as out of disk space or duplicate key.

asserts_warning: The number of warnings raised since this process started.

BackgroundFlushing_flushes: The number of times the database has flushed all writes to disk.

BackgroundFlushing_total_ms: The number of milliseconds (ms) that the mongod processes have spent writing (i.e. flushing) data to disk.

Connections_available: The number of unused available connections that the database can provide. Consider this value in combination with the value of current to understand the connection load on the database, and the UNIX ulimit Settings document for more information about system thresholds on available connections.

Connections_current: The number of connections to the database server from clients. This number includes the current shell session. Consider the value of available to add more context to this datum. This figure will include the current shell connection as well as any inter-node connections to support a replica set or sharded cluster.

Cursor_timedOut: The number of cursors that have timed out since the server process started. If this number is large or growing at a regular rate, this may indicate an application error.

Cursor_totalOpen: The number of cursors that MongoDB is maintaining for clients. Because MongoDB exhausts unused cursors, typically this value small or zero. However, if there is a queue, stale tailable cursors, or a large number of operations this value may rise.

globalLock_ActiveClients: The number of connected clients.

globalLock_CurrentQueue: The number of operations queued waiting for the lock. A consistently small queue, particularly of shorter operations should cause no concern. Also, consider this value in light of the size of queue waiting for the read lock (e.g. readers) and write lock (e.g. writers) individually.

Mem_mapped: The number of megabytes of memory mapped by the database. Because MongoDB uses memory-mapped files, this value will be roughly equivalent to the total size of your databases.

Mem_resident: The amount of RAM, in megabytes (MB), currently used by the database process. In normal use this value tends to grow. In dedicated database servers this number tends to approach the total amount of system memory.

Mem_virtual: The quantity, in megabytes (MB), of virtual memory used by the mongod process. With journaling enabled, the value of virtual is at least twice the value of mapped. If virtual value is significantly larger than mapped (e.g. 3 or more times), this may indicate a memory leak.

Network_bytesIn: The amount of network traffic, in bytes, received by this database. Use this value to ensure that network traffic sent to the mongod process is consistent with expectations and overall inter-application traffic.

Network_bytesOut: The amount of network traffic, in bytes, sent from this database. Use this value to ensure that network traffic sent by the mongod process is consistent with expectations and overall inter-application traffic.

Network_numRequests: The number of distinct requests that the server has received. Use this value to provide context for the bytesIn and bytesOut values to ensure that MongoDB network utilization is consistent with expectations and application use.

OpCounters_command: The number of commands sent to MongoDB.

OpCounters_delete: The number of Delete operations.

OpCounters_getmore: The number of GetMore operations.

OpCounters_insert: The number of Insert operations.

OpCounters_query: The number of Query operations.

OpCounters_update: The number of Update operations.

Oplog_Max_Size *(new in 4.5.4)*: The maximum size of the operation log

Oplog_Size *(new in 4.5.4)*: The size of the operation log

OplogTimeDiff_in_sec *(new in 4.5.4)*: The difference (in seconds) between the first entry in the log and the last entry in the log

Repl_command: The number of Replicated Commands issued to the database.

Repl_delete: The number of Replicated Delete operations.

Repl_getmore: The number of GetMore operations. This counter can be high even if the query count is low. Secondary nodes send getMore operations as part of the replication process.

Repl_insert: The number of replicated insert operations.

Repl_query: The number of Replicated Queries.

Repl_update: The number of Replicated Update Operations.

Replication_MyState *(new in 4.5.4)*: An integer between 0 and 10 that represents the replica state of the current member.

Index_missRatio: Ratio of index hits to misses. If there are a lot of index misses then you should look at your queries to see if they are optimally using your indexes. You may need to add new indexes to make the queries run faster. You can explain the query to see which index queries are hitting and the total execution time so you can compare query performance before and after adding the new indexes.

Microsoft SQL Server Metrics

Active cursors: The number of active cursors.

Active Temp Tables: The number of temporary tables/table variables in use.

Active Transactions: The number of active transactions for the database.

AU cleanup batches/sec: The number of batches per second that were completed successfully by the background task that cleans up deferred dropped allocation units.

AU cleanups/sec: The number of allocation units per second that were successfully dropped the background task that cleans up deferred dropped allocation units. Each allocation unit drop requires multiple batches.

Auto-Param Attempts/sec: The number of auto-parameterization attempts per second. Total should be the sum of the failed, safe, and unsafe auto-parameterizations. Auto-parameterization occurs when an instance of SQL Server tries to parameterize a Transact-SQL request by replacing some literals with parameters so that reuse of the resulting cached execution plan across multiple similar-looking requests is possible. Note that auto-parameterizations are also known as simple parameterizations in newer versions of SQL Server. This counter does not include forced parameterizations.

Average Latch Wait Time (ms): Average latch wait time (in milliseconds) for latch requests that had to wait.

Average Wait Time (ms): Average amount of wait time (in milliseconds) for each lock request that resulted in a wait.

Backup/Restore Throughput/sec: Read/write throughput for backup and restore operations of a database per second. For example, you can measure how the performance of the database backup operation changes when more backup devices are used in parallel or when faster devices are used. Throughput of a database backup or restore operation allows you to determine the progress and performance of your backup and restore operations.

Batch Requests/sec: The number of Transact-SQL command batches received per second. This statistic is affected by all constraints (such as I/O, number of users, cache size, complexity of requests, and so on). High batch requests mean good throughput.

Buffer cache hit ratio: Percentage of pages found in the buffer cache without having to read from disk. The ratio is the total number of cache hits divided by the total number of cache lookups over the last few thousand page accesses. The ratio is displayed as a percentage. After a long period of time, the ratio moves very little. Because reading from the cache is much less expensive than reading from disk, you want this ratio to be high. Generally, you can increase the buffer cache hit ratio by increasing the amount of memory available to SQL Server.

Bulk Copy Rows/sec: The number of rows bulk copied per second.

Bulk Copy Throughput/sec: Amount of data bulk copied (in kilobytes) per second.

By-reference Lob Create Count: Count of large object (lob) values that were passed by reference. By-reference lobs are used in certain bulk operations to avoid the cost of passing them by value.

By-reference Lob Use Count: Count of by-reference lob values that were used. By-reference lobs are used in certain bulk operations to avoid the cost of passing them by-value.

Cache Hit Ratio: Ratio between cache hits and lookups. The ratio is displayed as a percentage.

Cache Object Counts: The number of cache objects in the cache.

Cache Objects in use: The number of cache objects in use.

Cache Pages: The number of 8-kilobyte (KB) pages used by cache objects.

Cached Cursor Counts: The number of cursors of a given type in the cache.

Checkpoint pages/sec: The number of pages flushed to disk per second by a checkpoint or other operation that require all dirty pages to be flushed.

CLR Execution: Total execution time in CLR (microseconds)

Connection Memory (KB): Total amount of dynamic memory the server is using for maintaining connections.

Count Lob Readahead: Count of lob pages on which readahead was issued.

Count Pull In Row: Count of column values that were pulled in-row from off-row.

Count Push Off Row: Count of column values that were pushed from in-row to off-row.

Cursor memory usage: Amount of memory consumed by cursors in kilobytes (KB).

Count/sec: Times each type of cached cursor has been used. Cursor memory usage Amount of memory consumed by cursors in kilobytes (KB).

Cursor Requests/sec: The number of SQL cursor requests received by the server. Cursor worktable usage Number of worktables used by cursors. Data File(s) Size (KB) Cumulative size (in kilobytes) of all the data files in the database including any automatic growth. Monitoring this counter is useful, for example, for determining the correct size of Database pages Number of pages in the buffer pool with database content.

Cursor worktable usage: Number of worktables in use by cursors.

Data File(s) Size(KB): Cumulative size (Kb) of all data files in the database including any automatic growth. Monitoring this counter to determine the correct size of tempdb.

Database pages: Number of database pages in use.

DBCC Logical Scan Bytes/sec: The number of logical read scan bytes per second for database console commands (DBCC).

Deferred Dropped rowsets: The number of rowsets created as a result of aborted online index build operations that are waiting to be dropped by the background task that cleans up deferred dropped rowsets.

Dropped rowset cleanups/sec: The number of rowsets per second created as a result of aborted online index build operations that were successfully dropped by the background task that cleans up deferred dropped rowsets.

Dropped rowsets skipped/sec: The number of rowsets per second created as a result of aborted online index build operations that were skipped by the background task that cleans up deferred dropped rowsets created.

Errors/sec: The number of errors/sec.

Event Notifications Delayed Drop: The number of event notifications waiting to be dropped by a system thread.

Extent Deallocations/sec: The number of extents deallocated per second in all databases in this instance of SQL Server.

Extents Allocated/sec: The number of extents allocated per second in all databases in this instance of SQL Server.

Failed AU cleanup batches/sec: The number of batches per second that failed and required retry, by the background task that cleans up deferred dropped allocation units. Failure could be due to lack of memory or disk space, hardware failure, and other reasons.

Failed Auto-Params/sec: The number of failed auto-parameterization attempts per second. This should be small. Note that auto-parameterizations are also known as simple parameterizations in later versions of SQL Server.

Failed leaf page cookie: The number of times that a leaf page cookie could not be used during an index search since changes happened on the leaf page. The cookie is used to speed up the index search.

Failed tree page cookie: The number of times that a tree page cookie could not be used during an index search since changes happened on the parent pages of those tree pages. The cookie is used to speed up the index search.

Forced Parameterizations/sec: The number of successful forced parameterizations per second.

Forwarded Records/sec: The number of records per second fetched through forwarded record pointers.

Free list stalls/sec: The number of requests per second that had to wait for a free page.

Free Space in tempdb (KB): The amount of space (in kilobytes) available in FreeSpace Page.

FreeSpace Page Fetches/sec: Number of pages fetched/second by free space scans. These scans search for free space within pages already allocated to an allocation unit, to address requests to insert or modify record fragments.

FreeSpace Scans/sec: The number of scans per second that were initiated to search for free space within pages already allocated to an allocation unit to insert or modify record fragment. Each scan may find multiple pages.

Full Scans/sec: The number of unrestricted full scans per second. These can be either base-table or full-index scans.

Granted Workspace Memory (KB): Total amount of memory currently granted to executing processes such as hash, sort, bulk copy, and index creation operations.

Guided Plan Executions/sec: The number of plan executions per second in which the query plan has been generated by using a plan guide.

HTTP Authenticated Requests: The number of authenticated HTTP requests started per second. Index Searches/sec Number of index searches per second. These are used to start a range scan, reposition a range scan, revalidate a scan point, fetch a single index record, and search down the index to locate where to insert a new row.

Index Searches/sec: Number of index searches per second. These are used to start a range scan, reposition a range scan, revalidate a scan point, fetch a single index record, and search down the index to locate where to insert a new row.

Latch Waits/sec: The number of latch requests that could not be granted immediately.

Lazy writes/sec: The number of buffers written per second by the buffer manager's lazy writer. The lazy writer is a system process that flushes out batches of dirty, aged buffers (buffers that contain changes that must be written back to disk before the buffer can be reused for a different page) and makes them available to user processes. The lazy writer eliminates the need to perform frequent checkpoints in order to create available buffers.

LobHandle Create Count: Count of temporary lobs created.

LobHandle Destroy Count: Count of temporary lobs destroyed.

LobSS Provider Create Count: Count of LOB Storage Service Providers (LobSSP) created. One worktable created per LobSSP.

LobSS Provider Destroy Count: Count of LobSSP destroyed.

LobSS Provider Truncation Count: Count of LobSSP truncated.

Lock Blocks: Current number of lock blocks in use on the server (refreshed periodically). A lock block represents an individual locked resource, such as a table, page, or row.

Lock Blocks Allocated: Current number of allocated lock blocks. At server startup, the number of allocated lock blocks plus the number of allocated lock owner blocks depends on the SQL Server

Lock Memory (KB): Total amount of dynamic memory the server is using for locks.

Lock Owner Blocks: The number of lock owner blocks currently in use on the server (refreshed periodically). A lock owner block represents the ownership of a lock on an object by an individual thread. Therefore, if three threads each have a shared (S) lock on a page, there will be three lock owner blocks.

Lock Owner Blocks Allocated: Current number of allocated lock owner blocks. At server startup, the number of allocated lock owner blocks and the number of allocated lock blocks depend on the SQL

Lock Requests/sec: The number of new locks and lock conversions per second requested from the lock manager.

Lock Timeouts (timeout > 0)/sec: The number of lock requests per second that timed out, but excluding requests for NOWAIT locks.

Lock Timeouts/sec: The number of lock requests per second that timed out, including requests for NOWAIT locks.

Lock Wait Time (ms): Total wait time (in milliseconds) for locks in the last second.

Lock Waits/sec: The number of lock requests per second that required the caller to wait.

Log Bytes Flushed/sec: Total number of log bytes flushed.

Log Cache Hit Ratio: Percentage of log cache reads satisfied from the log cache.

Log Cache Reads/sec: Reads performed per second through the log manager cache.

Log File(s) Size (KB): Cumulative size (in kilobytes) of all the transaction log files in the database.

Log File(s) Used Size (KB): The cumulative used size of all the log files in the database.

Log Flush Wait Time Total wait time (in milliseconds) to flush the log.

Log Flush Waits/sec: The number of commits per second waiting for the log flush.

Log Flushes/sec: The number of log flushes per second.

Log Growths: Total number of times the transaction log for the database has been expanded.

Log Shrinks: Total number of times the transaction log for the database has been shrunk.

Log Truncations: Total number of times the transaction log for the database has been truncated.

Logical Connections: The number of logical connections to the system.

Logins/sec: Total number of logins started per second. This does not include pooled connections.

Logouts/sec: Total number of logout operations started per second.

Longest Transaction Running Time: The length of time (in seconds) since the start of the transaction that has been active longer than any other current transaction.

Mars Deadlocks: The number of Mars Deadlocks detected.

Maximum Workspace Memory (KB): Maximum amount of memory available for executing processes such as hash, sort, bulk copy, and index creation operations.

Memory Grants Outstanding: Total number of processes that have successfully acquired a workspace memory grant.

Memory Grants Pending: Total number of processes waiting for a workspace memory grant.

Mixed page allocations/sec: The number of pages allocated per second from mixed extents. These could be used for storing the IAM pages and the first eight pages that are allocated to an allocation unit.

Non-atomic yield rate: The number of non-atomic yields per second.

NonSnapshot Version Transactions: The number of currently active transactions that are not using snapshot isolation level and have made data modifications that have generated row versions in the Number of active cursor plans.

Number of active cursor plans: Number of cursor plans.

Number of Deadlocks/sec: The number of lock requests per second that resulted in a deadlock.

Number of SuperLatches: The number of latches that are currently SuperLatches.

Optimizer Memory (KB): Total amount of dynamic memory the server is using for query optimization.

Page compression attempts/sec: The number of pages evaluated for page-level compression. Includes pages that were not compressed because significant savings could be achieved. Includes all objects in the instance of SQL Server. For information about specific objects, see sys. dm_db_index_operational_stats (Transact-SQL).

Page Deallocations/sec: The number of pages deallocated per second in all databases in this instance of SQL Server. These include pages from mixed extents and uniform extents.

Page life expectancy: The number of seconds a page will stay in the buffer pool without references.

Page lookups/sec: The number of requests per second to find a page in the buffer pool.

Page reads/sec: The number of physical database page reads that are issued per second. This statistic displays the total number of physical page reads across all databases. Because physical I/O is expensive, you may be able to minimize the cost, either by using a larger data cache, intelligent indexes, and more efficient queries or by changing the database design.

Page Splits/sec: The number of page splits per second that occur as the result of overflowing index pages.

Page writes/sec: The number of physical database page writes issued per second.

Pages Allocated/sec: The number of pages allocated per second in all databases in this instance of SQL Server. These include pages allocations from both mixed extents and uniform extents.

Pages compressed/sec: The number of data pages that are compressed by using PAGE compression. Includes all objects in the instance of SQL Server. For information about specific objects, see sys.dm_db_index_operational_stats (Transact-SQL).

Percent Log Used: Percentage of space in the log that is in use.

Probe Scans/sec: The number of probe scans per second that are used to find at most one single qualified row in an index or base table directly.

Processes blocked: The number of currently blocked processes.

Range Scans/sec: The number of qualified range scans through indexes per second.

Readahead pages/sec: The number of pages read per second in anticipation of use.

Repl. Trans. Rate: The number of transactions per second read out of the transaction log of the publication database and delivered to the distribution database.

Safe Auto-Params/sec: The number of safe auto-parameterization attempts per second. Safe refers to a determination that a cached execution plan can be shared between different similar-looking Transact-SQL statements. SQL Server makes many auto-parameterization attempts some of which turn out to be safe and others fail. Note that auto-parameterizations are also known as simple parameterizations in later versions of SQL Server. This does not include forced parameterizations.

Scan Point Revalidations/sec: The number of times per second that the scan point had to be revalidated to continue the scan.

Skipped Ghosted Records/sec: The number of ghosted records per second skipped during scans.

Snapshot Transactions: The number of currently active transactions using the snapshot isolation level. N.B. The SOAP Empty Requests Number of empty SOAP requests started per second.

SOAP Empty Requests: The number of SOAP method invocations passed to the stored procedure (or template) with an empty string as its value (not a NULL value) in order to provide an included input parameter with no value assigned to it.

SOAP Method Invocations: The number of SOAP method invocations started per second.

SOAP Session Initiate Requests: The number of SOAP Session initiate requests started per second.

SOAP Session Terminate Requests: The number of SOAP Session terminate requests started per second.

SOAP SQL Requests: The number of SOAP SQL requests started per second.

SOAP WSDL Requests: The number of SOAP Web Service Description Language requests started per second.

SQL Attention rate: The number of attentions per second. An attention is a request by the client to end the currently running request.

SQL Cache Memory (KB): Total amount of dynamic memory the server is using for the dynamic SQL cache.

SQL Compilations/sec: The number of SQL compilations per second. Indicates the number of times the compile code path is entered. Includes compiles caused by statement-level recompilations in SQL Server. After SQL Server user activity is stable, this value reaches a steady state.

SQL Re-Compilations/sec: The number of statement recompiles per second. Counts the number of times statement recompiles are triggered. Generally, you want the recompiles to be low. In later versions of SQL Server, recompilations are statement-scoped instead of batch-scoped recompilations in Microsoft SQL Server 2000. Therefore, direct comparison of values of this counter between SQL Server and earlier versions is not possible.

Stored Procedures Invoked/sec: This counter reports the total number of activation stored procedures invoked by all queue monitors in the instance per second.

SuperLatch Demotions/sec: The number of SuperLatches that have been demoted to regular latches in the last second.

SuperLatch Promotions/sec: The number of latches that have been promoted to SuperLatches in the last second.

Table Lock Escalations/sec: The number of times locks on a table were escalated to the TABLE or HoBT granularity. Target pages Ideal number of pages in the buffer pool.

Target pages: Ideal number of pages in the buffer pool.

Target Server Memory (KB): Total amount of dynamic memory the server can consume.

Task Limit Reached: The number of times that a queue monitor would have started a new task, but did not because the maximum number of tasks for the queue is already running.

Task Limit Reached/sec: The number of times per second that a queue monitor would have started a new task, but did not because the maximum number of tasks for the queue is already running.

Tasks Aborted/sec: The number of activation stored procedure tasks that end with an error or are aborted by a queue monitor for failing to receive messages.

Tasks Running: The number of activation stored procedures that are currently running.

Tasks Started/sec: The number of activation stored procedures started per second by all queue monitors in the instance.

Temp Tables Creation Rate: The number of temporary tables/table variables created per second.

Temp Tables For Destruction: The number of temporary tables/table variables waiting to be destroyed by the cleanup system thread.

Total Latch Wait Time (ms): Total latch wait time (in milliseconds) for latch requests in the last second.

TOTAL_LOCK_TIME: The total time for which database is in locked state.

Total Server Memory (KB): The committed memory from the buffer pool (in kilobytes).

Transactions/sec: The number of transactions started for the database per second.

Unsafe Auto-Params/sec: The number of unsafe auto-parameterization attempts per second. For example, the query has some characteristics that prevent the cached plan from being shared. These are designated as unsafe. This does not count the number of forced parameterizations.

Update conflict ratio: The percentage of those transactions using the snapshot isolation level that have encountered update conflicts within the last second. An update conflict occurs when a snapshot isolation level transaction attempts to modify a row that last was modified by another transaction that was not committed when the snapshot isolation level transaction started.

Update Snapshot Transactions: The number of currently active transactions using the snapshot isolation level and have modified data.

Used leaf page cookie: The number of times a leaf page cookie is used successfully during an index search since no change happened on the leaf page. The cookie is used to speed up the index search.

Used tree page cookie: The number of times a tree page cookie is used successfully during an index search since no change happened on the parent page of the tree page. The cookie is used to speed up the index search.

User Connections: The number of users currently connected to SQL Server.

Version Cleanup rate (KB/s): The rate (in kilobytes per second) at which row versions are removed from the snapshot isolation version store.

Version Generation rate (KB/s): The rate (in kilobytes per second) at which new row versions are added to the snapshot isolation version store.

Version Store Size (KB): The amount of space (in kilobytes) in Version Store unit count The number of active allocation units in the snapshot isolation version store.

Version Store unit creation: The number of allocation units that have been created in the snapshot isolation store since the instance of the Database Engine was started.

Version Store unit truncation: The number of allocation units that have been removed from the snapshot isolation store since the instance of the Database Engine was started.

Workfiles Created/sec: The number of work files created per second. For example, work files could be used to store temporary results for hash joins and hash aggregates.

Worktables Created/sec: The number of work tables created per second. For example, work tables could be used to store temporary results for query spool, lob variables, XML variables, and cursors.

Worktables From Cache Ratio: Percentage of work tables created where the initial two pages of the work table were not allocated but were immediately available from the work table cache. (When a work table is dropped, two pages may remain allocated and they are returned to the work table cache. This increases performance.)

Write Transactions/sec: The number of transactions that wrote to the database and committed, in the last second.

Microsoft SQL Azure Metrics

avg_cpu_percent: Average compute utilization in the percentage of the limit of the service tier.

avg_data_io_percent: Average data I/O utilization in percentage based on the limit of the service tier.

avg_log_write_percent: Average write resource utilization in the percentage of the limit of the service tier.

avg_memory_percent: Average memory utilization in the percentage of the limit of the service tier.

MySQL Server Metrics

Aborted_clients: The number of clients that were aborted (because they did not properly close the connection to the MySQL server). For some applications, this can be OK, but for some other applications you might want to track the value, as aborted connects may indicate some sort of application failure.

Aborted_connects: The number of failed attempts to connect to the MySQL server.

Bytes_received: The number of bytes received from all clients.

Bytes_sent: The number of bytes sent to all clients.

Com_alter_table: The number of times each ALTERTABLE statement has been executed.

Com_create_index: The number of times each CREATE INDEX statement has been executed.

Com_create_table: The number of times each CREATE TABLE statement has been executed.

Com_delete: The number of times each DELETE statement has been executed.

Com_insert: The number of times each INSERT statement has been executed.

Com_optimize: The number of times each OPTIMIZE statement has been executed.

Com_select: The number of times each SELECT statement has been executed.

Com_update: The number of times each UPDATE statement has been executed.

Connections: The number of connection attempts (successful or not) to the MySQL server.

Created_tmp_disk_tables: The number of temporary tables on disk created automatically by the server while executing statements.

Created_tmp_files: How many temporary files mysqld has created.

Created_tmp_tables: The number of in-memory temporary tables created automatically by the server while executing statements. If Created_tmp_disk_tables is large, you may want to increase the tmp_table_size value to cause temporary tables to be memory-based instead of disk-based.

Handler_delete: The number of times that rows have been deleted from tables.

Innodb_buffer_pool_pages_data: The number of pages containing data (dirty or clean).

Innodb_buffer_pool_pages_dirty: The number of pages currently dirty.

Innodb_buffer_pool_pages_flushed: The number of buffer pool page-flush requests.

Innodb_buffer_pool_pages_free: The number of free pages.

Innodb_buffer_pool_pages_misc: The number of pages that are busy because they have been allocated for administrative overhead such as row locks or the adaptive hash index. This value can also be calculated as Innodb_buffer_pool_pages_total.

Innodb_buffer_pool_pages_total: The total size of the buffer pool, in pages.

Innodb_buffer_pool_read_ahead_rnd: The number of random read-aheads initiated by InnoDB. This happens when a query scans a large portion of a table but in random order.

Innodb_buffer_pool_read_requests: The number of logical read requests InnoDB has done.

Innodb_buffer_pool_reads: The number of logical reads that InnoDB could not satisfy from the buffer pool and had to do a single-page read.

Innodb_buffer_pool_wait_free: Normally, writes to the InnoDB buffer pool happen in the background. However, if it is necessary to read or create a page and no clean pages are available, it is also necessary to wait for pages to be flushed first. This counter counts instances of these waits. If the buffer pool size has been set properly, this value should be small.

Innodb_buffer_pool_write_requests: The number writes done to the InnoDB buffer pool.

Innodb_data_fsyncs: The number of fsync() operations so far.

Innodb_data_pending_fsyncs: The current number of pending fsync() operations.

Innodb_data_pending_reads: The current number of pending reads.

Innodb_data_pending_writes: The current number of pending writes.

Innodb_data_read: The amount of data read so far, in bytes.

Innodb_data_reads: The total number of data reads.

Innodb_data_writes: The total number of data writes.

Innodb_data_written: The amount of data written so far, in bytes.

Innodb_dblwr_pages_written: The number of doublewrite operations that have been performed.

Innodb_dblwr_writes: The number of pages that have been written for doublewrite operations.

Innodb_log_waits: The number of times that the log buffer was too small and a wait was required for it to be flushed before continuing.

Innodb_log_write_requests: The number of log write requests.

Innodb_log_writes: The number of physical writes to the log file.

Innodb_pages_created: The number of pages created.

Innodb_pages_read: The number of pages read.

Innodb_pages_written: The number of pages written.

Innodb_row_lock_current_waits: The number of row locks currently being waited for.

Innodb_row_lock_time: The total time spent in acquiring row locks, in milliseconds.

Innodb_row_lock_time_avg: The average time to acquire a row lock, in milliseconds.

Innodb_row_lock_time_max: The maximum time to acquire a row lock, in milliseconds.

Innodb_row_lock_waits: The number of times a row lock had to be waited for.

Innodb_rows_deleted: The number of rows deleted from InnoDB tables.

Innodb_rows_inserted: The number of rows inserted into InnoDB tables.

Innodb_rows_read: The number of rows read from InnoDB tables.

Innodb_rows_updated: The number of rows updated in InnoDB tables.

Key_blocks_used: The number of used blocks in the key cache. This value is a high-water mark that indicates the maximum number of blocks that have ever been in use at one time.

Key_read_requests: The number of requests to read a key block from the cache. Key_writes The number of physical writes of a key block to disk.

Key_reads: The number of physical reads of a key block from disk. If Key_reads is large, then your key_buffer_size value is probably too small. The cache miss rate can be calculated as Key_reads/Key_read_requests.

Key_write_requests: The number of requests to write a key block to the cache.

Key_writes: The number of physical writes of a key block from the `MyISAM` key cache to the physical disk.

Open_files: The number of files that are open. Open_streams The number of streams that are open (used mainly for logging).

Open_tables: The number of table cache misses. If the value is large, you probably need to increase table_cache. Typically you would want this to be less than 1 or 2 opened tables per second.

Opened_tables: The number of tables that have been opened. The number of tables that have been opened. If Opened_tables is big, your table_cache value is probably too small.

Qcache_free_blocks: The number of free memory blocks in the query cache.

Qcache_free_memory: The amount of free memory for the query cache.

Qcache_hits: The number of query cache hits.

Qcache_inserts: The number of queries added to the query cache.

Qcache_lowmem_prunes: The number of queries that were deleted from the query cache because of low memory.

Qcache_not_cached: The number of non-cached queries (not cacheable, or not cached due to the query_cache_type setting).

Qcache_queries_in_cache: The number of queries registered in the query cache.

Qcache_total_blocks: The total number of blocks in the query cache.

Questions: The number of statements that clients have sent to the server.

Seconds_Behind_Master: The number of seconds that the replica SQL thread is behind processing the primary binary log. This field directly drives MTTR. A high or increased number of Seconds_Behind_Master indicates a slow disk or poorly tuned machine running the database. If this value reaches 45 minutes or more, the ability to recover from a crash, or after a manual failover, is delayed. If this value is tracked by the Machine Agent through the MySQL extension and shows a generally positive slope, it indicates a death spiral.

Select_full_join: Joins performed without keys. This should be zero. This is a good way to catch development errors, as just a few such queries can decrease the system's performance.

Select_full_range_join: The number of joins that used a range search on a reference table.

Select_range: The number of joins that used ranges on the first table. This is normally not a critical issue even if the value is quite large.

Select_range_check: The number of joins without keys that check for key usage after each row. If this is not 0, you should carefully check the indexes of your tables.

Select_scan: Number of queries that performed a full table scan. In some cases these are OK but their ratio to all queries should be constant. if you have the value growing it can be a problem with the optimizer, lack of indexes or some other problem.

Slave_IO_Running: Whether the I/O thread for reading the primary's binary log is running.

Slave_open_temp_tables: The number of temporary tables that the replica SQL thread currently has open.

Slave_SQL_Running: Whether the SQL thread for executing events in the relay log is running.

Slow_launch_threads: The number of threads that have taken more than slow_launch_time seconds to create.

Slow_queries: The number of queries longer than --long-query-time or that are not using indexes. These should be a small fraction of all queries. If it grows, the system will have performance problems.

Sort_merge_passes: The number of merge passes that the sort algorithm has had to do. If this value is large, you should consider increasing the value of the sort_buffer_size system variable.

Sort_range: The number of sorts that were done using ranges.

SQL_Delay: The number of seconds that the replica lags behind the primary

Threads_cached: The number of threads in the thread cache.

Threads_connected: The number of currently open connections.

Threads_created: This should be low. Higher values may mean that you need to increase the value of thread_cache or you have the number of connections increasing, which also indicates a potential problem.

Threads_running: The number of threads that are not sleeping.

Oracle Server Metrics

bytes received via SQL*Net from client: Total number of bytes received from the client over Oracle Net Services

bytes received via SQL*Net from dblink: Total number of bytes received from a database link over Oracle Net Services

bytes sent via SQL*Net to client: Total number of bytes sent to the client from the foreground processes

bytes sent via SQL*Net to dblink: Total number of bytes sent over a database link

CPU used by this session: Amount of CPU time (in 10s of milliseconds) used by a session from the time a user call starts until it ends. If a user call completes within 10 milliseconds, the start and end user-call time are the same for purposes of this statistics, and 0 milliseconds are added.

db block changes: Closely related to consistent changes, this statistic counts the total number of changes that were part of an update or delete operation that were made to all blocks in the SGA. Such changes generate redo log entries and hence become permanent changes to the database if the transaction is committed.This approximates total database work. It statistic indicates the rate at which buffers are being dirtied (on a per-transaction or per-second basis, for example).

db block gets: The number of times a CURRENT block was requested. See Also: consistent gets

db block gets from cache: The number of times a CURRENT block was requested from the buffer cache. This is a subset of db block gets statistics value.

DBWR checkpoint buffers written: The number of buffers that were written for checkpoints

DBWR checkpoints: The number of times the DBWR was asked to scan the cache and write all blocks marked for a checkpoint or the end of recovery. This statistic is always larger than background checkpoints completed.

DML statements parallelized: The number of DML statements that were executed in parallel.

enqueue conversions: Total number of conversions of the state of table or row lock.

enqueue deadlocks: Total number of deadlocks between table or row locks in different sessions.

enqueue releases: Total number of table or row locks released.

enqueue requests: Total number of table or row locks acquired.

enqueue timeouts: Total number of table and row locks (acquired and converted) that timed out before they could complete.

enqueue waits: Total number of waits that occurred during an enqueue convert or get because the enqueue get was deferred

exchange deadlocks: The number of times that a process detected a potential deadlock when exchanging two buffers and raised an internal, restartable error. Index scans are the only operations that perform exchanges.

execute count: Total number of calls (user and recursive) that executed SQL statements.

gc current block receive time: The total time required for consistent read requests to complete. It records the round-trip time for all requests for consistent read blocks.

index fast full scans (direct read): The number of fast full scans initiated using direct read.

index fast full scans (full): The number of fast full scans initiated for full segments.

index fast full scans (rowid ranges): The number of fast full scans initiated with rowid endpoints specified.

lob reads: The number of LOB API read operations performed in the session/system. A single LOB API read may correspond to multiple physical/logical disk block reads.

lob writes: The number of LOB API write operations performed in the session/system. A single LOB API write may correspond to multiple physical/logical disk block writes.

logons current: Total number of current logons. Useful only in V$SYSSTAT.

Number of Small Reads: The total number of **physical reads** less the number of **physical read total multi block requests**.

Number of Small Writes: The total number of **physical writes** less the number of **physical write total multi block requests**.

opened cursors current: Total number of current open cursors.

Parallel operations not downgraded: The number of times parallel execution was executed at the requested degree of parallelism.

parse count (hard): Total number of parse calls (real parses). A hard parse is a very expensive operation in terms of memory use, because it requires Oracle to allocate a workheap and other memory structures and then build a parse tree.

parse count (total): Total number of parse calls (hard and soft). A soft parse is a check on an object already in the shared pool, to verify that the permissions on the underlying object have not changed.

parse time cpu: Total CPU time used for parsing (hard and soft) in 10s of milliseconds.

parse time elapsed: Total elapsed time for parsing, in 10s of milliseconds. Subtract parse time cpu from this statistic to determine the total waiting time for parse resources.

physical read bytes: Total size in bytes of all disk reads by application activity (and not other instance activity) only.

physical read IO requests: The number of read requests for application activity (mainly buffer cache and direct load operation) which read one or more database blocks per request. This is a subset of physical read total IO requests statistic.

physical read total bytes: Total size in bytes of disk reads by all database instance activity including application reads, backup and recovery, and other utilities. The difference between this value and physical read bytes gives the total read size in bytes by non-application workload.

physical read total IO requests: The number of read requests which read one or more database blocks for all instance activity including application, backup and recovery, and other utilities. The difference between this value and physical read total multi block requests gives the total number of single block read requests.

physical reads: Total number of data blocks read from disk. This value can be greater than the value of physical reads direct plus physical reads cache as reads into process private buffers also included in this statistic.

physical write bytes: Total size in bytes of all disk writes from the database application activity (and not other kinds of instance activity).

physical write IO requests: The number of write requests for application activity (mainly buffer cache and direct load operation) which wrote one or more database blocks per request.

physical write total bytes: Total size in bytes of all disk writes for the database instance including application activity, backup and recovery, and other utilities. The difference between this value and physical write bytes gives the total write size in bytes by non-application workload.

physical write total IO requests: The number of write requests which wrote one or more database blocks from all instance activity including application activity, backup and recovery, and other utilities. The difference between this stat and physical write total multi block requests gives the number of single block write requests.

physical writes: Total number of data blocks written to disk. This statistics value equals the sum of physical writes direct and physical writes from cache values.

physical writes direct: The number of writes directly to disk, bypassing the buffer cache (as in a direct load operation).

physical writes from cache: Total number of data blocks written to disk from the buffer cache. This is a subset of physical writes statistic.

process last non-idle time: The last time this process executed.

queries parallelized: The number of SELECT statements executed in parallel.

Read Percent:

recovery blocks read: The number of blocks read during recovery.

recursive calls: The number of recursive calls generated at both the user and system level. Oracle maintains tables used for internal processing. When Oracle needs to make a change to these tables, it internally generates an internal SQL statement, which in turn generates a recursive call.

recursive cpu usage: Total CPU time used by non-user calls (recursive calls). Subtract this value from CPU used by this session to determine how much CPU time was used by the user calls.

redo blocks written: This is the total number of redo blocks written. This statistic divided by "redo writes" equals the number of blocks per write.

redo buffer allocation retries: Total number of retries necessary to allocate space in the redo buffer. Retries are needed either because the redo writer has fallen behind or because an event such as a log switch is occurring.

redo log space requests: The number of times the active log file is full and Oracle must wait for disk space to be allocated for the redo log entries. Such space is created by performing a log switch.

redo log space wait time: Total elapsed waiting time for redo log space requests in 10s of milliseconds.

redo size: Total amount of redo generated in bytes.

redo synch time: Elapsed time of all redo synch writes calls in 10s of milliseconds

redo write time: Total elapsed time of the write from the redo log buffer to the current redo log file in microseconds.

redo writes: Total number of writes by LGWR to the redo log files. redo blocks written divided by this statistic equals the number of blocks per write.

session cursor cache count: Total number of cursors cached. This statistic is incremented only if SESSION_CACHED_CURSORS > 0. This statistic is the most useful in V$SESSTAT. If the value for this statistic in V$SESSTAT is close to the setting of the SESSION_CACHED_CURSORS parameter, the value of the parameter should be increased.

session cursor cache hits: The number of hits in the session cursor cache. A hit means that the SQL statement did not have to be reparsed. Subtract this statistic from parse count (total) to determine the real number of parses that occurred.

session logical reads: The sum of db block gets plus consistent gets. This includes logical reads of database blocks from either the buffer cache or process private memory.

session stored procedure space: Amount of memory this session is using for stored procedures.

Small IO Percent:

sorts (disk): The number of sort operations that required at least one disk write.

sorts (memory): The number of sort operations that were performed completely in memory and did not require any disk writes.

sorts (rows): Total number of rows sorted.

table scan rows gotten: The number of rows that are processed during scanning operations.

table scans (direct read): The number of table scans performed with direct read (bypassing the buffer cache).

table scans (long tables): Long (or conversely short) tables can be defined as tables that do not meet the short table criteria as described in table scans (short tables).

table scans (rowid ranges): During a parallel query, the number of table scans conducted with specified ROWID ranges.

transaction lock background get time: Useful only for internal debugging purposes.

transaction lock foreground wait time: Useful only for internal debugging purposes.

transaction rollbacks: The number of transactions being successfully rolled back.

TOTAL_LOCK_TIME: The total time for which database is in locked state.

user calls: The number of user calls such as login, parse, fetch, or execute

user commits: The number of user commits. When a user commits a transaction, the redo generated that reflects the changes made to database blocks must be written to disk. Commits often represent the closest thing to a user transaction rate.

user rollbacks: The number of times users manually issue the ROLLBACK statement or an error occurs during a user's transactions.

Some of the obsolete metrics were removed and the following metrics are included with 20.11 and later versions of Controller.

DB|Server Statistic|DB time: The sum of CPU consumption of all the Oracle process and the sum of non-idle wait time

DB|Server Statistic|OS System time used: The total CPU time used for system calls

DB|Server Statistic|OS User time used: The total CPU time used for user calls

DB|Server Statistic|SMON posted for instance recovery: The total count or number of times SMON posted for instance recovery

DB|Server Statistic|SMON posted for txn recovery for other instances: The total count or number of times SMON posted for other transactions recovery in an Oracle RAC or a cluster

DB|Server Statistic|java call heap live size: The Java call heap live size

DB|Server Statistic|java call heap total size: The total Java call heap size

DB|Server Statistic|java call heap used size: The Java call heap used size

DB|Server Statistic|non-idle wait count: The total count of non-idle wait events

DB|Server Statistic|non-idle wait time: The total elapsed time for non-idle events

PostgreSQL Server Metrics

blks_hit: Number of times disk blocks were found already in the buffer cache, so that a physical disk read was not necessary. This only includes hits in the PostgreSQL buffer cache, does not include the operating system's file system .cache).

blks_read: Number of disk blocks read from the database.

confl_bufferpin: Number of queries in the database that were canceled because of pinned buffers.

confl_deadlock: Number of queries in the database that were canceled because of deadlocks

confl_lock: Number of queries in the database that were canceled due because of timeouts.

confl_snapshot: Number of queries in the database that were canceled because of old snapshots.

confl_tablespace: Number of queries in the database that were canceled because of dropped tablespaces.

numbackends: Number of backends currently connected to the database.

size_mb:

tup_deleted: Number of rows deleted by queries in the database.

tup_fetched: Number of rows fetched by queries in the database.

tup_inserted: Number of rows inserted by queries in the database.

tup_returned: Number of rows returned by queries in the database.

tup_updated: Number of rows updated by queries in the database.

xact_commit: Number of transactions in the database that have been committed.

xact_rollback: Number of transactions in the database that have been rolled back.

Sybase ASE and IQ Server Metrics

active_connections: The number of users currently connected to the database

ActiveReq: Returns the number of server threads that are currently handling a request.

ActiveVersionsCount: Count of Active Transaction Versions.

ActiveVersionsCreateMB: Size in Mb of versions for active transaction.

ActiveVersionsDeleteMB: Size in Mb of versions for active transactions.

AIOs_delayed_due_to_engine_limit:

AIOs_delayed_due_to_os_limit:

AIOs_delayed_due_to_server_limit:

BytesReceived: Returns the number of bytes received during client/server communications. This value is updated for HTTP and HTTPS connections.

BytesReceivedUncomp: Returns the number of bytes that would have been received during client/server communications if compression was disabled. (This value is the same as the value for BytesReceived if compression is disabled.)

BytesSent: Returns the number of bytes sent during client/server communications. This value is updated for HTTP and HTTPS connections.

BytesSentUncomp: Returns the number of bytes that would have been sent during client/server communications if compression was disabled. (This value is the same as the value for BytesSent if compression is disabled.)

CacheFileDirty: Returns the number of cache pages that are dirty (needing a write).

CacheFree: Returns the number of cache pages not being used.

CacheHits: Returns the number of database page lookups.

CachePanics: Returns the number of times the cache manager has failed to find a page to allocate.

CachePinned: Returns the number of pinned cache pages.

CacheRead: Returns the number of cache reads.

CacheReplacements: Returns the number of pages in the cache that have been replaced.

CacheScavenges: Returns the number of times the cache manager has scavenged for a page to allocate.

DiskRead: Returns the number of disk reads.

mempages_alloced: The number of memory pages that have been allocated.

mempages_freed: The number of memory pages that have been de-allocated.

OtherVersionsCount: Shows count of other db versions. These versions will eventually be dropped when they are no longer referenced or referencable by active transactions.

OtherVersionsMB: Shows space usage in MB of other db versions. These versions will eventually be dropped when they are no longer referenced or referencable by active transactions.

ProcessCPU: Returns CPU usage for the database server process. Values are in seconds. This property is supported on Windows and Unix. This property is not supported on Windows Mobile.

ProcessCPUSystem: Returns system CPU usage for the database server process CPU. This is the amount of CPU time that the database server spent inside the operating system kernel. Values are in seconds. This property is supported on Windows and Unix. This property is not supported on Windows Mobile.

ProcessCPUUser: Returns user CPU usage for the database server process. Values are in seconds. This excludes the amount of CPU time that the database server spent inside the operating system kernel. This property is supported on Windows and Unix. This property is not supported on Windows Mobile.

RequestsReceived: Requests received by the server.

TempBufferCapacityCount: Capacity count of Temporary Buffers.

TempBufferCapacityMB: Capacity in Mb of Temporary Buffers.

TempBufferLockedCount: The number of Temporary Buffers that are locked.

TempBufferUsedCount: The number of Temporary Buffers in use.

total_bytes_received: The number of bytes received during client/server communications. This value is updated for HTTP and HTTPS connections.

total_bytes_sent: Returns the number of bytes sent during client/server communications. This value is updated for HTTP and HTTPS connections.

TOTAL_LOCK_TIME: The total time for which database is in locked state.

xacts: The number of transactions.

yields: Returns the number of times the Adaptive Server engine(s) yielded to the operating system.

Database Agent Events Reference

You can view Database Agent events on the **Database Monitoring > Events** window and in the `<agent_home>/logs/db-agent#.log` files. Database Agent events are of type Agent Diagnostic Event, with case-specific messages attached. See Monitor Events.

Agent Diagnostic Events

Initialize/re-initialize the DB Collector

Summary: Initialize/re-initialize the DB collector

Description: Sent when database collector has successfully started. This happens when you configure a new collector, start/restart the agent process (in which case you'll see this event once for each collector), or when the collector recovers from a failure to communicate with the monitored database (for example, when the database goes down for a while, then comes back).

Severity: Info

Initialize/re-initialize the Hardware Metric Collector

Summary: Initialize/re-initialize the hardware metric collector

Description: Same as the above, but refers to a hardware collector, not a database collector.

This is sent when hardware collector has successfully started. This happens when you configure a new collector, start/restart the agent process (in which case you'll see this event once for each collector), or when the collector recovers from a failure to communicate with the monitored hardware (for example, when the system goes down for a while, then comes back).

Severity: Info

Server Parameter [?] has been Changed from [?] to [?]

Summary: Server parameter [?] has been changed from [?] to [?]

Description: The Database Agent constantly monitors database configuration parameters. When a parameter changes, which generally means a database administrator made a deliberate change, the agent sends this event. AppDynamics Database Visibility users can use these events to track configuration changes in their databases.

Severity: Info

JDBC Driver-related Agent Diagnostic Events

Summary: The text of the message is the error from the vendor's JDBC driver.

Description: When the agent fails to collect data from a monitored database, it sends this event, and goes into sleep mode for one minute. It will then retry contacting the database every minute until it succeeds. Once it succeeds, it will send an Agent Event "initialize/re-initialize" message, as described above. This happens when the database goes down, or when you add a collector with a wrong password, or wrong hostname, etc.

When the Collector is in sleep mode because of an error, you'll see an indicator next to the database on the **Infrastructure** and **Infrastructure > Databases**, on **Configure > Collectors** panel. You will also see the full text of the error in the **Infrastructure > Events** panel.

Severity: Error

Wait State Filtering

Database wait states are pauses or delays in database activity. While Database Visibility collects data on all wait states, you may be more interested in some wait states than others. You can specify which wait states that you want to exclude from being reported by the agent so that you can focus on the critical wait states.

To specify wait states to exclude, navigate to **Configuration** > **Wait State Filtering**. Check the boxes of the wait states that you want to exclude. The wait states you exclude are no longer displayed in the Top 10 Query Wait States metric. In addition, any future queries that experience the wait states you exclude are not reported by the agent.

Configure Custom Metrics

Database Visibility extends monitoring by specifying SQL queries to run on the monitored database and the queries run during normal database activity. You can schedule these custom queries to run on regular intervals and collect the results in a custom metric. See Database Custom Metrics Window.

To specify custom queries to run:

1. Navigate to **Configuration > Custom Metrics > New**.
2. Complete the following fields:
    a. **Name**: The name you want to name your custom metric. Once you create a custom metric/event, you cannot change its name.
    b. **Custom Query Metric Type**: The type of result that you want from the query.
        i. Select the **Metric** type to generate metrics for which health rules can be defined. Metrics of this type are displayed in the **Metric Browser** or in the Custom Metrics tab of each collector.
        ii. Select the **Event Data** type to generate custom events whenever the custom query output is non-empty. If you want to be notified when a custom event occurs, you can create a policy that is triggered by that custom event. Custom events are displayed in the Events tab. If the custom query outputs multiple rows, the first 40 rows will be included in the event details. Each row displays a maximum of 5000 characters, including the column separators (|).
    c. **Database Type**: The database platform that you want to run the metric on.
    d. **Databases**: The database instances that you want to run the metric on. You can run the metric on all database instances of the specified database type, or on specific database instances that you specify.
    e. **Schedule**: The time interval at which you want to run the query. For metric type custom queries, the metric value reported in between the specified intervals is the value observed in the previous query execution.
    The timeout for configured custom metric depends on the frequency of running the query. This table provides the timeout details based on the time interval:

Interval (minutes)	Timeout (seconds)
1	10
5	15
10	30
30	60
60	120
360	180
720	300
1440	300

    f. **Query Text**: The query that you want to execute. If you are creating a custom metric of Metric type, its query must have one of the following return types:
        i. Positive integer. For example, the query below returns a positive integer.

```
SELECT COUNT(*) FROM employees
```

        ii. String and positive integer. For example, the query below returns a string and a positive integer.

```
SELECT name, salary FROM employees
```

Custom queries appear for all collectors, but the data only reflects the collector that you created it for. You can test the semantics of the query and also validate the results by clicking the **Test Query** button.

Custom query results are validated if the Database Agent >= 4.5.5.

The following points provide information about using limits when configuring custom metrics:

- The total number of custom metrics or events that you can create per account is the product of `total number of Collectors` and `dbmon.config.max.custommetric`.
  By default, the value of `dbmon.config.max.custommetric` is 40. You can adjust the limit up to 150 using the `dbmon.config.max.custommetric` property in the Controller Settings.
- Up to 150 custom metrics can be reported per collector per minute.
- You can increase the custom metrics limit from the agent configuration properties by using the key, `dbagent.custom.metric.reportable.per.server` and the `Max` value 150. See Database Agent Configuration Properties for information about the key value.
- The maximum limit configured in the agent takes precedence and overrides the maximum limit configured at the account level and Controller level. Therefore, the priority order is as following:

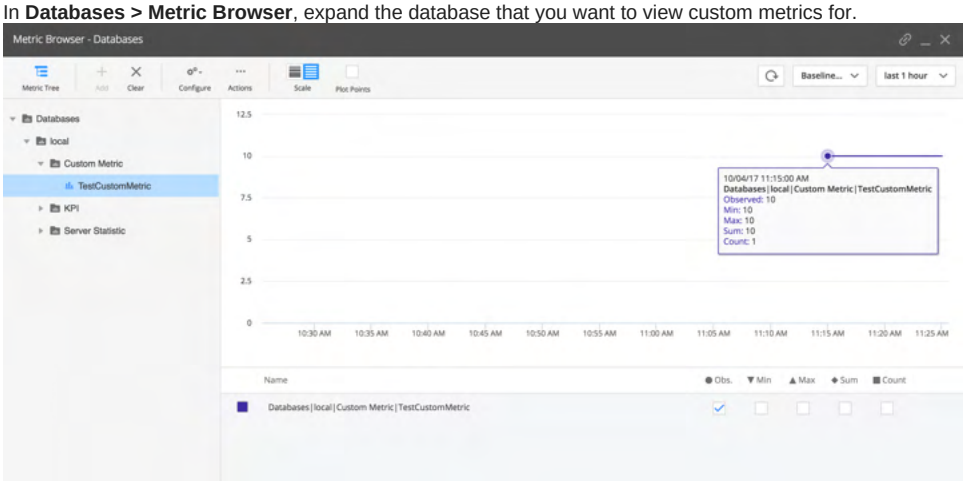
- Agent-level configuration
- Account-level configuration
- Controller-level configuration

Custom metrics are supported for all relational databases: MySQL, Microsoft SQL Server, Oracle, PostgreSQL, DB2, and Sybase.

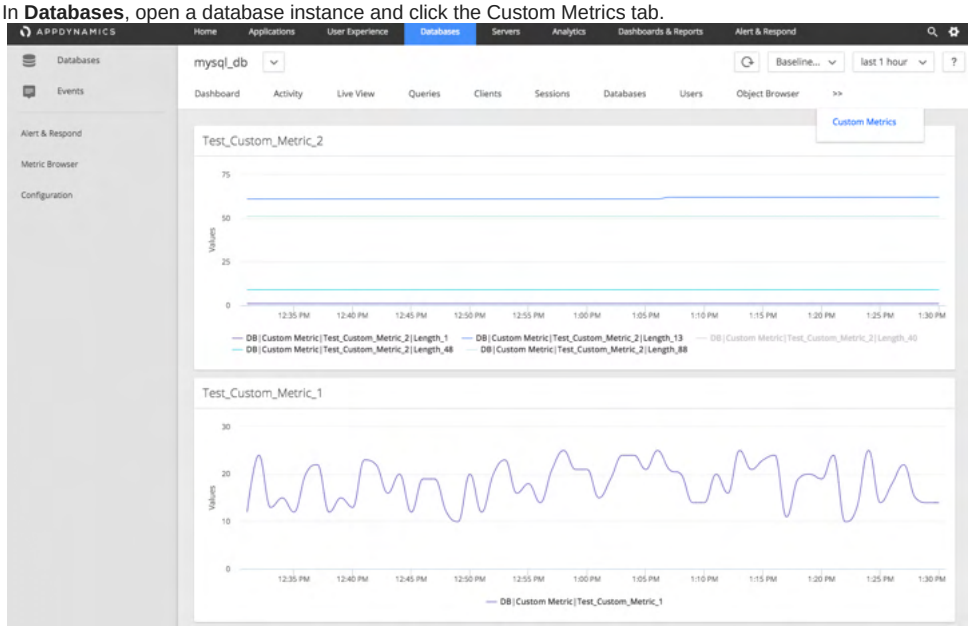
View Custom Metrics

You can view your custom metrics in the following locations:

- In **Databases > Metric Browser**, expand the database that you want to view custom metrics for.



- In **Databases**, open a database instance and click the Custom Metrics tab.



Configure Query Literals Security

In SQL, literals are the string or integer values representing data retrieved from the database. By default, literals are removed from SQL queries, as they can contain sensitive user data such as social security numbers, CC numbers, and so on.

To show literals in the queries, you can disable the **Remove literals from the queries** option.

1. Click **Databases**.
2. Click **Configuration**.
3. In the Security section, uncheck **Remove literals from the queries**.

Once you uncheck the option, literals will always be displayed in the queries. You can revert to hiding the literals by checking the option.

To record the audit logs when this option is enabled or disabled, ensure to configure the Controller audit report with the object name as **Remove Literals** and object type as **DBMON_ACCOUNT_CONFIGURATION**. See Controller Audit Log.

Integrate Database Visibility with Server Visibility

You can integrate Database Visibility with Server Visibility to capture accurate hardware metric details from the Machine Agent (with Server Visibility) to the Database Visibility dashboard. These metrics display without the need for an SSH connection to the host machine. Additionally, this integration helps the Server Visibility machine to automatically connect to the Database Visibility server when both the Database Agent and the Server Visibility Agent capture the same IP address.

When you integrate Server Visibility with Database Visibility, you can view these metrics on the Database Dashboard:

- **CPU Busy%** instead of CPU System% and CPU User%
- **Memory Used%**
- **Disk IO** (in Ops/sec for Reads and Writes) instead of Disk IO in KB/sec
- **Network IO** (in KB/sec for Outgoing and Incoming)

Requirements

To view the hardware metrics from Server Visibility to the Database Visibility dashboard, you must ensure to meet the following requirements:

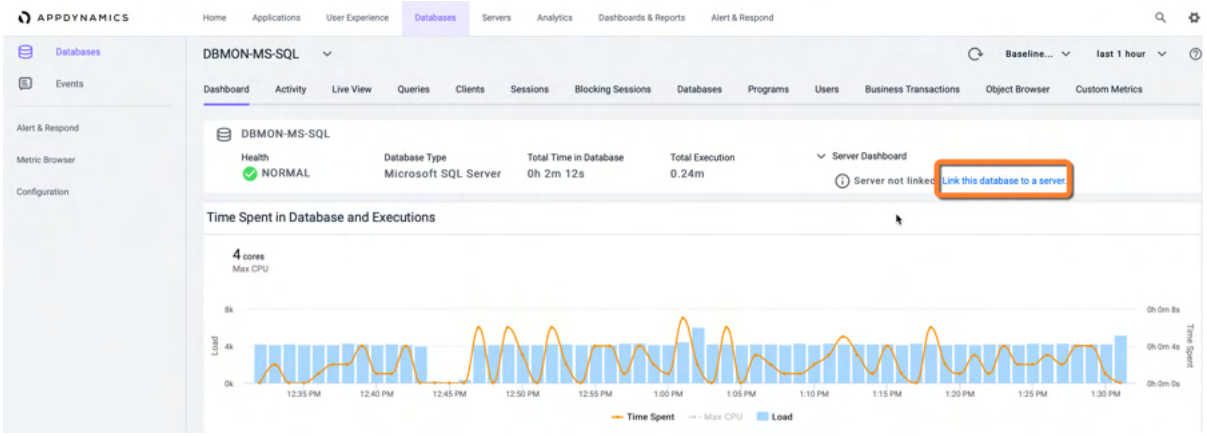
- Install and register Machine Agent.
- Enable Server Visibility on the Machine Agent. See Server Visibility.
- Disable the Hardware Monitoring section for the database collector. If you Configure the Database Agent to Monitor Server Hardware, then the Database Agent displays the hardware metrics based on the collector configuration using the SSH connection.

View Hardware Metrics

You can view the Server Dashboard section on the Databases dashboard when the Machine Agent and the Database Agents are registered and connected.

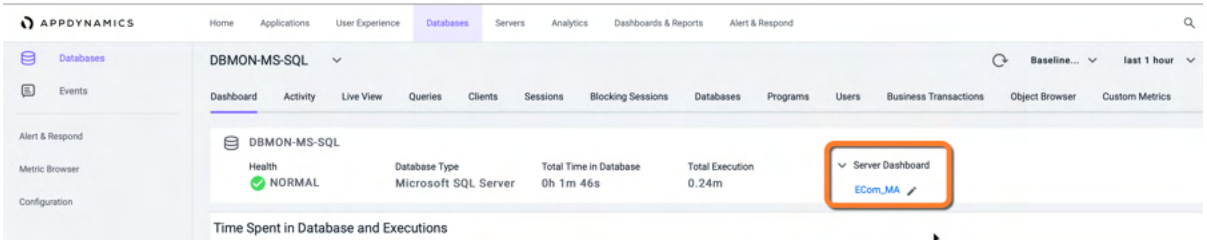
You can manually connect or disconnect the Database Visibility Server with the Server Visibility Server by using the URL, **Link this database to a server**, described in the Server Dashboard section. You may require to manually connect the database node to the Server Visibility server in these scenarios:

- The IP addresses do not match and the servers are not connected automatically.
- You want to connect the database server to another Server Visibility machine.
- You do not require connecting a particular database server to a Server Visibility machine.

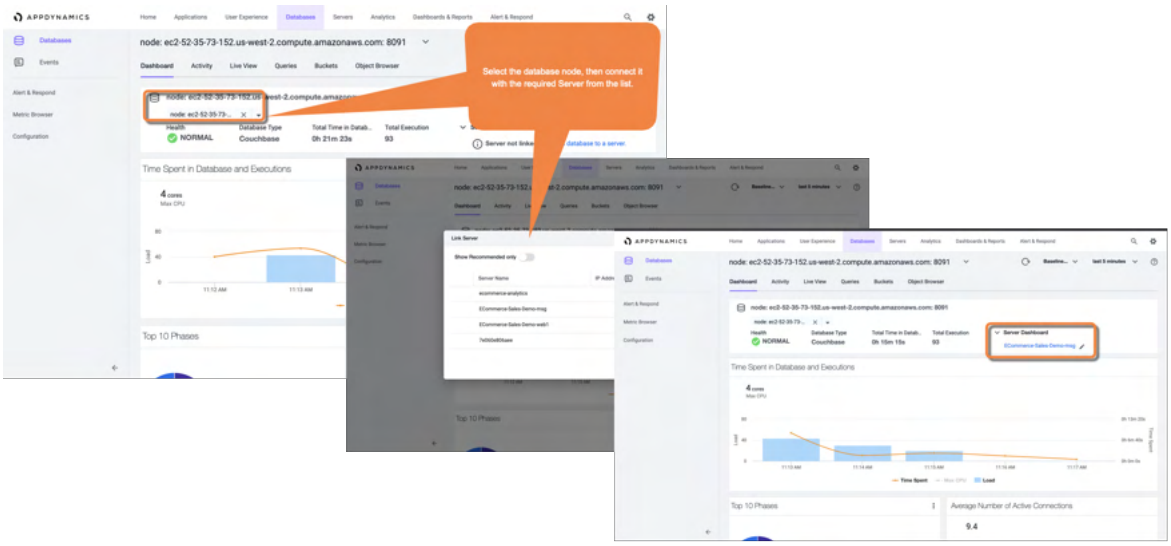


You can map multiple database nodes to a single Server Visibility machine.

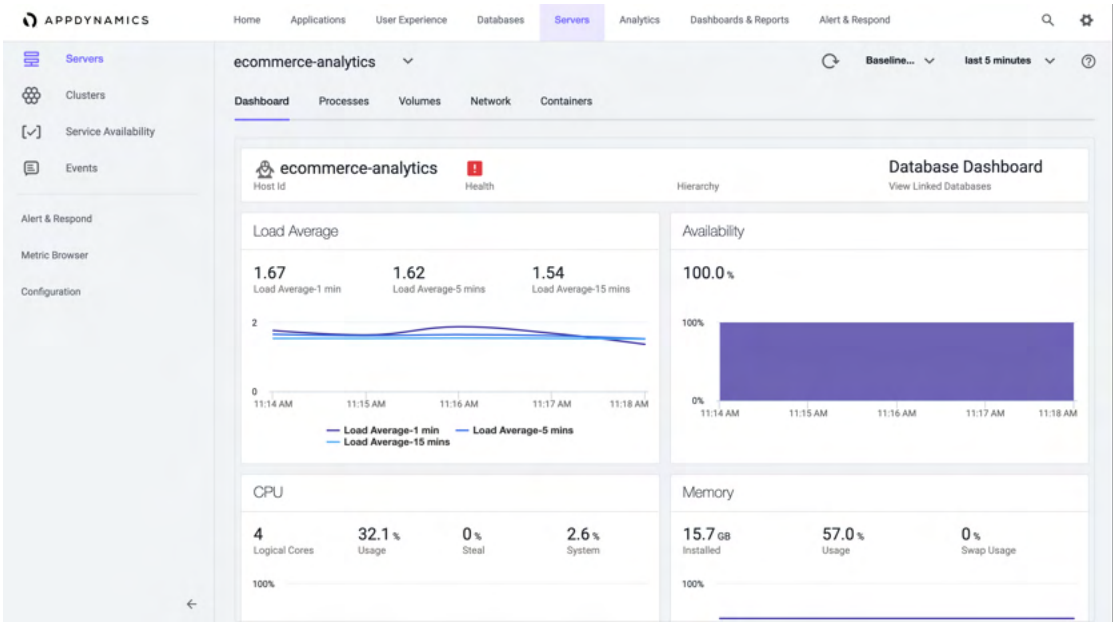
If the servers are connected, then you can find the link to the connected Server Dashboard under the **Server Dashboard** section.



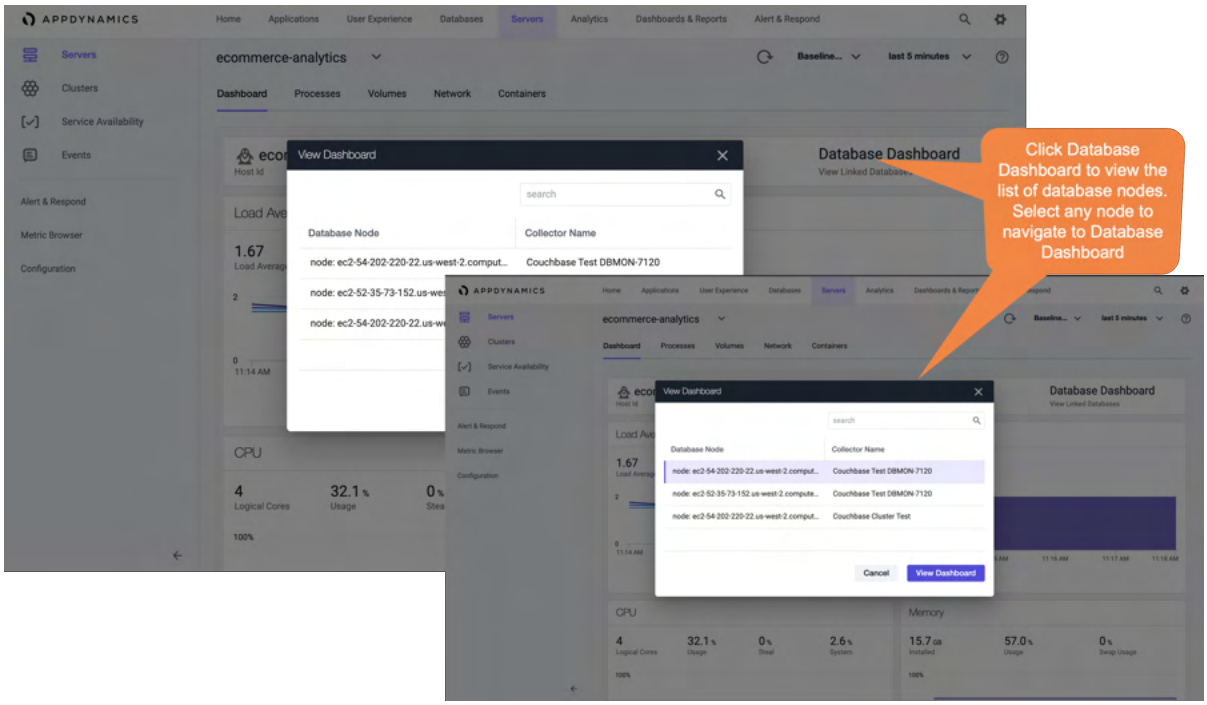
For a database cluster, you can find the link to each Server Dashboard corresponding to each database node.



When you click the connected server under the **Server Dashboard** section, the page redirects to the Server Visibility dashboard with all the hardware metrics.



On the Server Dashboard, you can click the **Database Dashboard**, which displays a dialog that includes the list of database nodes that are connected to the Server Visibility machine. You can select any database node to navigate to the Database Dashboard.


APPDYNAMICS
Home
Applications
User Experience
Databases
Servers
Analytics
Dashboards & Reports
Alert & Respond
Servers
Clusters
Service Availability
Events
Alert & Respond
Metric Browser
Configuration
ecommerce-analytics
Baseline...
last 5 minutes
Dashboard
Processes
Volumes
Network
Containers
Database Dashboard
View Linked Database...
View Dashboard
search
Database Node
Collector Name
node: ec2-54-202-220-22.us-west-2.comput...
Couchbase Test DBMON-7120
node: ec2-52-35-73-152.us-west-2.compute...
Couchbase Test DBMON-7120
node: ec2-54-202-220-22.us-west-2.comput...
Couchbase Cluster Test
Cancel
View Dashboard
Click Database Dashboard to view the list of database nodes. Select any node to navigate to Database Dashboard
CPU
Memory
Load Ave
1.67
4
Logical Cores
32.1 %
Usage
0 %
2.6 %
15.7 GB
57.0 %
0 %
11:14 AM
11:16 AM
11:17 AM
11:18 AM

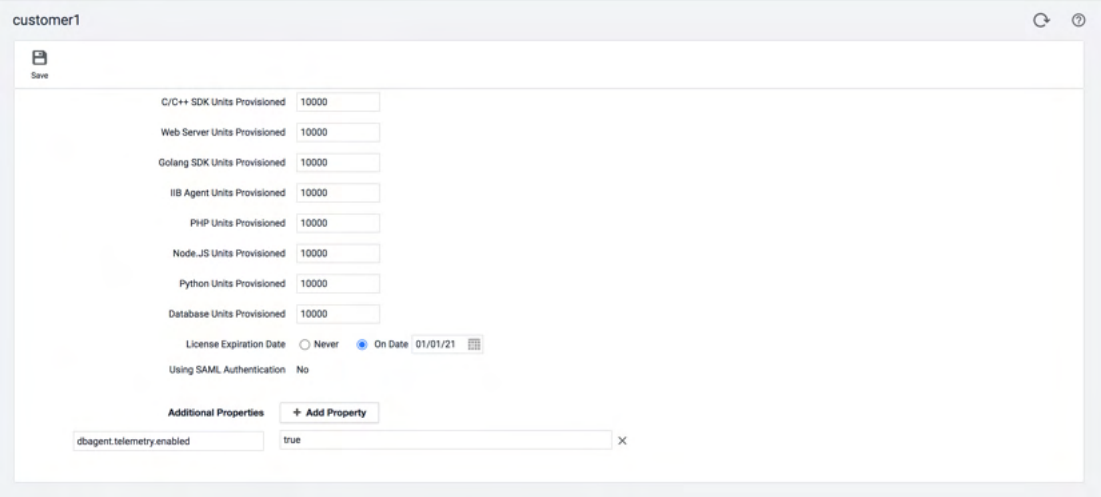
Database Agent Telemetry

You can monitor the Database Agent metrics and configure health rules for the agent. To add a health rule or view the metrics along with the status of the agent, you must enable the agent telemetry property.

Enable Database Agent Telemetry

You can enable the agent telemetry property at the account level on the Controller or at the agent level. If you configure the property on the Controller, all metrics display for each Database Agent. Whereas, if you configure the property in the agent configuration, the metrics display only for the configured agent.

For account level on Controller, use the Administration Console to add the `dbagent.telemetry.enabled` feature flag and set the value to `true`.



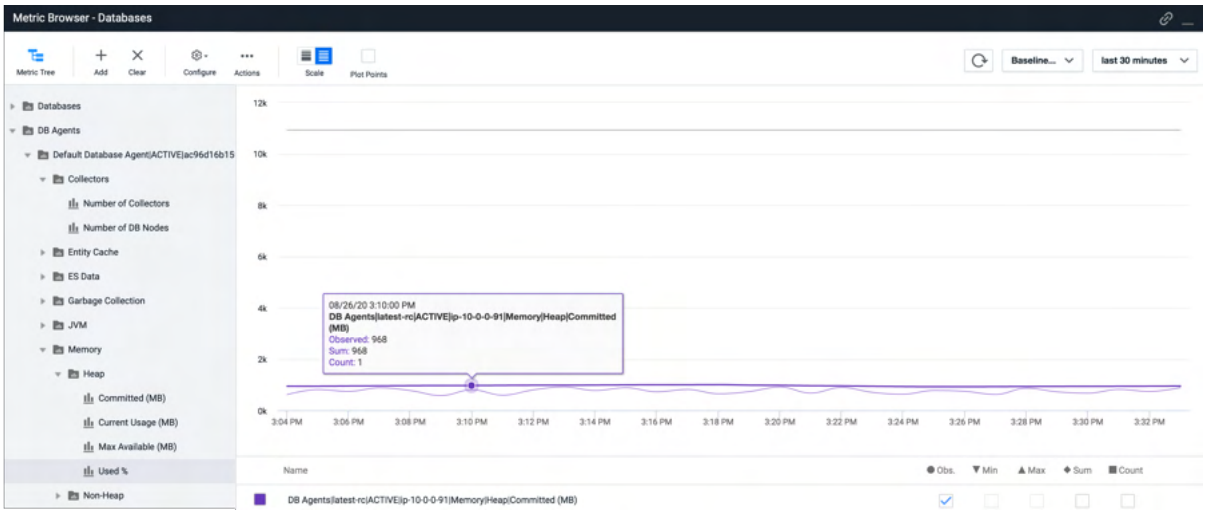
Similarly, for only specific Database Agents, start the Database Agent with the `-Ddbagent.telemetry.enabled=true` system properties.

> Each agent is associated with a unique host identifier and name. If two or more agents have the same name and unique identifier, the agents report metrics in the same node. Therefore, start a Database Agent with a different name and unique host identifier.
>
> The unique host identifier is by default the host-name of the Database Agent host machine. You can override the unique host identifier by using the system property `-Dappdynamics.agent.uniqueHostId=<Some_unique_identifier_name>`. See Unique Host ID at Database Agent Configuration Properties.

To configure a health rule for Database Agent, see Database Health Rules and Alerts.

Monitor Database Agent Metrics

You can view the Database Agent metrics in the **Metric Browser**. After enabling the Database Agent telemetry, the metric tree displays the list of Database Agent names along with its status under **DB Agents**. Each Database Agent name includes the list of Database Agent metrics.



JVM and Memory Metrics

This table lists the JVM and Memory metrics of Database Agent:

Metric Name	Description
DB AGENT|Memory|Heap|Committed (MB)	Returns the memory in MB that is committed for the Java virtual machine (Database Agent) to be used for the heap
DB AGENT|Memory|Heap|Current Usage (MB)	Returns the amount of used memory in MB
DB AGENT|Memory|Heap|Max Available (MB)	Returns the maximum amount of memory in MB that can be used for memory management
DB AGENT|Memory|Heap|Used %	(Current Usage/Max Available) * 100
DB AGENT|Memory|Non-Heap|Committed (MB)	Returns the memory in MB that is committed for the Java virtual machine (Database Agent) to be used for the non-heap
DB AGENT|Memory|Non-Heap|Current Usage (MB)	Returns the amount of used memory in MB (for non-heap)
DB AGENT|JVM|Process CPU Burnt (ms /min)	CPU Time in milliseconds (ms) by this process
DB AGENT|JVM|Process CPU Usage %	CPU Used % in that minute. (CPU Time in that min/(Up Time in that min * number of processors))* 100 This metric is cumulative
DB AGENT|JVM|Threads|Current No. of Threads	Number of current live threads (including daemon and non-daemon)

Garbage Collections and Memory Pool Metrics

This table lists the Garbage Collections (GC) and the memory pool metrics:

Metric Name	Description
DB AGENT|Garbage Collection|Major Collection Time Spent Per Min (ms)	Time taken by Major GC at time of the metric collection
DB AGENT|Garbage Collection|Number of Major Collections Per Min	Major GC Count at time of the metric collection
DB AGENT|Garbage Collection|Minor Collection Time Spent Per Min (ms)	Time taken by minor GC at time of the metric collection
DB AGENT|Garbage Collection|Number of Minor Collections Per Min	Minor GC Count at time of the metric collection
DB AGENT|Garbage Collection|GC Time Spent Per Min (ms)	Major GC Time(if present) + Minor GC Time(if present)
DB AGENT|Garbage Collection|Memory Pools|Code Cache|Committed (MB)	Returns the memory in MB that is committed for the Java virtual machine (Database Agent) to be used for the non-heap memory pools
DB AGENT|Garbage Collection|Memory Pools|Code Cache|Current Usage (MB)	Returns the amount of used memory in MB for non-heap memory pools
DB AGENT|Garbage Collection|Memory Pools|Code Cache|Max Available (MB)	Returns the maximum amount of memory in MB that can be used for memory management for non-heap memory pools
DB AGENT|Garbage Collection|Memory Pools|Compressed Class Space|Committed (MB)	Returns the memory in MB that is committed for the Java virtual machine (Database Agent) to be used for the non-heap memory pools
DB AGENT|Garbage Collection|Memory Pools|Compressed Class Space|Current Usage (MB)	Returns the amount of used memory in MB for non-heap memory pools
DB AGENT|Garbage Collection|Memory Pools|Compressed Class Space|Max Available (MB)	Returns the maximum amount of memory in MB that can be used for memory management for non-heap memory pools
DB AGENT|Garbage Collection|Memory Pools|Metaspace|Committed (MB)	Returns the memory in MB that is committed for the Java virtual machine (Database Agent) to be used for the non-heap memory pools
DB AGENT|Garbage Collection|Memory Pools|Metaspace|Current Usage (MB)	Returns the amount of used memory in MB for non-heap memory pools

DB AGENT|Garbage Collection|Memory Pools|PS Eden Space|Committed (MB)	Returns the memory in MB that is committed for the Java virtual machine (Database Agent) to be used for the heap memory pools
DB AGENT|Garbage Collection|Memory Pools|PS Eden Space|Current Usage (MB)	Returns the amount of used memory in MB for heap memory pools
DB AGENT|Garbage Collection|Memory Pools|PS Eden Space|Max Available (MB)	Returns the maximum amount of memory in MB that can be used for memory management for heap memory pools
DB AGENT|Garbage Collection|Memory Pools|PS Old Gen|Committed (MB)	Returns the memory in MB that is committed for the Java virtual machine (Database Agent) to be used for the heap memory pools
DB AGENT|Garbage Collection|Memory Pools|PS Old Gen|Current Usage (MB)	Returns the amount of used memory in MB for heap memory pools
DB AGENT|Garbage Collection|Memory Pools|PS Old Gen|Max Available (MB)	Returns the maximum amount of memory in MB that can be used for memory management for heap memory pools
DB AGENT|Garbage Collection|Memory Pools|PS Survivor Space|Committed (MB)	Returns the memory in MB that is committed for the Java virtual machine (Database Agent) to be used for the heap memory pools
DB AGENT|Garbage Collection|Memory Pools|PS Survivor Space|Current Usage (MB)	Returns the amount of used memory in MB for heap memory pools
DB AGENT|Garbage Collection|Memory Pools|PS Survivor Space|Max Available (MB)	Returns the maximum amount of memory in MB that can be used for memory management for heap memory pools

The listed metrics are for Java HotSpot(TM) 64-Bit Server JVM and 1.8.0_212 Java version. These metrics might change depending on the JVM.

Cache Usage Metrics

The following table lists the cache usage metrics that are accumulated metrics for all collectors that are monitored by the Database Agent.

The following details help in understanding the metrics mentioned in the table:

- `entities` stores all the entities with its hash and id
- `unregQueryCache` stores all unregistered query entities till its registration
- `unregCaches` stores all unregistered entities (except Server, Query, Query Plan) till its registration
- `unregQueryPlanCache` stores all unregistered query plan entities till its registration
- `counters` store all the unregistered wait states till they get converted to DTO's and get Uploaded to Controller/ES
- `catalog` stores all the unregistered query statistics till they get converted to DTO's and get Uploaded to Controller/ES

MetricName	Description
DB AGENT|Entity Cache|Total Entities	The total number of all types of entities stored in EntityCache All entity types are taken into account except `Server`
DB AGENT|Entity Cache|Registered Entities	Number of entities registered in that minute All entity types are taken into account except Server
DB AGENT|Entity Cache|Pending Entities	Number of pending entities waiting in `unRegCache`, `unRegQueryCache`, or `unRegQueryPlanCache` to get registered All entity types are taken into account except `Server`
DB AGENT|Entity Cache|Duplicate Entities	Number of entities that are for re-registration All entity types are taken into account except `Server`
DB AGENT|Entity Cache|Unregistered Query Cache Size (KB)	Total query size in KB of queries that are present in the cache(`unregQueryCache`), waiting for registration Only the Query entity type is taken into account.
DB AGENT|ES Data|Cache Usage|Number Of Wait Counters	Number of Wait Counters present in the cache, waiting to get uploaded
DB AGENT|ES Data|Cache Usage|Number of Query Stats	Number of Query Stats present in the cache, waiting to get uploaded
DB AGENT|ES Data|Successful ES Uploads|Number of Wait Counters	Number of Wait Counters uploaded to ES in that min

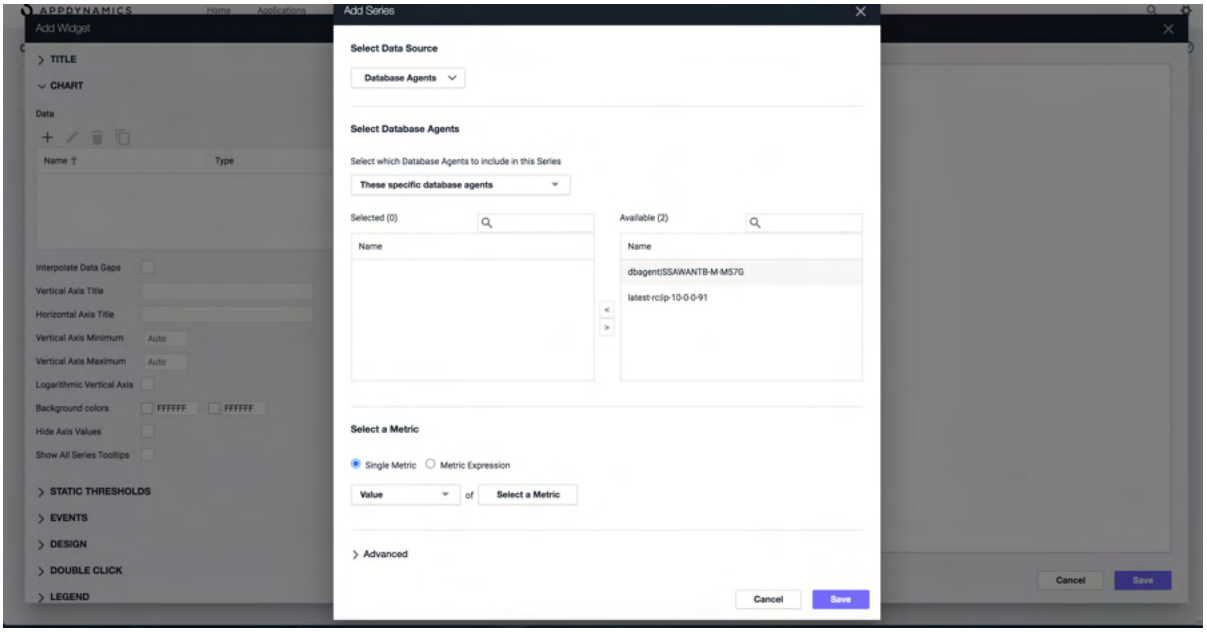
DB AGENT|ES Data|Successful ES Uploads|Number of Query Stats	Number of Query Stats uploaded to ES in that min

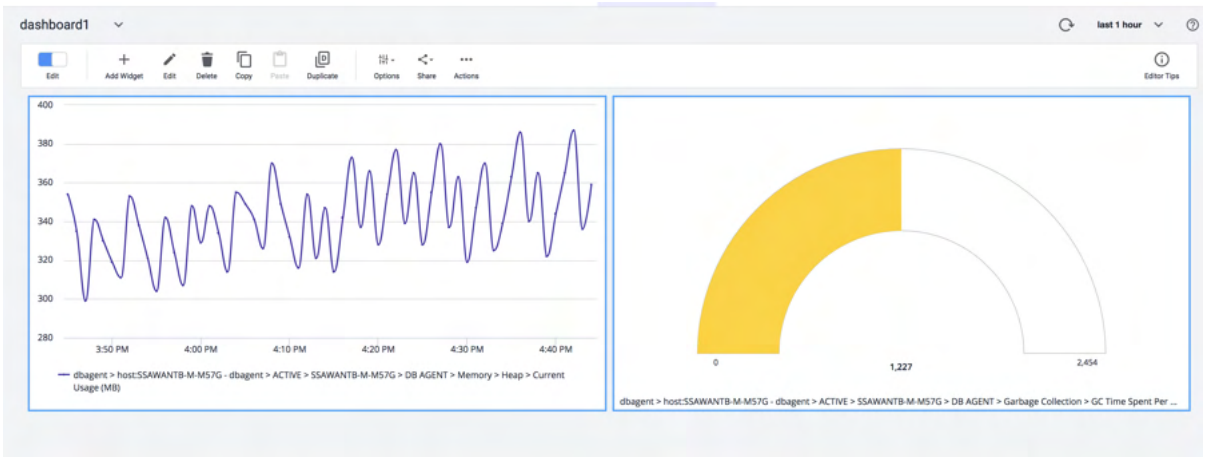
Monitor Database Agent Metrics on Database Custom Dashboard

You can monitor the Database Agent metrics on **Custom Dashboard**.

To create a custom dashboard:

1. Click **Dashboards & Reports > Create Dashboard**.
2. Add a widget.
3. Add the data by selecting the data source as **Database Agents**. See Custom Dashboards.





Infrastructure Visibility

AppDynamics Infrastructure Visibility provides end-to-end visibility on the performance of the hardware running your applications. You can use Infrastructure Visibility to identify and troubleshoot problems that can affect application performance such as server failures, JVM crashes, and network packet loss.

Infrastructure Visibility provides these metrics:

- CPU busy/idle times, disk and partition reads/writes, and network interface utilization (Machine Agents)
- Packet loss, round trip times, connection setup/tear down errors, TCP window size issues, and re-transmission timeouts (Network Visibility, additional license required)
- Disk/CPU/memory utilization, process, and machine availability (Server Visibility, additional license required)

Machine Agent

- Requirements and Supported Environments
- Install the Agent
- Configure the Agent
- Enable SSL for the Agent
- Administer the Agent
- Extensions and Custom Metrics
- Hardware Resources Metrics
- Machine Agent Metric Collection

Server Visibility

- Requirements and Supported Environments
- Monitoring Servers
- Configuring Docker Visibility
- Configuring Health Rules to Monitor Servers
- Tier Metric Correlator
- Server Tagging
- Service Availability Monitoring

Network Visibility

- Requirements and Supported Environments
- Set Up Network Visibility on Linux
- Administer the Network Agent on Linux
- Set up Network Visibility on Windows
- Administer the Network Agent on Windows
- Network Visibility Metrics
- FAQs for Network Visibility
- Workflows and Use Cases
- Advanced Operations

Monitoring Kubernetes and Containers

- Monitoring Kubernetes with the Cluster Agent
- Monitoring Containers with Docker Visibility
- Container Metrics

Overview of Infrastructure Visibility

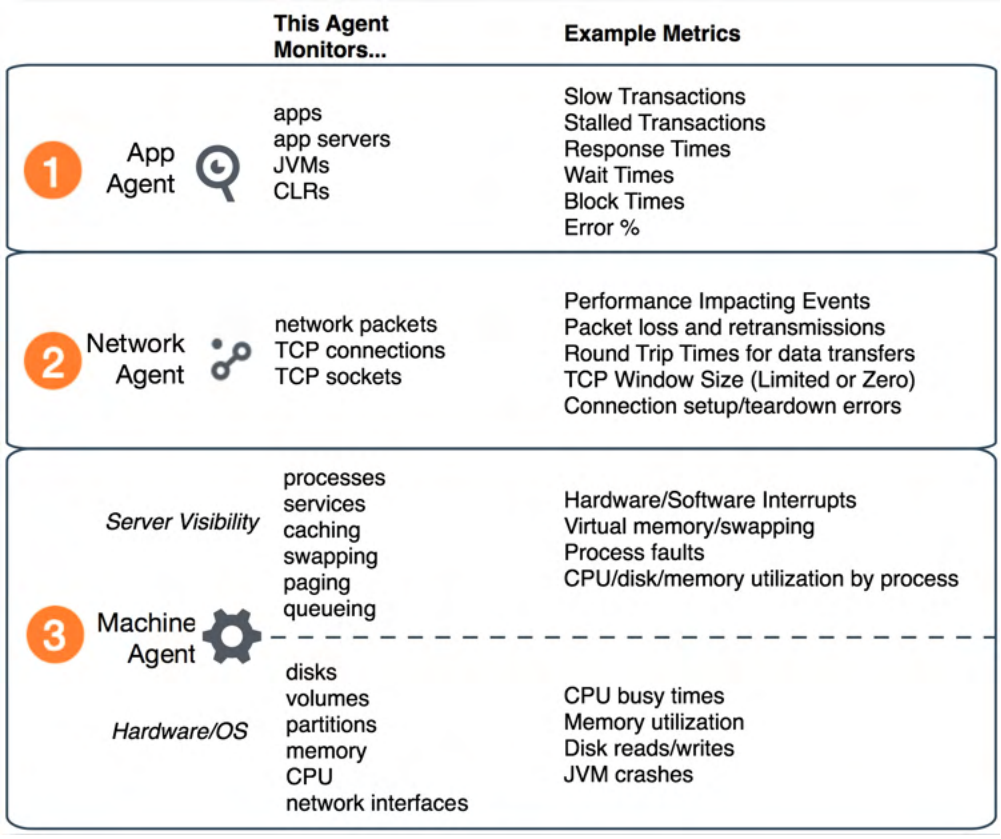
You can determine the root cause of application issues by looking at application, network, server, and machine metrics that measure infrastructure utilization.
For example, the following infrastructure issues may slow down your application:

- Too much time spent in garbage collection of temporary objects (application metric)
- Packet loss between two nodes that results in re-transmissions and slow calls (network metric)
- Inefficient processes that result in high CPU utilization (server metric)
- Excessively high rates of reads/writes on a specific disk or partition (hardware metric)

Infrastructure Visibility enables you to isolate, identify, and troubleshoot these types of issues. Infrastructure Visibility is based on a Machine Agent that runs with an App Server Agent on the same machine. These two agents provide multi-layer monitoring:

1. The App Server Agent collects metrics about applications and identifies applications, tiers, and nodes with slow transactions, stalled transactions, and other application-performance issues.
2. The Network Agent monitors the network packets sent and received on each node and identifies lost/re-transmitted packets, TCP bottlenecks, high round trip times, and other network issues.
3. The Machine Agent collects metrics at two levels:
    - Server Visibility metrics for local processes, services, and resource utilization.
    - Basic machine metrics for disks, memory, CPU, and network interfaces.

This multi-layer monitoring enables you to determine possible correlations between application issues and service, process, hardware, network, or other issues on the machine.



Network Visibility

Network Visibility monitors traffic flows, network packets, TCP connections, and TCP ports. Network Agents leverage the APM intelligence of App Server Agents to identify the TCP connections used by each application. Network Visibility includes:

- Detailed metrics about dropped/re-transmitted packets, TCP window sizes (Limited/Zero), connection setup/tear down issues, high round trip times, and other performance-impacting issues
- Network Dashboard that highlights network KPIs (Key Performance Indicators) for tiers, nodes, and network links
- Right-click dashboards for tiers, nodes, and network links that enable quick drill-downs from transaction outliers to network root causes
- Automatic mapping of TCP connections with application flows
- Automatic detection of intermediate load balancers that split TCP connections
- Diagnostic mode for collecting advanced diagnostic information for individual connections

Server Visibility

Server Visibility monitors local processes, services, and resource utilization. You can use these metrics to identify time windows when problematic application performance correlates with problematic server performance on one or more nodes.

Server Visibility is an add-on module to the Machine Agent. With Server Visibility enabled, the Machine Agent provides the following functionality:

- Extended hardware metrics such as machine availability, disk/CPU/virtual-memory utilization, and process page faults
- Monitor application nodes that run inside Docker containers and identify container issues that impact application performance
- The Tier Metric Correlator which enables you to identify load and performance anomalies across all nodes in a tier
- Import and define server tags used to query, filter, and compare related servers using custom metadata
- Monitor internal or external HTTP and HTTPS services
- Support for grouping servers so you can apply health rules to specific server groups
- Support for defining alerts that trigger when certain conditions are met or exceeded based on monitored server hardware metrics

Basic Machine Metrics

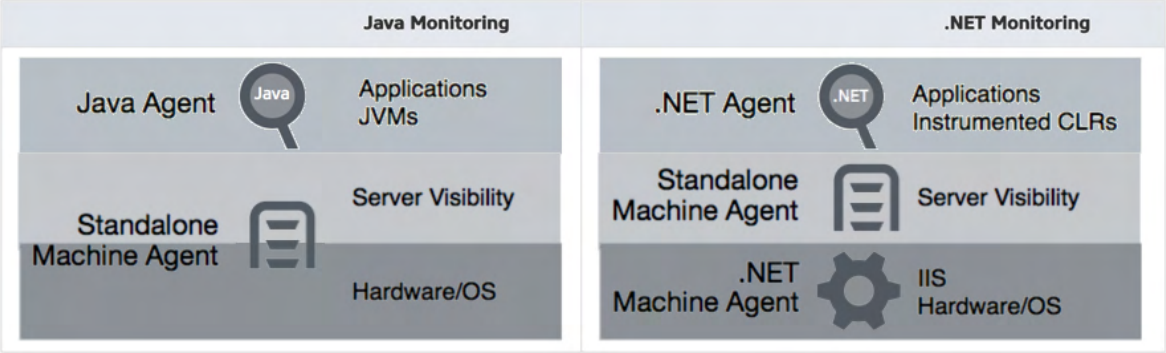
The Machine Agent collects basic hardware metrics from the server's OS and provides the following functionality:

- Basic hardware metrics from the server's OS such as CPU and memory utilization, throughput on network interfaces, and disk and network I/O
- Support for creating extensions to generate custom metrics
- Support for running remediation scripts to automate your runbook procedures. You can optionally configure the remediation action to require human approval before starting the script.
- JVM Crash Guard for monitoring JVM crashes and optionally running remediation scripts

Java and .NET Infrastructure Monitoring

Infrastructure Visibility uses different agents to monitor Java and .NET environments:

- The Java Agent collects metrics for business applications and JVMs. The Machine Agent collects Server Visibility and hardware/OS metrics.
- The .NET Agent collects metrics for business applications and instrumented CLRs. The .NET Agent includes a .NET Machine Agent that collects IIS and hardware/OS metrics (see Monitor Windows Hardware Resources). The Machine Agent collects Server Visibility metrics.



Infrastructure Visibility Strategies

You can use these strategies to locate infrastructure issues that affect application performance:

- Transaction snapshots for slow or stalled transactions – Use snapshots to correlate infrastructure metrics for the specific node so that you can identify the root cause of slow or stalled transactions.
- Metric correlation –
  - One example workflow is to open the Node Dashboard for a mission-critical server with a machine agent installed and then cross-compare data in the following tabs:
    - JVM (application performance)
    - JMX (server performance)
    - Server (hardware resource consumption)
  - The Network Dashboard includes right-click dashboards for tiers, nodes, and network links. Use these dashboards to find correlations between application issues and network root causes.
  - The Tier Metric Correlator enables you to identify load and performance anomalies in a tier composed of a cluster of nodes running on containers or servers.
- Health rules – Configure health rules on metrics such as garbage collection time, connection pool contention, or CPU usage to catch issues early in the cycle before any impact on your business transactions.
- Infrastructure rules, policies, and alerts –
  - Create health rules on metrics such as garbage collection time, connection pool contention, or CPU usage to catch issues early in the cycle before any impact on your business transactions.
  - Define policies that trigger actions (such send an email, start diagnostics, or perform a thread dump) when Infrastructure metrics report a critical level.
  - You can configure alerts for JVM and CLR crashes respectively using JVM Crash Guard and the .NET Machine Agent.
  - Configure the agent to run scripts in response to critical events (for example, restart an application or JVM in response to a crash).

With the right monitoring strategy in place, you can be alerted to problems and fix them before user transactions are affected.

Hardware Resources Metrics

This page describes the basic hardware metrics collected by the Machine Agent and the additional metrics collected by the Machine Agent for Server Visibility. Not all of the metrics that appear in the Hardware Resources branch of the Metric Browser tree are provided by the Machine Agent; some are collected by the Database Agent and are used in the Database Monitoring UI, while others might be custom metrics added by another extension run by your Machine Agent.

For most metrics in the Metric Browser, you can select any point in the graph to view more information about the metric observed at that point in time. The information includes the metric identifier, date and time of the observation, along with any of the following values relevant to the metric:

- **Obs** (observed value): Average of all data points seen for that interval. For a cluster or a time rollup, this represents the weighted average across nodes or over time.
- **Min**: Minimum data point value seen for that interval
- **Max**: Maximum data point value seen for that interval
- **Sum**: Sum of all data point values seen for that interval. For the Percentile Metric for the App Agent for Java, this is the result of the percentile value multiplied by the Count.
- **Count**: Number of data points generated for the metric in that interval. The collection interval for infrastructure metrics varies by environment. For example:
    - For AIX, HP-UX, Mac OS X and Z/OS: CPU and memory metrics are gathered every two seconds and averaged over a period of one minute. Machine agent network and disk metrics are gathered at one-minute intervals.
    - For Windows, Linux, and Solaris: All metrics are collected at one-minute intervals and aggregated over one-minute intervals.

CPU Metrics

Metric Name	Description	Basic or Server Visibility	Windows	Linux [1]	Solaris	AIX	Default Monitoring Mode [3]
%Idle	Percentage of time the CPU was idle; the CPU had completed its tasks and has nothing to do.	Basic	✓	✓	✓	✓	KPI
%Busy	Percentage of time the CPU was busy processing system or user requests; this metric includes CPU Stolen time.	Basic	✓	✓	✓	✓	KPI
%Stolen [4]	Usually, stolen time is the percentage of time a virtual CPU waits for a real CPU while the hypervisor is servicing another virtual processor.	Basic	✓	✓	✓	✓	KPI
%Nice	Percentage of CPU time spent on low priority processes.	Server Visibility	✗	✓	✗	✗	Advanced
System	Percentage of time the CPU was busy processing kernel code.	Server Visibility	✓	✓	✓	✓	KPI
User	Percentage of time the CPU was busy processing non-kernel code.	Server Visibility	✓	✓	✓	✓	KPI
IOWait	Percentage of CPU time spent waiting for an I/O request.	Server Visibility	✗	✓	✓	✗	KPI
%Irq	Percentage of CPU time spent servicing hardware interrupts.	Server Visibility	✗	✓	✗	✗	Diagnostic
%SoftIrq	Percentage of CPU time spent servicing software interrupts.	Server Visibility	✗	✓	✗	✗	Advanced
%Busy 95th Percentile [2]	The CPU %Busy percentage was at this level or lower 95% of the time.	Server Visibility	✗	✓	✗	✓	KPI
IOWait 95th Percentile [2]	The CPU %Busy percentage was at this level or lower 95% of the time.	Server Visibility	✗	✓	✗	✓	KPI

1 See Linux Kernel Processes.

2 See Percentile Metric Reporting must be Enabled.

3 See Default Monitoring Mode.

4 See Stolen Times are Reported Differently.

Memory Metrics

Metric Name	Description	Basic or Server Visibility	Windows	Linux [1]	Solaris	AIX	Default Monitoring Mode [3]
Total (MB)	The total amount of memory.	Basic	✓	✓	✓	✓	KPI

Used %	The percentage of memory used.	Basic	✓	✓ [4]	✓	✓	KPI
Used (MB)	The amount of memory used.	Basic	✓	✓ [4]	✓	✓	Advanced
Free %	Percentage of free or unused memory available for processes.	Basic	✓	✓ [4]	✓	✓	Advanced
Free (MB)	The total amount of free or unused memory available for processes.	Basic	✓	✓ [4]	✓	✓	Advanced
Swap Free (MB)	The total amount of free swap space. Reported for each hierarchical group.	Server Visibility	✓	✓	✓	✓	Advanced
Swap Total (MB)	The total amount of allocated swap space. Reported for each hierarchical group.	Server Visibility	✓	✓	✓	✓	KPI
Swap Used (MB)	The amount of swap space used. Reported for each hierarchical group.	Server Visibility	✓	✓	✓	✓	Advanced
Swap Used %	The percentage of available swap space used. Reported for each hierarchical group.	Server Visibility	✓	✓	✓	✓	KPI
Swap Free %	Percentage of free swap space. Reported for each hierarchical group.	Server Visibility	✓	✓	✓	✓	Advanced
Pages Paged In 95th Percentile [2]	The paging-in rate to memory was at this level or lower 95% of the time.	Server Visibility	✗	✓	✗	✓	KPI
Pages Paged Out 95th Percentile [2]	The paging-out rate from memory was at this level or lower 95% of the time.	Server Visibility	✗	✓	✗	✓	KPI
Pages Swapped In 95th Percentile [2]	The swapping rate of pages from disks was at this level or lower 95% of the time.	Server Visibility	✗	✓	✗	✓	KPI
Pages Swapped Out 95th Percentile [2]	The swapping rate of pages to disks was at this level or lower 95% of the time.	Server Visibility	✗	✓	✗	✓	KPI

1 See Linux Kernel Processes.

2 See Percentile Metric Reporting must be Enabled.

3 See Default Monitoring Mode.

4 See Configure Free/Used Memory Metric Calculation on Linux.

Disk and Partition Metrics

The agent reports metrics for each disk partition and for disks in aggregate. Only mounted partitions and local partitions are reported.

Metric Name	Description	Basic or Server Visibility	Windows	Linux	Solaris	AIX	Default Monitoring Mode[1]
KB read /sec	The number of KB per second read from all disks and partitions.	Basic	✓	✓	✓	✓	Advanced
KB written /sec	The average amount of data per second written to all disks and partitions.	Basic	✓	✓	✓	✓	Advanced
Reads/sec	Number of read operations per second performed on all disks and partitions.	Basic	✓	✓	✓	✓	KPI
Writes/sec	Number of write operations per second performed on all disk and partitions.	Basic	✓	✓	✓	✓ [5]	KPI
Avg IO Utilization (%)	The average time spent processing read/write requests on all disks and partitions as a percentage of the total reported time window. Databases often report high disk I/O utilization due to frequent read/write requests. For example, if the agent detects read/write processing in 55 out of 60 seconds, the Avg IO Utilization for that minute is 92%. This metric does not measure the amount of available disk space or read/write request sizes.	Server Visibility	✓	✓	✓	✓	KPI
<partition> %CPU Time	The percentage of CPU processing consumed by a partition during read and write operations.	Basic	✗	✓	✗	✓	Diagnostic
<partition> Avg Service Time (ms)	Time in milliseconds spent performing read and write operations across one partition.	Basic	✗	✓	✓	✓	Diagnostic
<partition> Avg Queue Time (ms)	Time in milliseconds that a read or write request is in the queue before it gets processed across one partition.	Basic	✗	✓	✗	✓	KPI
<partition> KB read /sec	The average amount of data per second read from one specific partition.	Basic	✓	✓	✓	✓	Diagnostic
<partition> KB written /sec	The average amount of data per second written to one specific partition.	Basic	✓	✓	✓	✓	Diagnostic
<partition> Reads/sec	Number of read operations per second performed on one specific partition.	Basic	✓	✓	✓	✓	KPI

<partition> Writes /sec	Number of write operations per second performed on one specific partition.	Basic	✓	✓	✓	✓[5]	KPI
<partition> Space Available	The amount of unused or free disk space on a specific partition in KB.	Basic	✓	✓[4]	✓	✓	KPI
<partition> Space Used	The amount of used or unavailable disk space on a specific partition in KB.	Basic	✓	✓[4]	✓	✓	Advanced
<partition> Avg IO Utilization (%)	The average time spent processing read/write requests as a percentage of the total reported time window. Databases often report high disk I/O utilization due to frequent read/write requests. For example, if the agent detects read/write processing in 55 out of 60 seconds, the Avg IO Utilization for that minute is 92%. This metric does not measure the amount of available disk space or read/write request sizes.	Server Visibility	✓	✓	✓	✓	KPI
<partition> Avg read time (ms)	The average time in milliseconds required to service a read request by all disks or by one specific disk or across one partition.	Server Visibility	✓[2]	✓	✓	✗	KPI
<partition> Avg write time (ms)	The average time in milliseconds required to service a write request across one partition.	Server Visibility	✓[2]	✓	✓	✗	KPI
<partition> Queue Time 95th Percentile [3]	The queue time for read/write requests was this long or shorter 95% of the time for the reported time window.	Server Visibility	✗	✓	✗	✓	KPI

1 See Default Monitoring Mode.

2 For some versions of Windows Server 2008 and Windows Vista, the Hardware Resources|Disks|<partition>|Avg Read Time (ms) and Hardware Resources|Disks|<partition>|Avg Write Time (ms) metrics will be reported as 0. This is due to a known bug with Windows. To resolve this issue, download the hotfix: https://support.microsoft.com/en-us/kb/961435.

3 See Percentile Metric Reporting must be Enabled.

4 See Configure Disk Usage Metric Collection on Linux.

5 See HardwareMonitor and JavaHardwareMonitor Calculate "Writes/Sec" Differently on AIX Machines

Volume Metrics

AppDynamics Server Visibility retrieves the volume space metrics on POSIX systems using the `df` command. The volume metrics are reported across all listed volumes and for each volume at a specified mount point, such as `"/boot"`. Only local volumes are reported.

Metric Name	Description	Basic or Server Visibility	Windows	Linux [1]	Solaris [2]	AIX	Default Monitoring Mode[3]
Total (MB)	The amount of storage space available (used and free) across all listed volumes or at the specified mount point. On Linux, the space reserved for root is not counted in the available space.	Server Visibility	✓	✓	✓	✓	KPI
Free (MB)	The amount of unused or free space across all listed volumes or on the selected volume at the specified mount point. On Linux, the space reserved for root is not counted in the available space.	Server Visibility	✓	✓	✓	✓	Advanced (*volume*)
Used (MB)	The amount of storage space in use across all listed volumes or on the selected volume at the specified mount point.	Server Visibility	✓	✓	✓	✓	Advanced (*volume*)
Used (%)	The percentage of storage space in use across all listed volumes or on the selected volume at the specified mount point.	Server Visibility	✓	✓	✓	✓	KPI
Count	The number of partitions for which storage space metrics are collected.	Server Visibility				✓	KPI
<partition> Free (MB)	The amount of unused or free space for <partition>.	Server Visibility				✓	KPI
<partition> Total (MB)	The amount of storage space available (used and free) for <partition>.	Server Visibility				✓	KPI
<partition> Used (%)	The percentage of storage space in use for <partition>.	Server Visibility				✓	KPI
<partition> Used (MB)	The amount of storage space in use for <partition>.	Server Visibility				✓	Advanced

1 Only **/dev** volumes are monitored on Linux.

2 Only **/dev/dsk** and **/rpool** volumes are monitored on Solaris.

3 See Default Monitoring Mode.

Load Metrics

The load metrics are reported for each machine. The CPU % (reported as part of the basic Machine Agent metrics) is the percentage of the CPU consumed by processes that are currently running. Load takes into account processes that are waiting to run. These metrics are shown as percentages in the Server Dashboard and are scaled by 100 in the Metric Browser.

Metric Name	Description	Basic or Server Visibility	Windows	Linux	Solaris	AIX	Default Monitoring Mode[1]
Last 1 minute	CPU Load, presented as an average over the last 1 minute.	Server Visibility	✓	✓	✓	✓	KPI
Last 5 minutes	CPU Load, presented as an average over the last 5 minutes.	Server Visibility	✓	✓	✓	✓	KPI
Last 15 minutes	CPU Load, presented as an average over the last 15 minutes.	Server Visibility	✓	✓	✓	✓	KPI

1 See Default Monitoring Mode.

Machine Metrics

Metric Name	Description	Basic or Server Visibility	Windows	Linux	Solaris	AIX	Default Monitoring Mode[1]
Availability	The percentage of time the Machine Agent was reporting to the Controller. In the Server Visibility UI, this provides a percentage with 6 digits of precision (for example, 100.0000) to measure availability. The Machine Agent sends a heartbeat to the AppDynamics Controller once per minute to indicate the agent is reporting. Reported for each machine.	Server Visibility	✓	✓	✓	✓	KPI

1 See Default Monitoring Mode.

Network Metrics

The Machine Agent is configured to ignore virtual networks by default. See Machine Agent Settings for Server Visibility.

Metric Name	Description	Basic or Server Visibility	Windows	Linux	Solaris	AIX	Default Monitoring Mode[1]
Incoming KB[2]	The volume of data received by all monitored network devices.	Basic	✓	✓	✓	✓	Advanced
Incoming KB /sec	The amount of data per second received by all monitored network devices.	Basic	✓	✓	✓	✓	KPI
Incoming packets[2]	The number of packets received by all monitored network devices.	Basic	✓	✓	✓	✓	Advanced
Incoming packets/sec	The number of data packets per second received by all monitored network devices.	Basic	✓	✓	✓	✓	Advanced
Outgoing KB[2]	The volume of data sent by all monitored network devices.	Basic	✓	✓	✓	✓	Advanced
Outgoing KB /sec	The volume of data sent per second by all monitored network devices.	Basic	✓	✓	✓	✓	KPI
Outgoing packets[2]	The number of packets sent by all monitored network devices.	Basic	✓	✓	✓	✓	Advanced
Outgoing packets/sec	The number of data packets sent per second by all monitored network devices.	Basic	✓	✓	✓	✓	Advanced
Incoming Errors/min	The number of incoming packet errors the network incurs every minute.	Server Visibility	✓	✓	✓	✓	KPI
Outgoing Errors/min	The number of outgoing packet errors the network incurs every minute.	Server Visibility	✓	✓	✓	✓	KPI
Avg Utilization (%)	The average network utilization as a percentage of the maximum possible throughput. This metric is not reported if the agent cannot determine the throughput (not supported for some devices and Linux versions). The percentage is rounded to the nearest integer. Therefore, very low utilization may be reported as 0%. Reported for: • Servers • <group> • <network interface>	Server Visibility	✓	✓	✓	✓	KPI
<network interface> Incoming KB	The volume of data received by the selected network interface.	Basic	✓	✓	✓	✓	Advanced

<network interface> Incoming KB /sec	The volume of data received per second by the selected network interface.	Basic	✓	✓	✓	✓	KPI
<network interface> Incoming packets	The number of data packets received by the selected network interface.	Basic	✓	✓	✓	✓	Advanced
<network interface> Incoming packets/sec	The number of data packets received per second by the selected network interface.	Basic	✓	✓	✓	✓	Advanced
<network interface> Outgoing KB	The volume of data sent by the selected network interface.	Basic	✓	✓	✓	✓	Advanced
<network interface> Outgoing KB /sec	The volume of data sent per second by all monitored network interface.	Basic	✓	✓	✓	✓	KPI
<network interface> Outgoing packets	The number of data packets sent per second by the selected network interface.	Basic	✓	✓	✓	✓	Advanced
<network interface> Outgoing packets/sec	The number of data packets sent per second by the selected network interface.	Basic	✓	✓	✓	✓	Advanced
<network interface> Incoming Errors/min	The number of incoming packet errors the network incurs every minute.	Server Visibility	✓	✓	✓	✓	KPI
<network interface> Outgoing Errors/min	The number of outgoing packet errors the network incurs every minute.	Server Visibility	✓	✓	✓	✓	KPI

1 See Default Monitoring Mode.

2 The ServerMonitoring and JavaHardwareMonitor extensions calculate unidirectional throughput (packet/KB) metrics differently. ServerMonitoring reports the total number of packets /bytes in the last minute. JavaHardwareMonitor reports the total number of packets/bytes from the time the Machine Agent started.

Process Metrics

The following metrics are aggregated and reported for each process or process class (except Total Process Count, which measures all processes observed by the agent). See Machine Agent Settings for Server Visibility to modify the default process monitoring.

Metric Name	Description	Basic or Server Visibility	Windows	Linux [1]	Solaris [2]	AIX	Default Monitoring Mode[3]
Count	The number of processes in this class consuming CPU or memory resources or the total number of processes in this class.	Server Visibility	✓	✓	✓	✓	Diagnostic
CPU Used (%)	Percentage of the CPU bandwidth used by all processes in a process class. A process using 100% CPU is executing on all processors on the system. If there are four cores on the machine and the process is executing four threads, each executing on one core, then the process can use up to 400% CPU. If there are four cores on the machine and the process is executing one thread on one core, then the process can use up to 25% CPU.	Server Visibility	✓	✓	✓	✓	KPI
Major Faults/sec	Number of major page faults caused by all processes in this class.	Server Visibility	✗	✓	✗	✓	Diagnostic
Minor Faults/sec	Number of minor page faults caused by all processes in this class.	Server Visibility	✗	✓	✗	✓	Diagnostic
Memory Used (%)	Percentage of memory consumed by the top 10 consuming processes or the percentage of memory used by all processes in this class.	Server Visibility	✓	✓	✓	✓	KPI
Memory Used (KB)	Amount of memory used by all processes in this class.	Server Visibility	✓	✓	✓	✓	Advanced
Memory Virtual (KB)	Current size of the virtual address space that the process is using. Use of virtual address space does not necessarily imply a corresponding use of either disk or main memory pages. Virtual space is finite and, by using too much, the process can limit its ability to load libraries.	Server Visibility	✓	✓	✓	✓	Advanced
Thread Count	The number of kernel threads used by all processes in this class.	Server Visibility	✓	✓	✓	✓	Diagnostic

Total Process Count	All individual processes observed by the Machine Agent, *before* it filters out unreported processes and groups the remaining processes into classes. You can use this metric to identify anomalies in the number of processes running on a specific machine. For example, if the average Total Process Count on a machine is usually 200 to 300, and then suddenly rises to 2000 or more, this could indicate a problem on that machine. To collect this metric, the Machine Agent captures the number of processes every 30 seconds and calculates the average number of processes per minute. • Observed is the average number of processes per minute. • Min and Max are the 30-second buckets with the least and most processes. • Count and Sum are the number of process captures each minute and the sum of the processes in each capture for that minute (capture1 + capture2).	Server Visibility	✓	✓	✓	✓	Diagnostic

1 See Linux Kernel Processes.

2 When collecting Process metrics on Solaris, the Machine Agent observes and captures only the first 80 characters of each process name and argument list. The agent considers only the first 80 characters of each process string when it applies allowlists ("always monitor this process") and blocklists ("never monitor this process") defined in ServerMonitoring.yml.

3 See Default Monitoring Mode.

Service Availability Metrics

Reported for each service that is configured.

Metric Name	Description	Basic or Server Visibility	Windows	Linux	Solaris	AIX	Default Monitoring Mode[1]
Response Time (ms)	The elapsed time between sending a request and receiving a response from the monitored service.	Server Visibility	✓	✓	✓	✓	KPI
Response Size (bytes)	The size of the response received from the monitored service.	Server Visibility	✓	✓	✓	✓	KPI
Success Rate (%)	The percentage of successful requests over all requests made to the service.	Server Visibility	✓	✓	✓	✓	KPI

1 See Default Monitoring Mode.

Enable Percentile Metric Reporting

You must enable percentile metric reporting on both the Controller and the Machine Agent. By default, reporting is *disabled* on the Controller and *enabled* o n the Machine Agent.

- To enable/disable reporting on the Controller, log in to the Controller Administration Console and set the `sim.machines.percentile.percentileMonitoringAllowed` property. See Controller Settings for Machine Agents.
- To enable/disable reporting on the agent, open the `<machine_agent_home>/extensions/ServerMonitoring/conf/ServerMonitoring.yml` file and set the `percentileEnabled` property. See Machine Agent Settings for Server Visibility.

Configure Disk Usage Metric Collection on Linux

You can configure the Linux Machine Agent to calculate the following metrics similar to using the Linux `df` command:

- Servers > Volumes > /opt > Disk Usage
- Metric Browser > Disks and Partitions > <partition> > Space Available
- Metric Browser > Disks and Partitions > <partition> > Space Used

Linux includes a mechanism for reserving some disk space to ensure that the system remains functioning even if non-privileged users consume all other disk space. By default, these metrics do not include this reserved space. The total disk space reported by the agent might differ from the total disk space reported by the Linux `df` command or other sources.
To override this default, and to ensure that these metrics include this reserved space, run the agent with the following command-line argument:

```
-Dappdynamics.machine.agent.extensions.calcVolumeFreeAndUsedWithDfCommand=true
```

This feature is supported for Linux versions of the Machine Agent only.

Configure Free/Used Memory Metric Collection on Linux

You can configure how the Machine Agent calculates the amount of free and used memory on Linux machines. By default, the agent calculates any slab-reclaimable memory as used (not free) memory.
To configure the agent to calculate slab-reclaimable memory as free (not used) memory, run Machine Agent with the following command-line argument:

```
-Dappdynamics.machine.agent.extensions.countSlabReclaimableAsFreeMem=true
```

This setting affects the following metrics:

- Memory Usage (in Server Dashboard)
- Memory Used %, Memory Used MB, Memory Free %, and Memory Free MB (in Metric Browser)

Viewing Server Visibility Metrics

You can only view Server Visibility metrics in the Server Visibility version of the Metric Browser. You see this when you access the Metric Browser from the Servers tab in the top navigation bar of the Controller UI.

Machine Agent Versus .NET Machine Agent

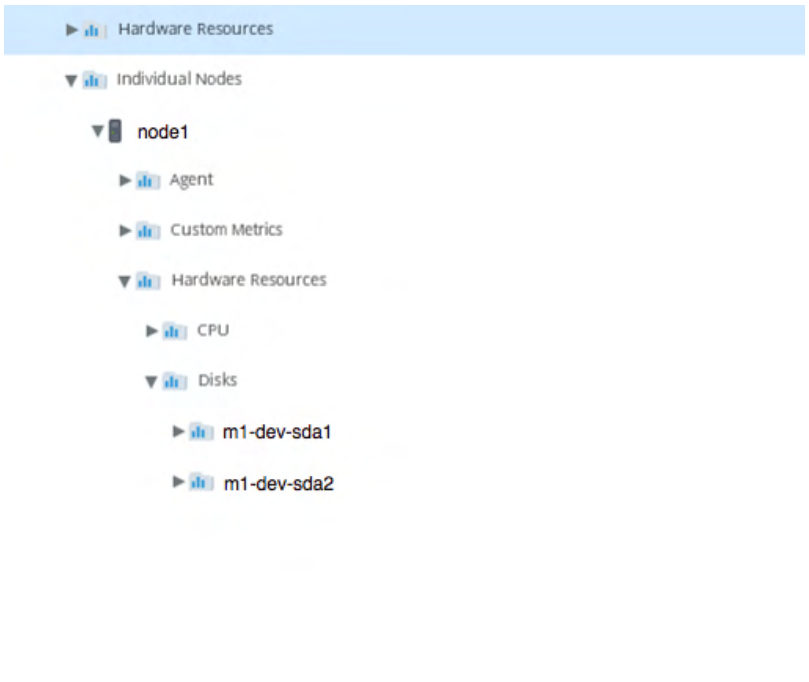
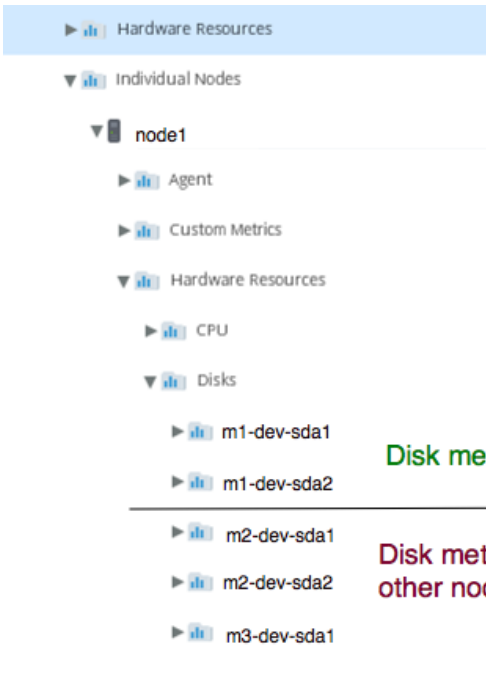
If a server has both Machine and .NET Agents installed, there may be differences in metric values reported by the Machine Agent and the .NET Agent due to different averaging rates and measurement methods. See .NET Compatibility Mode.

Streamlined Browsing Mode

The Server Metric Browser includes a Streamlined Browsing option for node metrics. Use this option to browse metrics for tiers that contain multiple nodes.

- With Streamlined Browsing enabled, each node in the browser tree view shows only metrics reported by the Machine Agent on the corresponding node.
- With Streamlined Browsing enabled, the **Hardware Resources** and **Custom Metrics** folders do not display.
- With Streamlined Browsing disabled, each node in the browser tree view also shows metrics for other nodes in the same tier (with no indication of the node that corresponds to which tier group).

This streamlining option is not enabled by default. AppDynamics recommends that you enable this option. To enable this option, go to: `http://<controller host>:<port>/controller/admin.jsp` and set `sim.metrics.metricBrowser.machineMetricMappings.enabled` to true.

Streamlined Browsing Enabled	Streamlined Browsing Disabled
	

Linux Kernel Processes

The Machine Agent has a processSelectorRegex setting that specifies the set of processes monitored by the agent. The default `regex` filters out most kernel processes on Linux machines. The agent considers monitored processes only when it calculates CPU, Memory, and Process metrics. These metrics may differ from metrics reported by other sources such as Linux commands.

Default Monitoring Mode

The **Default Monitoring Mode** column indicates the default category of each metric when Dynamic Monitoring Mode (DMM) is enabled. When this mode is enabled, a Machine Agent reports metrics based on the DMM setting on that server:

- KPI – Report Key Performance Indicator metrics only
- Diagnostic – Report KPI and Diagnostic metrics
- Advanced – Report all unfiltered metrics on the Machine Agent.

Load Average Percentages are Scaled by 100 in the Metric Browser

Load Average metrics are shown as float values (such as 0.70 or 1.05) in the Server Dashboard. In the Metric Browser, these metrics are multiplied by 100 to provide two decimal points of precision. If a server has an average load of 7.67, for example, the Server Dashboard shows the Load Average as 7.67 and the Metric Browser shows the Load as 767. To use one of the metrics in a health rule or custom dashboard, divide the metric by 100.

Stolen Times are Reported Differently

On Windows Machine Agents, if CPU %Stolen is not matching values reported by AppDynamics, it may be because Windows Performance Monitor's (Perfmon) counters operate at a granularity of 100ns for CPU metrics. For CPU %Stolen, the counters are provided by the Hypervisor, which operates on a granularity of 1ms. Windows Perfmon divides the CPU %Stolen counter (operating on 1ms) directly by time (in units of 100ns), which results in the CPU % Stolen values being reported at a very low percentage (~0.01%).

AppDynamics makes the correction between different units of time, which is why values reported by AppDynamics are greater than Perfmon's values by a factor of 10,000 (time in 1ms * 10,000 = time in 100ns). CPU %Stolen values reported by AppDynamics sometimes exceed 100% under high load. These CPU %Stolen values are a result of multiple cores being used by the guest machine, where %Stolen time is added across multiple cores.

To compute the %Stolen time:

1. On a PowerShell terminal in the monitored guest machine, run `Get-WmiObject Win32_PerfRawData_vmGuestLib_VCPU.`
2. Note the CpuStolenMs counter (in ms).
3. Note the Timestamp_PerfTime counter (in ms, take a look at Frequency_Perftime, this value should be 1000 (in hz)).
4. After a minute, run `Get-WmiObject Win32_PerfRawData_vmGuestLib_VCPU` again and take note of both the counters once more.
5. CPU %Stolen = 100% * (CpuStolenMS (at T2) - CpuStolenMs (at T1) / (Timestamp_PerfTime (T2) - Timestamp_PerfTime (T1)).

HardwareMonitor and JavaHardwareMonitor Calculate "Writes/Sec" Differently on AIX Machines

The Machine Agent calculates the Disk metric writes/sec on AIX machines differently, depending on whether the HardwareMonitor or the JavaHardwareMonitor extension is used. These extensions assume different block sizes when calculating these metrics. For HardwareMonitor, the block size is 4096 bytes. For JavaHardwareMonitor (based on SIGAR), the block size is 512 bytes. Because the number of writes/sec is calculated as <kb_written>/<block_size>, the HardwareMonitor results are lower than those calculated by JavaHardwareMonitor.

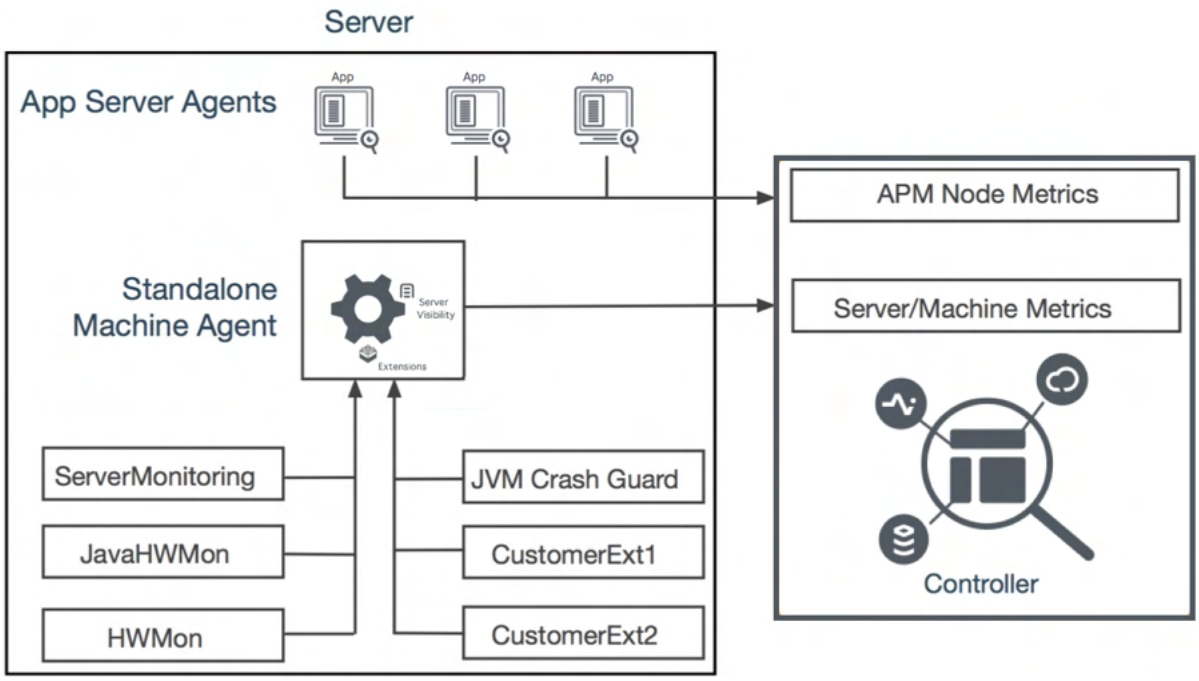
Standalone Machine Agent

You use the Machine Agent to collect basic hardware metrics. The Machine Agent is a Java program that has an extensible architecture enabling you to supplement the basic metrics reported in the AppDynamics Controller UI with your own custom metrics. The Machine Agent is also the delivery vehicle for AppDynamics Server Visibility, which provides an expanded set of hardware metrics and additional monitoring capability. See License Entitlements and Restrictions for a complete list of licenses for Server Visibility and the Machine Agent.

Functionality provided by the Machine Agent includes:

- Reporting basic hardware metrics from the server's OS, for example, %CPU and memory utilization, disk and network I/O
- Reporting metrics passed to the Controller by extensions
- Running remediation scripts for policy actions
- Running JVM Crash Guard

The Machine Agent collects infrastructure metrics from multiple extensions and forwards them to the Controller. You can also use these metrics to find correlations between infrastructure issues on one or more servers and application-performance issues reported by the App Agents.



Standalone Machine Agent Requirements and Supported Environments

This page lists the application environments and versions supported by the Machine Agent. Any environments or versions that are not listed are not supported.

Machine Agent Supported Platforms

Supported platforms and environments for the Machine Agent depend on the metric data collection extension and the machine's OS. See Machine Agent Metric Collection. Machine Agent >= 21.4.0 collects `diskstats` from Linux kernels versions >= 4.18.

The Azul JRE is included as a component in the AppDynamics products and modules. In response to how often Oracle produces and distributes JRE/JDK 8, AppDynamics replaced Oracle JRE with Azul JRE in the 20.5.0 Machine Agent.

JRE Requirements

The Machine Agent requires a Java Virtual Machine (JVM). Downloads for many of the supported operating systems include Azul JRE 1.8.0_262. The Machine Agent should work with most of the JVMs supported by the Java Agent JRE 1.8. However, it is extensively tested only for the Azul JRE.

- For tested platforms listed in the table, AppDynamics recommends using the latest available Machine Agent release from the AppDynamics Downloads page.
- For Linux Flavors, the Machine Agent is bundled with Azul JRE 11. For Windows and Solaris, the Machine Agent is bundled with Azul JRE 1.8

If you upgrade the Linux Machine Agent to 21.1.0, then you must also upgrade all extensions to their latest versions to support Machine Agent 21.1.0.

- To run the Machine Agent on other platforms (such as AIX or HP-UX), use the unbundled Machine Agent ZIP without the JRE. For these platforms, you must install JRE 1.8 on the Machine Agent host.
- If you are using a 64-bit Operating System, use only a 64-bit JRE.

A 64-bit long has a maximum and minimum value of 9223372036854775807 and -9223372036854775808, respectively. To handle large values for metrics, run the Machine Agent using a 64-bit JDK.

Bash Requirements

Linux and Solaris agents require Bash >=3.1.

Required Libraries

Linux systems must include the `procps` library.

This table provides instructions on how to install the library on some common flavors of Linux operating systems.

Linux Flavor	Command
Red Hat and CentOS	Use `yum` to install the library, such as: `yuminstallprocps`
Fedora	Install the library `RPM` from the Fedora website: `yuminstallprocps`
Ubuntu	Use `apt-get`, such as: `sudoapt-getinstallprocps`
Debian	Use a package manager such as `APT` to install the library (as described in the previous Ubuntu instructions).
SUSE 12	Use `zypper` to install the library, such as: `zypperinstallprocps`

Supported Environments

Tested Platforms

OS	Architecture x64 (64-bit)	Architecture SPARC (64-bit)	Versions
AIX	Yes	No	7.1, 7.2
Linux / CentOS	Yes	No	6
	Yes	No	7
Linux / Debian	Yes	No	8
	Yes	No	9

Linux / Fedora	Yes	No	28
	Yes	No	29
Linux / openSUSE Leap	Yes	No	42.3
	Yes	No	15.0
Linux / Red Hat Enterprise Linux	Yes	No	6
	Yes	No	7
	Yes	No	8
Linux / SUSE Linux Enterprise	Yes	No	12
	Yes	No	15
Linux / Ubuntu	Yes	No	14.04
	Yes	No	16.04
	Yes	No	18.04
Solaris	Yes	Yes	10
	Yes	Yes	11
Windows	Yes	No	Windows Server 2012 R2
	Yes	No	Windows Server 2016
	Yes	No	Windows 2019

Other Platforms

These other platforms (supported by JRE 1.8), should also be compatible with the Machine Agent. However, AppDynamics has not fully tested them.

- Oracle JRE 1.8
- IBM SDK, Java Technology Edition, Version 8
- HP JDK/JRE 8

You should be able to run the Bundled version of the Machine Agent on machines based on Power Architecture processors, including PowerPC processors, but Server Visibility is not supported on these platforms. For OS platforms like AIX or HP-UX, AppDynamics recommends using the unbundled Machine Agent ZIP without the JRE on machines with Power Architecture processors, including PowerPC processors.

Hardware and Sizing Requirements

Agent: One additional GB of RAM.

Controller: Although we recommend that you install the AppDynamics Controller on a dedicated server, in some cases the Machine Agent can co-exist with the Controller on the same system. The exact number of Machine Agents that can be supported depends on whether other agents are reporting to the same Controller and whether the Machine Agents have Server Visibility enabled (more metrics are generated under Server Visibility). See Controller System Requirements.

JVM Memory Requirements

The lightweight Machine Agent consumes minimal resources of your computer.

AppDynamics recommends the following additional Heap and PermGen space to accommodate the agent:

- `Maximum heap size (-Xmx): 256 MB`
- `Maximum PermGen heap size (-XX:MaxPermSize): 20 MB`

By default, the JVM maximum heap size is set to 256 MB. To override this setting, you can set this flag through the `JAVA_OPTS` environment variable on your system. For example, if you enter: `export JAVA_OPTS="-Xms20m -Xmx128m"` on Linux, this sets the initial heap size to 20 MB and the max heap size to 128 MB.

Extension Considerations

The exact CPU or memory overhead added by the agent can vary based on the extensions used and whether the extensions are Java. Java extensions require more heap space; the amount required depends on how you code the extension. If you start adding extensions, you should increase the maximum heap space to 256 MB or 512 MB. Additionally, you may need to increase the size of the initial memory allocation. Monitor the memory consumption of the agent to ensure that there are sufficient resources allocated to it. You can also enable Garbage Collection logging on the JVM to help tune the heap size for the extensions.

Example settings for increasing the initial heap size:

Linux and Unix systems: `<machine_agent_home>/bin/machine-agent -Xms64m`

Windows: `cscript <machine_agent_home>\machine-agent.vbs -Xms64m`

Machine Agent Metric Collection

The Standalone Machine Agent collects hardware metrics using default extensions appropriate to specific operating systems. The following table lists the metric collection extension and its supported OS information. In some limited cases, you may want to change the default collection extension. The table also lists the most common reasons for changing the default extension.

Supported environments, observation rates, configurability, some metric names, and definitions depend on the extension.

OS	Default Metric Collection Extension	Metrics Collected	Reason to Change the Default Extension	Supported Environments
Microsoft Windows	JavaHardwareMonitor (if Server Visibility is disabled) ServerMonitoring (if Server Visibility is enabled)	Basic Metrics Server Visibility Metrics	To Customize Metrics for Virtual Disks and External Network Traffic The HardwareMonitoring extension is not recommended on Windows. See Monitoring Windows Guidelines	Standalone Machine Agent Supported Environments
Linux	ServerMonitoring	Basic Metrics Server Visibility Metrics	To Customize Metrics for Virtual Disks and External Network Traffic	Standalone Machine Agent Supported Environments
Solaris / SunOS	HardwareMonitor (if Server Visibility is disabled) ServerMonitoring (if Server Visibility is enabled)	Basic Metrics Server Visibility Metrics	To Customize Metrics for Virtual Disks and External Network Traffic	Standalone Machine Agent Supported Environments
AIX	JavaHardwareMonitor	Basic Metrics	SIGAR (System Information Gatherer And Reporter) is not supported for your OS or Linux Distribution	SIGAR
HP-UX	JavaHardwareMonitor	Basic Metrics	SIGAR is not supported for your OS or Linux Distribution	SIGAR
Mac OS X	JavaHardwareMonitor	Basic Metrics	SIGAR is not supported for your OS or Linux Distribution	SIGAR
Solaris	HardwareMonitor (if Server Visibility is disabled) ServerMonitoring (if Server Visibility is enabled)	Basic Metrics Server Visibility Metrics	To Customize Metrics for Virtual Disks and External Network Traffic	Standalone Machine Agent Supported Environments

In case of a time discrepancy issue on a server (not related to daylight savings time), where time is reset manually, the Machine Agent stops reporting. During this interruption, application and server data are not reported. To continue monitoring, you must restart the Machine Agent.

Java Hardware Monitor

The JavaHardwareMonitor is based on SIGAR (System Information Gatherer And Reporter). SIGAR is a legacy method of collecting basic hardware metrics and is used in pre-4.1 versions and for machines running operating systems that are not supported by the ServerMonitoring extension. The following sections list scenarios where you might want to change the extension used to collect the machine metrics.

Customize Metrics for Virtual Disks and External Network Traffic

By default, the Standalone Machine Agent reports metrics for network-mounted and local disks only. Additionally, only the external network traffic is aggregated (to ensure backward compatibility with previous versions of AppDynamics). For operating systems using the ServerMonitoring extension, you can change to the JavaHardwareMonitor to configure specific disks and network interfaces to be monitored. See Configure Metrics for Virtual Disks and External Network Traffic - JavaHardwareMonitor Extension Only to customize the behavior of the JavaHardwareMonitor metrics

Metric Observation Rate

The JavaHardwareMonitor takes observations of metrics in two distinct ways:

- For disk and network metrics: One observation per minute. When you view the count for these metrics in the Metric Browser, you see the count of 1 per minute. If you select a 15-minute interval, the count would be 15 times 1 = 15 and so on.
- For CPU and memory metrics: One observation every two seconds. When you view the count for these metrics, you see the count of 30 per minute. If you select a 15-minute interval, the count would be 15 times 30 = 450 and so on.

Collect Basic Metrics Using JavaHardwareMonitor

Linux and Windows machines use the ServerMonitoring extension by default to report basic metrics. However if you have Server Visibility enabled, falling back to the JavaHardwareMonitor only affects the collection of the basic hardware metrics. The extended Server Visibility metrics are still collected correctly by the ServerMonitoring extension.

How to Change from ServerMonitoring to JavaHardwareMonitor

1. Stop the Machine Agent.
2. Disable basic ServerMonitoring:
    a. Edit the `ServerMonitoring.yml` file from `<machine_agent_home>/extensions/ServerMonitoring/conf/`.
    b. Change the value of `basicEnabled` to "`false`".
    The `basicEnabled` setting controls whether the Machine Agent reports basic hardware metrics through the ServerMonitoring extension. Setting this to `false` enables the JavaHardwareMonitor to report the basic hardware metrics using the legacy SIGAR-based reporting.
    c. Save the file.
3. Enable JavaHardwareMonitor:
    a. Edit `monitor.xml` from `<machine_agent_home>/monitors/JavaHardwareMonitor/`.
        - For Linux, remove this line: `<enable-override os-type="linux">false</enable-override>`.
        - For Windows, remove this line: `<enable-override os-type="windows">false</enable-override>`.
    b. Save the file.
4. Restart the Machine Agent.

Hardware Monitor

The HardwareMonitor extension is a collection of OS-specific scripts. SunOS and Solaris machines use this extension by default. For the operating systems that use the JavaHardwareMonitor by default, you may want to switch to an OS-specific monitor. You can switch to the OS-specific monitors when the JavaHardwareMonitor fails to report statistics and you see error logs similar to the following:

```
[Agent-Scheduler-1] 28 Nov 2013 13:13:55,435 ERROR SigarMinuteTask - Error fetching network statistics for
interface [eth0:17]. Blacklisting it.
[Agent-Scheduler-1] 28 Nov 2013 13:13:55,435 ERROR SigarMinuteTask - Error fetching disk i/0 statistics for
interface [eth0:17]. Blacklisting it.
```

Collect Basic Hardware Metrics using HardwareMonitor

How to Change from JavaHardwareMonitor to an OS-Specific Monitor

1. Stop the Machine Agent.
2. Disable the JavaHardwareMonitor:
    a. Locate and edit `monitor.xml` from `<machine_agent_home>/monitors/JavaHardwareMonitor/`.
    b. Change the value `<enabled>true</enabled>` to `<enabled>false</enabled>`.
    c. Save the file.
3. Enable the HardwareMonitor for the OS you want to monitor:
    a. Edit `monitor.xml` from `<machine_agent_home>/monitors/HardwareMonitor/.`
    b. Change `<enabled>false</enabled>` to `<enabled>true</enabled>`.
    c. Save the file.
4. Restart the machine agent.

Modify Free Memory Metric Configuration

If your Machine Agent installation is using an OS-specific HardwareMonitor for metric collection, then by default the agent reports free memory as the memory that is not used by any process nor in an I/O buffer or cache. It is more useful for the free memory metric to include memory that is in an I/O buffer or cache but can be available for new processes.

To include the memory in I/O buffers or cache that can be made available to new processes, modify the HardwareMonitor configuration:

1. Open `<machine_agent_home>/monitors/HardwareMonitor/config.sh`.
2. Set `REPORT_MEMORY_FREE_AS_MEMORY_AVAILABLE` to 1.
3. Restart the agent.

Monitoring Windows Guidelines

When using the Machine Agent on Windows:

- Check the server frequently to ensure that it has the latest Windows updates installed.
- If you have a Server Visibility license, use the ServerMonitoring extension.
- If you do not have a Server Visibility license, or if you want to collect Basic metrics only, use the JavaHardwareMonitor extension.
    - There is a known Windows issue when using the HardwareMonitor extension to collect metrics on Windows 8: The "Win32_Service" WMI class leaks memory in Windows Server 2008 R2 and in Windows 7. As a result, AppDynamics does not recommend using the HardwareMonitor extension on Windows. You should use JavaHardwareMonitor instead.
    - However, if you must use the HardwareMonitor extension on Windows, apply this Microsoft hotfix to the server: https://support.microsoft.com/en-us/kb/981314

Server Visibility on Windows

See Server Visibility Requirements and Supported Environments.

Running the Machine Agent and .NET Agent on the Same Machine

To collect Machine or Server metrics on a machine with Machine and .NET Agents installed, you must enable .NET Compatibility Mode on both the Controller and the Machine Agent. See .NET Compatibility Mode.

Configure Metrics for Virtual Disks and External Network Traffic - JavaHardwareMonitor Extension Only

By default, the Machine Agent reports metrics for network-mounted and local disks only. Additionally to ensure backward compatibility with previous versions of AppDynamics, only the external network traffic is aggregated.

You can customize the default behavior by modifying the auto-generated configuration file, `task-template.xml`. The `task-template.xml` file provides information about the current configuration of the Machine Agent and is created when you first start up the agent. It discovers the disks and network hardware on the machine where it is installed that the agent can monitor.

This configuration applies to the JavaHardwareMonitor extension only. The ServerMonitoring extension is recommended if you have a Server Visibility license and the monitored system meets the system requirements. See Server Visibility Requirements and Supported Environments

Customize the Default Machine Agent Metric Collection

1. Locate the `task-template.xml` file.
   This file is located in the `<machine_agent_home>/monitors/JavaHardwareMonitor/` directory when the Machine Agent starts the first time.
2. Edit the `task-template.xml` file:

   - To enable aggregation operation for localhost (lo) network metrics, change the value of the aggregate attribute (for the network element "lo") to "true".
   - To enable monitoring for a virtual disk, set the value of the enabled attribute to "true" for that disk.

   The following is a sample `task-template.xml` file:

```
<config>
    <disk aggregate="false" enabled="false">sunrpc</disk>
    <disk aggregate="true" enabled="true">/dev/sdb1</disk>
    <disk aggregate="false" enabled="false">proc</disk>
    <disk aggregate="false" enabled="false">none</disk>
    <disk aggregate="false" enabled="false">devpts</disk>
    <disk aggregate="true" enabled="true">/dev/sda1</disk>
    <disk aggregate="false" enabled="false">nfsd</disk>
    <disk aggregate="true" enabled="true">/dev/mapper/saas4-binlog</disk>
    <disk aggregate="false" enabled="false">sysfs</disk>
    <disk aggregate="false" enabled="false">tmpfs</disk>

    <network aggregate="true" enabled="true">lo</network>
    <network aggregate="true" enabled="false">sit0</network>
    <network aggregate="true" enabled="true">eth0:1</network>
    <network aggregate="true" enabled="true">eth0</network>
    <network aggregate="true" enabled="false">eth1</network>
</config>
```

3. To prevent the Machine Agent from overwriting the file, rename the `task-template.xml` file to `task.xml` .
4. To monitor a special device that is not enabled, add a file named `task.xml` in the `<machine_agent_home>/monitors/JavaHardwareMonitor/` directory.
   The format of the `task.xml` file must be exactly the same format as the `task-template.xml` file.
   Not all disks and networks must be included in the `task.xml` file. If the Machine Agent finds a disk or a network that is not in the `task.xml` file, then it applies the default properties.
5. Restart the Machine Agent.

View Hardware Metrics

Without having to enable Server Visibility, you can view basic machine metrics in these locations:

- **Metric Browser** – Go to **Application Infrastructure Performance** > **Hardware Resources**.
- **Node** page > Server tab – Go to **Tiers & Nodes**, select the node of interest, and go to the Server tab.
- **Custom Dashboards** – To add CPU metrics to your Custom Dashboard, add the metrics for any node on your target machine. Metrics collected for a node associated with the machine agent are for the server hosting that node. Any machine metrics collected will be the same for nodes that are on the same machine.

Server Visibility provides additional metrics and pages to view basic and server metrics. See Monitor Your Servers Using Server Visibility.

Hardware Resources Metrics describes metrics collected by the Machine Agent.

Customize Default Metrics

You can enable collection of metrics for virtual disks and external network traffic.

Add Custom Metrics

You can add script-based and Java-based custom monitoring extensions to collect additional metrics. For example, the ehCache monitoring extension available from AppDynamics Exchange can collect metrics to monitor Ehcache performance. The metrics appear in **Application Infrastructure Performance | Custom Metrics** section of the Metric Browser and which you can use on custom dashboards.

You can also send metrics to the Machine Agent using the HTTP listener. See Extensions and Custom Metrics.

Install the Standalone Machine Agent

This page provides requirements, permissions, and a procedure to install the Machine Agent. You can also use the AppDynamics Agent Installer to streamline the deployment of the Machine Agent. The Agent Installer simplifies the agent installation process.

To install Machine Agents in Kubernetes, see Install Infrastructure Visibility with the Cluster Agent Operator.

Make sure to thoroughly test your deployment in a staging or test environment before deploying it to production.

JRE Requirements

- JRE 1.8 is required.
- In 4.2, JRE 1.8 is bundled with the OS-specific Machine Agent installation downloads.
- The Machine Agent starts within its own JVM.
- You can use an existing JRE previously installed in your environment.

Permissions

- To avoid permission issues, install the agent as the same user who owns the Machine Agent files or as an administrator on the host machine.
- All files in the `<machine-agent-home>` installation directory should be readable by the Machine Agent.
- The user that runs the Machine Agent must have write privileges to the logging output directory and to the `/conf` directory in the agent installation directory.
- If you plan to enable the JVM Crash Guard, review the required permissions for JVM Crash Guard.
- To create a non-root user to run the Machine Agent, see Permissions Required to Run the Machine Agent.
- Windows permissions for files and subfolders are inherited by default from the parent folder (`<machine_agent_home>`). Appdynamics recommends that you restrict permissions to users authorized to start, stop, and configure the Machine Agent:
    - Read and Write permissions to all files and subfolders under `<machine-agent-home>`.
    - If running as a:
        - Terminal application, then restrict Read, Write, and Execute permissions for the file `<machine-agent-home>\bin\machine-agent.vbs`.
        - Service, then restrict Start, Stop, and Restart permissions for the Machine Agent service. You only need admin privileges to install the service. The Machine Agent runs under the local system account which has extensive privileges on the local system. However, if WMI access is revoked, then you must run the Machine Agent as Administrator. Typically, users do have WMI access. See User Account Control and WMI.
- Enable Windows Script Host for the Windows Machine Agent.

Installation Directory and Path Name Requirements

The following table lists characters that are not supported for the `<machine-agent-home>` directory name or in any directory in the path:

Not Supported on any OS	Not Supported on Linux	Not Supported on Windows
Whitespace characters	% #	% # \ / : * ? " < > |
An exclamation point at the end of the directory name		Control characters such as ASCII EOL, CR, etc.

Configuration

The `controller-info.xml` file contains the properties used to specify agent to Controller communications. To specify the Controller host, port, and account access key properties, see Plan the Machine Agent Configuration. Based on your installation requirements, you may need to configure additional properties.

Installation for New Standalone Machine Agent

The following steps are for *new* installations. To upgrade Machine Agent <= 4.3, see Upgrade the Machine Agent.

1. Plan the Machine Agent Configuration.
2. From AppDynamics Downloads, download the Machine Agent installation package for your OS environment onto the machine you want to monitor.

If there is no download bundle for your OS, use the Machine Agent zip file without the JRE, and use a separately downloaded JRE to run it. You need to download a separate JRE if the one already installed on the machine is a version earlier than JRE 1.8.

- Install Using the Non-JRE Zip File - Use this for OS environments other than Linux, Solaris, Windows, and Mac OS X.

    - Linux Install Using the RPM Package
    - Linux Install Using ZIP with Bundled JRE
    - Solaris Install Using ZIP with Bundled JRE
    - Windows Install Using ZIP with Bundled JRE
3. Verify the Machine Agent Installation.
4. Start the Machine Agent by executing `<machine_agent_home>/bin/machine-agent`. If you have `java` or `system` properties, you can add them to the end of the command. To review machine usage, enter `machine-agent -h`.

Standalone Machine Agent Installation Scenarios

Related pages:

- Plan the Machine Agent Configuration
- Where to Specify Machine Agent Configuration
- Machine Agent Configuration Properties

This page describes scenarios for new Machine Agent installations.

APM Machine Agent Installation Scenario

This scenario contains a host server running one or more instrumented applications and one Machine Agent. When you install the Machine Agent on the same server with any app agent, do not specify the application name and tier name. The one Machine Agent reports the hardware metrics to each node where the app agent and the Machine Agent have matching host IDs.

If you plan to enable Server Visibility on a Windows server where a .NET APM agent is installed, you must enable .NET Compatibility Mode on both the Controller and the Machine Agent. See .NET Compatibility Mode.

> **Licensing Note**
>
> >= 4.3 includes one Machine Agent license with each APM app agent. You can install this Machine Agent only on the same server with the app agent.

Independent Machine Agent Installation

This scenario contains a host server running only the Machine Agent without app agents. You do not need to specify an application or tier. You can view the Machine Agent metrics from the Servers list or the metric browser. This scenario requires a Server Visibility license.

> **Licensing Note**
>
> >= 4.3 requires a Server Visibility license for a Machine Agent installation.

If you are doing an Independent Machine Agent installation and you want the hardware metrics reported to nodes in an application, then you must configure the application name and tier name where you want the metrics to appear. The node name defaults to Node1 for the machine.

Using the values for the application name and tier where you want to see the metrics reported, configure these properties:

- Application name
- Tier name
- `uniqueHostId` for both the machine agent and app agent: use the same value

Unique Host ID Property

The `uniqueHostId` property is not required. However, if you do not define `uniqueHostId`, then the Machine Agent uses the Java API to retrieve the host ID. The results from the API can be inconsistent. The same JVM may return a different value for the same machine each time you restart the Machine Agent. As a workaround, AppDynamics recommends that you set the value of `uniqueHostId` to the host ID that you want to see in the UI. Use the same value for `uniqueHostId` for the App Agent.

Plan the Machine Agent Configuration

AppDynamics provides multiple ways to configure the Machine Agent. As part of planning your installation, determine which options work best in your environment:

- Controller-info.xml
- System properties
- Environment variables

See Configure the Machine Agent.

Collect Configuration Information

Determine your configuration requirements and collect the necessary information. See Machine Agent Configuration Properties.

Planning Item	Description
Determine your installation scenario	For new installations, see Machine Agent Installation Scenarios. For upgrades, see Upgrade the Machine Agent.
Location of the startup script	You add startup arguments and system properties in the startup script. If you are using a Java service wrapper, you need to know the location of the wrapper configuration. See Where to Specify Machine Agent Configuration.
Memory and CPU	See Machine Agent Requirements and Supported Environments.
Connecting the agent to the Controller	You need to know the Controller host, Controller port, and account access key where you want your metrics to be reported. The account access key is generated at the Controller installation time and is located on the AppDynamics Administration Console.
(Optional) SSL Communications	If you plan to use SSL to communicate with the Controller, see Enable SSL for Machine Agent. SSL-related properties include: • Controller SSL Enabled Property • Controller Keystore Password • Controller Keystore Filename • Force Default SSL Certificate Validation
Proxy Settings	If you plan to connect to the Controller using a proxy server, you need to know the Proxy host, Proxy port, and the path to the Proxy Password file.
Multi-tenant mode or SaaS Installations	In addition to the Account Access property, use the Account Name property to configure multi-tenant mode or SaaS account Information. This information is provided in the Welcome email from the AppDynamics Support Team. You can also locate this information in the `<controller_home>/initial_account_access_info.txt file.`
Deploying from a common directory	See Deploy Multiple Machine Agents From a Common Directory.
Extension considerations	If you plan to use a number of extensions in your environment, see the "Extension Considerations" section in Machine Agent Requirements and Supported Environments.
Enabling Server Availability and Service Availability Monitoring	Server Availability requires a separate license. See Enable Server Visibility.

Install Using the Non-JRE Zip File

This page describes how to install the Machine Agent using the zip file that does not include the JRE. You can use this zip file if your environment already has Java >= 1.8, or when installing on an OS environment that does not have an OS-specific installation package on the AppDynamics download site. The agent uses the JRE specified in the `$JAVA_HOME` environment variable or the default JRE for the server (enter `which java` in a terminal window).

Install the Machine Agent

1. Before installing, review Install the Machine Agent.
2. Download and unzip the ZIP file. Extract the contents to the agent installation directory, *<machine_agent_home>*.
3. Gather your configuration details and configure the agent by editing the `<machine_agent_home>/conf/controller-info.xml` file or by adding system properties to the JVM startup script file. See Plan the Machine Agent Configuration.
    a. (*Required*) Configure the Controller host name, port number, and account access key.
    b. (*Required for Multi-Tenant Mode or SaaS installations*) Configure the Agent Account Information. See Multi-Tenant Controller Accounts.
    c. (*Optional*) Review memory requirements. See Machine Agent Requirements and Supported Environments.
    d. (*Optional*) Configure the agent to use SSL. See Enable SSL for Machine Agent.
    e. (*Optional*) Configure the agent to use proxy settings. See Machine Agent Configuration Properties.
    f. (*Optional*) For application and tier name, see Machine Agent Installation Scenarios to determine if you need to specify an application name and tier name. If you are installing the Machine Agent on the same server with any APM app agent, do not specify the application name and tier name.
4. Start the Agent:
    - Windows in a terminal window: <machine_agent_home>`\bin\machine-agent.vbs`
    - Windows as a service: <machine_agent_home>`\InstallService.vbs`
    - Linux or Unix-like: <machine_agent_home>`/bin/machine-agent`

Linux Install Using the RPM Package

Related pages:

- Where to Specify Machine Agent Configuration
- Start and Stop the Machine Agent

This page describes how to install the Standalone Machine Agent on Linux systems that support the RPM Package Manager:

- CentOS
- RHEL
- Fedora
- openSUSE
- SUSE Linux Enterprise Server

For Linux systems that do not support RPM, use the JRE Bundled Zip Archive.

The RPM installer makes these changes to the host machine:

- Creates an appdynamics-machine-agent group and an appdynamics-machine-agent user
- Assigns ownership of certain files in the machine-agent directory to the appdynamics-machine-agent user

These changes are necessary to enable non-root users to configure and run an RPM-installed agent. To use a different user or group for the machine agent service, set the MACHINE_AGENT_USER and MACHINE_AGENT_GROUP environment variables in a shell for RPM installation.

```
sudo MACHINE_AGENT_USER=myuser MACHINE_AGENT_GROUP=mygroup rpm -ivh appdynamics-machine-agent.rpm
```

If the specified user or group does not exist, an error message appears and the RPM installation stops. To continue, you must define users and groups. If this is an issue in your environment, then you install the agent using the ZIP archive (see Linux Install Using ZIP with Bundled JRE).

Install the Machine Agent

1. Before installing, review Install the Machine Agent.
2. Download and install the RPM Package. With administrative privileges, enter the following CLI code where <pkg-name.rpm> is the name of the package for your environment, such as `appdynamics-machine-agent-<version>.x86_64.rpm`.

   ```
   sudo rpm -ivh <pkg-name.rpm>
   ```

   The agent files are installed in `opt/appdynamics/machine-agent` and the agent is added as a service.
3. Gather your configuration details and configure the agent by editing<machine_agent_home>/conf/controller-info.xml file or by adding system properties to the JVM startup script file.
   See Plan the Standalone Machine Agent Configuration.

   a. (*Required*) Configure the Controller host name, port number, and account access key.
   b. (*Optional*) Review memory requirements. See Machine Agent Requirements and Supported Environments.
   c. (*Optional*) Configure the agent to use SSL. See Enable SSL for Machine Agent.
   d. (*Optional*) Configure the agent to use proxy settings. See Machine Agent Configuration Properties.
   e. (*Required for Multi-Tenant Mode or SaaS installations*) Configure the Agent Account Information. See Multi-Tenant Controller Accounts.
   f. (*Optional*) Determine if you need to specify an application name and tier name. See Machine Agent Installation Scenarios.
   If you are installing the Standalone Machine Agent on the same server with any APM app agent, do not specify application name and tier name.
4. Start the Agent. You can start the agent as a service (requires sudo or root user) or from the command line. If `systemd` was detected during installation, you can use the `systemctl` command to start the agent service.
   - Using SysV – `service appdynamics-machine-agent start`
   - Using systemd – `systemctl start appdynamics-machine-agent`
   - From the Command Line – <machine_agent_home>`/bin/machine-agent`
   - From the Launcher – <machine_agent_home>`/etc/init.d/appdynamics-machine-agent start`

RPM Package Function

The RPM package manager installs the agent files to `opt/appdynamics/machine-agent`, creates symbolic links, sets environment variables, and adds the agent as a service.

Symbolic Links

The RPM package manager creates the symbolic links to agent scripts and configuration files in the `/etc` directory. You can edit these links to accommodate a different installation directory, JRE, or system user account running the agent.

Link for SysV Service Script

`/etc/init.d/appdynamics-machine-agent >> /opt/appdynamics/machine-agent/etc/init.d/appdynamics-machine-agent`

This script provides these service commands:

- `Start` – Starts the service, runs a script that starts the agent
- `Stop` – Stops the service
- `Restart` – Restarts the service
- `Status` – Returns the status of the service.

For example: `service appdynamics-machine-agent start`

Link for Environment Variables

SysV –

```
/etc/sysconfig/appdynamics-machine-agent >> /opt/appdynamics/machine-agent/etc/sysconfig/appdynamics-machine-agent
```

This script sets up the environment variables:

- `MACHINE_AGENT_HOME=/opt/appdynamics/machine-agent` – Specifies where the agent files are located.
- `JAVA_HOME=/opt/appdynamics/machine-agent/jre` – Specifies the JRE the agent uses. AppDynamics recommends that you use the RPM package installer that contains its own JRE. However, you can use a different JRE as long as it meets the JRE requirements.
- `MACHINE_AGENT_USER=root` – Specifies the system user for starting the agent (default is root). AppDynamics recommends that you create a non-root user to run the machine agent. The new user needs to have read-access to controller-info.xml and write access to the log file. See Permissions Required to Run the Machine Agent

systemd – The environment variables are in the service file:

```
/etc/systemd/system/appdynamics-machine-agent.service >> /opt/appdynamics/machine-agent/etc/systemd/system/appdynamics-machine-agent.service
```

"MACHINE_AGENT_USER" variable doesn't exist in the `systemd` service file. The variable is just `User`.

Link for Agent to Controller Communication

Sets up a link to the controller-info.xml file containing the properties for agent-to-controller communication.

```
/etc/appdynamics/machine-agent/controller-info.xml >> /opt/appdynamics/machine-agent/conf/controller-info.xml
```

Logging Configuration

The log4j.xml file controls the detail of information logged by the agent. By default, the logging level is set to `info`.

```
/etc/appdynamics/machine-agent/logging/log4j.xml >> /opt/appdynamics/machine-agent/conf/logging/log4j.xml
```

Adds the Agent as a Service

After the package is installed, the appdynamics-machine-agent runs `opt/appdynamics/machine-agent/bin/postInstall.sh` to add the agent to the services. The installer either copies the SysV script to add the service or, if `systemd` is detected, the installer copies the agent unit file `/etc/systemd/system/appdynamics-machine-agent.service` and adds the agent to the services using this unit file.

Installs a systemd Unit File for the Agent Service

If `systemcd` is detected on the system when you install the RPM package or run the `postinstall.sh` script, the `systemd` unit file for the agent service is copied to the following location: `/etc/systemd/system/appdynamics-machine-agent.service`

ExecStart: This option in the `systemd` service file points to a script that starts the agent as a daemon. If you did not install the Machine Agent in `/opt/appdynamics/machine-agent`, then change the path to `<machine-agent-home>/scripts/machine-agent-daemon`. If you did not edit the path, then it points to the correct place by default.

You can start and stop the agent service using the relevant `systemctl` commands. For information on all `systemctl` commands, see the systemctl man pages.

Command	Format

`start`
`stop`
`status`
`restart`
	`systemctl <command> appdynamics-machine-agent.service`

`disable`
`enable`
(Use the full path)
	`systemctl <command> /etc/systemd/system/appdynamics-machine-agent.service`

Linux Install Using ZIP with Bundled JRE

This page describes how to install the Standalone Machine Agent using the Linux ZIP archive that includes JRE 1.8.

It is easier to download and use the RPM package if your Linux distribution supports the RPM Package Manager.

Install the Agent

1. Before installing, see Install the Machine Agent.

   > Read/write privileges to the <machine-agent-home> installation directory are required.

2. Download and unzip the Agent zip bundle. From the command line, enter:

   ```
   unzip <zip-bundle.zip> -d <machine_agent_home>
   ```

   where <zip-bundle.zip> is the name of the zip archive for your environment and <agent_home> is the name of the directory where you want to install the agent.
3. Gather your configuration details and configure the agent by editing<machine_agent_home>/conf/controller-info.xml file or by adding system properties to the JVM startup script file. See Plan the Standalone Machine Agent Configuration.

   a. (*Required*) Configure the Controller host name, port number, and account access key.
   b. (*Optional*) Review memory requirements. See Machine Agent Requirements and Supported Environments.
   c. (*Optional*) Configure the agent to use SSL. See Enable SSL for Machine Agent.
   d. (*Optional*) Configure the agent to use proxy settings. See Machine Agent Configuration Properties.
   e. (*Required for Multi-Tenant Mode or SaaS installations*) Configure the Agent Account Information. See Multi-Tenant Controller Accounts.
   f. (*Optional*) Determine if you need to specify an application name and tier name. See Standalone Machine Agents and Applications.
   If you are installing the Standalone Machine Agent on the same server with any APM app agent, do not specify application name and tier name.
4. To add the Machine Agent as a service, see Add the Agent as a Service.
5. Start the Machine Agent by entering: `<machine_agent_home>/bin/machine-agent`. For example, from the <machine_agent_home> directory:

   ```
   ./bin/machine-agent
   ```

   To review Machine Agent usage, enter: `machine-agent -h`

   ```
   Usage: machine-agent [-dh] [-j JAVA_HOME] [-p pidfile] [-D prop] [-X prop]
   Start the machine agent.
       -d            daemonize (run in background)
       -p pidfile    write PID to <pidfile>
       -h
       --help        print command line options
       -D prop       set JAVA system property
       -X prop       set non-standard JAVA system property
   ```

6. (Optional) Verify the installation. See Verify the Machine Agent Installation.

Add the Agent as a Service Using SysV

1. Install the Standalone Machine Agent.
2. Create a link to `/etc/sysconfig`:

   ```
   ln -s <machine-agent-home>/etc/sysconfig/appdynamics-machine-agent /etc/sysconfig/appdynamics-machine-agent
   ```

3. Copy the machine agent to `/etc/init.d`. For example:

   ```
   cp <machine-agent-home>/etc/init.d/appdynamics-machine-agent /etc/init.d/appdynamics-machine-agent
   ```

4. Edit the environment variables in `/etc/sysconfig/appdynamics-machine-agent` configuration file:

   - MACHINE_AGENT_HOME – Specifies where the Machine Agent files are located.

- JAVA_HOME – Specifies the JRE the agent uses. AppDynamics recommends that you use the Machine Agent bundled with the JRE. However, you can use an existing JRE. (1.8 or later).
- MACHINE_AGENT_USER – Specifies the system user used to start the Standalone Machine Agent. By default this is root. AppDynamics recommends that you create a non-root user to run the Machine Agent. The new user needs to have read-access to controller-info.xml and write access to the log file. See Permissions Required to Run the Machine Agent.

5. Add the agent as a service. For example, enter:

```
chkconfig --add appdynamics-machine-agent
```

   For Ubuntu, you can use `update-rc.d` or `sysv-rc-conf`. See the Ubuntu documentation for details.
6. Start the agent service: `service appdynamics-machine-agent start.`
7. Verify that the Agent is reporting to the Controller. See Verify the Machine Agent Installation.

Add the Agent as a Service Using systemd

1. Install the Standalone Machine Agent.
2. Edit the environment variables and options in the service file as needed: <*machine-agent-home*>`/etc/systemd/system/appdynamics-machine-agent.service`
   a. MACHINE_AGENT_HOME – Specifies where the Machine Agent files are located.
   b. JAVA_HOME – Specifies the JRE the agent uses. Appdynamics recommends that you use the Machine Agent bundled with the JRE. However, you can use an existing JRE.
      – JRE >= 1.7 for = 4.3
      – JRE >= 1.8 for >= 4.4
   c. User – This option in the service file specifies the system user to use to start the Standalone Machine Agent. By default this is root. AppDynamics recommends that you create a non-root user to run the Machine Agent. The new user needs to have read access to controller-info.xml and write access to the log files. See Permissions for Non-Root User to Run the Machine Agent .
      The `systemd` service file does not include the "MACHINE_AGENT_USER" variable.
3. Copy the file:

```
cp <machine-agent-home>/etc/systemd/system/appdynamics-machine-agent.service \
/etc/systemd/system/appdynamics-machine-agent.service
```

4. Enable the Machine Agent to start at system startup:

```
systemctl enable appdynamics-machine-agent
```

5. Start the agent service:

```
systemctl start appdynamics-machine-agent
```

6. Check the service status:

```
systemctl status appdynamics-machine-agent
```

7. Verify that the Agent is reporting to the Controller. See Verify the Machine Agent Installation.

Solaris Install Using ZIP with Bundled JRE

This page describes how to install the Machine Agent on 64-bit Solaris using the ZIP archive that includes JRE 1.8.

> Adding the Machine Agent as a service is not supported on Solaris.

Install the Agent

1. Before installing, see Install the Machine Agent.
2. The Machine Agent on Solaris is supported only when you install it in Global zones.
3. The Solaris agent requires that the `nawk` (new awk) utility is installed. Verify that the Solaris host has the `nawk` utility installed.
4. Download one of the two installation Machine Agent Bundles for Solaris from the AppDynamics Download Site:
   - For Sparc machines: Machine - Agent Bundle - 64-bit `solaris-sparcv9.zip`
   - For x86 machine: Machine Agent Bundle - 64-bit `Solaris.zip`
5. Unzip the agent Zip bundle:

   ```
   unzip <zip-bundle.zip> -d <machine_agent_home>
   ```

   where <machine_agent_home> is the install directory.
6. Gather your configuration details and configure the agent by editing <machine_agent_home>/conf/controller-info.xml file or by adding system properties to the JVM startup script file. See Plan the Machine Agent Configuration.

   a. (*Required*) Configure the Controller host name, port number, and account access key.
   b. (*Required for Multi-Tenant Mode or SaaS installations*) Configure the Agent Account Information. See Multi-Tenant Controller Accounts.
   c. (*Optional*) Review memory requirements. See Machine Agent Requirements and Supported Environments.
   d. (*Optional*) Configure the agent to use SSL. See Enable SSL for Machine Agent.
   e. (*Optional*) Configure the agent to use proxy settings. See Machine Agent Configuration Properties.
   f. (*Optional*) Determine if you need to specify an application name and tier name. See Standalone Machine Agent Installation Scenarios. However, if you are installing the Standalone Machine Agent on the same server with any APM app agent, do not specify the application name and tier name.
7. Start the agent from the command line:

   ```
   % <machine_agent_home>/bin/machine-agent
   ```

8. Verify that the Agent is reporting to the Controller.

Windows Install Using ZIP with Bundled JRE

This page describes how to install Windows using ZIP with bundled JRE.

For some versions of Windows Server 2008 and Windows Vista, the Avg Read and Write Time disk metrics are reported as 0. This is due to a known Microsoft Windows bug. See Microsoft Knowledge Base. If this affects you, download the available hotfix. This refers to the following metrics:

- `Hardware Resources|Disks|<mount_point>|Avg Read Time (ms)`
- `Hardware Resources|Disks|<mount_point>|Avg Write Time (ms)`

Install the Agent

1. Before installing, see Install the Machine Agent.
2. Install all available Windows updates.
3. If you have not restarted your machine recently, you must restart it. Failure to do so may cause your machine to experience a CPU spike when you start the Machine Agent.
4. Download and unzip the Windows ZIP bundle. Extract the contents to the agent installation directory `<machine_agent_home>`.
5. Gather your configuration details and configure the agent by editing the <machine_agent_home>/conf/controller-info.xml file or by adding system properties to the JVM startup script file. See Plan the Machine Agent Configuration.
    a. (*Required*) Configure the Controller host name, port number, and account access key.
    b. (*Required for Multi-Tenant Mode or SaaS installations*) Configure the Agent Account Information. See Multi-Tenant Controller Accounts.
    c. (*Optional*) Review memory requirements. See Machine Agent Requirements and Supported Environments.
    d. (*Optional*) Configure the agent to use SSL. See Enable SSL for Machine Agent.
    e. (*Optional*) Configure the agent to use proxy settings. See Machine Agent Configuration Properties.
    f. (*Optional*) Determine if you need to specify an application name and tier name. See Standalone Machine Agent Installation Scenarios. However, if you are installing the Standalone Machine Agent on the same server with any APM app agent, do not specify the application name and tier name.
6. Start the agent. If no `-D` options are required on the command line, you can run the following `.vbs` scripts by selecting them from Windows Explorer.
    a. As a Windows Service - Administrative privileges are required. You can specify AppDynamics-specific `-D` options from the command line or edit the controller-info.xml. You can also add other JVM properties as needed.
    Open a terminal window and enter:

    ```
    cscript <machine_agent_home>\InstallService.vbs <jvm_options>
    ```

    You can now start or stop the AppDynamics Machine Agent service from Windows Services.
    b. From an application in the terminal window, enter:

    ```
    cscript <machine_agent_home>\bin\machine-agent.vbs.
    ```

    The usage parameters for `machine-agent.vbs` are:

    Using machine-agent.vbs

    ```
    > machine-agent.vbs -h
    Usage: machine-agent.vbs[-h] [-j JAVA_HOME][-Dprop1 ...] [-Xprop2 ...]
    Start the machine agent.
        -h              print command line options
        -j JAVA_HOME    set java home for the agent
        -Dprop1         set standard system properties for the agent
        -Xprop2         set non-standard system properties for the agent
    ```

    The following is an example of machine-agent.vbs:

    ```
    > cscript machine-agent.vbs -Dappdynamics.controller.hostName=192.168.1.20 -Dappdynamics.
    controller.port=8090
    ```

Deploy Multiple Machine Agents From a Common Directory

You can deploy the Standalone Machine Agent on multiple hosts from one central directory. This setup provides an easier method to update the agent because each agent instance starts from one set of executables in one central location.

Workflow Description

1. Set up the central agent directory:
    a. Install the Machine Agent executables and config files in a central directory, such as an NFS-mounted shared directory.
    b. Specify properties common to all remote agents in this central directory.
2. Configure each remote host to start an instance of the agent from this central directory:
    a. Specify properties that are unique to the agent on that host, either in a startup script or on the command line.
    b. Configure a startup script to start an instance of the Machine Agent from the central agent directory.

Set Up the Central Agent Directory

1. Download the latest Machine Agent ZIP file. See Download AppDynamics Software.
2. Unzip the downloaded file to the desired root directory (`<central-agent-root-directory>`) on the central host.
3. Specify the properties common to all Standalone Machine Agent instances running on all remote hosts. Examples of common properties include the account name, and the Controller host and port. You specify these properties in `<central-agent-root-directory>/conf/controller-info.xml`.

Configure Each Remote Host to Start an Agent Instance

On each Machine Agent host, specify the properties unique to that agent such as: runtime directory, application name, tier name, and node name. To specify these properties, configure `-D` parameters in the startup script or on the command line. For example, the startup script on a remote host might include the following command:

```
<central_agent_root_directory>/bin/machine-agent -Dappdynamics.agent.runtime.dir=<agent_runtime_dir>  -
Dappdynamics.agent.applicationName=<application_name>
```

You should specify a unique runtime directory for each remote host to ensure that log files for each agent instance are saved to a separate directory. The agent log directory is located under the agent runtime directory, which you specify in the `-Dappdynamics.agent.runtime.dir` system property. The log directory is directly under the runtime directory (`<agent-runtime-dir>/logs`).

- Specify the runtime directory as an absolute path in the startup script on the remote host:

    ```
    -Dappdynamics.agent.runtime.dir=<absolute_path_to_local_agent_runtime_directory>
    ```

- To change the log directory and file name from the default value, edit the file `<*machine-agent-home*>/confs/logging/log4.xml`. Change the value of the `"FileAppender" > "fileName"` and `"FileAppender" > "filePattern"` parameters to the desired directory and file name.
- The file-system permissions must allow creation of the log directory and file.

Verify the Standalone Machine Agent Installation

Related pages:

- FAQs and Troubleshooting for the Machine Agent
- Machine Agent Configuration Properties

Check Agent Logs

After a successful install, the agent log at `<machine_agent_home>/logs`, should contain the following message:

```
Started AppDynamics Machine Agent Successfully
```

If the agent log file is not present, the Machine Agent may not be accessing the command properties. To troubleshoot, check the application server log file where STDOUT is logged. It has the fallback log messages, which are useful for troubleshooting the agent.

Verify that the Agent is Reporting to the Controller

1. From the Controller Top Navigation Bar, select **Settings > AppDynamics Agents**.
2. Click the Machine Agents tab.
   The listing for the Machine Agent displays. If you do not see the Machine Agent listed, check your property settings in `<machine_agent_home>/conf/controller-info.xml`. See Resolve Machine Agent Installation Problems.

Machine Agent Not Reporting

If it seems that the Machine Agent is not reporting to the Controller, see Machine Agent Not Reporting article in the Community Knowledge Base.

Resolve Standalone Machine Agent Installation Problems

Related pages:

- FAQs and Troubleshooting for the Machine Agent
- Machine Agent Configuration Properties

Verify That the Agent Process is Running

Use the following command to verify that the agent process is running:

Linux:

```
ps -ef | grep machine
```

Windows:

1. Open a command line console.
2. Start the Task Manager and click the Processes tab.
3. The agent process should be running. If it is not running, stop and then restart the agent.

Resolve Agent Connectivity Problems

Make sure you have configured the Controller IP address, Controller port number, and Account Access Key in the agent startup command, script or plist, or in the controller-info.xml file.

After configuring, restart the agent and check the behavior. Machine Agent log files may also provide troubleshooting information.

If you start the Machine Agent, and it cannot register with the Controller or associate with the same node in the Controller, then the stack trace may indicate the reason.

For example, the following message in the stack trace may indicate that the application, tier, and node information was not provided during the Machine Agent startup command or in the controller-info.xml file.

```
<execution-output>System agent 239590 not associated with application, metric registration request refused.<
/execution-output>
```

Start and Stop the Standalone Machine Agent

This page contains a summary of the `start` and `stop` commands for the Machine Agent. See Permissions Required to Run the Machine Agent.

> **Running the Machine Agent with a Non-Bundled JRE**
>
> To run the Machine Agent with the default JRE on the machine (rather than the JRE bundled with the agent), you should first install a non-JRE version of the Machine Agent, and then run the non-JRE agent instead. See Install Using the Non-JRE Zip File.

Start the Machine Agent

If you do not use the Machine Agent bundled JRE, you need to specify the Java version and customize the `<path/to/java_location>` using the `-j` parameter and version JRE >= 1.8, see Machine Agent Requirements and Supported Environments and Install Using the Non-JRE Zip File. Make certain the path is valid and accessible for all users who start the Machine Agent.

```
<machine_agent_home>/bin/machine-agent -j <path/to/java_location>
```

RPM-based Linux Systems

If you installed using the Linux RPM, start the Machine Agent:

- SysV systems: `service appdynamics-machine-agent start`
- systemd systems: Use the Machine Agent service launcher and enter: `systemctl start appdynamics-machine-agent`

Linux and Solaris Systems

If you installed using the JRE bundled zip archive for Linux and Solaris, run the launcher:

- Linux: `<machine_agent_home>/bin/machine-agent -d -p <machine_agent_home>/pidfile`
- Solaris: `<machine_agent_home>/bin/machine-agent -d -p <machine_agent_home>/pidfile`

Windows Systems

See Windows Install Using ZIP with Bundled JRE.

Stop the Machine Agent

Non-JRE Bundled Zip Archive

If the Machine Agent process is running in the background, you can stop it by entering the `kill` command with the process ID as the argument. If it is running in the foreground in a console, press Ctrl+C to shut down the agent.

RPM-based Linux Systems

SysV: Use the service launcher:

```
% service /etc/init.d/appdynamics-machine-agent stop
```

systemd: Use the systemd unit file:

```
% systemctl stop appdynamics-machine-agent
```

Linux and Solaris Systems

Running in the Foreground: Press Ctrl+C to stop the agent process.

Running in the Background: Identify the process ID (PID) and use the `kill` command.

1. Identify the agent process: `ps -ef | grep machineagent.`
   The output provides the process ID (PID) of the Machine Agent process.
2. Stop the agent process: `kill <machine_agent_PID>`.

Windows

In the foreground in a console window: Use Ctrl+C to shut down the agent.

As a Windows Service: Stop the service using either of these methods:

- In the Windows Services application, select **AppDynamics Machine Agent** and click **Stop**.
- Use the `MachineAgentService` command:

```
C:\> <machine_agent_home>\bin\MachineAgentService.exe /stop
```

Mac OS X

As a background process: Enter the `kill` command with the process ID as the argument.

As a service: From the bash command line:

```
> sudo launchctl unload -w <machine_agent_home>/com.appdynamics.machineagent.plist
```

Permissions Required to Run the Machine Agent

This page describes the permissions needed to run the Machine Agent. During installation, the default user for running the Machine Agent is set to root. This is because the only user that is standard on a UNIX or Linux system is root and we do not want to create users on your system. We recommend that you create a non-root user, for example *<machine_agent_user>*, and assign the appropriate permissions to that user. See Install the Machine Agent.

For all environments, create a specific user with the necessary read/write/execute permissions.

- All files in the *<machine-agent-home>* installation directory should be readable by the Machine Agent.
- The user that runs the Machine Agent must have write privileges to the logging output directory and to the `/conf` directory in the agent installation directory.
- Additionally, the user that runs the Machine Agent needs execute access.

Important Notes

- You do not need to run the Machine Agent from a root or administrator account. However, if you enable the JVM Crash Guard on a monitored application running from a root or administrator account, then the Machine Agent requires root or administrator privileges to access the monitored application's JVM process and directory listings for crash files.
- You need to run with administrator or root privileges if you want to monitor networks or disks that are only available to the administrator or root user.
- The user that runs the Machine Agent must have write privileges to the `conf` and `logs` directories in the *<machine_agent_home>* directory.
- The Machine Agent implements a shutdown hook, so issuing the `kill` command (or Ctrl+C) from the operating system will cause the agent to perform a graceful shutdown.

Linux

- ip
- df
- awk
- basename
- cat
- date
- dmesg
- md5sum
- readlink
- sed
- uname
- ps

Windows

Windows permissions for files and subfolders are inherited by default from the parent folder (*<machine_agent_home>*). You should restrict permissions to users authorized to start, stop, and configure the Machine Agent:

- Read and Write permissions to all files and subfolders under `<machine-agent-home>`
- (If running as a terminal application) Read, Write, and Execute permissions for the file `<machine-agent-home>\bin\machine-agent.vbs`
- (If running as a service) Start, Stop, and Restart permissions for the Machine Agent service. You only need admin privileges to install the service. The Machine Agent runs under the local system account which has extensive privileges on the local system, so you do not need to run the Machine Agent as Administrator, unless WMI access is revoked. Normal users typically have WMI access. See https://technet.microsoft.com/en-us/library/cc771551.aspx.

Mac OS X, AIX, HP-UX, and Z/OS

There are no particular execute privileges required.

Solaris System Utilities

- awk
- netstat
- zpool
- egrep
- iostat
- prtconf
- pagesize
- kstat
- prstat
- grep
- vmstat

JVM Crash Guard

If you plan to enable JVM Crash Guard, see JVM Crash Guard for additional required permissions.

Configure the Standalone Machine Agent

Related pages:

- Plan the Machine Agent Configuration
- .NET Compatibility Mode

AppDynamics provides flexibility for configuring the Machine Agent so that you can choose the best fit for your deployment environment.

Not all options are available for all properties. See Machine Agent Configuration Properties.

Configure the Machine Agent Properties

You can configure the Machine Agent by:

- Editing the `controller-info.xml` file located in the *<machine_agent_home>*`/conf` directory.
- Adding Agent system properties (`-D`*<system_property>*) to the Machine Agent start-up script or on the command line.
- Using environment variables. To configure the Agent with environment variables, set the value of the environment variable in the environment where the monitored application runs and restart the Agent. Environment variables exist for most of the Agent settings in the `controller-info.xml file`, but not all settings are configurable through environment variables. For those settings, you need to use system properties or `controller-info.xml`. See Machine Agent Configuration Properties.

Empty spaces and special characters are not allowed in either the full path to the Machine Agent home directory, or in the directory name itself. If the Machine Agent does not start up, review the path to determine if it contains empty spaces or special characters. For example, the path `/opt/appdynamics/machine agent` is problematic, however, the path`/opt/appdynamics/machine-agent` works correctly.

The JVM system properties and environment variables override the settings in the `controller-info.xml` file. The Agent applies the first non-empty value for a configuration property. The Machine Agent applies configurations using the following sources (in order).

1. Environment variables
2. System properties passed in the `start` command for the JVM
3. Global configuration file: *<machine-agent-home>*`/conf/controller-info.xml`

See Where to Specify Machine Agent Configuration for OS environment and install package details

Example Configuration

Machine Agent controller-info.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<controller-info>    <controller-host>your-controller-host</controller-host>
    <controller-port>your-controller-port</controller-port>
    <account-access-key>your-access-key<account-access-key>
    <controller-ssl-enabled>true</controller-ssl-enabled>
    <enable-orchestration>false</enable-orchestration>
    <sim-enabled>false</sim-enabled>
    <unique-host-id>your-host-id</unique-host-id>
    <account-name>your-account-name</account-name>
</controller-info>
```

bash command-line Example

```
<machine_agent_home>/bin/machine-agent -Dappdynamics.controller.hostName=your-controller-host
-Dappdynamics.controller.port=your-controller-port -Dappdynamics.agent.accountAccessKey=your-access-key -
Dappdynamics.agent.uniqueHostId=your-host-id -Dappdynamics.agent.accountName=your-account-name
```

Where to Specify Machine Agent Configuration

Related pages:

- Install the Machine Agent
- Plan the Machine Agent Configuration
- Machine Agent Configuration Properties

You configure Agent system properties in different ways based on your operating system and whether you are starting the Agent from the command line or as a service.

Linux As a Service

- SysV service launcher:
  Specify the Agent system properties in the `<machine_agent_home>/etc/sysconfig/appdynamics-machine-agent` configuration file, and edit the `JAVA_OPTS` environment variable.
- systemd:
  Specify the Agent system properties in the `<machine_agent_home>/etc/systemd/system/appdynamics-machine-agent.service` file, and edit the `JAVA_OPTS` environment variable.

Any UNIX-like System

When starting the Agent application on the command line for an UNIX-like system ((Linux, Solaris, Mac, AIX, and so on) using the `machine-agent` command, specify the agent system properties on the command line:

- Run agent in the background:
  ```
  % nohup <machine_agent_home>/bin/machine-agent -D<system_property1>=<value1> -D<system_property2>=<value2> ... &
  ```
- Run agent in the foreground:
  ```
  % <machine_agent_home>/bin/machine-agent -D<system_property1>=<value1> -D<system_property2>=<value2> ...
  ```

Windows

When installing and starting a Windows service (you need admin privileges), specify AppDynamics-specific -D options on the command line or in `controller-info.xml`.
Add other JVM properties on the command line:

```
<machine_agent_home>\cscript InstallService.vbs InstallService.cmd <jvm_options>
```

Mac OS X

Start the Agent service using the `<machine_agent_home>/osx-install.sh` script and specify the Agent system properties on the command line:

```
> sh <machine_agent_home>/osx-install.sh -D<system_property1>=<value1> -D<system_property2>=<value2> ...
```

When you run the `<machine_agent_home>/osx-install.sh` script, the `<machine_agent_home>/com.appdynamics.machineagent.plist.template` is updated with the installation directory and the java properties set for the Machine Agent.

Machine Agent Configuration Properties

This page describes the Agent configuration properties, including `controller-info.xml` elements, system property options (on the command line or in the startup script), and environment variables (where applicable). You configure Agent system properties based on your operating system and installation package.

The Agent updates dynamically in response to Agent configuration property changes, so you do not need to restart the Agent.

System Property Syntax

- System properties are case-sensitive
- Values that contain spaces must be enclosed with double-quotes

Reference

.NET Compatibility Mode

You must enable this mode if you want to collect and view Machine or Server metrics on a server with Machine and .NET Agents installed. See .NET Compatibility Mode.

Element in controller-info.xml: `<dotnet-compatibility-mode>`

System Property:

```
-Dappdynamics.machine.agent.dotnetCompatibilityMode
```

Environment Variable: N/A

Type: boolean

Default: false

Required: This mode is required if you want to collect and view Machine or Server metrics on a server with Machine and .NET Agents installed.

Account Access Key

The account access key used to authenticate with the Controller. This key is generated during installation, and you can locate the key by reviewing the license information in the Controller Settings. See License Management.

Element in controller-info.xml: `<account-access-key>`

System Property: `-Dappdynamics.agent.accountAccessKey`

Environment Variable: `APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY`

Type: String

Default: None

Required: Prior to version 4.1, this property was required only for SaaS and multi-tenant Controllers. For versions 4.1 and later, the account access key property is required to authenticate all agent to Controller communications.

Example: `-Dappdynamics.agent.accountAccessKey=165e65645-95c1-40e3-9576-6a1424de9625`

Account Name

The account name used to authenticate with the Controller. If you are using the AppDynamics SaaS Controller, the Account Name is provided in the Welcome email sent by AppDynamics.

Element in controller-info.xml: `<account-name>`

System Property: `-Dappdynamics.agent.accountName`

Environment Variable: `APPDYNAMICS_AGENT_ACCOUNT_NAME`

Type: String

Default: None

Required: For AppDynamics SaaS Controller and multi-tenant users, but not for single-tenant mode (the default). When the Agent is registered with an AppDynamics SaaS Controller, features used to run Remediation Scripts are disabled. If you reconfigure the Agent `controller-info.xml` to register with a non-SaaS or on-premises Controller, the Agent can run local scripts as usual.

Agent Logging Directory

Deprecated. `appdynamics.agent.logs.dir` should be used instead. Behavior identical to Agent Logs Directory: `<appdynamics.agent.logs.dir>/logs/`.

System Property: `-Dappdynamics.agent.logging.dir`

Environment Variable: N/A

Type: String

Default: None

Required: No

Agent Logs Directory

Sets the logs directory for log files for nodes that use this agent installation. If this property is specified, all agent logs are written to `<appdynamics.agent.logs.dir>/logs`. See Deploy Multiple Machine Agents From a Common Directory when deploying multiple Machine Agents from a common directory. This property overrides the directory specified through the `appdynamics.agent.runtime.dir`. If `appdynamics.agent.logs.dir` property is not provided, the logs directory will be created on the same level as the Machine Agent installation folder. No changes are expected in `log4j.xml` file.

System Property: `-Dappdynamics.agent.logs.dir`

Environment Variable: N/A

Type: String

Default: None

Required: No

Agent Runtime Directory

Sets the runtime directory for all runtime files (such as logs) for nodes that use this Agent installation. If you specify this property, then all Agent logs are written to `<agent-runtime-dir>/logs/node-name`. Use when deploying multiple Machine Agents from a common directory. See Deploy Multiple Machine Agents From a Common Directory.

System Property: `-Dappdynamics.agent.runtime.dir`

Environment Variable: N/A

Type: String

Default: None

Required: No

Container Process Selector Blacklist Regex

Any container with a process matching this regex is ignored and is not registered in the Controller.

System Property: `-Dappdynamics.docker.container.process.selector.blacklist.regex`

Environment Variable: `APPDYNAMICS_DOCKER_CONTAINER_PROCESS_SELECTOR_BLACKLIST_REGEX`

Type: String

Default: None

Required: No

Controller Host

This is the host name or IP address of the AppDynamics Controller, for example: 192.168.1.22 or *myhost* or *myhost.abc.com*. This is the same host used to access the AppDynamics browser-based user interface.

Element in controller-info.xml: `<controller-host>`

System Property: `-Dappdynamics.controller.hostName`

Environment Variable: `APPDYNAMICS_CONTROLLER_HOST_NAME`

Type: String

Default: None

Required: If the Enable Orchestration property is false.
If Enable Orchestration is true, and if the Agent is deployed in a compute cloud instance created by an AppDynamics workflow, do not set the Controller host unless you want to override the auto-detected value. See Enable Orchestration Property.

Controller Keystore Filename

By default, the Agent looks for a Java truststore file named `cacerts.jks` in the `conf` directory in the Agent home. Use this property to enable full validation of Controller SSL certificates with a different Java truststore file. See Enable SSL for the Standalone Machine Agent.

Element in controller-info.xml: `<controller-keystore-filename>`

System Property: N/A

Environment Variable: N/A

Type: String

Default: None

Required: No

Controller Keystore Password

The plain text or encrypted value of the Controller certificate password. To encrypt or obfuscate passwords, see Encrypt Agent Credentials.

Element in controller-info.xml: `<controller-keystore-password>`

System Property: N/A

Environment Variable: N/A

Type: String

Default: None

Required: No

Controller Port

The HTTP(S) port of the AppDynamics Controller. This is the same port that you use to access the AppDynamics browser-based user interface. If the Controller SSL Enabled property is set to true, specify the HTTPS port of the Controller; otherwise, specify the HTTP port. See Controller SSL Enabled Property.

Element in controller-info.xml: `<controller-port>`

System Property: `-Dappdynamics.controller.port`

Environment Variable: `APPDYNAMICS_CONTROLLER_PORT`

Type: Positive Integer

On-premises Default: port 8090 for HTTP and port 8181 for HTTPS

SaaS Default: For the SaaS Controller Service, use port 443 for HTTPS connections.

Required: If the Enable Orchestration property is false.

If Enable Orchestration is true, and if the Agent is deployed in a compute cloud instance created by an AppDynamics workflow, do not set the Controller port unless you want to override the auto-detected value. See Enable Orchestration Property.

Controller SSL Enabled

Specifies whether the Agent should use SSL (HTTPS) to connect to the Controller. If SSL Enabled is true, set the Controller Port property to the HTTPS port of the Controller. See Controller Port Property.

Element in controller-info.xml: <controller-ssl-enabled>

System Property: `-Dappdynamics.controller.ssl.enabled`

Environment Variable: `APPDYNAMICS_CONTROLLER_SSL_ENABLED`

Type: Boolean

Default: false

Required: No

Create Node if Absent

Force the Machine Agent to create an APM node when the Agent registers with the Controller.

Element in controller-info.xml: `<create-node-if-absent>`

System Property: `-Dappdynamics.machine.agent.registration.createNodeIfAbsent`

Environment Variable: N/A

Type: Boolean

Default: true

Required: No. If you set the app/tier/node in your `controller-info.xml` file (existing upgrades or by accident), you can prevent the Machine Agent from creating APM nodes by setting this flag to false. See Machine Agent Installation Scenarios.

Enable Docker Visibility

To enable Docker Visibility on the Agent, manually add the docker-enabled element in `controller-info.xml` setting, and set the flag to `true`.

Element in controller-info.xml: <docker-enabled>true</docker-enabled>

System Property: -Dappdynamics.docker.enabled

Environment Variable: APPDYNAMICS_DOCKER_ENABLED

Type: Boolean

Default: false

Required: Yes

Enable HTTP Listener

When set to `true`, this property enables the Machine Agent HTTP listener. You can send metrics to the Machine Agent using its HTTP listener. You can report metrics through the Machine Agent by making HTTP calls to the Agent instead of piping to the Agent through `sysout`.

Element in controller-info.xml: N/A

System Property: `-Dmetric.http.listener`

Environment Variable: N/A

Type: Boolean

Default: false

Required: No

Enable Orchestration

Enables the Machine Agent workflow task execution when set to `true`. It also enables auto-detection of the Controller host and port when the app server is a compute cloud instance created by an AppDynamics orchestration workflow. In a cloud computing environment, auto-detection is necessary for the Create Machine tasks in the workflow to run correctly. The Machine Agent polls for task executions only when orchestration is enabled. If the host machine on which this Agent resides is not created through AppDynamics workflow orchestration, this property should be set to `false`. See Controller Host Property and Controller Port Property.

Element in controller-info.xml: `<enable-orchestration>`

System Property: N/A

Environment Variable: N/A

Type: Boolean

Default: false

Required: No

Force Default SSL Certificate Validation

Used to override the default behavior for SSL validation.

This property has three states:

> *true:* Forces the Agent to perform full validation of the certificate sent by the Controller, enabling the Agent to enforce the SSL trust chain. Use this setting when a public certificate authority(CA) signs your Controller SSL certificate.
>
> *false:* Forces the Agent to perform minimal validation of the certificate. This property disables full validation of the Controller's SSL certificate. Use this setting when full validation of a SaaS certificate fails.
>
> *unspecified:* The validation performed by the Agent depends on the context:
>
> - If the Agent is connecting to a SaaS Controller, full validation is performed.
> - If the Agent is connecting to an on-premises Controller and the `cacerts.jks` file is present, then full validation is performed using the cacerts.jks file.
> - If the Agent is connecting to an on-premises Controller, and there is no `cacerts.jks` file, then minimal validation is performed

Element in controller-info.xml: N/A

System Property: `-Dappdynamics.force.default.ssl.certificate.validation`

Environment Variable: N/A

Type: Boolean

Default: None

Required: No

Enable Dynamic Monitoring Mode (DMM)

When this option is enabled, the Agent reports metrics based on the Dynamic Monitoring Mode specified for that Agent in the Controller. When this option is disabled, the Agent reports all metrics based on its local configuration; DMM settings on the Controller have no effect. Disabling DMM on an Agent is recommended only for mission-critical servers and other machines for which you are sure you want to collect all available metrics at all times. See Dynamic Monitoring Mode and Server Visibility.

Element in controller-info.xml: <dynamic-monitoring-enabled>

System Property: appdynamics.machine.agent.dynamicMonitoring.enabled

Environment Variable: APPDYNAMICS_DYNAMIC_MONITORING_ENABLED

Type: Boolean

On-premises Default: True

SaaS Default: True

Required: No

HTTP Listener Port

To enable the Machine Agent HTTP listener, you must also specify the HTTP listener port.

Element in controller-info.xml: N/A

System Property: `-Dmetric.http.listener.port`

Environment Variable: N/A

Type: Numeric

Default: 8293

Required: Only if the HTTP listener is enabled.

Log4j

Logging functionality is done via Apache Log4j2. Use this property to provide a custom location for log4j configuration. In this case, you need to ensure that all file destinations are valid and contain absolute paths.

Element in controller-info.xml: N/A

System Property: `-Dlog4j.configurationFile`

Environment Variable: N/A

Type: Numeric

Default: None

Required: No

Machine Hierarchy

You need a Server Visibility license to use this feature.

This setting enables you to group servers together into arbitrary hierarchies by specifying a hierarchical path to the server. The server hierarchy displays in the Metric Browser and on the Server Dashboard. The server hierarchy is also used to select subgroups of machines for health rules. The last element of the path indicates the server name (a name of your choice). This name appears as the Name on the Servers list. If the path contains spaces, then you must it enclose it in double-quotes. See Machine Agent Hierarchy.

Element in controller-info.xml: `<machine-path>`

System Property: `-Dappdynamics.machine.agent.hierarchyPath`

Environment Variable: `APPDYNAMICS_MACHINE_HIERARCHY_PATH`

Type: ASCII string with path elements that are separated by a "|" (bar).

Default: The value specified by Unique Host ID. If the last part of the machine hierarchy is empty, the Unique Host ID is the machine name. For example, if machine hierarchy is "Data Center 1|Rack 2|" and Unique host ID is "Host ID 3", then the machine hierarchy will become "Data Center 1|Rack 2|Host ID 3".

Required: No

Limitation: The length of the characters composing the machine-path up to, but not including, the last pipe cannot exceed 95 characters.

Examples:

- *System Properties:* `-Dappdynamics.machine.agent.hierarchyPath= "Data Center 1|Rack 2|Machine3"`
- *controller-info.xml:*

```
<machine-path>
     "Data Center 1|Rack 2|Machine3"
</machine-path>
```

- *Environment Variable:* `APPDYNAMICS_MACHINE_HIERARCHY_PATH="Data Center 1|Rack 2|Machine3`

Proxy Password File

The absolute path to the file containing the password of the user that is authenticated by the proxy host. The password must be the first line of the file and must be in clear (unencrypted) text. See Encrypt Agent Credentials.

Element in controller-info.xml: N/A

System Property: `-Dappdynamics.http.proxyPasswordFile`

Environment Variable: N/A

Type: String

Default: None

Required: No

Example: `-Dappdynamics.http.proxyPasswordFile=/path/to/file-with-password`

Proxy User Name

The name of the user that is authenticated by the proxy host.

Element in controller-info.xml: N/A

System Property: `-Dappdynamics.http.proxyUser`

Environment Variable: N/A

Type: String

Default: None

Required: No

Server Visibility Enabled

Enable Server Visibility on the Agent. This requires a Server Visibility license.

Element in controller-info.xml: `<sim-enabled>`

System Property: `-Dappdynamics.sim.enabled`

Environment Variable: `APPDYNAMICS_SIM_ENABLED`

Type: Boolean

Default: false

Required: Required to enable Server Visibility. See Enable Server Visibility.

Service Availability Update Interval

This setting controls the time, in milliseconds, to wait between sending Service Availability periodic events to the Controller. See Service Availability.

Element in controller-info.xml: `<sam-event-update-interval-millis>`

System Property: `-Dappdynamics.machine.agent.sam.event.updateIntervalMillis`

Environment Variable: N/A

Type: Positive integer

Default: 300000 ms (5 minutes)

Required: No

Unique Host ID

This property logically partitions a single physical host or virtual machine. In the context of installing the Machine Agent, the unique Host ID property is not required. However, if you do not define a unique Host ID, then the Machine Agent uses the Java API to retrieve the host ID. The results from the API can be inconsistent, and may cause the same JVM to return a different value for the same machine, each time the Machine Agent is restarted. To avoid these issues, AppDynamics recommends that you set the value of unique Host ID to the host ID that you want to see in the UI.

Element in controller-info.xml: `<unique-host-id>`

System Property: `-Dappdynamics.agent.uniqueHostId`

Environment Variable: `APPDYNAMICS_AGENT_UNIQUE_HOST_ID`

Type: ASCII string without spaces and must be unique across the entire managed infrastructure.

Default: None

Required: Optional, but recommended.

Use Simple Hostname

By default (unless overridden with the `uniqueHostId` system property), the Agent determines the host name of the OS it is running in using reverse DNS lookup. In some circumstances, this host name may be set as the fully qualified domain name of the host name. If this property is set to `true`, the Agent removes any domain name and uses the simple hostname to identify the host. In cases where the host name is an IP address (which happens if the DNS lookup fails) then the full IP address in string form is used. The host name is used in mapping metrics gathered by the Machine Agent to application nodes. See Unique Host ID Property.

Element in controller-info.xml: `<use-simple-hostname>`

Type: Boolean

Default: False

Required: No

For example: If this property is set to true '`server.mydomain.com`' becomes 'server'.

Independent Standalone Machine Agent Install Scenario

Typically, you only use the following properties if you are installing the Machine Agent on a server that does not have any AppDynamics App Agents installed.

Application Name

The name of the logical business application that this JVM node belongs to. This is not the deployment name (ear/war/jar) on the application server. If a business application of the configured name does not exist, it is created automatically.

Element in controller-info.xml: `<application-name>`

System Property: `-Dappdynamics.agent.applicationName`

Environment Variable: `APPDYNAMICS_AGENT_APPLICATION_NAME`

Type: String

Defaults: None

Required: No. See Machine Agent Installation Scenarios.

Node Name

The name of the JVM node. When not specified, this defaults to `Node1` for the Machine Agent.

Element in controller-info.xml: `<node-name>`

System Property: `-Dappdynamics.agent.nodeName`

Environment Variable: `APPDYNAMICS_AGENT_NODE_NAME`

Type: String

Defaults: None

Required: No. See Machine Agent Installation Scenarios.

Tier Name

The name of the logical tier that this JVM node belongs to. This is not the deployment name (ear/war/jar) on the application server. If a tier of the configured name does not exist, it is created automatically.

Element in controller-info.xml: `<tier-name>`

System Property: `-Dappdynamics.agent.tierName`

Environment Variable: `APPDYNAMICS_AGENT_TIER_NAME`

Type: String

Defaults: None

Required: No. See Machine Agent Installation Scenarios.

Controller Settings for Standalone Machine Agents

Related pages:

- Database Size and Data Retention
- Controller Settings for Server Visibility

This page describes Controller Admin settings that are specific to the Machine Agent. You need the root user password to change these settings.

You can change the amount of Machine Agent data you retain in the Controller database by changing the retention period for Machine Agent snapshots. Lo wering the retention settings purges data that is aged out by the new retention setting and reduces the amount of data stored by the Controller on an ongoing basis. Consider the amount of Machine Agent data to retain when tuning the size of the Controller database.

Change the Controller Settings for Machine Agents

1. Log in to the Controller administration console using the root user password. See Access the Administration Console.

   ```
   http://<controller host>:<port>/controller/admin.jsp
   ```

   Use the root user password to access the Admin console when the Controller is installed in single- or multi-tenant mode. See Update the Root User and Glassfish Admin Passwords.
2. Select **Controller Settings**.
3. Change the settings as needed, and then select **Save**. Changes to the settings occur the next time you restart the Agent and it connects to the Controller.

Controller Settings Reference for Machine Agent

Property Name	Property Description	Default	Allowed Values
`machine.agent.in-progress.actions.timeout`	This property determines the length of time the Controller waits for the Agent to perform an IN-PROGRESS action before it stops the action.	86400000 ms (1 day)	1 to 840 hours (5 weeks)
`machine.agent.max.new.actions.per.min`	Maximum number of new actions dispatched per minute for each Machine Agent. This is the maximum number of "runbook" actions sent to the Machine Agent. For example, you may request the Machine Agent to run a script because of a health rule violation, if the number of requests in a one minute period is greater than the preset value, then the action is executed the next minute.	15 per minute	1 to 28999999
`machine.agent.pending.actions.timeout`	Determines how long the Controller waits for the Agent to perform a PENDING action before it stops the action.	1800000 ms (12 hrs)	
`machine.agent.snapshots.buffer.size`	Size in Megabytes of the in-memory buffer storing machine snapshots uploaded from Agents prior to database flush.	50 MB	
`machine.snapshots.retention.period`	Time in hours to retain server snapshot data.	336 hours (2 weeks)	1 to 840 hours (5 weeks)

Enable SSL for Standalone Machine Agent

Related pages:

- Secure the Platform
- Controller SSL and Certificates
- Install the Machine Agent

This page describes how to configure the AppDynamics Machine Agent to connect to the Controller using SSL. It assumes that you use a SaaS Controller or have configured the on-premises Controller to use SSL.

The Machine Agent supports extending and enforcing the SSL trust chain when in SSL mode.

Plan SSL Configuration

Gather this information:

- The Controller SSL port:
  - For SaaS Controllers: SSL port is 443
  - For on-premises Controllers: Default SSL port is 8181, but you may configure the Controller to listen for SSL on another port
- The signature method for the Controller's SSL certificate:
  - A publicly known certificate authority (CA) signed the certificate. This applies for DigiCert, Verisign, Thawte, and other commercial CAs.
  - A CA internal to your organization signed the certificate. Some companies maintain internal certificate authorities to manage trust and encryption within their domain.
  - The Controller uses a self-signed certificate.

Establish Trust for the Controller's SSL Certificate

To establish trust between the Machine Agent and the AppDynamics Controller, you must create an agent truststore that contains the root certificate for the authority that signed the Controller's certificate.

1. Obtain one of the following root certificates:
   - DigiCert Global Root CA for the AppDynamics SaaS Controller
   - The root certificate for the publicly known certificate authority (CA) that signed the certificate for your on-premises Controller
   - The root certificate for the internal CA that signed the Controller certificate for your on-premises Controller
2. Run the Java `keytool` command to create the Agent truststore:

   ```
   keytool -import -alias rootCA -file <root_certificate_file_name> -keystore cacerts.jks -storepass
   <truststore_password>
   ```

   For example:

   ```
   keytool -import -alias rootCA -file /usr/home/appdynamics/DigicertGlobalRootCA.pem -keystore cacerts.jks
   -storepass MySecurePassnword
   ```

   Note the truststore password; you will need this later to configure the Machine Agent.

3. Install the Agent truststore to the Agent configuration directory:

   ```
   <machine_agent_home>/conf/
   ```

Secure the Machine Agent Truststore

AppDynamics recommends you take the following security measures to prevent tampering with the Machine Agent truststore:

- Secure the truststore file through filesystem permissions:
  - Make the Agent truststore readable by any user
  - Make the truststore owned by a privileged user
  - Make the truststore writable only by the specified privileged user
- Secure the `controller-info` configuration file so that it is only readable by the Agent runtime user and only writable by a privileged user:

  <machine_agent_home>`/conf/controller-info.xml`

Enable SSL for the Machine Agent

1. Configure the following system properties in the `controller-info.xml`: `<machine_agent_home>/conf/controller-info.xml`. See Machine Agent Configuration Properties for full details on each property.
    - *Controller Host:* Should be the same as either the Common Name or the Subject Alternative Name (SAN) in the certificate configured for the Controller.

      ```
      <controller-host>common_name_in_certificate.com</controller-host>
      ```
    - *Controller Port:* The SSL port for the Controller. It is 443 for AppDynamics SaaS.

      ```
      <controller-port>443</controller-port>
      ```
    - *Controller SSL Enabled:* true

      ```
      <controller-ssl-enabled>true</controller-ssl-enabled>
      ```
    - *Controller SSL Password:* The plain text password for the Agent truststore.

      ```
      <controller-keystore-password>MySecurePassword</controller-keystore-password>
      ```

      If you have enabled the Secure Credential Store, encrypt the password you enter here. See Encrypt Agent Credentials.
    - *Controller Keystore Filename:* The path of the Agent truststore relative to `<machine_agent_home>/conf`. This is required if you use a truststore other than the default `<machine_agent_home>/conf/cacerts.jks`.

      ```
      <controller-keystore-filename>../conf/cacerts.jks</controller-keystore-filename>
      ```

      > You can specify the Controller port and enable SSL for the Controller in the Machine Agent startup script, but you must specify the truststore password and filename in the `controller-info.xml` file.

2. Restart the Machine Agent.

Sample controller-info.xml with SSL and Secure Credential Store Encryption Enabled

```
<?xml version="1.0" encoding="UTF-8"?>
<controller-info>
        <controller-host>mycompany.saas.appdynamics.com</controller-host>
        <controller-port>443</controller-port>
        <controller-ssl-enabled>true</controller-ssl-enabled>
    <!-- Encrypted Controller keystore / agent trust store password -->
        <controller-keystore-password>Tw49bd0hdCMBoQ5pfMMuYA/cA5B4pouVPkv48ovRm6c=</controller-keystore-
password>
        <controller-keystore-filename>../../conf/cacerts.jks</controller-keystore-filename>
        ...
    <!-- Secure Credential Store configuration -->
    <!-- Enable the Secure Credential Store -->
    <use-encrypted-credentials>true</use-encrypted-credentials>
    <!-- Path to they secure credential keystore -->
    <credential-store-filename>/opt/appdynamics/secretKeyStore</credential-store-filename>
    <!-- Obfuscated secure credential keystore password -->
    <credential-store-password>n/8GvAZsKk4gM3Z6g+XQ1w==</credential-store-password>
</controller-info>
```

Keystore Certificate Extractor Utility

The Keystore Certificate Extractor Utility exports certificates from the Controller's Java keystore and writes them to an Agent truststore. You can run this utility with the Agent distribution on the Controller:

```
<controller_home>/appserver/glassfish/domains/domain1/appagent
```

1. Execute `kr.jar` and include the following parameters:
    - The full path to the Controller's keystore:

      ```
      <controller_home>/appserver/glassfish/domains/domain1/config/keystore.jks
      ```

    - The truststore output file name. By default, the Machine Agent looks for `cacerts.jks`.
    - The password for the Controller's certificate, which defaults to "changeit". If you do not include a password, the extractor applies the password "changeit" to the output truststore.

```
java -jar kr.jar <controller_home>/appserver/glassfish/domains/domain1/config/keystore.jks cacerts.
jks <controller_certificate_password>
```

2. Install the Agent trust store to the Agent configuration directory:

```
<machine_agent_home>/conf/
```

.NET Compatibility Mode

This page explains how to set compatibility mode for the .NET Agent. Compatibility mode allows you to collect and view machine or server metrics on a server with Machine and .NET Agents installed.

Before You Begin

Review the list of requirements and general information about compatibility mode:

- .NET Compatibility Mode is required for metrics from a Machine Agent and a .NET Agent on the same server to be visibly associated with the same node in the Controller UI.
- Differences may exist between metric values reported by the Machine Agent and the .NET Agent due to different averaging rates and measurement methods.
- You must enable .NET Compatibility Mode on the Machine Agent.
- When .NET Compatibility Mode is enabled:
  - Events from Service Availability and Custom Extensions appear in the Servers tab, but are not associated with .NET applications in the Controller UI.
  - The Controller UI may show the server name with "`-java-MA`" appended to the host ID. Thus, a Machine Agent with a `<unique-host-id>` set to "ABC" might appear as "`ABC-java-MA`" in the Controller UI. This is expected behavior; the "`-java-MA`" suffix indicates that the host has a Machine Agent running in .NET Compatibility Mode.
- Do not specify the `<application-name>`, `<tier-name>`, or `<node-name>` properties on the Machine Agent. These properties are used with a Machine Agent on a host that has no other AppDynamics Agents installed.

Enable Compatibility Mode

To enable .NET Compatibility Mode:

1. Enable .NET Compatibility Mode on the Agent using one of two methods:
   - Set these options in `<machine_agent_home>/conf/controller-info.xml`:
     a. `<dotnet-compatibility-mode>`**true**`</dotnet-compatibility-mode>`
        You may need to add this line to the `controller-info.xml` file.
     b. `<unique-host-id>`*`<unique-host-id-of-dot-net-agent>`*`</unique-host-id>`
        For both Agents to report metrics to the same node, the Machine Agent must use the same case-sensitive Unique Host ID used by the .NET Agent. See FAQs and Troubleshooting.
   - Set the `./bin/machine-agent` system property to: `./bin/machine-agent -Dappdynamics.machine.agent.dotnetCompatibilityMode=true`
2. Save the `controller-info.xml` file.
3. Start or restart the Machine Agent to apply the changes.

Extensions and Custom Metrics

Using the Machine Agent, you can supplement the existing metrics in the AppDynamics Controller UI with your own custom metrics. There are many extensions currently available on the AppSphere Community site. Some are created by AppDynamics and some have been created by users.

Similar to built-in metrics, your custom metrics are subject to the following AppDynamics features:

- Automatic baselines and anomaly detection
- Availability to display on custom dashboards
- Availability to use in policies
- Visibility of all metrics in the Metric Browser and on the Infrastructure tab (where you can display external metrics and AppDynamics metrics on the same graph)

Add New Custom Metrics

To create custom metrics, you create a monitoring extension. In your extension, you define the name and path of your metric (where it appears in the metric browser tree), what type of metric it is (sum, average, and so on), and how the data for the metric should be rolled up as it ages. One Agent can run many extensions, although you may need to increase the amount of memory for the JVM Agent. You can have multiple copies of the same extension if they are in different directories.

A custom metric can be common across nodes or associated with a specific tier. When you create a metric, you specify the path in which it will appear in the metric tree. To create a common custom metric, use the root tree path Custom Metrics in your metric declaration. To create a tier-specific metric, specify the metric path associated with that component.

If your application uses a large number of AppDynamics extensions with the Machine Agent, you may need to increase the size of the memory allocation:

For Linux and Unix-like Systems

```
% <machine_agent_home>/bin/machine-agent -Xms64m
```

For Windows

```
> <machine_agent_home>\bin\machine-agent.cmd -Xms64m
```

The wildcard feature is not supported for custom metrics.

Monitoring Extension Types

You can implement custom metrics using these mechanisms:

- Using a script:
  You can write a shell script (Linux and Unix-like systems) or batch file (Windows) to report custom metrics every minute to the Machine Agent. The Machine Agent passes these metrics on to the Controller. See Build a Monitoring Extension Using Scripts.
- Using Java:
  Your custom metrics may be too complicated to collect using a script. For example, you may need to perform complex calculations or call a third-party API to retrieve the metrics. In this case, you can extend the `JavaServersMonitor` class to collect the metrics and report them to the Machine Agent. Your Java program extends the `JavaServersMonitor` class to provide your custom functionality. See Build a Monitoring Extension Using Java.
- Using HTTP:
  If you enable the Agent HTTP listener, you can post HTTP requests to the Machine Agent to send it custom metrics every minute. To do this, start the Machine Agent with a Jetty HTTP listener. See Machine Agent HTTP Listener.

Build a Monitoring Extension Using Scripts

Related pages:

- Extensions and Custom Metrics
- Build a Monitoring Extension Using Java
- Machine Agent HTTP Listener

You can write a monitoring extension script (also known as a custom monitor or hardware monitor) to add custom metrics to the metric set that AppDynamics already collects and reports to the Controller. Your script reports the custom metrics every minute to the Machine Agent. The Machine Agent passes these metrics to the Controller.

This page describes the steps for adding custom metrics using a shell script and includes an example.

Review Existing Extensions

Before creating your own extension, review the extensions that have been created and shared among members of the AppDynamics community. New extensions are added continuously. It is possible that someone has already created what you need or something close enough that you can download and use it after making a few modifications.

See the AppDynamics Exchange for free downloads.

Agent Configuration Requirements

Confirm that you have correctly configured the Machine Agent in the `controller-info.xml` file and on the Agent `start` command on the command line. See Machine Agent Configuration Properties.

Create a Monitoring Extension

To create a monitoring extension using a script:

1. Create your script. See Create the Script File.
2. Create a `monitor.xml` configuration file. See Create the monitor.xml File.
3. Create a subdirectory, `<your_extension_dir>`, in `<machine_agent_home>/monitors`. See Create a directory under the Machine Agent monitors directory.
4. Copy your script file and the `monitor.xml` file into the new subdirectory.
5. Restart the Machine Agent.

Define Your Metrics

Metric names must be unique within the same metric path but need not be unique for the entire metric hierarchy. AppDynamics recommends using short metric names so that the whole name is visible when displayed in the Metric Browser. Prepend the metric path to the metric name when you upload the metrics to the Controller.

Metric Processing Qualifiers

The Controller has various qualifiers for how it processes a metric regarding aggregation, time rollup, and tier rollup. There are three types of metric qualifiers:

1. Aggregator qualifier
2. Time roll-up qualifier
3. Cluster roll-up qualifier

In the script, specify the metric qualifiers after the name-value pair for the metric. A typical metric entry in the script file has the following structure:

```
name=<metric name>,value=<long value>,aggregator=<aggregator type>, time-rollup=<time-rollup strategy>, cluster-rollup=<cluster-rollup strategy>
```

Aggregator Qualifier

The *aggregator* qualifier specifies how the Machine Agent aggregates the values reported during a one-minute period. Specify the aggregator qualifier as a `ggregator="aggregator type"`. This value is an enumerated type. If no value is reported during that minute, no data is reported to the Controller, and an UNCHANGED notice appears in the Machine Agent log for that metric. Valid values are:

Aggregator Type	Description
AVERAGE	(Default) Average of all reported values in that minute
SUM	Sum of all reported values in the minute, causes the metric to behave like a counter

OBSERVATION	Last reported value in the minute

Time Roll-Up Qualifier

The *time-rollup* qualifier specifies how the Controller rolls up the values when it converts from one-minute granularity tables to 10-minute granularity tables, and 60-minute granularity tables over time. The value is an enumerated type. Valid values are:

Roll Up Strategy	Description
AVERAGE	Average of all one-minute values when adding it to the 10-minute granularity table; the average of all 10-minute values when adding it to the 60-minute granularity table
SUM	Sum of all one-minute values when adding it to the 10-minute granularity table; the sum of all 10-minute values when adding it to the 60-minute granularity table
CURRENT	Last reported one-minute value in that 10-minute interval; the last reported 10-minute value in that 60-minute interval

Cluster Roll-Up Qualifier

The *cluster-rollup* qualifier specifies how the Controller aggregates metric values in a tier (a cluster of nodes). The value is an enumerated type. Valid values are:

Roll up Strategy	Description
INDIVIDUAL	Aggregates the metric value by averaging the metric values across each node in the tier
COLLECTIVE	Aggregates the metric value by adding up the metric values for all the nodes in the tier

For example, if a tier has two nodes, Node A and Node B, and Node A has three errors per minute and Node B has seven errors per minute, then the INDIVIDUAL qualifier reports a value of five errors per minute, and COLLECTIVE qualifier reports ten errors per minute. INDIVIDUAL is appropriate for metrics such as % CPU Busy, where you want the value for each node. COLLECTIVE is appropriate for metrics, such as Number of Calls, where you want a value for the entire tier.

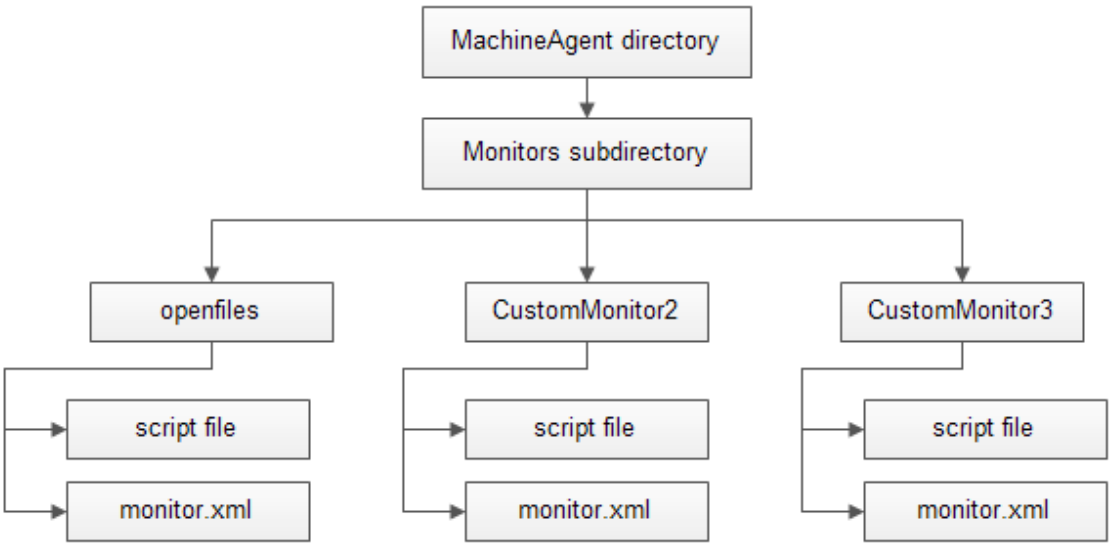
Add a Monitoring Extension Script

To add a monitoring extension script:

Create a Subdirectory Under the Machine Agent Monitors Directory

The `<machine_agent_home>/monitors` directory is the repository for the Machine Agent extensions. For each new extension, create a subdirectory under the `/monitors` directory. The user running the Agent requires read, write, and execute permissions to this subdirectory.

For example to create an extension that monitors open files in the JVM, create a subdirectory named "`openfiles`" under `<machine_agent_home>/monitors`. The structure looks like:



Create the Script File

A script writes data to STDOUT. The Machine Agent parses STDOUT and sends information to the Controller every minute. Use these instructions to create the script file:

> **For Windows custom metrics, PowerShell and VBScript are recommended over .bat files**
>
> To generate custom metrics on Windows, AppDynamics recommends that you use PowerShell and VBasic scripts instead of `.bat` files. When a standard Windows batch (`.bat`) script echoes metric names, it surrounds the names with quotes. The quotes will cause the Machine Agent to ignore these metrics. PowerShell and VBasic scripts do not have this issue.

1. Specify a `name-value` pair for the metrics.
   Each metric has a `name-value` pair that is converted to a java `'long'` value. A typical metric entry in the script file has this structure:

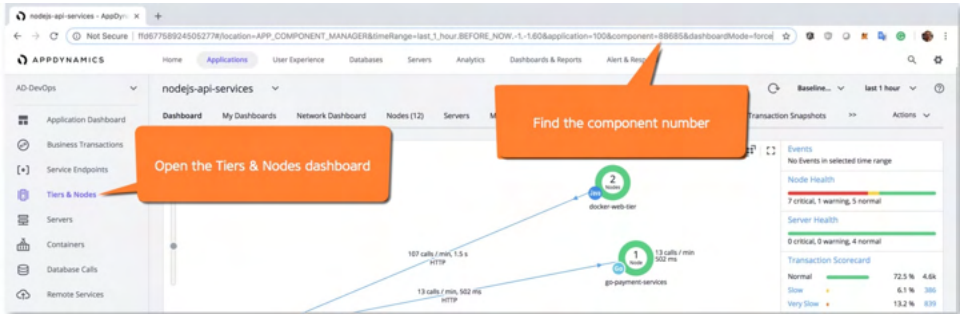
   ```
   name=<metric name>,value=<long value>,aggregator=<aggregator type>, time-rollup=<time-rollup strategy>,
   cluster-rollup=<cluster-rollup strategy>
   ```

   Use the following format:

Form	Format
Standard Form	`Hardware Resources| Instrument Name=Instrument Value`
Fully Qualified Form	`Hardware Resources| <metric name>,value=<long value>`

2. Define the category of the metric, for example:
   a. Infrastructure (for the default hardware metrics, see Machine Agent)
   b. JVM
   c. Custom Metrics, where Custom Metrics must have the path prefixes:
      - Custom Metrics
      - Server|Component:<*tier-name-or-tier-id*>
3. Metrics with the Custom Metrics prefix are common across all tiers in your application. Metrics with the Server|Component:<*tier-name-or-tier-id*> prefix appear only under the specified tier.

   To find the component ID of a tier, open the dashboard for the tier and review the URL. The ID appears as the component value in the URL:



The Machine Agent has to be associated with the target or destination for the metrics. If you attempt to publish metrics to a tier that is not associated with the Machine Agent, the metrics are not reported.

The "|" character separates the branches in the metric hierarchy notifying the Controller where the metric should appear in the metric tree:

```
Custom Metrics|Hardware Resources|Disks|Total Disk Usage %
Custom Metrics|Hardware Resources|Disks|Disk 1|Current Disk Usage %
```

You can insert a custom metric next to an existing type of metric. For example, the following declaration causes the custom metric named `pool usage` to appear next to the `JMX` metrics:
Server|Component:18|JMX|Pool|First|pool usage

4. You can then use the metric in health rules as would other types of JMX metrics.
   To monitor multiple metrics with the same script file, edit the script to write a different line for each one to STDOUT. For example:

```
name=Custom Metrics|Hardware Resources|Disks|Total Disk Usage %, value=23
name=Custom Metrics|Hardware Resources|Disks|Disk 1|Current Disk Usage %, value=56
```

Copy the Script File to the Subdirectory Created in Step 1

Ensure that the Agent process has execute permissions for the script file and for the contents of the file.

Create the monitor.xml File

For each custom monitoring extension script, create a `monitor.xml` file. The `monitor.xml` file executes the script file created in Step 2. You can edit the following sample file to create your own file:

```
<monitor>
    <name>HardwareMonitor</name>
       <type>managed</type>
        <description>Monitors system resources - CPU, Memory, Network I/O, and Disk I/O.</description>
        <monitor-configuration>     </monitor-configuration>
        <monitor-run-task>
           <!--  Edit execution-style as needed. -->
           <execution-style>continuous</execution-style>
           <name>Run</name>
           <type>executable</type>
           <task-arguments></task-arguments>
            <executable-task>
                <type>file</type>
                <!--  Use only one file element per os-type. -->
                 <file os-type="linux">linux-stat.sh</file>
                 <file os-type="mac">macos-stat.sh</file>
                 <file os-type="windows">windows-stat.bat</file>
                 <file os-type="solaris">solaris-stat.sh</file>
                 <file os-type="sunos">solaris-stat.sh</file>
                 <file os-type="aix">aix-stat.sh</file>
           </executable-task>
         </monitor-run-task>
</monitor>
```

The `os-type` attribute is optional for the `executable-task` file element when only one `os-type` is specified. One `monitor.xml` file executes one script per `os-type`.

1. Select the execution style: `continuous` or `periodic`.

Execution Style	Description	Example

continuous	Select `continuous` if you want data collection averaged over time. For example, average CPU usage over a minute.
	For the monitor to be declared as '`continuous`', the script should also run in an infinite loop. This ensures that the script keeps running until the Machine Agent process is terminated.
		`while [ 1 ]; do`
		... the actual script goes here ...
		`sleep 60`
		`done`

periodic	Select `periodic` to report data from system performance counters periodically. The periodic task runs every minute by default, and the data is aggregated.
	
	To specify a different frequency, use the `execution-frequency-in-seconds` element. The execution frequency must be less than 60. For periodic execution style, you can also specify the timeout setting (shown in the example).
		`<monitor-run-task>`
		
		`...`
		
		`<execution-style>periodic</execution-style>`
		`<execution-frequency-in-seconds>30<`
		`/execution-frequency-in-seconds> <execution-timeout-in-secs>30</execution-timeout-in-secs>`
		
		`...`
		
		`</monitor-run-task>`

2. Add the name of your script file to the `<file>` element in the `monitor.xml` file. Be sure to use the correct `os-type` attribute. The `os-type` value should match the value returned from calling `System.getProperty("os.name")`.

```
<file os-type="your-os-type">{script file name}</file>
```

   You can use either the relative or absolute path of the script.

Copy the monitor.xml file to the Subdirectory Created in Step 1

Restart the Machine Agent

> **Required Agent Properties**
>
> Ensure that you have correctly configured the Agent in the `controller-info.xml` file and on the Agent `start` command on the command line. See Database Agent Configuration Properties.

After restarting the Machine Agent, you should see following message in your log file:

```
Executing script [<script_name>] on the console to make sure your changes work with the machine agent.
```

Verify Execution of the Monitoring Extension Script

To verify the execution of extension, wait at least one minute and check the metric data in the Metric Browser.

You can now create alerts based on any of these metrics.

Example: Create a Monitoring Extension for Open Files

To create a custom monitor for monitoring all the open files for JVMs:

1. Create a new directory in the custom monitor repository.
2. Create the script file. You can review these two examples:

   Modify this UNIX script for the specific process name (for example: Author, Publish, and so on).

```
lookfor="<process name 1>"
pid=`ps aux | grep "$lookfor" | grep -v grep | tr -s " " | cut -f2 -d' '`
count1=`lsof -p $pid | wc -l | xargs`

lookfor="<process name 2>"
pid=`ps aux | grep "$lookfor" | grep -v grep | tr -s " " | cut -f2 -d' '`
count2=`lsof -p $pid | wc -l | xargs`

echo "name=JVM|Files|<process name 1>,value="$count1
echo "name=JVM|Files|<process name 2>,value="$count2
```

The following Windows `.bat` example reports a metric to the Controller if it a Java process is running on the machine.

To generate custom metrics on Windows, AppDynamics recommends that you use PowerShell and VBasic scripts instead of `.bat` files.
When a standard Windows batch (`.bat`) script echoes metric names, it surrounds the names with quotes. The quotes will cause the Machine Agent to ignore these metrics. PowerShell and VBasic scripts do not have this issue.

```
SETLOCAL enabledelayedexpansion

REM  Check to see if there is a java process running
TASKLIST /FI "IMAGENAME eq java.exe" 2>NUL | find /I /N "java.exe">NUL
if "%ERRORLEVEL%"=="0" (
    SET metric="Custom Metrics|Process|java|Running,value=1"
    REM This strips the quotes that are added by the bat script
    REM so that the machine agent can understand the metric
    @echo !metric:"=!
)
```

3. Create the following `monitor.xml` file and point it to the UNIX script shown in Step 2.

```
<monitor>
    <name>MyMonitors</name>
    <type>managed</type>
    <description>Monitor open file count </description>
    <monitor-configuration>
    </monitor-configuration>
    <monitor-run-task>
        <execution-style>continuous</execution-style>
        <name>Run</name>
        <type>executable</type>
        <task-arguments>
        </task-arguments>
        <executable-task>
            <type>file</type>
            <file>openfilecount.sh</file>
        </executable-task>
    </monitor-run-task>
</monitor>
```

Build a Monitoring Extension Using Java

Related pages:

- Machine Agent Configuration Properties
- Build a Monitoring Extension Using Scripts

You can create Java monitoring extensions that enable the Machine Agent and Server Visibility to collect custom metrics, which you define and provide, and report them to the Controller. This is an alternative to adding monitoring extensions using scripts.

When you capture custom metrics with a monitoring extension, they are supported by the same AppDynamics services that you receive for the standard metrics captured with the AppDynamics application and Machine Agents. These services include automatic baselining, anomaly detection, display in the Metric Browser, availability for display on custom dashboards, and availability for use in policies to trigger alerts and other actions.

This page describes how to create a monitoring extension in Java.

To the Agent, a monitoring extension is a task that runs on a fixed schedule and collects metrics.

Before You Begin

Before creating your own extension from scratch, review the extensions on the AppDynamics community. See GitHub for a free download of the extensions

New extensions are constantly being added. It is possible that someone has already created what you need, or something close enough, that you can download it and use it after making a few modifications.

Create a Monitoring Extension in Java

To create a monitoring extension in Java:

1. Create your extension class. See Create the Monitoring Extension.
2. Create a `monitor.xml` configuration file. See Create the monitor.xml File.
3. Create a subdirectory (<*`your_extension_dir`*>) under <*`agent_home`*>`/monitors`.
4. Place the extension class file and the `monitor.xml` file (plus any dependent `jar` files) in <*`your_extension_dir`*>.
5. Enter the Controller access information and credentials. See Configure the Machine Agent.

   > Make sure that your `controller-info` and command line parameters are correctly configured. Required properties include: Controller name, port number, and account access key.

6. Restart the Machine Agent.

Create the Monitoring Extension Class

Create a monitoring extension class by extending the `AManagedMonitor` class in the `com.singularity.ee.agent.systemagent.api` package. This package is included in the `MachineAgent.jar` file.

The monitoring extension class performs these tasks:

- Populates a hash map with the values of the metrics that you want to add to AppDynamics. You obtain these metrics based on your environment and the source from which you derive your custom metrics.
- Defines the type of metrics to collect using the `MetricWriter` class. See Metric Processing Qualifier.
- Uploads the metrics to the Controller using the `execute()` method of the `AManagedMonitor` class.

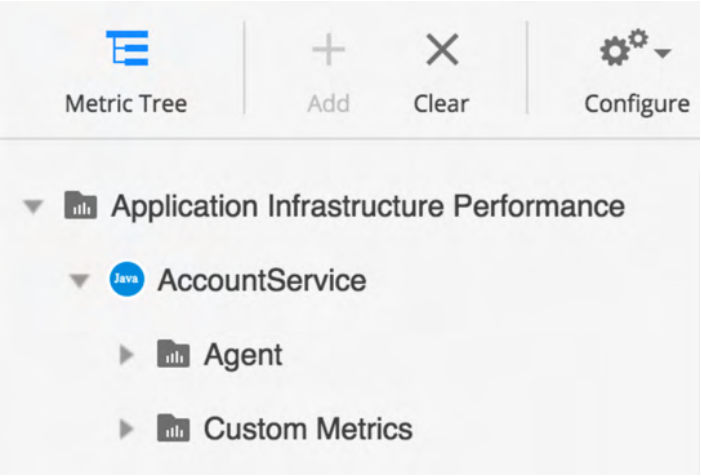
See Extension_Class_Source.txt for example source code.

Metric Path

All custom metrics processed by the Machine Agent and Server Visibility appear in the **Metric Browser > Application Infrastructure Performance**. Use the "|" character to specify the path from Application Infrastructure Performance to the custom metric. If the metrics apply to a specific tier, use the metric path for the tier, with "Component" followed by a colon ":" and the tier name or tier ID.

For example to associate a metric with tier AccountService, you would specify the metric path as: `Server|Component:AccountService|Custom Metrics|Path|`

The metric would then appear under the tree in the Metric Browser:



You can report a custom metric only to the tier that is associated with the Machine Agent. If you attempt to publish metrics to a different tier, the metrics are not reported.

You can insert a custom metric next to an existing type of metric. For example, the following declaration causes the custom metric named pool usage to appear next to the JMX metrics: name="Server|Component:<*tier-name*>|JMX|Pool|First|pool usage",value=10

You can then use the metric in health rules similar to other types of JMX metrics.

To test the appearance of your custom metric in the Controller API. post the metric data to the Machine Agent's REST API. Pass the path, name type and values of the metric as URL arguments. See Machine Agent HTTP Listener.

Metric Names

Metric names must be unique within the same metric path but need not be unique for the entire metric hierarchy.

AppDynamics recommends that you use short metric names so that they are visible when they are displayed in the Metric Browser.

Prepend the metric path to the metric name when you upload the metrics to the Controller.

Metric Processing Qualifier

The Controller has various qualifiers for how it processes a metric regarding aggregation, time roll-up, and tier roll-up.

There are three types of metric qualifiers:

- Aggregator qualifier
- Time roll-up qualifier
- Cluster roll-up qualifier

You specify these options with the enumerated types provided by the MetricWriter class.

Aggregation Qualifier

The *aggregator* qualifier specifies how the Machine Agent aggregates the values reported during a one-minute period.

Specify the aggregation qualifier as `aggregator="aggregator type"`

This value is an enumerated type. Valid values are:

Aggregator Type	Description
`METRIC_AGGREGATION_TYPE_AVERAGE`	Default. Average of all reported values in that minute.
`METRIC_AGGREGATION_TYPE_SUM`	Sum of all reported values in that minute, causes the metric to behave like a counter.
`METRIC_AGGREGATION_TYPE_OBSERVATION`	Last reported value in the minute. If no value is reported in that minute, the last reported value is used.

Time Roll-Up

The *time-rollup* qualifier specifies how the Controller rolls up the values when it converts from one-minute granularity tables to 10-minute granularity tables, and 60-minute granularity tables over time.

Roll up Strategy	Description
`METRIC_TIME_ROLLUP_TYPE_AVERAGE`	Average of all one-minute data points when adding it to the 10-minute or 60-minute granularity table.
`METRIC_TIME_ROLLUP_TYPE_SUM`	Sum of all one-minute data points when adding it to the 10-minute or 60-minute granularity table.
`METRIC_TIME_ROLLUP_TYPE_CURRENT`	Last reported one-minute data point in that 10-minute or 60-minute interval.

Cluster Roll-Up

The *cluster-rollup* qualifier specifies how the Controller aggregates metric values in a tier.

Roll up Strategy	Description
`METRIC_CLUSTER_ROLLUP_TYPE_INDIVIDUAL`	Aggregates the metric value by averaging the metric values across each node in the tier.
`METRIC_CLUSTER_ROLLUP_TYPE_COLLECTIVE`	Aggregates the metric value by adding up the metric values for all the nodes in the tier.

For example, if a tier has two nodes, Node A and Node B, and Node A has three errors per minute and Node B has seven errors per minute, the INDIVIDUAL qualifier reports a value of five errors per minute and COLLECTIVE qualifier reports ten errors per minute. INDIVIDUAL is appropriate for metrics such as % CPU Busy where you want the value for each node. COLLECTIVE is appropriate for metrics such as Number of Calls where you want a value for the entire tier.

Sample Monitoring Extension Class

The `NGinXMonitor` class retrieves the following metrics from the Nginx Web Server, and adds them to the metrics reported by AppDynamics:

- `Active Connections:` number of active connections
- `Accepts:` number of accepted requests
- `Handled:` number of handled requests
- `Requests:` total number of requests
- `Reading:` number of reads
- `Writing:` number of writes
- `Waiting:` number of keep-alive connections

The source for the extension class is included as an attachment.

Create monitor.xml

Create a `monitor.xml` file with a `<monitor>` element to configure how the Machine Agent executes the extension.

1. Set the `<name>` to the name of your Java monitoring extension class.
2. Set the `<type>` to `managed`.
3. The `<execution-style>` can be `continuous` or `periodic.`
    - Continuous means to collect the metrics averaged over time; for example, average CPU usage per minute. In continuous execution, the Machine Agent invokes the extension once and the program runs continuously, returning data every 60 seconds.
    - Periodic means to invoke the monitor at a specified frequency. In periodic execution, the Machine Agent invokes the extension, runs it briefly, and returns the data on the schedule set by the `<execution-frequency-in-seconds>` element.

    > Do not set the `<execution-frequency-in-seconds>` greater than 300 seconds (five minutes). The extension must collect metrics at least once every five minutes.

4. If you select `periodic` for the execution style, set the frequency of collection in `<execution-timeout-in-secs>` element. The default frequency is 60 seconds.
   If you select `continuous`, this setting is ignored.
5. Set the `<type>` in the `<monitor-run-task>` child element to `java`.
6. Set the `<execution-timeout-in-secs>` to the number of seconds before the extension times out.
7. Specify any required task arguments in the `<task-arguments>` element. The default arguments that are specified here are the only arguments that the extension uses. They are not set anywhere else.
8. Set the `<classpath>` to the `jar` file that contains your extension's classes. Include any dependent `jar` files, separated by semicolons.
9. Set the `<impl-class>` to the full path of the class that the Machine Agent invokes.

Sample monitor.xml Files

This attached monitor.xml file configures the NGinXMonitor monitoring extension. This extension executes every 60 seconds.

This attached monitor.xml file configures the MysqlMonitor. This monitor executes every 60 seconds, has four required task arguments and one optional task argument and one dependent `jar` file.

Standalone Machine Agent HTTP Listener

You can send metrics to the Machine Agent using its HTTP listener. You can report metrics through the Machine Agent by making HTTP calls to the Agent instead of piping to the Agent through `sysout`.

Activate the HTTP Listener

The HTTP listener is not enabled by default. To activate the HTTP listener, restart the Machine Agent and set the `metric.http.listener` system property to `true`. Optionally, you can specify the port for the listener with system property. The HTTP listener can only accept calls from localhost.

The `-D system properties` are:

- `metric.http.listener:` Required. Set to `true`.
- `metric.http.listener.port:` Optional. Set to the port to be used, defaults to 8293.

```
<machine_agent_home>/bin/machine-agent -Dmetric.http.listener=true -Dmetric.http.listener.port=<port_number> -
Dmetric.http.listener.host=0.0.0.0
```

If starting the Agent by invoking the Machine Agent JAR, ensure that you place the options before the `JAR` name in your start-up command. For example:

```
java -Dmetric.http.listener=true  -jar MACHINE_AGENT_HOME/machineagent.jar
```

Create Metrics

Use to post custom metrics to the Machine Agent for uploading to the Controller. Define one or more metrics in the body of the request as `JSON` data.

URI

POST /api/v1/metrics

Metric Definition Fields

Parameter	Description
`metricName`	Name for the metric as it will appear in the Controller UI.
`aggregatorType`	How the metrics should be aggregated. Options are: • `AVERAGE`: The average of all one-minute data points when adding it to the 10-minute or 60-minute granularity table. • `SUM:` The sum of all one-minute data points when adding it to the 10-minute or 60-minute granularity table. • `OBSERVATION:` Last reported one-minute data point in that 10-minute or 60-minute interval.
`Value`	A 64-bit integer value for the metric.

Format

```
POST /api/v1/metrics

 [

  {

    "metricName": "Custom Metrics|Test|MetricFromRESTClient1",

    "aggregatorType": "AVERAGE",

    "value": 10

  }

]
```

Create Events

Use to post custom events to the Machine Agent for uploading to the Controller. Define one or more events in the body of the request as `JSON` data.

URI

POST /api/v1/events

Event Definition Fields

Parameter Name	Description
`eventSeverity`	Severity of the event, from INFO, WARN, ERROR.
`type`	A string describing the event subtype. The event will be of type CUSTOM, subtype will be the value of this parameter.
`summaryMessage`	A summary of the event.
`properties`	Event properties. These properties are retrieved along with events by the Controller in a given query and provide a means for filtering the events. The maximum size of a key is 500 characters and the maximum size of a value is 5000 characters. The values can be a numeric or string value.
`details`	Arbitrary key-value details for the event; similar in constraints to properties but are retrieved in a separate call. Use this to store details that should only be retrieved when requested by the user, which avoids the expense of retrieving this data in the usual event calls.

Format

```
POST /api/v1/events
[
  {
    "eventSeverity": <event_severity>,
    "type": "<event_type>",
    "summaryMessage": "<event_summary>",
    "properties": {
      "<key>": {
        <user-specified_object>
      },...
    },
    "details": {
      "<key>": "<value>"
    }
  },
  {
    "eventSeverity": <event_severity>,
    "type": "<event_type>",
    "summaryMessage": "event_summary>",
    "properties": {
      "<key>": {
        <user-specified_object>
      },...
    },
    "details": {
      "<key>": "<value>"
    }
  },...
]
```

Legacy Machine Agent HTTP APIs

The following API endpoints are supported for backward compatibility, but are not extended or enhanced in future versions.

Upload Metrics

You can use `GET` or `POST` to upload metrics to the Metric Browser under **Application Performance > *Tier***, where the tier is the one defined for the Machine Agent.

The format for `GET` is:

```
GET /machineagent/metrics
```

For example:

```
http://host:port/machineagent/metrics?name=Custom Metrics|Test|My Metric&value=42&type=average
```

The format for `POST` is:

```
POST /machineagent/metrics
```

with header:

```
Content-Type: application/xml
```

with body content:

```
<?xml version="1.0"?>
<request>
<metric name="[name of metric 1]", type="[aggregation type]", value="[value of metric 1]" />
<metric name="[name of metric 2]", type="[aggregation type]", value="[value of metric 2]" />
...
<metric name="[name of metric n]", type="[aggregation type]", value="[value of metric n]" />
</request>
```

Example:

```
http://host:port/machineagent/metrics
```

Example of body content:

```
<request>
<metric name="Custom Metrics|Test|My Metric 1", type="AVERAGE", value="22" />
<metric name="Custom Metrics|Test|My Metric 2", type="SUM", value="98737" />
<metric name="Custom Metrics|Test|My Metric 3", type="CURRENT", value="93" />
</request>
```

Valid values for type are:

- `AVERAGE:` Average of all one-minute data points when adding it to the 10-minute or 60-minute granularity table.
- `SUM:` Sum of all one-minute data points when adding it to the 10-minute or 60-minute granularity table.
- `CURRENT`: Last reported one-minute data point in that 10-minute or 60-minute interval.

Upload Events

Send events using HTTP `GET` requests to upload events to the Machine Agent. The format is:

```
GET /machineagent/event
```

For example:

```
http://localhost:8293/machineagent/event?type=<event_type>&summary=<summary_text>
```

`Event_type` is one of the following:

- error
- info
- warning

Administer the Standalone Machine Agent

These pages describe procedures and provide reference material relevant to the AppDynamics Pro Administrator.

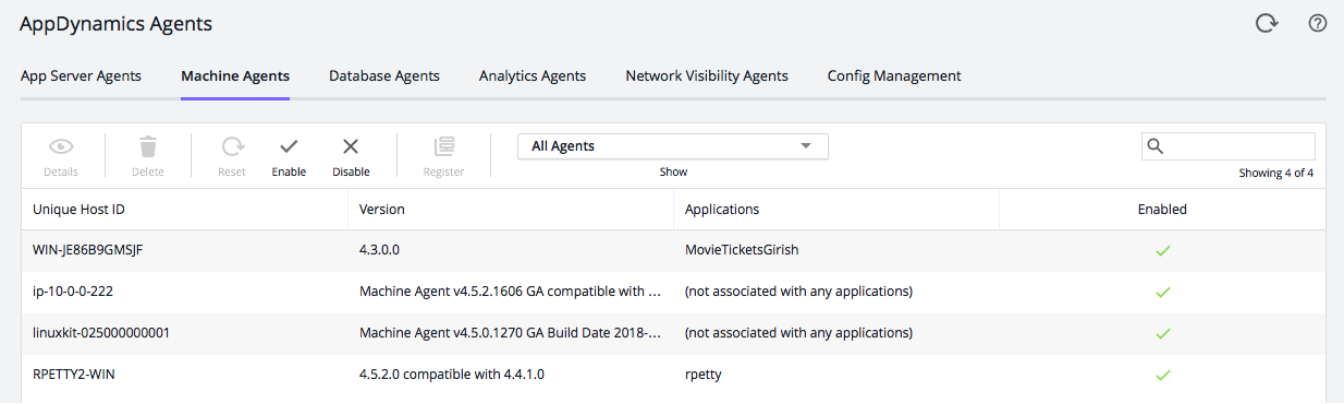
Manage Machine Agents

This page describes how to manage Machine Agents.

View Details

Using an AppDynamics Administrator account, click **Settings > AppDynamics Agents > Machine Agents** to see details and manage your installed Machine Agents.

- **Hostname**: Name of the machine on which the Agent is installed.
- **Version**: Version of the installed Machine Agent software.
- **Applications**: Name of the application with which the Agent is associated.
- **Enabled:** Whether the Agent is enabled.



Available Actions

Reset Selected Machine Agent

Restarts the selected Machine Agent and runs it with any updated configuration files on the Agent host, and any changes made on the Controller that affect the Agent. The reset functionality is only available for Machine Agents <= 4.2.

View Machine

Shows summary details about the Machine Agent:

- **Name:** Name of the machine on which the Agent is installed
- **IP Address:** IP address of the machine on which the Agent is installed
- **OS:** Operating system running the Agent
- **Status:** Field displays status information: Initialized
- **Install Time:** Date and time Agent was installed
- **Last Start Time:** Date and time Agent was last restarted

Disable and Enable Agents

Disabling the Agent temporarily suspends its collection of metrics. You can quickly re-enable the Agent when desired.

> **Permission:**
>
> You must have the application-level permission, *configure agent properties*, to enable and disable Agents.

When the Machine Agent is disabled, the Agent stops reporting hardware metrics and metrics from any extensions installed on the Agent until you re-enable it. The disabled Machine Agent no longer reports events and any installed extensions are also stopped. While disabled, the Agent continues to consume a license. The Agent maintains a 'heartbeat' connection to the Controller so that it can be 'enabled' again quickly. The Agent persists its enabled /disabled state even after the Machine Agent is restarted. Once re-enabled, all normal Machine Agent activities resume.

Delete Selected Machine Agent

The delete functionality on the admin page is to delete the underlying machine metadata for an orphaned machine. A machine is considered "orphaned" when it is no longer associated with any APM nodes or with Server Availability. You cannot delete a machine that is not an orphan. To delete a machine that is not an orphan, you must first delete all nodes on that machine, and then delete the machine from the Servers list. You can select multiple machines to delete.

You can also delete a Machine Agent from the Servers List.

- Server Visibility Administrator permission is required to delete machines.
- You can delete machines from the Servers list regardless of whether they have Server Visibility enabled. You can select multiple machines to delete.
- Deleting a machine from the Servers list deletes the underlying machine metadata that displays on the AppDynamics Agents Admin page, but only if no APM nodes are currently associated with the same machine.
- Deleting a machine that still has APM nodes currently associated with it removes the entry from the Servers list, but retains the underlying machine metadata. That information continues to display on the AppDynamics Agents admin page.

Upgrade the Standalone Machine Agent

This page describes how to upgrade an existing installation of the Machine Agent. The instructions have been tested and certified to work for Machine Agent >= 4.3.

For a new installation, see Install the Machine Agent.

> **Machine Agent >= 4.3 Recommended**
>
> AppDynamics recommends using Machine Agent >= 4.3 for this release. Machine Agent <= 4.1 is no longer supported. Although Machine Agents may be compatible with future releases of the Controller, AppDynamics only provides support for Machine Agent versions with the Controller for two years after the release of the Agent.

Perform the Upgrade

1. Do the following (as needed) before you update the Agent:
    a. Review the Release Notes for changes that affect your environment.
    b. Back up the *<machine_agent_home>* directory to a *<machine_agent_home_BACKUP>* so you can revert to the previous installation, if required.
    c. *If you are upgrading the Controller as well as the Agents:* Upgrade the Controller Using the Enterprise Console, and then return to this procedure.
    d. *If you have been collecting Log Analytics data by running the Analytics Agent via the Machine Agent:* Do the following.
        i. Preserve the old watermark file so that you can copy it to the new *<analytics-agent-home>*.
            Machine Agent location: *<machine-agent-home>*`/monitors/analytics-agent/conf/watermark`
            Analytics Agent location: *<analytics-agent-home>*`/conf/watermark`
        ii. Preserve your pre-existing job files so that you can copy them to the new *<analytics-agent-home>*.
            Machine Agent location: *<machine-agent-home>*`/monitors/analytics-agent/conf/job`
            Analytics Agent location: *<analytics-agent-home>*/conf/job
2. Download the installer suitable for your OS environment from the AppDynamics Download Center.
3. Stop all instances of the Machine Agent that are currently running from the *<machine_agent_home>* that you want to upgrade. See Start and Stop the Machine Agent
4. Update your Machine Agent installation:
    - RPM-based Linux installation:
      Update the existing package. For example: `sudo rpm -U <package>`
    - RPM-based installation on a systemd Linux system (such as CentOS 7, Ubuntu, newer Fedora Core systems, newer OpenSUSE systems, and newer SUSE Enterprise Linux systems).
      If you don't know if your Linux installation is using systemd, do a clean install to avoid issues. Remove the old package and then install the new one.
    - For all other scenarios:
      First, delete the *<machine_agent_home>* for the old Agent. Make sure you retain the *<machine_agent_home_BACKUP>* you created in Step 1.
      Then, install the Machine Agent as described Linux Install Using the RPM Package for your specific installation in Install the Machine Agent.

      > If you are running the Analytics Agent as an extension to the Machine Agent, failure to clear out the old directory or use a new location may cause your embedded Analytics Agent to fail to start.

5. Configure the Agent settings in *<machine_agent_home>*`/conf/controller-info.xml`. See Machine Agent Configuration Properties

    > The RPM-based Linux installer retains the configuration settings for the previous Agent by default.
    >
    > If you used another installer and want to maintain your previous Agent configuration, copy the *<machine_agent_home_BACKUP>*`/conf/controller-info.xml` file to the new installation directory.

6. Before you start the updated Agent, perform the following steps (as required):
    a. *If you are updating on Windows:* When you install the new Machine Agent in the existing directory, you may see two "`InstallService`" and "`UninstallService`" files. Use the `.vbs` files (VbScript) and delete the `.cmd` files (Windows Command Script), which are leftover from your previous version.
    b. *If the agent install package includes a new JRE:* Update any related custom scripts that use the Machine Agent JRE.
    c. Update all users who have the Account Administrator role for the account and add the appropriate SIM roles (SIM admin and SIM user) to other users in the account, as needed.
7. Start the Agent, verify the Agent installation, and verify that the Agent is reporting to the Controller as described for your specific installation in Install the Machine Agent.

Uninstall the Standalone Machine Agent

To uninstall the Machine Agent:

1. Stop the Machine Agent (or service). For the commands for your environment, see Start and Stop the Machine Agent.
2. If you installed the Machine Agent as a service, delete the service using the commands described on this page.
   For Windows and Mac OS X, the commands to remove the service also stop the service cleanly.
3. Delete the `<machine-agent-install>` installation directory.

Remove the Agent Service

Linux RPM

Execute the following command:

```
sudo rpm -ev appdynamics-machine-agent
```

Windows

Execute the following command:

```
> UninstallService.vbs
```

Mac OS X

Execute the following command:

```
sudo launchctl unload -w <machine_agent_home>/com.appdynamics.machineagent.plist
```

Delete Using AppDynamics Manage Agents

If you delete a Machine Agent from the Controller UI (as described in Manage Machine Agents), but do not shut down the JVM that the Machine Agent runs on, the Machine Agent reappears in the UI the next time it connects to the Controller. To prevent a Machine Agent from connecting to the Controller, you must remove the Machine Agent settings from the JVM configuration. This frees the license associated with the Agent in the Controller and makes it available for use by another Machine Agent.

FAQs and Troubleshooting for the Machine Agent

Machine Agent and App Agent on the Same Machine, but Report to Different Nodes

Issue

A machine has a Machine Agent and an App Server Agent running together, however, the Agents report to different nodes. When you review the Metric Browser, for example, the application metrics appear under node *abc* but machine metrics appear under node *abc.mydomain.com*.

Resolution

Check if the hostnames match. If they do not match, then:

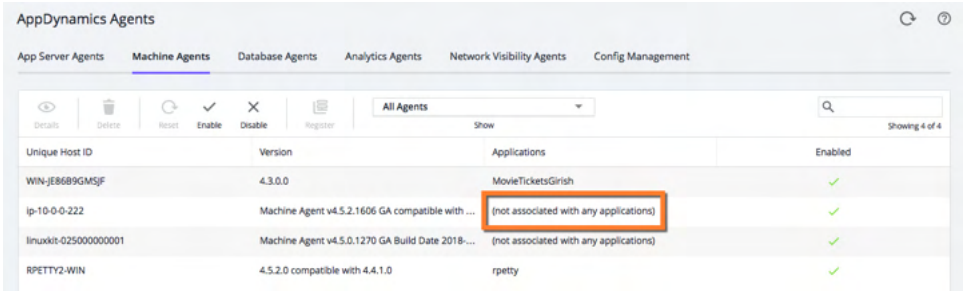
1. Set the `<unique-host-id>` for the Machine Agent to the same `<unique-host-id>` as the App Server Agent.
2. Restart the Machine Agent to apply the previous changes.

The `<unique-host-id>` on the Machine Agent and the `<unique-host-id>` for the App Server Agent must match exactly. These settings are case-sensitive and ensure that both Agents report metrics to the same node.

To verify that the hostnames match:

1. Click the gear icon ( ) in the Controller menu and select **AppDynamics Agents**.
2. Select the Machine Agents tab.
3. Review the **Applications** field for the Agent name.

   In addition, for consistency in naming in the Servers tab, use the `<machine path>` attribute to set the hostname. By using the machine path, you do not need to use the fully qualified domain name (FQDN) for some Agents, and the hostname for others. The reported hostname is still available to view by checking the option in the Agents Admin view.



4. Review the ID for the App Server Agent:
   a. Go to the App Server Agents tab.
   b. Search for the App Server Agent to which you want to associate the Machine Agent.
   c. Review the **Unique Host ID** field for the App Server Agent installed on that machine.
5. Set the `<unique-host-id>` for the Machine Agent to the same `<unique-host-id>` as the App Server Agent.
6. Restart the Machine Agent to apply the previous changes. Your Agents will appear as associated after a few minutes

Identify the Machine Agent Process

Linux and Unix-like Systems
To identify which process is running the Machine Agent, enter:

```
ps -ef | grep machine
```

Windows Systems

In the Windows Services Application, search for the AppDynamics Machine Agent service.

Metric Values for Disk and Network are Zero

When a 32-bit JRE is used with a 64-bit operating system, you may see values of zero for disk and network metrics. To solve this problem, use a 64-bit JRE with the 64-bit operating system.

Flow Maps and the Machine Agent

The Machine Agent monitors a particular machine (also called "host server") and not a particular application server; therefore, the Machine Agent can report to multiple nodes running on the same machine. However, a flow map displays the communication between different nodes during application execution, or the business transaction flow from tier to tier. A Machine Agent cannot be a part of the flow and is not shown in the flow map.

Server Health Indicator

Health rule violations based on metrics monitored by the Machine Agent are included in the health indicator on the various application, tier, and node dashboards. The health indicator is driven by health rule violations in the given time period configured on hardware metrics collected by the Machine Agent. Some health rules are configured by default. To configure additional health rules, see Configure Health Rules and Configure Health Rules to Monitor Servers.

Server Visibility

Related pages:

- Enable Server Visibility
- Monitor Your Servers Using Server Visibility
- Hardware Resources Metrics

The AppDynamics Server Visibility module uses the Machine Agent to provide extended hardware metrics. Server Visibility includes additional windows in the Controller UI that enable you to quickly see underlying infrastructure issues impacting your application performance and rapidly troubleshoot hardware performance problems.

Requirements

- Linux, Windows, Solaris, and AIX
- Server Visibility license for monitored machines
- Server Visibility enabled on the Machine Agent

> Before you enable Server Visibility:
>
> 1. Review the Server Visibility Requirements and Supported Environments.
> 2. Install and configure the Machine Agent on a machine with a supported OS as described in Install the Machine Agent.
>
> Without having to enable Server Visibility, you can view basic hardware metrics.

Feature Comparison

Feature	Server Visibility	Machine Agent
Supported Platforms	Linux, Windows, Solaris, AIX	Linux and Unix-like systems, MacOS, Windows, AIX
Extensions and Custom Metrics	Yes	Yes
Monitor Docker container metrics from the Docker host	Yes	No
Tier Metric Correlator	Yes	No
Dynamic Monitoring Mode	Yes	No
JVM Crash Guard	Yes	Yes
Remediation Scripts	Yes	Yes
Graphical User Interface	Yes	Functionality is limited to viewing Machine Agent metrics in the Servers List
Extended Hardware Metrics	Yes	No
Server metrics in transaction snapshots	Yes	No
Server KPIs in application flow map	Yes	No
Health rules for extended hardware metrics	Yes	No
Licensing	Server Visibility	Machine Agent

Permissions

Server Visibility provides two predefined roles, and two related permissions:

- Server Visibility Administrator
- Server Visibility User
- View Server Visibility permission
- Configure Server Visibility permission - enables configuring Service Availability

See Server Visibility Permissions.

Server Visibility User Interface

The Servers list provides key performance metrics for all your servers on a single panel.

The Servers > Dashboard shows key performance metrics for the selected machine, including the top ten consumers of CPU and memory.

The Servers > Volumes panel displays performance metrics for disks, partitions, and volumes for the selected machine.

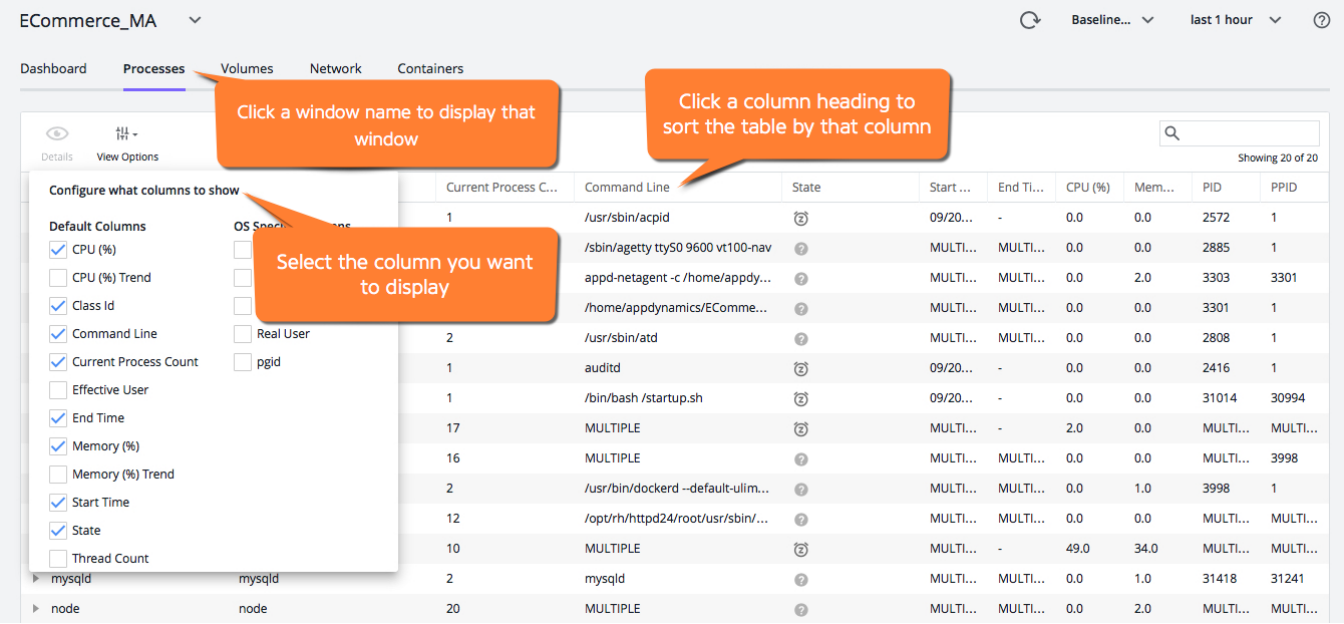
The Servers > Network panel shows network performance metrics for network interfaces on the selected machine.

The Servers > Processes panel shows a configurable set of performance metrics for processes.

The Servers > Containers panel lists the monitored containers used by an application.

Using the Server Visibility UI

The Server Visibility user interface uses many of the same mechanisms that are common to the various panels of the Controller UI. The sample image illustrates these mechanisms.



Search Results and Sorting

Pagination is the default behavior on the **Servers** and **Containers** list pages. The information is retrieved on demand, and additional records are retrieved and displayed as you scroll. Search is only available for the **Name** and **Host ID** columns. Sorting is only available on the **Hostname**, **Container**, **Metric %**, and **Health** fields.

Server Visibility Requirements and Supported Environments

To enable Server Visibility, review these system requirements guidelines.

System Requirements

Server Visibility requires a supported version of the Machine Agent to be installed on the monitored server. See Machine Agent Requirements and Supported Environments.

Server Visibility features are available for AIX, Linux, Windows, and Solaris. You need a Server Visibility license to enable and use Server Visibility features. Server Visibility is enabled by default on AppDynamics Controllers. You must explicitly enable Server Visibility on your Standalone Machine Agent to start sending the expanded set of Server Visibility metrics.

You can install and run Machine Agents on other supported platforms, however, Server Visibility features are not available. For OS platforms like AIX or HP-UX, AppDynamics recommends using the unbundled Machine Agent ZIP without the JRE.

Limitations

- Total number of Machine Agents and containers reporting to a single Controller is 30,000.
- Total number of App Agents reporting to a single Controller is 100,000.
- Monitored servers are considered stale 30 days after they go offline. They are purged from the Controller database when that time limit is reached.
- Docker Visibility is supported on Linux only.

Server Visibility on Windows

Review these notes:

- It is good practice to check the server frequently to ensure that it has the latest Windows updates installed. WMI (Windows Management Instrumentation) updates are especially important because the Agent uses WMI to collect Server Visibility metrics.
- The monitored server should have at least four cores or multiple CPUs. WMI processes can be highly resource-intensive, especially on Windows Server 2012.
- If you do not have a Server Visibility license, or to collect Basic metrics only, configure the Agent to use the `JavaHardwareMonitor` extension. Server Visibility should not be enabled on the Agent.
- The Windows Machine Agent uses these extensions by default:
  - `JavaHardwareMonitor` (for Basic metrics)
  - `ServerMonitoring` (for Server Visibility metrics)
- The `HardwareMonitor` extension is not recommended on Windows.
- To enable Server Visibility on a Windows server with a .NET APM Agent installed, you must enable .NET Compatibility Mode on both the Controller and the Standalone Machine Agent.

Enable Server Visibility

Before you enable Server Visibility:

1. Review the Server Visibility Requirements and Supported Environments.
2. Install and configure the Machine Agent on a machine with a supported OS as described in Install the Machine Agent.

> **Server Visibility License Required**
>
> Infrastructure Visibility is an add-on module to the Machine Agent and requires a Server Visibility license on the Agent.

Enable Server Visibility

1. Set the "`sim enabled`" property to "`true`". See Where to Specify Machine Agent Configuration, and configure the value based on your configuration choice:
   - Element in `controller-info.xml`: `<sim-enabled>`
   - System Property: `-Dappdynamics.sim.enabled`
   - Environment Variable: `APPDYNAMICS_SIM_ENABLED`
2. Restart the Machine Agent. See Start and Stop the Machine Agent.

Monitor Your Servers Using Server Visibility

This page describes the Server Visibility panels in the Controller UI:

Discover Normal Server Activity

Related pages:

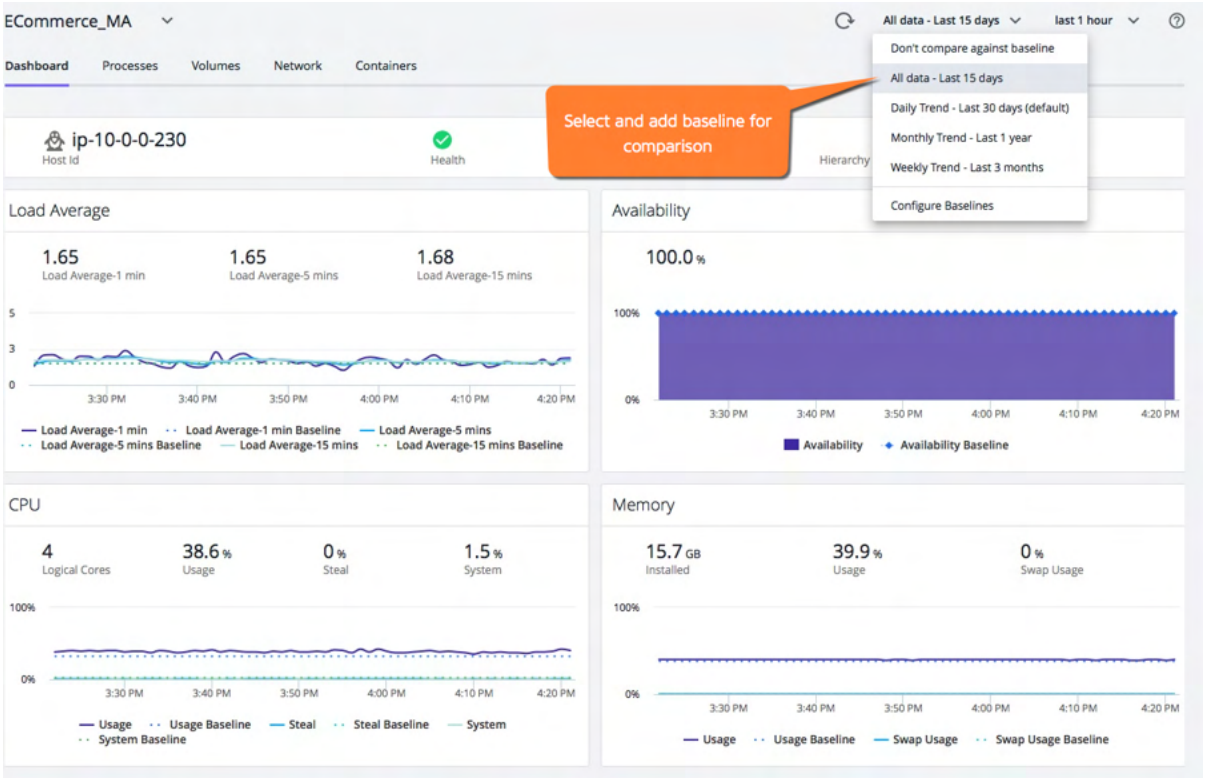
- Dynamic Baselines
- Hardware Resources Metrics

The Server Visibility module automatically learns to detect performance anomalies using baselines that are specific to your server environments.

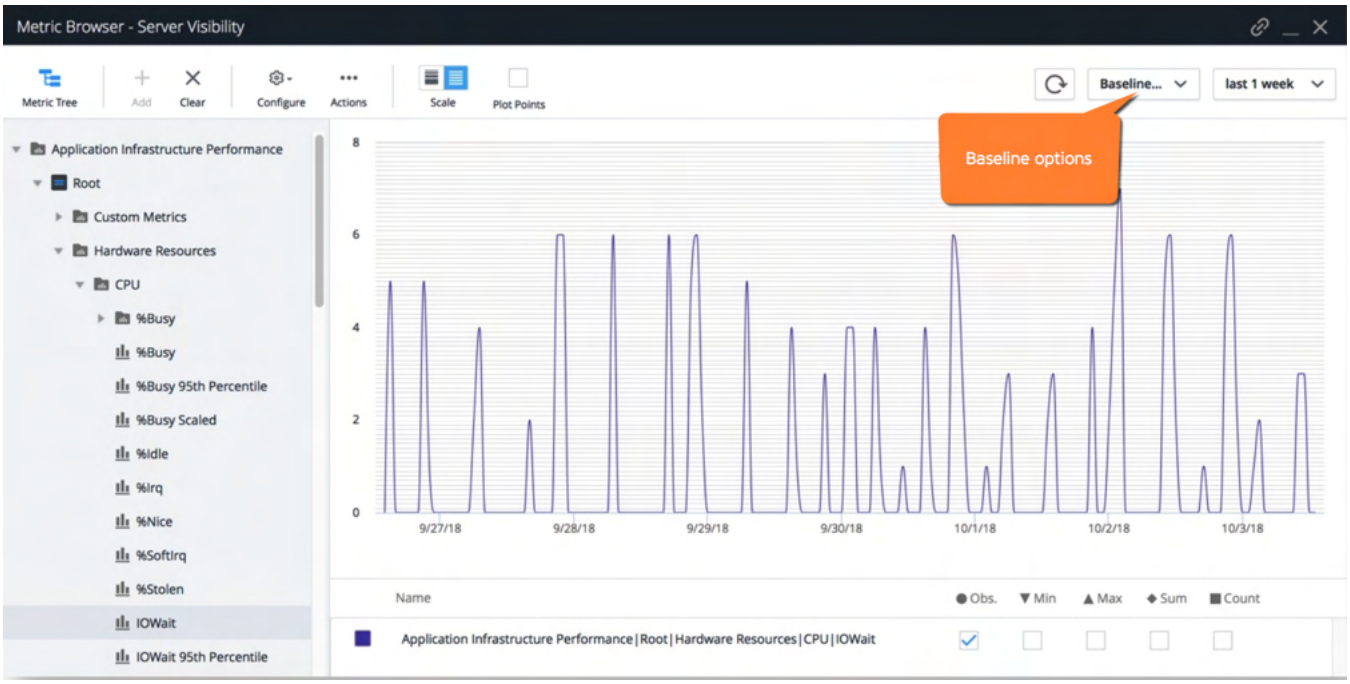
AppDynamics creates baselines by collecting metrics from your monitored servers over defined periods of time. This establishes what is normal for your environment so you can create health rules to alert you when metric deviations occur from the normal range. You can also create your own baselines.

View Performance Metrics Compared to Baselines

You can compare performance metrics to their dynamic baselines on the Server Dashboard.



You can visualize performance metrics and review deviations from expected behaviors established by the baseline from the **Metric Browser - Server Visibility** panel.


Metric Browser - Server Visibility
Metric Tree
Add
Clear
Configure
Actions
Scale
Plot Points
Baseline...
last 1 week
Baseline options
Application Infrastructure Performance
Root
Custom Metrics
Hardware Resources
CPU
%Busy
%Busy
%Busy 95th Percentile
%Busy Scaled
%Idle
%Irq
%Nice
%SoftIrq
%Stolen
IOWait
IOWait 95th Percentile
8
6
4
2
0
9/27/18
9/28/18
9/29/18
9/30/18
10/1/18
10/2/18
10/3/18
Name
Obs.
Min
Max
Sum
Count
Application Infrastructure Performance|Root|Hardware Resources|CPU|IOWait

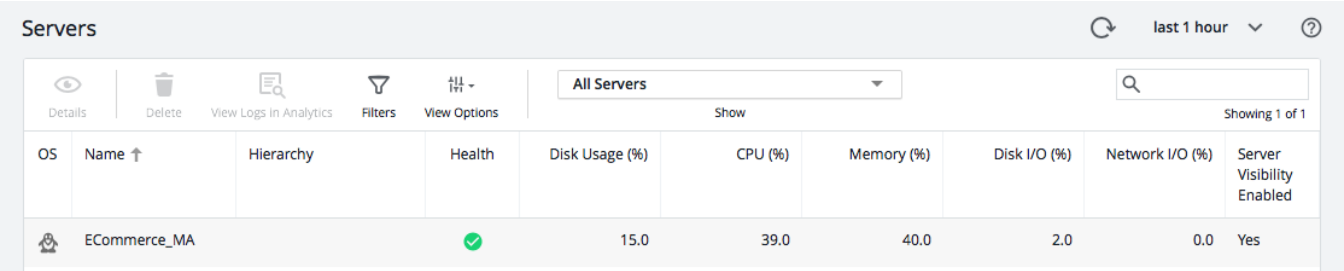
Servers List

To access: AppDynamics Home > **Servers**

The **Servers** list shows all machines registered by a Machine Agent. This list shows metric data for Server-Visibility-enabled machines only. Machines registered with App or .NET Agents, with no Machine Agent running, do not appear in this list.

On the Servers list you can:

- See the complete list of Machine Agents reporting to this Controller. The **Name** is taken from the value in `<machine-path>` in `controller-info.xml`. The last string after the last | (pipe) is used. For example, if the value is `<machine-path>Group | myName<machine-path>`, then the name is `myName`. For `<machine-path>JustName<machine-path>`, then the name is `JustName`.
- Filter the list using the dropdown to show all servers (the default) or just servers with Server Visibility enabled or disabled.
- Use **View Options** to select the columns shown for each server in the table. You can show or hide metrics such as **CPU Steal**, **Load Average**, **Memory Swap Used**, and **Network I/O**. You can also show or hide properties such as Host ID and JVM version.
- See key performance metrics for your monitored servers at a glance.
- Change the time period of the metrics displayed.
- See an overview of the health of the servers, as determined by whether any health rules have been violated. See View Health Rule Status in the UI.
- Click the violations icon ( ) to view health rule violations for a server.
- Click on any point on a spark chart to see the percentage usage at that time.
- Sort the chart by any of the columns.
- Right-click on a set of selected servers to
  - Delete the servers from the Servers list. See "Delete Selected Machine Agent" in Manage Machine Agents.
  - View and analyze log data for the servers (if you have an Application Analytics license). See Analytics.
  - Set the **Dynamic Monitoring Mode** on the servers. See Dynamic Monitoring Mode and Server Visibility.
- See additional details of servers that are enabled for Server Visibility. Double-click a selected server to see server-specific dashboard, and details of the server's volumes, network, and processes.
- Filter data according to the "liveness" of servers. A Machine Agent that reports metrics from a server is considered live. Eventually, servers can transition to a terminated or unrecoverable state. By specifying a time range in the Server Dashboard, you can filter servers which were live and reporting metrics during that time range. To enable this option for an individual account:
  1. Log in to the Controller admin page: `<controller-hostname>/controller/admin.jsp`
  2. From **Accounts**, open the account page of interest.
  3. Click **Controller Settings** > **Controller Configurations** and enter:
     a. Name = `entity.liveness.filter.enabled`
     b. value = `true`
     c. Name =`entity.liveness.tracker.enabled`
     d. value = `true`
- Disable automatic sorting of the Servers List table when it first loads. This option is useful if the Controller is monitoring hundreds or thousands of Machine Agents and the table is taking a long time to load. This option disables the ability to sort the table by any metric or metric trend column. To enable this option for an individual account, do the following:
  1. Log in to the Controller admin page: `<controller-hostname>/controller/admin.jsp`
  2. From **Accounts**, open the account page of interest.
  3. Click **+ Add Property** and enter:
     a. Property = `SIM_DISABLE_MACHINE_SORTING`
     b. value = `true`



Available Metrics in the Servers List

Metric Name	Description
Hierarchy	Specifies the root of the hierarchical path to the server. See the machine hierarchy property in this topic: Machine Agent Configuration Properties.
Health	Indicates if server health rules have been violated on this machine.
Disk Usage % trend	The percentage usage trend over time of storage space in use across all listed volumes, partitions, and disks.

CPU (%)	The percentage of time the CPU was busy processing system or user requests.
CPU Trend	The trend over time of CPU usage.
Memory (%)	The percentage of memory used.
Memory Trend	The trend over time of memory usage.
Disk I/O (%)	The percentage of time spent performing read and write operations across one or more disks, volumes, or partitions.
Disk IO Trend	The trend over time of disk usage.
Network I/O (%)	The average network utilization of network bandwidth for all monitored network devices. The network utilization percentage is the total incoming and outgoing bytes per second, divided by the maximum speed of the network interface. In some cases, if the Network I/O percentage value is too small compared to the network speed (10,000 Mb/s), the value is rounded to zero on the Servers List. In this case, the Network I/O % shows as zero, while the drilled-down details of the machine might show network traffic.
Network Trend	The trend over time of network usage.
Server Visibility Enabled	Indicates if Server Visibility is enabled for this machine. Values=Yes, No, and Not Supported. Not Supported indicates that Server Visibility is not available for that machine OS.
Icon	Representation of the machine's Operating System

Server Dashboard

To access Server Dashboard, select **AppDynamics Home > Servers > Dashboard**.

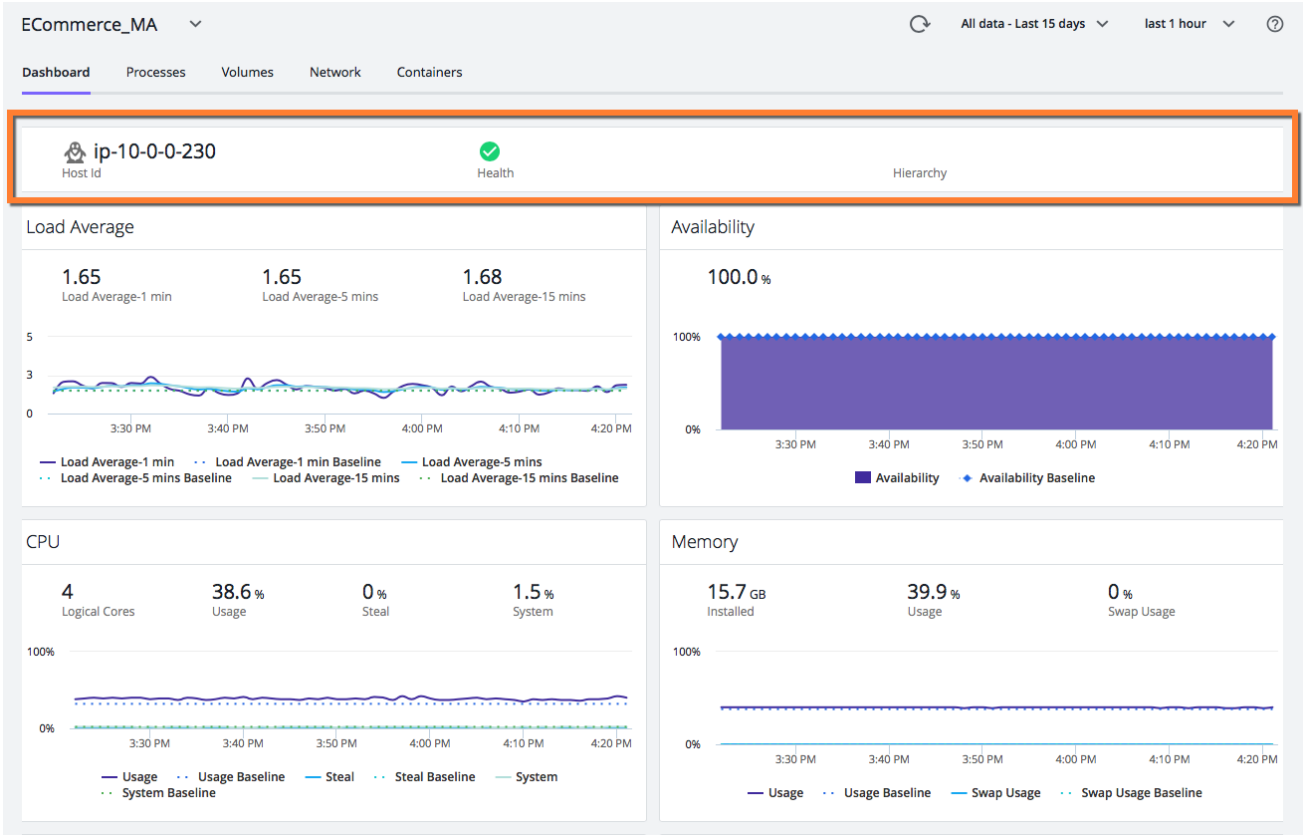
Select a server from the list and double-click a server name to display its Dashboard. In this Dashboard, you can:

- View charts of key performance metrics for the selected monitored servers:
  - Server availability
  - CPU, memory, and network usage percentages
  - Server properties
  - Disk, partition, and volume metrics
  - Top ten processes consuming CPU resources and memory
- Change the time period of the metrics displayed.
- View an assessment of the overall health of the server based on any health rule violations. See View Health Rule Status in the UI.
- View the hierarchy or grouping of the server as specified in the `controller-info.xml` using the machine-path configuration property.
- Click on any point on a chart to see the metric value for that time.
- Sort the chart using any of the columns as a sorting key.
- Find and switch to other Server Dashboards from the dropdown next to server tier.
- View an aggregate of the top ten processes by CPU usage, and top ten processes by memory.

Server Dashboard Metrics

Host ID, Health, Hierarchy (*Top Pane*)

- **Host Id**: This is an ID for the server that is unique to the AppDynamics Controller.
- **Health**: Shows the overall health of the server. Hover over the health icon for details; for example, the following appears when hovering over the exclamation mark (!) in a red box, "There are Critical Health Rule Violations on this machine during the time range."
- **Hierarchy**: This is an arbitrary hierarchy to group your servers together, specified by Machine Hierarchy Property.



Load Average

The CPU load presented as an average over the past 1 minute, 5 minutes, and 15 minutes. The CPU load is the percentage of the CPU consumed by processes that are currently running or that are waiting to run. These metrics are shown as percentages in the Server Dashboard and are scaled by 100 in the Metric Browser.

Availability

- **Availability**: Percentage of time the Machine Agent is able to communicate with the Controller. The Machine Agent sends availability information in the form of a heartbeat to the AppDynamics Controller once a minute to indicate the Agent is alive. The heartbeat may show as available when other functions of the Machine Agent and Server fail. The availability only indicates that the Machine Agent is communicating with the Controller, and does not indicate the overall health of the functions within the Machine Agent.
- **time series chart**: Shows the Machine Agent availability trend over time. There are situations where the Machine Agent is able to communicate with the Controller while other functions of the Machine Agent, or processes on the server, are non-functional. Do not use the time series metric to indicate the overall availability of the server.

CPU

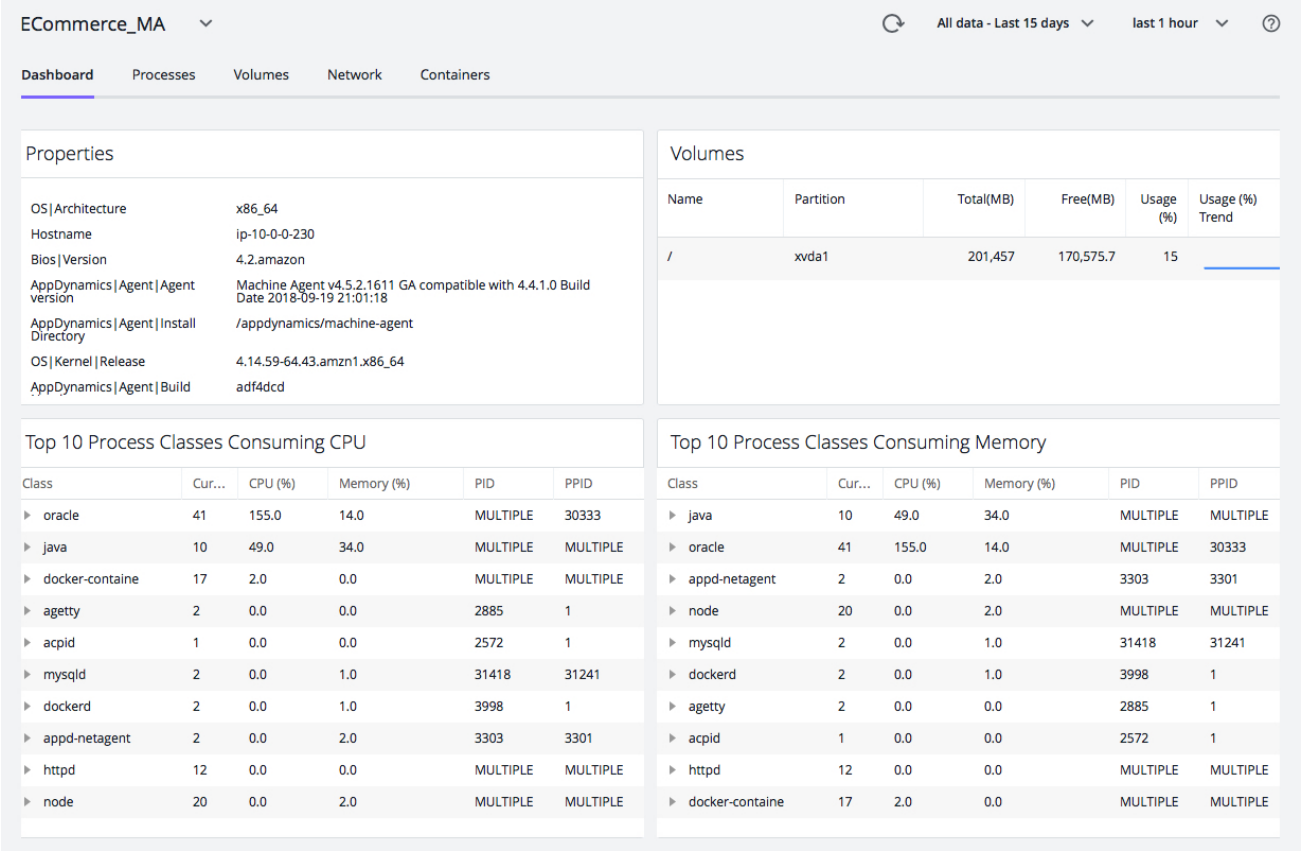
- **CPU Usage**: Average usage of CPU over the chosen time-range.
- **# of Cores**: Number of cores for the CPU.
- **time series chart**: Shows CPU busy percentage trend over time.

Memory

- **Installed**: Total amount of memory, free and used, on the server.
- **Memory Usage**: Percentage of memory used.
- **Swap Usage**: Swap usage tracks the swap file utilization. A swap file enables an operating system to use hard disk space to simulate extra memory. When the system runs low on memory, it swaps a section of RAM being used by an idle program onto the hard disk to free up memory for other executing programs. When the idle, swapped out program begins executing again, it is moved back to RAM, potentially displacing some other idle program. This causes a large amount of hard disk reading and writing that can slow down the system considerably.
- **time series chart**: Shows the memory usage trend over time.

Network

- **Interfaces**: Number of network interfaces on the server.
- **Outgoing**: Kilobytes of data sent per second for all monitored network devices.
- **Incoming**: Kilobytes of data received per second for all monitored network devices.
- **time series chart**: Shows the network incoming and outgoing volume trend over time.



Volumes

- **Total**: Storage space, free and used, on the disk, partition or volume. For Linux systems, this does not include disk space reserved by the kernel.
- **Free**: Total storage space available. For Linux systems, this does not include disk space reserved by the kernel.
- **Usage(%)**: Percentage of storage space in use across each disk, partition, and volume.
- **time series chart**: Shows the storage usage trend over time.

Top 10 Processes Consuming CPU

- **Count**: Number of processes in this class.
- **CPU (%)**: Percentage of CPU resources consumed by all processes in this class.
- **Memory (%)**: Percentage of memory consumed by all processes in the class.
- **PID**: Process ID.
- **PPID**: Parent Process ID.

Top 10 Processes Consuming Memory

- **Count**: The number of processes in this class consuming memory resources.
- **CPU (%)**: Percentage of CPU consumed by all processes in this class.
- **Memory (%)**: Percentage of memory consumed by all processes in this class.
- **PID**: Process ID.
- **PPID**: Parent Process ID.

Server Process Metrics

Related pages:

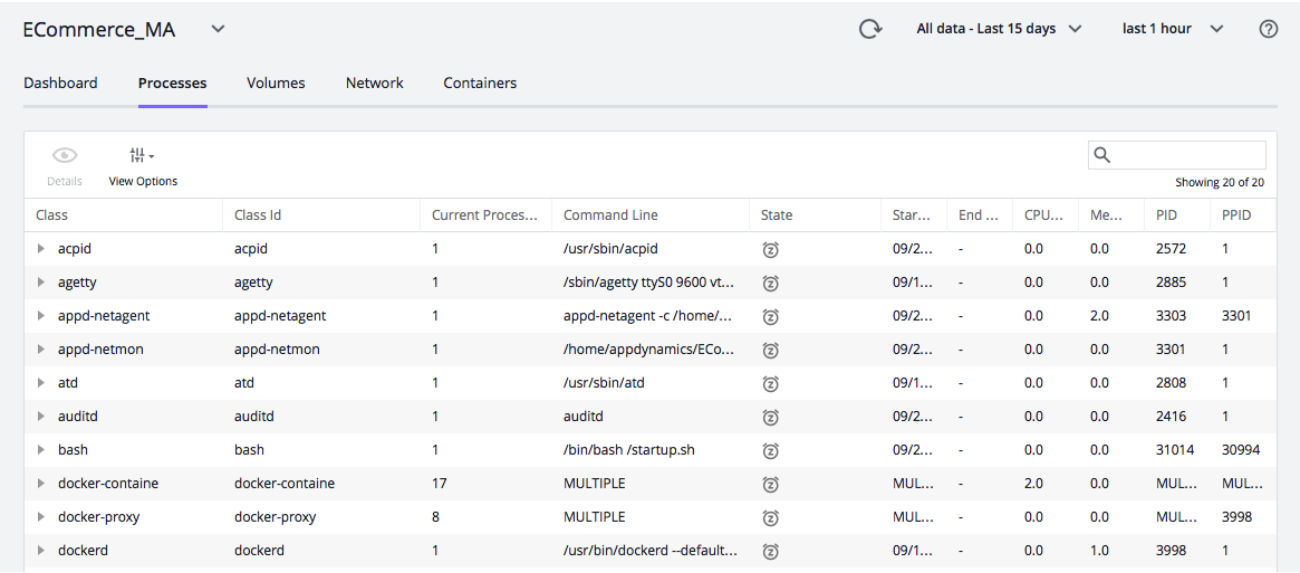
- Hardware Resources Metrics
- Enable Server Visibility
- Machine Agent Settings for Server Visibility

To access Server Process Metrics, select **AppDynamics Home** > **Servers** > (double-click a server) > **Processes**.

You can configure which processes AppDynamics monitors, how they are grouped, the number of processes to monitor, and how long they must be alive before monitoring them. See Machine Agent Settings for Server Visibility

From the Server Processes tab, you can:

- View all the processes active during the selected time period. The processes are grouped by class as specified in the `ServerMonitoring.yml` file.
- View the full command line that began this process by hovering over the process entry in the **Command Line** column.
- Expand a process class to see the processes associated with that class.
- Select **View Options** to configure which columns to display in the chart.
- Change the time period of the metrics displayed.
- Sort the chart using the columns as a sorting key. You cannot sort by CPU Trend or Memory Trend on sparkline charts.
- Review CPU and Memory usage trends.



Class	Class Id	Current Proces...	Command Line	State	Star...	End ...	CPU...	Me...	PID	PPID
acpid	acpid	1	/usr/sbin/acpid		09/2...	-	0.0	0.0	2572	1
agetty	agetty	1	/sbin/agetty ttyS0 9600 vt...		09/1...	-	0.0	0.0	2885	1
appd-netagent	appd-netagent	1	appd-netagent -c /home/...		09/2...	-	0.0	2.0	3303	3301
appd-netmon	appd-netmon	1	/home/appdynamics/ECo...		09/2...	-	0.0	0.0	3301	1
atd	atd	1	/usr/sbin/atd		09/1...	-	0.0	0.0	2808	1
auditd	auditd	1	auditd		09/2...	-	0.0	0.0	2416	1
bash	bash	1	/bin/bash /startup.sh		09/2...	-	0.0	0.0	31014	30994
docker-containe	docker-containe	17	MULTIPLE		MUL...	-	2.0	0.0	MUL...	MUL...
docker-proxy	docker-proxy	8	MULTIPLE		MUL...	-	0.0	0.0	MUL...	3998
dockerd	dockerd	1	/usr/bin/dockerd --default...		09/1...	-	0.0	1.0	3998	1

Metrics for Server Processes

The following information displays for each monitored class and process:

Default Columns

- **Class**: The process class.
- **Count**: The number of processes in this class.
- **Command Line**: The command that began the process.
- **State**: An icon represents the process state: sleeping, running, terminated, zombie or multiple. Hover over the icon to discover its state. The State column displays a question mark when there is more than one process associated with the class. Expand the class to review the state of the related processes.
- **Effective User**: The name of the user account that started the process.
- **Start Time**: The time, as set on the Controller machine, when the process started.
- **End Time**: The time, as set on the Controller machine, when the process ended.
- **CPU (%)**: The percentage of CPU resources by all processes in this class.
- **CPU Trend**: A chart that shows CPU usage over the selected time period.
- **Memory (%)**: The percentage of memory resources by all processes in this class.
- **Memory Trend**: A chart that shows memory usage over the selected time period.
- **PID**: Process ID.
- **PPID**: ID of the parent process.

OS-Specific Columns

The column information is specific to processes monitored on a Linux server.

- **pgid**: Process group ID.
- **Real Group**: The process real group ID.
- **Real User**: The process real user ID.
- **Effective Group**: The user ID the kernel uses to determine the process permissions when using shared resources such as: message queues, shared memory, and semaphores.
- **Nice Level**: The priority used to indicate the amount of CPU to afford the process or the process priority, where -20 is the highest priority, and 19 or 20, is the lowest priority.

Windows Process Collection Configuration

When too many processes slow down the performance of the Machine Agent on Windows, you must adjust the system environment variables to accommodate. Specifically, in relation to Windows-based operating systems supported by the AppDynamics platform. See System Requirements.

Microsoft Windows process collection is enabled by default in the Controller and is missing the Effective User value due to performance issues. To retrieve the Effective User value, set the `APPDYNAMICS_ENABLE_PROCESS_OWNER_INFORMATION` environment variable. Although you can retrieve this information, this will most likely slow your process collection and may lead to additional issues, such as high CPU usage and the failure to report metrics. To disable process collection, set the `APPDYNAMICS_MACHINE_AGENT_STOP_PROCESS_COLLECTION` to speed up your performance at the expense of losing process metrics and metadata.

Windows System Environment Variables

This information is specific to processes monitored on a Windows server.

1. **Default:** Retrieve all Windows processes and metadata. Starting with Machine Agent 4.5.4, Effective User information is excluded by default to improve performance.
2. `APPDYNAMICS_ENABLE_PROCESS_OWNER_INFORMATION`: Retrieve all Windows processes and metadata, including the Effective User, but the Machine Agent may experience performance issues. This is available starting with Machine Agent 4.5.4.
3. `APPDYNAMICS_MACHINE_AGENT_STOP_PROCESS_COLLECTION`: Stops all Windows process metrics and metadata collection, but improves Machine Agent performance. If you set this variable, then the `APPDYNAMICS_ENABLE_PROCESS_OWNER_INFORMATION` variable is ignored. This is available starting with Machine Agent 4.4.0.

These environment variables only need to be set; you do not need to assign a value to the variable.

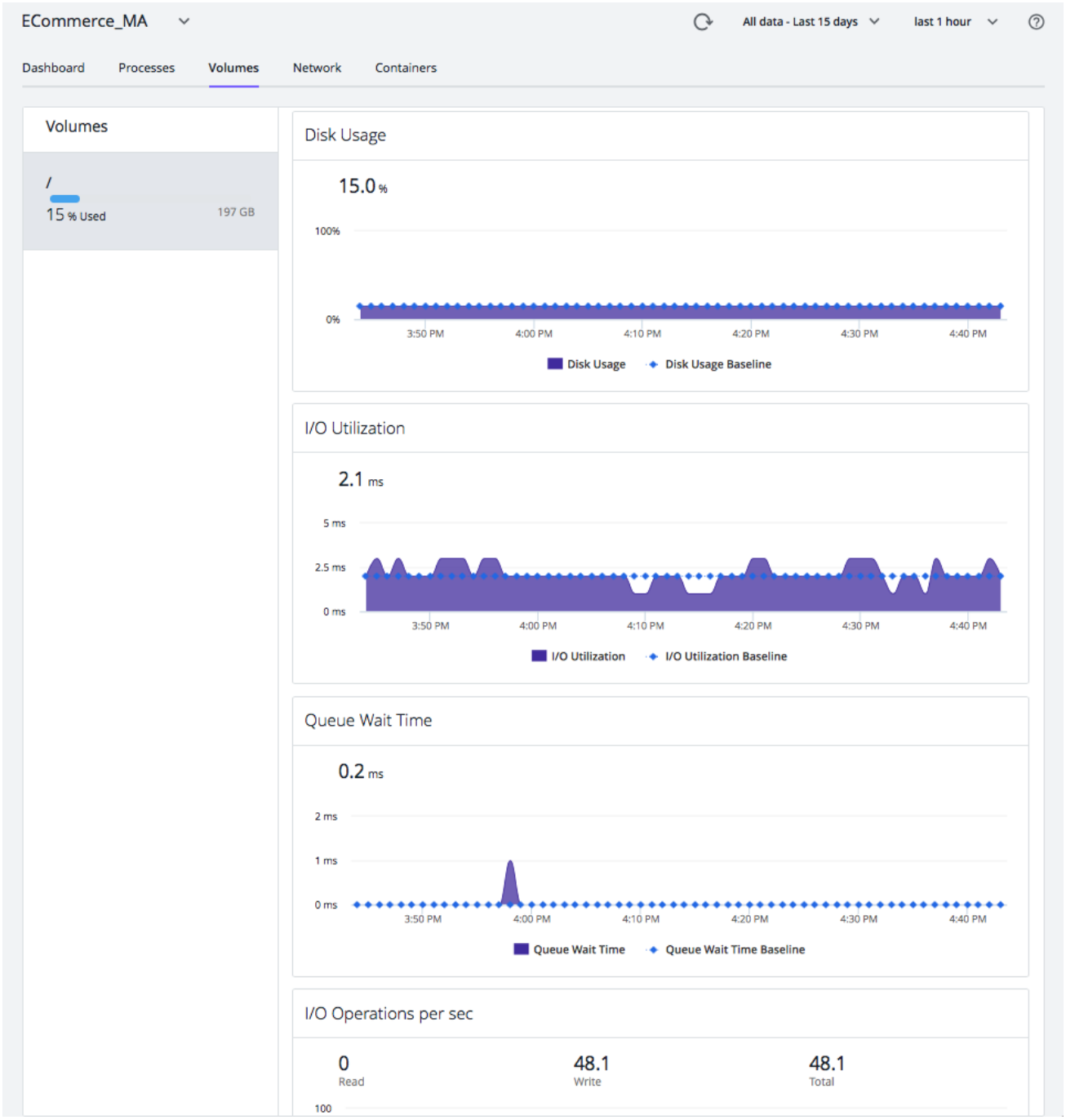
You must restart the Machine Agent for these changes to take effect.

Server Volumes Metrics

To access Server Volumes Metrics, select **AppDynamics Home > Servers >** double-click server > **Volumes**.

From the Server Volumes tab, you can:

- View the list of volumes, the percentage used, and total storage space available on the disk, partition, or volume
- View disk usage and I/O utilization, rate, operations per second, and wait time
- Change the time period of the metrics collected and displayed
- Click on any point on a chart to review the metric value for that time



Metrics in the Server Volumes Tab

For each selected disk, partition or volume, these columns are available for the **Server Volumes** panel.

- **Disk Usage %**: The percentage of storage space in use (To configure how this metric is calculated on Linux, see Configure Disk Usage Metric Collection on Linux).
- **I/O Utilization %**: The percentage of time spent performing read and write operations
- **I/O Rate read**: The number of kilobytes per second of data reads
- **I/O Rate write**: The number of kilobytes per second of data writes
- **I/O Operations per sec read**: The number of read operations per second
- **I/O Operations per sec write**: The number of write operations per second
- **I/O Wait Time read**: The percentage of time the CPU was waiting for read operations to complete
- **I/O Wait Time write**: The percentage of time the CPU was waiting for write operations to complete

- Volume space metrics on POSIX systems are obtained using the "`df`" command.
- On Linux, the space reserved for `root` is not counted in the available space.

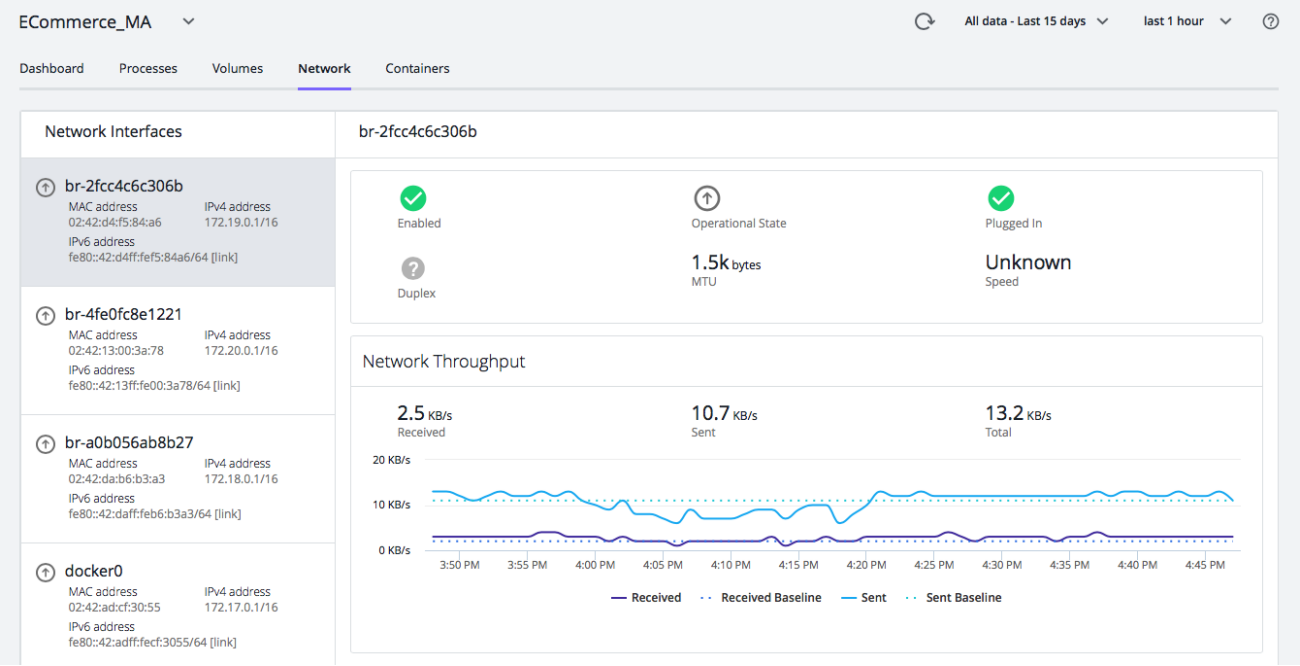
Server Network Metrics

To access Server Network Metrics, select **AppDynamics Home > Servers >** double-click server > **Network**.

From the Server Network tab, you can:

- View the MAC, IPv4, and IPv6 IP address for each network interface.
- View the state of the network interface:
    - Enabled
    - Functional (its operational state)
    - Equipped with an Ethernet cable that is plugged in
    - Operating in full or half-full duplex mode
- View the maximum transmission unit (MTU) or size (in bytes) of the largest protocol data unit that the network interface can pass
- View the speed of the Ethernet connection in Mbit/sec
- View network throughput in kilobytes/sec and packet traffic
- Change the time period of the metrics displayed
- Hover over any point on a chart to review the metric value for that time

If no metrics display, the network device speed cannot be found for some devices and some Linux versions.



Server Network Metrics

These columns are available for the Server Network panel.

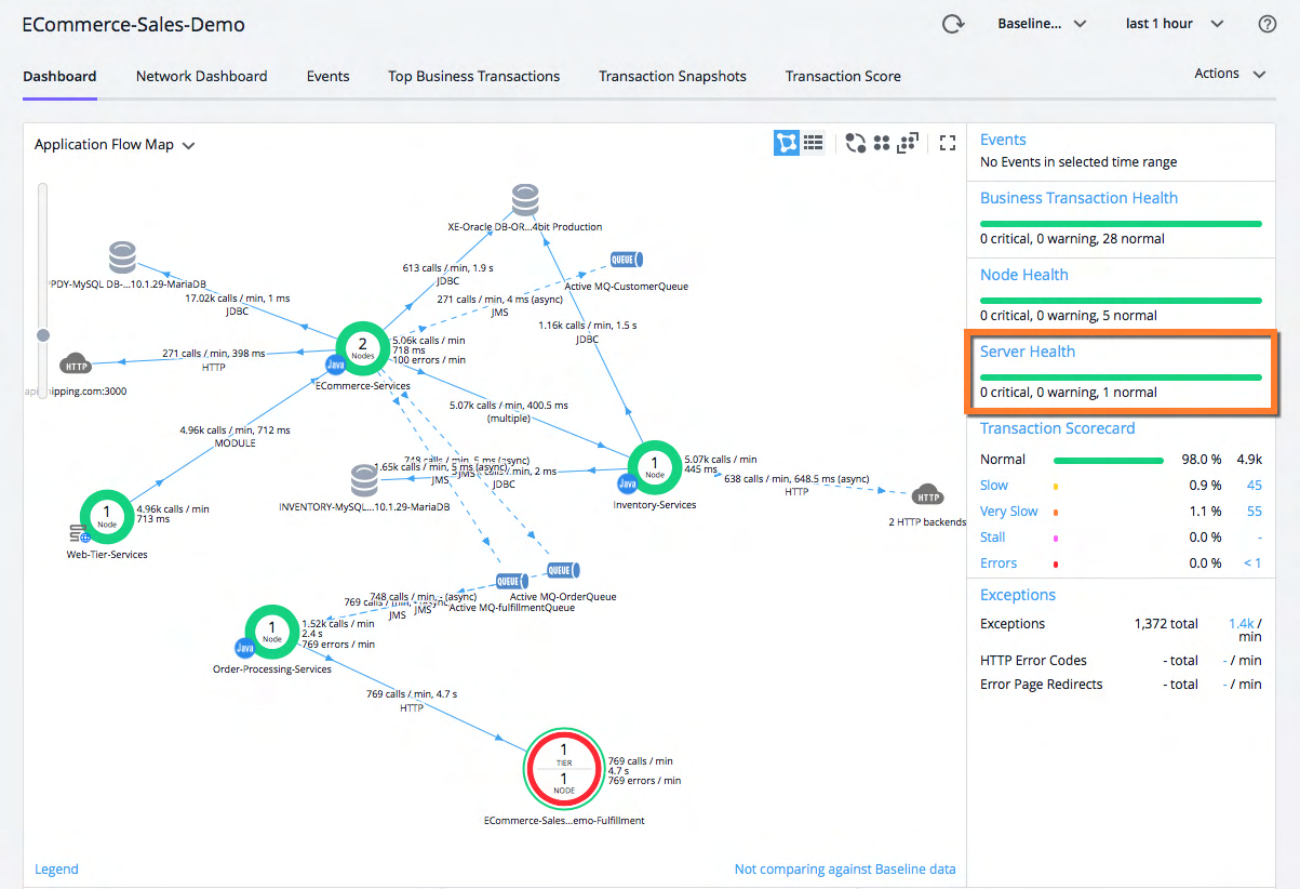
For the selected network interface:

- **Network Throughput received**: The volume of data received per second
- **Network Throughput sent**: The volume of data sent per second
- **Packets per sec incoming**: The number of data packets received per second
- **Packets per sec outgoing**: The number of data packets sent per second

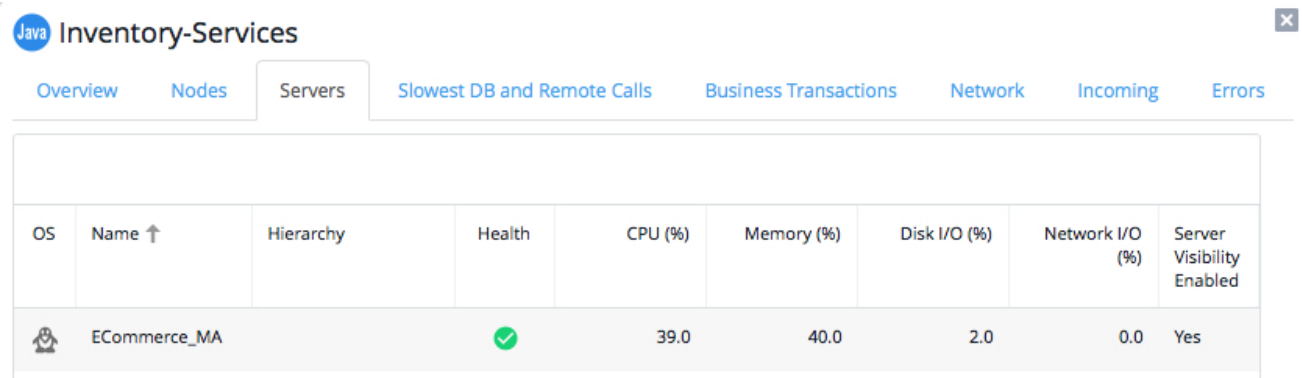
Navigating Between Server and Application Contexts

With Server Visibility enabled, you can access server performance metrics in the context of your applications. See Monitor Your Servers Using Server Visibility.

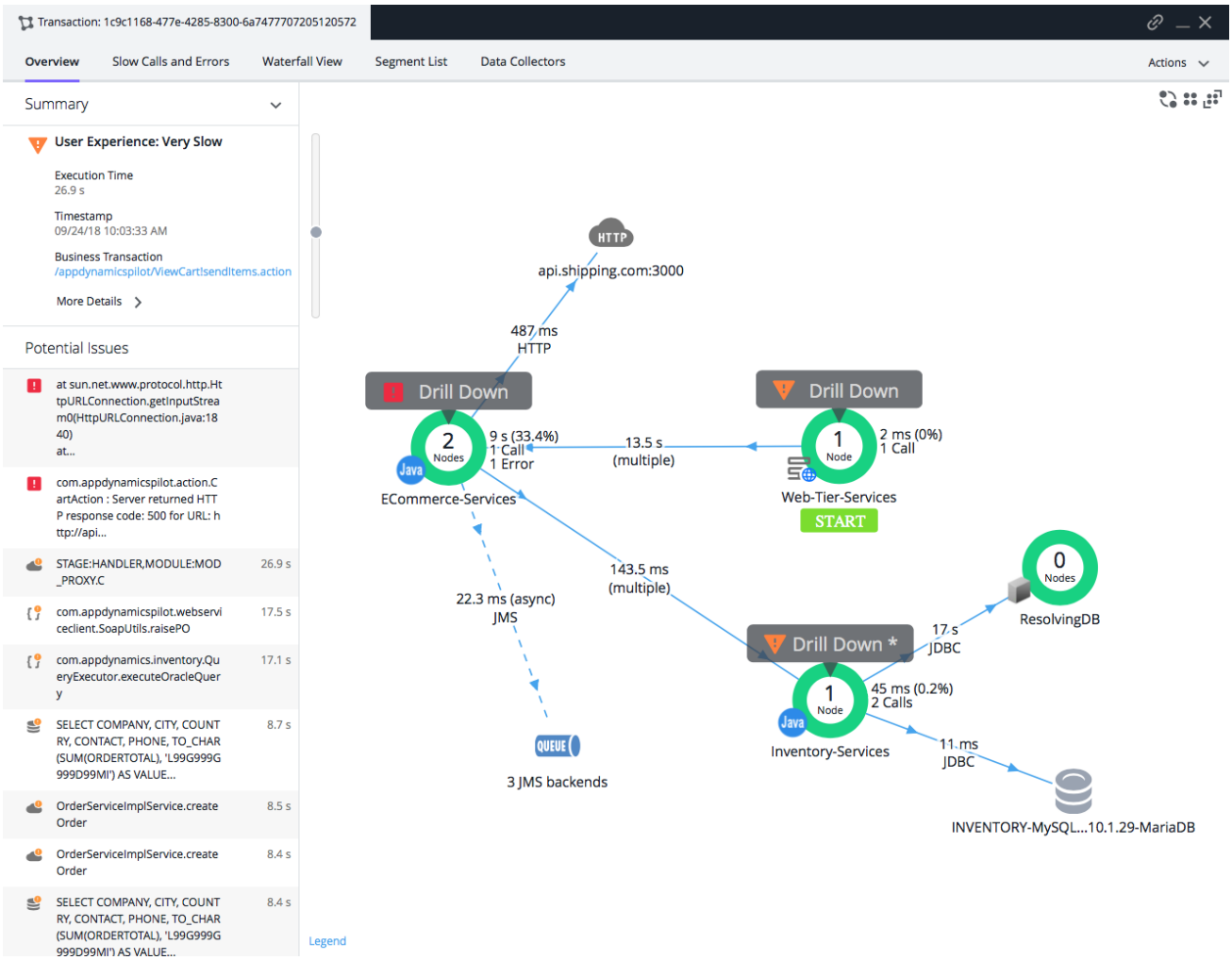
After logging in to the AppDynamics Controller, and locating your application flow maps, you can drill down to Server metrics for a selected tier or node. The Server health summary shows the number of critical/warning/normal conditions for the application's servers.



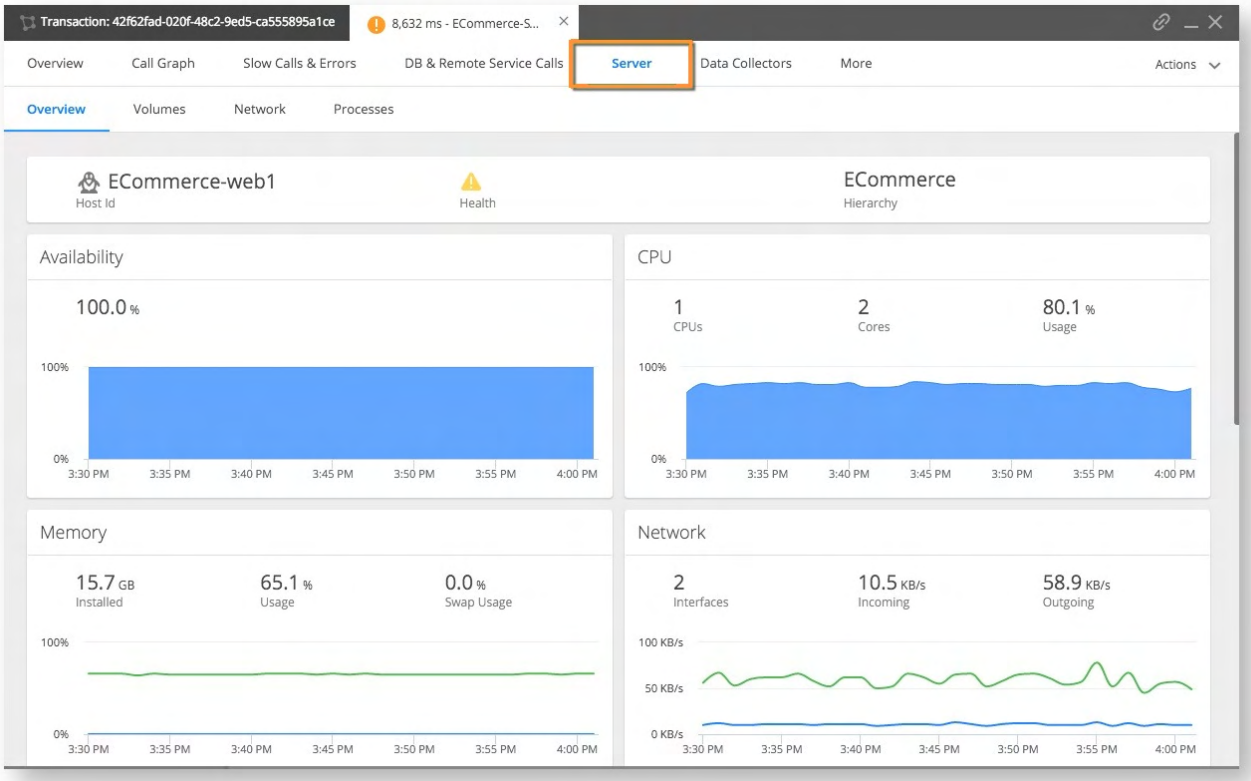
Select a tier, such as in this example, *Inventory-Services*, to review the server details.



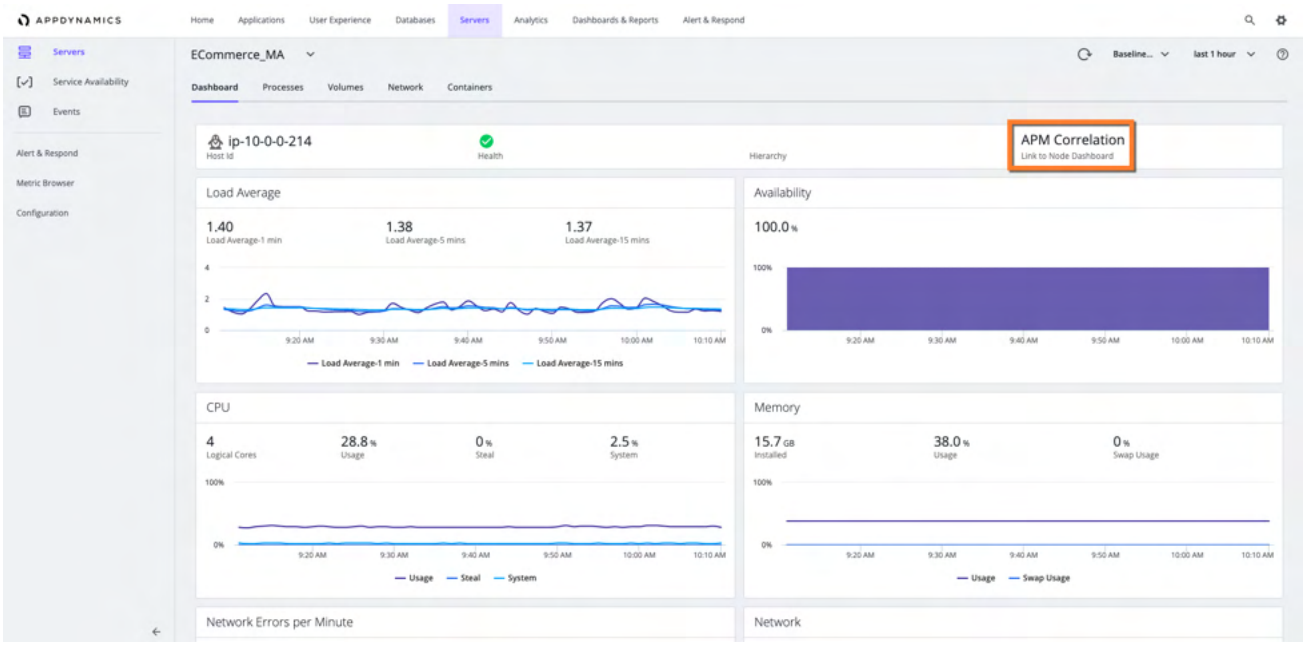
From the list of transaction snapshots, you can drill down to server metrics to determine if there is anything suspicious contributing to slow transactions. Double-click a snapshot of interest, drill down into the call to view the Server tab and its associated metrics.



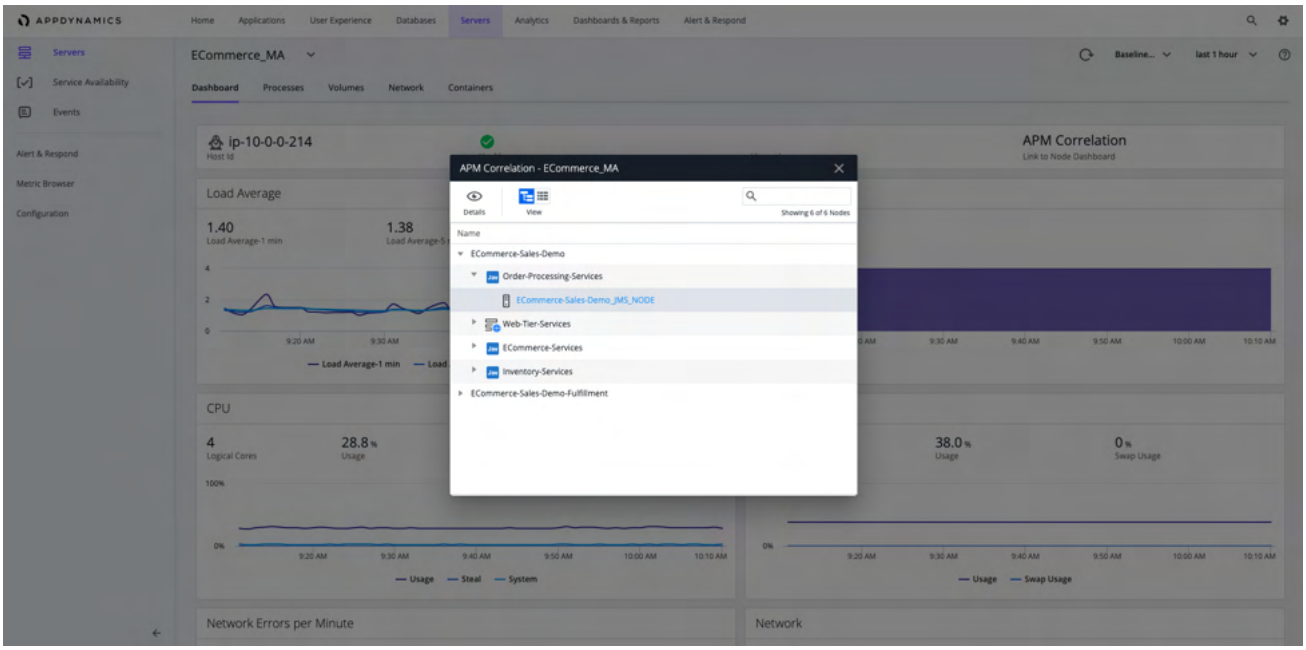
From the snapshot Server tab, you can review **CPU**, **Memory**, and **Network** utilization, and determine which processes are consuming server resources.



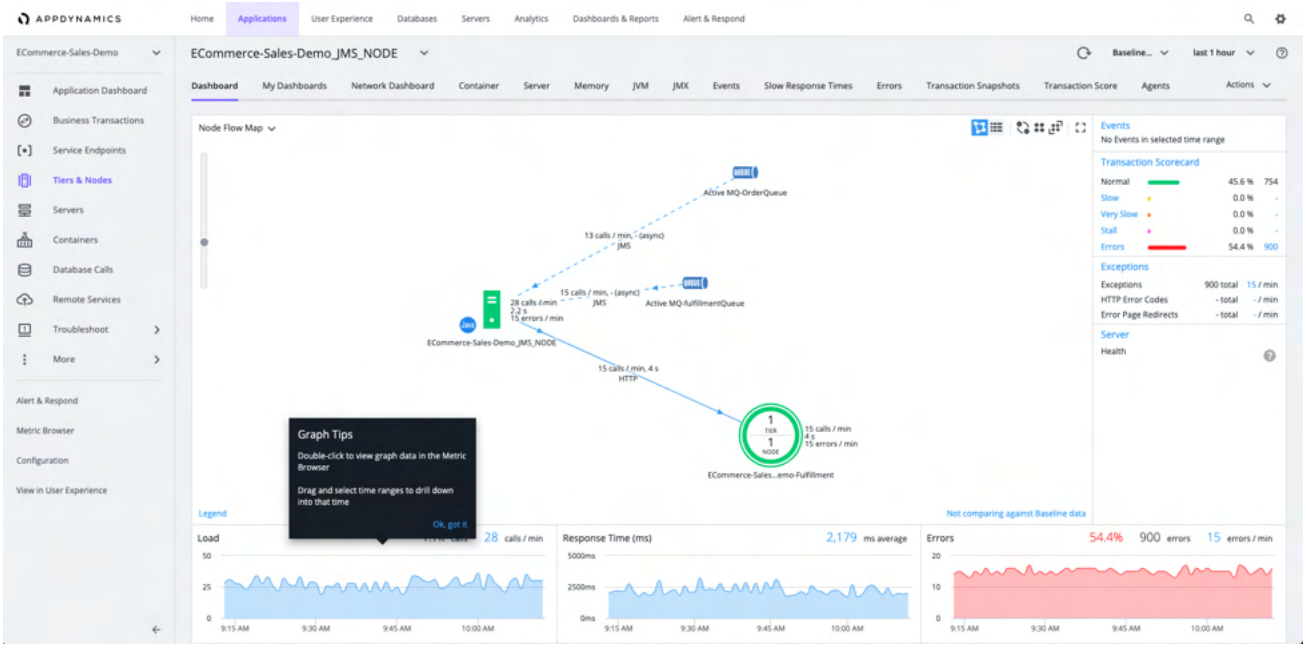
From the Servers Dashboard tab, select **APM Correlation**.



Select the **APM Correlation** link to review the APM node associated with the server. If there is more than one APM node running on the server, a dialog displays enabling you to select the APM node you want.



By selecting the APM node, its associated **Dashboard** displays.



Dynamic Monitoring Mode and Server Visibility

You can selectively control the number of metrics reported by individual Machine Agents (including Server Visibility Agents). Rather than have all Agents report all the metrics all the time, you can configure individual Agents to collect metrics.

- Key Performance Indicator metrics only (*KPI mode*)
- KPI and Diagnostic metrics (*Diagnostic mode*)
- All available metrics (*Advanced Diagnostic mode*)

This provides the flexibility to report KPI metrics on most machines, and then increase the metric level on specific servers for deeper visibility to diagnose problems. You can increase scalability on the Controller and conserve metric bandwidth on the network with no sacrifice in visibility.

Every Basic and Server Visibility metric has a default Dynamic Monitoring Mode (DMM) class: KPI, Diagnostic, and Advanced Diagnostic. See Hardware Resources Metrics.

Important Notes

- You can disable DMM on individual Agents. When DMM is disabled on an Agent, that Agent will report all metrics regardless of whether DMM is enabled or disabled on the Controller. Disabling DMM on an Agent is recommended only for mission-critical servers and other machines for which you are sure you want to collect all metrics. See Machine Agent Configuration Properties.
- If you switch the monitoring mode in the Controller from a more-inclusive to a less-inclusive mode, the Metric Browser will show values for the newly-excluded metrics for one hour after the switch. For example, if you switch from Diagnostic to KPI mode, for any Diagnostic metric, the Metric Browser will report a steady line (at 0 or the last-reported value) for one hour after the switch; then the line will disappear. This is standard behavior in the Metric Browser for an Agent when it stops reporting a specific metric.
- Each Machine Agent has a set of configurable settings that specify the volumes, networks, and processes to report and to ignore. For example, you can define a process allowlist (report on matching processes) and blocklist (ignore matching processes). An Agent will not report metrics for items excluded by its local settings, regardless of the monitoring mode. See Machine Agent Settings for Server Visibility.
- If you have any custom health rules based on Diagnostic or Advanced Diagnostic metrics, DMM might cause these rules to generate "false-positive" alerts. (Standard, non-custom health rules are not affected.) If you have any health rules similar to this, edit the rule to use a KPI metric instead of the Diagnostic or Advanced Diagnostic metric.

Initial Setup

DMM is enabled by default. To configure DMM:

1. Log in to the Controller administration console using the `root` user password. See Access the Administration Console.

   ```
   http://<controller host>:<port>/controller/admin.jsp
   ```

2. Set these options:
   a. `sim.machines.dmm.defaultMode` = `KPI` (Sets the default mode to KPI on all Machine Agents)
   b. `sim.machines.dmm.dmmAllowed` = `true` (Enables Dynamic Monitoring Mode on the Controller)
3. For servers that require additional visibility, increase the monitoring mode to **Diagnostic** or **Advanced** Diagnostic. To change the monitoring mode of one or more agents:
   a. Log in to the Controller UI.
   b. Click **Servers**.
   c. In the **Servers** table, check the box next to one or more servers whose monitoring mode you want to change.
   d. Right-click the server and select **Set DMM Mode**. A dialog box appears.
   e. Select the desired monitoring mode.
   f. Click **Save**.

Workflow Example

After you complete the initial setup, you can set the Dynamic Monitoring Mode on individual Machine Agents as required. The following is a workflow example:

- The DevOps team for a large enterprise monitors its IT infrastructure using Machine Agents (1,000-plus Agents monitoring servers in hundreds of locations). All Agents are initially set to KPI mode.
- One Agent reports a lot of disk read/write operations (KPI metric) on critical-server-A.
- Set the Agent DMM on critical-server-A to **Diagnostic**.
- Monitor the amount of data read and written for the entire disk, and for each partition (Diagnostic metrics), on critical-server-A.
- If the Diagnostic metrics do not indicate the source of the problem, and further investigation is needed, set the agent DMM to **Advanced Diagnostic**.
- Monitor average queue times and read/write times for each partition (Advanced Diagnostic metrics) on critical-server-A.
- When advanced diagnostics are no longer required on critical-server-A, set the agent DMM back to **KPI**.

Tier Metric Correlator

The Tier Metric Correlator enables you to identify load and performance anomalies across all nodes in a tier. Suppose you have a tier composed of a cluster of nodes running on containers or servers. You expect all the nodes to behave exactly the same under the same load conditions. How can you monitor this cluster for anomalies and outliers? You can use the Tier Metric Correlator to answer the following questions:

- Are all the nodes behaving within the expected band of performance, or do some nodes have outliers (slow calls, stalls, and errors)?
- In what time windows, and on which nodes, are these outliers occurring?
- Are outliers associated with a specific node cluster—for example, a cluster running a canary release?
- Are any resource issues (such as high CPU I/O or paging) correlated with these outliers?
- Are calls getting distributed evenly across all nodes in the tier?

This page describes two example use cases:

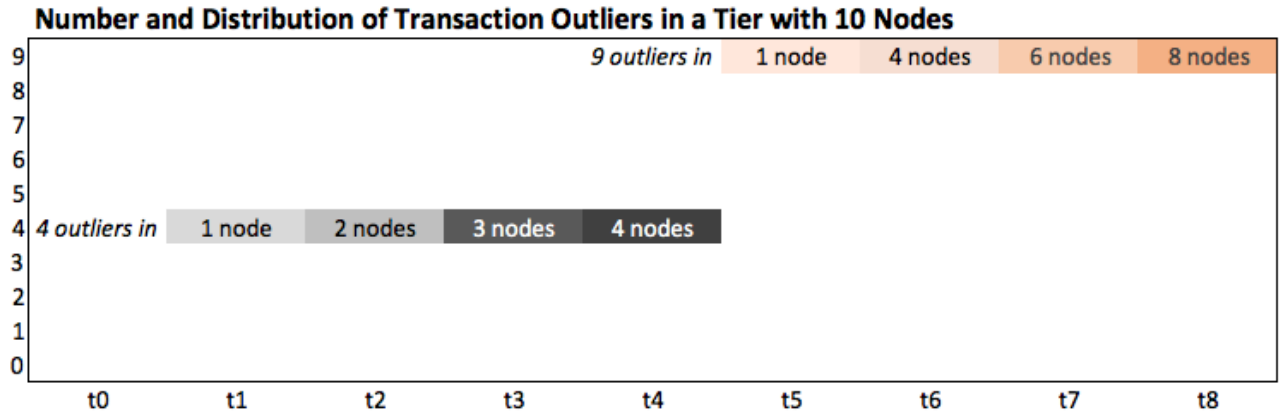
- Use Tier Metric Correlation to quickly identify balances and imbalances in the distribution of calls across all nodes in a tier.
- Use Tier Metric Correlation to monitor and troubleshoot a canary deployment scenario. This example shows how you can easily compare performance across node clusters, identify nodes with transaction outliers, and determine if any resource issues are causing these outliers.

Key Concepts

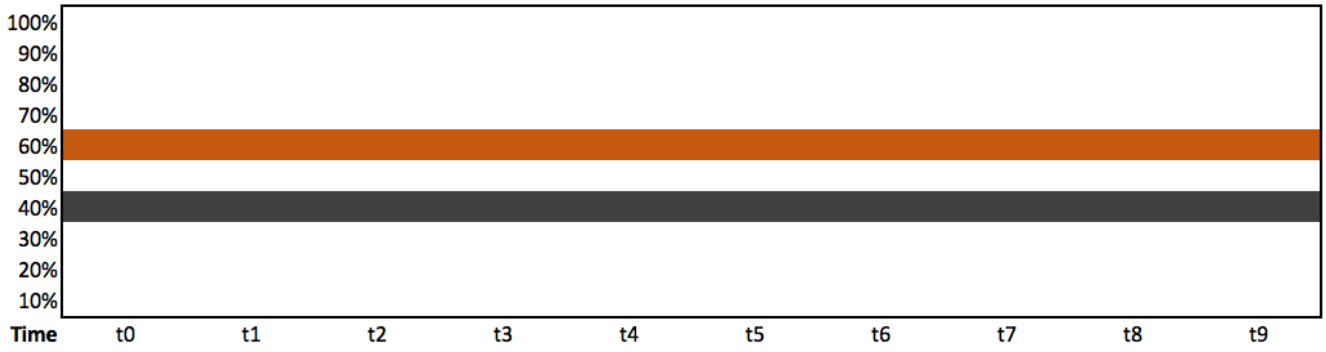
Transaction Outliers

The first step in metric correlation is to determine the *transaction outliers* for a tier: the number of Slow Calls, Very Slow Calls, and Stalled Calls whose response times are significantly outside the norm for that tier. The Transaction Outliers Heatmap visualizes the rate of these outliers and their distribution across all nodes in the tier.

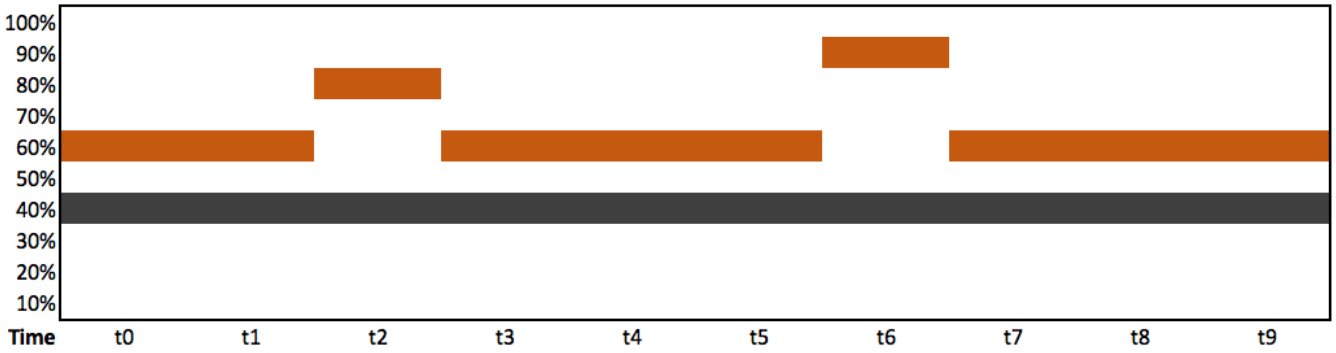
A heatmap is a time-series chart with an extra dimension: the color intensity of each bar shows the distribution of outliers across all nodes. The darker the hue, the more nodes have outliers. The chart colors the bars in shades of gray (fewer outliers) and orange (more outliers).



Heatmaps enable you to easily identify the normal bands of performance and any outliers for all nodes in a tier. In this example, all nodes fall within two bands:



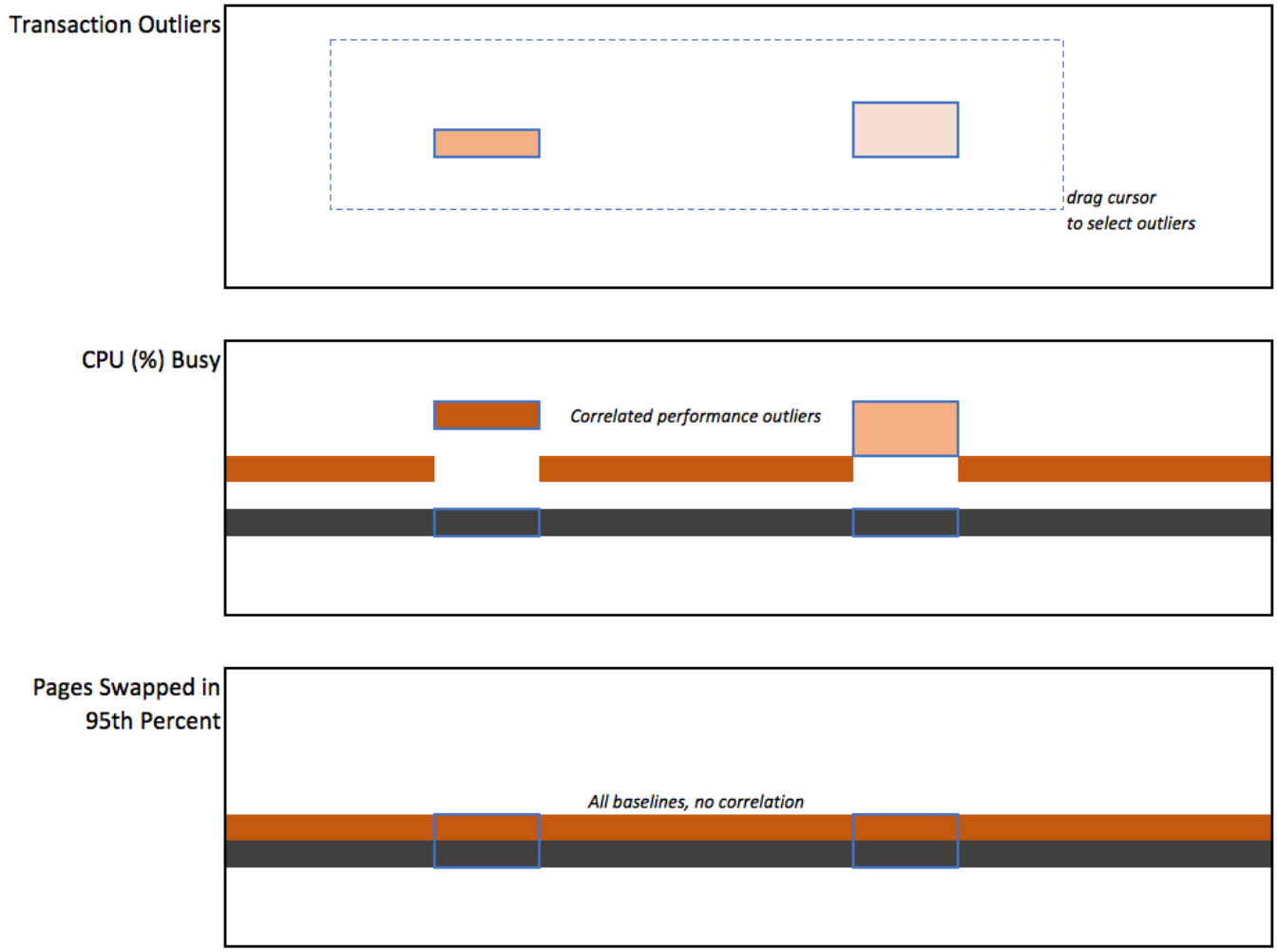
This heatmap highlights two time windows where the performance metric is noticeably higher for 50% of the nodes:



Correlated Metrics

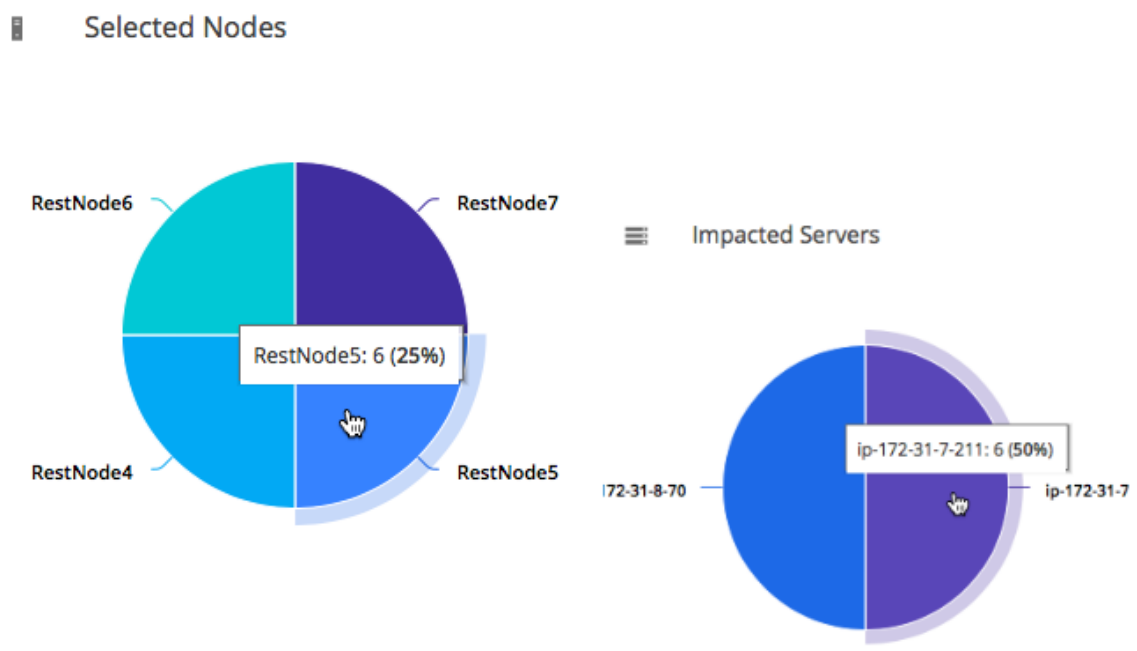
The Correlated Metrics heatmaps enable you to correlate transaction outliers with specific resource metrics. The Correlate Metrics panel enables you to easily identify performance metrics that correlate (and do not correlate) with transaction outliers of interest. This example shows three heatmaps:

1. Transaction outliers
2. Correlated metric
3. Uncorrelated metric (no outliers)



Troubleshooting Nodes and Servers

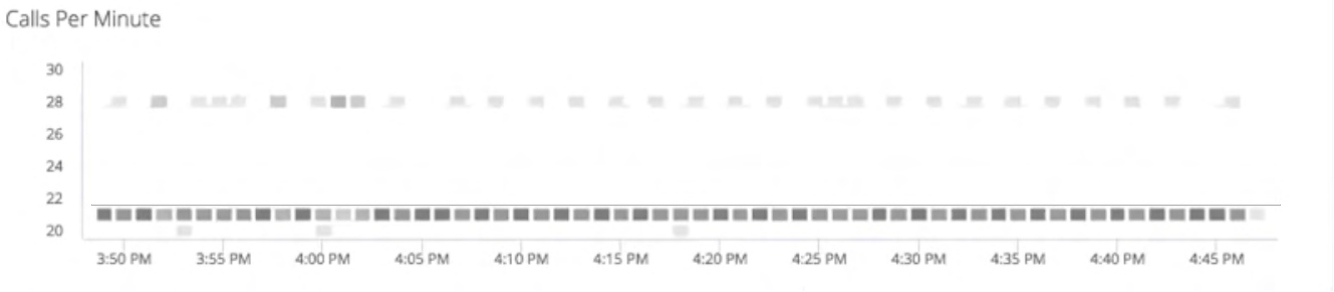
When you select a set of transaction outliers, the Selected Nodes and Impacted Servers charts show the distribution of these outliers across all nodes in the tier. This makes it easy to determine if these outliers are associated with specific node clusters. Double-click a pie chart to troubleshoot the node or server.



Example Use Cases

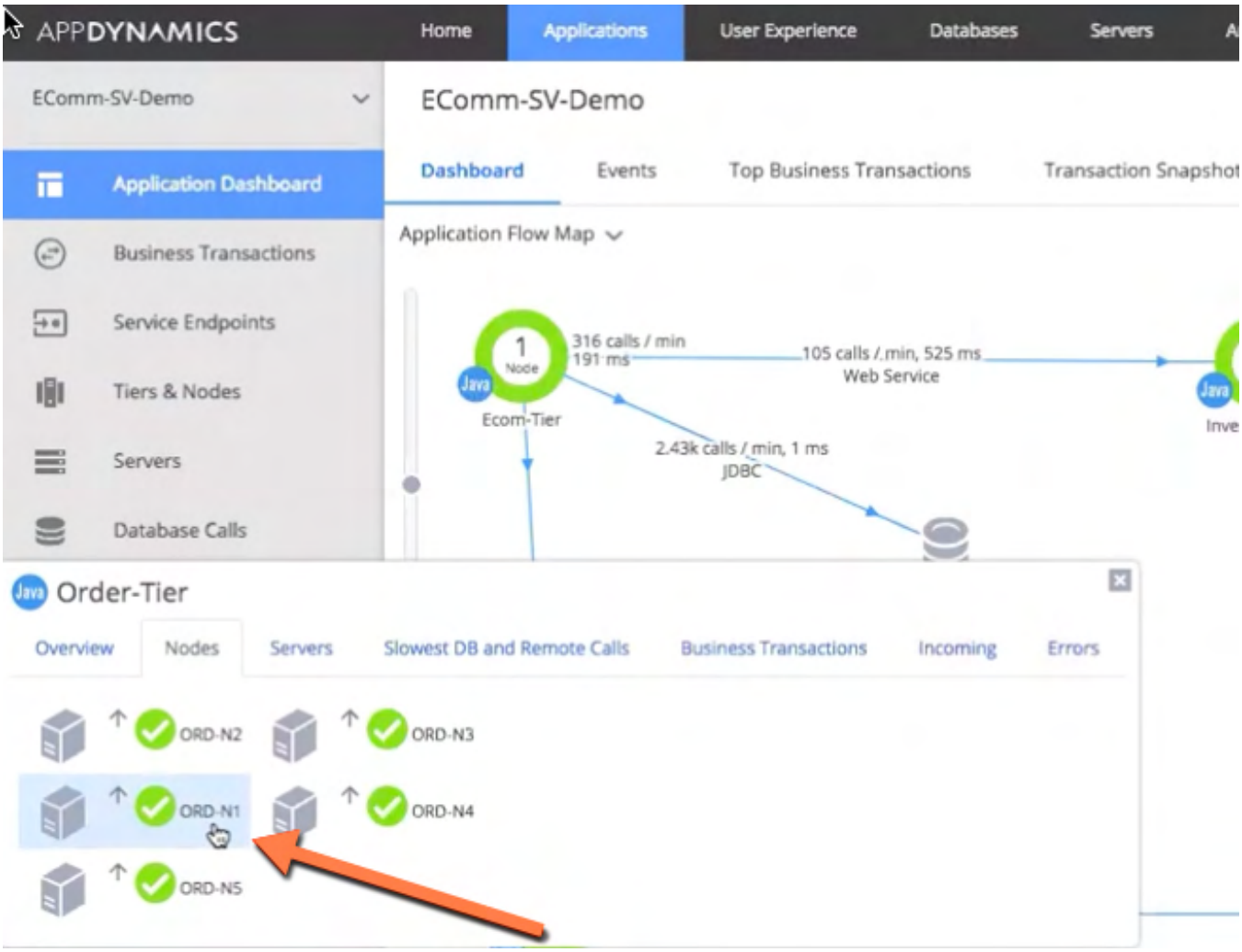
Comparing Load Distributions

In this example, a DevOps engineer wants to ensure that transaction calls are getting distributed evenly to all the nodes in a tier. The engineer accesses the **Tiers & Nodes** view, right-clicks a tier, and selects **Correlate Metrics**. Looking at the Calls Per Minute heatmap, the engineer can immediately see that the band of performance is 20-22 calls per minute, but for some nodes, the rate is higher during certain intervals. The engineer decides to investigate the relevant load balancer and finds that a simple misconfiguration is causing the device to distribute calls unevenly at certain times. Using heatmaps, they can identify and fix a minor issue before it has a significant impact on their team's mission-critical applications.



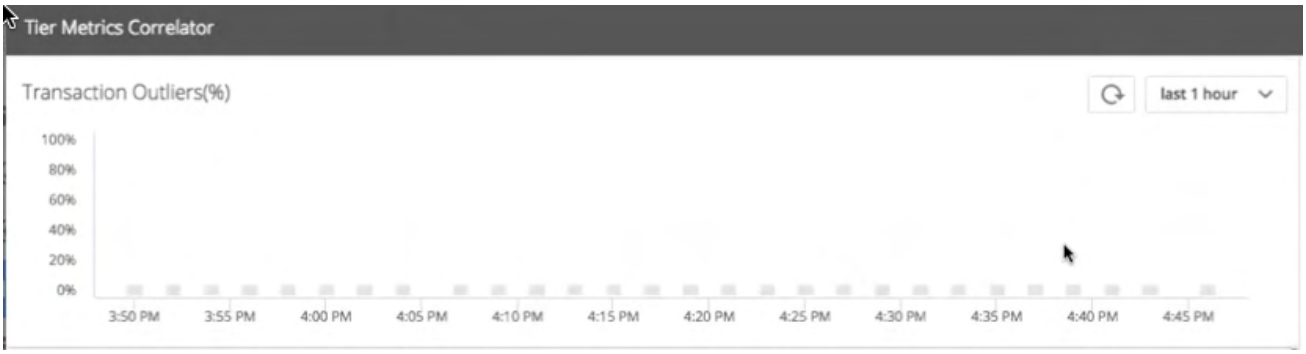
Canary Deployment Testing

A DevOps engineer is responsible for a four-tier e-commerce application. The Order-Tier has five nodes running version 1.0 of the service. The engineer deploys a "canary" (version 1.1 of the service) on one node. Before the engineer deploys 1.1 on all nodes, they want to determine if there is any performance degradation on this node.

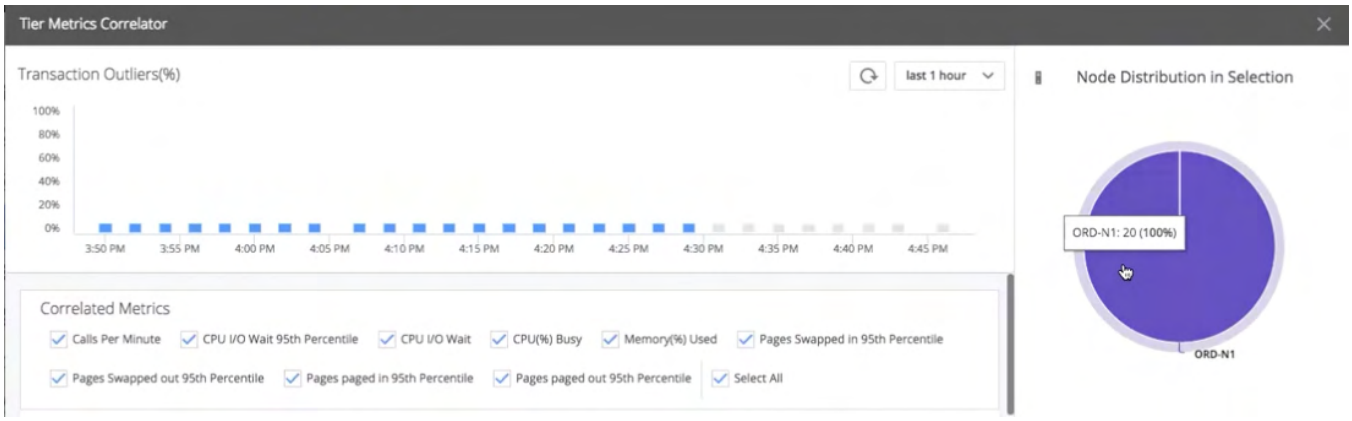


The engineer opens the Controller, accesses the **Tiers & Nodes** view for the application of interest, right-clicks the Order-Tier, and selects **Correlate Metrics**. The Tier Metric Correlator displays.

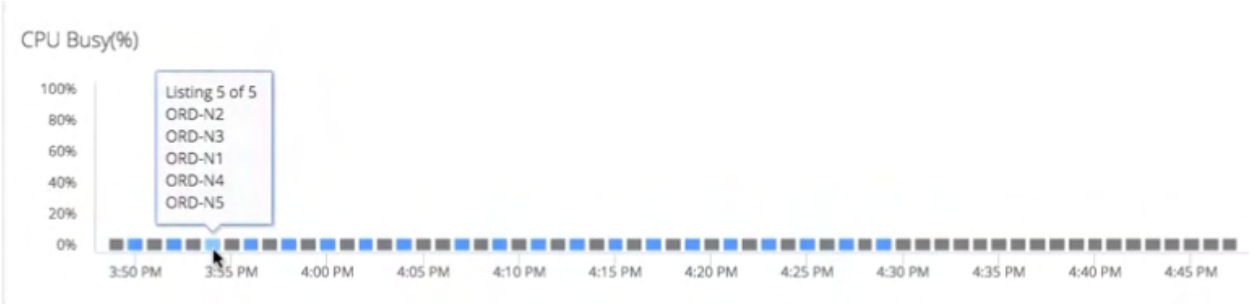
The Transaction Outliers heatmap shows that some calls are outliers: Errors, Slow Calls, Very Slow Calls, or Stalled Calls whose response times are significantly higher than the band of performance for that tier.



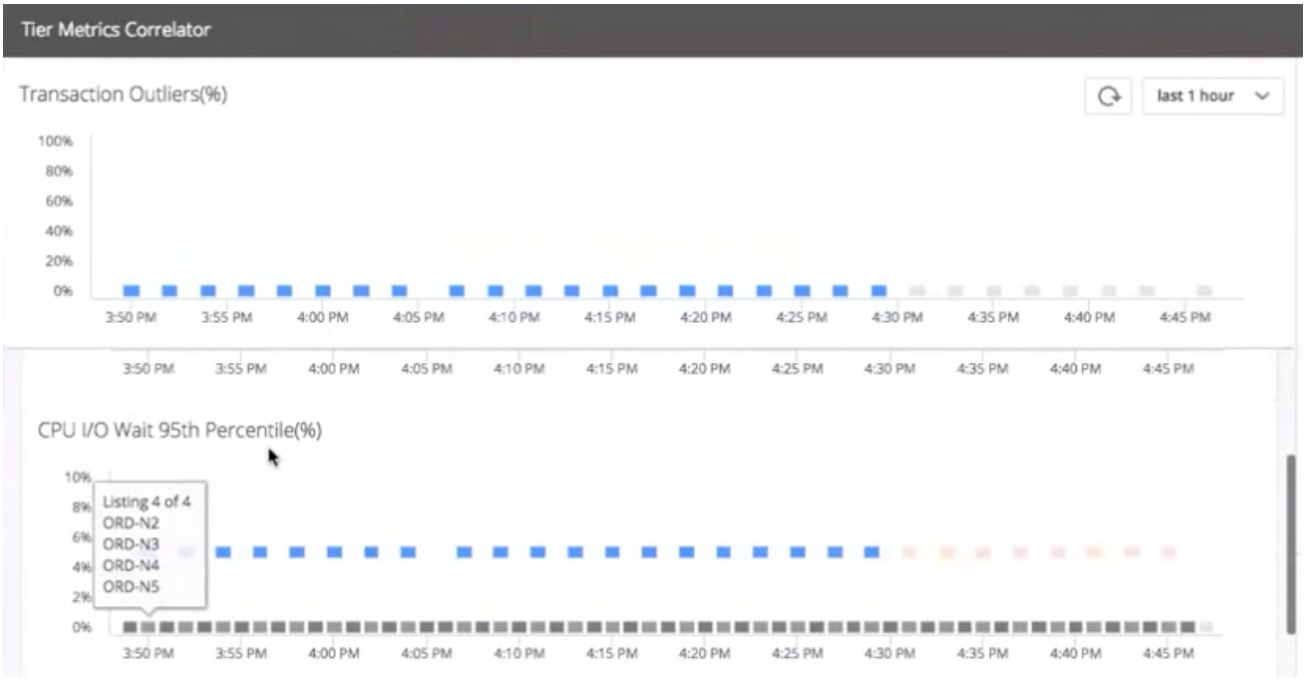
The first question is: *Are these outliers associated with our "canary node" (ORD-N1)?* The engineer drag-selects a set of these outliers. The **Node Distribution in Selection** pie chart (right) shows that all outliers are associated with the canary node. Clearly, the new code is not performing as well as the old code.



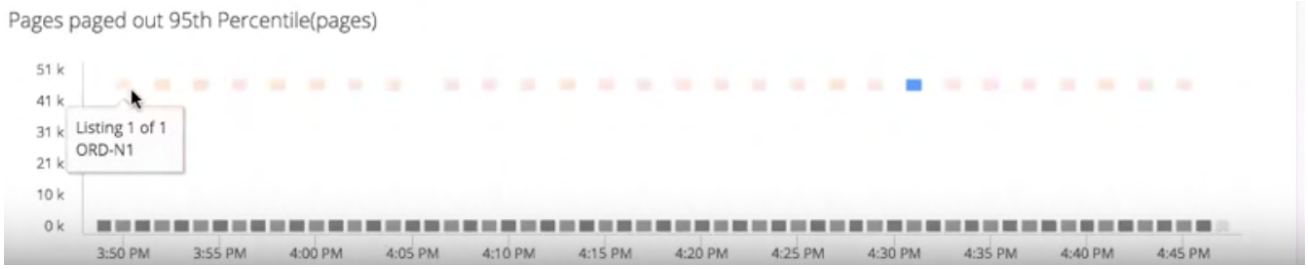
The next question is: *Are any resource issues causing these outliers?* The engineer examines the **Correlated Metrics** heatmaps to look for metrics that correlate with the outliers on the canary node. Most heatmaps show no correlation. For example, **CPU Busy%** shows that all nodes stay within the band of performance of 0-20%.



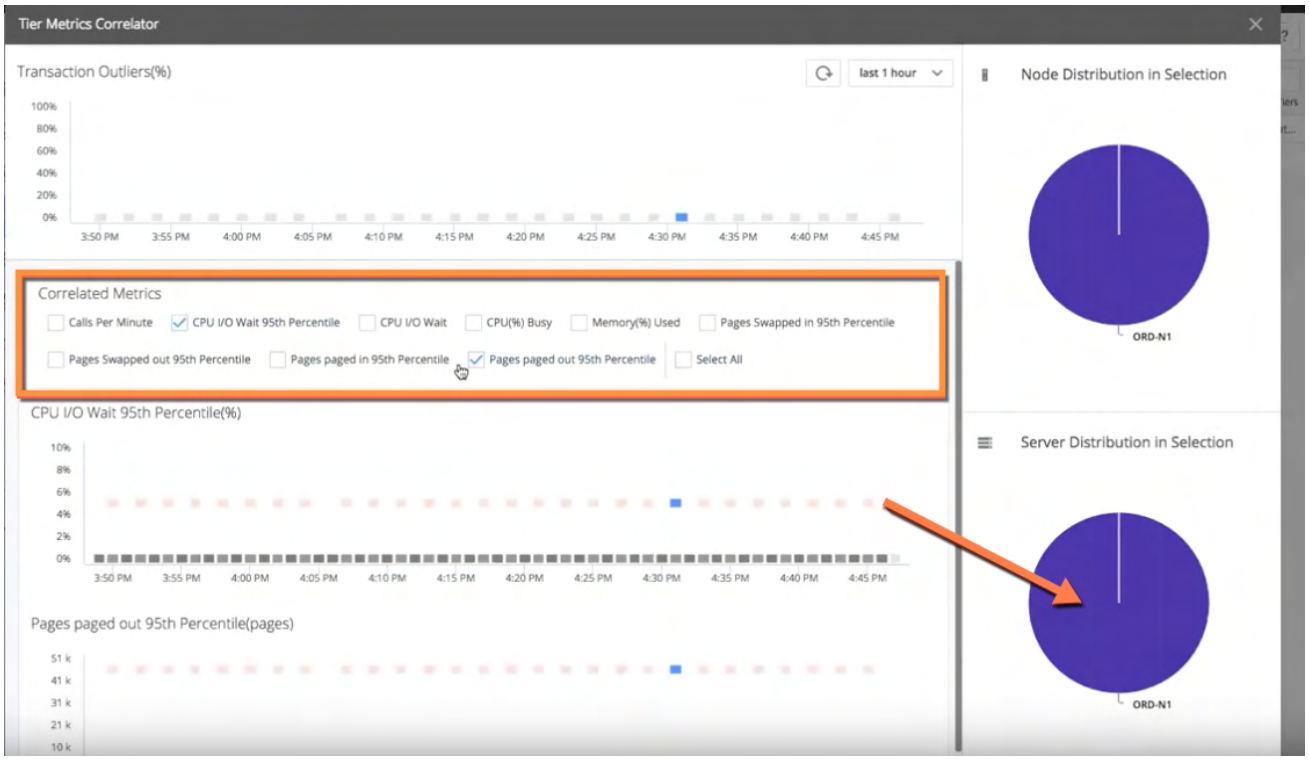
However, the CPU I/O Wait 95th Percentile(%) heatmap shows a strong correlation: All the metric outliers occur on the canary node, while all other nodes remain within the band of performance.



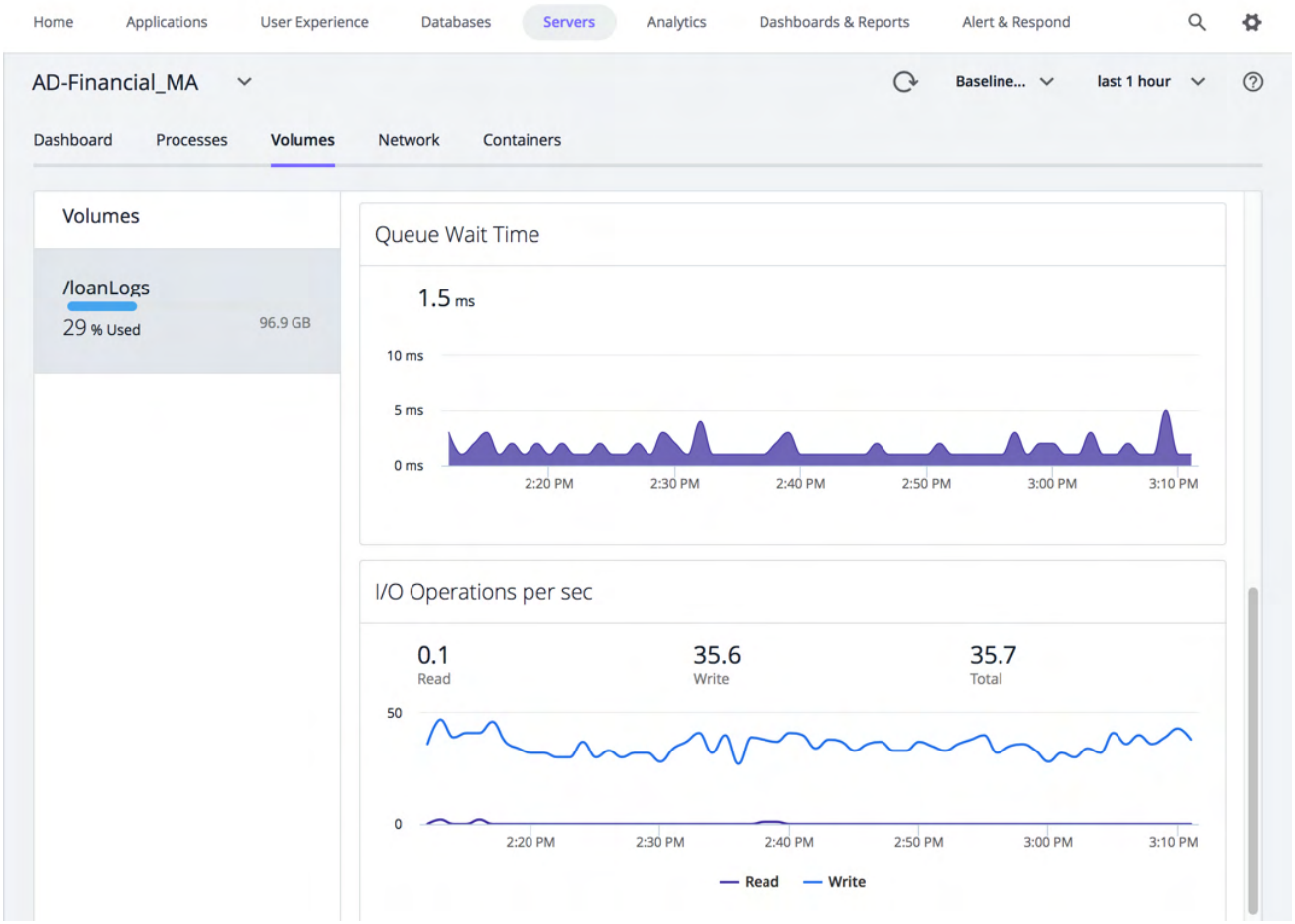
The **Pages paged out 95th Percentile (pages)** heatmap also shows a strong correlation with the transaction outliers on the canary node. With just a few clicks, they can immediately see that the canary node is performing worse; that the node has a CPU I/O problem; and that the CPU I/O problem is related to paging, which indicates a disk problem.



To reduce the visual noise and highlight the correlations, the engineer unchecks all the uncorrelated metrics. The next step is to investigate and troubleshoot the underlying server. To drill down into the canary node, they double-click on the **Server Distribution in Selection** pie chart (bottom right).



The Server Dashboard for the canary node displays. The engineer selects the Volumes tab and sees many spikes in I/O operations and queue wait times. They decide that the canary code is not ready to deploy to the entire tier, and need to re-examine the canary code, fix the regression, and re-test.



Enabling Tier Metric Correlation

You enable Tier Metric Correlation per account on the Controller.
If you are using a SaaS Controller, contact AppDynamics Support.
If you are using an on-premises Controller:

1. Log in to the Controller administration console using the `root` user password:
   `http://<controller host>:<port>/controller/admin.jsp`
2. Access the Accounts page, and select the account for which you want to enable this feature.
3. Select **Add Property** in the accounting settings and add:
   `ENABLE_SIM_HEATMAPS` = true

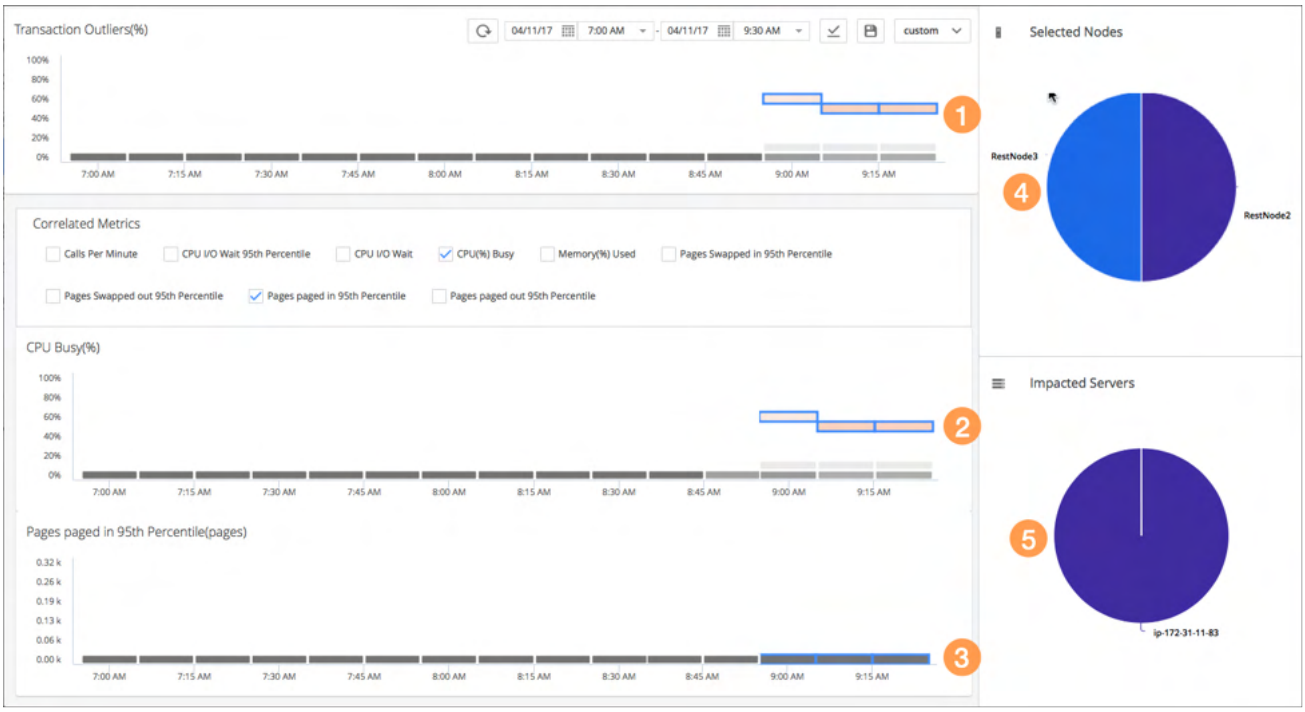
To correlate percentile metrics, you must enable percentile metric reporting on both the Controller and the Machine Agent. By default, reporting is *disabled* on the Controller and *enabled* on the Agent.

- To enable/disable reporting on the Controller, log in to the Controller administration console and set the `sim.machines.percentile.percentileMonitoringAllowed` property. See Controller Settings for Machine Agents
- To enable/disable reporting on the Agent, open the `<machine_agent_home>/extensions/ServerMonitoring/conf/ServerMonitoring.yml` file and set the `percentileEnabled` property. See Machine Agent Settings for Server Visibility

Workflow Description

These steps outline the workflow description:

- Access the **Tiers & Nodes** view for the application of interest.
- Right-click the tier and select **Correlate Metrics**. The Tier Metric Correlator displays.
- Identify *transaction outliers* (1) for the tier – calls flagged as Slow, Very Slow, Stalls, or Errors (see Business Transaction Performance).
- Drag the cursor to select the outliers of interest. The pie charts on the right show the distribution of these outliers by node and server.
- Identify any *correlated metrics* (2) for the outliers. Unselect checkboxes for non-correlated metrics (3).
- Identify and troubleshoot the nodes (4) and servers (5) where the outliers are occurring. Double-click a pie slice to open the dashboard view. You can use the correlated metrics to guide your troubleshooting.


Transaction Outliers(%)
04/11/17
7:00 AM
04/11/17
9:30 AM
custom
100%
80%
60%
40%
20%
0%
7:00 AM
7:15 AM
7:30 AM
7:45 AM
8:00 AM
8:15 AM
8:30 AM
8:45 AM
9:00 AM
9:15 AM
1
Correlated Metrics
Calls Per Minute
CPU I/O Wait 95th Percentile
CPU I/O Wait
CPU(%) Busy
Memory(%) Used
Pages Swapped in 95th Percentile
Pages Swapped out 95th Percentile
Pages paged in 95th Percentile
Pages paged out 95th Percentile
CPU Busy(%)
2
Pages paged in 95th Percentile(pages)
0.32 k
0.26 k
0.19 k
0.13 k
0.06 k
0.00 k
3
Selected Nodes
RestNode3
RestNode2
4
Impacted Servers
5
ip-172-31-11-83

Service Availability Monitoring

Effective immediately, AppDynamics is ending support for Service Availability Monitoring. Customers who have never purchased SAM will no longer be able to do so. Renewals and additional licenses by existing customers of SAM that are purchased before June 2021, for a one-year term only, are permitted. All existing users of SAM can continue to use it. See Support Advisory: Service Availability Monitoring End of Life (EOL) Notice.

To access Service Availability Monitoring, select **AppDynamics Home > Servers** > **Service Availability** (left navigation menu).

The Service Availability Monitoring feature of Server Visibility enables you to monitor internal or external HTTP and HTTPS services. You configure service monitoring from the Controller UI. After you configure monitoring for a service, Service Availability Monitoring evaluates the service:

- The Agent evaluates each response, based on your specified violation rules, and flags each response as failed (rule violation) or successful (no violation).
- The service monitor maintains a rolling buffer of evaluated responses. This buffer has a configurable window size (number of evaluated responses), success threshold (number of successful responses), and failure ratio (number of failed responses).
- The monitor collects evaluated responses until the response buffer is full. Then, it evaluates the service as:
    - NORMAL - Successful responses are greater than or equal to the success threshold.
    - CRITICAL - Failed responses are greater than or equal to the failure threshold.
- When the buffer is full, the monitor re-evaluates the service every time it evaluates a new response.

For example, with the following values:

- Success threshold = 3
- Failure threshold = 1
- Results window size = 5

The Agent waits until five check results are received, and from these five results, if there is one failure, then the target state is CRITICAL. If there are at least three successful results, then the target state is NORMAL.

Licensing and System Requirements

Service Availability Monitoring requires a separate license and a Server Visibility license. Server Visibility is currently available for Linux, Windows, and Solaris. See License Management and Server Visibility Requirements and Supported Environments

Viewing the Monitored Services

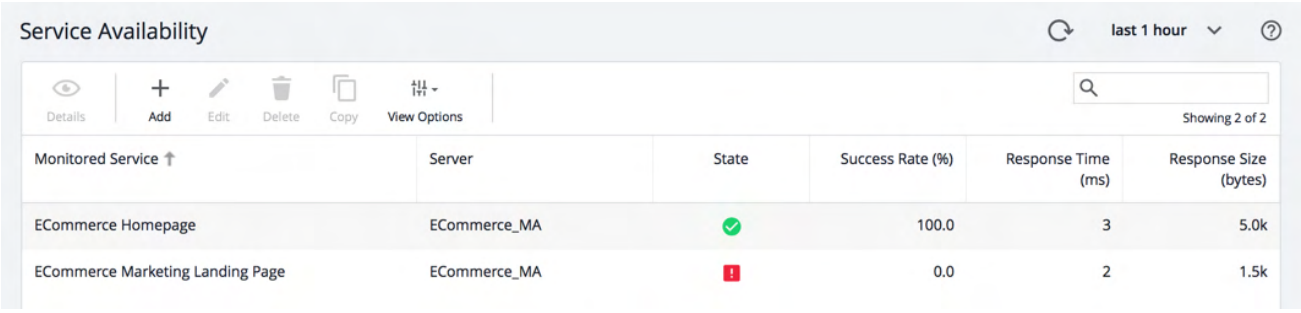
You can view the monitored services in the Monitored Services list. The state is determined by evaluating the Response Validators during the results window. Response Validators are rules you configure that are used to evaluate against the responses received from the service.

The possible states are:

- NORMAL - The number of successful responses in the result window is greater than or equal to the configured success threshold.
- CRITICAL - The number of failed responses in the result window is greater than or equal to the configured failed threshold.
- UNKNOWN - If the Machine Agent does not provide any data.

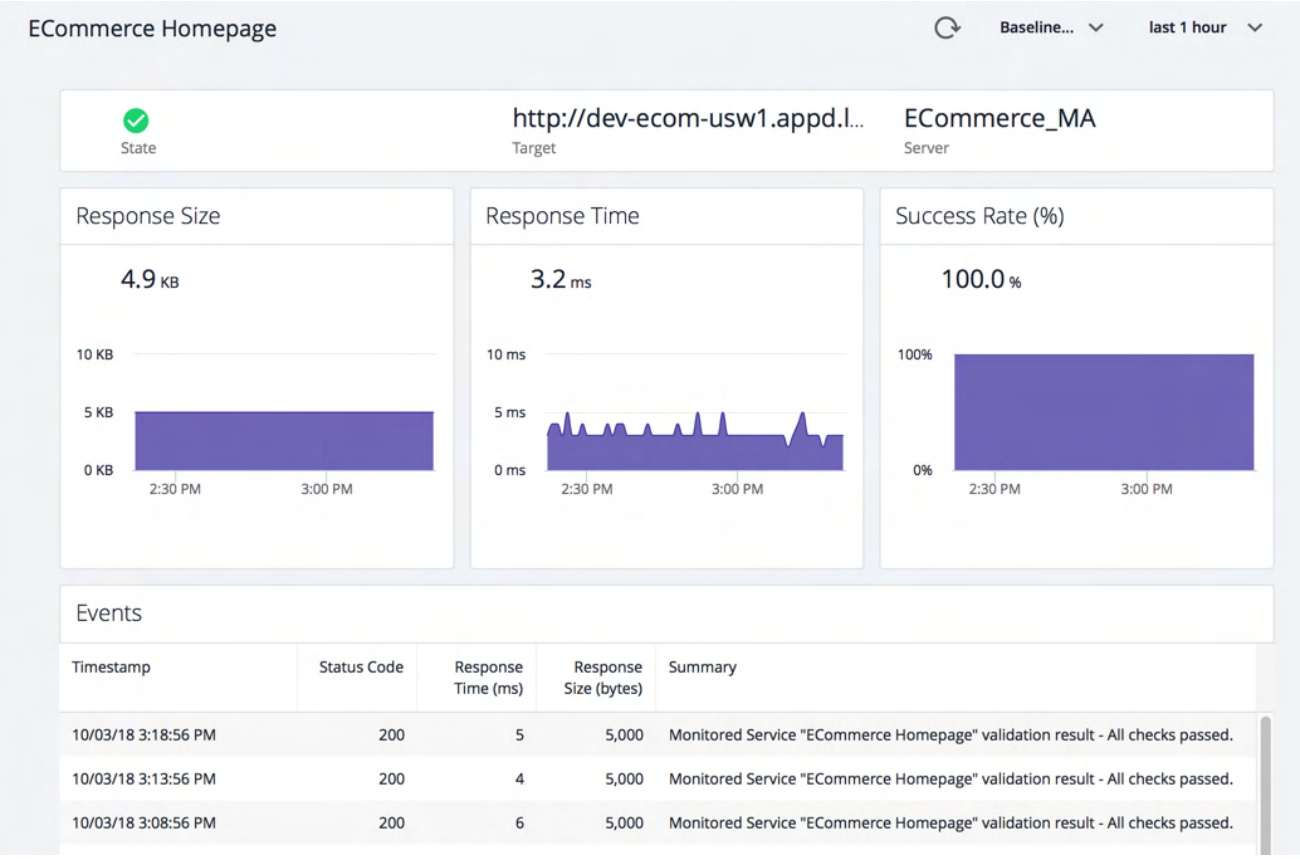
Monitored Services List

From the top navigation bar, select **Servers > Service Availability** to view a list of monitored services. Use the action toolbar to manage your monitored services by adding new ones, viewing details, and so on.



Monitored Service Details

Select a monitored service, and double-click to view the associated details.



Configure Service Availability

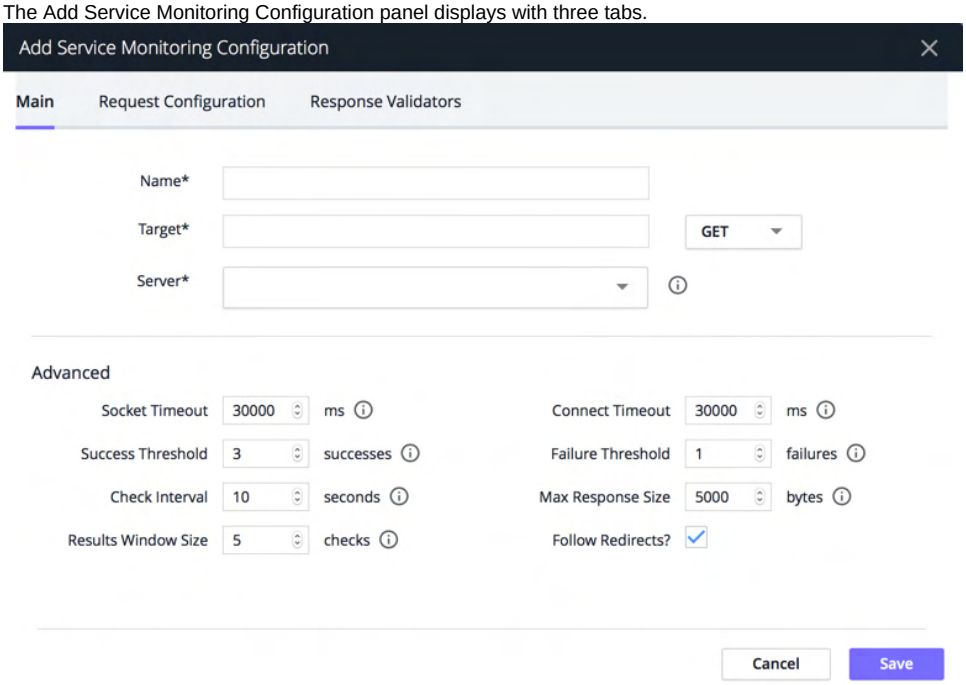
By default, configuration changes take effect within at least two minutes of the change, so you do not need to restart the Machine Agent after changing the configuration. If the protocol, target, or method of a service are changed once a service is created, the state of the service is set to UNKNOWN and is re-evaluated with the new configuration.

To configure this feature, users need the *Configure Server Visibility* permission. See Server Visibility Permissions

To access the configuration panel:

1. From the Controller top navigation bar, select **Servers > Service Availability**.
   A list of monitored services display, if any are configured.

2. To add a service, select **Add**.
   The Add Service Monitoring Configuration panel displays with three tabs.



3. From the Main tab, specify the required fields and save the configuration. Each field has an associated tooltip that provides help. See Main Configuration Panel.
4. From the Request Configuration tab, specify your request headers. See Request Configuration
5. From the Response Validators tab, configure your validation rules. See Response Validators

Main Configuration Panel

Field	Description	Default	Required
Name	Your name for this target configuration. This name displays in the Service Availability list.	N/A	Yes
Target	The resource to be monitored, for example, *http://myThirdPartyService.com/data*. Specify which HTTP method to use to send the request (`GET`, `POST`, `HEAD`)	N/A	Yes
Server	Name of the Machine Agent performing the monitoring. Only servers that are enabled for Server Visibility display in the dropdown.	N/A	Yes
Socket Timeout	How long to wait, in milliseconds, after a successful connection for a complete HTTP response.	30000	Yes
Success threshold	Number of required successes within the results panel for a NORMAL state.	3	Yes
Check Interval	Interval time in between checks in seconds. ⓘ Setting this interval to more than 60 seconds will result in visible gaps in the metric graphs for monitored services.	10	Yes
Results Window Size	Number of most recent results to use in determining the state of the service.	5	Yes
Connect Timeout	How long to wait (in ms) for the service to respond to a connection request.	30000	Yes
Failure threshold	Number of required failures within the result window for a CRITICAL state.	1	Yes
Max response size	Maximum response size to collect in bytes.	5000	Yes
Follow redirects	Deprecated. Follow redirect to determine the state of service.	No	No

Restricted Addresses

By default, these addresses are restricted:

- Loop back address: `localhost, 127.0.0.1, ::1(IPv6)`

- Site local address: `192.168.0.0/16, 10.0.0.0/8, 172.16.0.0/12, fec0::/8(IPv6)`
- Link local address: `169.254.0.1 - 169.254.255.254, fe80::/10(IPv6)`
- Any local address: `0.0.0.0, ::(IPV6)`
- Multicast address: `224.0.0.0/4(224.0.0.0 to 239.255.255.255), ff00::/8(IPV6)`

Request Configuration

You can define a list of customized headers to send with requests. For example, the list can mimic desktop or mobile browsers. You can also define a request body for `POST` requests. You can specify an "Authorization" header", if credentials are required.

Protocol	Field	Value Type	Default	Description
HTTP	header name and value pair	String	empty	Customized header to send with requests.
	body	String	empty	Any data to send with the request. Usually used for POST.

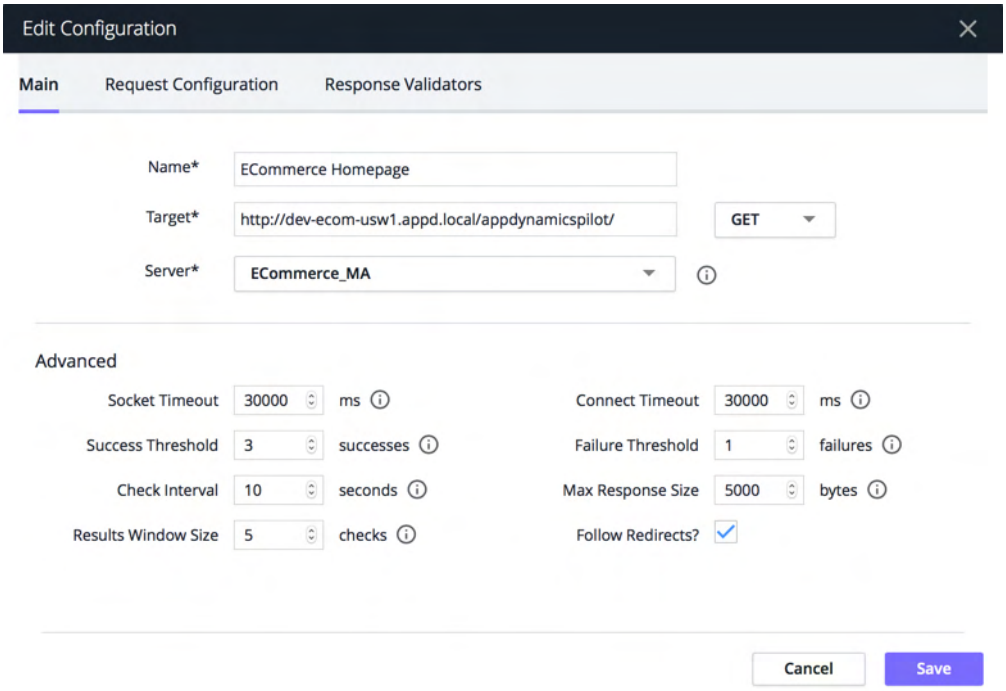
Response Validators

You can provide a list of rules for the Machine Agent to use to validate the health of a monitored service. If any rule is violated, the response is considered failed.

For each rule you can specify:

- HTTP property
- operator
- value

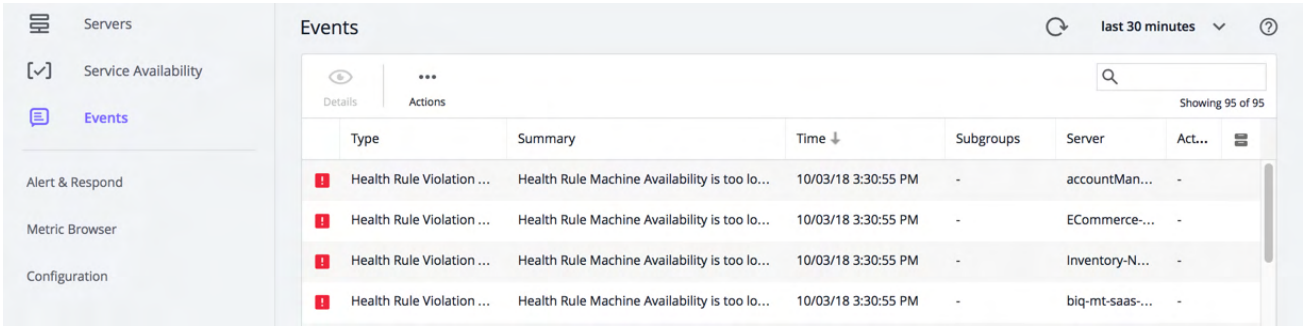
The list of operators varies depending on the property selected in the first dropdown.



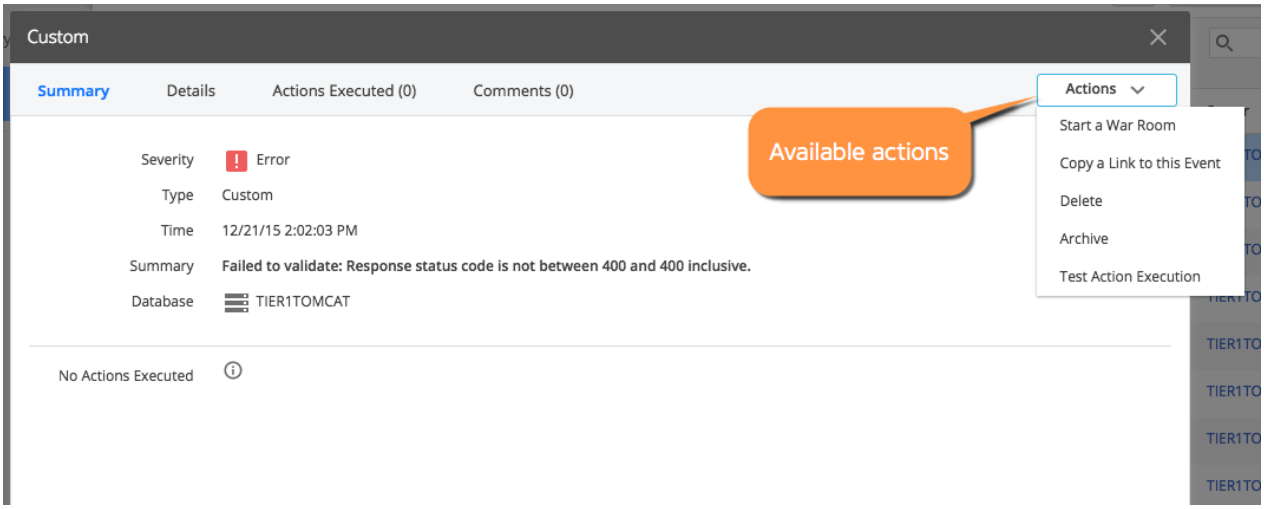
Monitoring the Service Health

An event is sent to the Controller for every state change and periodically (by default every 5 minutes). You can change the update interval by setting the `System property: appdynamics.machine.agent.sam.event.updateIntervalMillis`. See Configuration Property Reference

The events are visible from the Monitored Service details panel and from the Server Visibility Events list.



Double-click the event to review details.



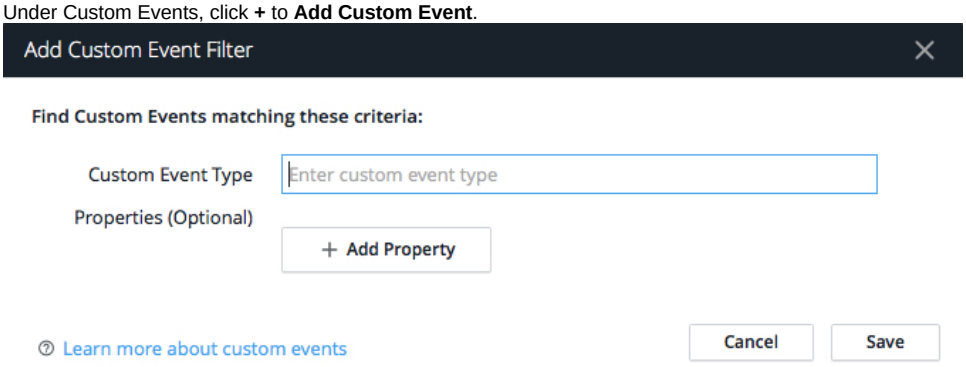
The event properties are as follows:

- `Category: CUSTOM`
- `Event Type: Service Availability`
- `Property: severity` Values can be ERROR (failure) or INFO (passing).

Configuring Alerts

To create alerts for your monitored services:

1. From the Servers panel, select **Alert & Respond**.
2. Select **Policies**.
3. Select **Create Policy Manually**.
   If policies already exist, you may not see this option. Click **+** to add a policy manually.
4. Under Custom Events, click **+** to **Add Custom Event**.



5. In the Actions section of the Policy panel, add the action you want to execute if the policy violates and save the policy.

Server Visibility Events

To access Server Visibility Events, from the top navigation bar > **Servers** > **Events**.

You can view Machine Agent events on the Server Events list, and on the Application Events list for applications associated with the Machine Agent. See Monitor Events.

Machine Agent events include:

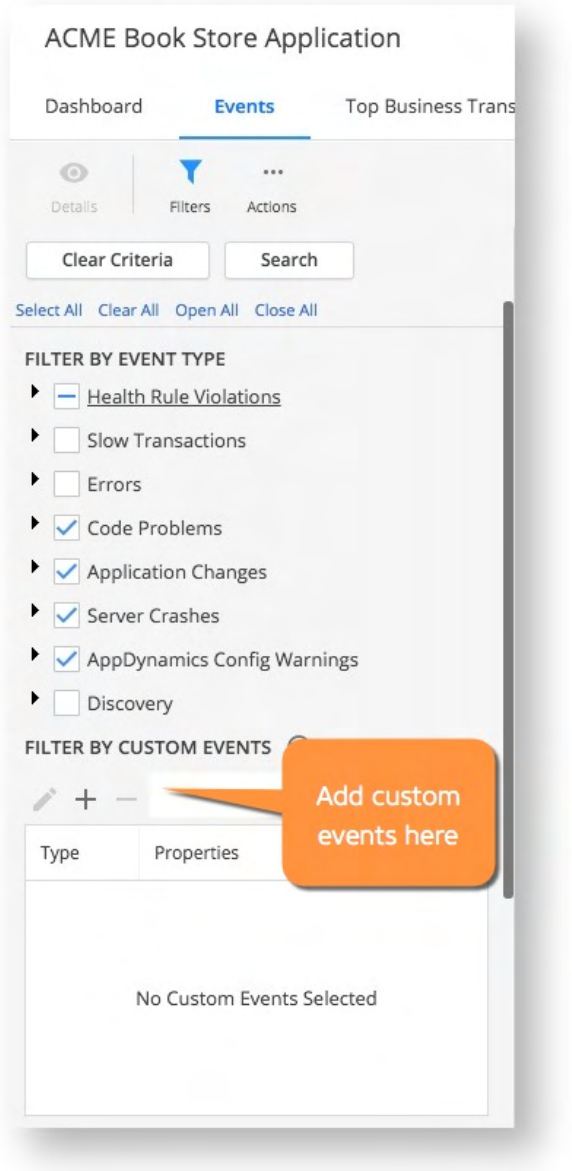
- Custom Events
- Agent Internal Diagnostic Events

Custom Events

Custom events include events generated by the extensions, and Service Availability Monitoring events. Service Availability Monitoring events are custom events that are sent to the Controller periodically, and for every state change in the service's health.

Events triggered by extensions are custom events and also display on the **Servers > Events** list.

To view custom events in the application Events list, you can add them to the event list filters.



Agent Internal Diagnostic Events

AGENT_DIAGNOSTICS

- Description – Diagnostic information concerning Agent activity, such as resetting a Machine Agent. The reset functionality is available for pre-4.2 version Machine Agents only.
- Category – AppDynamics Internal Diagnostics
- Name – `EventMessage.MACHINE_AGENT_RESET_SUCCEEDED`

Machine Agent Settings for Server Visibility

You can modify the default configuration for Server Visibility. This page describes the configurable settings and their default values.

Edit this configuration file: *<machine-agent-home>*/extensions/ServerMonitoring/conf/ServerMonitoring.yml

Edit YML Files

When editing YML files:

- If you make any changes to ServerMonitoring.yml, or to any other YML file, make sure that the modified file meets standard YML syntax rules.
  Follow these important YML good practices:
  - Do not include any tab characters. Use whitespace characters only to indent fields.
  - Use the exact same number of whitespace characters to indent entries that are on the same level.
  - Use a plain-text editor, not a WYSIWIG editor, to edit the file. Use a monowidth/monospaced font to view the contents.
  - Always save using UTF-8 encoding.
  - Test and verify the edited file using an online YML syntax tester.
- The safest way to edit a setting in a YML file is to:
  - Copy the line you want to edit, and paste the copy into a new line. Make sure that you select, copy, and paste the entire line.
  - Comment out the original line and edit the copy, as desired.
- To add or edit a regular expression in this file, you should test and verify the regular expression using an online regex tester.
- Before you make any changes to this file, review the option descriptions on this page, and the comments in the file, carefully before you change a setting.
- The Agent updates dynamically in response to Agent configuration property changes, so you do not need to restart the Agent after you update this file.

Configurable Server Visibility Settings

Setting	Description
basicEnabled	Indicates whether the Machine Agent should report the basic metrics through the SIM extension. Set this to false to use Sigar to report basic metrics. This setting only affects monitoring on Linux. Default = true
volumeMonitorConfig: maxNumberVolumes	Do not report more than *N* volumes, where *N* = maxNumberVolumes. Default = 5 Changing this setting can affect the resource consumption of your deployment. Before you increase this setting, verify that your application environment and Controller can process the increased resource requirements.
volumeMonitorConfig: whitelistSelectorRegex	Volumes with names that match this regular expression are always reported, up to the maximum specified by maxNumberVolumes. Default = ""
volumeMonitorConfig: blacklistSelectorRegex	Volumes with names that match this regular expression are excluded. This setting is useful for filtering out irrelevant metrics. If a volume name matches both the blacklist and whitelist regexes, metrics for that network are reported (the whitelist takes priority) up to the maximum specified by maxNumberVolumes. The default ServerMonitoring.yml file does not include a blacklistSelectorRegex field for volumes. If you want to add one, use the same indentation and formatting as the volumeMonitorConfig:whitelistSelectorRegex field.
volumeMonitorConfig: samplingInterval	Specifies a custom sampling interval for collecting volume metrics on Linux. Default = 3000

`networkMonitorConfig:` `maxNumberNetworks`	Do not report more than *N* networks, where *N* = `maxNumberNetworks`. Default = 5 Changing this setting can affect the resource consumption of your deployment. Before you increase this setting, verify that your application environment and Controller can handle the increased resource requirements.
`networkMonitorConfig:` `whitelistSelectorRegex`	Networks with names that match this regular expression are always reported, up to the maximum specified by `maxNumberNetworks`. To report metrics for one or more virtual networks, specify a regex that matches the virtual network names. When collecting Process metrics on Solaris, the Standalone Machine Agent observes and captures only the first 80 characters of each process name and argument list. This means that the Agent considers only the first 80 characters of each process string when it applies whitelists. Default = ""
`networkMonitorConfig:` `blacklistSelectorRegex`	Networks with names that match this regular expression are excluded. This setting is useful for filtering out irrelevant metrics. If a network name matches both the blacklist and whitelist regexes, metrics for that network are reported (the whitelist takes priority) up to the maximum specified by `maxNumberNetworks`. The default regex excludes virtual networks. To monitor a set of one or more virtual networks, edit `whitelistSelectorRegex` to include the networks of interest. To monitor all virtual networks, change the `blacklistSelectorRegex` to an empty string. When collecting Process metrics on Solaris, the Standalone Machine Agent observes and captures only the first 80 characters of each process name and argument list. This means that the Agent considers only the first 80 characters of each process string when it applies blacklists. Default = "^veth.*|^vnet.*"
`defaultProcessClassSelector`	The default "class selector" based on a *`class_selector_regex`*. If this regex is defined, and a match is found in the process command line, the class name is the first group occurrence of that regex in the command line. Default = ""

`processClassSelectorRegexList`	A list of *`class_name`*:*`regex`* mappings. If the command line for a process matches regex, the metrics for that process are assigned to *`class_name`*. This setting is useful when you want to ensure that high-priority processes get reported, even if the number of defined classes is higher than the `maxNumberMonitoredClasses` setting.
	
	For example:
	
	`processMonitorConfig:`
	
	`processClassSelectorRegexList :`
	
	`machineAgentTasks: '.*java.*machineagent.*'`
	
	`controllerTasks: '.*java.*controller.*'`
	
	`nextOne: '.*svchost.*'`
	
	The Machine Agent assigns a process to a class as follows:
	
	1. Assign to the first match in `processClassSelectorRegexList`.
	2. If there is no match for the `processClassSelectorRegexList` (step1), apply the `defaultProcessClassSelector` regex to the command line.
	3. If there is no match for the `defaultProcessClassSelector` regex (step 2), use the process name (truncated if the name exceeds the `maxClassIdLength`).
	
	These steps outline the recommended workflow for updating this list:
	
	1. The default `ServerMonitoring.yml` file includes an example that is commented out. If you are updating the default list for the first time, you should:
	a. Create a copy of the example.
	b. Uncomment the copy (remove the <!-- and --> comment tags), and edit it as needed.
	2. Choose the process(es) that you want to monitor on the host machine.
	3. Create a regex to match the process name(s) of interest. You should test the regex using an online regex validator.
	4. Add the regex to the list. You should order the mappings by priority, highest to lowest. If a command line matches multiple regexes, the first match is used.
	5. Save the `ServerMonitoringConfiguration.yml` file.
	6. Wait 15 minutes or longer for the updated list to take effect, and then verify that the matching process(es) display in the Controller UI.

`samplingInterval`	Indicates how often to gather metric data. Units in milliseconds. Default = 30000 (30 seconds)
`maxClassIdLength`	Specifies the maximum process class name length. Any process class name that is longer than the specified maximum is truncated. The global maximum of the process class name is 100. If this variable is set to be greater than 100, then the process name is truncated at 100. Default = 50
`processSelectorRegex`	Contains a regular expression that specifies which processes should be monitored by the Machine Agent. The regular expression is compared against the full command line that was used to start the process. The default regular expression will filter out any processes where the command line ends with a close bracket (']'). For Linux, the process arguments could not be found, which usually indicates a kernel process. For Windows, no processes should end with a bracket character, so the regex should include all processes on Windows. Default = ""^.+[^]]$""
`minLiveTimeMillisBeforeMonitoring`	Specifies the minimum amount of time a process must be alive before it is monitored by the Machine Agent. Use this to prevent the Machine Agent from being overloaded by monitoring short-lived processes. Units in milliseconds. Default = 60000 (60 seconds)
`maxNumberMonitoredClasses`	Specifies the maximum number of process classes that the Machine Agent monitors. The processes that are reported are using the highest CPU and memory that match the regex specified by `processSelectorRegex`. Default = 20 Changing this setting can affect the resource consumption of your deployment. Before you change this setting verify your application environment and Controller can process the increased resource requirements.
`defaultDiskSectorSize`	Specifies the default sector size (in bytes) for each disk if the Machine Agent cannot determine the sector size. This value is used to calculate the number of bytes read/written for the disk. Default = 512
`memoryMonitorConfig`	Specifies a custom sampling interval for collecting memory metrics on Linux. Default = 3000
`cpusMonitorConfig`	Specifies a custom sampling interval for collecting CPU metrics on Linux. Default = 3000

`tag`	A list of user-defined tags for the individual server. Use these tags to query, filter, aggregate, and compare related servers. See Server Tagging
	Each tag is specified by a `key-value` pair. You can define tag names up to 127 unicode characters, and tag values up to 255 unicode characters. Define each key on a separate line. All key-value strings should be within single quotes. If a key has multiple values, delineate the list with commas:
	`<key>: [ <value> ]`
	`tags:`
	`  'Location': ['NYC', 'Data Center', 'Server Room 7']`
	`  'Environment': ['preProduction']`

Docker Visibility Settings

See Configuring Docker Visibility.

Process Limits

These additional configurable settings are available in the Controller Admin UI:

- Total number of processes displayed in the UI for a single query is 5000 processes per call.
- Total number of processes tracked per account. The default value is 10000 processes per account.

See Controller Settings for Server Visibility.

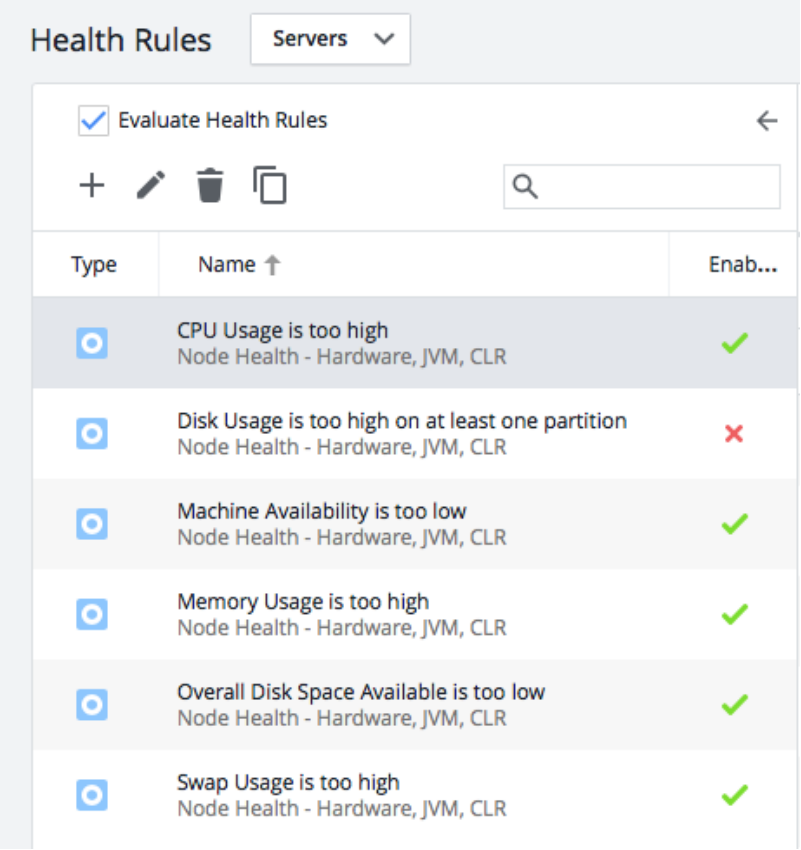
Configure Health Rules to Monitor Servers

You can configure Server Visibility to alert you when certain conditions are met or exceeded based on monitored server hardware metrics. If you are new to AppDynamics, the Getting Started Wizard can help you through the process. Alternatively, you can create alerting policies manually if you are already familiar with the process.

You configure Health Rules, Actions, Policies, and Email Digests for monitoring servers similar to how you would configure these for monitored applications, with some minor changes.

Health Rules

Default health rules based on several usage metrics are enabled for Server Visibility. You can modify these and set up your own health rules. Select **Alert & Respond** > **Health Rules** to view a list of the rules on the Controller.



From the Affects tab of the **Health Rule** wizard, you can choose to apply the Server Health Rule to:

- Subgroups (All or selected subgroups)
- Machines
    - All machines in the active account
    - Machines within selected subgroups
    - Selected machines
    - Machines whose names match certain criteria

The health rule is violated when specified critical or warning conditions are met.

Hierarchies, Groups, and Subgroups

You can apply health rules to subgroups or machines within selected subgroups. Subgroups available on the Affects tab of the Health Rules wizard are machines that are grouped into hierarchies based on the Machine Hierarchy Property configuration for the Standalone Machine Agent. See Machine Agent Hierarchy.
The subgroups are the leaf group a machine is in. For example, there are three machines:

- A is in group Data Center 1|Rack 1
- B is in group Data Center 1|Rack 2
- C is in group Data Center 1|Rack 1

A and C are in the same subgroup, B is in a different one.

Policies

Server Policy Actions are triggered when any or select Health Rule Violation Events occur. Unlike Application Policies, Server Policies cannot be based on custom events.

Actions

Actions are performed when a health rule is violated and a policy is triggered. Actions can be email or SMS message notifications, HTTP requests, or custom actions that have been uploaded to the Controller. You can also use email and HTTP request templates for alert and respond actions.

Email Digests

Email Digests enables you to send notification of chosen health rule violation events to specific email addresses at a specified frequency, such as every 2 hours.

Controller Settings for Server Visibility

This page describes Controller Admin settings that are specific to Server Visibility. You can apply these Controller settings and permissions from either the **Account** or **Controller** tabs. Changes to the settings take effect the next time the Agent is restarted and connects to the Controller.

You need the root user password to change these settings.

You cannot set all configuration properties at the account-level. Check the Description column for each property to determine whether you can enable a specific Server Visibility Property. See Roles and Permissions and Manage Users and Groups

To change the Controller Admin settings for Machine Agents, see Controller Settings for Machine Agents

Server Visibility Property	Description	Default
`sim.docker.enabled`	Enable Docker Visibility.	`true`
`sim.docker.machine.container.limit`	The global limit for the number of containers to monitor per machine. You can specify this in the Access the Administration Console as a Controller setting (all accounts) or as an account setting for individual accounts. The maximum limit you can specify is 120 containers.	`15`
`sim.docker.monitorAPMContainersOnly (previously known as sim.docker.infraMode.enabled)`	When set to `true`, allows you to monitor APM containers only. If the value is set to `false`, monitors all the containers on the machine except the Machine Agent container. You can apply this property at an account level and enable the monitoring of containers with and without APM. This property was formerly named `sim.docker.infraMode.enabled`.	`true`
`sim.exceptions.stacktrace.enabled`	When this setting is enabled, Controller stack traces are sent in the error response when a client request encounters an error. Setting this to `true` is a security risk, and should only be set to `true` if you understand the consequences, and if directed by a Support Engineer.	`false`
`sim.machines.agent.process.maxClasses`	The global maximum for the number of process classes collected per machine. • If `maxClasses` (global setting in Controller) is lower than `maxNumberMonitoredClasses` (local setting in `serverMonitoringConfig.yml`), the local setting is overridden and the global `MaxClasses` is the effective maximum for that machine. • If `maxClasses` (global) is higher than `maxNumberMonitoredClasses` (local), the local setting is the effective maximum for that machine. See Machine Agent Settings for Server Visibility Increasing this setting can affect the resource consumption of your deployment. Before you increase this setting, verify that your application environment and Controller can process the increased resource requirements.	`20`
`sim.machines.container.hostid.conflict.check.enabled`	When using Docker Visibility, if the `unique host ID` setting is not configured to use `container ID` in host network mode, the Machine Agent automatically registers the container using the `container ID` as the `host ID`. This setting was introduced in version 20.8 of the Controller, with the default value = `false`. As of Controller >= 20.11 the default value is `true`. If set to true any container with `setting --network=host` and Machine Agent version is <= 20.6 the container does not register.	`true`
`sim.machines.count.maxPerAccount`	A maximum number of machines allowed per account. Any additional machines will not appear. If you have more than 2000 machines reporting to one Controller and need to increase this number, be aware that you might need to increase the `sim.machines.registrations.maxPerSecondPerAccount` setting as well. Changing this setting can affect the resource consumption of your deployment. Before you change this setting, verify your application environment and Controller can process the increased resource requirements.	`2000 /account`

`sim.machines.deleteStaleMachines.maxLimit`	Set the number of machines to be deleted every 10 minutes, which is one cycle of the purger. For better performance, you can purge up to 700 machines per cycle.	`100`
`sim.machines.dmm.defaultMode`	Sets the Dynamic Monitoring Mode and Server Visibility default on the Controller. The allowable settings are: • `KPI` – Collect Key Performance Indicator metrics only • `DIAGNOSTIC` – Collect KPI and Diagnostic metrics • `ADVANCED_DIAGNOSTIC` – Collect All available metrics *You must enter one of these exact strings*. If you enter a different value (such as "advanced-diagnostic"), the Controller will not update the setting and some servers might not appear in the Servers list.	KPI
`sim.machines.dmm.dmmAllowed`	With this option enabled, the Controller collects metrics from each Standalone Machine Agent based on the Dynamic Monitoring Mode and Server Visibility setting on that Agent. If this option is disabled, the Controller collects all available metrics from each Agent.	`false`
`sim.machines.dotNet.HostIdConflictCheckAllowed`	With this option enabled, the Controller throws an exception if a Standalone Machine Agent tries to register with the Controller while both of the following conditions are true: 1. A .NET Agent on the same machine is already registered with the same `<host-id>` specified for the Standalone Machine Agent, and 2. .NET Compatibility Mode is disabled on either the Controller or the Standalone Machine Agent. The generated exception acts as a reminder that that .NET Compatibility Mode mode must be enabled on both the Controller and the Standalone Machine Agent. Do not disable this option unless instructed by AppDynamics Support.	`true`
`sim.machines.hostidMappingAllowed`	You must enable this mode if you want to collect and view Machine or Server metrics on a server with Machine and .NET Agents installed. See .NET Compatibility Mode.	`false`
`sim.machines.offline.toStaleTimeoutMillis`	How much time, in milliseconds, to wait before considering an offline machine to be stale and marked for deletion. If this value is too high, it prevents fresh data from coming in. If the value is too short, it means less history. As of Controller >= 21.2.0 two properties are used in conjunction to purge stale machines: • use `sim.machines.offline.toStaleTimeoutMillis` (172800000 ms) to determine which machines are considered stale. • the `sim.machines.registrations.update.frequency` setting (86400000 ms) to determine the interval to purge stale machines. You may wish to change either or both settings according to your needs. You can specify these in the Access the Administration Console as a Controller setting (all accounts) or as an account setting for individual accounts. If this setting is modified in the Controller then a restart is required. If this setting is modified at the account level, you do not need to restart. Under rare circumstances old logic leads to a machine being deemed eligible for purge earlier than customer configuration.	`172800000 ms` (2 days)
`sim.machines.registrations.hashCache.enabled`	Prevent subsequent registration requests from fetch data and compare if nothing has changed in the request payload.	`true`
`sim.machines.registrations.maxPerSecondPerAccount`	Set the maximum number of registrations allowed per second per account. This setting controls the rate at which concurrent registrations are processed and prevents one account on a multi-tenant Controller from monopolizing resources needed to register the Agents. The default value assumes that the Machine Agents are started evenly across a minute. If you are running 2000 or more machines, you may need to increase this setting to 600/sec. A suggested formula is: ((number of Machine Agents, version 4.2 or higher / 60)). If you use a deployment script that starts all your Agents within a few seconds of each other, you may need to further adjust the max registrations per second. Changing this setting can affect the resource consumption of your deployment. Before you change this setting, verify your application environment and Controller can process the increased resource requirements.	60

`sim.machines.registrations.update.frequency`	Updates the timestamp of the machine on a specific frequency, regardless of stale timeout value. This allows the purger to use only metadata for purging machines.	`86400000 ms` (1 day)
`sim.machines.reuse.enabled`	Reuse SIM Machine entities to handle the ephemeral environment. Support is currently limited to Docker container machines. If set to `false`, every new container will be considered a new machine.	`true`
`sim.machines.simAllowed`	This setting allows you to enable or disable Server Visibility from the Controller.	`true`
`sim.machines.stale.purgeIntervalMillis`	An interval in milliseconds that determines when stale machines are deleted from the Controller database. If the duration of this value is too short, it might overload the server. If the value is too high, then stale machines are deleted more slowly.	`21600000 ms` (6 hours)
`sim.machines.tags.maxPerAccount`	The maximum number of unique tags per account.	`1000`
`sim.machines.tags.maxPerMachine`	The maximum number of tags per machine.	`50`
`sim.metrics.metricBrowser.machineMetricMappings.enabled`	Enables or disables the Metric Browser to display the metrics reported per machine. When this feature is enabled, tier level aggregated metrics, available under Tier node in the Server metric browser, are not visible. With Streamlined Browsing enabled, the **Hardware Resources** and **Custom Metrics** folders do not display.	`true`
`sim.metrics.store.customMetrics.simNodeEnabled`	Enables or disables the persistence of custom metric values to machines monitored by Server Visibility. If the setting is: • `true`: The collector stores and displays custom metric values for applications and machines. Custom metrics are displayed in Metric Browsers for both Applications and Server Visibility. • `false`: The collector stores and displays custom metric values for applications only. Custom metrics are displayed in the Metric Browser for Applications only (not for Server Visibility). Disable this setting if you are collecting a lot of custom metrics. This will prevent potential spikes in the number and rate of metric values stored for individual machines.	`true`
`sim.metrics.trackRegisteredEnabled`	Enable Streamlined Browsing for node metrics. Use this option when you are browsing metrics for tiers that contain multiple nodes.	`false`
`sim.perAccountProperty.enabled`	Enables a user to configure properties at an account level.	`true`
`sim.perAccountProperty.syncFrequencyInMillis`	The interval in milliseconds for the per Account Property data to sync with a database.	300000
`sim.perAccountProperty.syncInBackground`	Enables syncing the per Account Property data with a database in the background. For better performance, set this value to `true`.	`true`
`sim.processes.bulkDelete.enabled`	Delete processes in bulk instead of fetching one by one. For better performance, set this value to true.	`true`
`sim.processes.delete.maxCount`	Set the maximum number of processes to be deleted every 10 minutes, which is one cycle of the purger. For better performance, set the delete `maxCount` processes value to 55000 per cycle.	`10000`
`sim.processes.count.maxPerAccount`	The maximum number of processes stored per account.	`300000`
`sim.processes.count.maxPerMachine`	The maximum number of processes that each Machine Agent can monitor. Increasing this setting can affect the resource consumption of your deployment. Before you increase this setting, verify that your application environment and Controller can process the increased resource requirements.	`1000`

`sim.processes.creation.maxConcurrent`	The maximum number of processes that can be registered simultaneously by the Controller. The default is 5000 processes. The Machine Agent will retry to register the processes if the first attempt fails. Until the process request is accepted and processed by the Controller, processes within that request will be missing from the Process Details page. The limit works as follows: Case 1: Machine Agent sends a request with 6000 processes to register, and the limit is set to 5000, then this request is rejected until the limit is increased. Case 2: Two Machine Agents (MA1 and MA2) each try to register 3000 processes and the limit is set to 5000. Both MA1 and MA2 requests are received by the Controller and only one of the requests will be processed (3000 + 3000 > 5000). So, the Controller can only process one request and rejects the other one. Case 3: Two Machine Agents (MA1 and M2) each try to register 500 processes and the limit is set to 5000. Both requests arrived at the Controller and since 500 + 500 < 5000, then both requests are processed, and the processes are registered.	`5000`
`sim.processes.query.maxResultLimit`	The maximum number of processes that the UI can show (for example, in the **Server Dashboard >** Processes tab). Suppose the time range is 2 weeks and the Agent reported over 10000 processes during that interval. If `maxResultLimit = 5000`, the UI will show 5000 processes.	`5000`
`sim.processes.registrations.maxPerSecondPerAccount`	Set the maximum number of process requests handled per second per account. For better performance, set to 600 /sec. On a multi-tenant Controller, the total number of Agents per second should not exceed 600/sec.	`60/sec`
`sim.processes.stale.purgeIntervalMillis`	The number of milliseconds between consecutive deletes of stale processes for an account.	`3600000 ms` (1 hour)
`sim.processes.terminated.toStaleTimeoutMillis`	The number of milliseconds before a terminated process is considered stale and can be deleted to make space for new data. You can specify these in the Access the Administration Console as a Controller setting (all accounts) or as an account setting for individual accounts.	`172800000 ms` (2 days)

Machine Agent Hierarchy

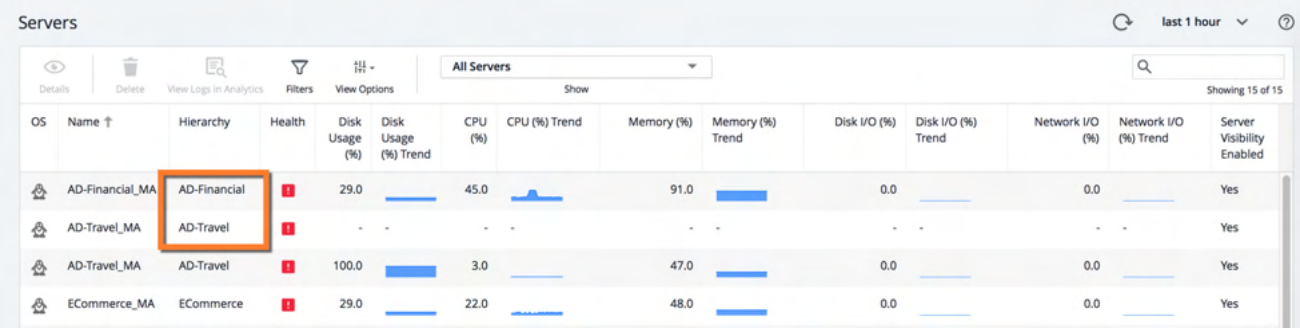
To group servers together so that you can apply health rules to the specific server groups, use the Machine Hierarchy property. This property enables you to group servers into arbitrary hierarchies by specifying a hierarchical path to the server. You can group servers based on departments, geographical locations, such as data centers, or other organizational units. You can then create health rules that apply to these departments.

The server hierarchy displays in the Metric Browser, on the Servers list, and on Server Visibility dashboard.

You need a Server Visibility license to use this feature.

You can specify the Machine Hierarchy property using `controller-info.xml`, a system property, or an environment variable.

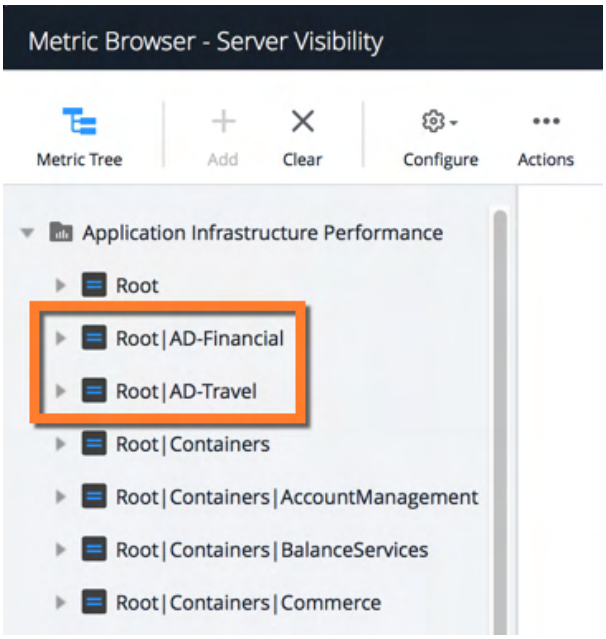
For example, to group servers into geographical locations, perhaps representing different data centers, you could group the following:



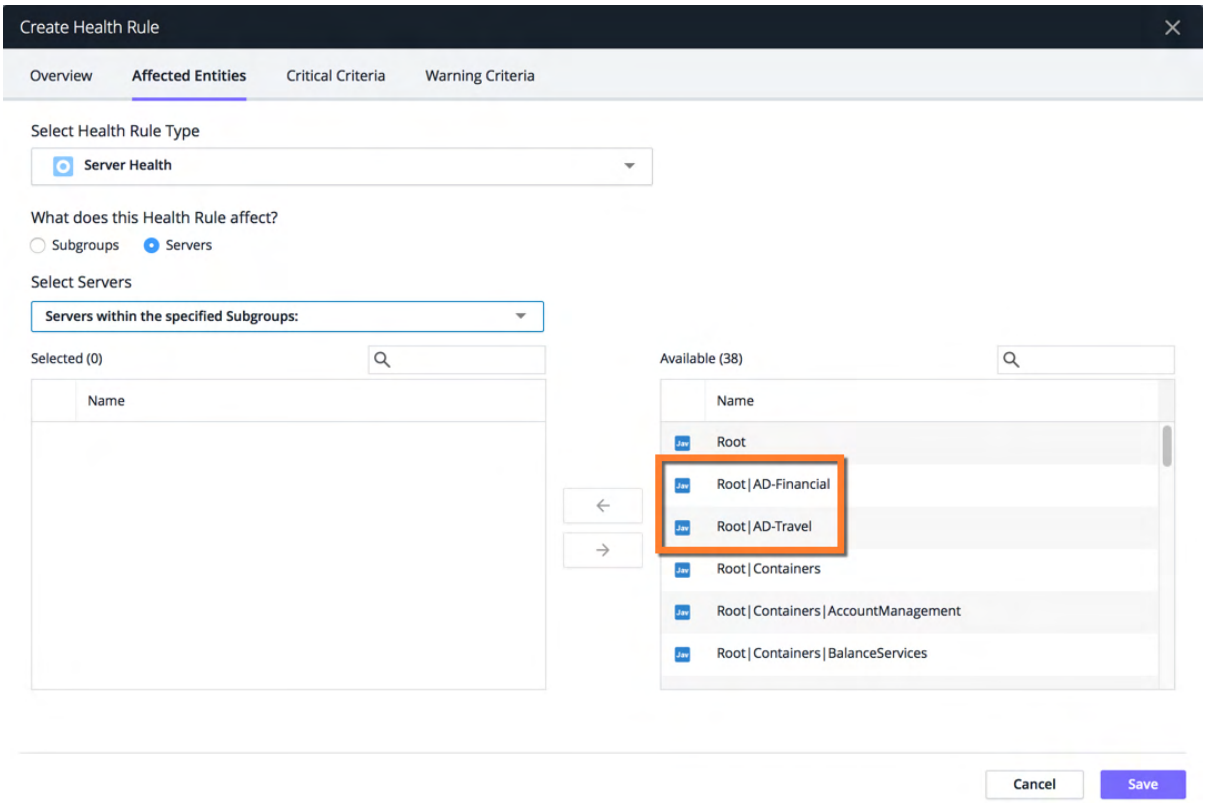
Use the `machine-path` element in the `controller-info.xml` configuration file to specify the tier and node:

```
<machine-path>AD-Financial|Node1</machine-path>
```

The Metric Browser view for this example:



When creating health rules, you can select one or more subgroups from the Affected Entities tab:


Create Health Rule
Overview
Affected Entities
Critical Criteria
Warning Criteria
Select Health Rule Type
Server Health
What does this Health Rule affect?
Subgroups
Servers
Select Servers
Servers within the specified Subgroups:
Selected (0)
Name
Available (38)
Name
Root
Root|AD-Financial
Root|AD-Travel
Root|Containers
Root|Containers|AccountManagement
Root|Containers|BalanceServices
Cancel
Save

Server Tagging

Use Server Tagging to query, filter, and compare related servers using custom metadata. You can tag related servers based on OS, location, tier, owner, or any other relevant criteria. Server tags provide additional context to server metrics.

For example, you may want to specify deployment-version tags to the servers in a specific cluster, and then use those tags to identify anomalies in server metrics during a new deployment.

Server Tagging requires a Server Visibility license. Server Tagging is currently available for Linux, Windows, and Solaris.

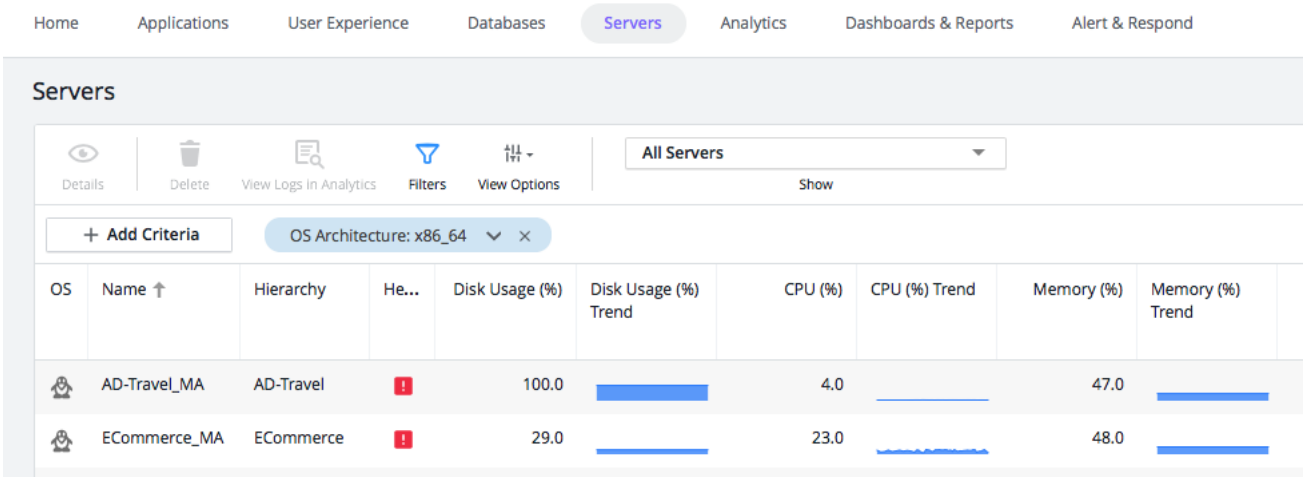
Server Visibility can import the tags (described in the table) automatically.

Source	Tags	Controller and Agent Requirements
Custom	Server Visibility can import: • Custom tags defined in `ServerMonitoringConfig.yml`. See Defining and Viewing Tags for a Server. • Custom tags defined in Amazon EC2 (See Tagging EC2 Resources in the Amazon AWS documentation).	• Controller >= 4.4 • Standalone Machine Agent >= 4.4
Machine Agent	These tags display in the Server Dashboard under Tags: • OS Architecture • OS Kernel • AppDynamics Agent Version	• Controller >= 4.4 • Standalone Machine Agent >= 4.2
Amazon Web Services	These tags display in the Server Dashboard under Tags as `AWS|<tag>`. See Importing Tags from Amazon Web Services. • `resource-id` • `availability-zone` • `region` • `ami-id` • `instance-type` • `security-group`	• Controller >= 4.4 • Standalone Machine Agent >= 4.4
Docker	Server Visibility can import user-defined tags and system container-level tags. These display under **Container Details > Tags** as `Docker|<tag>`. The exact set of tags imported can vary depending on the system on which the container is running.	• Controller >= 4.4.3 • Standalone Machine Agent >= 4.4.3
Kubernetes OpenShift	Server Visibility can import ReplicaSet and pod tags. These appear under **Container Details > Tags** as `K8s|<tag>`.	• Controller >= 4.4.3 • Standalone Machine Agent >= 4.4.3

View Servers by Tag in the Controller UI

To view filter servers by tag in the Servers list, select **Filters** and add the criteria you want for the tag filter. If you specify multiple criteria, the filter performs an AND search (for criteria with different keys) and an OR search (for criteria with the same key). This example shows the tag criteria filters for all servers with:

1. A **Tier** tag that equals **ECommWeb** OR **ECommInventory**, AND
2. An **OS Architecture** tag that equals **x86_64**.



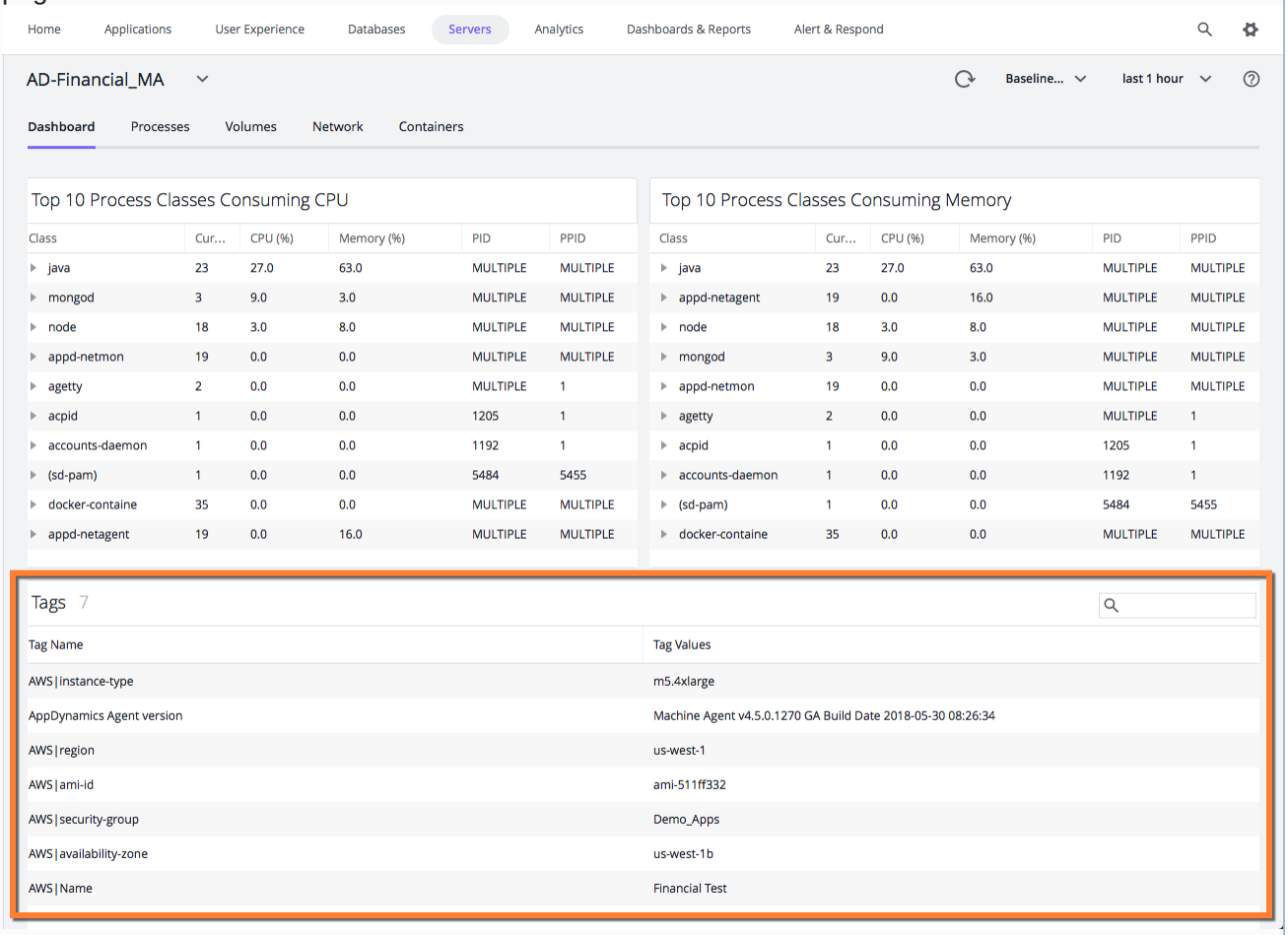
Define and View Tags for a Server

You can specify custom tags as a set of key-value strings in the Agent config directory. Each Agent can support up to 50 tags by default.

Follow these suggested best practices to define server tags:

If an Agent is deployed in Amazon Web Services, the Agent can auto-detect and import tags defined in AWS. The Controller also auto-assigns a set of default tags to each server.

The primary use case for custom tags is to specify information that is not already specified in AWS and auto-assigned tags. Before you specify custom tags in the Agent config directory, you should review the current set of tags. Access the Server Dashboard and view the Tags pane at the bottom of the page:



Be careful not to define duplicate tags in the `YML` file.

Configure the following option in `<machine_agent_home>/extensions/ServerMonitoring/ServerMonitoringConfig.yml`:

Setting	Description

`tag`	A list of user-defined tags for the individual server. You can use these tags to query, filter, aggregate, and compare related servers. Each tag is specified by a key-value pair. You can define tag names up to 127 Unicode characters and tag values up to 255 Unicode characters. Define each key on a separate line. All key-value strings should be within single quotes. If a key has multiple values, delineate the list with commas:
	`<key>: [ <value> ]`
	`tags:`
	`  'Location': ['NYC', 'Data Center', 'Server Room 7']`
	`  'Environment': ['preProduction']`

Import Tags from Amazon Web Services

Detailed information about Amazon Web Services is outside the scope of AppDynamics documentation.

Before Server Visibility can import tags from AWS, you must set up IAM roles:

1. Log in to the AWS console: https://console.aws.amazon.com/iam/
2. Create a role with read access to EC2 tags.
   Specifically, the role must have an AWS Managed Policy with the required permissions (such as, AmazonEC2ReadOnlyAccess) attached.
3. Add this role to your EC2 instance.
   For information about these steps, search for "To create an IAM role using the IAM console" and "Attaching an IAM Role to an Instance" in this page.
4. Run the Machine Agent with Amazon Web Services enabled.

Configuration Options for Server Tagging

To edit these settings, log in to the Controller administration console using the `root` user password. See Access the Administration Console.

Server Visibility Property	Description	Default
`sim.machines.tags.enabled`	Enable or disable server tags for all servers	`True`
`sim.machines.tags.aws.enabled`	Enable or disable the collection of AWS tags for all servers	`True`
`sim.machines.tags.aws.pollingInterval`	Length of time in milliseconds between each polling of AWS tags	21600000 (6 hours)
`sim.machines.tags.maxPerMachine`	Maximum number of unique tags per account	50
`sim.machines.tags.maxPerAccount`	Maximum number of unique tags per account	500

Configuration Options for Docker Tags

You can configure the Machine Agent to collect different types of tags. By default, all tags are collected. To turn off tag collection, set `dockerTagsEnablec` to "`false`".

1. Edit the *<machine_agent_home>*`/extensions/DockerMonitoring/DockerMonitoringConfig.yml` file.
2. Under the `containerMonitoringConfig` section, set `dockerTagsEnabled: "false"`

```
# WARNING: Before making any changes to this file read the following section carefully
#
# After editing the file, make sure the file follows the yml syntax. Common issues include
# - Using tabs instead of spaces
# - File encoding should be UTF-8
#
# The safest way to edit this file is to copy paste the examples provided and make the
# necessary changes using a plain text editor instead of a WYSIWYG editor.
#

# samplingInterval indicates how often to gather metric data. Units in milliseconds.
samplingInterval: 30000

containerMonitoringConfig:
    # containerProcessSelectorRegex defines regular expression to evaluate the processes in
    # each running container to be monitored by the machine agent. The regular expression is
    # compared against each process full command line within running
    # If the pattern matches, then the machine agent start monitoring it.
    containerProcessSelectorRegex: ".*[ ]-Dappdynamics.*"
    dockerTagsEnabled: "false"
```

Configuration Options for AWS Tags

You can configure the Machine Agent to collect different types of tags. By default, all tags are collected. To turn off tag collection, set `awsTagsEnabled` to "`false`".

1. Edit the *<machine_agent_home>*`/extensions/ServerMonitoring/conf/ServerMonitoring.yml` file.
2. Set `awsTagsEnabled: "false"`

```
//ServerMonitoring.yml
# WARNING: Before making any changes to this file read the following section carefully
#
# After editing the file, make sure the file follows the yml syntax. Common issues include
# - Using tabs instead of spaces
# - File encoding should be UTF-8
#
# The safest way to edit this file is to copy paste the examples provided and make the
# necessary changes using a plain text editor instead of a WYSIWYG editor.
#
awsTagsEnabled: "false"
# samplingInterval indicates how often to gather metric data. Units in milliseconds.
samplingInterval: 30000
...
```

Configuration Options for Kubernetes and OpenShift Tags

You can configure the Machine Agent to collect different types of tags. By default, all tags are collected. To turn off tag collection, set `k8sTagsEnabled` to "`false`".

1. Edit the <*machine_agent_home*>`/extensions/ServerMonitoring/conf/ServerMonitoring.yml` file.
2. Set `k8sTagsEnabled: "false"`

```
//ServerMonitoring.yml
# WARNING: Before making any changes to this file read the following section carefully
#
# After editing the file, make sure the file follows the yml syntax. Common issues include
# - Using tabs instead of spaces
# - File encoding should be UTF-8
#
# The safest way to edit this file is to copy paste the examples provided and make the
# necessary changes using a plain text editor instead of a WYSIWYG editor.
#
k8sTagsEnabled: "false"
# samplingInterval indicates how often to gather metric data. Units in milliseconds.
samplingInterval: 30000
...
```

Tagged Metrics

Beta Disclaimer

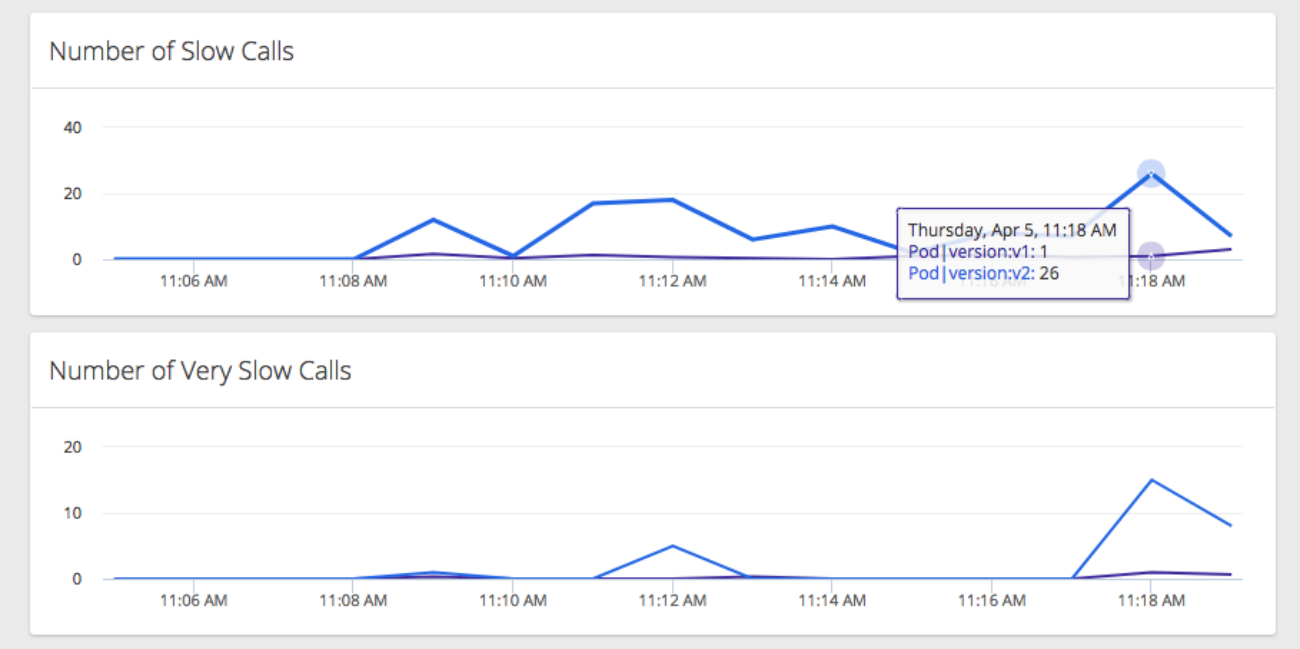
This documentation mentions features that are currently beta. AppDynamics reserves the right to change the features at any time before making them generally available as well as never making them generally available. Any buying decisions should be made based on features and products that are currently generally available.

Tagged Metrics

The **Tagged Metrics** panel enables you to aggregate all nodes in a tier based on a server tag (user-defined, or imported from AWS or Kubernetes) and then compare performance metrics between the node groups in one set of charts. For example, you can quickly compare the performance on all "blue nodes" and "green nodes" during a canary deployment.

1. In the Controller left-side panel, select **More > Tagged Metrics**.
2. In the **Tagged Metrics** panel, select the server tag and the tier of interest.
3. Compare the response times, call rates and transaction outliers (errors, slow calls, very slow calls) between the tag-based node groups.

The following chart shows how you can easily compare performance between different sets of nodes. For example, you want to analyze node performance for a tier with 26 nodes running v1 of the software, and one node running v2. You select the "version" tag, which represents a Kubernetes label. The charts show very clearly that the rate of slow and very slow calls has increased dramatically on the v2 node.



This feature is disabled by default.

To enable Tagged Metrics:

1. Log in to the administration console: `http:<controller-hostname>:8080/controller/admin.jsp` or `https:<controller-hostname>:443/controller/admin.jsp`
2. In Controller Settings, search for "`tagged.metrics.enabled`" and set this field to `true`.
3. In Account Settings:
    a. Select the account to enable.
    b. Under **Account > Additional Properties**, select **Add Property**, add the "`tagged.metrics.enabled`" property, and set it to `true`.

Network Visibility

Deployment Support

With Network Visibility, you can:

- Answer the key question quickly and easily: *Is the network to blame for any performance issues in my application?*
- Identify the root cause of performance issues in the application, the network, or in application/network interactions (for example, how an application or server utilizes the network).
- Pinpoint the traffic flows, individual nodes, and transport connections where network or application/network issues are occurring.
- Collect detailed metrics that show how the network is performing, and how well your application uses network connections and resources.
- Collect and report targeted troubleshooting information to network, IT, DevOps, and other teams.

Installation and Administration on Linux

- Set Up Network Visibility on Linux
- Set Up the Network and App Agents on Linux
- Administer Network Agents on Linux

Installation and Administration on Windows

- Set Up Network Visibility on Windows
- Install the App Agent on Windows
- Administer Network Agents on Windows

Using Network Visibility

- Managing Network Agents in the Controller
- Monitor Network Visibility Metrics
- FAQs for Network Visibility
- Advanced Operations

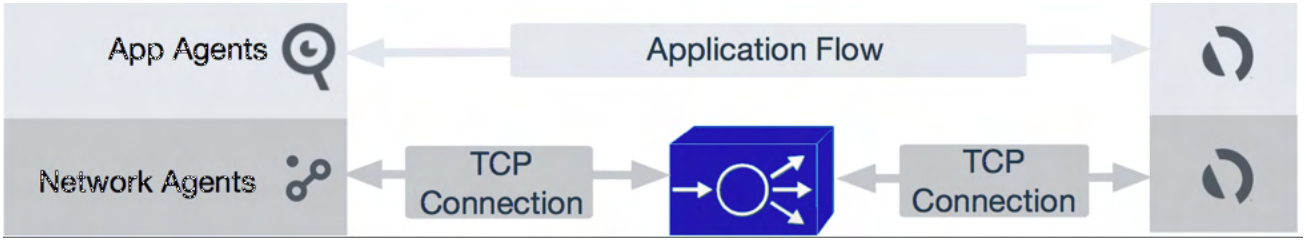
Reference

- Network Visibility Concepts
- Workflows and Example Use Cases
- User Interface
- Network Visibility Events

Network Visibility Overview

Application Intelligence and Network Visibility

Network Visibility extends the application intelligence of AppDynamics APM down the stack from the application to the network. With "app-only" visibility, it can be easy to mistakenly blame (or not blame) the network when an application issue arises. Network Visibility can help reduce or eliminate the guesswork involved in identifying root causes. Network Agents and App Agents, working together, automate the work of mapping TCP connections to the application flows that use them. Network Agents can identify intermediate load balancers (which often split TCP connections) and correlate the connections on either side of these devices.



The Agent-based approach of Network Visibility provides these advantages over standard approaches to network monitoring:

- More cost-efficient than using network monitoring appliances, which often view traffic from a few central locations
- Especially useful for distributed environments and multi-tier applications that span multiple network segments
- Works in cloud and hybrid networks, unlike most network-monitoring solutions

Drill Down to the Root Cause

If network issues are affecting your application, Network Visibility can help you determine the cause.

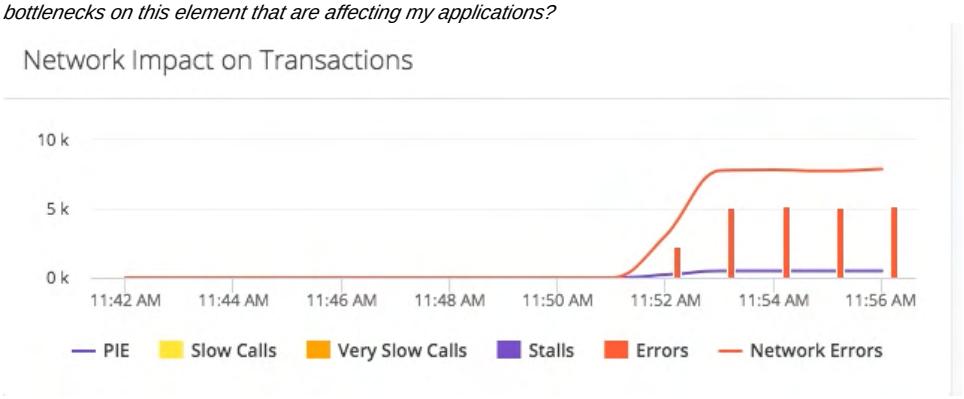
- You see a spike in transaction outliers in the Application Dashboard. Are network issues to blame?
- Switch over to the Network Dashboard. Each tier, node, and link shows network KPIs (Key Performance Indicators) that measure the network health of that element. Use baselining to highlight network elements with KPIs outside the baseline.
- To diagnose a tier, node, or network link, right-click and select **View Metrics**. In the right-click dashboard, look for network metrics with spikes that correlate with the spikes in your transaction outliers. This is often provides direction to the network root cause.
- If a network element requires more in-depth troubleshooting, configure the relevant Network Agents to collect metrics on the individual Connections used by that element. You can then
    - Click a node or link and view KPIs for the Connections used by the relevant nodes.
    - Right-click a Connection and view detailed metrics in a right-click dashboard or in the Metric Browser.

Network Visibility Metrics

Network Visibility collects and displays these metric types:

- Network KPIs (Key Performance Indicators) provide high-level, at-a-glance measures of whether the network is affecting the performance of the monitored application. The Network Flow Map shows KPIs for each tier, node, and link.
- The PIE (Performance Impacting Events) metric enables you to see immediately if there are any such events on a connected client, server, or network link.

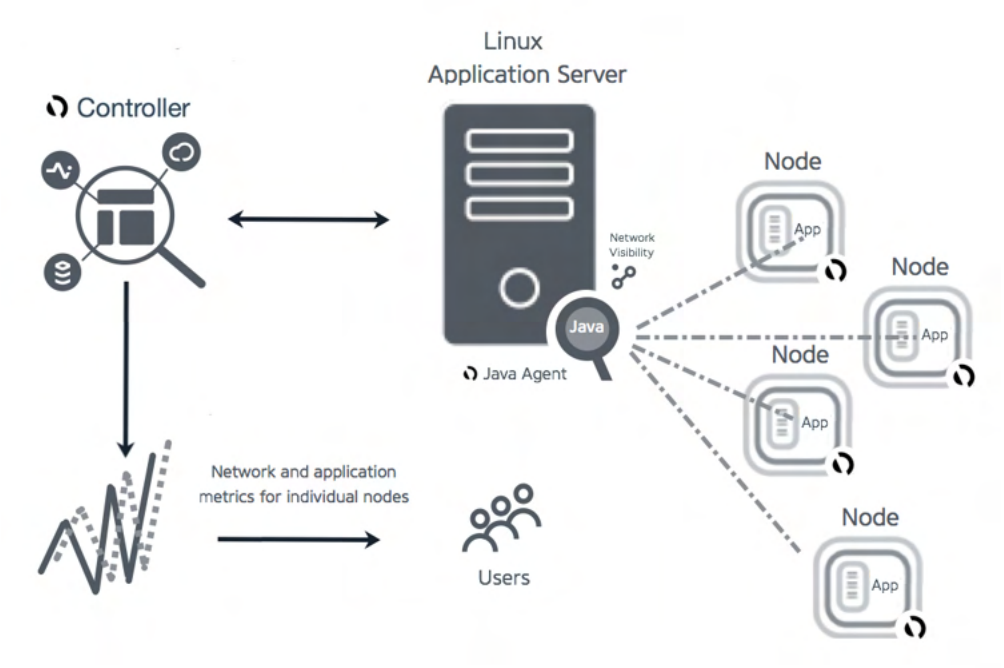
- If the KPI metrics indicate an issue with a specific element, you can view additional metrics for that element to identify root causes. Right-click an element and select **View Metrics**. The metrics and charts in the right-click dashboard are all designed to answer the question: *Are there any bottlenecks on this element that are affecting my applications?*



- To perform in-depth analysis, you can view detailed TCP Flow metrics in the Metric Browser.
- You can view node metrics to evaluate the health of TCP sockets and network interfaces.

Network Visibility for Multiple App Nodes on the Same Server

Network Agents can monitor multiple nodes that are associated with the same IP address because they run on the same physical or virtual server. The Agent monitors each node individually and calculates network metrics for each node. These metrics are based on the ingress/egress traffic for each individual node, not aggregate traffic for the IP address of the host on which the node is running.



Network Visibility Supported Environments

Related pages:

- License Management
- Java Supported Environments
- App Server Agent Supported Environments

This page describes application environments, operating systems, and version support for the Network Agent.

General Requirements

With a Network Visibility license, you can use Network Visibility features on Linux and Windows. You must have a supported version of the Network Agent to be installed on the monitored server. See License Entitlements and Restrictions.

By default, Network Visibility is disabled in the Controller. To enable Network Visibility, set the agent node property `socket-enabled=true`. See App Agent Node Properties (S).

Network Visibility is not supported on JDK 1.6. To run the Network Agent, upgrade to JDK >= 1.8.

Network Visibility is compatible with Kubernetes and Docker, but not Docker Swarm.

Network Visibility on Linux

Operating System Requirements

Network Visibility is supported on Linux hosts with Java app server agents only.

Each host on Linux must have:

- Network Visibility Agent >= 4.4
- Java Agent >= 4.4 with an App Agent license
- Controller >= 4.4 with a Network Visibility license

You must have `sudo` or `root` access permissions on the Agent host to install the Network Agent. You do not need `sudo` or `root` access to run the agent.

Distributions

The following Linux distributions support the Network Agent based on glibc >= 2.12:

- CentOS >= 6 (32-bit and 64-bit)
- Ubuntu >= 14 (32-bit and 64-bit)
- Red Hat Enterprise Linux >= 6 (32-bit and 64-bit)
- Fedora >= 24 (32-bit and 64-bit)

You must pre-install the following libraries for the Network Agent host:

- libdl.so.2
- libpthread.so.0
- librt.so.1
- libm.so.6
- libc.so.6
- libncurses.so.5
- libtinfo.so.5
- libstdc++.so.6
- libgcc_s.so.1

Network Visibility on Windows

Operating System Requirements

Each Windows host must have:

- Network Visibility Agent >= 4.5.7.
- An account with administrative privileges on the Windows machine where you want to install the Network Agent.

Additionally, each Windows host must also have:

- .NET Agent >= 4.5.15 with an App Agent License. See Install the .NET Agent and License Management.
- Controller >= 4.5.14 compatible with the .NET Agent.

OR

- Java Agent >= 4.5.4.24386 with an App Agent license. See Install the Java Agent and License Management.
- Controller >= 4.5.4.24386 compatible with the Java Agent. See Agent and Controller Compatibility.

> To install AppDynamics for testing, you need to verify system requirements, prepare the host, and perform the Controller installation. See Upgrade the Controller Using the Enterprise Console and Install the Controller Using the CLI.

Distributions

The following Windows distributions support the Network Agent on Java applications:

- Windows Server 2019 (64-bit)
- Windows Server 2016 (64-bit)
- Windows Server 2012 (64-bit)
- Windows Server 2012 R2 (64-bit)

The following Windows runtime environments support the Network Agent on .NET applications (starting with .NET Framework version 3.5):

- Microsoft IIS versions 6.0, 7.0, 7.5, 8.0, 8.5, 10
- Managed Windows Services
- Managed Standalone Applications

You can use the Network Visibility features with the .NET Agent if you have installed:

- Network Agent >= 4.5.7
- .NET Agent >= 4.5.15
- AppDynamics Controller>= 4.5.14

You can monitor .NET applications that are implemented using synchronous calls. However, Directory Search and ADO.NET are not supported.

> Support for monitoring network traffic for asynchronous .NET calls is limited. If your asynchronous .NET calls are not reporting network traffic correctly, contact AppDynamics Support to report your use case and include the libraries, API, and framework involved in the transaction. This information may be shared with the Product team for future enhancements.

Limitations

The following **Network Flow Map** features are not supported:

- Federated Flow Maps
- Visualization of flows between web servers and APM entry tiers

The Network Flow Map does not filter out connections on the selected time range.

Cross-application correlation of a .NET application with Network Visibility is not supported for .NET environments.

The Network Agent cannot monitor multiple app server agents on the same host if these agents report to different Controllers. All app server agents must report to the same Controller.

The **Network Dashboard** does not show data for Health Rules or Health Rule violations. To view any Health Rule violations, including those based on Network Visibility data, access the **Application Dashboard**.

Potential Issues

Network Visibility cannot monitor cross application flows that use Jersey web servlets. As a workaround, disable Jersey servlet instrumentation and business transactions on the flows will be recognized as servlets instead of web services:

1. Select the application of interest in the Controller UI.
2. Select **Configuration > Instrumentation > Transaction Detection**.
3. Disable instrumentation for Jersey Servlet and Jersey 2.x Servlet.

If you open the **Network Flow Map** for an individual node, the KPI metrics for `node-to-load-balancer` and `node-to-TCP-endpoint` links show network KPIs for all nodes in the parent tier (instead of KPIs for the individual node only). To view KPIs for the individual node, open the link popup and review the Connection KPIs.

Set Up Network Visibility on Linux

To set up Network Visibility on Linux, see:

1. Network Visibility Supported Environments
2. Install the Network Agent on Linux
3. Set Up the Network and App Agents on Linux
4. Administer the Network Agent on Linux

Install the Network Agent on Linux

Install the Network Agent Independent of Standalone Machine Agent

You can install and run the Network Agent independently of the Standalone Machine Agent using the ZIP, RPM, or DEB installers.

ZIP Network Agent Install

Important Notes

- If you are using Network Visibility to monitor applications running in Docker containers, you must install the Network Agent in a container as well. See Docker and Network Visibility.
- To run the install script, log in as `root`.
- For all other operations, log in as the designated agent administrator (<*appd-agent-administrator*>) for that host.

1. Check System Requirements. See Network Visibility Supported Environments.
2. Download and unzip the Agent installer:

   Login User = Agent Administrator

   a. Log in to the Agent host as `<appd-agent-administrator>`.
   b. Access the AppDynamics Downloads Portal, download the Network Agent installer, and unzip it.
   c. In a terminal window, `cd` to the folder where you downloaded the Network Agent ZIP package.
   d. Enter the following command, where <*network_agent_home*> is the folder where you want to install the agent software:
   `unzip <installer.zip> -d <network_agent_home>`

   The parent path should not include any directory names with spaces:

   `/opt/appdynamics/networkVisibility` (correct)

   `/opt/appdynamics/network visibility` (incorrect)

3. Install the Agent:

   Login User = root

   a. Log out, and then log in as `root`.
   b. Go to <*network_agent_home*> and run the install script:
   `cd <network-agent-home>`
   `./install.sh`

4. Start and verify the Agent:

   Login User = Agent Administrator

   a. Log out, and then log in as *appd-agent-administrator*.
   b. Open a CLI window and enter these commands:
      i. <*network-agent-home*>`/bin/start.sh` (starts the Agent)
      ii. <*network-agent-home*>`/bin/appd-netviz.sh status` (verifies that the Agent is running and shows PID of the Agent process)

5. If you experience any problems or issues, see FAQs for Network Visibility.
6. If you are installing the Network Agent as part of a Set Up Network Visibility on Linux workflow, proceed to Set Up the Network and App Agents on Linux.

RPM Network Agent Install

Important Note

If you are using Network Visibility to monitor applications running in Docker containers, you must install the Network Agent in a container as well. See Docker and Network Visibility.

1. Check System Requirements. See Network Visibility Supported Environments.
2. If you are logged in to the Agent host as a non-root user, log out and then log back in as `root`.
3. Access the AppDynamics Downloads Portal and download the Network Agent installer.

4. Open a terminal window and enter these commands:
    a. `sudo rpm -ivh <installer.rpm>` (installs the Agent under *opt/appdynamics/netviz*)
    b. `sudo service appd-netviz start` (starts the Agent)
    c. `sudo service appd-netviz status` (verifies that the Agent is running)
5. If you experience any problems or issues, see FAQs for Network Visibility.
6. If you are installing the Network Agent as part of a Set Up Network Visibility on Linux workflow, proceed to Set Up the Network and App Agents on Linux.

DEB Network Agent Install

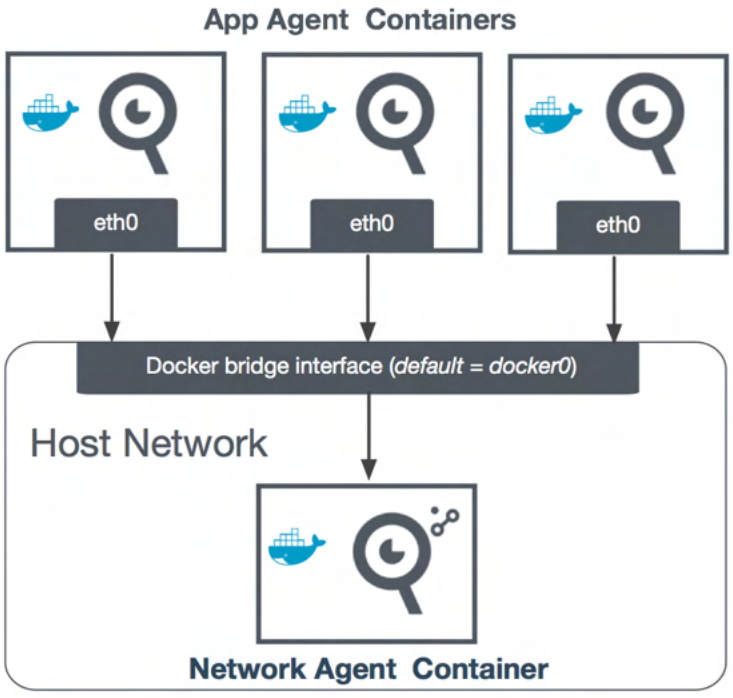
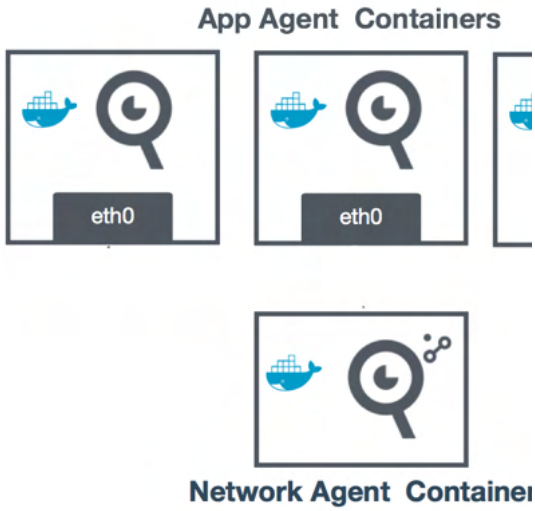
> **Important Note**
>
> If you are using Network Visibility to monitor applications running in Docker containers, you must install the Network Agent in a container as well. See + Docker and Network Visibility.

1. Check System Requirements. See Network Visibility Supported Environments.
2. If you are logged in to the Agent host as a non-root user, log out and then log back in as `root`.
3. Access the AppDynamics Downloads Portal and download the Network Agent installer.
4. Open a terminal window and enter the following commands:
    a. `sudo dpkg -i <installer.deb>` (installs the Agent under *opt/appdynamics/netviz*)
    (starts the agent, outputs *" appd-netviz start/running. process <pid #>"*)
    b. `sudo service appd-netviz status` (verifies the Agent status)
5. If the Agent does not start, enter: `sudo service appd-netviz start`.
6. If you experience any problems or issues, see FAQs for Network Visibility.
7. If you are installing the Network Agent as part of a Set Up Network Visibility on Linux workflow, proceed to Set Up the Network and App Agents on Linux.

Install the Network Agent in a Docker Container

The Network Agent can monitor applications running in Docker containers on the same host. You can deploy one Network Agent per host and collect individual metrics for every local application monitored by a Java App Agent in a container.

AppDynamics recommends that you install the Network Agent in a separate container in Docker Host network mode on the same host as the application containers. This recommendation applies when the App Agent containers are running in Docker Bridge or Docker Host mode.

Example 1: App Agents in Separate Network	Example 2: App Agents in Host Networ
	

Install the Network Agent and App Agents

1. Paste the sample Dockerfile into a text file and save the file. This sample Dockerfile contains commands for installing the Network Agent.

```
FROM centos:centos7

RUN yum update -y && yum install -y \
  net-tools \
  iproute \
  tcpdump \
  curl  \
  unzip   \
  sysvinit-tools \
  openssh-clients

WORKDIR /netviz-agent

ARG NETVIZ_ZIP_PKG

# copy NetViz agent contents
ADD ${NETVIZ_ZIP_PKG} .

# run the agent install script
RUN unzip *.zip && ./install.sh \
    && sed -i -e "s|enable_netlib = 1|enable_netlib = 0|g" ./conf/agent_config.lua \
    && sed -i -e "s|WEBSERVICE_IP=.*|WEBSERVICE_IP=\"0.0.0.0\"|g" ./conf/agent_config.lua

# default command to run for the agent
CMD  ./bin/appd-netagent -c ./conf -l ./logs -r ./run
```

2. Navigate to the directory where you saved the Dockerfile. Build the Docker image by entering:

```
docker build --build-arg NETVIZ_ZIP_PKG=/path/to/netviz-agent-pkg.zip -t appd-netviz .
```

3. Push the Docker image to your Docker Trusted Registry.
4. Deploy your Docker image using this sample Docker Compose file:

```
version: '2'
services:
  appd-netviz-agent:
    image: path/to/your/docker/image # docker registry image
    network_mode: "host"
    restart: unless-stopped
    ports:
      - '3892:3892'
    cap_add:
      - NET_ADMIN
      - NET_RAW
```

Alternatively, enter:

```
docker run -d --network=host --cap-add=NET_ADMIN --cap-add=NET_RAW path/to/your/netviz/docker/image
```

5. Set up the Network Agent. Follow the single-tenant setup instructions on Set Up the Network and App Agents on Linux.

> For single-tenant setup, you do not need to enable netlib.

Specify the Docker Host Gateway IP on the App Agent

If you are running the App Agent in a Docker container outside of the Docker Host network (as shown in Example 1 in the diagram), complete these steps.

1. In the host for your App Agent, enter these environment variables:

```
export APPDYNAMICS_NETVIZ_AGENT_HOST=$(ip -4 addr show docker0 | grep -Po 'inet \K[\d.]+')
export APPDYNAMICS_NETVIZ_AGENT_PORT=3892
```

2. Set the values for the port and host address.
    a. If you are using an App Agent version 4.5.1 or earlier:
    Open the `<app-agent-install-dir>/<version-number>/external-services/netviz/netviz-service.properties` file and set these properties:
        - `netviz.agent.host.address`
        - `netviz.agent.api.service.port`
    b. If you are using an App Agent version 4.5.2 or later:
    Run the application Docker image by passing the AppDynamics environment variables which were exported in Step 1. For example:

```
docker run --env "APPDYNAMICS_NETVIZ_AGENT_HOST=$APPDYNAMICS_NETVIZ_AGENT_HOST"
--env "APPDYNAMICS_NETVIZ_AGENT_PORT=$APPDYNAMICS_NETVIZ_AGENT_PORT" <application_docker_image>
```

Install the Network Agent as an Extension of the Machine Agent (Bundled)

You can install and run the Network Agent as an extension of the Standalone Machine Agent (ZIP only) if you want to install both Agents at the same time. This option is available on Linux 32-bit and 64-bit ZIP installers >= 4.4.2, only.

Important Notes

- If you are using Network Visibility to monitor applications running in Docker containers, you must install the Network Agent in a container as well. See + Docker and Network Visibility.
- To run the install script, log in as `root`.
- For all other operations, log in as the designated agent administrator `<appd-agent-administrator>` for that host.

1. Check System Requirements. See Network Visibility Supported Environments.
2. Download and unzip the Standalone Machine Agent:

Login User = Agent Administrator

    a. Log in to the Agent host as *<appd-agent-administrator>*.
    b. Access the AppDynamics Downloads Portal, download the Standalone Machine Agent installer, and unzip it.
    c. Install the Standalone Machine Agent using the Linux ZIP installer as described in the Linux Install Using ZIP with Bundled JRE page.

3. Install the Standalone Machine Agent and Network Visibility extension:

Login User = root

    a. Log out and then log in as `root`.
    b. Open a CLI window and enter:
        i. `cd <standalone-machine-agent-home>/extensions/NetVizExtension`
        ii. `./install-extension.sh`

4. Enable the Network Visibility extension and start the Agent:

Login User = Agent Administrator

    a. Log out and then log in as *<appd-agent-administrator>*.
    b. Enter the CLI command:
        i. `<machine-agent-home>/bin/machine-agent -d -p <machine-agent-home>/pidfile` (starts the Machine Agent)

5. If you experience any problems or issues, see FAQs for Network Visibility.
6. If you are installing the Network Agent as part of a Set Up Network Visibility on Linux workflow, proceed to Set Up the Network and App Agents on Linux.

Set Up the Network and App Agents on Linux

Network Agent 4.5.7 and Java Agent 4.5.11 eliminate the need to separately configure the Network Agent in a multi-tenant setup when running multiple nodes on the same machine. If you use the Network Agent 4.5.7 with Java Agent <= 4.5.10, you need to configure the Network Agent in a multi-tenant setup. See Network Agent and Java Agent Installation Instructions.

Network Agent and Java Agent Installation Instructions

If you are using	then
Network Agent >= 4.5.7 and Java Agent >= 4.5.11	Install the Network Agent.
Network Agent >= 4.5.7 and Java Agent <= 4.5.10	Install the Network Agent in a multi-tenant setup or a single tenant setup. You do not need to manually enable the `netlib` parameter during the multi-tenant setup, it is enabled by default.
Network Agent <= 4.5.6	Install the Network Agent in a multi-tenant setup or a single tenant setup. You need to manually enable the `netlib` parameter during the multi-tenant setup.
Java App Agent <= 4.4	Update the Java Agent.

Install the Network Agent

If you are using Java Agent >= 4.5.13, you do not need to manually enable the socket collection. It is enabled automatically. You can skip Step 1 and proceed to Step 2.

Enable Socket Collection on the App Agent

1. Configure the Java App Agent startup command to perform these tasks, as shown in the example:
   - Enable `appdynamics.socket.collection.bci.enable` for the Network Agent to map network metrics to application flows. For example:
     ```
     java -javaagent:<app_server_agent_home>/javaagent.jar \
     -Dappdynamics.socket.collection.bci.enable=true
     ```
2. The following examples illustrate how editing the Agent startup command can differ, depending on the framework of the monitored application. See Agent Installation by Java Framework and its relevant framework documentation.

Framework	Example (App Agent ZIP)
IBM WebSphere and InfoSphere	In the WebSphere/InfoSphere UI, navigate to the Java Virtual Machine properties page for the monitored server. Then, add `javaagent` and `appdynamics.socket.collection.bci.enable=true` as a Generic JVM argument. For example: `javaagent:/usr/appd/agents/apm/javaagent.jar -Dappdynamics.socket.collection.bci.enable=true`
Apache Tomcat	Add the full command as a Catalina environment variable in file `CATALINA_BASE/bin/setenv.sh` (Tomcat 6 and later). For example: `export CATALINA_OPTS="$CATALINA_OPTS -javaagent:/usr/appd/agents/apm/javaagent.jar -Dappdynamics.socket.collection.bci.enable=true"`

Glassfish	Add the full command as a `-javaagent` JVM option in the GlassFish domain. You can add the Agent using the asadmin tool. For example:
	
	`export LD_PRELOAD=/usr/appd/agents/netviz/lib/appd-netlib.so`
	
	`glassfish4\bin\asadmin create-jvm-options "-`
	
	`javaagent\:/usr/appd/agents/apm/javaagent.jar:-Dappdynamics.socket.collection.bci.enable=true"`

JBoss Standalone	Add the full command to the `standalone.sh` file. For example: `export JAVA_OPTS="$JAVA_OPTS -javaagent:/usr/appd/agents/apm/javaagent.jar -Dappdynamics.socket.collection.bci.enable=true"`
Jetty	Add the full command to the `start.ini` config file in the Jetty base directory. For example: `--exec -javaagent:/usr/appd/agents/apm/javaagent.jar -Dappdynamics.socket.collection.bci.enable=true`

3. To verify that the App Agent is running correctly, enter:
   `strings /proc/<app_pid>/environ`

Specify the Docker Host Gateway IP on the App Agent

These steps are required only if the App Agent is running in a Docker container outside of the Docker Host network.

1. Open the following file: `<app-agent-install-dir>/<version-number>/external-services/netviz/netviz-service.properties`
   (You might need to unzip `external-services/netviz.zip` to access this file.)
2. For the `netviz.agent.host.address`, specify the Gateway IP of the Docker Host network interface.
   To determine this IP, enter this command on the Docker host:
   `docker inspect <app_container_name> | grep Gateway`
3. Restart the App Agent.
4. To determine if the Docker Host network is connected, enter this command from within the container:
   `curl -k -v <host-network-IP>:3892`
   If you receive a response, even if it is `404 Not Found`, then the network is connected.

Restart the App (*IBM JVM Only*)

If the monitored app is running in an IBM JVM, you must restart the app for the Network Agent to detect and monitor any persistent connections.

Multi-Tenant Setup (*Multiple Apps on the Agent Host*)

If you want the Network Agent to monitor multiple app nodes on the same host, you perform the following steps *for each App Agent*.

If you are using Network Agent <= 4.5.6, enable the `netlib` parameter manually as described in Step 1.

Enable Netlib on the Network Agent

1. Open the following file in a text editor: `<network_agent_home>/conf/agent_config.lua`
2. Edit the `enable_netlib` parameter:

```
npm_config = {
      log_destination = "file",
      log_file = "agent.log",
      debug_log_file = "agent-debug.log",
      disable_filter = 1,
      mode = "Advanced",
      enable_netlib = 1,
      lua_scripts_path = ROOT_DIR .. "/scripts/netagent/lua/",
}
```

Install or Update the App Agent (if needed)

The Network Agent requires Java App Server Agent >= 4.4. Update each App Agent that is running an earlier version. See Install the Java Agent

Extract the NetViz External Services Folder on the App Agent

1. Navigate to the folder: `<app-agent-install-dir>/<version-number>/external-services`

2. If not already extracted, extract the `netviz.zip` archive so that the ZIP contents are under this folder:
   `<app-agent-install-dir>/<version-number>/external-services`

Preload NetLib and Enable Socket Collection on the App Agent

1. You must configure the Java App Agent startup command to perform two additional tasks, as shown in the following:
   - Preload the `appd-netlib` library so the Network Agent can collect network metrics for each individual node (rather than for the entire host)
   - Enable `appdynamics.socket.collection.bci.enable` for the Network Agent to map network metrics to application flows
     For example:
     ```
     LD_PRELOAD=/<network-agent-home>/lib/appd-netlib.so \
     java -javaagent:<app_server_agent_home>/javaagent.jar \
     -Dappdynamics.socket.collection.bci.enable=true
     ```

The following examples illustrate how editing the Agent startup command can differ, depending on the framework of the monitored application. See Agent Installation by Java Framework and its relevant framework documentation.

Framework	Example (App Agent ZIP)
IBM WebSphere and InfoSphere	In the WebSphere/InfoSphere UI, navigate to the Java Virtual Machine properties page for the monitored server. Then add `javaagent` and `appdynamics.socket.collection.bci.enable=true` as a Generic JVM argument. For example: `javaagent:/usr/appd/agents/apm/javaagent.jar -Dappdynamics.socket.collection.bci.enable=true` You can add `LD_PRELOAD` as a JVM environment variable. This example describes how to add an environment variable on a Cognos server using IBM Connections 6.0.0: a. Log in to the WebSphere Application Server administrative console of the Cognos server. b. Click **Servers** > **Server Types** > **WebSphere application servers**. c. Click the `cognos_server` link. d. Click **JAVA** > **Process Management** > **Process definition** > **Environment Entries** > **New** to add this entry: `LD_PRELOAD=/usr/appd/agents/netviz/lib/appd-netlib.so` 1. In the WebSphere/InfoSphere UI, navigate to the Java Virtual Machine properties page for the monitored server. 2. Add `javaagent` and `appdynamics.socket.collection.bci.enable=true` as a Generic JVM argument. For example: `javaagent:/usr/appd/agents/apm/javaagent.jar -Dappdynamics.socket.collection.bci.enable=true` You can add `LD_PRELOAD` as a system environment variable. This example describes how to add an environment variable on a Cognos server using IBM Connections 6.0.0: 1. Log in to the WebSphere Application Server administrative console of the Cognos server. 2. Go to **Servers** > **Server Types** > **WebSphere Application Servers**. 3. Click `cognos_server`. 4. Go to **Java** > **Process Management** > **Process Definition** > **Environment Entries** > **New** to add this entry: `LD_PRELOAD=/usr/appd/agents/netviz/lib/appd-netlib.so`

Apache Tomcat	Add the full command as a Catalina environment variable in file `CATALINA_BASE/bin/setenv.sh` (Tomcat 6 and later). For example:
	`export LD_PRELOAD=/<network-agent-home-directory>/lib/appd-netlib.so`
	`export CATALINA_OPTS="$CATALINA_OPTS -javaagent:/<java-agent-home-directory>/javaagent.jar -Dappdynamics.socket.collection.bci.enable=true"`

Glassfish	Add the full command as a `-javaagent` JVM option in the GlassFish domain. You can add the Agent using the asadmin tool. For example:
	`export LD_PRELOAD=/<network-agent-home-directory>/lib/appd-netlib.so`
	`glassfish4\bin\asadmin create-jvm-options`
	`"-javaagent\:/<java-agent-home-directory>/javaagent.jar:-Dappdynamics.socket.collection.bci.enable=true"`

JBoss Standalone	Add the full command to the `standalone.sh` file. For example:
	`export LD_PRELOAD=/<network-agent-home-directory>/lib/appd-netlib.so`
	`JAVA_OPTS="$JAVA_OPTS -`
	`javaagent:/<java-agent-home-directory>/javaagent.jar -Dappdynamics.socket.collection.bci.enable=true"`

Jetty	For a standalone Jetty using `start.jar`:
	
	a. Add the full command to the `start.ini` config file in the Jetty base directory. For example:
	`exec javaagent:/usr/appd/agents/apm/javaagent.jar -Dappdynamics.socket.collection.bci.enable=true`
	b. Create a script to start Jetty. For example, `startJetty.sh`:
	
	`export LD_PRELOAD=/usr/appdynamics/agents/netviz/lib/appd-netlib.so java -jar start.jar`
	
	For service with embedded Jetty:
	
	a. Create a script to start Jetty. For example, `startJetty.sh`:
	`export LD_PRELOAD=/<network-agent-home-directory>/lib/appd-netlib.so java -javaagent:/<java-agent-home-directory>/javaagent.jar -Dappdynamics.socket.collection.bci.enable=true -jar <embedded_jetty_jar>`

To verify that the App Agent is running correctly, enter:
`strings /proc/<app_pid>/environ`

Specify the Docker Host Gateway IP on the App Agent

These steps are required only if the App Agent is running in a Docker container outside of the Docker Host network.

1. Open the file: `<app-agent-install-dir>/<version-number>/external-services/netviz/netviz-service.properties`
   (You might need to unzip `external-services/netviz.zip` to access this file.)
2. For the `netviz.agent.host.address`, specify the Gateway IP of the Docker Host network interface.
   To determine this IP, run this command on the Docker host:
   `docker inspect <app_container_name> | grep Gateway`
3. Restart the App Agent.
4. To determine if the Docker Host network is connected, run this command from within the container:
   `curl -k -v <host-network-IP>:3892`
   If you receive a response, even if it is `404 Not Found`, then the network is connected.

Restart the App (*IBM JVM Only*)

If the monitored app is running in an IBM JVM, you must restart the app for the Network Agent to detect and monitor any persistent connections.

Single-Tenant Setup (*One App Only on the Agent Host*)

If you want the Network Agent to monitor one app node only on the same host, perform these steps:

- 1 Install or Update the App Agent (if needed)
- 2 Extract the NetViz External Services Folder on the App Agent
- 3 Enable Socket Instrumentation
- 4 Specify the Docker Host Gateway IP on the App Agent
- 5 Restart the App (IBM JVM Only)

Install or Update the App Agent (if needed)

Network Visibility requires a Java App Agent >= 4.4 installed on the same host as the Network Agent. See Install the Java Agent.

Extract the NetViz External Services Folder on the App Agent

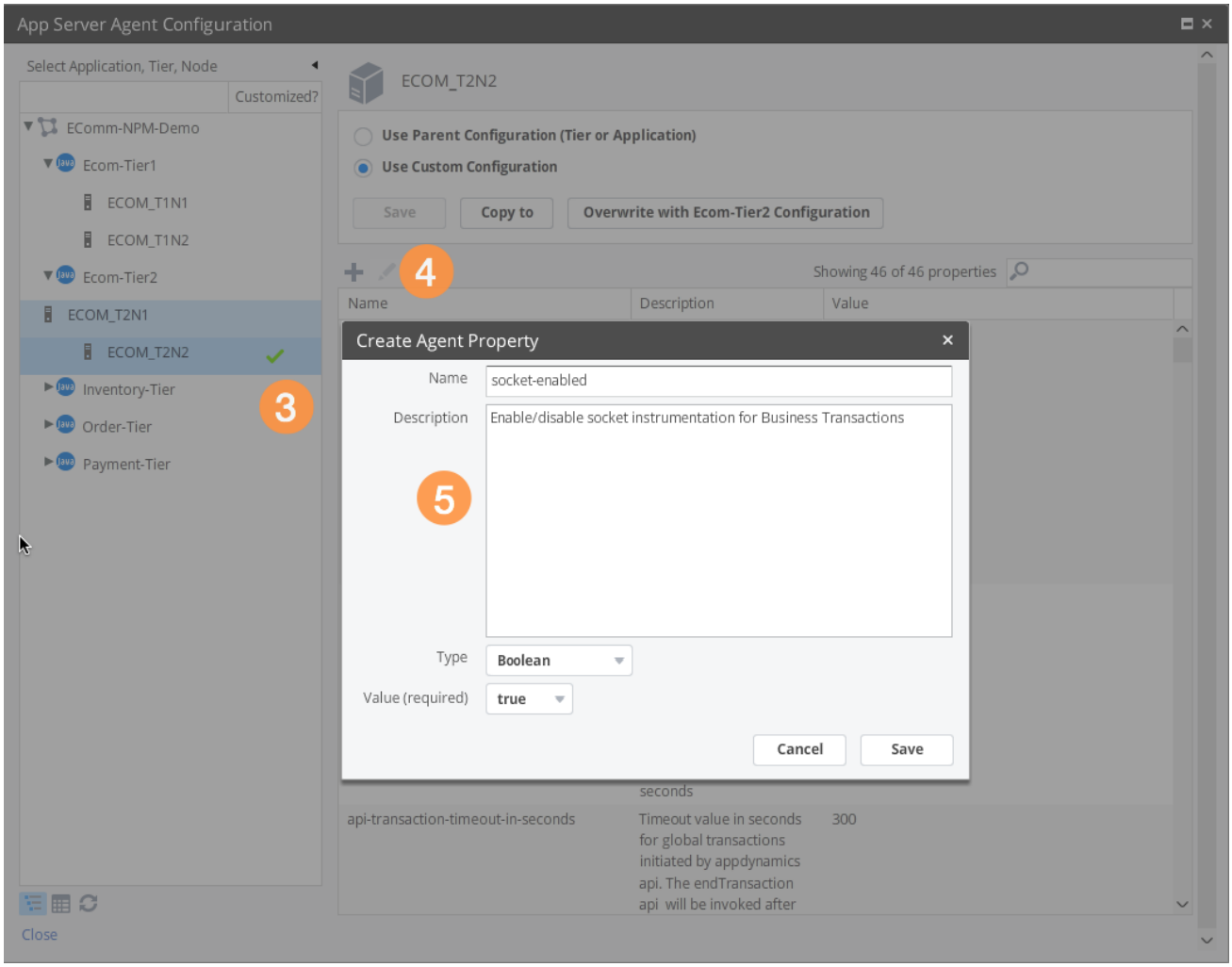
1. Navigate to folder: `<app-agent-install-dir>/<version-number>/external-services`
2. If not already extracted, extract the `netviz.zip` archive so that the ZIP contents are under this folder:
   `<app-agent-install-dir>/<version-number>/external-services`

Enable Socket Instrumentation

This is required for the Agent to map network metrics to application flows.

1. In the Controller, click the gear icon () and select **AppDynamics Agents** > **App Server Agents**.
2. Select the Agent in the table and click **Configure**. The App Server Agent Configuration page appears.
3. Select the application, tier, and node in the treeview (left).
4. Select **Use Custom Configuration** and then click **+ Create Agent Property**.
5. In the Edit Agent Property dialog, specify:

a. Name = **socket-enabled**
b. Type = **Boolean**
c. Value = **true**



Specify the Docker Host Gateway IP on the App Agent

These steps are required only if the App Agent is running in a Docker container outside of the Docker Host network.

1. Open the file: `<app-agent-install-dir>/<version-number>/external-services/netviz/netviz-service.properties`
   (You might need to unzip `external-services/netviz.zip` to access this file.)
2. For the `netviz.agent.host.address`, specify the Gateway IP of the Docker Host network interface.
   To determine this IP, run this command on the Docker host:
   `docker inspect <app_container_name> | grep Gateway`
3. Restart the App Agent.
4. To determine if the Docker Host network is connected, run this command from within the container:
   `curl -k -v <host-network-IP>:3892`
   If you receive a response, even if it is `404 Not Found`, then the network is connected.

Restart the App (*IBM JVM Only*)

If the monitored app is running in an IBM JVM, you must restart the app for the Network Agent to detect and monitor any persistent connections.

Administer the Network Agent on Linux

See the following pages for details:

ZIP Network Agent Operations

You install the Network Agent as part of an overall workflow for setting up Network Visibility. The supported workflows are:

- Multi-Tenant Setup
- Single-Tenant Setup

See Set Up Network Visibility on Linux.

> The NetViz Agent needs to be started separately from the Machine Agent when using a bundled approach.

Install the ZIP Network Agent

You can install the Agent using the Standalone Machine Agent or the Network Agent ZIP.

- Install Using the Standalone Machine Agent ZIP
- Install Using the Network Agent ZIP

Install Using the Standalone Machine Agent ZIP

You can install the Network Agent as an extension using the Standalone Machine Agent installer. This option is available on Linux 32-bit and 64-bit ZIP installers, >= 4.4.2, only.

1. Check System Requirements. See Network Visibility Supported Environments.
2. If you are installing the Network Agent as part of the Multi-Tenant Install workflow, stop each application of interest if it is monitored by an installed App Agent. You will restart each application after you install and set up the Network Agent.
3. Install the Standalone Machine Agent using the Linux ZIP installer as described in the Linux Install Using ZIP with Bundled JRE page.
4. Run the install script using `sudo`:
   ```
   cd <standalone-machine-agent-home>/NetVizExtension/agent/
   sudo ./install.sh
   ```
5. Enable the `NetVizMonitoring` extension:
   a. Open this file in a plain-text editor such as vi or Notepad++:
      `<standalone-machine-agent-home>/NetVizExtension/conf/netVizExtensionConf.yml`
   b. Set the `start` setting to `true` and save the file.
6. Start the Standalone Machine Agent. The `NetVizMonitoring` extension should start automatically. To start the Agent on Linux, enter:
   `<machine_agent_home>/bin/machine-agent -d -p <machine_agent_home>/pidfile`
7. Verify that the Network Agent is running:
   `<standalone-machine-agent-home>/NetVizExtension/agent/bin/appd-netviz.sh status`
   The output provides the process ID (PID) of the Network Agent process.
   See FAQs for Network Visibility.
8. If you are installing the Network Agent as part of a Set Up Network Visibility on Linux workflow, proceed to the next step in the workflow:
   - Multi-Tenant Setup
   - Single-Tenant Setup

Install Using the Network Agent ZIP

1. Check System Requirements. See Network Visibility Supported Environments.
2. If you are installing the Network Agent as part of the Multi-Tenant Install workflow, stop each application of interest if it is monitored by an installed App Agent. You will restart each application after you install and set up the Network Agent.
3. Verify that you have `sudo` or `root` access permissions on the host where you want to install the Network Agent. (You do not need `sudo` or `root` access to run the Agent.)
4. Open a command prompt on the host, enter the `mount` command, and check the mounted file system for the directory where you plan to install the agent. The file system should not have the `noexec` and `nsuid` flags set.
5. Download the `network-agent` package to the machine.
6. In a terminal window, `cd` to the folder where you downloaded the Network Agent ZIP package.
7. Enter the following command, where `<network_agent_home>` is the folder where you want to install the Agent software:
   `unzip <installer.zip> -d <network_agent_home>`

   > The parent path should not include any directory names with spaces:
   >
   > `/opt/appdynamics/networkVisibility` (correct)
   >
   > `/opt/appdynamics/network visibility` (incorrect)

8. Run these commands as the administrator or root user on the host machine:
   a. This command is required *only* if you are logged in as a different user from the one that will configure and run the Agent:
      `chown -R <network-agent-user-group>:<network-agent-user> <network-agent-home>`
   b. Go to `<network_agent_home>` and run the install script:
      `cd<network-agent-home>`
      `./install.sh`

9. Start the Agent by running the following command as the <*network_agent_user*>:
   <*network-agent-home*>/bin/start.sh
10. Verify that the Network Agent is running:
    a. Identify the agent process: <*network-agent-home*>/bin/appd-netviz.sh status
       The output provides the process ID (PID) of the Network Agent process.
    b. See FAQs for Network Visibility
11. If you are installing the Network Agent as part of a Set Up Network Visibility on Linux workflow, proceed to the next step in the workflow:
    - Multi-Tenant Setup
    - Single-Tenant Setup

Administer the ZIP Network Agent

> **Run commands as <network_agent_user>**
>
> You should run the following commands as a <*network_agent_user*> with permissions to configure and run the Agent on the host machine.

Operation	Command	Notes

Start Agent	<*network-agent-home*>/bin/start.sh	If the Agent starts successfully, the following displays:
		`Starting AppDynamics Network Monitoring appd-netviz...`
		`OK`
		`appd-netviz running.. <pid_of_process>`

Stop Agent	<*network-agent-home*>/bin/stop.sh	If the Agent stops successfully, the following displays:
		`Stopping AppDynamics Network Monitoring appd-netviz...`
		`OK`

Check Agent status	<*network-agent-home*>/bin/appd-netviz.sh status	This command shows whether the Agent is running or not. The following displays: `appd-netviz start/running, process <pid #>`

Restart Agent	<*network-agent-home*>/bin/appd-netviz.sh restart	If the Agent restarts successfully, the following displays:
		`Stopping AppDynamics Network Monitoring appd-netviz...`
		`OK`
		`Starting AppDynamics Network Monitoring appd-netviz...`
		`OK`
		`appd-netviz running.. <pid_of_process>`

Uninstall the ZIP Network Agent

1. Open a terminal and cd to the <*network-agent-home*> directory.
2. Stop the Agent by entering: ./bin/stop.sh
3. cd to the <*network-agent-home*> parent directory: cd ..
4. Remove the Agent directory by entering: rm -rf <*network-agent-home*>

RPM Network Agent Operations

You install the Network Agent as part of an overall workflow for setting up Network Visibility. The three supported workflows are:

- Multi-Tenant Install
- Single-Tenant Install with App Restart
- Single-Tenant Install with no App Restart

See Set Up Network Visibility on Linux.

Install the RPM Agent

1. Check System Requirements. See Network Visibility Supported Environments.
2. Stop each application of interest if it is monitored by an installed App Agent. You will restart each application after you install and set up the Network Agent.
3. Verify that you have `sudo` or `root` access permissions on the machine where you want to install the Network Agent. (You do not need `sudo` or `root` access to run the Agent.)
4. Open a command prompt on the host, enter the `mount` command, and check the mounted file system for the directory where you plan to install the Agent. The file system should not have the `noexec` and `nsuid` flags set.
5. Download the `network-agent` package to the machine.
6. Verify that you have `sudo` or `root` access permissions on the machine where you want to install the Agent.
7. Download the Agent package to the machine.
8. Open a terminal window and enter: `sudo rpm -ivh <installer.rpm>`
   The Network Agent software is installed under `/opt/appdynamics/netviz`.
9. To start the Agent, enter: `sudo service appd-netviz start`
10. Verify that the Network Agent is running: `sudo service appd-netviz status`
11. If you experience any problems or issues, see FAQs and Troubleshooting for Network Visibility.
12. If you are installing the Network Agent as part of a Set Up Network Visibility on Linux workflow, proceed to the next step in the workflow:
    - Multi-Tenant Install
    - Single-Tenant Install with App Restart
    - Single-Tenant Install with no App Restart

Administer the RPM Agent

> You should run the following commands as a `<network_agent_user>` with permissions to configure and run the Agent on the host machine.

Operation	Command	Notes
Start Agent	`sudo service appd-netviz start`	If the Agent starts successfully, the following displays: `Starting appd-netviz (via systemctl): [  OK  ]`
Stop Agent	`sudo service appd-netviz stop`	If the Agent stops successfully, the following displays: `Stopping appd-netviz (via systemctl):  [  OK  ]`
Restart Agent	`sudo service appd-netviz restart`	If the Agent restarts successfully, the following displays: `Stopping appd-netviz (via systemctl):  [  OK  ]`

Check Agent status	`sudo service appd-netviz status`	The following example illustrates status output:
		
		`appd-netviz.service - Appdynamics Network Monitoring`
		`  Loaded: loaded (/usr/lib/systemd/system/appd-netviz.service; disabled; vendor`
		`  Active: active (running) since Thu 2016-09-29 14:52:42 PDT; 14s ago`
		`  Process: 3981 ExecStart=/opt/appdynamics/netviz/bin/appd-netmon (code=exited,`
		`  Process: 3973 ExecStartPre=/bin/chown -R appd-netviz:appd-netviz /opt/appdynam`
		`  Process: 3968 ExecStartPre=/bin/mkdir -p /opt/appdynamics/netviz/run/ (code=ex`
		`Main PID: 3983 (appd-netmon)`
		`  Tasks: 12 (limit: 512)`
		`  CGroup: /system.slice/appd-netviz.service`
		`  3983 /opt/appdynamics/netviz/bin/appd-netmon`
		`  3984 appd-netagent -l /opt/appdynamics/netviz/scripts/netagent/lua/`
		`  3985 appd-netcor -l /opt/appdynamics/netviz/scripts/netcor/lua/`
		
		`Sep 29 14:52:42 localhost.localdomain systemd[1]: Starting Appdynamics Network M`
		`Sep 29 14:52:42 localhost.localdomain systemd[1]: Started Appdynamics Network Mo`
		`lines 1-15/15 (END)`

Uninstall the RPM Agent

To uninstall the Network Agent, enter:

```
sudo rpm -e appd-netviz
```

This command generates no output. To confirm that the Agent is removed, enter the status command again: `sudo service appd-netviz status`

DEB Network Agent Operations

You install the Network Agent as part of an overall workflow for setting up Network Visibility. The three supported workflows are:

- Multi-Tenant Install
- Single-Tenant Install with App Restart
- Single-Tenant Install with no App Restart

To determine the best workflow for your environment, see Set Up Network Visibility on Linux.

Install the DEB Network Agent

1. Check System Requirements. See Network Visibility Supported Environments.
2. If you are installing the Network Agent as part of the Multi-Tenant Install workflow, stop each application of interest if it is monitored by an installed App Agent. You will restart each application after you install and set up the Network Agent.
3. Verify that you have `sudo` or `root` access permissions on the machine where you want to install the Agent. (You do not need `sudo` or `root` acce ss to run the Agent.)
4. Open a command prompt on the host, enter the `mount` command, and check the mounted file system for the directory where you plan to install the Agent. The file system should not have the `noexec` and `nsuid` flags set.
5. Download the Agent package to the machine.
6. Enter:
   `sudo dpkg -i <installer.deb>`
   The Network Agent software is installed under `/opt/appdynamics/netviz`.
   The Agent should start automatically with the following message: `appd-netviz start/running. process <pid #>`
   To verify the Network Agent status, enter: `sudo service appd-netviz status`
7. If the Agent does not start, enter: `sudo service appd-netviz start`
8. If you experience any problems or issues, see FAQs and Troubleshooting for Network Visibility.
9. If you are installing the Network Agent as part of a Set Up Network Visibility on Linux workflow, proceed to the next step in the workflow:
   - Multi-Tenant Install
   - Single-Tenant Install with App Restart
   - Single-Tenant Install with no App Restart

Administer the DEB Network Agent

Operation	Command	Notes
Start Agent	`sudo service appd-netviz start`	If the Agent starts successfully, the following displays: `appd-netviz start/running. process <pid #>`

Stop Agent	`sudo service appd-netviz stop`	If the Agent stops successfully, the following displays:
		`appd-netviz stop`
		`appd-netviz stop/waiting`

Restart Agent	`sudo service appd-netviz restart`	
Check Agent Status	`sudo service appd-netviz status`	This command shows whether the Agent is running or not. The following displays: `appd-netviz start/running, process <pid #>`

Uninstall the DEB Network Agent

The Network Agent software is installed under `/opt/appdynamics/netviz`. To uninstall the Agent, enter:

```
sudo dpkg -P appd-netviz
```

Managing Network Agents in the Controller

Controller Operations for Network Agents

Using an AppDynamics Administrator account, select **Settings > AppDynamics Agents > Network Visibility Agents** to see details and manage your installed Machine Agents. You can perform most operations on multiple Agents: Shift or Ctrl+Click the Agents of interest, right-click, and select the operation. These Agent settings are defined in the Controller and retain their state when an Agent is stopped and restarted.

Right-click one or more Agents to:

- View Packet Capture Configuration
- Start Packet Capture
- Disable Agent(s) or Enable Agent(s)
- Change Dynamic Monitoring Mode
- Delete Agent from System

View Packet Capture Configuration

View and edit configuration settings for a single network Agent. See Packet Captures.

Start Packet Capture

Start a packet capture on one or more Agents.

Disable Agent(s) or Enable Agent(s)

When disabled, the Agent temporarily suspends its collection of network metrics. You can quickly re-enable the Agent when you want.

While disabled, the Agent continues to consume a Network Visibility license. The Agent maintains a "heartbeat" connection to the Controller so you can enable it again quickly. The Agent persists its enabled/disabled state after it restarts and resumes all normal activities after it is re-enabled.

Change Dynamic Monitoring Mode

Defines the set of metrics that the Agent collects. In general, you should run the Agent in KPI unless you need to diagnose network performance on the node or Connections monitored by that Agent. See Dynamic Monitoring Mode and Network Visibility.

Delete Agent from System

Deletes the underlying Network Visibility metadata for an orphaned Network Agent. A Network Agent is considered "orphaned" when the Agent host no longer has any App Server Agents installed or running. To delete a Network Agent, you must first delete all APM nodes on that host. Account Administrator privileges are required to delete Network Agents.

Check Network Agent Status

To check the status of your Network Agents, access **Tiers & Nodes** and then select **Network** in the Show Data pulldown.

Network Visibility with Kubernetes

This page describes how to configure Network Visibility to monitor applications running on Kubernetes.

Network Visibility monitors an application's network interactions and reports key performance metrics. These metrics isolate an application's network issues from its application issues.

The Network Agent uses a REST API to open a TCP port in each node for the application containers to communicate with the Network DaemonSet container. This communication enables monitoring between pods and nodes. You can deploy the agent as a DaemonSet in each node that has the host mode enabled.

Before You Begin

Before you create a Docker image for the DaemonSet and configure the agent, verify the following requirements:

- You have at least one pod with a Java Agent >= 4.4 deployed to the same cluster as the Network Agent.
- The TCP port 3892 is not already used by the node. The application pods use port 3892 to communicate with the DaemonSet.

Create a Network Visibility DaemonSet Docker Image

To configure Network Visibility with Kubernetes, you must first create a Docker image for the Network Visibility DaemonSet and push the image to your Docker Trusted Registry.

1. Paste this sample Dockerfile into a text file and save the file.

```
FROM ubuntu:14.04

ARG NETVIZ_ZIP_PKG
RUN groupadd -r appd-netviz && useradd -r -g appd-netviz appd-netviz

RUN apt-get update && apt-get install -y \
  net-tools \
  tcpdump \
  curl  \
  unzip   \
  ssh-client \
  binutils \
  build-essential \

WORKDIR /netviz-agent

# copy NetViz agent contents
COPY $NETVIZ_ZIP_PKG .

# run the agent install script and disable netlib
RUN unzip $NETVIZ_ZIP_PKG && ./install.sh \
    && sed -i -e "s|enable_netlib = 1|enable_netlib = 0|g" ./conf/agent_config.lua \
    && sed -i -e "s|WEBSERVICE_IP=.*|WEBSERVICE_IP=\"0.0.0.0\"|g" ./conf/agent_config.lua

RUN chown -R appd-netviz:appd-netviz /netviz-agent
RUN setcap cap_net_raw=eip /netviz-agent/bin/appd-netagent
USER appd-netviz

# default command to run for the agent
CMD  ./bin/appd-netagent -c ./conf -l ./logs -r ./run
```

2. Navigate to the directory where you saved the Dockerfile and build the Docker image by entering the following command:

```
$ docker build --build-arg NETVIZ_ZIP_PKG=/path/to/netviz-agent-pkg.zip -t appd-netviz .
```

3. Push the Docker image to your Docker Trusted Registry.

Configure Network Visibility with Kubernetes

1. Create a `yaml` file with the following configuration:

```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: DaemonSet
metadata:
  name: appd-netviz-agent
spec:
  selector:
    matchLabels:
      name: appd-netviz-agent
  template:
    metadata:
      name: appd-netviz-agent
      labels:
        name: appd-netviz-agent
    spec:
      hostIPC: true
      hostNetwork: true
      containers:
      - name: appd-netviz-agent
        image: path/to/your/Docker/image # docker registry image
        resources:
          requests:
            memory: "250Mi"
            cpu: "0.5"
          limits:
            memory: "2Gi"
            cpu: "1"
        securityContext:
          capabilities:
            add: ["NET_RAW"]
        ports:
        - containerPort: 3892
          hostPort: 3892
      imagePullSecrets:
        - name: your-registry-key # add the registry key, kubectl create secret ...
```

2. In the configuration file, update the following fields:
   - `image` (under `containers`): the file path to the DaemonSet image in your Docker Trusted Registry.
   - `name` (under `imagePullSecrets`): the key for your Docker Trusted Registry.
3. Deploy the Network Agent in Kubernetes by entering the following command:

```
$ kubectl apply -f MyConfigFile.yaml
```

Configure Network Visibility to Monitor Application Pods

After deploying Network Visibility in Kubernetes, you must deploy at least one pod with a Java Agent >= 4.4 to the same cluster as the Network Agent. This allows you to map network metrics to application flows.

1. Open the application's deployment configuration `yaml` file with Kubernetes in a text editor and set the `APPDYNAMICS_NETVIZ_AGENT_HOST` and `APPDYNAMICS_NETVIZ_AGENT_PORT` values:

```
- name: APPDYNAMICS_NETVIZ_AGENT_HOST
  valueFrom:
    fieldRef:
      fieldPath: status.hostIP
- name: APPDYNAMICS_NETVIZ_AGENT_PORT
  value: 3892
```

If you are using a version of the Java Agent that is <= 4.5.2, you must open the `<app-agent-install-dir>/<version-number>/external-services/netviz/netviz-service.properties` file to set these values.

2. In the Controller UI, enable socket instrumentation to map network metrics to application flows. See Set Up the Network and App Agents.

Set Up Network Visibility on Windows

To set up Network Visibility on Windows, see:

1. Network Visibility Supported Environments
2. Install the Network Agent on Windows
3. Install the App Agents on Windows

> Network Visibility on Windows supports both Java and .NET applications.

4. Administer the Network Agent on Windows
5. Verify the Network Visibility Installation on Windows

Install the Network Agent on Windows

The Network Visibility on Windows requires a supported version of the Network Agent to be installed on the server.

To monitor Java applications in your Controller, install the AppDynamics Java Agent on each server that hosts applications that you want to monitor.

You need a Network Visibility license to enable the Network Agent and its features.

No additional configuration is required on your AppDynamics Controller; Network Visibility is enabled by default.

Agent Install Directory

Network Agent is packaged as a zip bundle. The Network Agent install directory <*network_agent_home*> is the directory where you install the Agent. You can unzip the bundle at any location which is your <*network_agent_home*> directory.

It contains these administrative files:

- `install.bat` to install and start the Network Visibility Agent
- `start.bat` to start the Network Visibility Agent
- `status.bat` to check the status of the Network Visibility Agent
- `stop.bat` to stop the Network Visibility Agent
- `uninstall.bat` to uninstall the Network Visibility Agent

As part of the installation, `install.bat` installs the required VC redistributable on the system along with NPcap (if it does not already exist). If Npcap does exist on the system, then the installation of NPcap is skipped. To listen to the traffic on localhost, NPcap should be running in LoopBack mode. If NPcap is already installed on the system and not running in loopback mode, you must re-install it using loopback mode.

Network Agent and Java Agent Installation Instructions

Depending on the Network Agent version and Java Agent version, you can install the Network Agent in a multi-tenant or single-tenant setup:

If you are using	Multi-tenant Setup	Single-tenant Setup
Network Agent >= 4.5.7, and Java Agent >= 4.5.11	Install	Install
Network Agent >= 4.5.7, and Java Agent <= 4.5.10	Not supported	Install

Install Network Agent on Windows using the ZIP Installer

These steps describe how to install and run the Network Agent for Windows using the ZIP installer.

1. Review the Network Visibility Requirements and Supported Environments.
2. Download the Network Agent Windows ZIP package from the AppDynamics Downloads Portal and unzip the package.
    a. Verify your `administrator` permissions on the host where you want to install the Network Agent.
    b. Extract the contents to the Agent installation directory <*network_agent_home*>.
3. Double-click the `install.bat` file to install the Network Agent and start the AppDynamics Network Visibility Service.

    > If you run the `install.bat` script without `administrator` privileges, a dialog displays this message: "Do you want to allow this app to make changes to your device?"
    >
    > Click **Yes** to continue with the installation.

4. To verify if the Network Agent is running, navigate to the folder that contains the `status.bat` file.
    a. Double-click the `status.bat` file.
    b. Locate the AppDynamics Network Visibility Service and verify its status in the **Status** column.
    Alternately, check if AppDynamics NetViz Service is listed in services.msc.

See Administer the Network Agent on Windows.

Uninstall Network Agent Using the ZIP

To uninstall the Network Agent:

1. Open a Windows command prompt and **cd** to the <*network-agent-home*> directory.
2. Uninstall the Network Agent by entering: `\<network-agent-home>\uninstall.bat.`

Install Network Agent on Windows using the MSI Package

To install the Network Agent using the MSI package:

1. Review the Network Visibility Requirements and Supported Environments.
2. Download the MSI installer package from the AppDynamics Download Center.

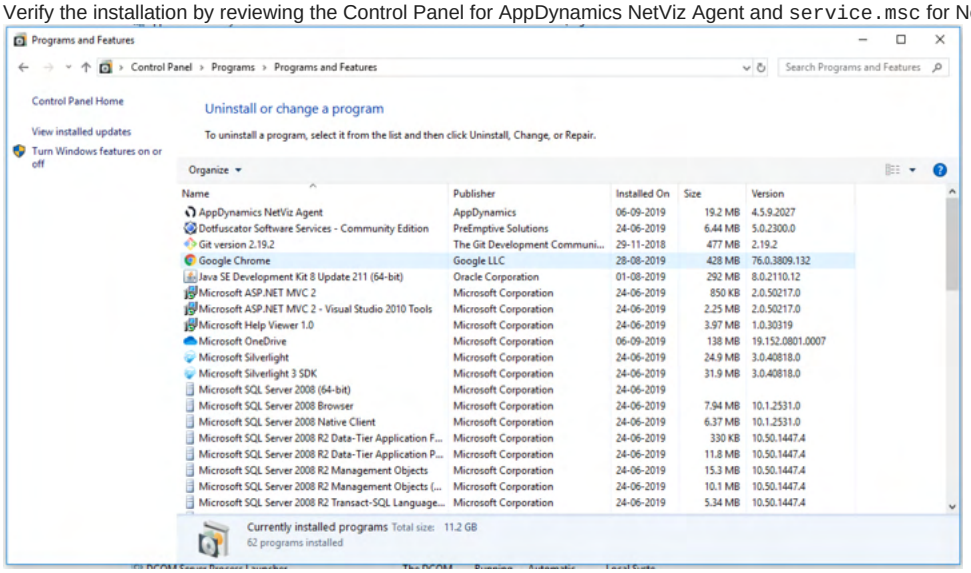
3. Run the MSI installer package.
4. Read the End User License Agreement and accept the terms. Click **Next**.
5. If required, change the Network Agent installation directory on the **Destination Folder** panel.

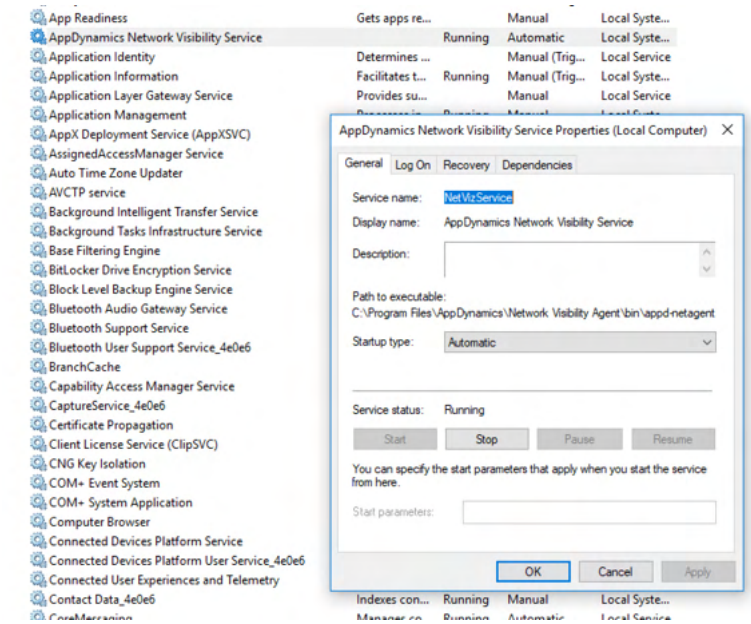
> You can also change the destination of the logs directory later using the AppDynamics Agent Configuration utility.

6. Click **Next**. The **Ready to install** page displays.
7. Click **Install**. The **User Account Control** panel displays and prompts you for permission to install.
8. From the **User Account Control** panel, click **Yes** to allow the installer to make changes to the device. Wait for the installation to complete.

> If you do not have administrator privileges, the installer prompts you to provide the password for an administrator account.

9. Click **Finish**.
10. Verify the installation by reviewing the Control Panel for AppDynamics NetViz Agent and `service.msc` for NetVizService.





Uninstall Network Agent Using the MSI Package

To uninstall the Network Agent:

1. Open the Windows Control Panel and select **AppDynamics NetViz Agent**.
2. Click **Uninstall**.

3. From the **User Account Control** panel, click **Yes** to uninstall the Network Agent.

Install the App Agent on Windows

Instrumenting an application adds the AppDynamics Application Agent, known as an App Agent, into the runtime process of the application. An App Server is a server application that is constantly running. You can deploy applications on the App Server.

Install the Java App Agent on Windows

To monitor Java applications in the Controller, you need to install the AppDynamics Java Agent >= 4.5.4, on each server that hosts applications that you want to monitor.

- To install using the Agent Download Wizard in the Controller, see Java Agent.
- To install and configure the AppDynamics Java Agent into the application JVM, see Install the Java Agent.

Set Up the Java App Agent on Windows

If you want the Network Agent to monitor an App Node on the same host:

1. Enable Socket Instrumentation.
2. Restart the App (*IBM JVM only*).

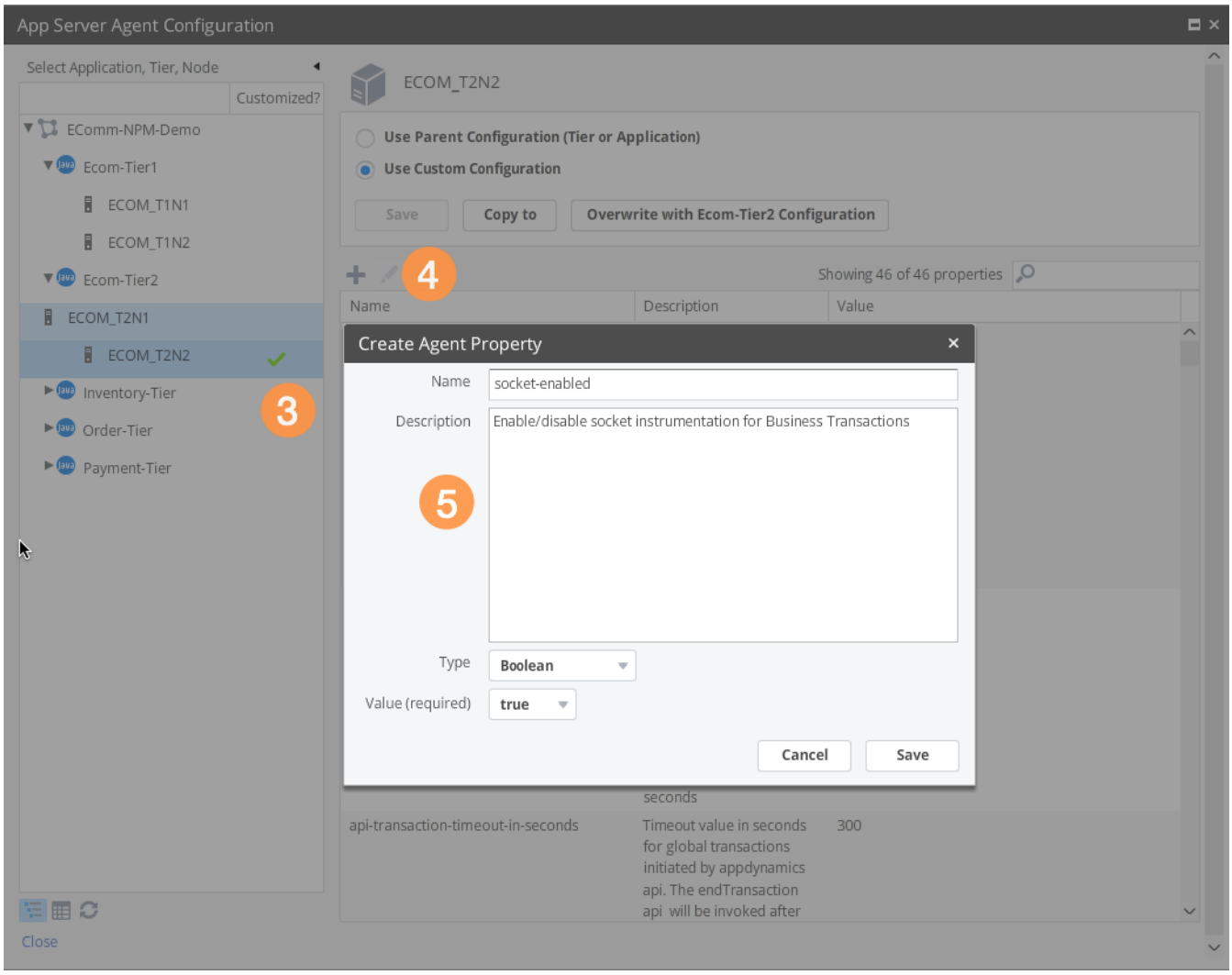
Enable Socket Instrumentation

> If you are using Java Agent >= 4.5.13, you need not manually enable the socket collection. It is enabled automatically. You can skip Step 1 and proceed with Step 2.

For the Agent to map network metrics to application flows, perform these steps:

1. In the Controller, click the gear icon ( ) and select **AppDynamics Agents** > **App Server Agents**.
2. Select the Agent in the table and click **Configure**. The App Server Agent Configuration page appears.
3. Select the application, tier, and node in the tree-view (left).
4. Select Use Custom Configuration and then click **+**. The **Create Agent Property** dialog appears.
5. From the **Create Agent Property** dialog, specify:

    a. Name = **socket-enabled**
    b. Type = **Boolean**
    c. Value = **true**



Restart the App (*IBM JVM only*).

If the monitored App is running in an IBM JVM, you must restart the App for the Network Agent to detect and monitor any persistent connections.

Install and Set Up .NET App Agent on Windows

To monitor .NET applications with AppDynamics, you need to install the .NET Agent on the servers where the applications run.

You only need to install the Agent once per server (even if you want to monitor more than one application on the server).

- To install the .NET agent for IIS applications using the Getting Started wizard in the Controller, see .NET Agent.
- To install and configure the AppDynamics .NET Agent using the MSI package, see Install .NET Agent.

Set Up the .NET App Agent on Windows

If you want the Network Agent to monitor an app node on the same host, Enable Socket Instrumentation.

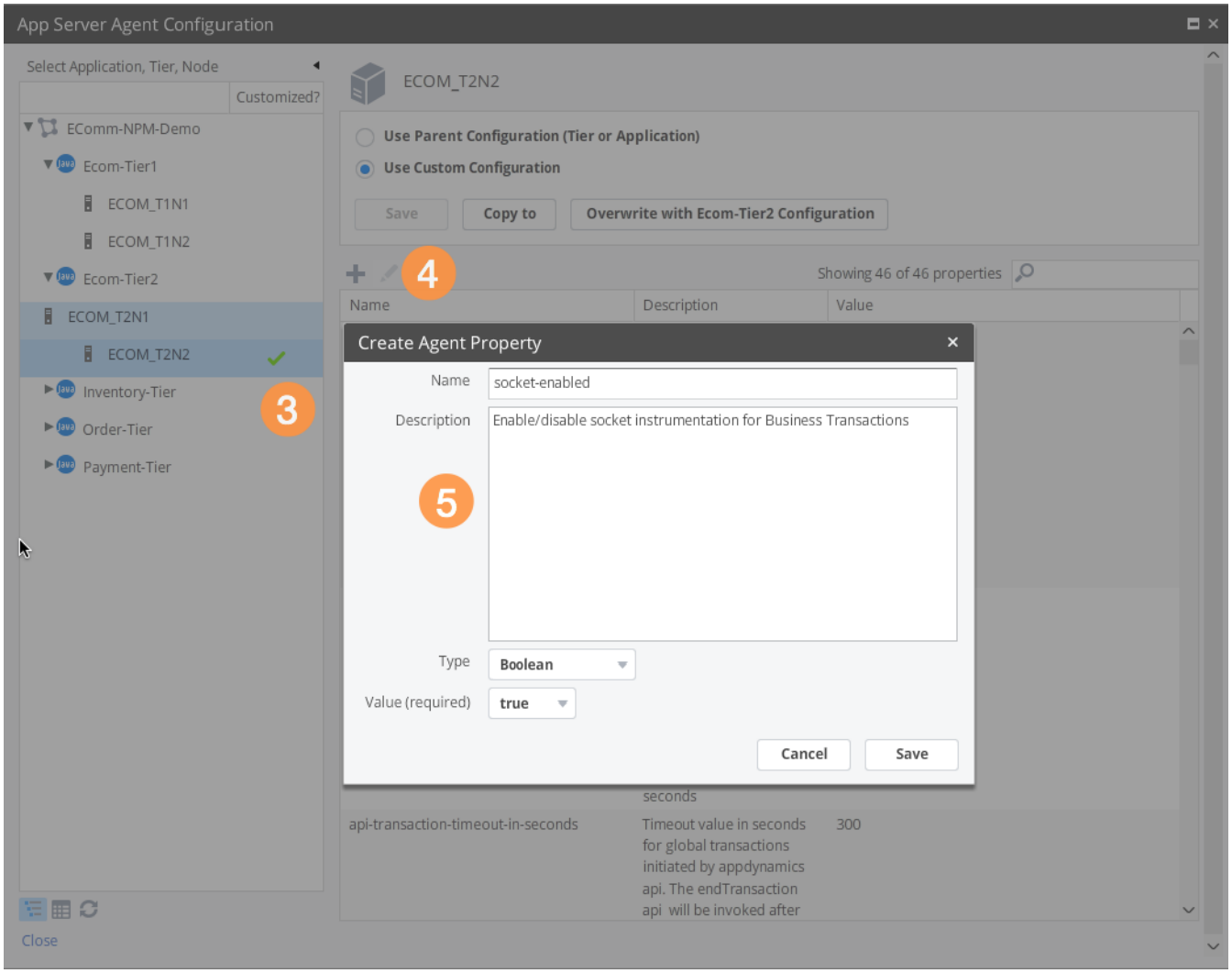
Enable Socket Instrumentation

For the Agent to map network metrics to application flows, perform these steps:

1. In the Controller, click the gear icon ( ) and select **AppDynamics Agents** > **App Server Agents**.
2. Select the Agent in the table and click **Configure**. The App Server Agent Configuration page appears.
3. Select the application, tier, and node in the tree-view (left).
4. Select Use Custom Configuration and then click **+**. The **Create Agent Property** dialog appears.
5. In the **Create Agent Property** dialog, specify:

    a. Name = **socket-enabled**
    b. Type = **Boolean**

c. Value = **true**


App Server Agent Configuration
Select Application, Tier, Node
Customized?
EComm-NPM-Demo
Ecom-Tier1
ECOM_T1N1
ECOM_T1N2
Ecom-Tier2
ECOM_T2N1
ECOM_T2N2
Inventory-Tier
Order-Tier
Payment-Tier
Close
ECOM_T2N2
Use Parent Configuration (Tier or Application)
Use Custom Configuration
Save
Copy to
Overwrite with Ecom-Tier2 Configuration
4
Showing 46 of 46 properties
Name
Description
Value
3
Create Agent Property
Name
socket-enabled
Description
Enable/disable socket instrumentation for Business Transactions
5
Type
Boolean
Value (required)
true
Cancel
Save
seconds
api-transaction-timeout-in-seconds
Timeout value in seconds for global transactions initiated by appdynamics api. The endTransaction api will be invoked after
300

Administer the Network Agent on Windows

Related pages:

- License Management
- Java Supported Environments

Depending on how you installed and deployed the Network Agent, there may be more than one method to administer the Network Agent. AppDynamics provides both GUI and command-line tool for common operations, such as starting and stopping Network Agents and their services. See Managing Network Agents in the Controller.

This page describes how to administer the Network Agent using command-line tools for the Network Agent ZIP. You can manage the Network Agent using these commands:

- `install`
- `start`
- `status`
- `stop`
- `uninstall`

Ensure that you run all the scripts using administrator privilege. If you run the scripts without administrator privilege, the scripts prompt for the privilege request after a three second timeout. You can start these scripts from Windows Explorer or from the command line. If you run the scripts from the command line, the `install`, `start` and `stop` scripts have a `-quiet` option. The `-quiet` option runs the scripts without prompts, messages, or dialogs.

Operation	Command	Notes
Install Agent	`\<`*`network-agent-home`*`>\install.bat`	The `install.bat` installs the Network Visibility Agent and starts the AppDynamics Network Visibility Service.

Start Agent	`\<`*`network-agent-home`*`>\start.bat`	If the Agent starts successfully, this message displays:
		`Starting the network agent service`
		`SERVICE_NAME: NetVizService`
		`TYPE : 10 WIN32_OWN_PROCESS`
		`STATE : 4 RUNNING`
		`(STOPPABLE, NOT_PAUSABLE, ACCEPTS_PRESHUTDOWN)`
		`WIN32_EXIT_CODE : 0 (0x0)`
		`SERVICE_EXIT_CODE : 0 (0x0)`
		`CHECKPOINT : 0x0`
		`WAIT_HINT : 0x0`
		`PID : 7556`
		`FLAGS :`

Check Agent status	`\<`*`network-agent-home`*`>\status.bat`	This command shows if the Agent is running or not.
		`SERVICE_NAME: NetVizService`
		`TYPE : 10 WIN32_OWN_PROCESS`
		`STATE : 4 RUNNING`
		`(STOPPABLE, NOT_PAUSABLE, ACCEPTS_PRESHUTDOWN)`
		`WIN32_EXIT_CODE : 0 (0x0)`
		`SERVICE_EXIT_CODE : 0 (0x0)`
		`CHECKPOINT : 0x0`
		`WAIT_HINT : 0x0`

Stop Agent	`\<`*`network-agent-home`*`>\stop.bat`	If the Agent stops successfully, this message displays:
		`SERVICE_NAME: NetVizService`
		`TYPE : 10 WIN32_OWN_PROCESS`
		`STATE : 1 STOPPED`
		`WIN32_EXIT_CODE : 0 (0x0)`
		`SERVICE_EXIT_CODE : 0 (0x0)`
		`CHECKPOINT : 0x0`
		`WAIT_HINT : 0x0`

Uninstall Agent	`\<`*`network-agent-home`*`>\uninstall.bat`	If the Agent uninstalls successfully, this message displays:
		`Stopping service NetVizService.`
		`NetVizService is stopped.`
		`Service AppDynamics NetViz Service uninstalled.`

Manage Network Agent on Windows

You can start or stop the Network Agent using `start.bat` or `stop.bat`. You require admin privileges to start or stop the Network Agent.

You can view the services that are currently running on the system using AppDynamics NetViz Service.

Network Visibility Metrics

Network Visibility can collect these types of metrics:

- KPI Metrics in Network Dashboard and Application Flow Map
- Key Performance Indicator (KPI) metrics are essential measures of whether a network element is performing successfully. The Network Browser shows KPIs for tiers, nodes, network links, and the entire application
- KPI Metrics in Right-Click Dashboards
  You can right-click on a network element and perform network troubleshooting in the right-click dashboard. These dashboards make it easy to drill down to network elements and metrics that correlate with application-performance issues.
- TCP Connection Metrics in Metric Browser
  When you narrow down a network problem to a specific network location, you can configure Network Agents to collect advanced metrics for the relevant connections.
- Node Metrics in Network Dashboard and Metric Browser
  The Network Agent also collects node-level metrics for monitoring the TCP stack health, collisions, and errors on physical interfaces.

KPI Metrics in Network Dashboard and Application Flow Map

Some network metrics are considered Key Performance Indicators (KPIs) because they are essential measures of whether the network or a specific part of the network, is performing successfully. Network Agents collect network KPIs for all monitored tiers, nodes, network links, and application flows. You can view KPI metrics for these elements:

View Network KPIs

For the Entire Application

The **Network Dashboard** includes Throughput, TCP Loss, and PIE charts that measure network health for the entire application.

For a Network Link

- **Network Dashboard** – The flow map shows KPIs for each link. Click a link to view KPI time charts and KPIs per Connection.
- Right-click dashboard – Right-click a link in the Network Dashboard and select **View Metrics**. See KPI Metrics in Right-Click Dashboards.

For a Tier

- **Network Flow Map** – Click a tier to view KPI time charts.
- Right-click dashboard – Right-click a tier in the Network Dashboard and select **View Metrics**. See KPI Metrics in Right-Click Dashboards.

For a Node

- **Tiers & Nodes** page – Set the Show Data menu to **Network**. The table shows network KPIs and the Network Agent status for each node.
- **Node Dashboard** – Right-click the node and select **View Metrics**. See KPI Metrics in Right-Click Dashboards.

For a Connection

Network Agents do not collect metrics for Connections by default. To collect metrics for the Connections used by a specific node, change the Monitoring Mode on the Agent. See Dynamic Monitoring Mode and Network Visibility.

Connections Explorer – Access the Network Dashboard and select **Connections**. The Explorer shows network KPIs for all network links. Drill down into a network link to view the Connections used on that link. If a Network Agent has collected metrics for a Connection, you can:

- View network KPIs in the Explorer table.
- Right-click and select **View Metrics**. See KPI Metrics in Right-Click Dashboards.
- Right-click and select **View Metric Browser**. See TCP Connection Metrics in Metric Browser.

KPI Metric Descriptions

Metric	Description	Default Monitoring Mode
Throughput	The average rate of Application *X* traffic sent and received over one Connection, or a set of Connections. Correlated spikes along with Response Time indicate bottlenecks.	KPI
Latency	The average TCP Latency (round-trip time) for a packet to go from one application tier the other, and back again. When TCP effects are not significant, Latency primarily measures *network* round trip times. Network latency is primarily a factor of the physical distance between two nodes and remains consistent over time. Correlated spikes along with Response Time indicate that TCP effects between two tiers (such as packet queueing and delayed acknowledgments) are affecting application performance.	KPI
TCP Loss (mille)	The average number of packets (per 1000) that were sent but never received. Packet loss can degrade application performance significantly. TCP detects and re-transmits lost packets; this ensures reliable transmission but introduces delays. • Correlated spikes with Response Time indicates that network congestion between the sender and receiver is affecting application performance. • Correlated spikes with Throughput indicates that a tier is sending more data than the receiving tier or the network can handle.	KPI

Errors	The number of TCP messages sent indicating an error in setting up the connection (SYN errors) or tearing down the connection (FIN errors) for a specific TCP session.	KPI
PIE (Performance Impacting Events)	The rate of PIE (Performance Impacting Events), which indicate performance issues on one node, both nodes, or the network between two nodes: • Client Limited and Client Zero Window events indicate that the Client node is receiving data at a higher rate than it can process. • TCP Retransmission Timeouts (RTO) events indicate that the network is losing packets between the client and the server node. RTOs can cause severe performance degradations. • Server Limited and Server Zero Window events indicate that the Server node is receiving data at a higher rate than it can process.	KPI

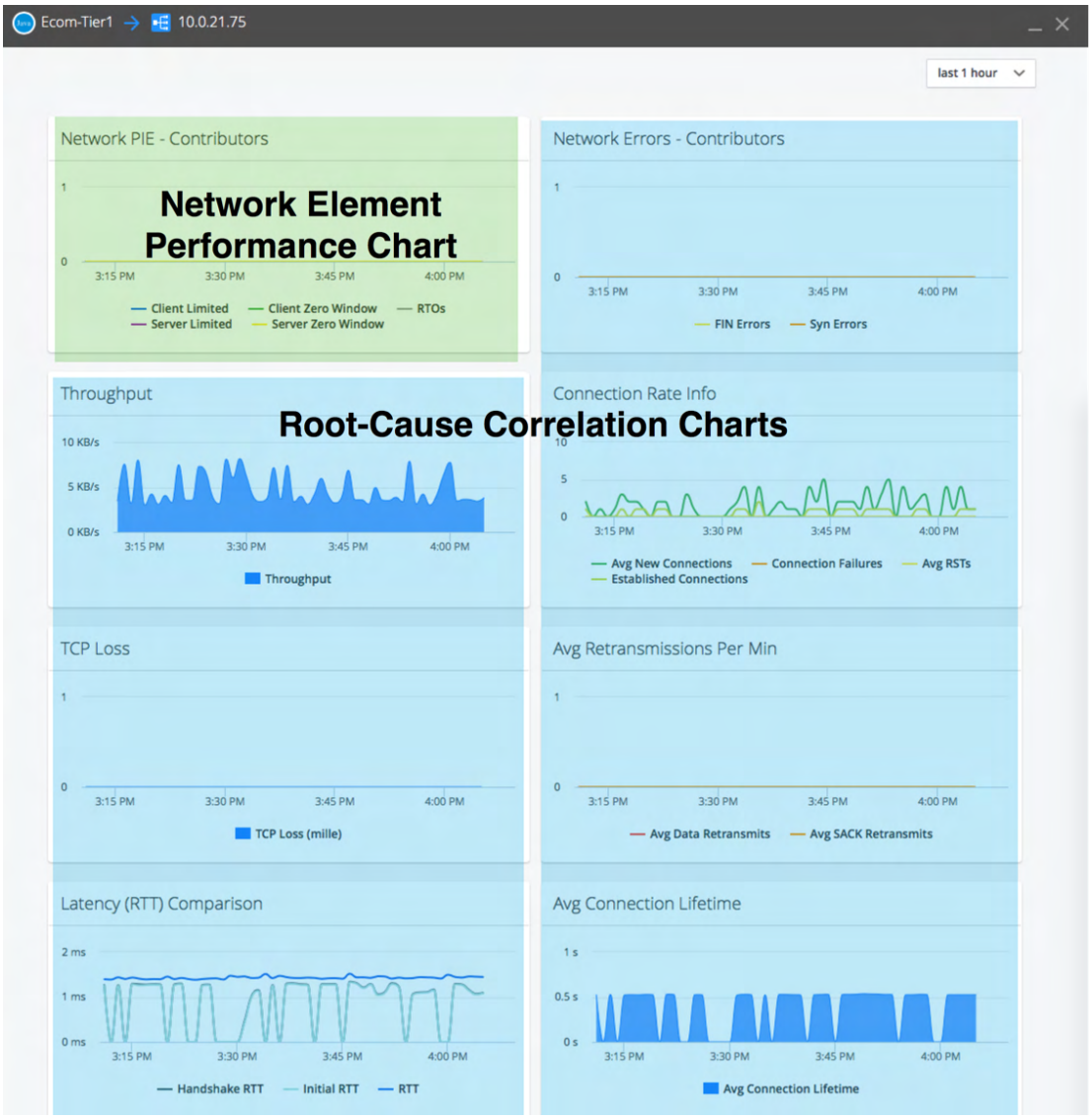
KPI Metrics in Right-Click Dashboards

To diagnose a network element, right-click and select **View Metrics**. You can troubleshoot these network elements:

- Tier – The performance chart (top left) shows the rate or application performance outliers on the relevant nodes (Errors and Slow/Very Slow /Stalled Calls) and the Key Performance Indicators for all connections used by those nodes (Errors and PIE).
- Network Link – The performance chart (top left) shows the rate of Performance Impacting Events for all member connections of that link.
- Network Connection – You can troubleshoot a connection from the Connection Explorer or from the Connections tab in a link popup.
- You can troubleshoot a transaction from a Transaction Snapshot.
    - You can right-click an application flow and select **View Network Metrics**.
    - You can also drill down to a node where a transaction delay/stall/error occurred and access the Network tab.

To troubleshoot a tier, link, or connection in the Network Dashboard, right-click the network element and select **View Metrics**. The top-left chart in the dashboard shows the overall network or network/application performance of the element. To troubleshoot the element, search for correlations between the performance chart (top left) and the other charts on the page.

> **Tip**
>
> To best highlight metric spikes and variations, you can switch between the linear and logarithmic scale in each chart. Click the **settings** button (top-right corner of the chart) to switch between scales.



Right-Click Chart Descriptions

Chart	Description	Default Monitoring Mode
Network Impact on Transactions (Tiers)	This chart highlights the possible impact of Performance Impacting Events on the app-transaction outliers (Transaction Errors and Slow, Very Slow, and Stalled Calls shown in the Transaction Scorecard). If you see spikes in transaction outliers and correlated spikes in PIE or errors, this indicates that the network is affecting application performance. Search for correlated spikes in the other charts to identify specific issues and root causes: connection errors, packet loss, re-transmissions, high-latency connections, and so on.	KPI
Host Stack KPIs	Interface errors indicate issues on the node's physical interfaces. See Interface Metrics. TCP Wait sockets can result in significant delays or errors for the application, or service, that relies on that socket. Many simultaneous WAIT sockets can prevent applications and services from creating new connections.	KPI
Network PIE - Contributors	Use Performance Impacting Elements (PIE) to identify the location of actual or potential bottlenecks: • Client Limited and Client Zero Window events indicate a possible problem on the client node • RTOs indicate a possible problem on the network path between two nodes • Server Limited and Server Zero Window events indicate a possible problem on the server node	KPI
Network Errors - Contributors	Use this chart to identify network errors that can affect application performance: • FIN Errors – Errors when tearing down a TCP connection • Syn Blackholes - Connection attempts that went unanswered and resulted in a failure • Syn Resets – Connection attempts that were explicitly refused by the other host • RST on Established - TCP connection resets	KPI
Throughput	Traffic throughput for the application of interest on the network element. This chart measures application payload data only; TCP, IP, and other packet headers are excluded.	KPI

Connection Rate Info	Spikes in this chart, that coincide with spikes in slows calls or applications errors, indicate a possible issue with how the application is using TCP.
	
	• New Connections – Number of new connection attempts
	• Connection Errors – The rate of TCP connection setup (SYN) errors and connection teardown (FIN) errors
	• RSTs (resets) – A TCP reset is an immediate closing of a connection. Not all connection resets indicate a problem, but you should investigate any spike in resets that coincides with a spike in application errors or slow transactions /calls. Connection resets can occur for various reasons, such as:
	  • Inability to create connections by the server
	  • Intermediate network elements, such as load balancers, firewalls, and so on, due to misconfiguration or other errors
	• Current Established Connections
		KPI

TCP Loss (mille)	The number of packets lost (sent but not received) per 1000 packets sent. "Per mille" is a percentage with one additional digit of precision. TCP detects lost packets and retransmits each lost packet until it receives an ACK (acknowledgment) from the peer. Spikes in TCP Loss indicate that the network is over utilized.	KPI
Retransmissions per Minute	• Data Retransmits – The percentage of packets that were retransmitted. This metric includes SYN and FIN retransmissions • SACK Retransmits – The percentage of data packets that were retransmitted due to selective acknowledgments (SACK, a TCP Feature)	KPI
Latency (RTT) Comparison	This chart compares the average TCP round-trip times (RTTs) for different types of packet request and responses. • Handshake RTT – Average RTT for the initial 3-way handshake (SYN, SYN/ACK, ACK) to set up a connection. If Handshake RTTs are significantly higher than Initial RTTs, this indicates that the client node is taking a long time to set up the connection. • Initial RTT – Average RTT for the initial SYN packets (between SYN and SYN-ACK or SYN-ACK and ACK). If Initial RTTs are high, the delay may be due to an intervening firewall, asymmetric routing, or other network issues that causes the network to treat SYN packets differently from data packets. • RTT – Average TCP RTT for data request/responses—Time for TCP Data segments to be transmitted and acknowledged from the peer. This is different from Application Response Time metrics that measure the time an application takes to process a request.	KPI
Connection Lifetime	TCP is most efficient when using long, stable connections. Connection setups and teardowns are very time-consuming and resource-intensive. The more short-term connections are generated, the worse TCP performance will be because most of the time is spent creating connections and because "Slow-Start" in TCP optimal TCP bandwidth is not achieved.	KPI

Node Metrics in Network Dashboard and Metric Browser

The Network Agent collects various TCP socket metrics at the host level for the local node on which it is installed. Use these metrics to monitor the TCP stack health, collisions, and errors on all the physical interfaces, and CPU consumption of the Network Agent.

TCP Socket Metrics in the Network Dashboard

These metrics measure the overall TCP socket health of an individual node. To view these metrics, select a node: access the Network Dashboard, click a tier, select the Nodes tab, and select the node of interest. The TCP Socket metrics for the selected node appear on the right side of the dashboard.

Metric	Description	Default Monitoring Mode
Established	The number of sockets that are in the established state	KPI
Embryonic	The number of sockets that are in the process of being set up	KPI
Wait	The number of sockets that are unavailable for new connections because they are in the process of being closed. When a socket sends a Connection Close (FIN) message, the socket goes through three states: 1. FIN_WAIT1 – Socket has sent the FIN and is waiting for an ACK from the remote socket 2. FIN_WAIT2 – Socket receives ACK, and waits for FIN (final connection close) from the remote socket 3. TIME_WAIT – The socket remains open to handle any packets for the TCP connection that might still be in the network. The TIME_WAIT state can last from one to four minutes, depending on the implementation. A WAIT socket can result in significant delays or errors for the application, or service, that relies on that socket. Many simultaneous WAIT sockets can prevent applications and services from creating new connections.	KPI
Time Wait	The FIN_WAIT states are relatively short (in milliseconds), while the TIME_WAIT state is relatively long (one to four minutes). As a result, the Wait and Time Wait metrics should be fairly similar. Significant differences indicate a significant network or other delays between the selected node and another host.	KPI

Interface Metrics in the Metric Browser

These metrics measure the number of collisions and errors on the physical interfaces of the node. To view these metrics, drill down in the Metric Browser by selecting:

Application Infrastructure Performance > <*tier-name*> **> Individual Nodes >** <*node-name*> **> Host > Interface**

Metric Name	Description / Notes	Monitoring Mode
# **Collisions**	Number of collision errors seen on all the interfaces, when an interface was blocked from sending a frame due to an Ethernet collision detection	KPI
# **Receive Drops**	Number of packets dropped while receiving by all interfaces on the host	KPI
# **Receive Errors**	Number of collision errors seen on all the interfaces, when an interface could not receive a frame due to a physical problem on the interface or the connected cable	KPI
# **Transmit Drops**	Number of packets dropped while transmitting by all interfaces on the host	KPI
# **Transmit Errors**	Number of collision errors seen on all the interfaces, when an interface could not send a frame due to a physical problem on the interface or the connected cable	KPI
# **Total Errors**	The total number of Ethernet or physical errors detected on the interface	KPI

TCP Connection Metrics in Metric Browser

Network Visibility Agents can calculate an extensive set of metrics based on the TCP flows observed by the Network Agents. In addition to KPI, PIE, and Troubleshooting metrics, you can view advanced metrics for network elements of interest (tiers, nodes, links, and connections) in the Metric Browser. Using time-based charts, you can detect and analyze these behavior traits:

- TCP flow setup/teardown times and errors
- Distribution of TCP flows for an element by throughput, setup time, lifetime, and round-trip time
- Flows closed by TCP resets
- Distribution of flows based on TCP configuration options (Selective Acknowledgement, Timestamp)

- For these metrics, the Metric Browser uses the terms *Connection* and *Flow* interchangeably. See Flows, Links, and Connections.
- Network Agents do not collect metrics for individual Connections by default. The recommended workflow is to enable TCP Flow metric collection only when you need to diagnose a performance issue on an associated node or Connection. See Dynamic Monitoring Mode and Network Visibility.

Viewing TCP Flow Metrics

To view advanced network metrics, open the Metric Browser and navigate to Application Infrastructure Performance. The Metric Browser shows connection /flow metrics for the following aggregations (highlighted in red):

- **Application Infrastructure Performance >**
  - **Advanced Network >**
    - **Flows >**
      - **All Connections/Flows for Application**
  - *tier-name*
    - **Advanced Network >**
      - **Flows >**
        - **Call from** <*tier*> **to** <*service*>
          - **Call from** <*node*> **to** <*node-or-service*>
            - **Individual Connection/Flow** (<*node*> to <*service*>)
          - **All Connections/Flows for Network Link** (<*tier*> to <*service*>)
        - **All Connections/Flows for tier**

TCP Flow Metric Descriptions

Metric Name	Description and Notes	Default Monitoring Mode
# Client Limited	The number of TCP window updates sent by a client node indicating that it is receiving data at a faster rate than the application can process	KPI
# Client Zero Window	If a client's TCP buffer is full, it sends Zero as the TCP receive window size to indicate that it cannot receive more data. The data transfer then stops until the client can process the data in its buffer. A high rate of Client TCP Zero Window messages indicates a problem with either: • The TCP configuration on the node, or • The client-side application using that flow	KPI
# Connection Errors	The sum of Syn Resets + Syn Blackholes + TCP Resets - Established	KPI
# Connection Requests	The total number of connection requests (successful and unsuccessful) sent during the selected time window	KPI
# Current Established Connections	The number of established (setup) flows/connections	KPI
# Data Retransmits	Number of TCP data packets that were retransmitted for all TCP Connections	KPI

# Delayed Acks (Data Piggy Back)	The average number of times a receiving node sent an ACK (acknowledgment) by "piggybacking" the ACK onto another message	Advanced
# Delayed Acks (Timeouts)	The average number of times a receiving node sent an ACK (acknowledgment) because the Delayed ACK timer expired This is a "worst-case" delay for the Delayed ACK algorithm and occurs most often when Nagle's algorithm and Delayed ACK are enabled on the sending and receiving node, respectively. A high rate of delayed ACKs may contribute significantly to the average Latency on a Connection.	Advanced
# Errors	The number of TCP messages sent indicating an error in setting up the connection (SYN errors), or tearing down the connection (FIN errors)	KPI
# Fin Errors	The number of errors generated while tearing down the connections (TCP FIN errors). Many connections in FIN wait states can cause delays in creating new connections.	KPI
# Flows (<1KB) # Flows (1k - 10k) # Flows (10k - 100k) # Flows (100k-1MB) # Flows (1MB - 10MB) # Flows (>10MB)	Use these metrics to analyze the distribution of flows by throughput	Advanced
# Flows - Handshake (1SD) # Flows - Handshake (2SD) # Flows - Handshake (3SD)	The number of TCP flows with connection-setup times that are 1, 2, or 3 Standard Deviations outside (higher or lower than) the average for that connection group Under ideal conditions, all flows should be within 1SD of the average	Advanced
# Flows - Lifetime (1SD) # Flows - Lifetime (2SD) # Flows - Lifetime (3SD)	The number of TCP flows with lifetimes that are 1, 2, or 3 Standard Deviations outside (higher or lower than) the average. Under ideal conditions, all flows should be within 1SD of the average. 2SD or 3SD flows indicate inconsistent flow treatment along the network path. These inconsistencies can occur when a network service decreases the available bandwidth intentionally (*bandwidth throttling*) or prioritizes some types of traffic over others (*traffic shaping*). Short-lived connections, even if they are intermittent, may indicate an issue worth investigating. The more short-lived connections get generated, the more resources are spent setting up and tearing down these connections.	Advanced

# Flows - RTT (1SD) # Flows - RTT (2SD) # Flows - RTT (3SD)	Number of flows whose Round Trip Times are 1, 2, or 3 Standard Deviations higher than the average for all flows in the parent group. Under ideal conditions, all flows should be within 1SD of the average. High-RTT connections, even if they are intermittent, may indicate an issue worth investigating. If the average response time for an eCommerce web app is two seconds (acceptable), but 5% of transactions are 20 seconds or higher (not acceptable), this can result in a significant number of unhappy customers and lost revenue.	Advanced
# Flows - TCP Data Rxmt	Number of flows within which any packet was retransmitted. Retransmissions are an indication of packet loss.	Advanced
# Flows - TCP Resets	The average number of flows that were closed by a TCP Reset	KPI
# Flows - w/o TCP SACK	Number of flows with the TCP Selective Acknowledgment (SACK) option disabled. With SACK enabled, a receiver can send SACK packets to acknowledge receipt of multiple data packets in the case of lost segments. This improves network performance by reducing the number of retransmissions. With SACK disabled, the receiver must resend all the packets after the last lost segment, even if they were received by the peer.	Advanced
# Flows - w/o TCP Timestamp	Number of flows with the TCP Timestamp option disabled. Use this option to calculate more accurate Round Trip Times.	Advanced
# IP Fragment Count	The total number of IP fragments attributes to this TCP connection group. A high level of fragmentation indicates network issues which can severely affect application performance.	KPI
# Loss Pkts	The total number of packets that were lost (sent but never received)	Advanced
# Nagle Delays	The number of times a message-send event was delayed because the sending node had to wait for previously-sent data to be acknowledged (ACK'd). This occurs most often when Nagle's algorithm and Delayed ACK are enabled on the sending and receiving node, respectively.	Advanced
# PIE Events	Use Performance Impacting Elements (PIE) to identify the location of actual or potential bottlenecks: • Client Limited and Client Zero Window events indicate a possible problem on the client node • RTOs indicate a possible problem on the network path between two nodes • Server Limited and Server Zero Window events indicate a possible problem on the server node	KPI
# Retransmission Timeouts	Retransmission Timeouts (RTOs) indicate network packet loss which results in retransmission of data when the TCP retransmission timer expires (Timer to make sure data is ACK'ed) • This timer varies from 200ms-3 sec by default (different for each operating systems and their versions) • It causes severe performance degradations because a considerable amount of time is wasted resending lost data • TCP falls back to the "Slow Start" phase impacting the performance even more	KPI
# SACK Retransmits	Average number of packets that were retransmitted due to a SACK message that indicated an unreceived packet	KPI
# Server Limited	The number of TCP window updates sent by a server node indicating that it is receiving data at a faster rate than the application can process	KPI
# Server Zero Window	If a server's TCP buffer is full, it sends Zero as the TCP receive window size to indicate that it cannot receive more data. The data transfer then stops until the server can process the data in its buffer. A high rate of Server TCP Zero Window messages indicates a problem with either: • The TCP configuration on the node, or • The client-side application using that flow	KPI
# Syn Blackholes	The number of connection attempts that went unanswered and resulted in a failure. Syn blackholes can severely impact application performance.	KPI
# Syn Resets	The number of connection attempts that were explicitly refused by the other host. Syn resets can severely impact application performance.	KPI

# TCP Resets - Established	The average number of times an established TCP flow was reset	KPI
# TCP Resets - Fin	The average number of times a TCP flow was reset while it was in the process of getting closed	KPI
# TTL Changes (1 - 2 hops) # TTL Changes (3 - 4 hops) # TTL Changes (>=5 hops)	The number of routing-hop changes experienced by this connection. Frequent variations in routing-hop changes indicate routing problems in the network and can severely affect app performance. Under ideal conditions, the number of hops should be consistent.	Advanced
Avg # TTL Hops	The average number of routing-hop changes experienced by this connection	Advanced
Delayed Acks Lag (usec)	The average amount of lag that Delayed ACKs are adding to the overall Latency of the Connection	Advanced
Initial RTT (usec)	Round trip time for the initial two SYN packets (between SYN and SYN-ACK, or SYN-ACK and ACK depending upon whether the Agent is running on client or server)	KPI
Latency - RTT (usec)	Average round-trip latency (from packet transmission to acknowledgment) for all packets	KPI
Lifetime (usec)	Average lifetime (from initial setup to final teardown) for TCP sessions in the Connection group	KPI
Loss (per mille)	The number of packets lost per 1000 packets sent. "Per mille" is a percentage with one additional digit of precision.	KPI
Nagle Delays Lag (usec)	The average amount of lag that Nagle delays are adding to the overall Latency of the Connection	Advanced
Pkts per Sec	Average rate of packets sent and received	KPI
Rx Pkts per Sec	Average rate of packets received	Advanced
Rx Throughput (BPS)	Average rate of bytes received	Advanced
TCP Handshake (usec)	Round trip time for the initial three-way connection setup for all flows in the parent group: 1. SYN (client --> server) 2. SYN-ACK (client <-- server) 3. ACK (client --> server)	KPI
Throughput (BPS)	Throughput (bytes per second) for the application of interest on all TCP sessions	KPI
Tx Pkts per Sec	Rate of packets sent	Advanced

Tx Through put (BPS)	Rate of traffic received	Advanced

FAQs for Network Visibility

General

- Why are there no connection metrics for a tier, node, or network link?

By default, Network Agents do not collect connection metrics. The recommended workflow is to identify the link with the network issue and configure the relevant agents to collect metrics for the relevant connections. See Dynamic Monitoring Mode and Network Visibility.

- The Network Agent cannot register with the Controller. What should I do?

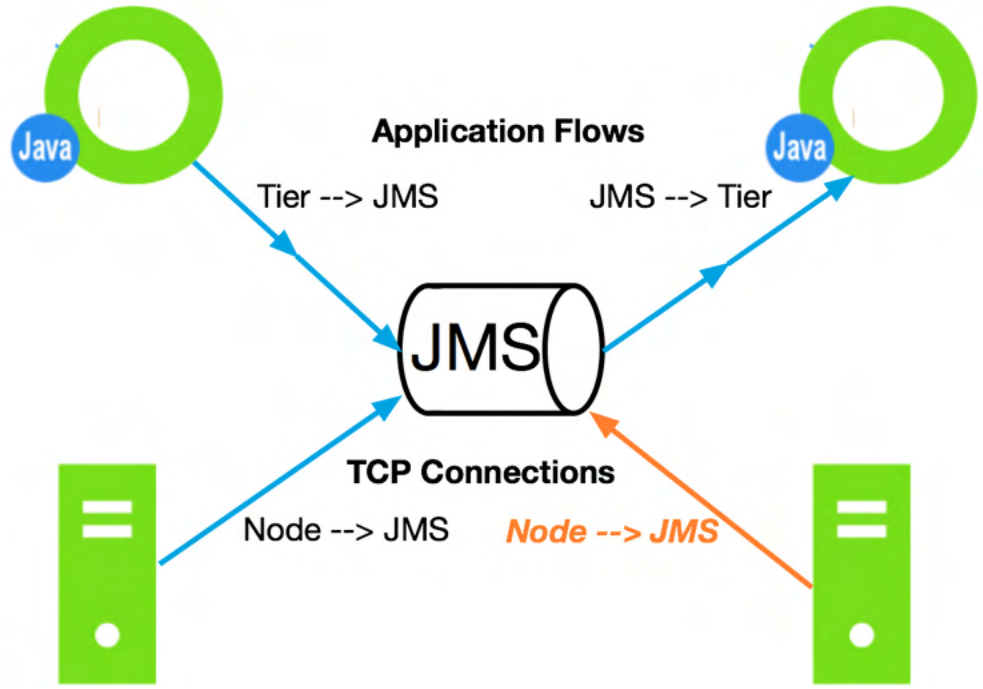
If a Network Agent cannot register with the Controller:

  - Check that the user account has a Network Visibility product license.
  - If the user account has license rules defined, verify that the correct number of license units have been allocated.
  - To change the number of allocated units in a rule:
    - Go to **Controller Settings > License > Rules**.
    - Edit the **License Rule** of interest. (There may be only one License Rule, named **Default**.)
    - In **General**, set the **Allocated Units** field for the Network Visibility license and apply the change.

- In some cases, I see that an application flow for a JMS queue goes in one direction but a TCP connection used by that queue goes in the opposite direction. Why is that?

Typically, the network links and TCP connections used by an application flow have the same direction (source destination) as the flow itself. You may notice different directions, however, if two tiers transfer data through a JMS queue. In some JMS implementations, the individual nodes in each tier initiate the TCP connection to the queue, so the direction is always: node (source) queue (destination).

Some of these connections may be used by an application flow in the opposite direction: queue (source) tier (destination).



- How do I change the Network Agent communications port?

When you start the Network Agent, the `appd_netmon` process spawns the `appd-agent` process. By default, these two processes communicate over TCP port 3892. If this port is already in use, you should see a log message in one or both files. To configure the Network Agent to use a different port:

1. Use the `netstat` command to verify that the new port is not in use.
2. Update the Network Agent:
   a. Use a text editor to open this file: `<network_agent_home>/conf/agent_config.lua`.
   b. Set the `port` option to the new TCP port number (under `webserver_config`).

```
webserver_config = {
   port = <new-port-number>,
   request_timeout = 10000,
   threads = 2,
  }
```

   c. Save the file and restart the Network Agent.

3. Update the App Agent:
    a. Use a text editor to open this file: <*app_agent_home*>/<*version-number*>/external-services/netviz/netviz-service.properties.
    b. Set the `netviz.agent.api.service.port` option to the new TCP port number.
    c. Save the file and restart the App Agent.

- The application I want to monitor uses TCP port 32768 or higher. How do I configure the Network Agent to monitor this port?

1. Use a text editor to open this file: <*network_agent_home*>/conf/agent_config.lua
2. If you plan to monitor any application or service that uses any TCP ports higher than 32767, uncomment the `application_service_ports` block and specify these ports as a comma-separated list in the `ports` option:

    Original:

    ```
    --[[
    application_service_ports = {
    ports = "",
    }
    --]]
    ```

    Edited:

    ```
    application_service_ports = {
    ports = "40000, 41000",
    }
    ```

- How can I replace the `AppDynamicsNetMQ.dll` with another version of NetMQ?

1. Download the `NetMQ.dll` version and its dependencies.
2. ILMerge them into one `dll` file named `AppDynamicsNetMQ.dll`.
3. Include the `AppDynamicsNETMQ.dll` file in the .NET Agent home directory, such as `c:\Program Files\AppDynamics\AppDynamics .NET Agent`.
4. Restart the monitored application.

- Why does running the Dynamic Service increase the DNS resolution time of the application?

If you specify `FQDN/hostname` as a value to define the Network Visibility property `netviz.agent.host.address` in the `netviz-service.properties` file, the Dynamic Service resolves the FQDN/hostname to its respective IP address frequently, resulting in increased resolution time.

To resolve this problem, use IP Address as a value to define the Network Visibility property `netviz.agent.host.address`. Alternately, you can also increase the value of `networkaddress.cache.ttl` depending on your application requirements.

- Are there any additional fully qualified domain name (FQDN) configurations for the `agent_config.lua` file?

```
ip_config = {
expire_interval = 20,
retry_count = 5,
}
```

Refresh Timeout: `expire_interval` defines when an IP in the .NET Agent IP cache needs to be re-resolved for FQDN or flushed if not associated to any flowgroup. The expire interval ranges from 1 to 30 mins (maximum).

Retry Count: `retry_count` defines how many times an IP tried to resolve for FQDN. Every resolution happens after `expire_interval` time. you can update the default value in situations where the DNS record for the IP is created and there is a lot of churn in the IP to FQDN mappings.

Logs

- How do I resolve the error, "`ERROR MsgZmq::Bind: zmq_bind failed: File name too long`"?

This error occurs when the file name is too long. The complete path to the file is included in the file name.

To resolve this error, move the Network Agent to a smaller absolute path.

1. Create a soft (symbolic) link for the longer directory name: `$ ln -s /long/dir/name /short/dir/name`.
2. Go to the short directory: `$ cd /short/dir/name`.
3. Run the install.sh script: `$ sudo ./install.sh`.
4. Change the LD_PRELOAD parameter in application to use the library from the short directory name: `$ LD_PRELOAD=/short/dir/name /lib/appd-netlib.so <command to start app>`.

- Why do I see the error, "`ERROR cw_flowgroup_uri_cb: Failed to write data on connection`"?

This error occurs when the Network Agent supports a large number of AppServer Agents. If this error occurs frequently, restart the Agent using an increased number of thread counts in `webserver_config` in `agent_config.lua` config file.

- Why do I see the error, "`ERROR ip_flowgrp_lookup: flowgrp alloc failed`"?

Connections (`Source IP : Source Port : Destination IP : Destination Port`) in Network Agent are represented as flows. Aggregating these connections on a source port provides flow group, represented as (`Source IP : Destination IP : Destination Port`). The Network Agent can monitor only a certain number of flow groups. This error occurs when the maximum number of flow groups is reached and the Network Agent cannot monitor any new incoming flow groups.

- Why do I see the error, "`DEBUG adns_resolve: ip resolve error: Name or service not known`"?

AppDynamics resolves IP addresses to provide fully qualified domain name (FQDN) information to the Controller for the IP addresses visible in the system. This error occurs when some IP addresses cannot be resolved.

Network Visibility Concepts

The these concepts are central to Network Visibility:

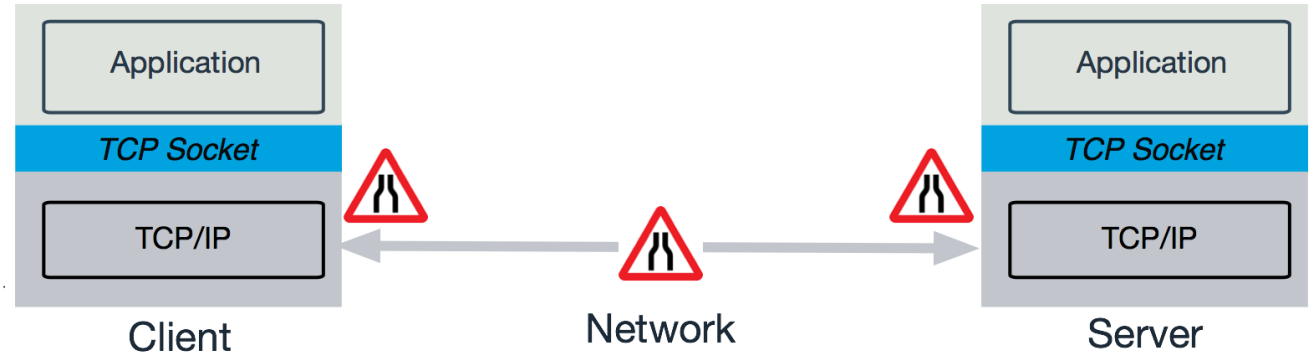
- Network Visibility can identify two types of performance bottlenecks:
  - Network-performance bottlenecks
  - App/Network-interaction bottlenecks (for example, how an application tier or node uses network resources)
- The Network Dashboard shows the network-level view of a monitored application. You can quickly identify network elements with performance issues (red/yellow links or tiers), view Key Performance Indicators for each network element, and troubleshoot individual elements (right-click an element and select **View Metrics**).
- Network Agents can detect load balancers and TCP endpoints, which appear in network flow maps (but not application flow maps), and sometimes function as traffic sources and destinations.
- Network Visibility can collect metrics for individual application flows, TCP connections, and network links.
- To support in-depth troubleshooting, you can configure Network Agents to collect detailed diagnostic metrics for nodes and connections with performance issues.

Identifying Performance Bottlenecks

Performance Bottleneck Categories

Network Visibility can pinpoint performance bottlenecks down to a specific tier, node, or network path. Either of these bottlenecks can slow down an application significantly:

- Application/Network bottlenecks
  Many bottlenecks identified as "network issues" actually have their root cause defined by how an application or a server uses network resources. Network Visibility can detect many different types of app/network bottlenecks, as well as bottlenecks due to TCP socket issues on a specific node.
- Network bottlenecks
  A network bottleneck indicates that a specific section of the network cannot process the application loads between the source and destination nodes. Symptoms of a network bottleneck include a high rate of packet loss or high round-trip times (RTTs) between two nodes.



Load Balancers and TCP Endpoints

You may notice objects in a Network Flow Map that do not show up in an Application Flow Map. This is because these Network Flow Map objects represent devices that are detected by the Network Agents, and are relevant to the monitored application, but the detected devices do not have Network Agents installed.

Load Balancers

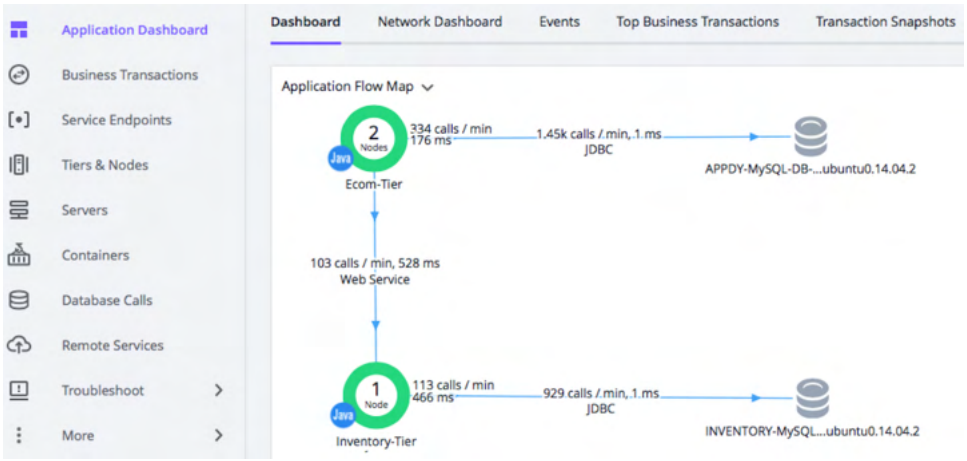
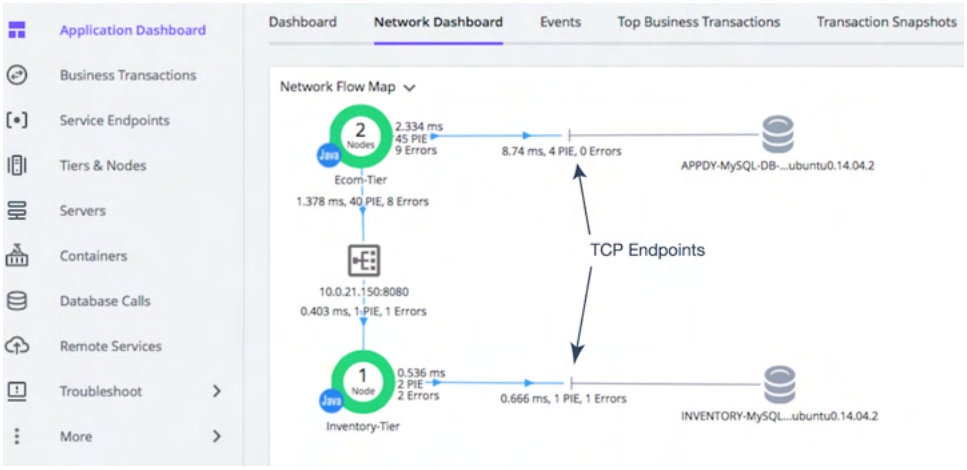
Network Visibility detects load balancers (LB) automatically, and displays them in Network Flow Maps. Load balancers seek to optimize the distribution of network traffic and often hide internal IP addresses from external nodes. Application Flow Maps do not show load balancers because these devices do not function as application nodes.

An intermediate load balancer splits an application message into two separate connections: node A connection to a load balancer, and the load balancer connection to node B (node A -> LB, and LB -> node B). This additional layer of detail in the Network Browser makes it easier to identify network segments that have specific performance issues.

TCP Endpoints

The Controller uses TCP endpoints to represent flows where the monitoring vantage point is from one end. The Network Agent monitors the connection between a monitored node and a TCP endpoint. However, the Network Agent cannot determine if this endpoint is on another node, or on an intermediate device (such as a load balancer) that splits the connection in two.

These diagrams show two different views of the same application:

The **Application Dashboard** shows two tiers connecting to two MySQL instances. The Application Flow Map shows application messages between monitored nodes that are directly visible to App Agents.	
The **Network Dashboard** shows TCP endpoints on links between tiers and nodes. The Network Flow Map shows Network Agents installed (left) and nodes that the Network Agents cannot see directly (right).	

This example shows how a Network Agent monitors a connection between an Inventory-Tier node (left), and an Inventory Database service. A TCP endpoint divides the link into:

1 A colored line representing the connection visible to the Network agent, and

2 A gray line representing a connection that is not instrumented by Network Visibility. A gray line connection link without Network Visibility indicates one of two things:

- The TCP endpoint and an Inventory Database service are *on the same device* with an intermediate device that splits the connection in two.
- The TCP endpoint and an Inventory Database service are *on two devices* using a separate connection that does not allow for network visibility.

For these reasons, gray line connection links do not have associated metrics.

Flows, Links, and Connections

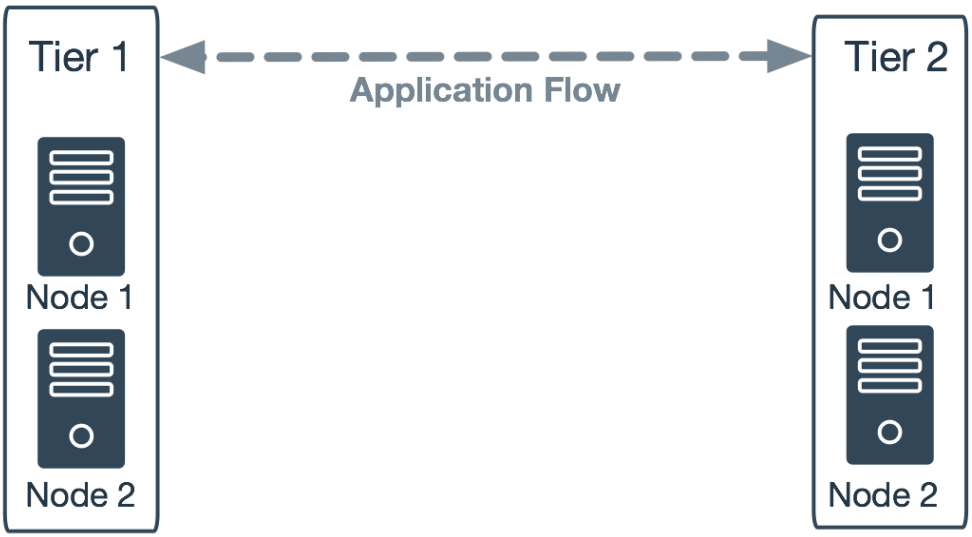
Network Visibility can collect metrics for these object types:

- Network links between tiers, or between tiers and load balancers/TCP endpoints
- Connections between nodes, or between nodes and load balancers/TCP endpoints

Troubleshooting a performance issue in Network Visibility often involves drilling down from an application issue (flow) to a network issue (link) to the root cause (connection). See Workflows and Example Use Cases.

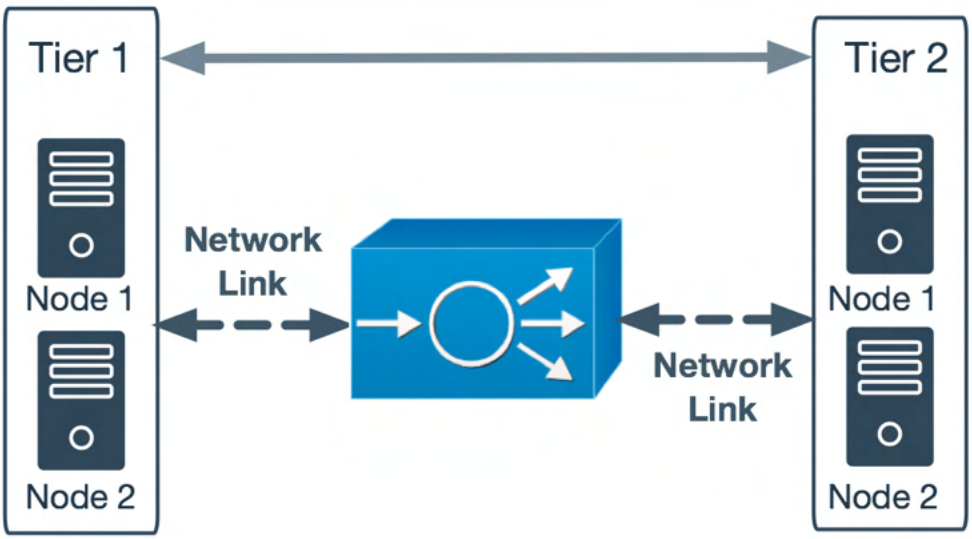
Application Flows

App Agents collect metrics for *application flows* between tiers, as shown in the Application Flow Map. This diagram shows the calls, transactions, and messages for application *x* between tier 1 and tier 2.



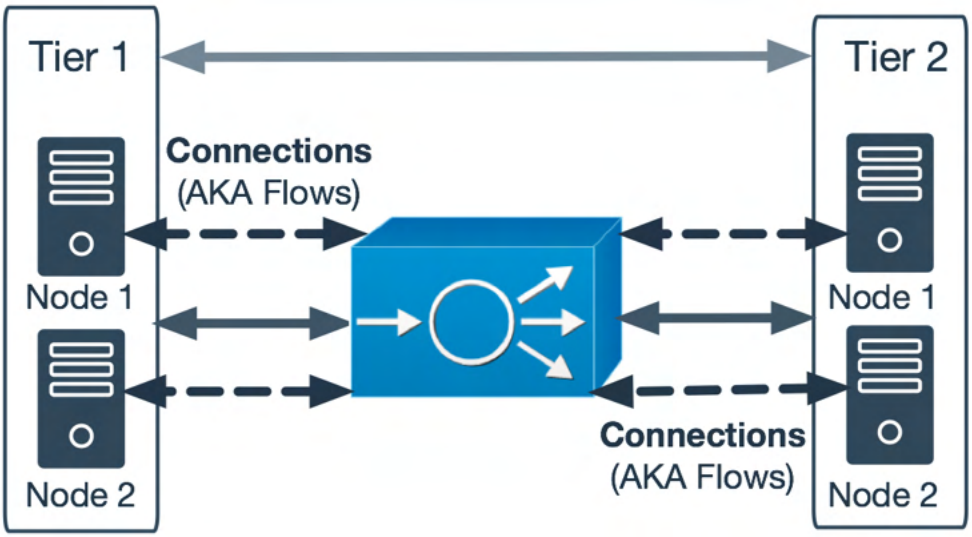
Network Links

Network Agents collect metrics for *network links*, which you can view in the **Network Dashboard**. A network link is the set of all traffic between two tiers, or between a tier and an intermediate load balancer, or TCP endpoint. If an application flow transports traffic through a load balancer, Network Visibility creates a link on each side and calculates metrics for each link. This diagram shows the network traffic for application *x* sent and received by the two tiers through a load balancer.



TCP Connections

Individual nodes send and receive traffic over connections. In Network Visibility, a c*onnection* is a set of all traffic with the same source IP address, destination IP address, destination port, and protocol. This diagram shows the individual connections used by the four nodes to send and receive traffic for application *x* through the load balancer.



Dynamic Monitoring Mode and Network Visibility

Like Server Visibility Agents, Network Agents support Dynamic Monitoring Mode (DMM). Rather than have all Network Agents report all metrics all the time, you can run each Agent in one of three modes:

- KPI Mode – Network KPI metrics for all monitored objects (application flows, tiers, nodes, and network links) except individual connections
- Diagnostic Mode – Network KPI metrics for all monitored objects, including connections
- Advanced Diagnostic Mode – All network metrics for all monitored objects, including connections

Every Network Visibility metric has a default DMM class (KPI, Diagnostic, and Advanced Diagnostic). See Network Visibility Metrics

By default, all Network Agents run in KPI mode. The recommended workflow is to:

1. Run all Network Agents in KPI mode.
2. When you notice a performance issue on a specific node or network link, increase the metric level on the associated Network Agents to Diagnostic, and collect KPI metrics for the connections.
3. Based on the connection KPIs, identify the connections with performance issues.
4. To troubleshoot an individual connection, increase the metric level on the associated Network Agents to Advanced Diagnostic, and collect advanced metrics for the connection.
5. When the issue is resolved, reset the Agents back to KPI mode.

Change the DMM on a Network Agent

1. Click the gear icon () in the top-right corner of the Controller page, select **AppDynamics Agents**, and go to the Network Visibility Agents table.
2. Select the Agents of interest, right-click, and select **Change Dynamic Monitoring Mode**.

The Network Agent retains its DMM setting even if the Agent is stopped and restarted. See Managing Network Agents in the Controller.

Connection Diagnostics and Network Bottlenecks

Network Agents can collect an extensive set of individual connection metrics within a network link. Network Visibility can detect these TCP performance issues:

- The data-receive window is too small (or zero) on the client or the server, which slows down the transfer of data.
- One or more servers are experiencing errors setting up or tearing down the connection for an individual TCP session.
- The client and server take a long time to set up an individual session.
- The application is using many short-lived connections. TCP is most efficient when long, stable connections are used.
- Some TCP sessions have unusually high round-trip times (RTTs). When TCP is performing well, RTTs are stable and determined by the network path between two nodes.

The Network Agent does not collect any connection metrics in KPI mode (the default setting). To diagnose a node or network path that is monitored by an Agent, you can change the Dynamic Monitoring Mode on a Network Agent.

Example Workflow

After you complete the initial setup, you can set the Dynamic Monitoring Mode on individual Network Agents, as needed. This is an example workflow:

- The DevOps team for a large enterprise monitors its IT infrastructure using Network Agents on critical servers. All Agents are initially set to KPI mode.
- The Network Dashboard shows a spike in latency on a network link between tier-A and tier-B.
- A DevOps team member:
    - Sets the Network Agent DMM on the tier-A and tier-B servers to Diagnostic.
    - Collects KPI metrics for the connections between the tier-A and tier-B nodes.
    - Identifies one high-latency connection between nodes TA-N1 and TB-N3, based on the Connection KPIs. The KPIs for all other connections are within acceptable bounds.
    - Sets the Network Agent DMM on TA-N1 and TB-N3 to Advanced Diagnostic, and the DMM on all other Agents to KPI.
    - Notes a set of spikes in Nagle delays on this connection. These spikes correspond to the latency spikes noticed on the network link.
    - Reconfigures TCP on the two nodes, and monitors the connection. The Nagle and latency spikes no longer occur.
    - Resets DMM on the TA-N1 and TB-N3 Agents back to KPI.

Workflows and Example Use Cases

Network Visibility makes it easy to answer the question: *Is the network impacting my application's performance?* This page describes the high-level troubleshooting workflows and includes several examples of how to use these workflows to solve real-world network problems.

Application Troubleshooting

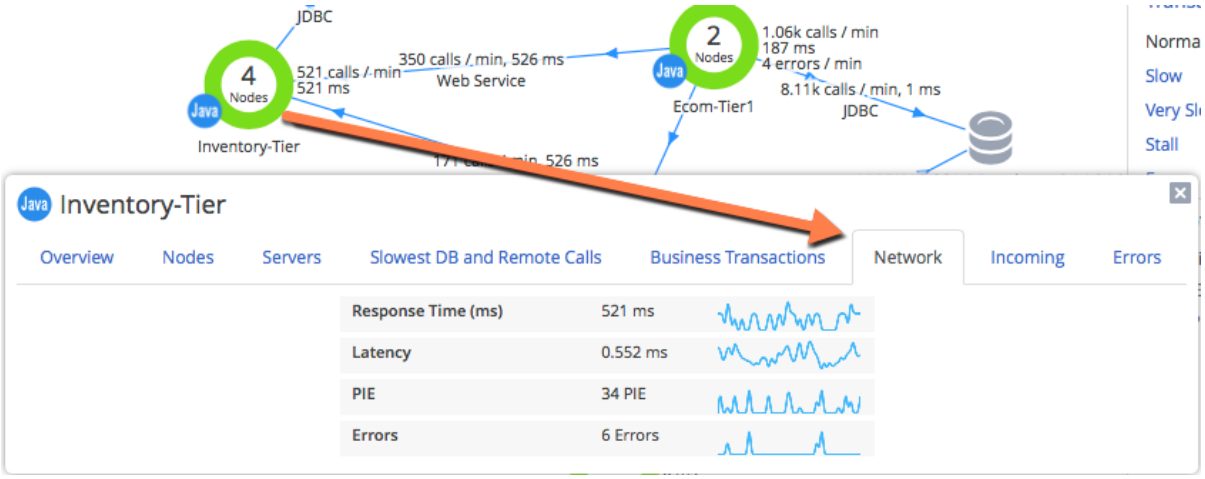
From the Application Dashboard and I see slow, very slow, error, or stalled transactions. Are network conditions to blame?

- Application Dashboard: Are network conditions to blame for the issue?
- Network Dashboard: Which network tiers and links are associated with these network conditions?
- Tier Dashboard: Which network metrics on this tier correlate with my outlier transactions?
- Network Dashboard (Advanced Diagnosis): On which TCP connections are these network issues occurring?

Application Dashboard: Are network conditions to blame for the issue?

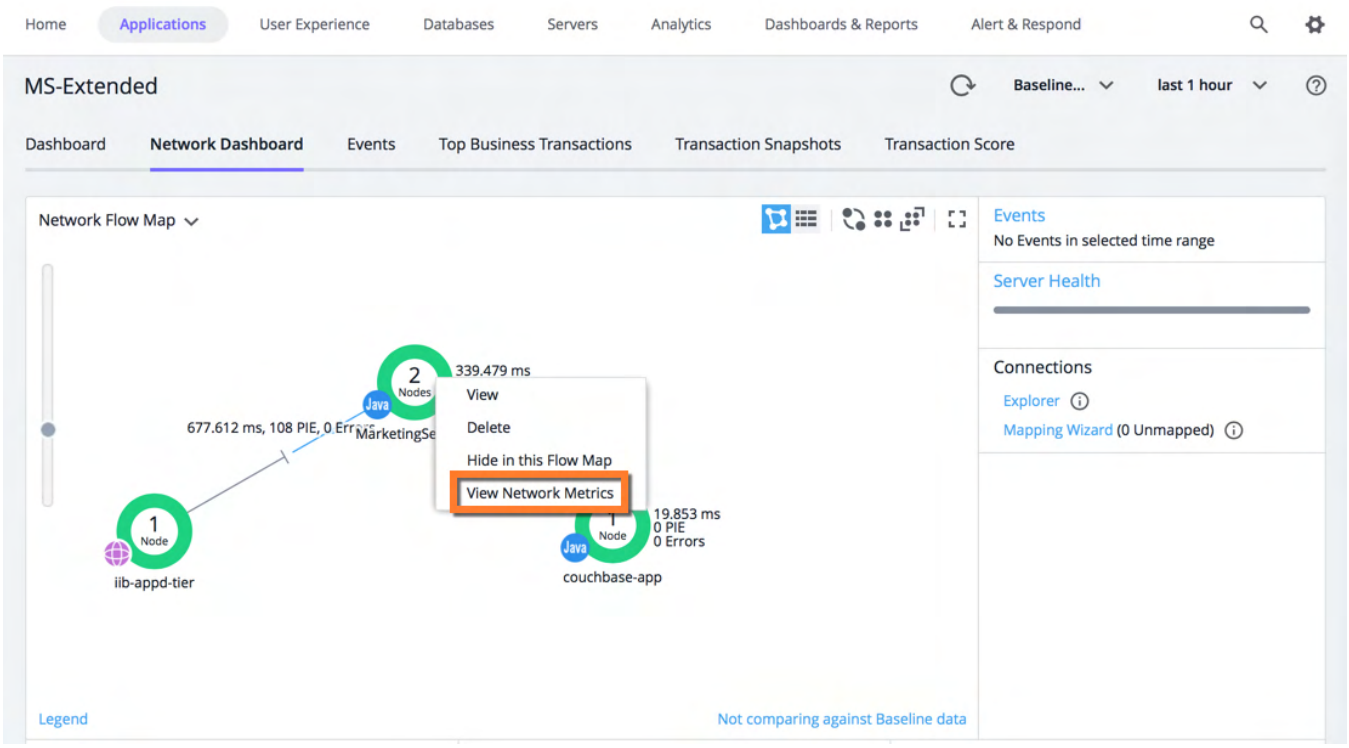
Network Agents collect network-performance metrics for each application tier. If you notice performance issues in the Application Flow Map, open a popup and click Network tab. Do you see correlations between the Response Time and the other charts? If so, it indicates that the Response Time is affected by:

- TCP effects between the two tiers (Latency)
- Performance Impacting Events at the client tier, server tier, or the network path between the tiers (PIE)
- Errors setting up or tearing down TCP connections (Errors)



Network Dashboard: Which network tiers and links are associated with these network conditions?

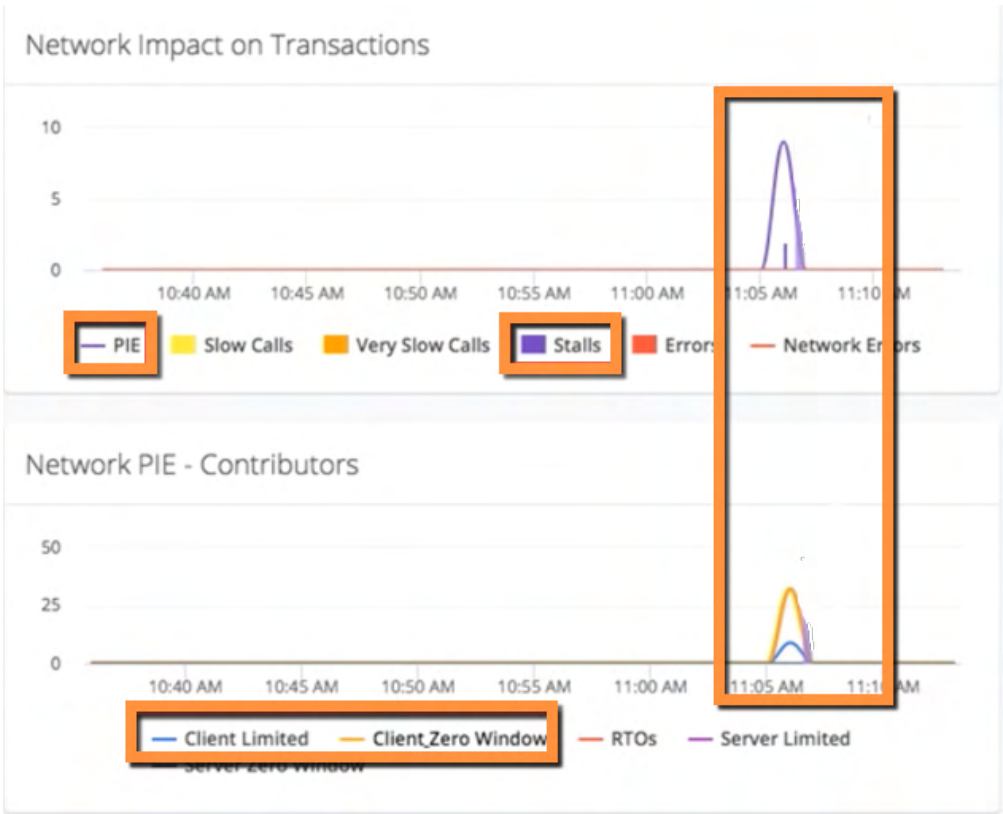
Go to the Network Dashboard, which shows the TCP and network context for the application. Each tier and link has a View Metrics right-click option that opens a context-sensitive dashboard. The dashboard charts make it easy to determine network root causes. Start from the top-left chart in each dashboard, then search for metric correlations in the other charts.



Tier Dashboard: *Which network metrics on this tier correlate with my outlier transactions?*

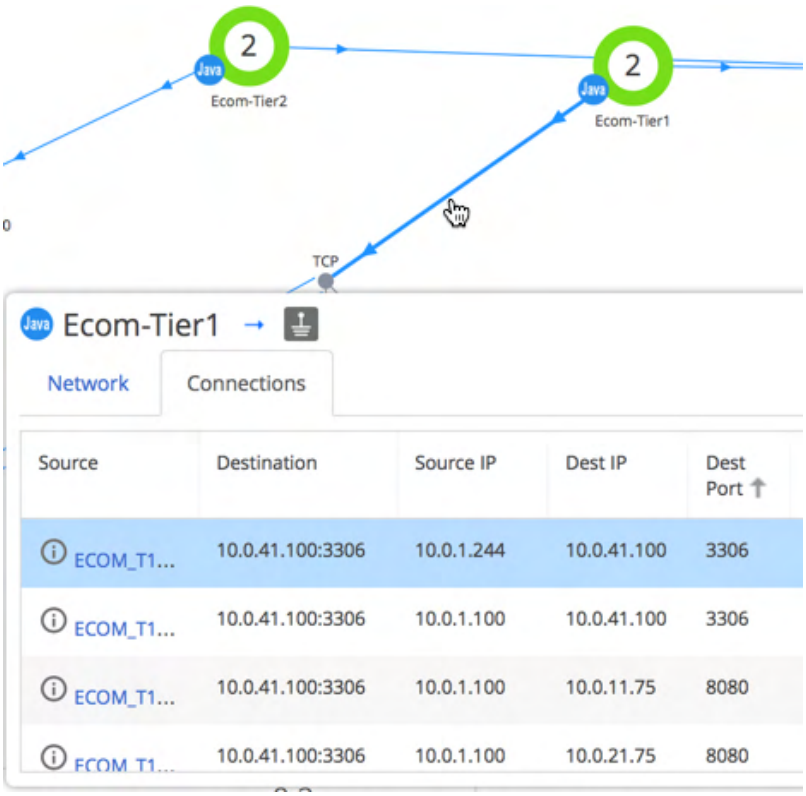
The link and tier dashboards both have a Network Pie - Contributors chart, which answers the question: *Where are these network conditions occurring – on the client tier, server tier, or the network link*?

In this example, the top chart shows that a spike in Stalled transactions is correlated with a spike in PIE (Performance Impacting Events); the bottom chart shows that these events occurred on the client tier. With this information, you can then locate other correlated metrics and identify the issues on the client tier.



Basic Diagnosis

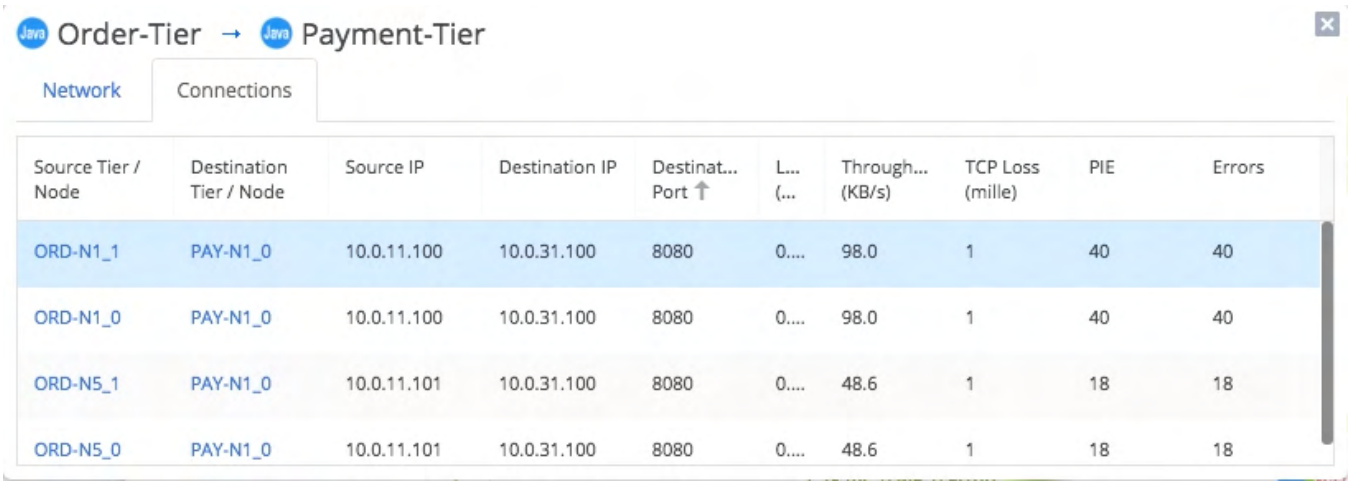
You now have a set of correlated network metrics and associated tiers/links. In many cases, this information is enough to identify and resolve the issue. You can contact your network team and say: *I see the following network issues on tier X, link Y, and so on, and these issues correlate with performance issues in my application.* You can send them a URL to the Network Dashboard and say: *Click on link Y and look at the connections. The issues are happening on one or more of these connections.* This often provides enough information for the networking team to identify and resolve the issue.



Network Dashboard (Advanced Diagnosis): On which TCP connections are these network issues occurring?

If you need to collect more information, you can drill down to the TCP connection(s) where the network conditions are occurring. Click the relevant network link and select **Connections** in the popup. Sort the table based on the KPI metrics for the monitored connections (Throughput, TCP Loss, PIE, Errors). You can then right-click a connection and view metrics in the Metric Browser, or in a context-sensitive dashboard.

> By default, Network Agents do not monitor individual connections. The recommended workflow is to enable connection monitoring only after you narrow an issue down to a specific set of links. See Dynamic Monitoring Mode and Network Visibility



Transaction Diagnosis

I'm troubleshooting a specific outlier (Slow, Very Slow, Error, or Stalled) transaction. Are network conditions to blame?

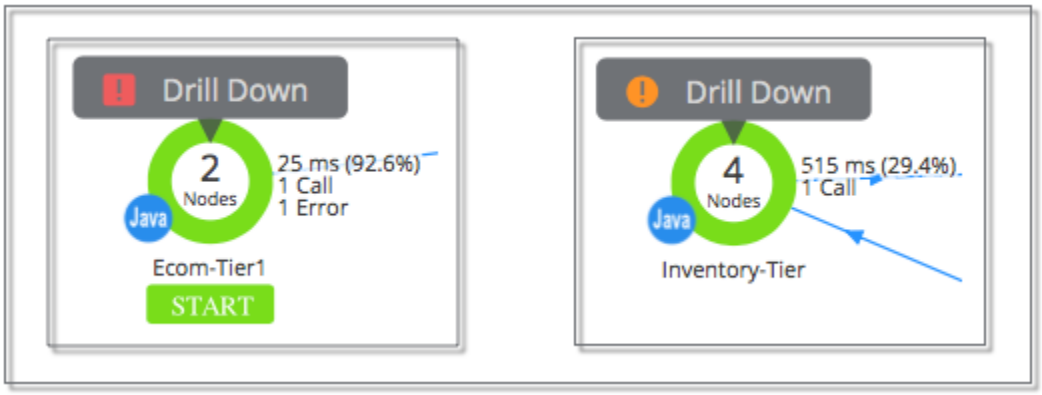
- Transaction Snapshots list: Is the network to blame for a specific slow, stalled, or error transaction?
- Network Dashboard for Transaction Snapshot: Are there any correlations between problematic transactions and network-performance issues around the time of this transaction?

Transaction Snapshots list: Is the network to blame for a specific slow, stalled, or error transaction?

You can diagnose the network performance for an individual transaction in the transaction viewer. Double-click a snapshot in a Transaction Snapshots list.

Transaction Snapshot viewer: Where (at which tier) are the delays, stalls, or errors occurring?

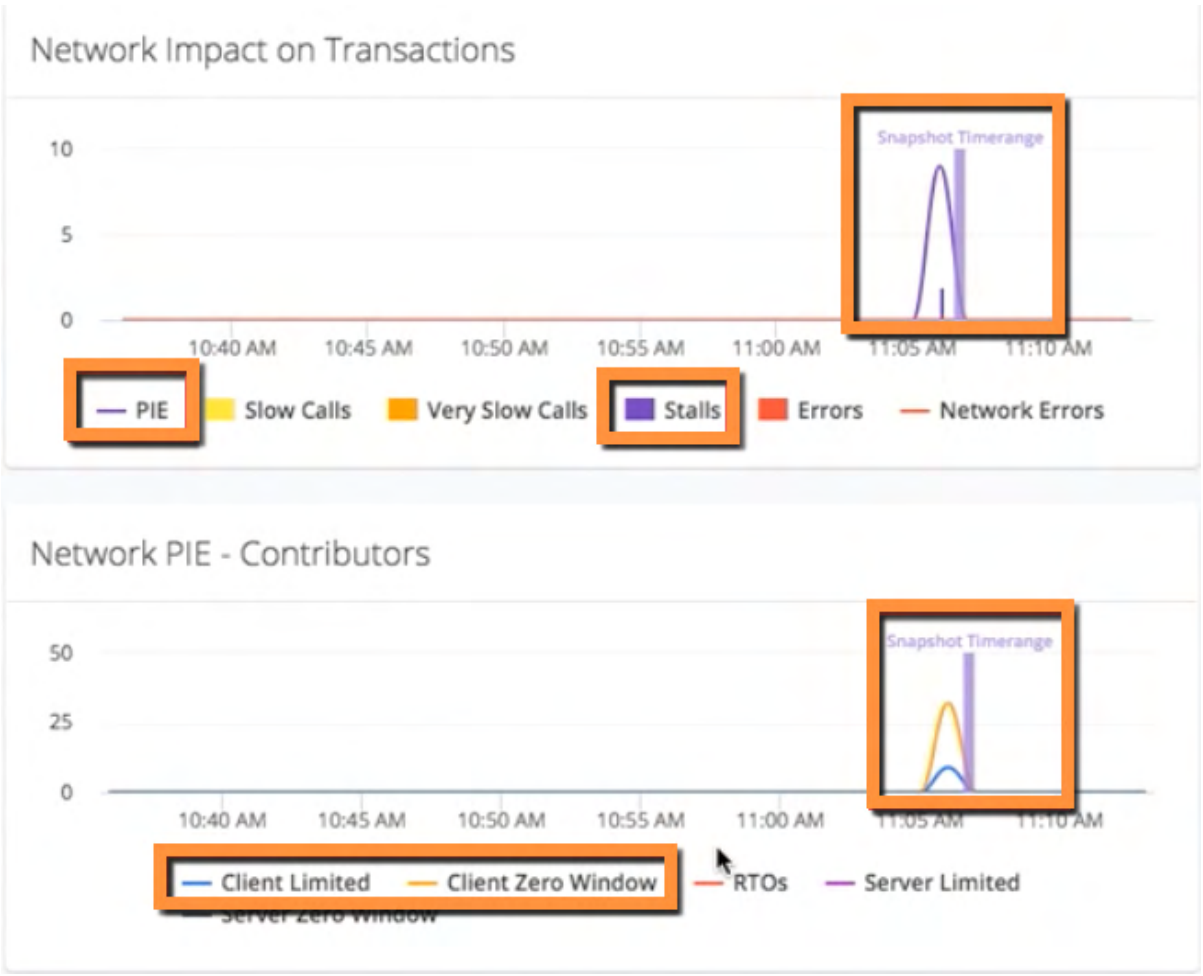
When the Transaction viewer appears, click **Drill Down** on the tier where the delays, stalls, or errors are occurring.



In the Tier Overview tab, click **Network**.

Network Dashboard for Transaction Snapshot: Are there any correlations between problematic transactions and network-performance issues around the time of this transaction?

The transaction dashboard shows the rate of transaction outliers, Performance Impacting Events, and Network Errors when the transaction occurred. Each chart has the Snapshot Time range highlighted. The Network Impact on Transactions chart (top left) highlights any correlations between transaction outliers, Network Errors, and Performance Impacting Events. You can then look for correlations in other charts and drill down to the root cause.



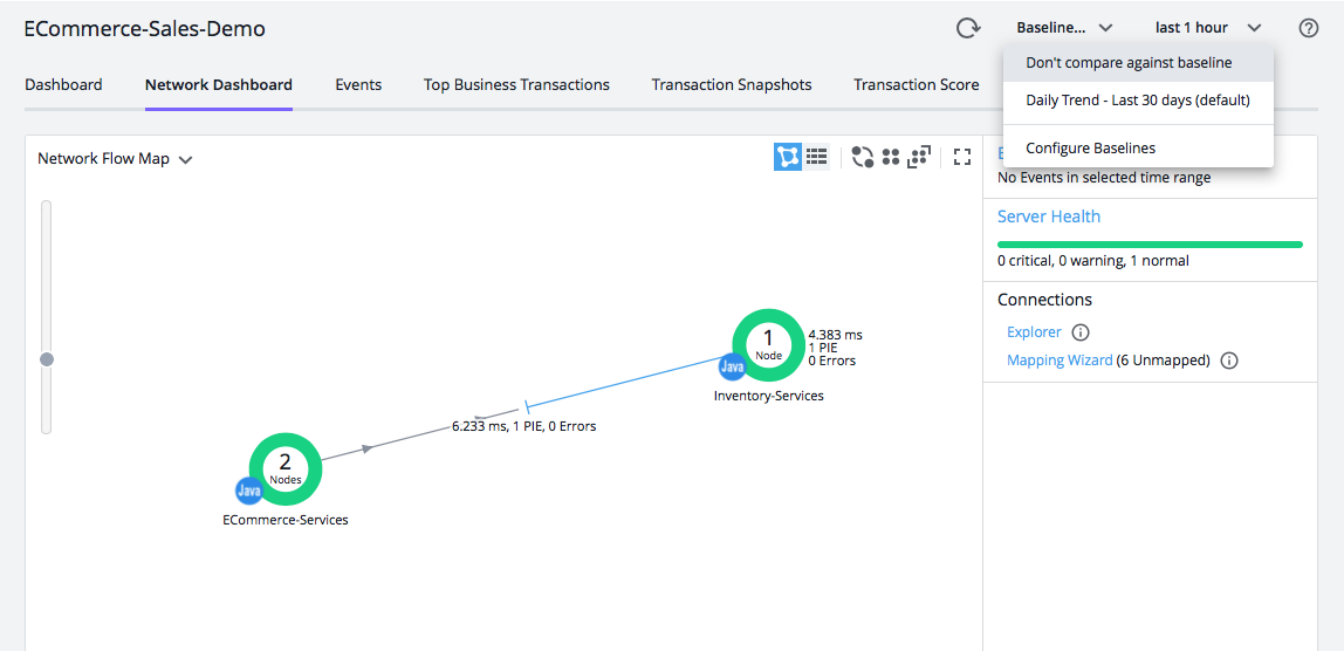
Network Health Check

Currently, I do not see any application issues, however, I would like to monitor my network and identify any potential issues before they start to affect my applications.

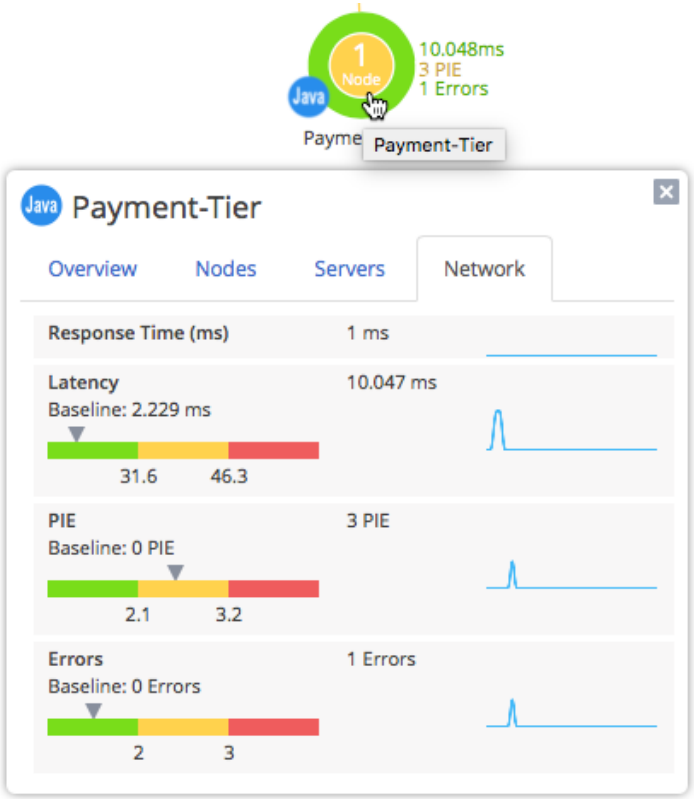
- Network Dashboard: Is any tier or link performing outside its baseline?
- Network Dashboard: Do any network links or connections have significant performance issues?

Network Dashboard: Is any tier or link performing outside its baseline?

To perform a quick network health check, go to the Network Dashboard and enable baselining. Elements with network KPIs that exceed the baseline for that element are colored yellow or red.

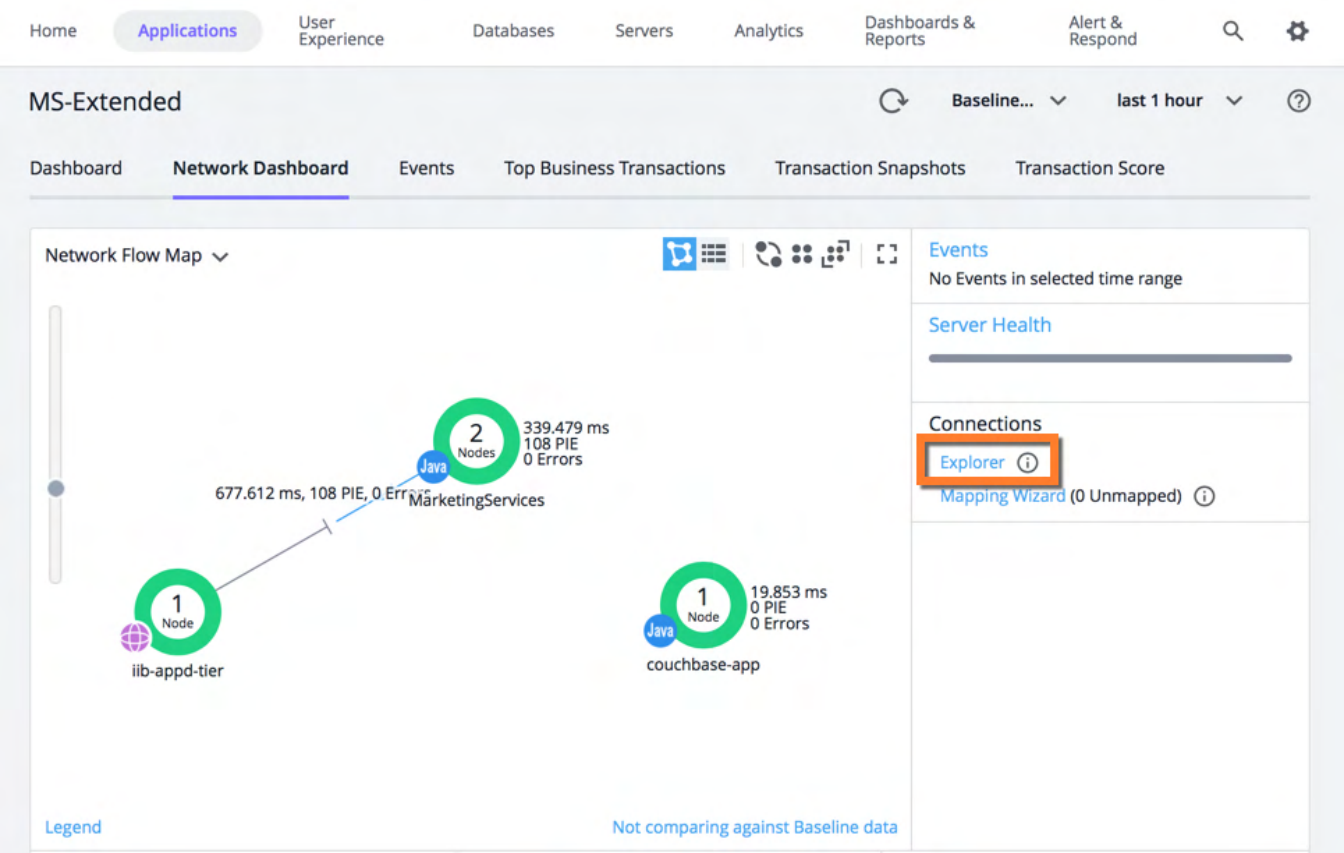


If the network KPIs for a tier or link are outside the performance baseline for that element, you can view baselines and outliers for that element in the Network tab of the popup. If an element has KPI outliers, you can also diagnose that element in a context-sensitive dashboard (right-click and select **View Metrics**).



Network Dashboard: Do any network links or connections have significant performance issues?

To see immediately if any links or connections have performance issues, open the Connections Explorer (click the **Explorer** link to the immediate right of the Network Flow Map). The KPIs of all network links and connections display in one table. You can sort the table based on KPIs or Errors, and expand each network-link to view KPIs for all connections within that link.



Example Use Cases

- Network Congestion - Example Use Case
- Packet Loss - Example Use Case
- TCP Port Exhaustion - Example Use Case
- TCP Window Bottlenecks - Example Use Case
- Traffic Throttling - Example Use Case

Network Congestion - Example Use Case

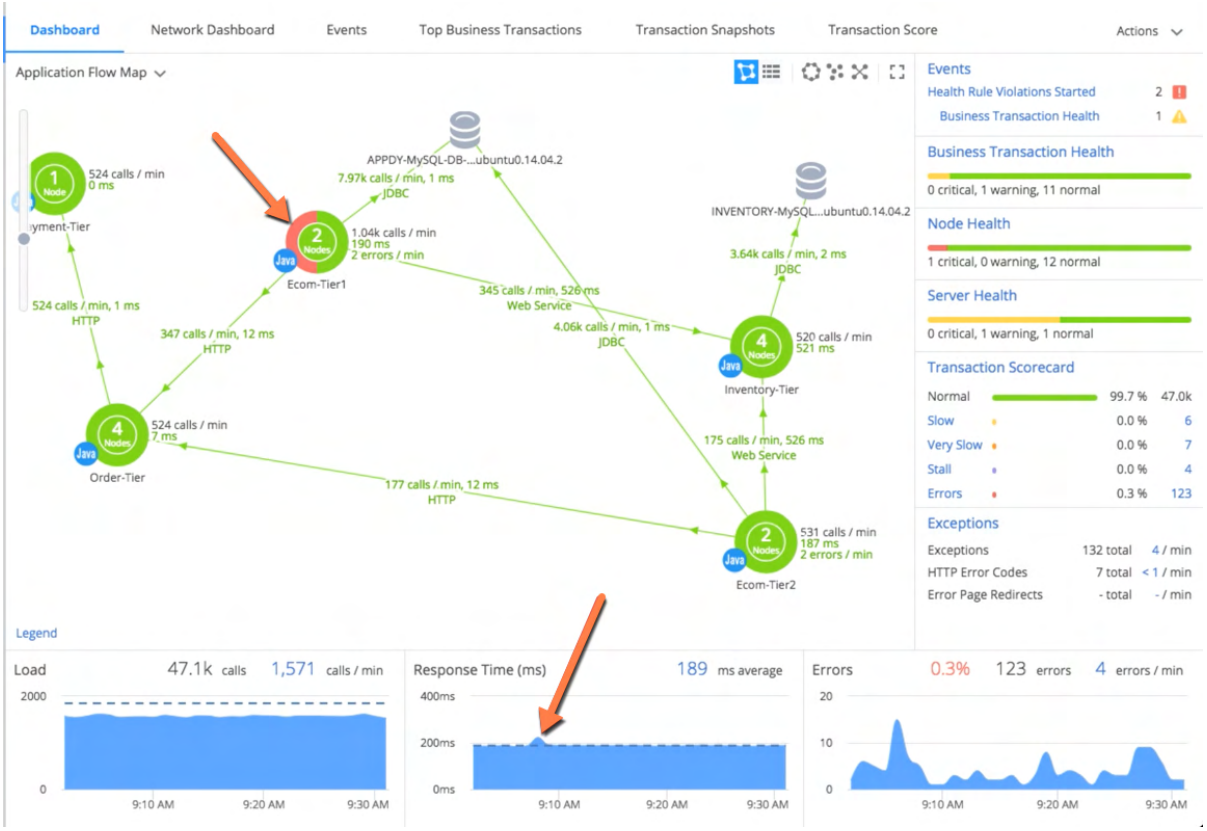
Network congestion may be caused by:

- A user may launch a "bandwidth-hogging" backup job during a period of peak network utilization.
- A server may drop packets at seemingly random intervals due to a TCP misconfiguration.

Traditional monitoring tools such as SNMP are good at detecting persistent network bottlenecks, but often miss intermittent bottlenecks. With Network Visibility, you can easily identify and diagnose intermittent network bottlenecks that affect application performance.

Application Symptoms

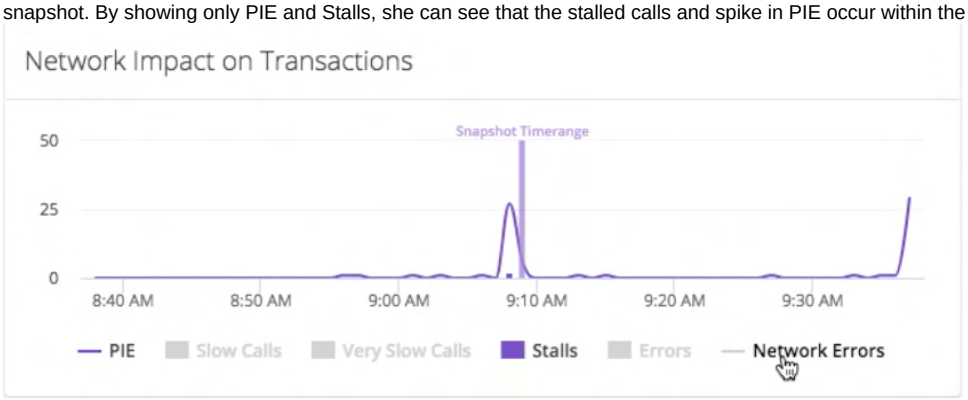
A DevOps engineer is responsible for monitoring a mission-critical app. One day, she opens the Application Dashboard and notices that Ecom-Tier1 has suddenly gone partially red, which indicates performance degradation on one of the nodes in this tier. She also sees a small spike in response times for the overall application. She decides to investigate.



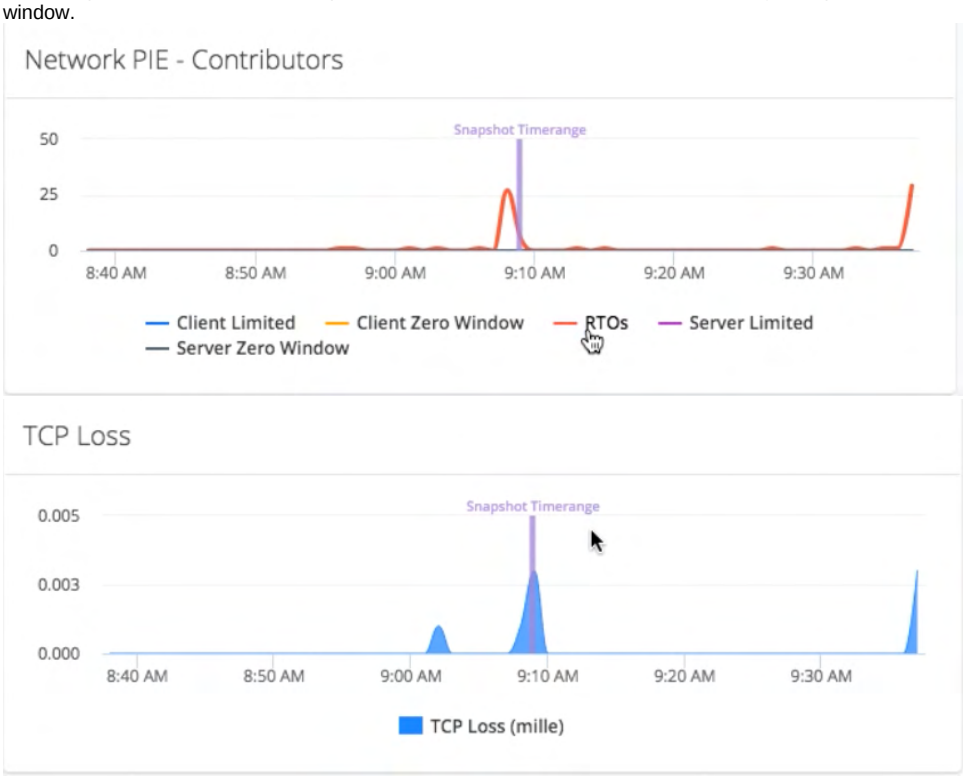
Network Diagnosis

1. To determine if there is a network problem, she:
    a. Goes to the Transaction Snapshots page and filters the list to show only stalled calls.
    b. Double-clicks a specific call to view the transaction flow map. Because it was a stalled call, the entire flow map does not display.
    c. Clicks **Drill Down** to open the Transaction Dashboard, and then switches to the Network tab to analyze the network performance during this call.

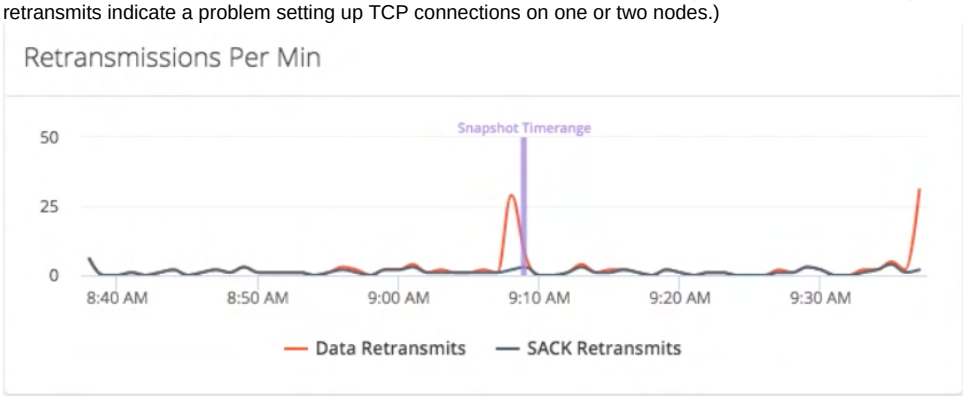
2. The Network Impact on Transactions chart (top left) shows a spike in Performance Impacting Events (PIE) around the time of the transaction snapshot. By showing only PIE and Stalls, she can see that the stalled calls and spike in PIE occur within the same time window.



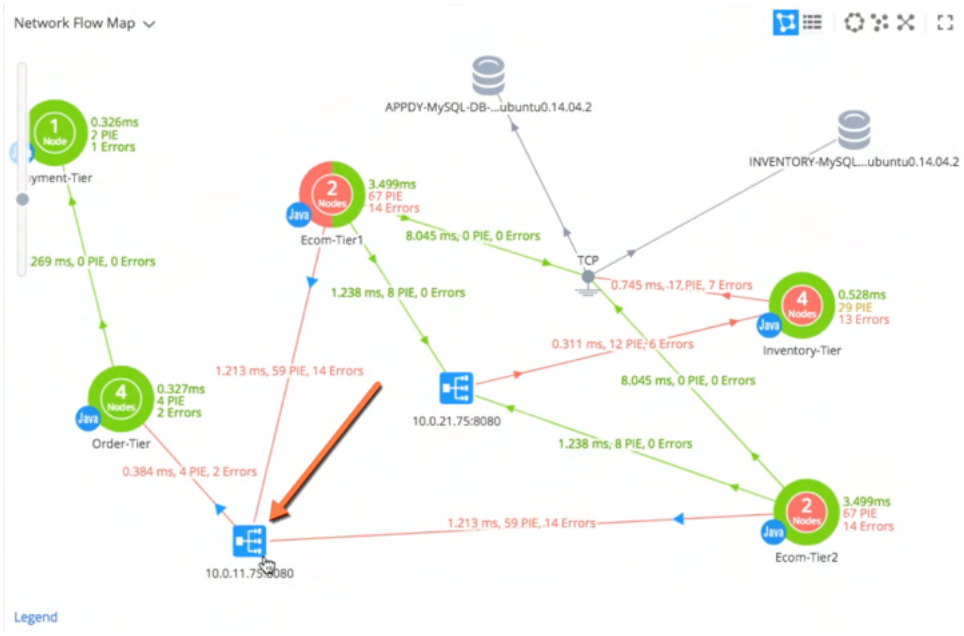
3. In the Network Pie - Contributors chart, she can see that all the Performance Impacting Events are retransmission timeouts (RTOs). This indicates packet loss on the network paths between the two tiers, which is confirmed by the spike in TCP Loss that occurs within the same time window.



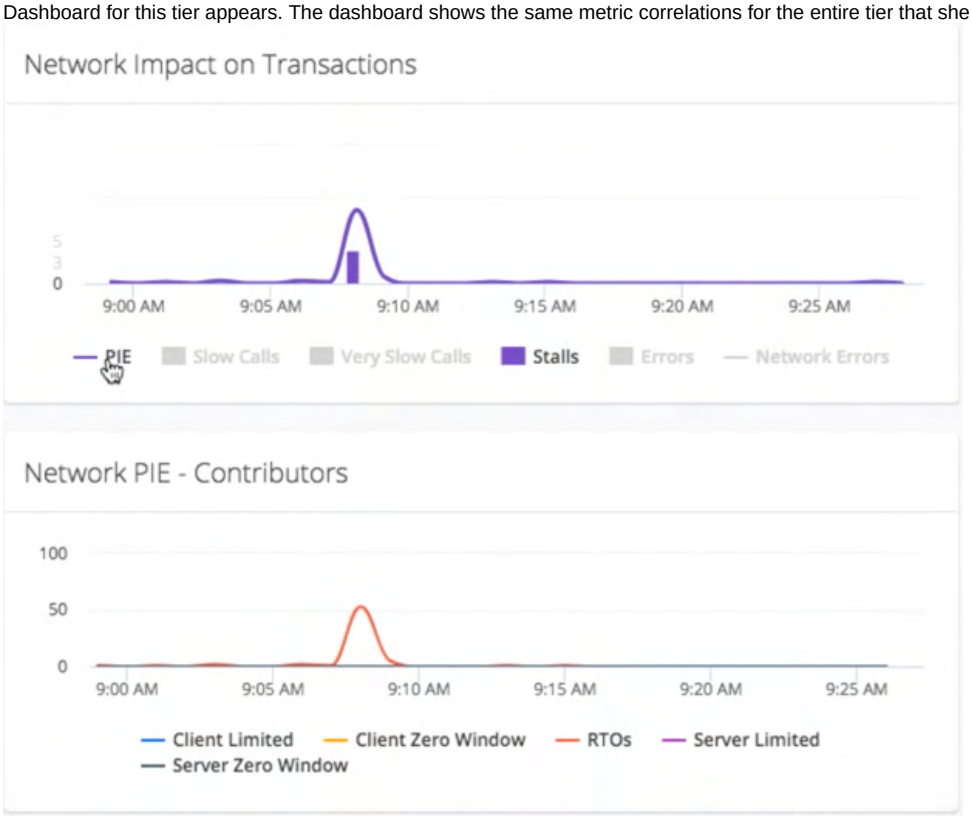
4. She knows there is a correlation between stalled calls and retransmissions, and wants to learn more about these events. The Retransmissions Per Min chart indicates that all these are all data retransmits, which indicate a problem on the network path between two nodes. (SACK retransmits indicate a problem setting up TCP connections on one or two nodes.)



5. In a few seconds, she has identified a chain of correlation: stalled calls > PIE > Retransmissions > Data retransmissions on a network path. The next step is to identify where in the network these retransmissions are occurring. She switches over to the Network Dashboard, and looks for network links with high PIE scores. She sees the PIE scores on two links are 59 (much higher than any other links). Both of these links are used by ECom-Tier1 and Ecom-Tier2 to connect with Order-Tier through a load balancer.



6. Given this information, she decides to investigate one of these tiers. She right-clicks Ecom-Tier1 and selects **View Metrics**. The Dynamic Dashboard for this tier appears. The dashboard shows the same metric correlations for the entire tier that she saw for the individual transaction.



7. Now that she has confirmed that data retransmissions are spiking for the entire tier, she wants to identify the TCP connections on which these events are occurring. She clicks the network link between Ecom-Tier and the load balancer. She can see that that PIE is occurring on both connections:
   ECOM_T1N1:8080 < – > Load_Balancer_10.0.11.77:8080
   ECOM_T1N2:8080 < – > Load_Balancer_10.0.11.77:8080
8. Given this information, she contacts the network-management team in her organization and says: "I can see that there was intermittent loss from 9:07-9:10 AM on the following TCP connections, and that these correlate with a spike in stalled calls." The network team can now investigate the network paths used by these connections and determine if the problem exists on the load balancer or elsewhere.

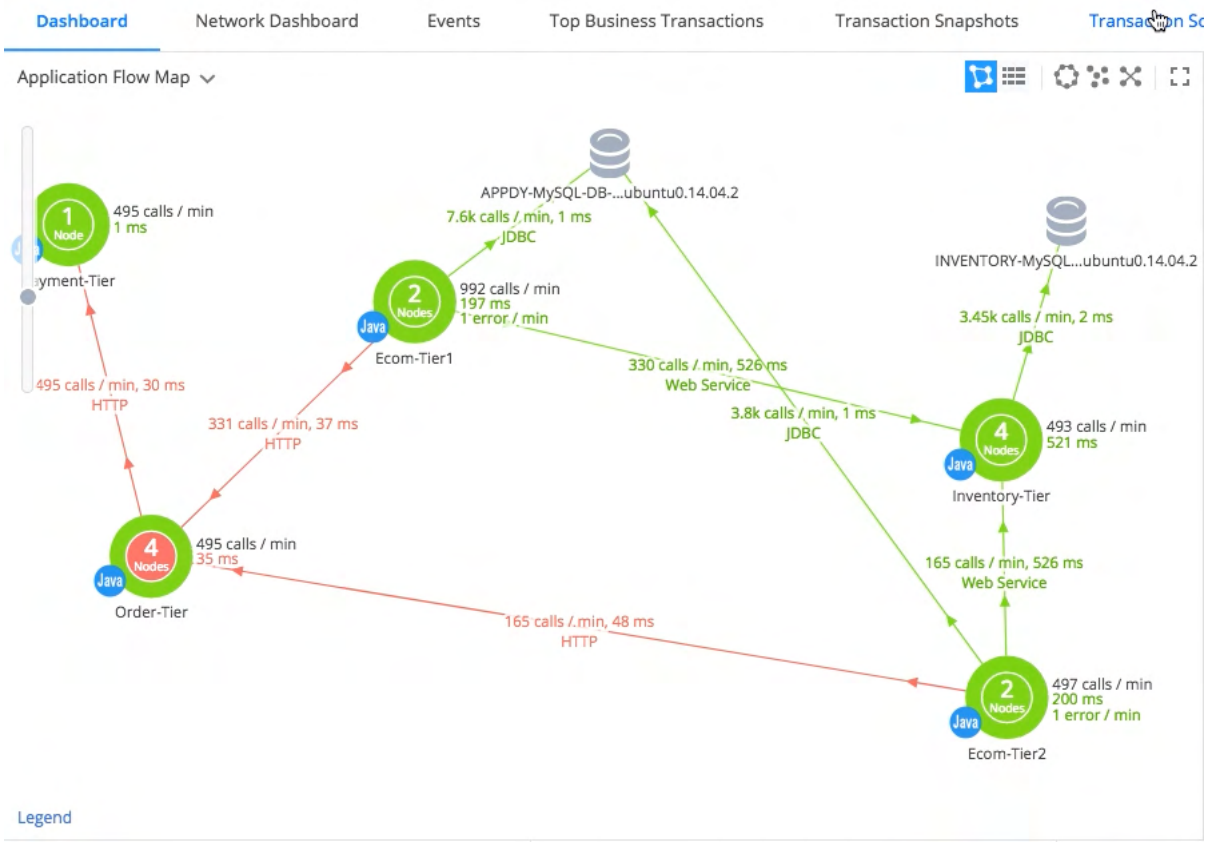
Packet Loss - Example Use Case

In some cases, packet loss can occur on a server or intermediate device. The problem may be hardware-related; perhaps the device simply cannot handle peak traffic loads, or there may be faulty cabling on one or more network interfaces.

The problem may be software-related; a bug or misconfiguration that results in intermittent packet loss at seemingly random intervals. When packet loss affects application performance, you can use Network Visibility to pinpoint the nodes, devices, and TCP connections where the packets are getting dropped.

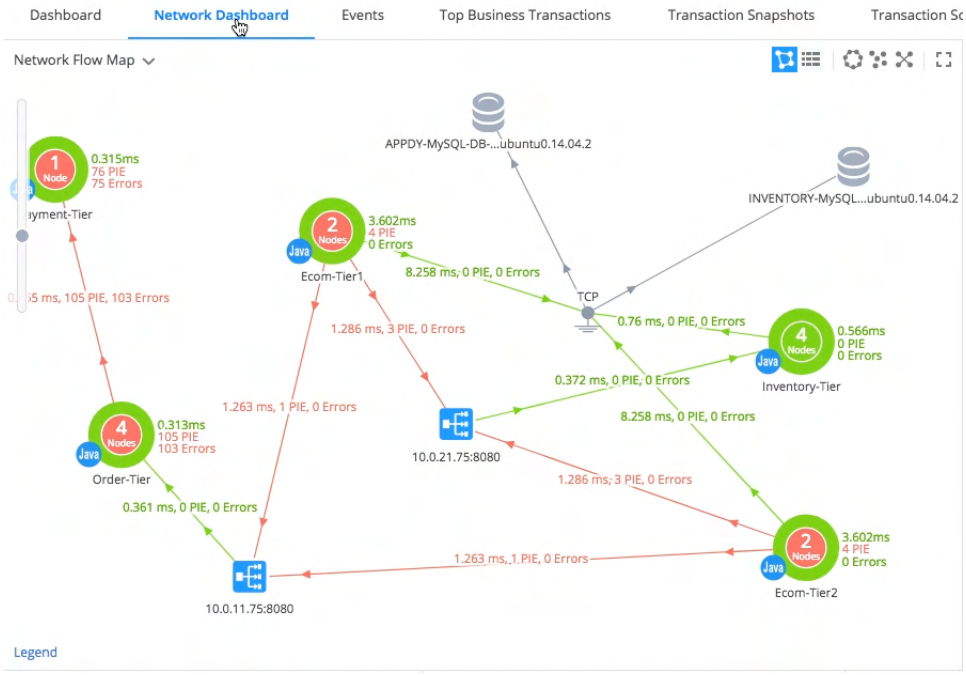
Application Symptoms

A DevOps engineer is responsible for monitoring the performance of a mission-critical app. She opens the **Application** Dashboard notices that the Order-Tier, and the application flows to and from this tier, have suddenly turned red. She decides to investigate.

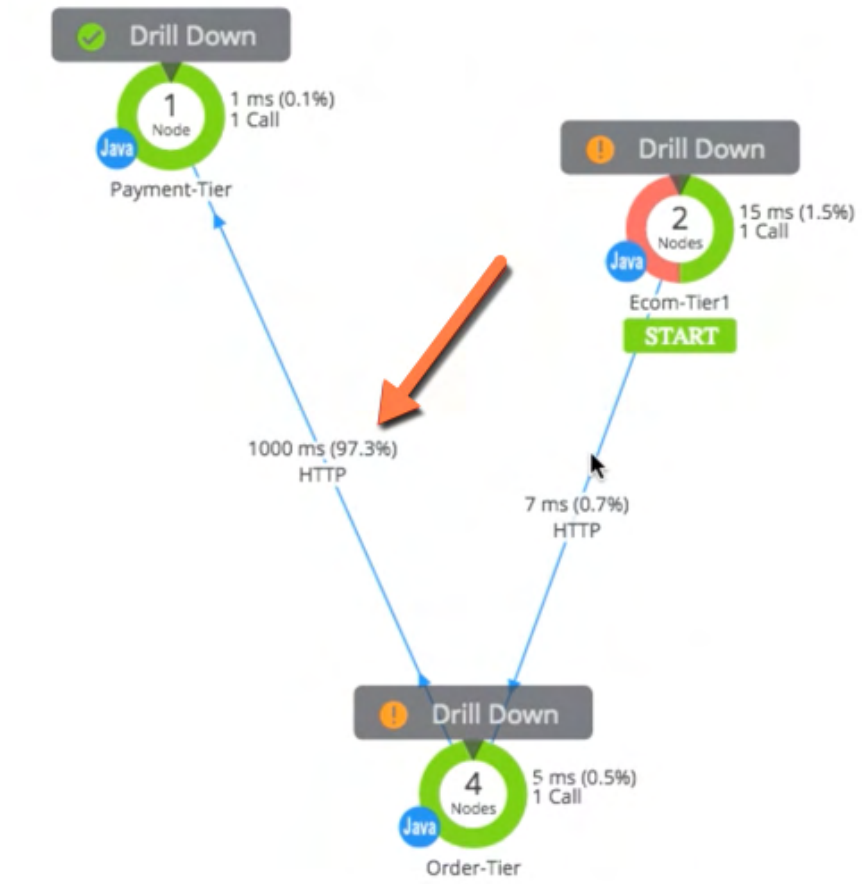


Network Diagnosis

1. She switches over to the **Network** Dashboard and sees that the Order-Tier and Payment-Tier, and the network link between them, show a spike in PIE and network errors.

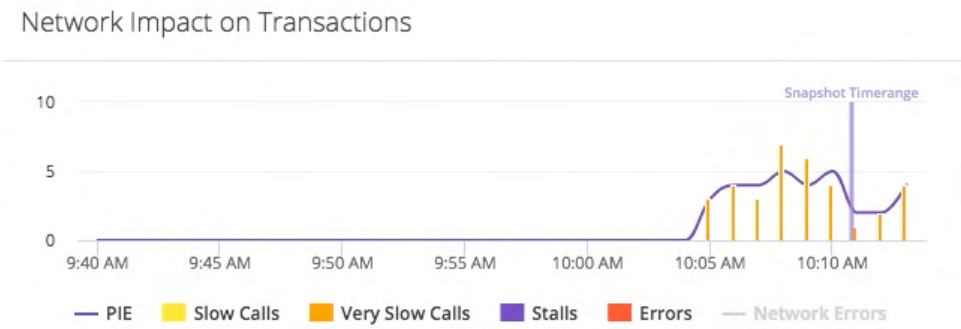


2. She decides to analyze an individual transaction. She goes to the **Transaction Snapshots** page and double-clicks a Very Slow transaction. The **T ransaction Overview** shows that 97% of the delay is on the link between Order-Tier and Payment-Tier. She drills down into the Order-Tier and goes to the Network tab.

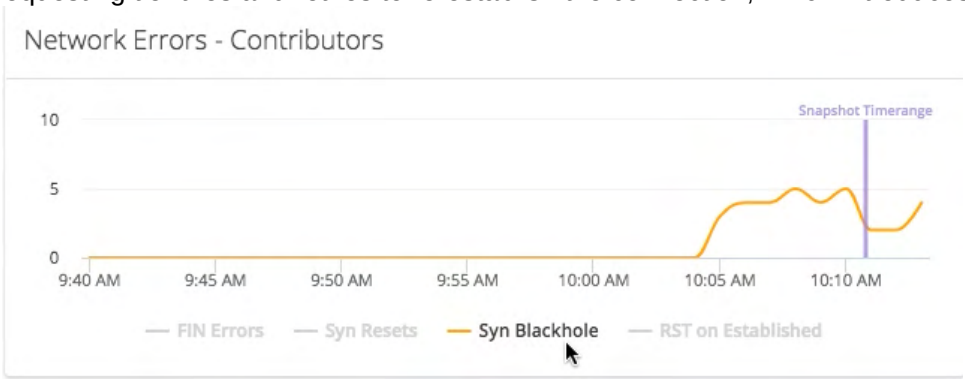


3. Scanning the dashboard for this transaction, she sees immediately that:

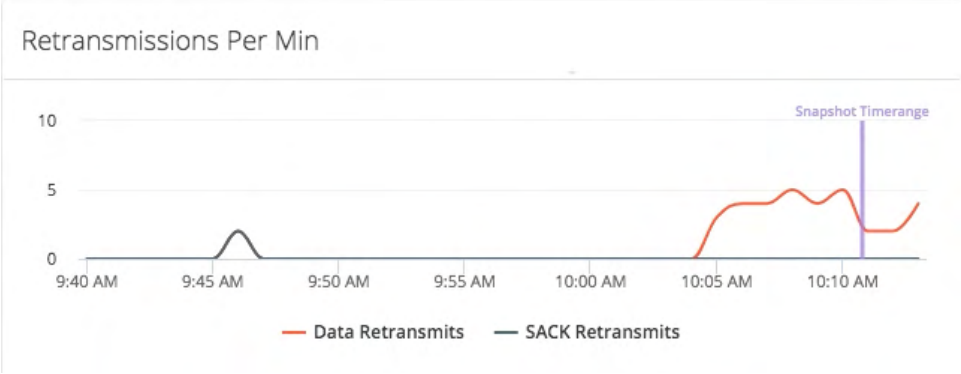
a. The Network Impact on **Transactions** graph (top left) shows a spike in PIE and Very Slow Calls around the snapshot time range.



b. She sees the correlation between Errors and Very Slow Calls, so she reviews the Network Errors - Contributors chart. She sees immediately that all these errors are Syn Blackholes. The Order-Tier node is trying to establish connections to the Payment-tier node, but something in the middle is silently dropping the connection. When this happens, it usually indicates that a firewall or other intermediate device is dropping packets. The requesting tier tries and retries to re-establish the connection, which introduces significant delays.

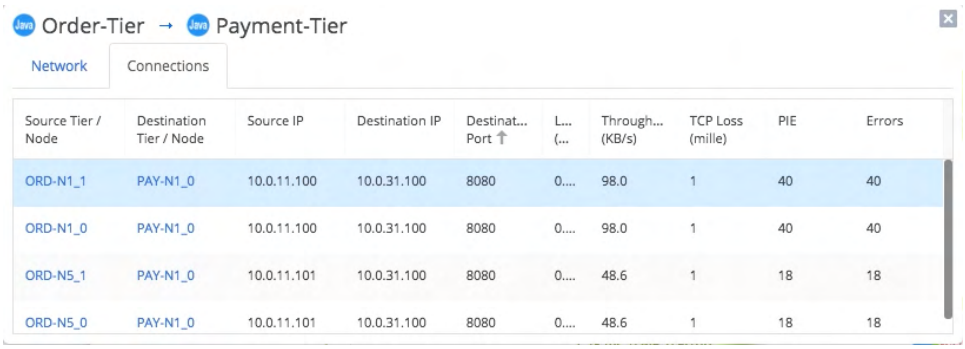


c. The Retransmissions Per Min chart confirms that the retransmissions during the time window of interest are all Data Retransmits, which means that the drops are occurring on an intermediate device rather than at an application node.



4. She now wants to pinpoint the specific TCP connections where the packet drops are occurring. She enables **Advanced Diagnostic** monitoring on the relevant Network Agents to collect TCP diagnostic metrics. She then goes to the **Network** Dashboard and clicks the links between Order-Tier and

Payment-Tier. This provides all of the connections with elevated PIE and errors.



Source Tier / Node	Destination Tier / Node	Source IP	Destination IP	Destinat... Port	L... (...	Through... (KB/s)	TCP Loss (mille)	PIE	Errors
ORD-N1_1	PAY-N1_0	10.0.11.100	10.0.31.100	8080	0....	98.0	1	40	40
ORD-N1_0	PAY-N1_0	10.0.11.100	10.0.31.100	8080	0....	98.0	1	40	40
ORD-N5_1	PAY-N1_0	10.0.11.101	10.0.31.100	8080	0....	48.6	1	18	18
ORD-N5_0	PAY-N1_0	10.0.11.101	10.0.31.100	8080	0....	48.6	1	18	18

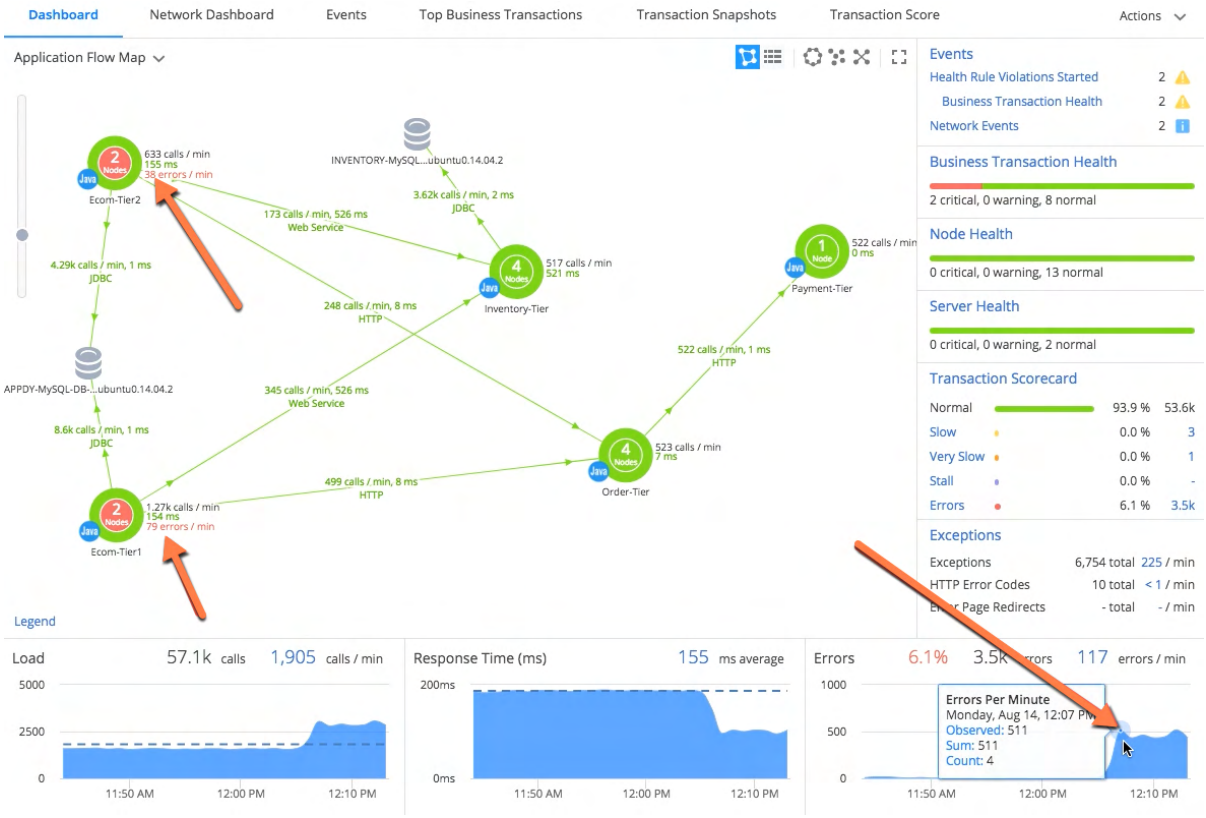
As a result, she contacts the network-management team in her organization and says: "We're seeing intermittent SYN blackholes and retransmissions on these connections, can you please investigate and fix."

TCP Port Exhaustion - Example Use Case

A TCP socket is a specific TCP port on a specific node. Port exhaustion occurs when a node runs out of available ports. When an application stops using a specific port, the port enters a "time-wait state" before it becomes available for use by another application. If a node runs out of available ports to create new connections, the symptom appears as a spike in application errors.

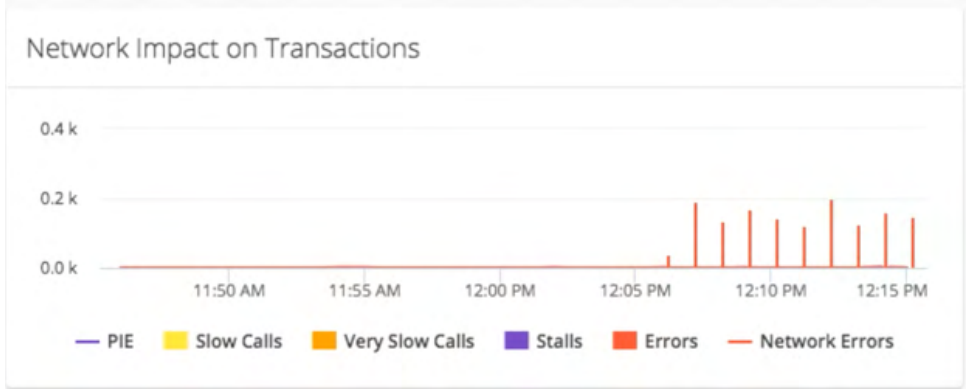
Application Symptoms

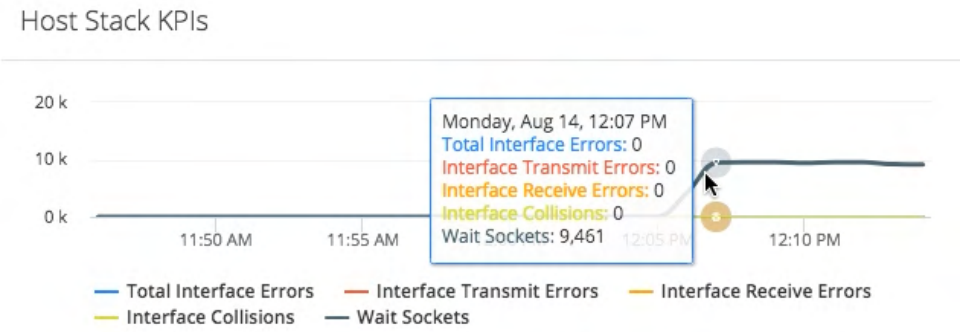
A DevOps engineer is responsible for monitoring the performance of a mission-critical app. She opens the **Application** Dashboard and notices a sudden spike in transaction errors on the Ecom-Tiers and for the entire application. She also sees that response times for the entire App are going down and load is increasing. She decides to investigate.



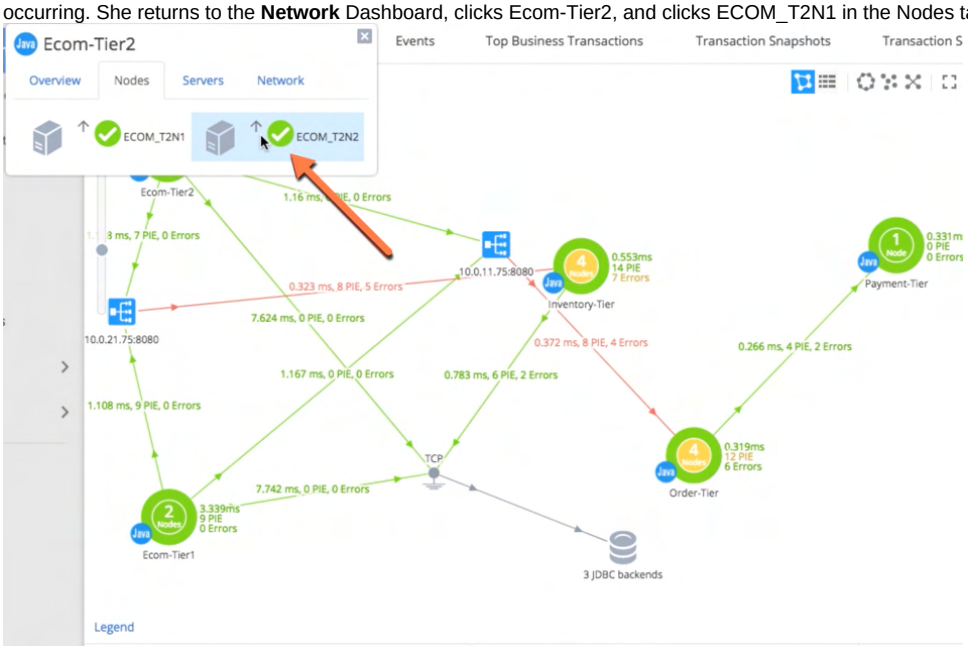
Network Diagnosis

1. She switches over to the **Network** Dashboard, right-clicks Ecom-Tier2, and selects **View Metrics**. The Host Stack KPIs show an increase in wait sockets at the same time as the increase in Error transactions.

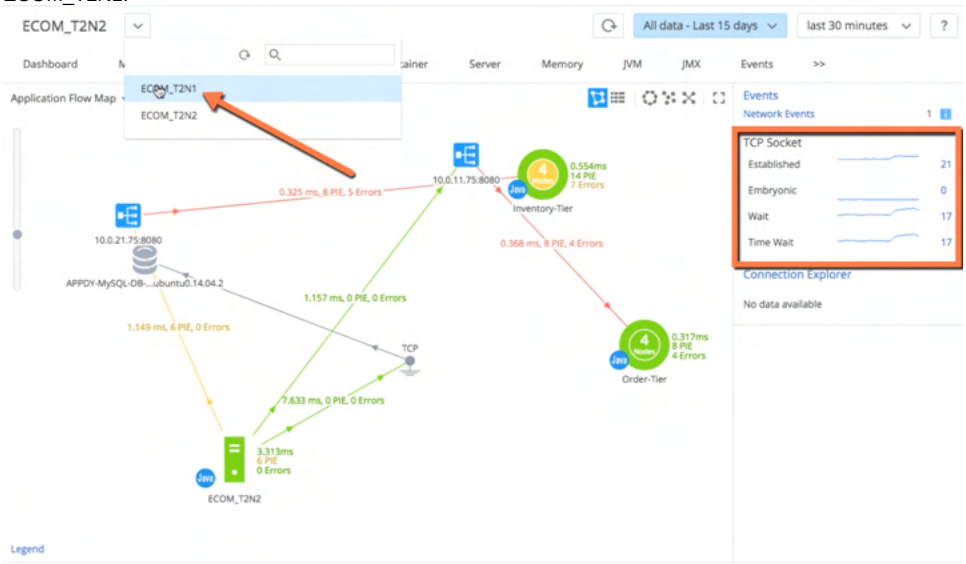




2. She knows there are many wait sockets for the entire tier. She wants to pinpoint the specific node (or nodes) where these wait sockets are occurring. She returns to the **Network** Dashboard, clicks Ecom-Tier2, and clicks ECOM_T2N1 in the Nodes tab.



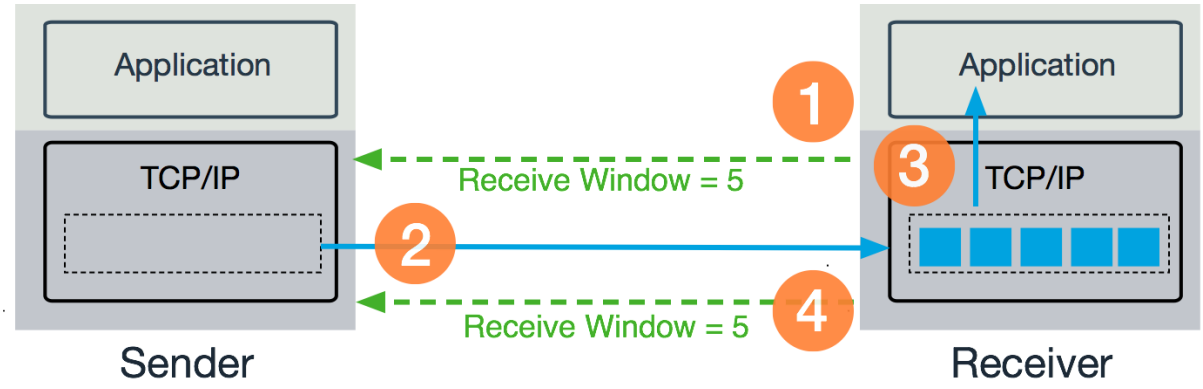
3. The TCP Socket chart (right) for node ECOM_T2N2 shows that the level of wait sockets is in an acceptable range. She switches over to view ECOM_T2N1.



4. The TCP Socket chart for this node shows a significantly high number of wait sockets. She knows now that the spike in application errors is occurring because this node is running out of available sockets.

TCP Window Bottlenecks - Example Use Case

Back pressure is common in microservices and service-oriented applications, especially in distributed environments. Back pressure occurs when two services communicate and one service gets overwhelmed. This diagram shows the sequence of events:

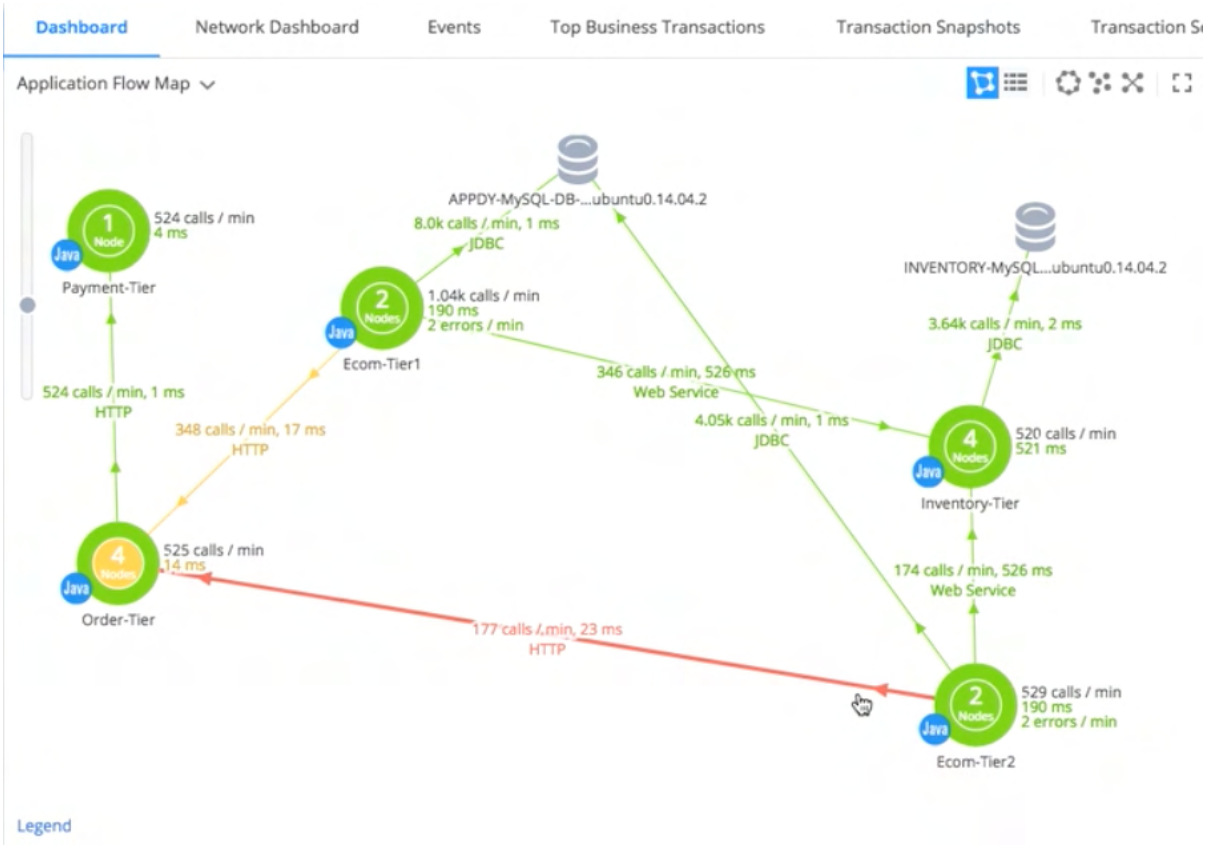


1. Receiver has a TCP Receive buffer for incoming packets. It advertises the available buffer space to the sender.
2. Sender sends a chunk of data based on the advertised window size.
3. Application on the receiver processes the packets in the buffer and frees up space in the buffer.
4. Receiver advertises the new window size.

This process works smoothly until the receiving node cannot process incoming packets quickly enough. In this case, the receiver sends TCP Limited or TCP Zero messages to the sender and the transfer slows down.

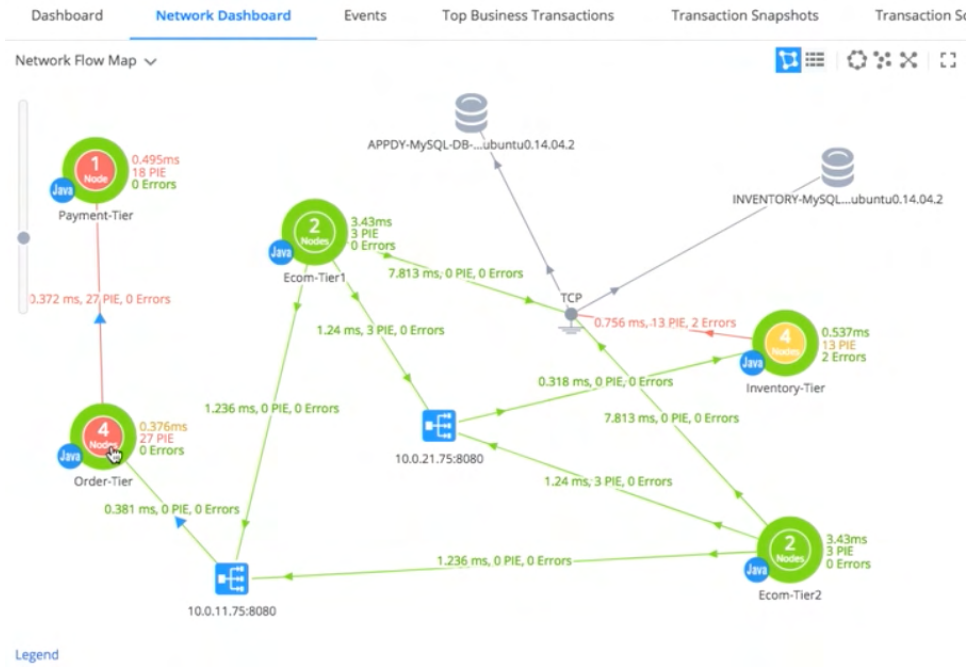
Application Symptoms

A DevOps engineer is responsible for monitoring a mission-critical app. She scans the **Application** Dashboard and notices that a link and tier have turned yellow and another link has turned red. She decides to investigate.

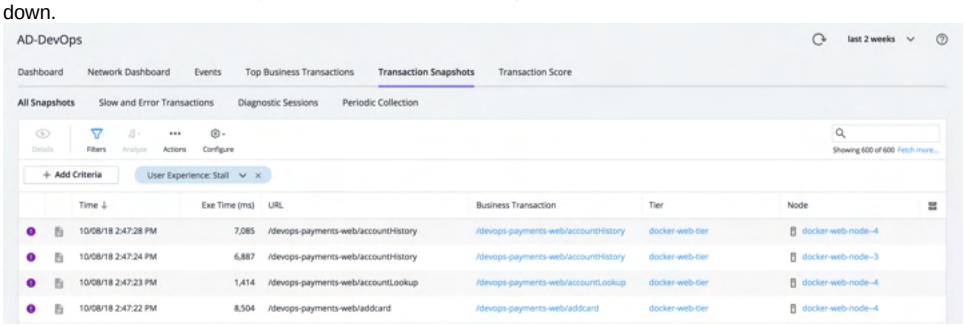


Network Diagnosis

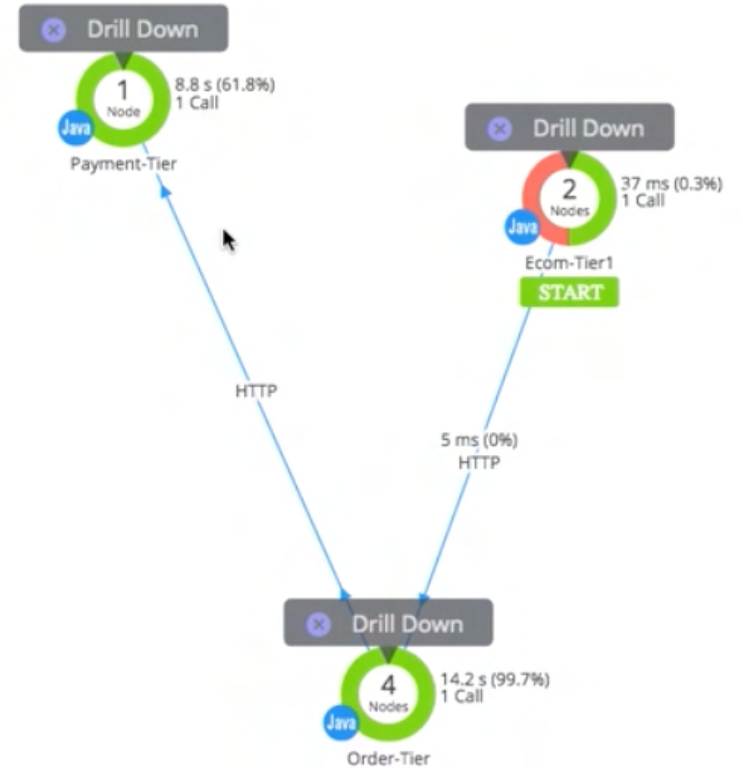
1. She switches over to the **Network** Dashboard and sees a significant increase in Performance Impacting Events (PIE) between the Order-Tier and the Payment-Tier. While the **Application** Dashboard shows performance issues on the upstream Order-Tiers, the **Network** Dashboard implies that the problem is further downstream—between the Order-Tier and Payment-Tier.



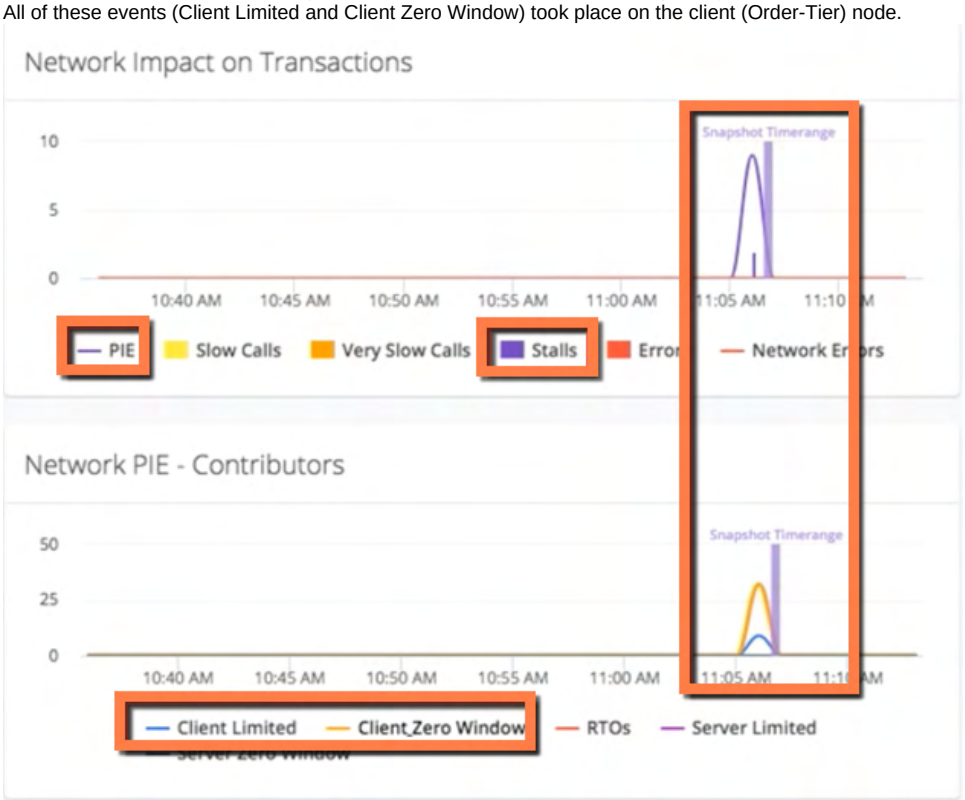
2. To troubleshoot further, she goes to the **Transaction Snapshots** list, filters on stalled transactions, and double-clicks on a transaction to drill down.



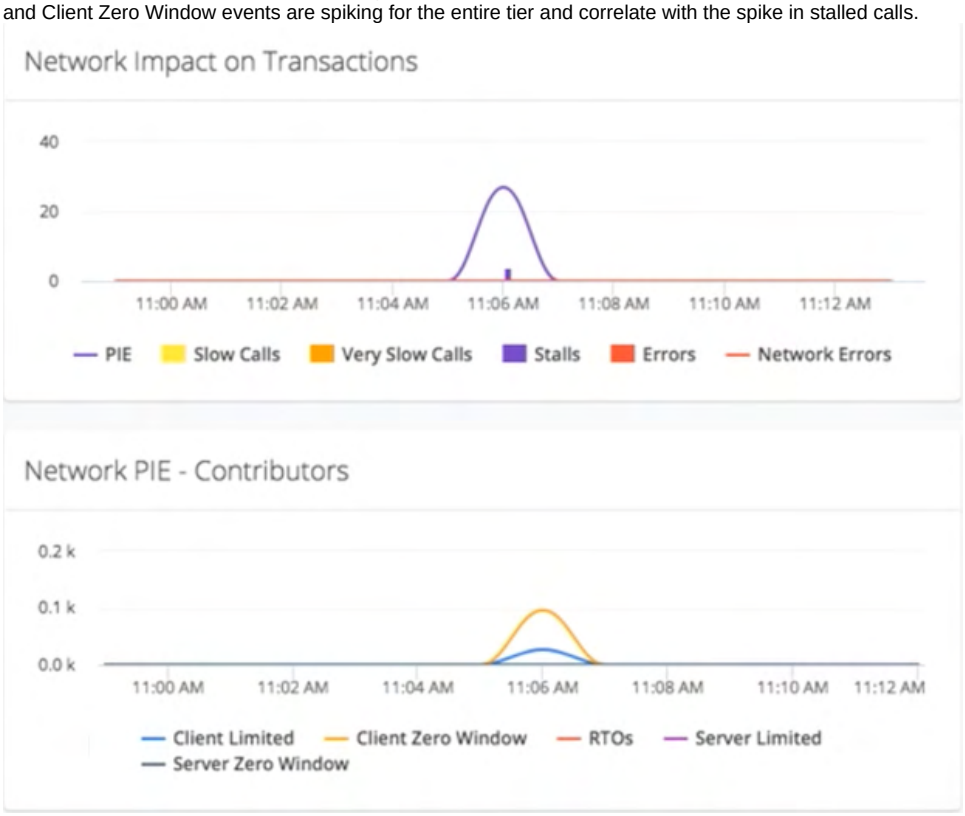
3. She drills down into the Order-Tier, since nearly all of the transaction time (99.7%) occurred at this tier.



4. In the **Transaction Drilldown** page, she switches over to the **Network** view. Scanning the dashboard, she can see immediately that:
   a. The transaction correlates with a spike in stalled calls and Performance Impacting Events.
   b. All of these events (Client Limited and Client Zero Window) took place on the client (Order-Tier) node.



5. She returns to the **Network** Dashboard, right-clicks on the Order-Tier, and chooses **View Metrics**. She immediately sees that the Client Limited and Client Zero Window events are spiking for the entire tier and correlate with the spike in stalled calls.



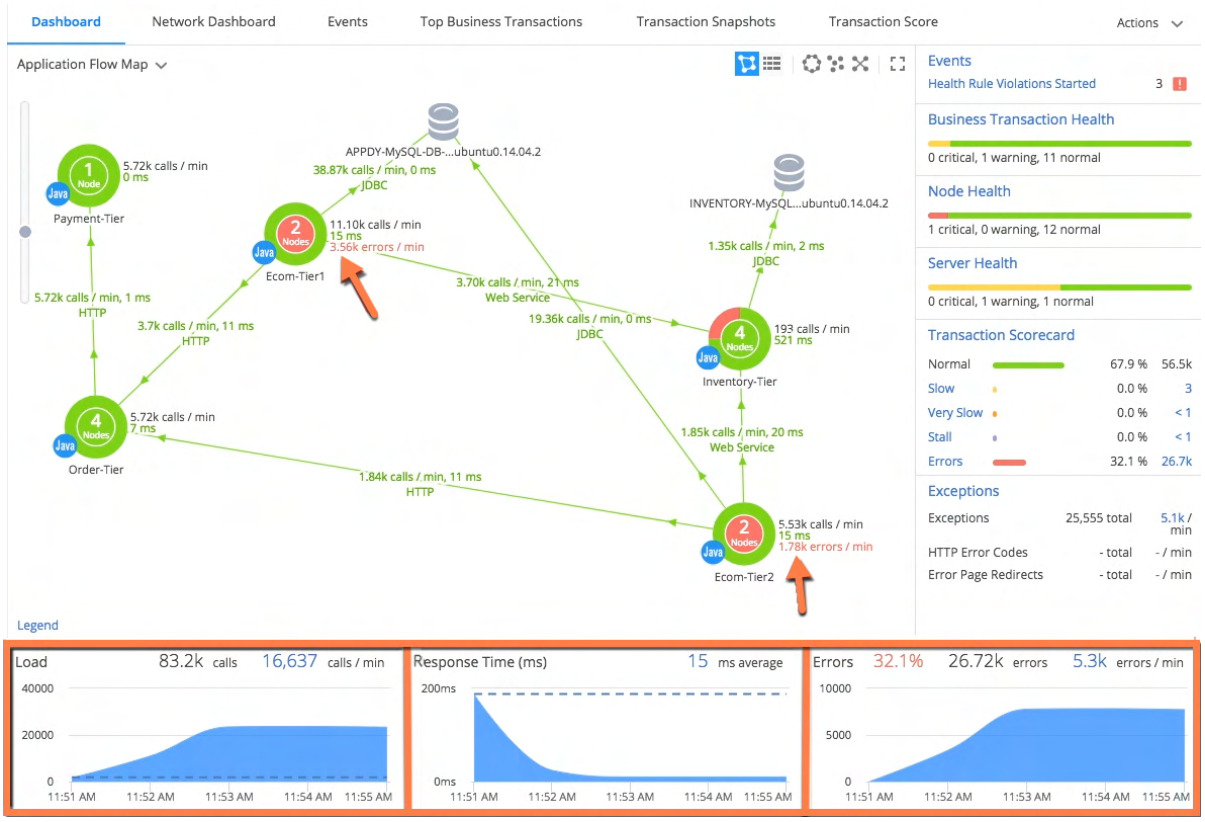
Traffic Throttling - Example Use Case

Firewall throttling occurs when a firewall or other intermediate device prioritizes some connections over others, or denies some connections. This may be due to traffic policies explicitly defined on the device, or to one or more misconfigurations.

Application Symptoms

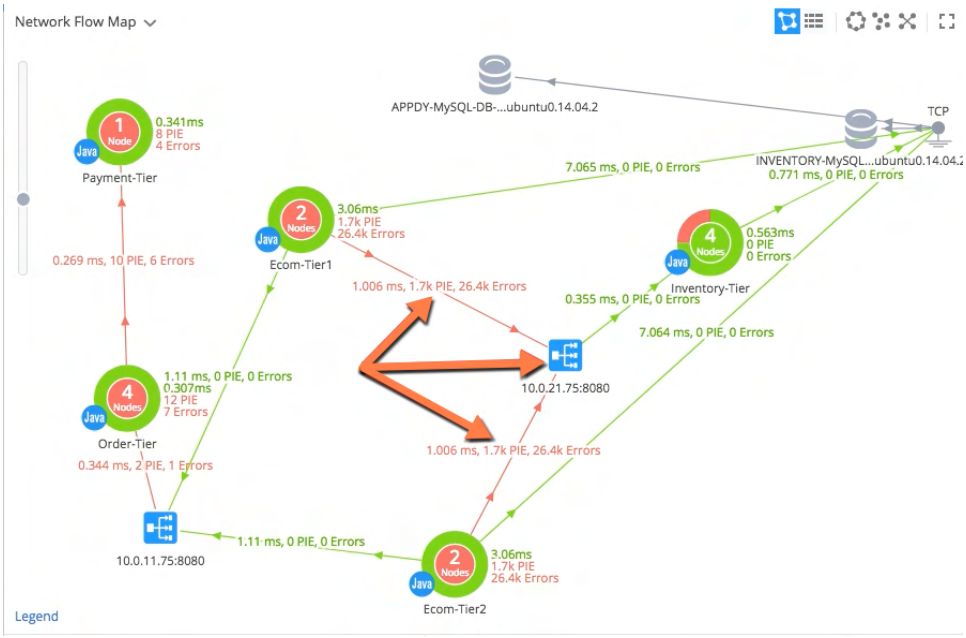
A DevOps engineer is responsible for monitoring the performance of a mission-critical app. She scans the **Application** Dashboard and notices that:

- Ecom-Tier1 and Ecom-Tier2 are showing many errors
- Traffic Loads and Errors are increasing, while response times are decreasing (bottom charts)

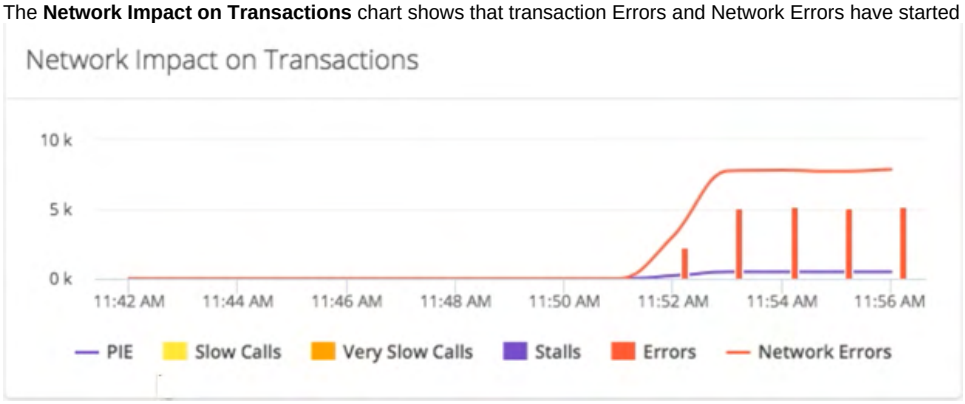


Network Diagnosis

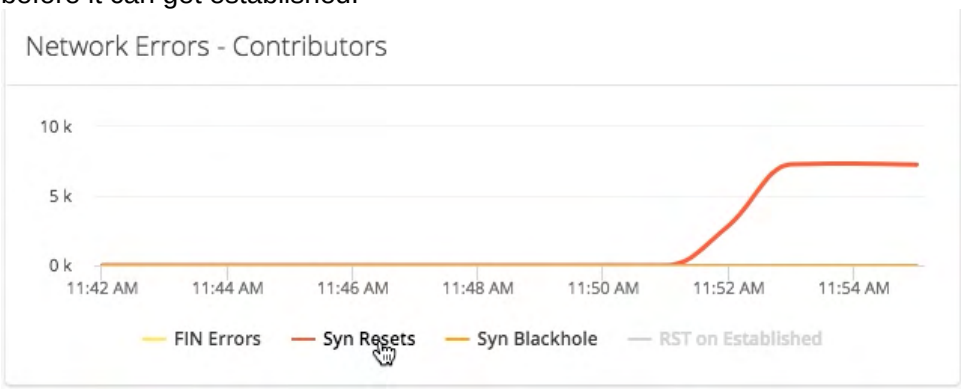
1. She switches over to the **Network** Dashboard and sees immediately that many network errors are occurring on the links between the Ecom-Tiers and the load balancer in the center.



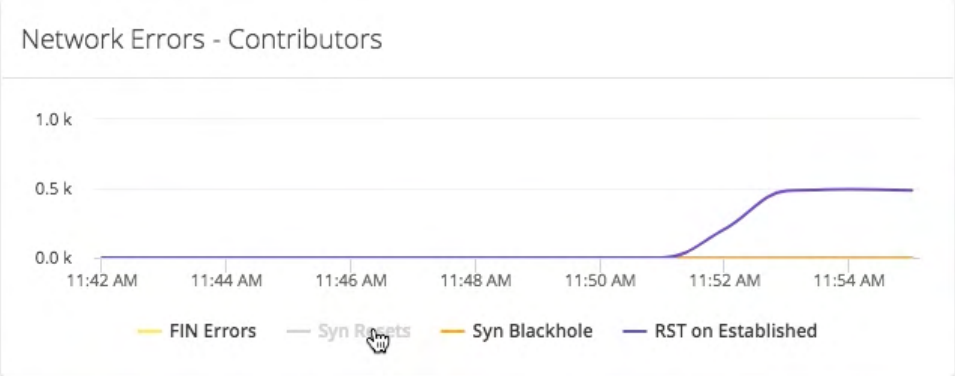
2. She right-clicks Ecom-Tier1 and selects **View Metrics**. She notices:
   a. The **Network Impact on Transactions** chart shows that transaction Errors and Network Errors have started increasing at the same time.



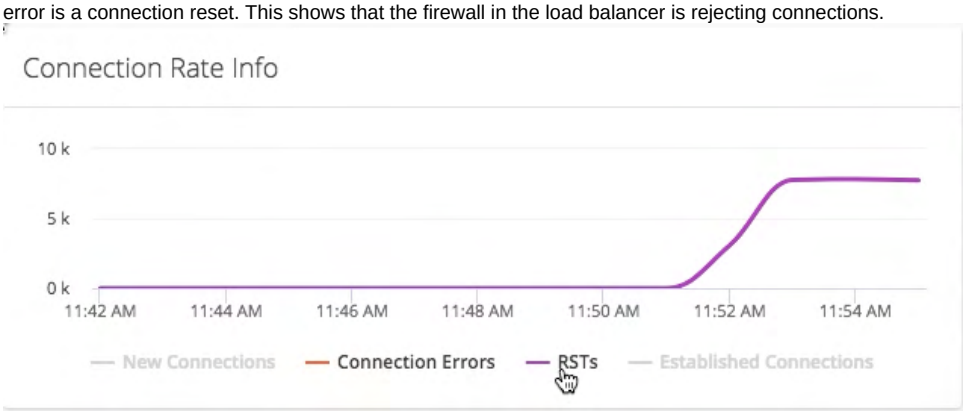
   b. The **Network Errors - Contributors** chart shows that two types of Network Errors are increasing:
      i. Syn Resets — This reset occurs when the firewall explicitly rejects a connection request before it can get established.



ii. RST on Established — This reset occurs when the firewall shuts down an established connection due to traffic-throttling or other policy on the device.



c. The **Connection Rate Info** chart shows that the rate of Connection Errors and Resets are exactly the same where every connection error is a connection reset. This shows that the firewall in the load balancer is rejecting connections.



User Interface

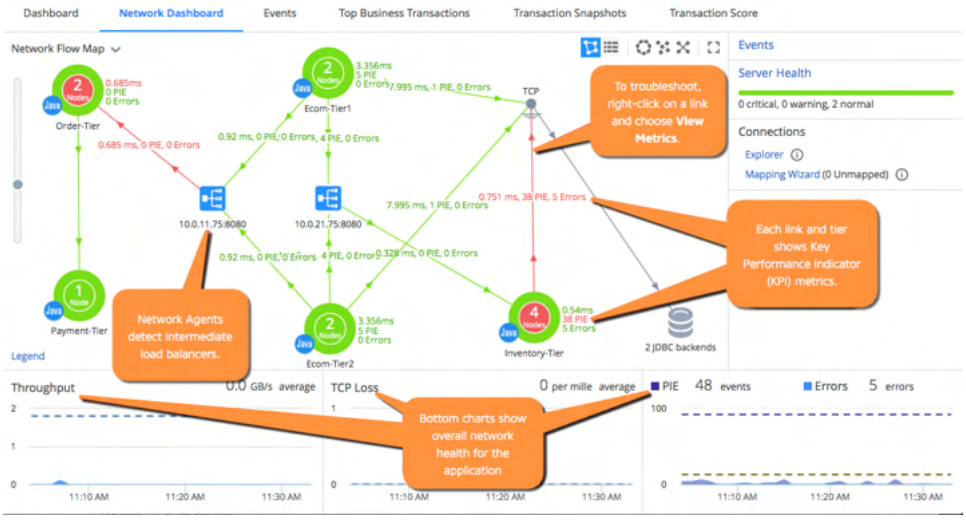
This page describes:

Network Dashboard

The **Network** Dashboard provides a network-layer view of your application where you can quickly determine if any part of the network is impacting application performance. The bottom charts show the overall network health. Each tier and link shows network KPI (Key Performance Indicator) metrics. Network Agents detect intermediate load balancers and TCP endpoints. You can:

- Enable baselining to visualize network elements with performance issues (red/yellow links and tiers)
- View network metric charts in tier/link popups
- Find root causes in a specific network link. Right-click and select **View Metrics** to open a context-sensitive dashboard for the element.
    - The tier dashboard shows correlations between outlier (Slow/Very Slow/Stalled/Error) transactions and network-performance metrics
    - The link dashboard shows network-performance issues on the client tier, server tier, and network path
- Find root causes in a specific connection. When you narrow the issue down to a specific network link, you can:
    - Configure the Network Agents to collect detailed metrics for the individual Connections used on that link.
    - Drill down to find the root cause in an individual Connection.
- Copy IP addresses and ports for individual connections from the **Network** Dashboard enabling you to forward this information to operations and other personnel when troubleshooting an issue. To copy IP addresses and ports:

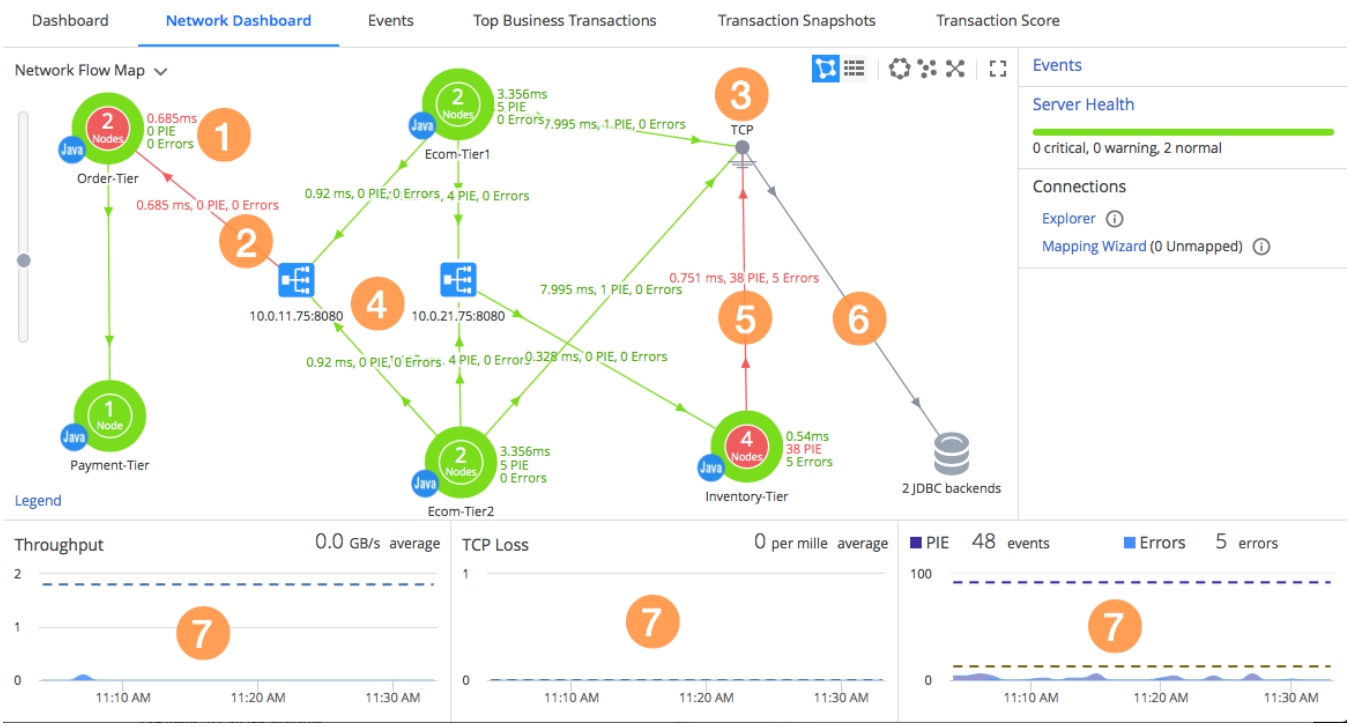
    1. Click a link to open the link popup, and go to the Connections tab.
    2. Select the connections of interest (use Ctrl-click to select multiple connections).
    3. Right-click the selection and select **Copy IP addresses to clipboard**.



Important Notes

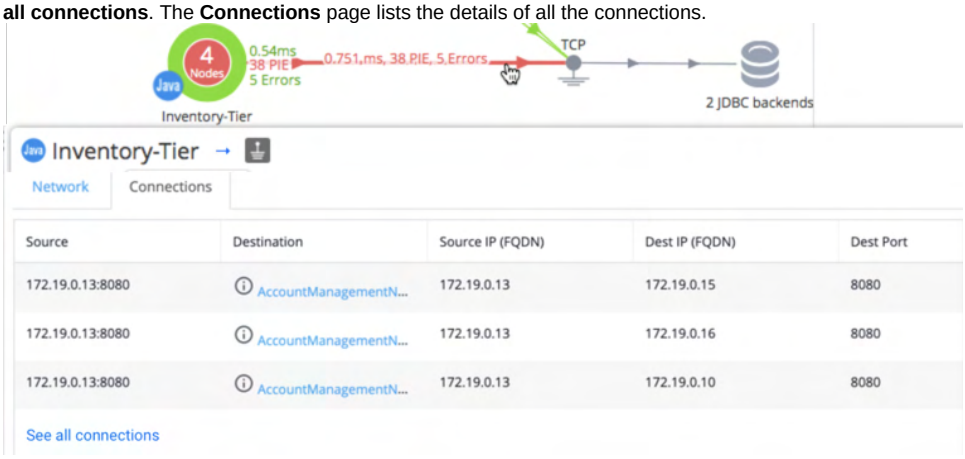
- In many cases, the **Application** Dashboard and **Network** Dashboard might show different topologies for the same application. This is the expected behavior. Unlike App Server Agents, Network Agents detect intermediate network devices between nodes, and consider these devices when collecting metrics. The two dashboards show the same application but from different perspectives: an application perspective and a multi-layer, network/application perspective.
- This release does not support these Network Flow Map features:
    - Federated Flow Maps
    - Visualization of flows between web servers and APM entry tiers
    - The Network Flow Map does not filter out connections based on the selected time range
- In some cases, the KPIs for a tier or link might be different in the Network tab versus the Flow Map. The popup shows the latest data; the Flow Map updates data every two minutes. Any discrepancy between KPI values is due to this difference in reporting times.
- If you open the Network Flow Map for an individual node, the KPI metrics for node-to-load-balancer and node-to-tcp-endpoint links show KPIs for all nodes in the parent tier (instead of KPIs for the individual node only). To view KPIs for the individual node, open the link popup and review the Connection KPIs.

Network Dashboard Reference



The Network Dashboard shows:

- Network-based key performance Indicators (KPIs) (1) for the monitored nodes in each tier:
  - Average Latency (#ms) for individual packets
  - Number of Performance Impacting Events (PIE)
  - Number of Errors
- KPIs for all network links (2). Click a load balancer or a network element to review the set of five associated Connections. (A gray link indicates a connection that a Network Agent detected but did not directly monitor.) If a link or load balancer is associated with more connections, click **See all connections**. The **Connections** page lists the details of all the connections.



- The **Network** Dashboard shows a TCP endpoint (3) icon when a Network Agent observes traffic between a monitored node and an endpoint (IP address, TCP port, and protocol), but cannot determine if the endpoint represents:
  - An application node that sends or receives traffic for the application, or
  - An intermediate device (proxy or load balancer) that transfers traffic to and from another node over a separate connection
- Network Agents automatically detect load balancers (4), which often mask internal IP addresses and have the effect of splitting an in-flight application message into two separate TCP connections. The auto-detection of TCP endpoints and load balancers between application nodes enable you to easily identify the exact locations where network issues are occurring.
- Links are colored in the **Network** Dashboard as:
  - Green/yellow/red, to visualize the network performance compared to the overall performance baseline for that link (5)
  - Blue, if baselining is disabled
  - Gray, for connections between application nodes and TCP endpoints. (6) In this example, the gray link indicates "The connected node and endpoint are either the same device, or two devices exchanging traffic over a separate, unmonitored connection."

- The **Network** Dashboard also shows KPIs 7 for the entire application. You can review:
  - Overall throughput that the application places on the network
  - Rate of network packet loss (packets sent but never received)
  - Number of Performance Impacting Events (PIE), which indicate potential problems on one or more nodes or connections
  - Number of errors for the entire application

Connection Explorer

Related pages:

- KPI Metrics in Right-Click Dashboards
- Network Visibility Metrics

The Connection Explorer (**Explorer** link, right side of **Network** Dashboard) shows all application flows and TCP Connections for an application in one table. Use the Connection Explorer for large, complex applications. You can sort the table based on network KPIs to locate flows with performance issues; then you can drill down and analyze the supporting connections in right-click Connection Dashboards. See Flows, Links, and Connections.

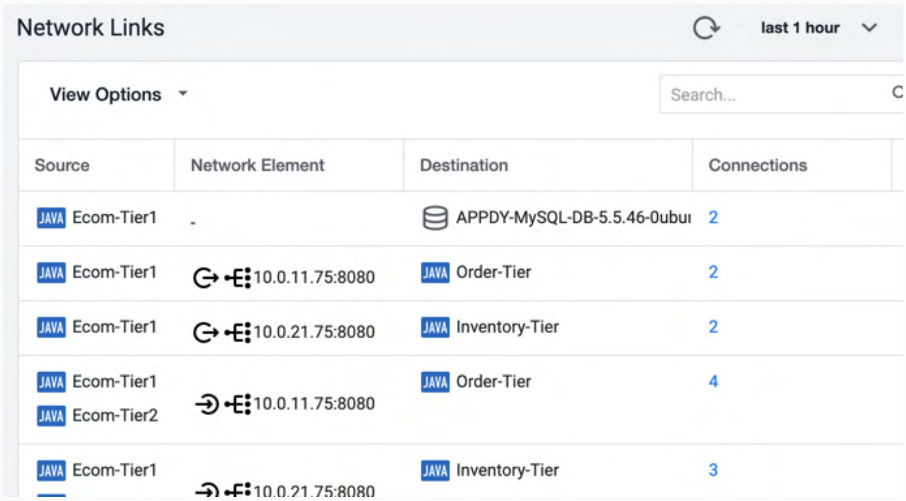
The Connection Explorer shows information for:

- Network Links
- Connections

Network Links

The Explorer shows this information for each network link. Select **View Options** to show or hide information as appropriate.

- Source
- Destination
- Network Element

> **Tip**
>
> If two flows connect through a Load Balancer (LB), the table shows the LB IP:port and the connection source:
>
> Flow direction: LB --> Tier 
>
>
> Flow direction: Tier --> LB 
>
>
> To see the relationships between application tiers and intermediate Load Balancers (LBs) more clearly, you can drag the Network Element heading so that the columns are ordered **Source** | **Network Element** | **Destination**.
>
> 
>

- Connections
- Latency
- Errors
- PIE
- TCP Loss (Mille)

Connections

To view the load balancer details for a connection, click the connection. Select **View Options** to show or hide information as appropriate.

- Source IP, or Fully Qualified Domain Name (when available) for Load Balancers and TCP Endpoints
- Destination IP, or Fully Qualified Domain Name (when available) for Load Balancers and TCP Endpoints

Custom Dashboards for Network Visibility

You can create Custom Dashboards based on Network Visibility metrics using the standard workflows. To add a Network Visibility metric to a custom dashboard:

1. In the Add Widget pane, click **+** (under **Data**).
2. Under **Data Source**, select the application of interest.
3. Under **Select a Metric Category**, select **Custom (use any metrics)**.
4. Under Select a Metric, navigate to the metric of interest under one of these:
    - **Application Infrastructure Performance > Advanced Network**
    - **Application Infrastructure Performance >** <*tier-name*> **> Advanced Network**

See Network Visibility Metrics.

Network Visibility Events

This page lists the Network Visibility events and severity.

Packet Capture Events

Sub-type: PCAP (set by Network Agent)

Description	Severity
Packet capture successful, Packet capture ID: <*id*>, Captured file: <*file*.pcap>	Info
Packet capture already active, Packet capture ID: <*id*>	Info
Packet capture couldn't be started, Storage location </loc/path> does not exist, Packet capture ID: <*id*>	Err
Not enough space to start packet capture, Packet capture ID: <*id*>	Err
Packet capture failed. Capture directory </*loc*/*path*> does not exist, Packet capture ID: <*id*>	Err
Packet capture successful and exported, Packet capture ID: <*id*>, Captured file: <*file*.pcap>	Info
Packet capture successful but export failed, Packet capture ID: <*id*>, Captured file: <*file*.pcap>	Err
Packet capture not started. Agent Disabled, Packet capture ID: <*id*>	Err

Dynamic Monitoring Mode Events

Sub-type: Config (set by Network Agent)

Description	Severity
Network Visibility Agent mode change successful, Last DMM: <Kpi/Diagnostic/Advanced>, Current DMM: <Kpi/Diagnostic/Advanced>	Info
Network Visibility Agent mode change failed, Current DMM: <Kpi/Diagnostic/Advanced>	Err

State-Change Events

Sub-type: Config (set by Network Agent)

Description	Severity
Network Visibility Agent state change successful, Current state: <Enabled/Disabled>	Info
No change in Network Visibility Agent state, Current state: <Enabled/Disabled>	Info
Network Visibility Agent state change failed as Packet capture active, Current state: <Enabled/Disabled>	Err
Network Visibility Agent state change failed, Current state: <Enabled/Disabled>	Err
Network Visibility Agent started successfully, Current state: <Enabled/Disabled>	Info

Start Events

Description	Severity
Network Visibility Agent registered successfully	Info
Network Visibility Agent registration failed due to an invalid ID	Err
Network Visibility Agent started successfully, PID: <pid>	Info

Advanced Operations

Network Visibility includes these advanced operations:

- Resolve Unmapped Connections – In some cases, there is not enough context to map a TCP Connection to an application flow automatically. This page describes how to resolve these connections manually.
- Packet Captures – A packet capture is a snapshot of live network traffic. Packet captures are very useful for advanced, in-depth network diagnostics and troubleshooting.

Packet Captures

A packet capture is a snapshot of live network traffic. Use packet captures for in-depth network diagnostics and troubleshooting. When you discover a network issue that affects your applications, you can capture traffic using Network Visibility Agents and send the resulting data to your network or Ops team for further analysis.

Network Agents save packet captures as pcap files. A wide variety of network analysis tools support pcap: Wireshark, tcpdump, Windump, and so on. Packet captures are supported on Linux platforms only.

> **Restrict packet capture privileges to authorized users only**
>
> Packet capture files include "raw" application data that might contain sensitive information. Any user with Account Owner or Administrator privileges can perform packet captures. For this reason, these roles should be assigned to authorized users only. See Roles and Permissions.

Before You Begin

> You must perform this setup *on each host* before you can capture packets on that host.

1. In the Controller, click the gear icon in the top right () and select **AppDynamics Agents** > **Network Visibility Agents**.
2. Right-click the Agent to set up and select **View Packet Capture Configuration**.
3. Set the capture settings:
   - **Duration (sec)** – Make it long enough to capture at least one Business Transaction over the link that you want to troubleshoot
   - **Size** – The maximum size for any single capture file
     Packet Capture Filename Prefix – You must specify a prefix. It is good practice to include the hostname or another string that clearly identifies the node. The resulting pcap filename includes the prefix, the IP address, the interfaces captured, and the timestamp.
     For example, if you specify a prefix of `DataCenterNYC--`, the resulting pcap will have filename: `DataCenterNYC--ip-10-0-21-101_any_1_2017_09_28_17_58_03.pcap`
4. Set the storage settings:
   - **Path** – If remote storage is disabled, the Agent stores capture files in this folder on the Agent host
     - The specified folder must exist on the Agent host. The default path is `/opt/appdynamics/netviz/pcap`.
     - The *<network-agent-user>* account (the one used to install and run the Network Agent) should have read and write permissions to this folder
   - **Maximum Allotted Space** – Maximum storage allotted for all capture files. This setting applies to both the Agent host and the remote server. As new capture files are created, the Agent deletes older files to free up space.
5. Set the storage settings (SCP server):
   - **Remote Storage (upload to SCP Server)** – With this option enabled, the Agent uploads the capture file to the specified server when the packet-capture operation ends
   - **Host/Port/Username/Password/Path**
     - The local path must be defined on the remote server
     - The specified user account must have write permissions on the specified path

Best Practices for Packet Captures

Packet Capture files can get very large, very quickly. When a capture job is in progress, the Network Agent captures all bytes in all packets on all network interfaces that it monitors. The size of the capture file depends on the capture duration, and the rate of packets sent and received on the network interfaces of the node. The duration should be long enough to capture a few Business Transaction calls between the two nodes, but no longer.

If you want to retain any capture file for archiving or extended analysis, copy the file from the storage folder as soon as the capture completes. This ensures that it does not get overwritten by newer files.

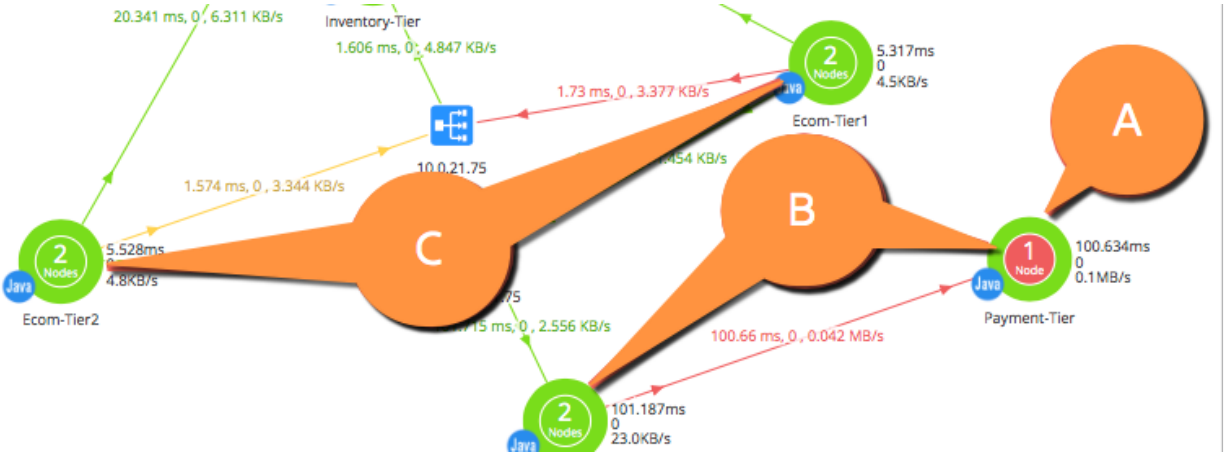
Packet capture operations generate a number of Network Visibility Events that you can use for monitoring and troubleshooting.

Create a Packet Capture

- Determine the Nodes to Capture
- Start the Packet Capture

Determine the Nodes to Capture

- Go to the Network Dashboard, set the reporting duration to the last five minutes, and verify that the network issue you need to troubleshoot is currently active.
- Note the node(s) where you need to capture packets.
  - To troubleshoot a node, capture on the node (**A**).
  - To troubleshoot a link, capture on the two connecting nodes on each side of the link (**B**).
  - If the link is bisected by a load balancer, capture on both sides of the load balancer (**C**).



Start the Packet Capture

When you start a capture, the Agent captures all packets sent and received by all network interfaces on the Agent host. When the Agent stops capturing (as specified by the **Duration (sec)** capture setting), it saves the pcap file in the folder specified by **Storage** settings).

There are two capture methods:

- From the Agents Page
- From the Node Dashboard

From the Agents Page

Use this method to capture on one or more nodes:

1. In the Controller, click the gear icon in the top right ( ) and select **AppDynamics Agents** > **Network Visibility Agents**.
2. Select the Agents on the nodes where you want to capture. Use Ctrl-click or Shift-click to select multiple Agents.
3. Right-click a selected Agent and select **Start Packet Capture**.

From the Node Dashboard

Use this method to capture on a single node:

1. Drill down to the node in the Network Browser:
    a. Go to **Tiers & Nodes**, right-click the node, and select **View**.
    b. When the Node view appears, go to the Network Browser.
2. Right-click the node and select **Start Packet Capture**.

Resolve Unmapped Connections

After you set up the App Server Agent and the Network Agent together, the Network Agent automatically:

- Monitors TCP connections
- Identifies the application flow that uses each connection
- Maps the connection to the flow

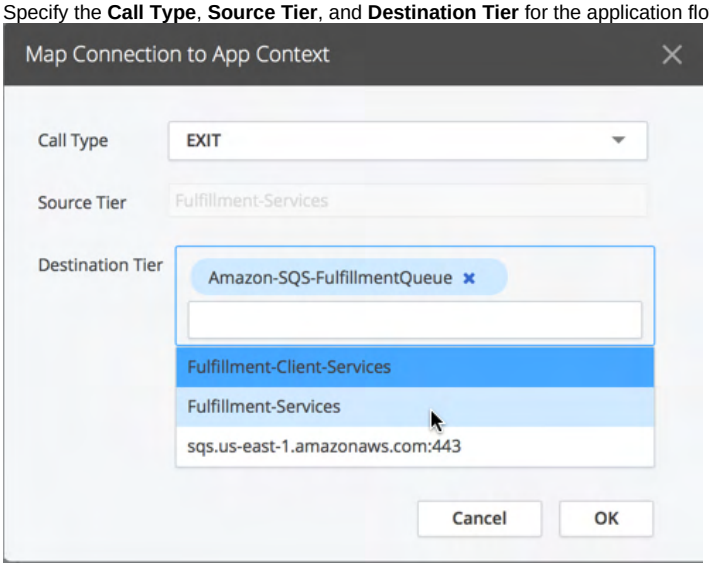
However in some cases, there is not enough context to associate a connection with a specific application flow resulting in an *unmapped* connection. In this case, you need to map the connection manually. If a connection remains unmapped, the Network Agent collects metrics for that connection but cannot associate these metrics with any monitored application flow.

To resolve an unmapped connection:

1. Go to the **Network** Dashboard and select **Mapping Wizard** (under **Connections** on the right). You can also open this wizard directly from a **Tier** Dashboard, if that tier has any unmapped connections.
2. The **Connection Mapping Wizard** appears. The left pane shows all tiers with unmapped connections. When you select a tier, the **Connections** list (right pane) shows all the unmapped connections for that tier. The workflow is to iterate through each tier, and specify the mapping for each connection for that tier.
   For each unmapped connection:
   a. Determine the application flow that uses the unmapped connection.

      > You need to know the source tier, destination tier, and the call type (`ENTRY/EXIT`) for the application flow. If you do not know this information, contact someone else who knows the application well.

   b. Select a tier in the left pane.
   c. In the right pane, group the IP addresses by Destination VIP or Prefix (bit mask).
   d. Right-click a connection and select **Map Connection(s)**.
   e. Specify the **Call Type**, **Source Tier**, and **Destination Tier** for the application flow that uses that connection.



Troubleshooting Network Visibility Problems

How do I troubleshoot the Network Agent?

Network Agent Logs

If you notice any performance issues related to the Agent:

1. Stop the Network Agent:
    a. For ZIP packages: Identify the Agent process and enter: `ps ef | grep \appd-netagent`
    The output provides the process ID (`PID`) of the Network Agent process: `kill <network_agent_PID>`
    b. For RPM and DEB packages, enter: `sudo service appd-netviz stop`
2. Review these log files under `<network-agent-install>/logs`:
    a. `appd-netmon.log`
    b. `appd-netagent.log.log`

Network Agent Health Rule

Use this default Health Rule to troubleshoot Network Agents: `Network-Host: Packet drops too high`
This rule triggers an alert when packets get dropped between a Network Agent and the host interface. A high rate of packet drops on the host can result in inaccurate metrics.

> If the issue persists, run the `collect.sh` script available in the `bin/` folder in the `agent` directory, and share the logs with your support contact to expedite troubleshooting. This script is available for Network Agent >= 4.5.9.

Monitor Cloud Applications

In cloud environments, services and components are added and removed continuously. AppDynamics provides robust support to monitor applications in these dynamic environments.

Container and Orchestration Technologies

Docker

- Monitor Containers with Docker Visibility
- Blog: Composing Containers for Monitoring
- Blog: Leveraging Docker Store Images with Built-In AppDynamics

Kubernetes

- Monitoring Kubernetes with the Cluster Agent
- Using Docker Visibility with Kubernetes
- Network Visibility with Kubernetes
- Using Docker Visibility with Red Hat OpenShift
- Webinar: Monitoring Kubernetes in the Cloud on EKS, AKS, and GKE

Cloud Platforms

Amazon Web Services

- Serverless APM for AWS Lambda
- AWS Controller Deployment Guide
- Customize Flow Maps
- AWS Example with Elastic Beanstalk
- AWS Monitoring Extensions
- Blog: How to Monitor the AWS Cloud and Save Money
- Blog: Amazon ECS or EKS? AppDynamics Supports Both
- Blog: Monitor Amazon EKS with AppDynamics

Google Cloud Platform (GCP)

- Webinar: Monitoring Kubernetes in the Cloud on EKS, AKS, and GKE
- Webinar: Automating the Software Development Lifecycle for Kubernetes

Microsoft Azure

- Deploy AppDynamics for Azure
- Install AppDynamics for Azure App Service
- Install AppDynamics for Azure Service Fabric
- Install AppDynamics for Azure Cloud Services
- Blog: How to Monitor .NET Core Apps

Hybrid Platforms

VMware Tanzu (formerly Pivotal Cloud Foundry)

Pivotal is now a part of the VMware Tanzu portfolio

- AppDynamics Application Performance Monitoring for VMware Tanzu
- AppDynamics Platform Monitoring for VMware Tanzu
- AppDynamics Application Analytics for VMware Tanzu

VMWare Tanzu Example Workflows:

- Monitoring GoLang Apps
- Monitoring Java Apps
- Monitoring .NET Framework Apps
- Monitoring .NET Core Linux Applications
- Monitoring .NET Core Windows Applications
- Monitoring Node.js Applications
- Monitoring Python Apps
- Blog: Getting Started with Pivotal Cloud Foundry and AppDynamics
- Webinar: Monitoring Pivotal Cloud Foundry Applications and Infrastructure

Red Hat OpenShift Version 3

- Monitoring Red Hat OpenShift Clusters
- Webinar: Kubernetes and OpenShift v3 with AppDynamics

Monitor Containers with Docker Visibility

Use the Machine Agent to monitor application nodes running inside Docker containers and to identify container issues that impact application performance. By viewing and comparing APM metrics with the underlying container and server/machine metrics, you can easily answer the question: Is my application problem purely an application problem, or is the root cause in the container or the server?

> Container monitoring requires a Server Visibility license >=4.3.3, for both the Controller and the Machine Agent.

You should deploy the Machine Agent inside a Docker container. The Machine Agent collects metrics for Docker containers on the same host, and server and machine metrics for the host itself. The Controller shows all monitored containers, for each host, and the container and host IDs, for each container.

In a BRIDGE networking mode, the containers take on the container ID as the host name. If networking is in host mode, then the containers take on the node name of the host ID. This means every container on that node has the same host ID. In this case you need to use the unique host ID settings. See Register the Container ID as the Host ID. When using Docker Visibility, if the unique host ID setting is not configured to use container ID in host network mode, the Machine Agent automatically registers the container using the container ID as the host ID. If you have an older version of the Controller or Machine Agent, AppDynamics recommends that you upgrade to Machine Agent version 20.7 or later. See Using Docker Visibility with Kubernetes.

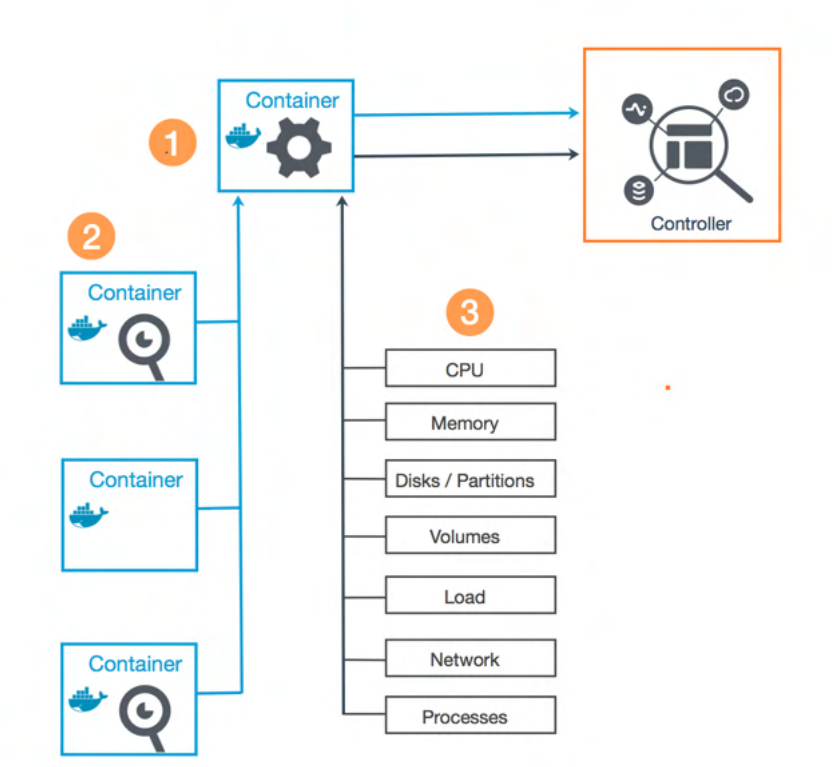
With Controller >=20.11.0:

- If the Machine Agent >=20.7.0, the Machine Agent automatically registers the container using the container ID as the host ID. No further action is needed.
- If the Machine Agent <=20.6.0 is configured incorrectly, the Controller rejects the misconfigured containers registration.

By default, the Machine Agent only monitors containers that have a running APM Agent. You can change this by setting the `sim.docker.monitorAPMContainersOnly` property on the Controller. See Controller Settings for Server Visibility.

> To deploy a Machine Agent on a host outside a Docker container, create a symbolic link: `ln -s / /hostroot` on the host. This symbolic link enables the Machine Agent to collect host metrics with Docker container metrics. When you deploy a Machine Agent inside a Docker container for monitoring, the symbolic link is automatically created when the volume mounts. To grant more restrictive permissions, enter this command to create symbolic links: `ln -s /proc /hostroot/proc`; `ln -s /sys /hostroot/sys`; `ln -s /etc /hostroot/etc`. You can make these links read-only because the AppDynamics Agent does not need write privileges to these directories.

This diagram illustrates how to deploy container monitoring:



- Install the Machine Agent (1) in a standalone container. The Machine Agent collects hardware metrics for each monitored container, as well as Machine and Server metrics for the host (3), and forwards the metrics to the Controller.

- The Machine Agent can monitor all containers that are running on that host, subject to established limits, and will report runtime metrics and metadata for every container. Additionally, if any of the containers have an APM Agent installed (2), then the Machine Agent also correlates the container metadata and runtime metrics with the associated APM Node.

Before You Start

1. AppDynamics recommends that you use Docker CE/EE 17.03 or Docker Engine 1.13 with this product. Some data might be missing if you use previous versions of Docker.
2. Container Monitoring is not supported on Docker for Windows, or Docker for Mac.
3. AppDynamics strongly recommends that you follow these guidelines, Instrument JVMs in a Dynamic Environment when monitoring container-based applications.
4. The Machine Agent can monitor up to 120 running containers per host. The default maximum number of containers per host is 100. With the cgroup enabled flag set to true, the Machine Agent can monitor up to 600 containers per host. See Configure Docker Visibility.
5. The Machine Agent collects metrics from containers with one or more running processes whose command line matches a configurable regex. By default, this regex matches all processes (*). To override this behavior, edit this regex in `<machine_agent_home>/extensions/DockerMonitoring/conf/DockerMonitoring.yml`:
   `containerMonitoringConfig:containerProcessSelectorRegex: ".*"`
6. The maximum number of containers you can monitor in one Controller depends on the Controller size, the total number of App Agents, and the current load. If you are using a SaaS Controller, please work with your Account Manager.
7. Please review:
   a. Best Practices for setting up App Agents and Standalone Machine Agents.
   b. Docker Visibility issues in the latest Release Notes.

Enable Container Monitoring

To enable Container Monitoring:

1. On the Controller, log in to the Administration Console and verify that **sim.docker.enabled** is set to **true**.
2. On the Agent, enable Server Visibility and Docker Visibility.

Container Monitoring Setup

The quickest and easiest way to run the Machine Agent with Container Monitoring enabled is to use one of the official images from the Docker Store: https://store.docker.com/images/appdynamics. These images are produced by AppDynamics, based on certified base images from the Docker Community, and can either be run directly or used as base images for your own application containers. For full details of how to download and run containers based on these official images, see the documentation posted on the Docker Store. To build your own base images, the full source code for building these images is posted to GitHub. You can use this as a pattern for your own builds: https://github.com/Appdynamics/appdynamics-docker-images.

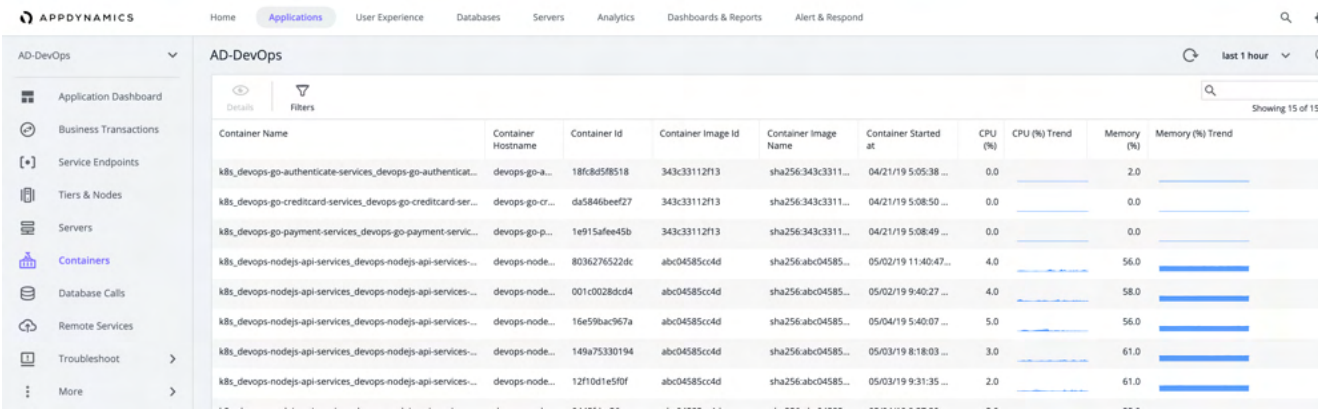
For the Machine Agent to monitor containers running on the server, configure these settings (See Configure Installation Options):

- **Server Visibility Enabled** – Enable Server Visibility
- **Docker Enabled** – Enable Docker Visibility
- **Volume Mounts** – Specify:
  - Volume mounts to allow read-only access to the underlying file system (`/proc`, `/etc` and `/sys`). This allows the Server Agent to collect host-level metrics for containers running on the server.
  - The UNIX domain socket on which the Docker daemon is configured to listen for API calls.

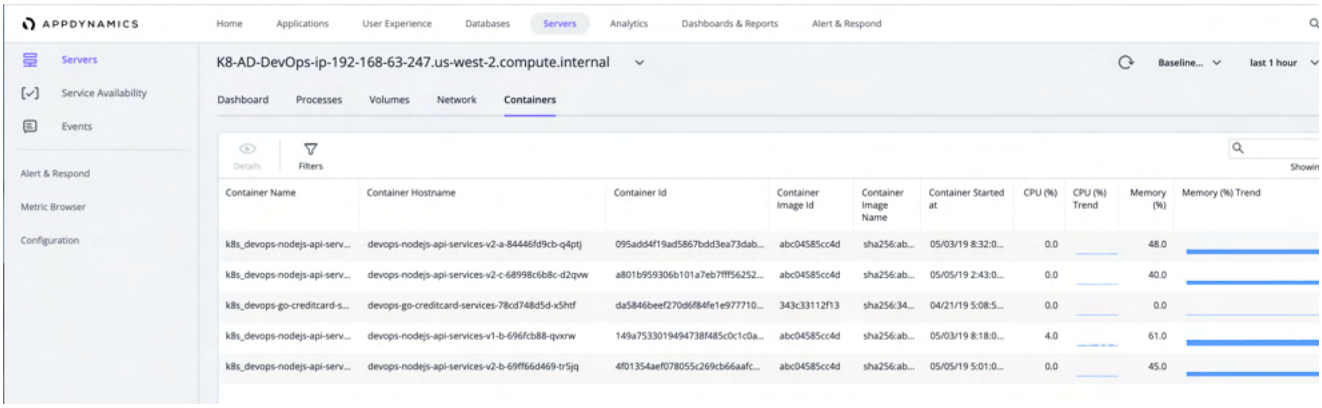
View Container Details

To view container metadata and metrics in the Controller:

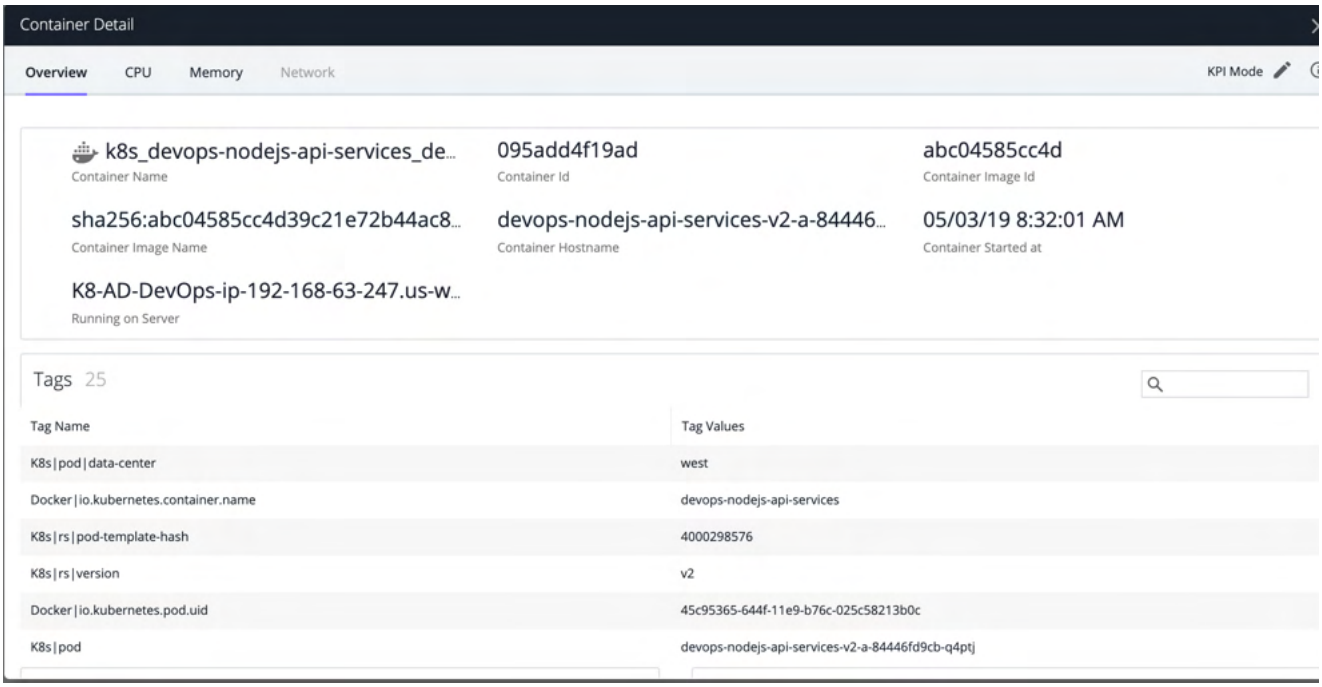
- In the **Applications** Dashboard, go to **Containers** to see all monitored containers used by the application.



- In the **Servers** Dashboard, go to **Containers** to see all monitored containers on that host.



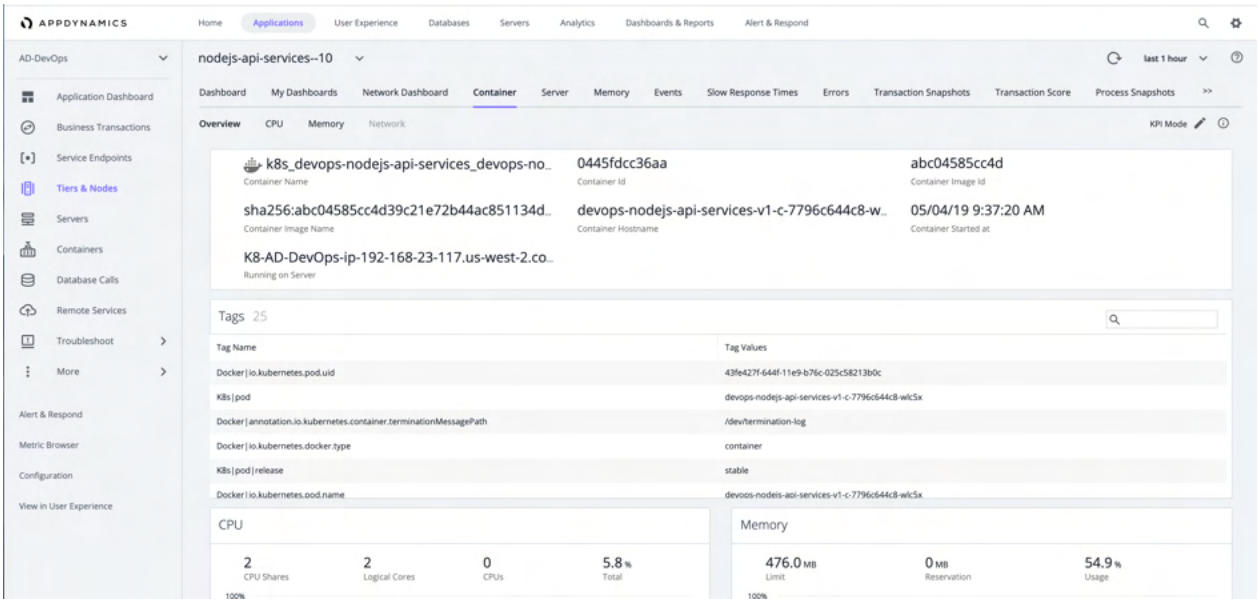
- To open the Container Dashboard, right-click on the container name, and choose **View Details**.



The Container Details view contains these tabs, which provide an overview of the health and resource usage for the container:

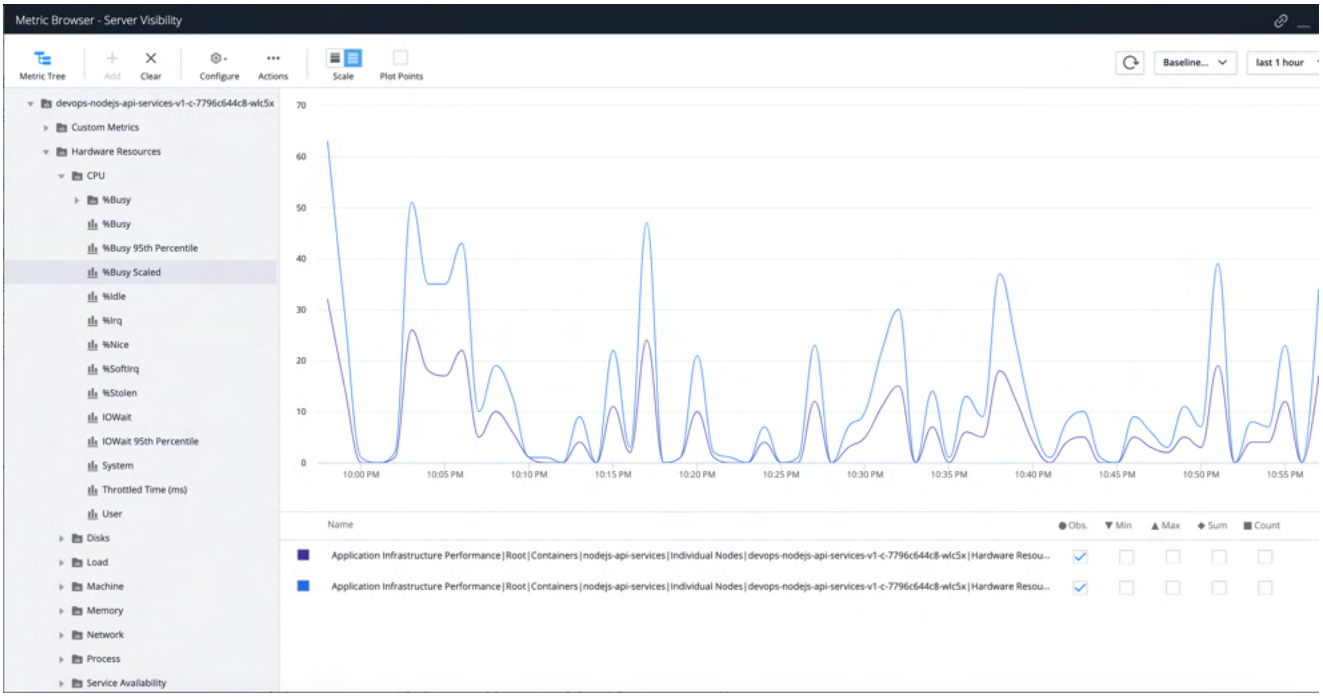
- **Overview** - Container metadata, Tags (name-value pairs derived from Docker/Kubernetes) plus AWS tags where applicable, and single chart views for CPU, memory, network and disk usage
- **CPU** - CPU Usage and Throttled Time metrics
- **Memory** - Memory Usage and Memory Fault metrics
- **Network** - Network Usage metrics (only available when running in Diagnostic mode), see "Enable Dynamic Monitoring Mode (DMM)" under Machine Agent Configuration Properties.

- The Node Dashboard also includes a Containers tab for the container in which that node is running.



View Container Metrics Using the Metric Browser

To view time-series metric data for containers, double-click one of container metric graphs (CPU, Memory, Network, Disk) to open the Metric Browser with the displayed metric selected. The Metric Browser tree displays the full set of metrics available for that container, and you can add these to the Metric Browser display by double-clicking the metric you wish to select. See Container Metrics.



Troubleshooting Containers Using Dynamic Monitoring Mode

Using Dynamic Monitoring Mode (DMM), you can selectively control the number of Container metrics reported for individual containers. Rather than have all Agents report all metrics for all containers all the time, you can configure individual Agents to collect:

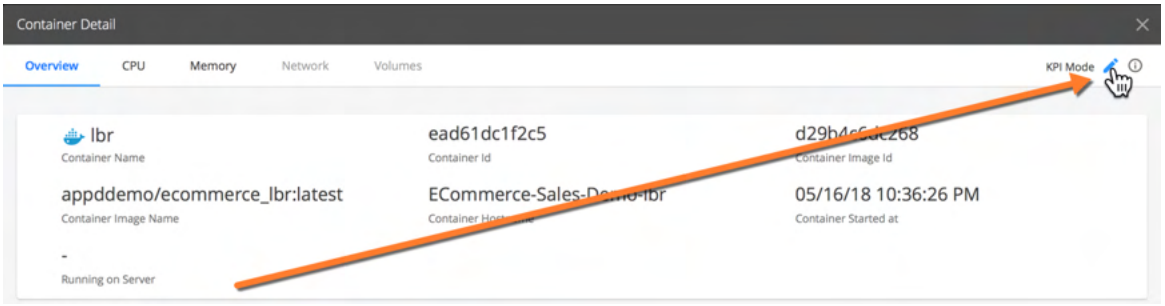
- Key Performance Indicator metrics only (*KPI mode*)
- KPI and Diagnostic metrics (*Diagnostic mode*)
- All available metrics (*Advanced Diagnostic mode*)

This provides the flexibility to report KPI metrics only on most containers, and then increase the metric level on specific containers when you need deeper visibility to diagnose problems. You can increase scalability on the Controller and conserve metric bandwidth on the network with no sacrifice in visibility.

Every Docker Visibility metric has a default DMM class (KPI, Diagnostic, and Advanced Diagnostic). For details, consult the Container Metrics table: the Default Monitoring Mode column indicates the default category of each metric when Dynamic Monitoring Mode is enabled.

When you notice a container that requires additional analysis or troubleshooting, change the DMM on that container to collect more metrics:

1. Open the **Container Detail** panel for the container of interest: Open the Containers table (**Containers**, left menu) and double-click the container.
2. Click the **Edit** (pen) button in the top-right corner, change the mode to **Diagnostic** or **Advanced Diagnostic**, and click the checkmark to apply the changes.
3. Collect additional metrics for your analysis or troubleshooting.
4. When you no longer need to collect additional metrics for that container, set the DMM on that container back to KPI.



Note:

- Depending on the type of metrics collected by the Agent, the Controller has three different DMM defaults. These defaults are all set to KPI (the lowest setting):
  - `sim.machines.dmm.containerAware.defaultMode`
    Default for Hardware Resources Metrics on servers with Docker Visibility enabled
  - `sim.machines.dmm.container.defaultMode`
    Default for Docker Container Metrics on containers
- The default for both settings is KPI (the lowest setting).
- AppDynamics recommends that you leave these global settings at their defaults. If you need to collect advanced metrics for a specific server or container, increase the DMM on an "as-needed" basis.
- When a container monitored by a Java App Agent shuts down and restarts, any overridden DMM specified for the shut-down container is lost. The DMM for the restarted container automatically resets to the global default specified by `sim.machines.dmm.container.defaultMode`.
- If you switch the monitoring mode in the Controller from a more-inclusive to a less-inclusive mode, the Metric Browser will show values for the newly-excluded metrics for one hour after the switch. For example, if you switch from Diagnostic to KPI mode. For any Diagnostic metric, the Metric Browser will report a steady line (at 0, or the last-reported value) for one hour after the switch; then the line will disappear. This is standard behavior in the Metric Browser for an Agent when it stops reporting a specific metric.

Configure Docker Visibility

Machine Agent Settings for Docker Visibility

You can configure the Docker Visibility settings in the file: *<machine-agent-home>*/extensions/DockerMonitoring/conf/DockerMonitoring.yml

Before you edit this file, please review "Editing YML Files: Important Notes" in Machine Agent Settings for Server Visibility.

Property	Description	Default
cgroupEnabled	The Machine Agent can collect Memory and CPU metrics for Docker containers using cgroupdata. This method is much faster than using the Docker API (the default metric-collection method). You can enable cgroupmetriccollection on all platforms except Amazon Linux OS. With the cgroupEnabled flag set to true, the Machine Agent can monitor up to 600 containers per host. When cgroup is enabled, you do not have to update the Docker Metric Collector Pool Size. The max container limit is independent of this. This mode is disabled by default. To enable cgroupmetriccollection, run the agent with the following command-line option set to **true**: `java -Dappdynamics.docker.container.collection.cgroup.enabled=true -jar machineagent.jar`	true
containerProcessSelectorRegex	The Machine Agent can monitor up to 150 running containers (depending on the pool size configuration, described below). The agent collects metrics from containers with one or more running processes whose command line matches a configurable regex. By default, the value of this regex is ".*" and matches all containers irrespective of the processes running in them. If you cannot restart the APM agents, or if you to want to filter APM agents based on another command-line argument, you can override this behavior by editing this regex. Default = ".*"	".*"
docker.metric.collector.pool.size	Specify the local limit for the number of containers the agent can monitor. The effective limit for each agent is the minimum amount for: • The sim.docker.machine.container.limit on the Controller (default limit = 15 containers), or • The Docker Metric Collector Pool Size on the agent (default pool size = 3, or 90 containers). You can change this setting to monitor up to 120 containers (pool size = 4) or 150 containers (pool size = 5). The trade-off is that increasing the pool size might lead to higher resource consumption on the agent host. To change the pool size, set these properties: • System Property: -Dappdynamics.docker.metric.collector.pool.size • Environment Variable: APPDYNAMICS_DOCKER_METRIC_COLLECTOR_POOL_SIZE	default limit = 15 containers default pool size = 3, or 90 containers
dockerTagsEnabled	Enable (default) or disable the collection of Docker tags from containers monitored by the Machine Agent.	

`docker.use.container.name.selector`	Use the Machine Agent to identify containers by container names only.
	
	By default for Docker monitoring, the Machine Agent obtains a list of containers. For each container, the Machine Agent retrieves all running processes inside the container. Then, the Machine Agent matches the container to be monitored by the process full command line. Because there are many containers, the selection process lasts longer than a minute, causing gaps in reporting container metrics.
	
	The Docker Use Container Name Selector configuration option provides an alternative way to select containers to be monitored by using the container's name. This option allows the Machine Agent to run significantly faster since the Machine Agent does not have to gather all the processes for each container.
	
	To enable matching with the container name, you can either use an environment variable or a JVM property. If both are defined, the environment variable is used.
	
	• System Property: `-Dappdynamics.docker.use.container.name.selector=false`
	• Environment Variable: `APPDYNAMICS_DOCKER_USE_CONTAINER_NAME_SELECTOR=false`
	
	You can select which containers to be monitored by providing the regular expression, `containerNameSelectorRegex`, to match the container names. By default, the Machine Agent matches all container names. Regular expressions are part of the `extensions/DockerMonitoring/conf/DockerMonitoring.yml` file.
	
	To exclude containers from monitoring, use an environment variable or the JVM property. If both are defined, the environment variable is used.
	
	`# default value`
	`".*(machine-agent|machineagent).*"`
	
	`# environment variable`
	`export APPDYNAMICS_DOCKER_CONTAINER_NAME_SELECTOR_BLACKLIST_REGEX=".*(machine-agent|machineagent).*"`
	
	`# JVM property`
	`-Dappdynamics.docker.container.name.selector.blacklist.regex=".*(machine-agent|machineagent).*"`
		`# default value ".*(machine-agent|machineagent).*"`

Controller Settings for Docker Visibility

For information about how to configure these settings, see Controller Settings for Standalone Machine Agents.

Property	Description	Default
`sim.docker.apmNode.markHistorical.enabled`	Enables configuration to mark the APM node historical when a container running the app agent is stopped.	`true`
`sim.docker.enabled`	Enable the Docker Monitoring feature.	`true`
`sim.docker.machine.container.limit`	Global limit for the number of containers that each Machine Agent can monitor. The effective limit for each agent is the minimum of • The `sim.docker.machine.container.limit` on the Controller (default limit = 15 containers), or • The Docker Metric Collector Pool Size on the agent (default pool size = 3, or 90 containers). You can specify this in the Administration Console as a Controller setting (all accounts) or as an Account setting for individual accounts. 150 is the maximum limit you can specify.	15
`sim.machines.reuse.enabled`	Reuse SIM Machine entities to handle an ephemeral environment. Support is currently limited to Docker container machines. If set to `false`, each new container is considered a new machine.	`true`
`sim.machines.tags.enabled`	Enable or disable the import of Docker tags to the Controller.	`true`

Use Docker Visibility with Kubernetes

This page describes how to set up the Machine Agent with Docker Visibility enabled to run as a `DaemonSet` on a Kubernetes cluster.

Container visibility enables you to monitor containerized applications running inside Kubernetes pods and identify container issues that impact application performance. You deploy the Machine Agent as a Kubernetes `DaemonSet` in every node of a Kubernetes cluster. Deploying the Machine Agent as a `DaemonSet` ensures that every Kubernetes worker node runs the Machine Agent to collect critical resource metrics from the node host and associated Docker containers.

> Using Docker Visibility to monitor Kubernetes containers is no longer the preferred option and will be deprecated. Use the Cluster Agent instead as described in Monitor Kubernetes with the Cluster Agent. The Cluster Agent supports Kubernetes container visibility, as well as visibility into cluster health and capacity.

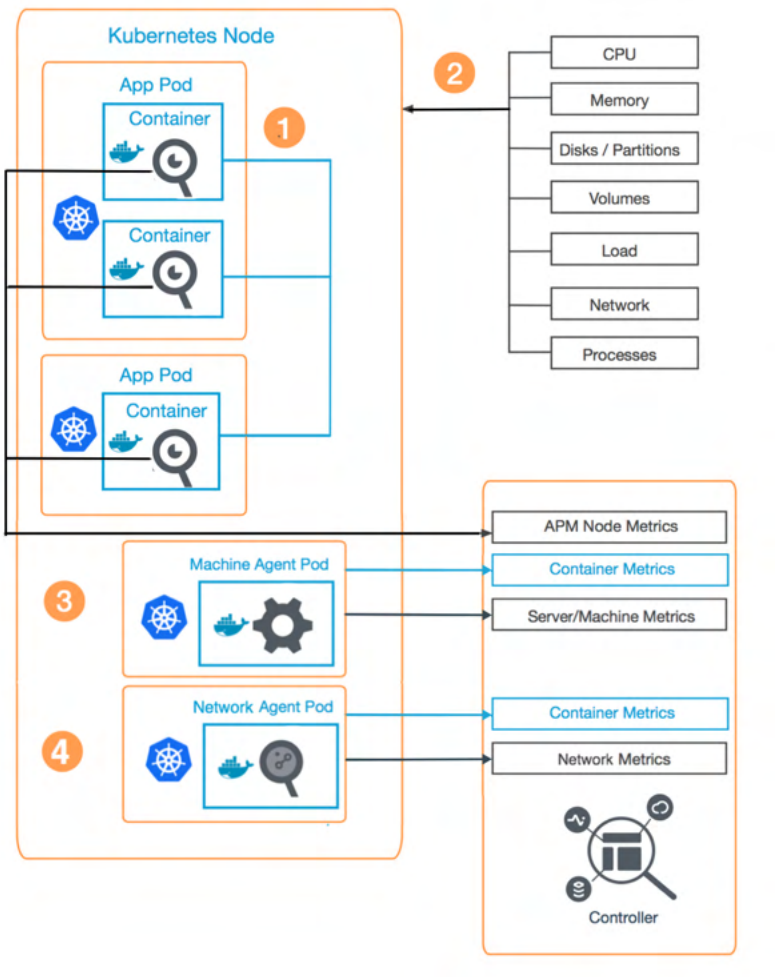
Container Visibility with Kubernetes

Deploy the Machine Agent in Docker-enabled mode. To configure and run the Machine Agent using Docker, see Configure Docker Visibility.

The Machine Agent:

- Identifies the containers managed by Kubernetes.
- Determines if these containers contain app server agents.
- Correlates containers with app server agents with the APM nodes for that application.

This diagram depicts the deployment scenario for container visibility in Kubernetes:



- Install the Machine Agent container (1) as a `DaemonSet` on each Kubernetes node.
- To collect APM metrics from any container in a pod, install the correct App Server Agent (2) in the container before deploying the pod.

- The Machine Agent collects resource usage metrics for each monitored container (3), as well as Machine and Server metrics for the host, and forwards the metrics to the Controller.
- (Optional) Install the Network Agent (4) as a `DaemonSet` on the node you want to monitor. The Network Agent collects metrics for all network c onnections between monitored application components and sends these metrics to the Controller.

Before You Begin

Review the requirements for Container Visibility with Kubernetes:

- A Machine Agent must run as a `DaemonSet` on every Kubernetes node you want to monitor.
- Every node to be monitored must have a Server Visibility license.
- Docker Visibility must be enabled on each Machine Agent.
- Both App Server Agents and Machine Agents are registered by the same account and are using the same Controller.
- If you have multiple App Server Agents running in the same pod, register the container ID as the host ID on each App Server Agent and Machine Agent.

Limitations

- Only the Docker Container Runtime is supported.
- Only Pod and ReplicaSet labels are supported.

Procedure

1. Enable Container Visibility
2. Register the Container ID as the Host ID
3. Configure the Cluster Role
4. Deploy the Machine Agent on Kubernetes

After the Machine Agent has been deployed on Kubernetes, you can add the App Server Agent into your image:

- Instrument Applications with Kubernetes
    - Resource Limits

Enable Container Visibility

Update the Controller to >= 4.4.3.

To enable Kubernetes Visibility in your environment, edit these parameters:

- Controller
    - `sim.machines.tags.k8s.enabled`: The value defaults to `true`. The global tags enabled flag has priority over this.
    - `sim.machines.tags.k8s.pollingInterval`: The value defaults to one minute. The minimum value you can set for the polling interval is 30 seconds.
- Machine Agent
    - `k8sTagsEnabled`: The value defaults to `true` and is specified in the `ServerMonitoring.yml` file.

Continue with Use Docker Visibility with Red Hat OpenShift. You can use the example `DaemonSet`, the sample Docker image for Machine Agent, and the sample Docker start script to set up the Machine Agent.

Register the Container ID as the Host ID

Install an app server agent in every container in a Kubernetes pod to collect application metrics. If multiple app server agents are running in the same pod, for example, in the RedHat OpenShift platform, you must register the container ID as the unique host ID, on both the app server agent and the Machine Agent, to collect container-specific metrics from the pod. Kubernetes pods can contain multiple containers and they share the same host ID. The Machine Agent cannot identify different containers running in a pod unless each container ID is registered as the host ID.

To register the container ID as the host ID:

1. Get the container ID from the `cgroup`.

    ```
    cat /proc/self/cgroup | awk -F '/' '{print $NF}'  | head -n 1
    ```

2. Register the app server agents.

    ```
    -Dappdynamics.agent.uniqueHostId=$(sed -rn '1s#.*/##; 1s/(.{12}).*/\1/p' /proc/self/cgroup)
    ```

3. Register the Machine Agent.

    ```
    -Dappdynamics.docker.container.containerIdAsHostId.enabled=true
    ```

Configure the Cluster Role

This sample cluster role definition provides read access to various Kubernetes resources. These permissions enable Kubernetes extensions to the Machine Agent and pod metadata collection. The role is called `appd-cluster-reader`, but you can rename it. The cluster role definition outlines various API groups that will be available for members of this role. For every API group, we define a list of resources that will be accessed and the access method. Because we need to retrieve information from these API endpoints, we need the read-only access, expressed by "`get`", "`list`," and "`watch`" verbs.

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: appd-cluster-reader
rules:
- nonResourceURLs:
      - '*'
  verbs:
      - get
- apiGroups: ["batch"]
  resources:
    - "jobs"
  verbs: ["get", "list", "watch"]
- apiGroups: ["extensions"]
  resources:
    - daemonsets
    - daemonsets/status
    - deployments
    - deployments/scale
    - deployments/status
    - horizontalpodautoscalers
    - horizontalpodautoscalers/status
    - ingresses
    - ingresses/status
    - jobs
    - jobs/status
    - networkpolicies
    - podsecuritypolicies
    - replicasets
    - replicasets/scale
    - replicasets/status
    - replicationcontrollers
    - replicationcontrollers/scale
    - storageclasses
    - thirdpartyresources
  verbs: ["get", "list", "watch"]
- apiGroups: [""]
  resources:
    - bindings
    - componentstatuses
    - configmaps
    - endpoints
    - events
    - limitranges
    - namespaces
    - namespaces/status
    - nodes
    - nodes/status
    - persistentvolumeclaims
    - persistentvolumeclaims/status
    - persistentvolumes
    - persistentvolumes/status
    - pods
    - pods/binding
    - pods/eviction
    - pods/log
    - pods/status
    - podtemplates
    - replicationcontrollers
    - replicationcontrollers/scale
    - replicationcontrollers/status
```

```
    - resourcequotas
    - resourcequotas/status
    - securitycontextconstraints
    - serviceaccounts
    - services
    - services/status
  verbs: ["get", "list", "watch"]
- apiGroups:
  - apps
  resources:
    - controllerrevisions
    - daemonsets
    - daemonsets/status
    - deployments
    - deployments/scale
    - deployments/status
    - replicasets
    - replicasets/scale
    - replicasets/status
    - statefulsets
    - statefulsets/scale
    - statefulsets/status
  verbs:
    - get
    - list
    - watch
- apiGroups:
  - apiextensions.k8s.io
  resources:
    - customresourcedefinitions
    - customresourcedefinitions/status
  verbs:
    - get
    - list
    - watch
- apiGroups:
  - apiregistration.k8s.io
  resources:
    - apiservices
    - apiservices/status
  verbs:
    - get
    - list
    - watch
- apiGroups:
  - events.k8s.io
  resources:
    - events
  verbs:
    - get
    - list
    - watch
```

Once the role is defined, you must create cluster role bindings to associate the role with a service account. This example of a ClusterRoleBinding spec makes `appd-cluster-reader` service account a member of the `appd-cluster-reader-role` in the project "`myproject`". The naming is purely coincidental. The names of the service account and the cluster role do not have to match.

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: cluster-reader-role-binding
subjects:
- kind: ServiceAccount
  name: appd-cluster-reader
  namespace: myproject
roleRef:
  kind: ClusterRole
  name: appd-cluster-reader
  apiGroup: rbac.authorization.k8s.io
```

Deploy the Machine Agent on Kubernetes

You can deploy the AppDynamics Machine Agent in a single container image without an init container. By default, the Machine Agent is deployed to the cluster as a `DaemonSet` and distributes every Agent instance evenly across all cluster nodes. When required, you can configure the `DaemonSet` with node affinity rules or node anti-affinity rules to ensure that it is deployed to a desired set of nodes and not across the entire cluster. See Assigning Pods to Nodes to learn about node affinity.

To harvest pod metadata, the service account used to deploy the Machine Agent must have the `cluster-reader` role in OpenShift. The `cluster-reader` role is also required for the Kubernetes extensions to the Machine Agent.

```
# assigning cluster-reader role in OpenShift
oc adm policy add-cluster-role-to-user cluster-reader -z appd-account
```

If you are working with a vanilla Kubernetes distribution, it may not have a pre-built cluster role similar to `cluster-reader` in OpenShift. See ClusterRole Configuration.

Instrument Applications with Kubernetes

There are several approaches to instrumenting applications deployed with Kubernetes; the method you choose depends on your particular requirements and DevOps processes. To monitor an application container with AppDynamics, you must include an App Server Agent in that container by:

- Using an appropriate base image which has the App Server Agent pre-installed.
- Loading the App Server Agent dynamically as part of the container startup using an init container.
- Loading the App Server Agent and dynamically attaching it to a running process (where the language runtime supports it).

The third option is usually applicable only to Java-based applications since the JVM supports Dynamic Attach, a standard feature of the AppDynamics Java APM Agent. See Dynamic Java Instrumentation.

For the other options, it is common practice to make use of standard Kubernetes features such as Init Containers, ConfigMaps, and Secrets (as described in Deploying AppDynamics Agents to OpenShift Using Init Containers).

Resource Limits

The main application being monitored should have resource limits defined. Provide 2% padding for CPU and add up to 100 MB of memory.

To support up to 500 containers, you can configure the Machine Agent with the following resource requests and limits: `Mem = 400M, CPU = "0.1"` and `Mem = 600M, CPU = "0.2"`

AppDynamics provides a Kubernetes Snapshot Extension to monitor the health of the Kubernetes Cluster. When deploying this extension, only a single version of the extension should be deployed to the cluster. Do not include it in the `DaemonSet` to avoid duplicates and potential cluster overload. Instead, consider deploying the Machine Agent instance with this extension, in addition to the `DaemonSet`, as a separate deployment with a single replica for Server Visibility. In this case, you can drop the memory request to 250 M and disable the Machine Agent SIM and Docker.

Use Docker Visibility with Red Hat OpenShift

This page describes how to set up the Machine Agent with Docker Visibility enabled to run as a `DaemonSet` on an OpenShift cluster.

Docker Visibility supports OpenShift Version 3. With the Machine Agent, you can collect performance data from OpenShift clusters. By installing the agent as a `DaemonSet` on each host in the OpenShift cluster, the `DaemonSet` monitors each host on the OpenShift cluster and the corresponding containers that have an AppDynamics App Agent running. This process is similar to that of monitoring the Kubernetes cluster.

Using Docker Visibility to monitor OpenShift containers is no longer the preferred option and will eventually be deprecated. Use the Cluster Agent instead as described in Monitor Kubernetes with the Cluster Agent. The Cluster Agent supports OpenShift container visibility, as well as visibility into cluster health and capacity.

Before You Begin

This functionality requires a Server Visibility license.

- You must have an OpenShift user account with a `cluster-admin` role, such as `system:admin`.
- Install the OpenShift command line tool (`oc`)
- AppDynamics recommends that you use Docker CE/EE v17.03 or Docker Engine >= v1.13 with this product. Some data may be unavailable if you are using previous versions of Docker.

Machine Agent Installation and Configuration

The process includes:

- Set the Java Options
- Register the Container ID as the Host ID
- Create Project and Service Account
- Secure the Machine Agent
    - Run the Machine Agent Without cluster-reader Role
    - Run the Machine Agent Without Privileged Container Mode
    - Security Permissions for the Service Account
- Create a Docker Image of the Machine Agent
- Deploy the Machine Agent as DaemonSet
    - Verify Machine Agent Status

Set the Java Options

By default, the `MaxMetaspaceSize` is set to 100 MB in OpenShift. If your application operates within a small margin of its existing memory resource allocation, increase the allocation for the application. AppDynamics recommends allocating additional Metaspace Size space to accommodate the machine agent. Directly configure the environment variable `GC_MAX_METASPACE_SIZE` to set the `MaxMetaspaceSize` parameter in the deployment through a config map.

Register the Container ID as the Host ID

Install an App Server Agent in each container in a Kubernetes pod to collect application metrics. If multiple App Server agents are running in the same pod (for example, in the RedHat OpenShift platform), you must register the container ID as the unique host ID on both the App Server Agent and the Machine Agent to collect container-specific metrics from the pod. Kubernetes pods can contain multiple containers and they share the same host ID. The Machine Agent cannot identify different containers running in a pod unless each container ID is registered as the host ID.

To register the container ID as the host ID:

1. Get the container ID from the `cgroup`.

    ```
    cat /proc/self/cgroup | awk -F '/' '{print $NF}'  | head -n 1
    ```

2. Register the App Server Agents.

    ```
    -Dappdynamics.agent.uniqueHostId=$(sed -rn '1s#.*/##; 1s/docker-(.{12}).*/\1/p' /proc/self/cgroup)
    ```

3. Register the Machine Agent.

```
-Dappdynamics.docker.container.containerIdAsHostId.enabled=true
```

Create Project and Service Account

In OpenShift terminology, a *project* is a mechanism to isolate a group of users and their resources. An administrator can provide individual users or groups with permission to create projects and/or manage specific projects.

To isolate the Machine Agent from other projects, create a machine agent project:

```
oc new-project machine-agent
```

To create a service account, `ma`, with necessary privileges for the Machine Agent to obtain metrics:

1. Verify that the current project is `machine-agent`.

   ```
   oc status
   ```

2. Create a service account.

   ```
   oc create serviceaccount ma
   ```

   Service accounts provide a secure way to control OpenShift API access without sharing a regular user's credentials.
3. Assign the privileged security context constraints (SCC) to the service account. SCC determines the permissions and abilities of pods.

   ```
   oc adm policy add-scc-to-user privileged -z ma
   ```

   See Security Permissions for the Service Account for the detailed list of permissions for the service account.
4. Add the `cluster-reader` role to the service account.

   ```
   oc adm policy add-cluster-role-to-user cluster-reader -z ma
   ```

Secure the Machine Agent

You can deploy the Machine Agent with tighter security. Determine the level of security for the project and leverage security permissions to secure the Machine Agent.

Run the Machine Agent Without cluster-reader Role

Without the `cluster-reader` role, the Machine Agent cannot read information such as Pod and ReplicaSet from the OpenShift cluster. The agent can, however, collect other metrics except for the tags in the app server agent container.

Run the Machine Agent Without Privileged Container Mode

To disable this privilege, remove the section below from the `DaemonSet` YAML file.

```
securityContext:
          privileged: true
```

Without the Privileged Container mode, the Machine Agent cannot read files like `/hostroot/etc/passwd` and `/hostroot/proc/<pid>/etc`. Therefore, it cannot collect metrics such as processes and network.

Disabling this privilege has no effect on collecting the App Server Agent container metrics.

Security Permissions for the Service Account

The Service Account requires the following security permissions to provide maximum isolation to the project under which the Machine Agent is running:

Permission	Description
The cluster-reader role	This cluster-reader role allows the Machine Agent to read tags from the OpenShift cluster. For example: `oc adm policy add-cluster-role-to-user cluster-reader -z ma`
The privileged SCC	The privileged SCC allows the Machine Agent to be run as the root user. For example: `oc adm policy add-scc-to-user privileged -z ma`

Run as privileged container	Running as the privileged container allows the Machine Agent to perform operations such as reading files from `/etc` and reading process information from `/proc`.
	
	This privilege is configured in the DaemonSet YAML file as follows.
	
	`securityContext:`
	`  privileged: true`

Create a Docker Image of the Machine Agent

Copy the following files to a directory on a machine that can build the Docker image.

- Machine Agent Bundle - 64-bit Linux (zip)
  Download this bundle and rename it to `machine-agent.zip`.
- Dockerfile
  See the sample Docker file below.
- `start-appdynamics` script
  See the sample script below.

Build the image with your desired method and make it available in your image registry.

Deploy the Machine Agent as `DaemonSet`

1. Modify the sample `DaemonSet` to point to your Controller:
   a. Select the worker node.

```
nodeSelector:
        node-role.kubernetes.io/compute: "true"
```

   b. Change the following Controller information in the sample `DaemonSet`.

```
containers:
     - env:
        - name: APPDYNAMICS_CONTROLLER_HOST_NAME
          value: "<controller-host-name>"
        - name: APPDYNAMICS_CONTROLLER_PORT
          value: "<controller-port>"
        - name: APPDYNAMICS_CONTROLLER_SSL_ENABLED
          value: "true"
        - name: APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY
          value: "<account-access-key>"
        - name: APPDYNAMICS_AGENT_ACCOUNT_NAME
          value: "<your-account>"
        - name: APPDYNAMICS_SIM_ENABLED
          value: "true"
        - name: APPDYNAMICS_DOCKER_ENABLED
          value: "true"
```

c. Specify the Docker image.

```
image: "<your image>"
```

d. Enable the Machine Agent to run as the privileged user.

```
securityContext:
          privileged: true
```

e. If you have a different service account name, replace it with the existing one.

```
serviceAccount: ma
      serviceAccountName: ma
```

2. Create `DaemonSet` below:
   a. Verify that the current project is machine-agent.

```
oc status
```

   b. Create the machine-agent `DaemonSet`.

```
oc create -f machine-agent-daemonset.yaml
```

   The Machine Agent appears in the UI.

Verify Machine Agent Status

If the Machine Agent does not appear in the UI, check the status as follows:

1. Check the machine-agent `DaemonSet`.

```
oc get ds
```

2. Check the machine-agent pod.

```
oc get pod
```

   The pod name is `machine-agent-daemonset-<XXXX>`.
3. Check logs whether the last line is "Started AppDynamics Machine Agent Successfully."

```
oc get logs -f <machine-agent-daemonset-<XXXX>>
```

Sample Machine Agent Deployment Files

Below are sample `DaemonSet`, Dockerfile and Start Script that you can use as a reference for deploying the Machine Agent.

Sample `DaemonSet`

Modify the sample `DaemonSet` to suit your deployment scenario.

```
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  name: machine-agent-daemonset
  namespace: machine-agent
spec:
  selector:
    matchLabels:
      name: appdynamics-machine-agent
  template:
    metadata:
      labels:
        name: appdynamics-machine-agent
    spec:
      nodeSelector:
        node-role.kubernetes.io/compute: "true"
      containers:
      - env:
        - name: APPDYNAMICS_CONTROLLER_HOST_NAME
          value: "<your-hostname>"
        - name: APPDYNAMICS_CONTROLLER_PORT
          value: "<your-port>"
        - name: APPDYNAMICS_CONTROLLER_SSL_ENABLED
          value: "true"
        - name: APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY
          value: "<your-access-key>"
        - name: APPDYNAMICS_AGENT_ACCOUNT_NAME
          value: "<your-account>"
        - name: APPDYNAMICS_SIM_ENABLED
          value: "true"
        - name: APPDYNAMICS_DOCKER_ENABLED
          value: "true"
        image: <your-image>
        name: machine-agent
        securityContext:
          privileged: true
        volumeMounts:
        - mountPath: /hostroot
          name: hostroot
          readOnly: true
        - mountPath: /var/run/docker.sock
          name: docker-sock
      restartPolicy: Always
      imagePullPolicy: Always
      serviceAccount: ma
      serviceAccountName: ma
      volumes:
      - name: hostroot
        hostPath:
          path: /
      - name: docker-sock
        hostPath:
          path: /var/run/docker.sock
```

Sample Dockerfile

You can use this Dockerfile sample or the one in OpenShift Visibility Manifests.

```
# Sample Dockerfile for the AppDynamics Machine Agent
# This is provided for illustration purposes only

FROM ubuntu:16.04
# Install required packages
RUN apt-get update && \
    apt-get install -y unzip && \
    apt-get clean
# Install AppDynamics Machine Agent
ENV MACHINE_AGENT_HOME /opt/appdynamics/machine-agent/
ADD machine-agent.zip /tmp/machine-agent.zip
RUN mkdir -p ${MACHINE_AGENT_HOME}
# Include start script to configure and start MA at runtime
ADD start-appdynamics ${MACHINE_AGENT_HOME}
RUN chmod 774 ${MACHINE_AGENT_HOME}/start-appdynamics
# change files and directories to be owned by root so MA container user in root
group can access
RUN chgrp -R 0 /opt && \
    chmod -R g=u /opt
RUN chgrp -R 0 /tmp/machine-agent.zip && \
    chmod g=u /tmp/machine-agent.zip
# Changing directory to MACHINE AGENT HOME
WORKDIR ${MACHINE_AGENT_HOME}
# Configure and Run AppDynamics Machine Agent
CMD "./start-appdynamics"
```

Sample start-appdynamics Script

```
#!/bin/bash
unzip -oq /tmp/machine-agent.zip -d ${MACHINE_AGENT_HOME} && \
    rm /tmp/machine-agent.zip
# Start Machine Agent
./bin/machine-agent -j jre/
```

Monitor Pivotal Cloud Foundry

VMware Tanzu Application Service (PCF)

VMware Tanzu Application Service (TAS), formerly known as Pivotal Cloud Foundry (PCF), is an open-source cloud computing Platform as a Service (PaaS). Developers can develop, deploy, operate, and scale cloud-native applications for public and private clouds.

AppDynamics provides three tiles to support VMware TAS 2.x (>= v2.0.7) and is available on the VMware Tanzu Network to download and install in a TAS foundation.

VMware Tanzu Kubernetes Grid Integrated (PKS)

VMware Tanzu Kubernetes Grid Integrated (TKGI), formerly known as Pivotal Container Service (PKS), is VMware's product used to deploy and manage open-source Kubernetes. The Kubernetes monitoring that AppDynamics provides has been tested to with TKGI. See Monitor Kubernetes with the Cluster Agent to set up container and cluster visibility within TKGI.

AppDynamics Application Performance Monitoring for VMware Tanzu

To simplify APM setup, the AppDynamics Application Performance Monitoring for VMware Tanzu tile delivers a service broker to support an AppDynamics marketplace service. You can configure the Controller parameters in the tile and automatically publish to the marketplace after you install the tile.

To bind a TAS app to the marketplace service to consume the Controller credentials, enter:

```
$ cf bind my-tas-app my-appd-marketplace-service
```

To use the integrated AppDynamics buildpack support to start the APM Agent based on the parameters from the marketplace service, and start sending performance metrics to the Controller, perform a push:

```
$ cf push
```

The tile also installs the AppDynamics extension buildpack in the TAS environment. See the tile documentation for buildpack support and language-specific workflows to enable APM

AppDynamics Platform Monitoring for VMware Tanzu

The AppDynamics Platform Monitoring for VMware Tanzu tile provides visibility into the TAS platform where you deployed your AppDynamics-monitored applications. The tile deploys a Firehose Nozzle that collects KPI metrics and publishes them to a Controller. The tile also deploys a Dashboard App that automatically creates KPI health rules and custom dashboards associated with the health rules. The Dashboard App enables you to monitor the health and capacity of one, or more, TAS foundations. See the tile documentation.

AppDynamics Application Analytics for VMware Tanzu

The AppDynamics Application Analytics for VMware Tanzu tile supports collecting Transaction Analytics (in real-time) from TAS apps to connect application performance, user experience, and business outcomes. The tile deploys an Analytics Agent in the TAS environment that forwards transaction analytics events from TAS apps to an AppDynamics Events Service. See the tile documentation.

Container Metrics

To access the **Containers** Dashboard, select: AppDynamics Home > **Servers** > **Containers**.

The **Containers** Dashboard shows resource utilization metrics of your containerized applications and the underlying host machine. You can access the **Containers** Dashboard from the Servers tab in the Controller UI. The **Containers** page lists all monitored containers used by an application that is registered to the Controller.

Container Metrics

CPU

Name	Description	Default Monitoring Mode
%Busy	Percentage of time the CPU was busy processing system or user requests from this container; this metric includes CPU Stolen time.	KPI

%Busy Scaled	If the -cpus=<value> option is set in the container, this metric measures the scaled value of the CPUs property. If this value is not set, this metric measures the scaled value of the Logical CPUs property.
	
	To enable CPU Scaling:
	• Log in to the Administration Console: `http://<controller host>:<port>/controller/admin.jsp`
	• Select the user account you want.
	• Select **Add Property** and add:
	`key` = `sim.docker.container.cpuScaling`
	`value` = `true`
		KPI

Logical Cores	Number of logical cores on the host machine that are available to the container.	KPI
Shares	The -c or cpu-shares setting used to start the container.	KPI
Throttle Count	Number of times the container CPU is throttled. Use **Advanced** mode to view this metric as Throttled Time. If Throttled Time is less and the container is limited on computation, then this metric is available for you to review.	KPI
Throttled Time (ms)	Total amount of time the container CPU was throttled but did not receive enough host CPU. High throttle times indicate that the -c or cpu-shares setting (used to start the container) was too low.	KPI
Total %	Average utilization, of all available CPUs, by the container.	KPI

Disks

Name	Description	Default Monitoring Mode
Average IO Utilization %	Average disk utilization for read/write operations. This metric may be unavailable on some platforms.	KPI
<*disk*> Reads/sec	Number of read operations by the container on <*disk*>.	Diagnostic
<*disk*> Writes/sec	Number of write operations by the container on <*disk*>.	Diagnostic
KB Read /sec	The rate of data read by the container from all disks and partitions.	KPI
KB Written /sec	The rate of data written by the container to all disks and partitions.	KPI
MB Reads /sec (MB)	Number of read operations by the container on all disks and partitions.	KPI

MB Writes /sec	Number of write operations by the container on all disks and partitions.	KPI
Queue Size	Average number of container requests queued during the time bucket. Consistently high queue sizes indicate that the container is not receiving enough IO resources. This metric may be unavailable on some platforms.	KPI

Memory

Name	Description	Default Monitoring Mode
Cache Size (MB)	Total amount of data cached in memory.	Diagnostic
Failed Count	Number of times a memory allocation failure occurred because it exceeded the pre-set memory limit for that container; or, number of times the hard limit for the container was reached.	KPI
Page Major Fault Count	Number of major page faults caused by the container. Docker metric is pgmajfault. A high rate of page faults could indicate that the specified memory constraints for the container are too low.	KPI
Reservation	Soft limit based on the `--memory-reservation` used to start the container.	KPI
Resident Set Size (MB)	Total amount of memory that belongs to stacks, heaps, and anonymous memory maps.	KPI
Usage	Memory usage by the container, in bytes.	KPI
Used %	Memory utilization by the container as a percentage of the memory hard limit, based on the `--memory` option used to start the container.	KPI
Used MB	Memory utilization by the container, in MB.	KPI

Network

Name	Description	Default Monitoring Mode[1]
Count (KB)	Network Volume, count (KB), incoming from an external network and outgoing to an external network.	KPI
Rate (KB/s)	Network Rate, rate (KB/s), incoming from an external network and outgoing to an external network.	Diagnostic
Incoming (KB)	Network traffic received by the container.	KPI
Outgoing (KB)	Network traffic sent by the container.	KPI
Incoming (KB/sec)	Rate of network traffic received by the container.	Diagnostic
Outgoing (KB/sec)	Rate of network traffic sent by the container.	Diagnostic
Incoming Packets	Number of packets received by the container.	Diagnostic
Outgoing Packets	Number of packets sent by the container.	Diagnostic
Incoming Packets Dropped	Number of incoming packets dropped before reaching the container. A high count of dropped packets may indicate high congestion or an issue on the network path.	KPI
Outgoing Packets Dropped	Number of outgoing packets dropped after leaving the container. A high count of dropped packets may indicate high congestion or an issue on the network path.	KPI
Incoming Errors	Number of packets received with errors.	KPI
Outgoing Errors	Number of packets sent with errors.	KPI
<eth0> Incoming Errors	Number of collision errors when <eth0> could not receive a frame due to a problem on the interface.	Diagnostic
<eth0> Outgoing Errors	Number of collision errors when <eth0> could not send a frame due to a problem on the interface.	Diagnostic
<eth0> Incoming KB	Average KB of traffic received on <eth0>.	Diagnostic
<eth0> Outgoing KB	Average KB of traffic sent on <eth0>.	Diagnostic

<eth0> Incoming KB /sec	Average rate of traffic received on <eth0>.	Diagnostic
<eth0> Outgoing KB /sec	Average rate of traffic sent on <eth0>.	Diagnostic
<eth0>Incoming Errors	Number of packets received on <eth0> with errors.	Diagnostic
<eth0> Outgoing Errors	Number of packets sent on <eth0> with errors.	Diagnostic
<eth0> Incoming Packets Dropped	Number of incoming packets dropped before reaching <eth0>. A high count of dropped packets may indicate high congestion or an issue on the network path.	Diagnostic
<eth0> Outgoing Packets Dropped	Number of outgoing packets dropped after leaving <eth0>. A high count of dropped packets may indicate high congestion or an issue on the network path.	Diagnostic

Monitor Kubernetes with the Cluster Agent

Deployment Support

The Cluster Agent monitors the health of Kubernetes and OpenShift clusters and is supported on all major Kubernetes distributions. The Cluster Agent collects metrics and metadata for the entire cluster, including every node and namespace down to the container level. When you instrument applications with AppDynamics APM Agents, you can use the Cluster Agent to view both Kubernetes and APM metrics for those pods (if both the Cluster Agent and the APM Agents are reporting data to the same account on a Controller).

You can use the Cluster Agent to:

- Gain visibility into key Kubernetes metrics and events, and detect uptime and availability issues
- Track resource usage of pods relative to the defined requests and limits
- Diagnose issues that may prevent uptime or scalability issues, such as:
  - Pod failures and restarts
  - Node starvation
  - Pod eviction threats and pod quota violations
  - Image and storage failures
  - Pending or stuck pods
  - Bad endpoints (detects broken links between pods and application components)
  - Service endpoints in a failed state
  - Missing dependencies (Services, ConfigMaps, and Secrets)

Deploy on Kubernetes

- Install the Cluster Agent
- Configure the Cluster Agent
- Install Infrastructure Visibility with the Cluster Agent Operator

Cluster Agent

- Use the Cluster Agent
- Cluster Metrics
- Administer the Cluster Agent
- Monitor Cluster Health
- Monitor Kubernetes Events

Reference and Troubleshooting

- Install the Cluster Agent
- Validate the Cluster Agent Installation
- Troubleshoot the Cluster Agent
- Overview of Cluster Monitoring
- Cluster Agent Requirements and Supported Environments
- Cluster Metrics

Overview of Cluster Monitoring

The AppDynamics Cluster Agent is a lightweight Agent written in Golang used to monitor Kubernetes and OpenShift clusters. You can use the Cluster Agent to monitor and understand how Kubernetes infrastructure affects your applications and business performance. With the Cluster Agent, you can collect metadata, metrics, and events for a Kubernetes cluster. The Cluster Agent is supported on Red Hat OpenShift and cloud-based Kubernetes platforms, such as Amazon EKS, Azure AKS, and Rancher.

Cluster Monitoring in the Controller

The Cluster Agent monitors events and metrics of Kubernetes or OpenShift clusters. It also tracks the state of most Kubernetes resources: pods, replica sets, deployments, services, persistent volumes, nodes, and so on. The data is received through the Kubernetes API server and is sent to the AppDynamics Controller. See Cluster Metrics.

Cluster Agent and Server Visibility with Docker Monitoring

AppDynamics Server Visibility monitors the worker nodes at a more detailed level. Server Visibility includes the ability to monitor running containers through Docker Monitoring. However, the Cluster Agent also monitors the same containers through the Kubernetes API server. You cannot run Server Visibility with Docker Monitoring enabled, and the Cluster Agent concurrently. Doing so may result in your data being overwritten, and may lead to unpredictable behaviors. See Install Infrastructure Visibility with the Cluster Agent Operator.

> When you deploy both Agents to the cluster, ensure that you deploy the Machine Agent with Docker Visibility disabled. See Monitor Containers with Docker Visibility.

Cluster Dashboard Metrics

The Cluster Dashboard provides an overview of potential issues with cluster health, grouped by category and severity. It shows error events, evictions, node resource starvation, distribution of pod phases, and issues associated with:

- Applications
- Cluster configuration
- Image or storage access
- Security access errors
- Quota violations

The dashboard contains cluster resource capacity stats and resource usage data relative to the deployment requests and limits for CPU, Memory, and Storage. The dashboard also provides real-time statistics on the state of monitored objects on the cluster, best-practice violations, and missing dependencies. See Monitor Cluster Health.

Cluster Agent Health Rules

You can create Cluster Agent Health Rules based on cluster metrics. Because Health Rules for clusters are created using server health rules, the health rule violations for clusters show as a server health rule violation. When creating the Health Rule, in the **Affected Entities** section, select **Custom** , and then select your Cluster Agent from the list of machines. When setting the Critical/Warning conditions, the entire metric tree displays. Select the Cluster Agent metrics for which to create the Health Rule. To create Health Rules for Cluster Agent metrics, follow the Create a Health Rule procedure s.

You can create Health Rules for these metrics:

- Number of error events
- Number of evictions
- Number of threats
- Nodes with disk pressure
- Nodes with memory pressure
- CPU/Memory utilization

Cluster Agent Requirements and Supported Environments

This page describes the software requirements, compatibility with different Kubernetes-based software, Cluster Agent distribution, licensing, and performance specifications.

Software Requirements

The Cluster Agent is designed to run on Linux and deployed using the AppDynamics Operator.

The Cluster Agent requires:

- AppDynamics Controller >= 20.3.0.
- Kubernetes versions, 1.11, 1.13, or >=1.14, with the Kubernetes metrics-server deployed and enabled on the cluster.
- A cluster that you can access and monitor.
- Sufficient Server Visibility licenses. The Cluster Agent consumes one Server Visibility license. See License Management
- Access to Docker Hub or Red Hat Container Registry to pull the Cluster Agent Operator and Cluster Agent images, or access to an internal repository where these images are maintained. See Install the Cluster Agent.

> If you deploy the Cluster Agent to monitor your Kubernetes Cluster, it does not monitor worker nodes. To monitor worker nodes, you must install the Machine Agent which consumes additional Server Visibility licenses (one Standalone Machine Agent per node). See Install Infrastructure Visibility with the Cluster Agent Operator

The Cluster Agent is compatible with these cloud platforms:

Cloud Platform	Version
Rancher Kubernetes Engine (RKE)	• 1.0.4 with Kubernetes 1.17 • 1.2.1 with Kubernetes 1.19.3
Kubernetes	1.11, 1.13, 1.14, 1.17, 1.19.3 with the Kubernetes metrics-server deployed
Amazon EKS	1.11, 1.13, 1.14, 1.15, 1.16, 1.17, 1.18, and 1.19 with the Kubernetes metrics-server deployed
Azure AKS	1.11, 1.13, 1.14, 1.18, 1.19, 1.20 with the Kubernetes metrics-server deployed
Google GKE	1.14, 1.15, 1.16, 1.17, 1.18, 1.19, and 1.20
Red Hat OpenShift	3.11, 4.1, 4.2, 4.3, 4.4, 4.5, 4.6 and 4.7 The metrics-server is not shipped with OpenShift, you must deploy it separately. The CRI-O container runtime is supported on >=4.2.7.
kubectl	>=1.11.3

> Installing AppDynamics to test or evaluate involves verifying system requirements, preparing the host, and then installing the Controller. See Install the Controller Using the Enterprise Console and Install the Controller Using the CLI.

Cluster Agent Distribution

You can download the Cluster Agent from the AppDynamics Download site as a zip archive. The archive also contains distribution-specific configuration files for the deployment.

Licensing

The Cluster Agent requires a Server Visibility license. To get started, see Enable Server Visibility. If you already have a Server Visibility license, see Install the Cluster Agent for installing Cluster Agent on a Kubernetes cluster.

Cluster Agent Performance Certification

Cluster Agent Version	Performance Certification
<=20.10	You can monitor the certified stable limit of 750 pods and 1500 containers
>=20.11	• Cluster Agent Vertical Scaling: This is the certified stable limit: If you run one container per pod then you can monitor 2250 pods and 2250 containers. If you run two containers per pod, then you can monitor 1500 pods and 3000 containers. • Cluster Agent Horizontal Scaling: If the number of pods per cluster exceeds the limit, then you can deploy multiple Cluster Agents to monitor the cluster using Helm charts. See Install the Cluster Agent with Helm Charts. For the limit of Cluster Agents per Controller, see Cluster Agent and Pod Limits.

Cluster Agent and Pod Limits

On-Premises

You can configure the number of Cluster Agents based on the overall controller capacity. To configure the limit, see the `sim.cluster.agent.limit` description under Controller Settings for the Cluster Agent.

If your Cluster has more pods than can be monitored, the Cluster Agent ensures that the group of pods being monitored remains the same over time. If any of the reported pods are terminated in the future, then the Cluster Agent replaces the terminated pods with previously unreported pods and starts monitoring the newly included pods.
The total pods metric displayed on the **Pods List** page reflects only the number of pods up to the configured limit. See the `sim.cluster.pod.limit` description under Controller Settings for the Cluster Agent.

SaaS

To configure the limits, create an AppDynamics Support ticket.

Install the Cluster Agent

This page describes how to install the Cluster Agent. For optional post-installation steps, including instrumenting applications with App Server Agents, see Post-Install Workflow.

Installation Overview

You can install the Cluster Agent using either the Kubernetes CLI (`kubectl`) or Helm. In both cases, the Cluster Agent Operator is installed first, followed by the Cluster Agent. The exact configuration applied during the installation differs slightly based on whether the cluster is an OpenShift or Kubernetes cluster.

Installation Workflow

You must make these decisions during the installation workflow:

- The first decision is whether to use the pre-built Cluster Agent images published on Docker Hub or the OpenShift Container Registry. This is the simplest option when using the Kubernetes CLI or Helm. If you need to build a custom image to satisfy internal image policies, or if you require that the pre-built Cluster Agent image is stored in an internal container registry, see Build the Cluster Agent Container Image.
- The second decision is whether to install the Cluster Agent using the Kubernetes CLI or the Cluster Agent Helm Chart. See Install the Cluster Agent with the Kubernetes CLI and Install the Cluster Agent with Helm Charts.

The Cluster Agent requires that you install the Metrics Server in the cluster. If the metrics server is not already installed under the `kube-system` name space, then you can install it using the Kubernetes CLI, as described in Install the Cluster Agent with the Kubernetes CLI. If you are using the Cluster Agent Helm chart, you can set the `install.metrics-server` Helm value to `true` to install the Metrics Server subchart. See Install the Cluster Agent with Helm Charts.

After the installation completes, see Validate the Cluster Agent Installation for validation and troubleshooting procedures.

Post-Install Workflow

After you install and Validate the Cluster Agent, you can instrument the Kubernetes applications in the cluster with AppDynamics Server Agents. See Contanier Installation Options for an overview of the options.

To install the AppDynamics Server Visibility Agent or Network Agent on cluster nodes, see Install Infrastructure Visibility with the Cluster Agent Operator.

To enable log collection for failing pods on on-premises Controllers, see Enable Log Collection for Failing Pods.

Next Steps

Install the Cluster Agent with the Kubernetes CLI

Install the Cluster Agent with Helm Charts

This page describes how to install the Cluster Agent. For optional post-installation steps, including instrumenting applications with App Server Agents, see Post-Install Workflow.

Install the Cluster Agent with the Kubernetes CLI

This page describes how to install the Cluster Agent using the Kubernetes CLI which is an alternative to Install the Cluster Agent with Helm Charts.

Requirements

Before you begin, verify that you have:

- Installed kubectl >= 1.11.3
- Met the requirements described here: Cluster Agent Requirements and Supported Environments

Installation Procedure

1. If you have not already installed the `metrics-server`, then install it. See https://github.com/kubernetes-sigs/metrics-server for the most recent installation instructions.

   ```
   kubectl create -f https://github.com/kubernetes-sigs/metrics-server/releases/download/v0.3.7/components.yaml
   ```

2. Download the Kubernetes or OpenShift Cluster Agent bundle from the Download portal, unzip the contents of the bundle to the current directory:

   ```
   unzip appdynamics-cluster-agent-alpine-linux-<version>.zip
   ```

3. Create a `namespace` for `appdynamics` in your cluster:

   ```
   kubectl create namespace appdynamics
   ```

4. Install the Cluster Agent Operator using the correct Kubernetes and OpenShift version (where applicable):

5. Based on the Account Access Key for the Controller, create the Controller Access Key Secret that the Cluster Agent reports to:

   ```
   kubectl -n appdynamics create secret generic cluster-agent-secret --from-literal=controller-key=<access-key>
   ```

6. Edit `cluster-agent.yaml` to set the AppDynamics Controller details for the Cluster Agent to report to.
   a. Set the `nsToMonitor` or `nsToMonitorRegex` options to include the namespaces you want to monitor.
   b. Set the image tag to the version you are installing, `cluster-agent:21.2.0`, for example.
   For additional configuration tasks, such as configuring SSL for on-premises Controllers, proxy support, or pull secrets, see Configure the Cluster Agent.

   This example assumes the use of the pre-built Cluster Agent image on Docker Hub. See Build the Cluster Agent Container Image to build your own image.

   ```
   apiVersion: appdynamics.com/v1alpha1
   kind: Clusteragent
   metadata:
     name: k8s-cluster-agent
     namespace: appdynamics
   spec:
     appName: "dev-cluster"
     controllerUrl: "http://mycontroller.com:8080"
     account: "my-account"
     # docker image info
     image: "docker.io/appdynamics/cluster-agent:21.2.0"
     serviceAccountName: appdynamics-cluster-agent
     nsToMonitor: [appdynamics,namespace1,namespace2]
     stdoutLogging: "true"
   ```

7. Install the Cluster Agent:

```
kubectl create -f cluster-agent.yaml
```

See Validate the Cluster Agent Installation to validate and troubleshoot the installation. Once the install is validated, see Container Installation Options for options to install App Server Agents in applications in the cluster.

Install the Cluster Agent with Helm Charts

This page describes how to use Helm Charts to deploy the Cluster Agent.

Helm is a package manager for Kubernetes. Helm charts are a collection of files that describe a set of Kubernetes resources. The Cluster Agent Helm chart is a convenient method to deploy the Cluster Agent Operator and Cluster Agent. You can also use the Cluster Agent Helm chart to deploy multiple Cluster Agents in a single cluster. This may be necessary for larger clusters that exceed the pod monitoring limit for a single Cluster Agent. See Cluster Agent Requirements and Supported Environments.

Requirements

- Cluster Agent version >= 20.6
- Controller version >= 20.6
- Cluster Agent Helm charts are compatible with Helm 3.0

Install a Single Cluster Agent in a Cluster

1. Add the chart repository to Helm:

   ```
   helm repo add appdynamics-charts https://appdynamics.github.io/appdynamics-charts
   ```

2. Create a namespace for `appdynamics` in your cluster:

   ```
   kubectl create namespace appdynamics
   ```

3. Create a Helm values file, in the example called `values-ca1.yaml`.
   Update the `controllerInfo` properties with the credentials from your Controller.
   Update the `clusterAgent` properties to set the namespace and pods to monitor. See Configure the Cluster Agent for information about the available properties `nsToMonitor`, `nsToMonitorRegex`, `nsToExcludeRegex` and `podFilter`.

   values-ca1.yaml

   ```
   # AppDynamics controller info
   controllerInfo:
     url: https://<controller-url>:443
     account: <appdynamics-controller-account>
     username: <appdynamics-controller-username>
     password: <appdynamics-controller-password>
     accessKey: <appdynamics-controller-access-key>

   # Cluster agent config
   clusterAgent:
     nsToMonitorRegex: dev-.*
   ```

   See Configuration Options for values.yaml for more information regarding the available options. Also, you can download a copy of `values.yaml` from the Helm Chart repository using this command:

   ```
   helm show values appdynamics-charts/cluster-agent
   ```

4. If you have not installed the Kubernetes `metrics-server` in the cluster (usually located in the `kube-system` namespace), then set `install.metrics-server` to `true` in the values file to invoke the subchart to install it.

   ```
   install:
     metrics-server: true
   ```

   > Setting `install.metrics-server` installs `metrics-server` in the namespace with the `--namespace` flag which is located in the same namespace as the Cluster Agent.

5. Deploy the Cluster Agent to the `appdynamics` namespace:

```
helm install -f ./values-ca1.yaml "<my-cluster-agent-helm-release>" appdynamics-charts/cluster-agent
--namespace=appdynamics
```

Enable Auto-Instrumentation

Once you have validated that the Cluster Agent was successfully installed, you can add additional configuration to the `instrumentationConfig` section of the values YAML file to enable auto-instrumentation. In this example, `instrumentationConfig.enabled` has been set to `true`, and multiple `instrumentationRules` have been defined. See Auto-Instrument Applications with the Cluster Agent.

values-ca1.yaml with Auto-Instrumentation Enabled

```
# AppDynamics controller info
controllerInfo:
  url: https://<controller-url>:443
  account: <appdynamics-controller-account>
  username: <appdynamics-controller-username>
  password: <appdynamics-controller-password>
  accessKey: <appdynamics-controller-access-key>

# Cluster agent config
clusterAgent:
  nsToMonitorRegex: ecom|books|groceries

instrumentationConfig:
  enabled: true
  instrumentationMethod: Env
  nsToInstrumentRegex: ecom|books|groceries
  defaultAppName: Ecommerce
  appNameStrategy: namespace
  imageInfo:
    java:
      image: "docker.io/appdynamics/java-agent:latest"
      agentMountPath: /opt/appdynamics
      imagePullPolicy: Always
  instrumentationRules:
    - namespaceRegex: groceries
      language: dotnetcore
      imageInfo:
        image: "docker.io/appdynamics/dotnet-core-agent:latest"
        agentMountPath: /opt/appdynamics
        imagePullPolicy: Always
    - namespaceRegex: books
      matchString: openmct
      language: nodejs
      imageInfo:
        image: "docker.io/appdynamics/nodejs:20.6.0"
        agentMountPath: /opt/appdynamics
        imagePullPolicy: Always
      analyticsHost: <hostname of the Analytics Agent>
      analyticsPort: 443
      analyticsSslEnabled: true
```

After saving the `values-ca1.yaml` file with the added auto-instrumentation configuration, you must upgrade the Helm Chart:

```
helm upgrade -f ./ca1-values.yaml "<my-cluster-agent-helm-release>" appdynamics-charts/cluster-agent --
namespace appdynamics
```

Configuration Options

Config option	Description	Required

`deploymentMode`	Used for multiple cluster agent deployment in a single cluster	Optional
Image config options (Config options under `imageInfo` key in `values.yaml`)		
`imageInfo.agentImage`	Cluster agent image address in format `<registryUrl>/<registryAccount>/<project>`	Optional (Defaults to the Docker Hub image)
`imageInfo.agentTag`	Cluster agent image tag/version	Optional (Defaults to latest)
`imageInfo.operatorImage`	Operator image address in format `<registryUrl>/<registryAccount>/<project>`	Optional (Defaults to the Docker Hub image)
`imageInfo.operatorTag`	Operator image tag/version	Optional (Defaults to latest)
`imageInfo.imagePullPolicy`	Image pull policy for the operator pod	Optional
Controller config options (Config options under `controllerInfo` key in `values.yaml`)		
`controllerInfo.accessKey`	AppDynamics Controller accessKey	Required
`controllerInfo.account`	AppDynamics Controller account	Required
`controllerInfo.authenticateProxy`	true/false if the proxy requires authentication	Optional
`controllerInfo.customSSLCert`	Base64 encoding of PEM formatted SSL certificate	Optional
`controllerInfo.password`	AppDynamics Controller password	Required only when auto-instrumentation is enabled.
`controllerInfo.proxyPassword`	Password for proxy authentication	Optional
`controllerInfo.proxyUrl`	Proxy URL if the Controller is behind some proxy	Optional
`controllerInfo.proxyUser`	Username for proxy authentication	Optional
`controllerInfo.url`	AppDynamics Controller URL	Required
`controllerInfo.username`	AppDynamics Controller username	Required only when auto-instrumentation is enabled.
RBAC config		
`agentServiceAccount`	Service account to be used by the Cluster Agent	Optional
`createServiceAccount`	Set to true if `ServiceAccounts` mentioned are to be created by Helm	Optional
`operatorServiceAccount`	Service account to be used by the AppDynamics Operator	Optional
Agent pod config		
`agentPod.nodeSelector`	Kubernetes node selector field in the Cluster Agent pod spec	Optional
`agentPod.tolerations`	Kubernetes tolerations field in the Cluster Agent pod spec	Optional
`agentPod.resources`	Kubernetes CPU and memory resources in the Cluster Agent pod spec	Optional
Operator pod config		
`operatorPod.nodeSelector`	Kubernetes node selector field in the AppDynamics Operator pod spec	Optional
`operatorPod.tolerations`	Kubernetes tolerations field in the AppDynamics Operator pod spec	Optional
`operatorPod.resources`	Kubernetes CPU and memory resources in the AppDynamics Operator pod spec	Optional
Install switches		
`install.metrics-server`	True if metrics are to be installed. Metrics-server is installed in the same namespace as the agent.	Optional

Install Additional Cluster Agents in a Cluster

The Cluster Agent Helm Chart supports multiple Cluster Agents installation in a cluster. This may be necessary for larger clusters that exceed the pod monitoring limit for a single Cluster Agent. See Cluster Agent Requirements and Supported Environments.

Each additional Cluster Agents that is deployed must have different configuration from any previously deployed Cluster Agents. This is achieved by limiting the monitoring to a distinct set of namespaces and pods using the `nsToMonitor`, `nsToMonitorRegex, nsToMonitorExcludeRegex` and `podFilter` properties. See Configure the Cluster Agent.

The first Cluster Agent must be installed using the steps above where the default value of the `deploymentMode` property is not overridden and set to `PRIMARY`. Additional Cluster Agents must set `deploymentMode` to `NAMESPACED`.

To install additional Cluster Agents:

1. Create a new values file, called `values-ca2.yaml` as the example, that uses the same `controllerInfo` properties as the first Cluster Agent.
   Set `deploymentMode` to `NAMESPACED`.
   Add additional properties, such as `nsToMonitorRegex` and `podFilter`, to set the monitoring scope for this Cluster Agent.

   values-ca2.yaml

   ```
   deploymentMode: NAMESPACED

   # AppDynamics controller info
   controllerInfo:
     url: https://<controller-url>:443
     account: <appdynamics-controller-account>
     username: <appdynamics-controller-username>
     password: <appdynamics-controller-password>
     accessKey: <appdynamics-controller-access-key>

   # Cluster agent config
   clusterAgent:
     nsToMonitorRegex: stage.*

   podFilter:
     allowlistedLabels:
       - label1: value1
       - label2: value2
     blocklistedLabels: []
     allowlistedNames: []
     blocklistedNames: []
   ```

2. Create a namespace distinct from the previous namespace used for the first installation:

   ```
   kubectl create ns appdynamics-ca2
   ```

3. Install the additional Cluster Agent:

   ```
   helm install -f ./values-ca2.yaml "<my-2nd-cluster-agent-helm-release>" appdynamics-charts/cluster-agent --namespace=appdynamics-ca2
   ```

You can enable auto-instrumentation only in the first Cluster Agent using `PRIMARY` mode. The Helm Chart generates an error if auto-instrumentation was enabled by additional Cluster Agents using that `Namespaced` mode.

Cluster Agent Helm Chart Configuration Examples

These examples display various configurations for the Cluster Agent Helm chart:

Use the Cluster Agent Helm Chart to Enable Custom SSL

user-values.yaml

```
controllerInfo:
  url: https://<controller-url>:443
  account: <appdynamics-controller-account>
  username: <appdynamics-controller-username>
  password: <appdynamics-controller-password>
  accessKey: <appdynamics-controller-access-key>

  #=====
  customSSLCert: "<base64 of PEM formatted cert>"
  #=====

agentServiceAccount: appdynamics-cluster-agent-ssl     # Can be any valid name
operatorServiceAccount: appdynamics-operator-ssl       # Can be any valid name
```

Use the Cluster Agent Helm Chart to Enable the Proxy Controller

Without authentication:

user-values.yaml

```
controllerInfo:
  url: https://<controller-url>:443
  account: <appdynamics-controller-account>
  username: <appdynamics-controller-username>
  password: <appdynamics-controller-password>
  accessKey: <appdynamics-controller-access-key>

  #=====
  proxyUrl: http://proxy-url.appd-controller.com
  #=====

agentServiceAccount: appdynamics-cluster-agent-ssl     # Can be any valid name
operatorServiceAccount: appdynamics-operator-ssl       # Can be any valid name
```

With authentication:

user-values.yaml

```
controllerInfo:
  url: https://<controller-url>:443
  account: <appdynamics-controller-account>
  username: <appdynamics-controller-username>
  password: <appdynamics-controller-password>
  accessKey: <appdynamics-controller-access-key>

  #=====
  authenticateProxy: true
  proxyUrl: http://proxy-url.appd-controller.com
  proxyUser: hello
  proxyPassword: world
  #=====

agentServiceAccount: appdynamics-cluster-agent-ssl     # Can be any valid name
operatorServiceAccount: appdynamics-operator-ssl       # Can be any valid name
```

Use the Cluster Agent Helm Chart to add node`Selector` and `tolerations`

user-values.yaml

```
agentPod:
  nodeSelector:
    nodeLabelKey: nodeLabelValue
  tolerations:
    - effect: NoExecute
      operator: Equal
      key: key1
      value: val1
      tolerationSeconds: 11

operatorPod:
  nodeSelector:
    nodeLabelKey: nodeLabelValue
    anotherNodeLabel: anotherNodeLabel
  tolerations:
    - operator: Exists
      key: key1
```

Best Practices for Sensitive Data

We recommend using multiple `values.yaml` files to separate sensitive data in separate `values.yaml` files. Examples of these values are:

- `controllerInfo.password`
- `controllerInfo.accessKey`
- `controllerInfo.customSSLCert`
- `controllerInfo.proxyPassword`

Each `values` file follows the structure of the default `values.yaml` enabling you to easily share files with non-sensitive configuration properties yet keep sensitive values safe.

Default `user-values.yaml` File Example

user-values.yaml

```
deploymentMode: PRIMARY

imageInfo:
  agentImage: dtr.corp.appdynamics.com/sim/cluster-agent
  agentTag: latest
  operatorImage: docker.io/appdynamics/cluster-agent-operator
  operatorTag: latest
  imagePullPolicy: Always

controllerInfo:
  url: https://<controller-url>:443
  account: <appdynamics-controller-account>
  username: <appdynamics-controller-username>
  password: <appdynamics-controller-password>
  accessKey: <appdynamics-controller-access-key>

agentServiceAccount: appdynamics-cluster-agent-ssl     # Can be any valid name
operatorServiceAccount: appdynamics-operator-ssl       # Can be any valid name
```

user-values-sensitive.yaml

```
controllerInfo:
  password: welcome
  accessKey: abc-def-ghi-1516
```

When installing the Helm Chart, use multiple `-f` parameters to reference the files:

```
helm install -f ./user-values.yaml -f ./user-values-sensitive.yaml "<my-cluster-agent-helm-release>"
appdynamics-charts/cluster-agent --namespace ca-appdynamics
```

Validate the Cluster Agent Installation

This page describes how to validate a Cluster Agent installation using the Kubernetes CLI or Helm chart.

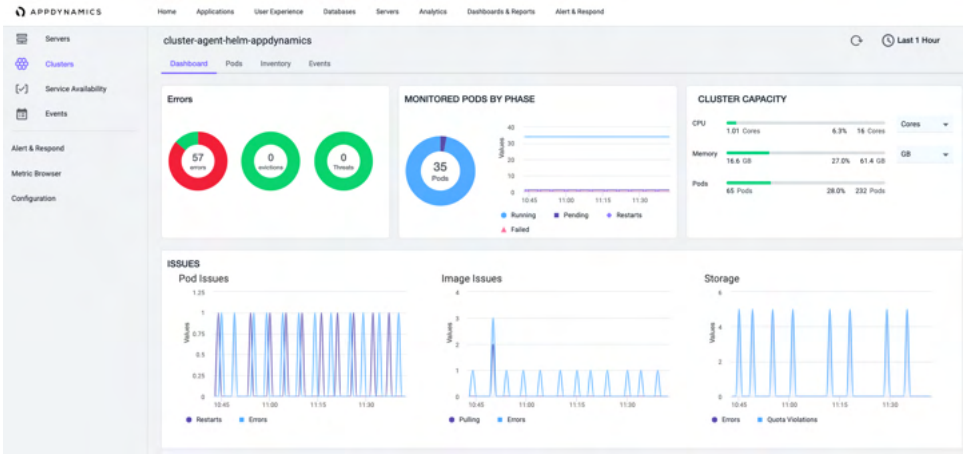
Cluster Agent Validation

1. Verify that the Cluster Agent and Cluster Agent Operator pods are running by reviewing the status:

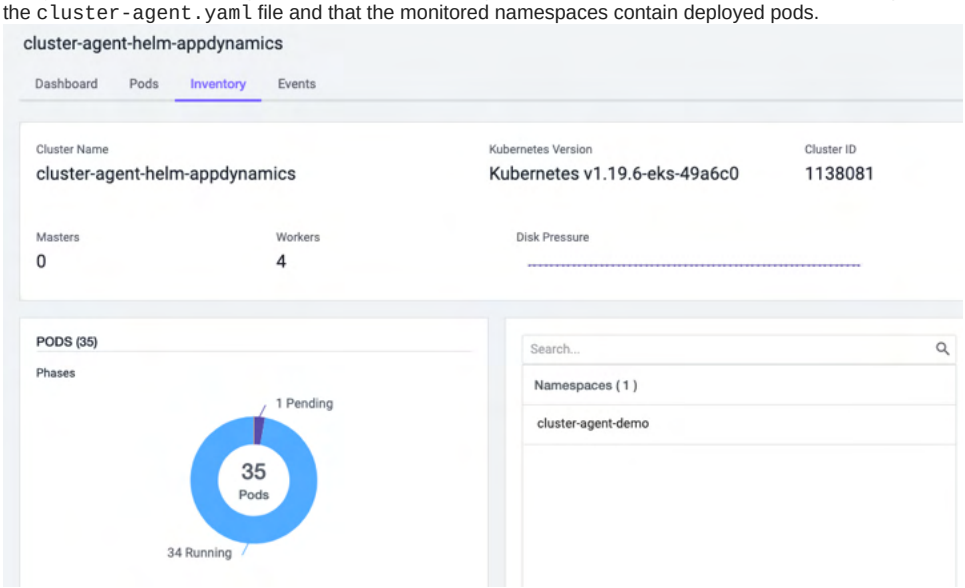
```
kubectl -n appdynamics get pods
NAME                                                READY   STATUS    RESTARTS   AGE
appdynamics-operator-7958f676d9-gvdcw               1/1     Running   0          4d21h
k8s-cluster-agent-5967c9b7b9-rkwzt                  1/1     Running   0          4d21h
```

2. Verify that the Cluster Agent is reporting to the Controller. If the Cluster Agent successfully registers with the Controller based on the credentials you entered in the `cluster-agent.yaml` file, and the `cluster-agent-secret` you created in the `appdynamics` namespace (which contains the Controller Account Access Key), then the Cluster Agent displays in the Controller UI under **Servers/Cluster** using the `appName` from `cluster-agent.yaml`.

   > It may take 5-10 minutes for the data to populate.



3. Verify the monitored namespaces. Under the **Inventory** dashboard, check that the monitored namespaces match the namespace configuration in the `cluster-agent.yaml` file and that the monitored namespaces contain deployed pods.



   If necessary, update the namespace configuration (`nsToMonitor`, `nsToMonitorRegex`) in `cluster-agent.yaml` and re-create the Cluster Agent.

```
kubectl -n appdynamics delete -f cluster-agent.yaml
kubectl -n appdynamics create -f cluster-agent.yaml
```

If you are still experiencing problems with the installation, see Troubleshoot the Cluster Agent.

Troubleshoot the Cluster Agent

This page describes steps to troubleshoot the Cluster Agent install. See Install the Cluster Agent and Validate the Cluster Agent Installation.

Troubleshoot a Cluster Agent Not Reporting to the Controller

If after installing the Cluster Agent the Cluster Dashboard does not appear in the Controller, it could be the result of a connectivity issue with the Controller.

1. Verify that a Server Visibility license is available. The Cluster Agent requires an available Server Visibility license to register successfully. See Cluster Agent Requirements and Supported Environments. From the Controller UI, check that a license is available under **Administration /License/Account Usage**.
2. Review the Cluster Agent events. If the Cluster Agent or Cluster Agent Operator fails to start, then review the events in the `appdynamics` na mespace:

   ```
   kubectl -n appdynamics get events

   # to sort by most recent events:
   kubectl -n appdynamics get events --sort-by='.lastTimestamp'
   ```

   You can review the Cluster Agent pod specification for additional events:

   ```
   kubectl -n appdynamics get pod <cluster-agent-pod> -o yaml
   ```

3. Review the Cluster Agent logs for errors regarding Controller communication. Open a command-line prompt and enter:

   ```
   kubectl -n appdynamics logs <cluster-agent-pod-name>
   ```

4. Verify the Cluster Agent configuration. The Cluster Agent checks for configuration changes once a minute. To verify that your configurations have been applied and that the Cluster Agent is using the new values, open a command-line prompt and enter:

   ```
   kubectl -n appdynamics describe cm cluster-agent-mon cluster-agent-log cluster-agent-config
   ```

5. Verify that the latest Cluster Agent is installed. If you are upgrading the Cluster Agent from a previous version, then the previous Operator YAML or image may not be compatible. You must reinstall the Cluster Agent Operator and Cluster Agent using the procedure described in Up grade the Cluster Agent.

Troubleshoot a Cluster Agent Not Reporting Metrics

If the Cluster Agent does not report metrics for certain containers, pods, or nodes, it may be due to a problem with the Kubernetes Metrics Server. If metrics are not reported by the Metrics Server, then the Cluster Agent is unable to report them.

To verify that the Metrics Server is sending metrics, enter this command from your cluster's primary node:

```
$ kubectl get --raw /apis/metrics.k8s.io/v1beta1/pods
```

If the output of the command does not show metrics for the container, there may be a problem with the Metrics Server. This example shows output from the Metrics Server:

```
{
   "kind":"PodMetricsList",
   "apiVersion":"metrics.k8s.io/v1beta1",
   "metadata":{
      "selfLink":"/apis/metrics.k8s.io/v1beta1/pods"
   },
   "items":[
      {
         "metadata":{
            "name":"replicaset-test-cjnsc",
            "namespace":"test-qe",
            "selfLink":"/apis/metrics.k8s.io/v1beta1/namespaces/test-qe/pods/replicaset-test-cjnsc",
            "creationTimestamp":"2019-09-23T10:24:46Z"
         },
         "timestamp":"2019-09-23T10:23:38Z",
         "window":"30s",
         "containers":[
            {
               "name":"appagent",
               "usage":{
                  "cpu":"1667384n",
                  "memory":"258672Ki"
               }
            }
         ]
      }
   ]
}
```


As the Metric Server collects metrics from nodes, pods, and containers, it logs all issues. To retrieve and view logs for the Metric Server, enter:

```
$ kubectl logs <metric-server pod name>  -n <namespace for metric-server(default value is: "kube-system")> --
tail <number of required lines of logs>
```

For example:

```
$ kubectl logs metrics-server-6764b987d-mtn7g -n kube-system --tail 20
```

The Metric Server logs may reveal why it could not collect metrics. For example:

```
E0920 11:44:54.204075       1 reststorage.go:147] unable to fetch pod metrics for pod test-qe/replicaset-
test-9k7rl: no metrics known for pod
E0920 11:44:54.204080       1 reststorage.go:147] unable to fetch pod metrics for pod test/replicaset1-458-
g9n2d: no metrics known for pod
E0920 11:44:54.204089       1 reststorage.go:147] unable to fetch pod metrics for pod kube-system/kube-proxy-
t54rc: no metrics known for pod
E0920 11:45:19.188033       1 manager.go:111] unable to fully collect metrics: unable to fully scrape
metrics from source kubelet_summary:ip-111.111.111.111: unable to fetch metrics from Kubelet ip-
111.111.111.111 (111.111.111.111): Get https://111.111.111.111:2222/stats/summary/: dial tcp 111.111.111.111:
2222: i/o timeout
```

Cluster Agent Restarts

If the Cluster Agent restarts, you can verify that a restart occurred from the pod details. To retrieve the pod details, enter:

```
kubectl get pods -n appdynamics
```

Sample output:

```
NAME                                          READY   STATUS    RESTARTS   AGE
appdynamics-operator-6fff76b466-qtx57         1/1     Running   0          4h18m
k8s-cluster-agent-perf-jg-6fc498d557-q7zst    1/1     Running   1          83m
```

If the Cluster Agent unexpectedly restarts, the `RESTARTS` count value will be greater than zero. You will have to explicitly reset both namespaces and the logs.

> AppDynamics strongly recommends that you do not overwrite the default `stdoutLogging: true` property value in the `cluster-agent.yaml` file. If you set this property to `false`, the `kubectl logs` command does not return logs.

Cluster Agent logs persist even if the Cluster Agent is restarted by Kubernetes. To view the Cluster Agent logs for the Cluster Agent pod (that restarted), enter:

```
kubectl -n appdynamics logs --previous ${CLUSTER_AGENT_POD_NAME}
```

If the Cluster Agent pod has restarted, the monitored namespaces (that you configured through the UI) are not preserved. If you configured namespaces through the UI, you should add the same namespaces to your `cluster-agent.yaml` file under `nsToMonitor`, and then apply the configuration. As a result, the Cluster Agent pod will retain the monitored namespaces when it restarts.

If you did not add namespaces to the `cluster-agent.yaml` file, you can reconfigure your monitored namespaces:

1. Go to **Appdynamics Agents > Cluster Agents >** {CLUSTER_AGENT} **> Configure**.
2. Add the namespaces to monitor.

See Add or Remove Namespaces.

APM Correlation on OpenShift 4.x

Container Runtime Interface using OCI (Open Container Initiative) compatible runtimes (CRI-O), is the default container runtime on Red Hat Openshift 4.x. If you use APM Agents with OpenShift 4.x, you must update the `UNIQUE_HOST_ID` to support the syntax required for CRI-O containers. This setting applies to both new and existing application containers. If you are running App Agents, then you must modify the App Agent `YAML` file.

To run App Agents with APM correlation on Openshift 4.x:

1. Open your App Agent `YAML` file.
2. Locate the `spec: > args:` section within the file.
3. Update the `UNIQUE_HOST_ID` argument in the `containers spec` using this example as a guide:

```
spec:
      containers:
      - name: client-api
        command: ["/bin/sh"]
        args: ["-c", "UNIQUE_HOST_ID=$(sed -rn '1s#.*/##; 1s/(.{12}).*/\\1/p' /proc/self/cgroup) &&
java -Dappdynamics.agent.uniqueHostId=$UNIQUE_HOST_ID $JAVA_OPTS -jar /java-services.jar"]
        envFrom:
        - configMapRef:
            name: agent-config
```

If APM Correlation is working correctly, when you click the **Pod Details** link, the link opens the APM Node Dashboard for that node.

Cluster Agents or Pods are not Visible in the Controller

If Agents or pods are not visible in the Controller, or if Agents or pods are not registered and reporting, review the `sim.cluster.agent.limit` and `sim.cluster.pod.limit` descriptions in Controller Settings for the Cluster Agent.

Cluster Agent Pods are not Created When Security Policy is Enabled

If you have Pod Security Policies applied in the cluster, add the following to the `cluster-agent-operator.yaml` file:

```
securityContext:
  runAsUser: 1000
```

This YAML file example displays the security context:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: appdynamics-operator
  namespace: appdynamics
  .
  .
spec:
  .
  .
  .
  template:
    .
    .
    securityContext:
      runAsUser: 1000
```

For Agent 20.4 and 20.5, add the `runAsUser: 1000` field in the `cluster-agent.yaml` file.

Cluster Agent Configuration Files

This page provides a reference for these internal configuration files used by the Cluster Agent:

- `agent-monitoring.yml`
- `bootstrap-config.yml`
- `logger-config.yml`

> AppDynamics recommends that you do not change the default values unless asked to do so by AppDynamics Support. Values supplied by the AppDynamics Operator always take precedence over the internal configuration files.

Agent Monitoring Configuration Reference

The `agent-monitoring.yml` file contains configuration information that can be modified by the Cluster Agent. The table lists the fields expected by the `agent-monitoring.yml` file. Use these fields to configure pods and containers monitored by the Cluster Agent:

Field name	Description	Default
`blocklisted-label`	(Optional) Field used to specify a Kubernetes label (a `key-value` pair). If this label exists, then the container, or pod, is not monitored. In the example, any pods with the label `appdynamics.exclude` set to `true` are blocklisted.	`appdynamics.exclude: true`
`blocklisted-names`	(Optional) Field used to specify which pods and containers are not monitored. In the example, the pod `ignored-pod` with associated containers `container3` and `container4` are not monitored.	Empty
`cluster-metric-collection-interval-seconds`	How often the Agent collects metrics in seconds Do not set this value to less than 30 seconds	`60`
`container-filter`	(Optional) Blocklist or allowlist pods and containers, based on the container names	Empty
`container-registration-batch-size`	Maximum number of containers per batch in one registration cycle	`25`
`container-registration-max-parallel-requests`	Maximum number of batches in one registration cycle	`3`
`metadata-collection-interval-seconds`	How often metadata is collected for containers and pods	`60`
`metric-collection-interval-seconds`	Sampling interval at which metrics should be collected periodically Do not set this value to less than 15 seconds > The Kubernetes Metrics Server's sampling interval must be the same as this configuration value. Any change to this configuration requires a re-deployment of the Kubernetes metrics-server add-on.	- Default metric collection sampling interval is `30` seconds - Default collection interval in `kubelet` is `15` seconds
`metric-upload-retry-count`	Number of times metric upload action to be attempted, if unsuccessful for the first time	`3`
`metric-upload-retry-interval-milliseconds`	Interval between consecutive metric upload retries, in milliseconds	`5`
`pod-registration-batch-size`	Maximum number of pods per batch in one registration cycle	`30`
`allowlisted-names`	(Optional) Field used to specify pods and containers to monitor. In the example, the pod `monitored-pod` with its containers `container1` and `container2` are monitored.	Empty

`agent-monitoring.yml` example:

```
metric-collection-interval-seconds: 30
container-filter:
  blocklisted-label:
    appdynamics.exclude:
      true
#  allowlisted-names:
#    pod-name1:
#      container-name1
#      container-name2
#    pod-name2:
#      container-name1
#      container-name2
#    pod-name3:
#      container-name1
#      container-name2
#  blocklisted-names:
#    pod-name:
#      container-name1
#      container-name2
#    pod-name2:
#      container-name1
#      container-name2
```

Bootstrap Configuration Reference

The `bootstrap-config.yml` file contains configuration information read by the Cluster Agent during start up. The table lists the fields expected by the `bootstrap-config.yml` file:

Field name	Description	Default
`account`	Name of the account on the Controller	Empty
`account-access-key`	Account access key	Empty
`cluster-name`	Unique name assigned to the monitoring cluster	Empty
`container-registration-interval-seconds`	How often the Cluster Agent sends container information to the Controller. The interval at which the Cluster Agent checks for running containers. Modify the default value only if you want to discover running containers more frequently; use the default value for most environments.	`120`
`controller-host`	Controller host name	Empty
`controller-port`	Controller port	Empty
`controller-ssl-enabled`	Denotes if the Controller is SSL-enabled	`false`
`event-upload-interval-seconds`	How often events are uploaded to the Controller, in seconds	`10`
`http-client-timeout-seconds`	Number of seconds after which the server call is terminated if no response is received from the Controller	`30`
`log-output-directory`	Directory to which the logs should be written	`logs`
`monitored-namespaces`	Namespaces to be monitored in the cluster; comma-separated strings	`default`

`bootstrap-config.yml` example:

```
account: <account-name>
account-access-key:<account-access-key>
controller-host: localhost
controller-port: 8090
cluster-name: <cluster-name>
monitored-namespaces: default
event-upload-interval-seconds: 10
container-registration-interval-seconds: 120
http-client-timeout-seconds: 30
log-output-directory: logs
# SSL configuration
controller-ssl-enabled: false
```

Logging Configuration Reference

By default, the AppDynamics Cluster Agent writes log files to the *<cluster_agent_home>*/logs/*<node_name>* directory.

You can control the logging level for the Cluster Agent by changing the value of the `log-level` parameter. For maximum debugging information, set the `logger-config.yml` logging level to `DEBUG`.

You can modify the `logger-config.yml` file and set values for these attributes:

Attribute	Description	Default	Type
`log-level`	Number of log details: `INFO`, `WARNING`, `DEBUG` or `TRACE`	`INFO`	String
`max-filesize-mb`	Maximum file size of the log in MB	`5`	Integer
`max-backups`	Maximum number of backups the log saves. When the maximum number of backups is reached, the oldest log file after the initial log file is deleted.	`3`	Integer
`write-to-stdout`	Write logging information to `stdout`. Options: `true`, `false`.	`true`	String

`logger-config.yml` example:

```
log-level: INFO
max-filesize-mb: 5
max-backups: 3
write-to-stdout: true
```

Configure the Cluster Agent

This page describes the contents of the Cluster Agent bundle downloaded from the Download portal, and how to perform common configuration tasks:

- Configure Proxy Support
- Configure the Cluster Agent to use SSL for on-premises Controllers
- Configure a Pull Secret

See Cluster Agent YAML File Configuration Reference for configuration option details.

> This page contains links to Kubernetes documentation. AppDynamics makes no representation as to the accuracy of Kubernetes documentation because Kubernetes controls its own documentation.

Directory Structure of the Cluster Agent Bundle

An unzipped Cluster Agent bundle contains this directory structure:

Cluster Agent Bundle Files

This table describes the Cluster Agent directory files:

File Name	Description
`cluster-agent.yaml`	File used to configure and deploy the Cluster Agent. • The `cluster-agent.yaml` file provides Controller details and starts the Cluster Agent. • Where values are specified in the AppDynamics Operator configuration, these values always take precedence over any internal configuration file.

`cluster-agent-operator.yaml`
 	
`cluster-agent-operator-1.14-or-less.yaml`
	Files used to deploy the Cluster Agent Operator. These files set the default values for Kubernetes, Amazon EKS, and AKS, including a minimal set of RBAC permissions.

`cluster-agent-operator-openshift.yaml`
 	
`cluster-agent-operator-openshift-1.14-or-less.yaml`
	Files used to deploy the Cluster Agent on Red Hat OpenShift. These files set the default values for Red Hat OpenShift, including a minimal set of RBAC permissions.

`docker`	Docker directory contains all files required to create the Cluster Agent image.
`Dockerfile`	`dockerfile` used to create the Alpine-based Cluster Agent image.
`Dockerfile-rhel`	`dockerfile` used to create the Rhel-based Cluster Agent image.
`LICENSE`	Latest EULA file attached with the Cluster Agent image.
`cluster-agent.zip`	Zip archive containing the Cluster Agent binaries and configuration files.
`helm-charts`	Folder used to build the charts for deploying the Cluster Agent using Helm in Kubernetes.

`README-rhel.md`
 	
`README-alpine.md`
	Contains instructions on how to start the Cluster Agent using your preferred operating system.

`start-appdynamics`	Script used to run the Cluster Agent within Docker.

Configure Proxy Support

1. Locate and edit the `cluster-agent.yaml` file.
2. Add a `proxyUrl` parameter to the `cluster-agent.yaml` file:

```
proxyUrl: <protocol>://<host>:<port>
```

3. (Optional) If the proxy server requires authentication:
    a. Add a `proxyUser`:

```
proxyUser: <user>
```

b. Create a `secret` with a `proxy-password`:

```
kubectl -n appdynamics create secret generic cluster-agent-proxy-secret --from-literal=proxy-password='<password>'
```

4. (Optional) If you are using SSL only for your proxy:
   a. Create a `secret` from a `.pem` certificate file (the certificate file must be named `proxy-ssl.pem`):

```
kubectl -n appdynamics create secret generic ssl-cert --from-file=proxy-ssl.pem
```

   b. Set a secret filename in the `cluster-agent.yaml` file:

```
customSSLSecret: “ssl-cert”
```

To use SSL with your proxy and your Controller, see Proxy and On-Premises Certificates Combined.

Configure the Cluster Agent to Use SSL for On-Premises Controllers

Cluster Agent SSL is automatically handled for SaaS Controllers.

Controllers with Public and Self-Signed Certificates

To configure SSL with a public or self-signed certificate, use `kubectl` to generate a secret. Enter this `kubectl` command, and include the path to your public or self-signed certificate:

```
kubectl -n appdynamics create secret generic ssl-cert --from-file=<path-to-your-self-signed-certs>/custom-ssl.pem
```

The certificate file must be named: `custom-ssl.pem`.

After your secret is created, you must add the `customSSLSecret` property with the secret name specified in the previous step to the `cluster-agent.yaml` file:

```
customSSLSecret: “ssl-cert”
```

Proxy and On-Premises Certificates Combined

If you have two different SSL certificates (one for the proxy server, and a different one for the on-premises Controller), then you can encapsulate both of them into a single secret:

```
kubectl -n appdynamics create secret generic ssl-cert --from-file=proxy-ssl.pem --from-file=<path-to-your-self-signed-certs>/custom-ssl.pem
```

The Cluster Agent pulls each certificate from the secret identified in the `customSSLSecret` attribute and uses it appropriately.

This example shows a `cluster-agent.yaml` file with the `customSSLSecret` attribute defined:

```
apiVersion: appdynamics.com/v1alpha1
kind: Clusteragent
metadata:
  name: k8s-cluster-agent-manual
  namespace: appdynamics
spec:
  # init agent configuration
  appName: "test-k8s-cluster-agent"
  controllerUrl: "https://<controller-url>:443" # always schema and port
  account: "<account-name>" # account
  # agent related properties
  # custom SSL secret name
  customSSLSecret: "ssl-cert"
  # logging properties
  logLevel: INFO
  logFileSizeMb: 7
  logFileBackups: 6
  # docker image info
  image: "<image-url>"
```

Configure a Pull Secret

If the Cluster Agent requires a secret to pull images from a container registry, use the Kubernetes API to create the secret and reference it in `cluster-agent.yaml`.

Set the `imagePullSecret` property in `cluster-agent.yaml` to the name of the secret created above (`myregcred`):

```
kind: Clusteragent
metadata:
  name: k8s-cluster-agent
  namespace: appdynamics
spec:
  appName: "mycluster"
  controllerUrl: "http://<appdynamics-controller-host>:8080"
  account: "<account-name>"
  image: "<your-docker-registry>/appdynamics/cluster-agent:tag"
  serviceAccountName: appdynamics-cluster-agent
  imagePullSecret: "myregcred"
```

Cluster Agent YAML File Configuration Reference

To configure the Cluster Agent, use the `cluster-agent.yaml` file included with the download package as a template. You can modify these parameters:

Parameter	Description	Example	Default	Dynamically Configurable?	Type	Required?
`account`	AppDynamics account name.	`admin`	N/A	No	String	Required
`appName`	Name of the cluster; displays in the Controller UI as your cluster name.	`k8s-cluster`	N/A	No	String	Required

`controllerUrl`	Full AppDynamics Controller URL, including protocol and port.	`HTTP: http://appd-controller.com:8090/`
		
		`HTTPS: https://appd-controller.com:443`
			N/A	No	String	Required

`customSSLSecret`	Provides the self-signed or public certificates to the Cluster Agent.	`"ssl-cert"`	N/A	No	String	Optional
`eventUploadInterval`	How often Kubernetes warning and state-change events are uploaded to the Controller in seconds. See Monitor Kubernetes Events.	`10`	10	No	Integer	Optional

`httpClientTimeout`	If no response is received from the Controller, number of seconds after which the server call is terminated.	`30`	30	No	Integer	Optional
`image`	Cluster Agent image.	`your-docker-registry/appdynamics/cluster-agent:latest`	N/A	No	String	Required
`imagePullSecret`	Credential file used to authenticate when pulling images from your private Docker registry or repository. Based on your Docker registry configuration, you may need to create a secret file for the AppDynamics Operator to use when pulling the image for the Cluster Agent. See Create a Secret by providing credentials on the command line.	`regcred`	N/A	No	String	Optional
`logFileSizeMb`	Maximum file size of the log in MB.	`5`	5	Yes	Integer	Optional
`logFileBackups`	Maximum number of backups saved in the log. When the maximum number of backups is reached, the oldest log file after the initial log file is deleted.	`3`	3	Yes	Integer	Optional
`logLevel`	Number of log details. `INFO`, `WARNING`, `DEBUG`, or `TRACE`.	`"INFO"`	`INFO`	Yes	String	Optional
`maxPodLogsTailLinesCount`	Number of lines to be tailed while collecting logs. To use this parameter, enable the log capturing feature. See Enable Log Collection for Failing Pods.	`500`	`500`	Yes	Integer	Optional

`nodeSelector`	The Cluster Agent pod runs on the node that includes the specified `key-value` pair within its labels property. See nodeSelector.	`nodeSelector:`
		
		`kubernetes.io/e2e-az-name: az1`
			N/A	No	map [string] [string]	Optional

`nsToMonitor`	Namespaces to be monitored in the cluster.	`nsToMonitor:`
		
		`- "default"`
		`- "appdynamics"`
			`default`	No	String List (Sequence)	Optional

`nsToMonitorRegex`	The regular expression for selecting the required namespaces to be monitored in the cluster. This parameter supersedes `nsToMonitor`. If you do not specify a value for this parameter, then the Cluster Agent uses the default value of `nsToMonitor`. ⓘ This parameter is supported in Cluster Agent >= 20.9, and Controller >= 20.10. See Edit Namespaces.	nsToMonitoRegex: .*	N/A	Yes	Regular expression	Optional
`nsToExcludeRegex`	The regular expression for the namespaces that must be excluded from the selected namespaces that matche the regular expression mentioned for `nsToMonitorRegex`. ⓘ This parameter is supported in Cluster Agent >= 20.9, and Controller >= 20.10. This parameter can be used only if you have specified a value for the `nsToMonitorRegex` parameter.	nsToExcludeRegex: ns.*	N/A	Yes	Regular expression	Optional

`podFilter`	Blocklist or allowlist pods based on:
	
	• Regular expressions for pod names
	• Pod labels
	
	Blocklisting or allowlisting by name takes preference over blocklisting or allowlisting by labels. For example, if you have the `podFilter` as :
	
	`podFilter:`
	
	`blocklistedLabels:`
	`- release: v1`
	
	`allowlistedNames:`
	`- ^podname`
	
	This blocks all the pods which have the label '`release=v1`' except for the ones which have the names starting with '`podname`'.
	
	• When a pod is listed as allowed by name and blocked by name, it will be allowlisted.
	• When a pod is listed as allowed by a label and blocked by a label, it will be allowlisted.
		`podFilter:`
		
		`blocklistedLabels:`
		`- label1: value1`
		
		`allowlistedLabels:`
		`- label1: value1`
		`- label2: value2`
		
		`allowlistedNames:`
		`- name1`
		
		`blocklistedNames:`
		`- name2`
			N/A	Yes	String	Optional

`proxyUrl`	Publicly accessible host name of the proxy.	`https://myproxy.example.com:8080`	N/A	No	String	Optional
`proxyUser`	Username associated with the basic authentication credentials.	`"user1"`	N/A	No	String	Optional

`resources`	Requests and limits of CPU and memory resources for the Cluster Agent.	`resources:`
		`  limits:`
		`  cpu: 300m`
		`  memory: "200Mi"`
		`  requests:`
		`  cpu: 200m`
		`  memory: "100Mi"`
			• CPU
			  • `request: 750m`
			  • `limit: 1250m`
			• Memory
			  • `request: 150Mi`
			
			`limit: 300Mi`
				Yes	Array	Optional

`stdoutLogging`	By default, the Cluster Agent writes to a log file in the `logs` directory. Additionally, the `stdoutLogging` parameter is provided to send logs to the container `stdout`.	`"true", "false"`	`true`	Yes	String	Optional

`tolerations`	An array of tolerations required for the pod. See Taint and Tolerations.	`tolerations:`
		
		`- effect: NoSchedule`
		
		`key: type value: test`
		
		`- effect: NoExecute`
		
		`key: node.kubernetes.io/not-ready operator: Exists tolerationSeconds: 600`
			N/A	No	Array	Optional

For specific auto-instrumentation configurations, see Auto-Instrument Applications with the Cluster Agent. Also the `.yaml` file includes the permissions for auto-instrumentation, which is enabled by default. If you do not want to use auto-instrumentation, you can remove the following text from the `.yaml` file:

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
 name: appdynamics-cluster-agent-instrumentation
subjects:
 - kind: ServiceAccount
 name: appdynamics-cluster-agent
 namespace: appdynamics
roleRef:
 kind: ClusterRole
 name: appdynamics-cluster-agent-instrumentation
 apiGroup: rbac.authorization.k8s.io
```

Cluster Agent File Example

This example shows a `cluster-agent.yaml` configuration file:

```
apiVersion: appdynamics.com/v1alpha1
kind: Clusteragent
metadata:
  name: k8s-cluster-agent
  namespace: appdynamics
spec:
  appName: "<app-name>"
  controllerUrl: "<protocol>://<appdynamics-controller-host>:8080"
  account: "<account-name>"
  # docker image info
  image: "<your-docker-registry>/appdynamics/cluster-agent:tag"
  nsToMonitor:
    - "default"
  eventUploadInterval: 10
  containerRegistrationInterval: 120
  httpClientTimeout: 30
  customSSLSecret: "<secret-name>"
  proxyUrl: "<protocol>://<domain>:<port>"
  proxyUser: "<proxy-user>"
  metricsSyncInterval: 30
  clusterMetricsSyncInterval: 60
  metadataSyncInterval: 60
  containerBatchSize: 25
  containerParallelRequestLimit: 3
  podBatchSize: 30
  metricUploadRetryCount: 3
  metricUploadRetryIntervalMilliSeconds: 5
  podFilter:
    # blocklistedLabels:
    #   - label1: value1
    # allowlistedLabels:
    #   - label1: value1
    #   - label2: value2
    # allowlistedNames:
    #   - name1
    # blocklistedNames:
    #   - name2
  logLevel: "INFO"
  logFileSizeMb: 5
  logFileBackups: 3
  stdoutLogging: "true"
  resources:
        limits:
                cpu: 300m
                memory: "200Mi"
        requests:
                cpu: 200m
                memory: "100Mi"
```

Install Infrastructure Visibility with the Cluster Agent Operator

This page describes how to install the Machine Agent and Network Agents in a Kubernetes cluster where the Cluster Agent Operator is installed.

The Cluster Agent Operator provides a custom resource definition called `InfraViz`. You can use `InfraViz` to simplify deploying the Machine and Network Agents as a `daemonset` in a Kubernetes cluster. Additionally, you can deploy these agents by creating a `daemonset YAML` which does not require the Cluster Agent Operator. For more information, see these examples.

To deploy the Analytics Agent as a `daemonset` in a Kubernetes cluster, see Install Agent-Side Components in Kubernetes.

Requirements

Before you begin, verify that you have:

- Installed kubectl >= 1.11.3
- Cluster Agent >= 21.3.1
- Met these requirements: Cluster Agent Requirements and Supported Environments.
- If Server Visibility is required, sufficient Server Visibility licenses based on the number of worker nodes in your cluster.
- Permissions to view servers in the AppDynamics Controller.

Installation Procedure

1. Install the Cluster Agent Operator. From this Alpine Linux example:

    a. Download the Cluster Agent bundle.
    b. Unzip the Cluster Agent bundle.
    c. Deploy the Cluster Agent Operator using the CLI specifying the correct Kubernetes and OpenShift version (if applicable):

    ```
    unzip appdynamics-cluster-agent-alpine-linux-<version>.zip
    kubectl create namespace appdynamics
    ```

2. Create a Cluster Agent secret using the Machine Agent access key to connect to the Controller. If a `cluster-agent-secret` does not exist, you must create one, see Install the Cluster Agent with the Kubernetes CLI (DOC-6096).

    ```
    kubectl -n appdynamics create secret generic cluster-agent-secret --from-literal=controller-key=<access-key>
    ```

3. Update the `infraviz.yaml` file to set the `controllerUrl`, `account`, and `globalAccount` values based on the information from the Controller's **License** page.
    To enable Server Visibility, set `enableServerViz` to `"true"` (shown in the `infraviz.yaml` configuration example).
    To deploy a Machine Agent without Server Visibility enabled, set `enableServerViz` to `"false"`.

infraviz.yaml Configuration File with Server Visibility Enabled

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: appdynamics-infraviz
  namespace: appdynamics
---
apiVersion: appdynamics.com/v1alpha1
kind: InfraViz
metadata:
  name: appd-infraviz
  namespace: appdynamics
spec:
  controllerUrl: "https://mycontroller.saas.appdynamics.com"
  image: "docker.io/appdynamics/machine-agent-analytics:latest"
  account: "<your-account-name>"
  globalAccount: "<your-global-account-name>"
  enableServerViz: "true"
  resources:
    limits:
      cpu: 500m
      memory: "1G"
    requests:
      cpu: 200m
      memory: "800M"
```

The `infraviz.yaml` configuration file example deploys a `daemonset` that runs a single pod per node in the cluster. Each pod runs a single container from where the Machine Agent, or Server Visibility Agent runs.

4. To enable the Network Visibility Agent to run in a second container in the same pod, add the `netVizImage` and `netVizPort` keys and values as shown in this configuration file example:

infraviz.yaml Configuration File with Second Container in a Single Pod

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: appdynamics-infraviz
  namespace: appdynamics
---
apiVersion: appdynamics.com/v1alpha1
kind: InfraViz
metadata:
  name: appd-infraviz
  namespace: appdynamics
spec:
  controllerUrl: "https://mycontroller.saas.appdynamics.com"
  image: "docker.io/appdynamics/machine-agent-analytics:latest"
  account: "<your-account-name>"
  globalAccount: "<your-global-account-name>"
  enableServerViz: "true"
  netVizImage: appdynamics/machine-agent-netviz:latest
  netVizPort: 3892
  resources:
    limits:
      cpu: 500m
      memory: "1G"
    requests:
      cpu: 200m
      memory: "800M"
```

5. Use `kubectl` to deploy `infraviz.yaml`

For environments where Kubernetes `PodSecurityPolicies` block certain pod security context configuration, such as privileged pods, you must deploy the `infraviz-pod-security-policy.yaml` before editing the `infraviz.yaml` file.

For environments where OpenShift `SecurityContextConstraints` block certain pod security context configuration, such as privileged pods, you must deploy the `infraviz-security-context-constraint-openshift.yaml` before editing the `infraviz.yaml` file.

6. Confirm that the `appd-infraviz` pod is running, and the Machine Agent, Server Visibility Agent, and Network Agent containers are ready:

```
kubectl -n appdynamics get pods
NAME                                READY   STATUS    RESTARTS   AGE
appd-infraviz-shkhj                 2/2     Running   0          18s
```

7. To verify that the agents are registering with the Controller, review the logs and confirm that the agents display in the **Agents Dashboard** of the Controller Administration UI. In the Controller, if Server Visibility is enabled, the nodes are visible under **Controller > Servers**.

```
kubectl -n appdynamics logs appd-infraviz-shkhj -c appd-infra-agent
...
Started AppDynamics Machine Agent Successfully
```

Mixed Operating System Clusters

The Cluster Agent Operator can deploy Machine Agent `daemonsets` to both Linux and Windows nodes. The value of the `nodeOS` property determines the deployment strategy in mixed operating system (OS) clusters. You can set this property to one of these options:

- `all`
- `linux`
- `windows`

Set `nodeOS: all` when both Linux and Windows `daemonsets` share the same properties that you can configure in a single `infraviz.yaml` file. When `nodeOS` is set to `all` or `windows`, you must set the `imageWin` property to a Windows Machine Agent image.

In this example, a single `infraviz.yaml` file is used to configure the Machine Agents that run on Windows and Linux nodes.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: appdynamics-infraviz
  namespace: appdynamics
---
apiVersion: appdynamics.com/v1alpha1
kind: InfraViz
metadata:
  name: appd-infraviz
  namespace: appdynamics
spec:
  controllerUrl: "https://mycontroller.saas.appdynamics.com"
  image: "docker.io/appdynamics/machine-agent-analytics:latest"
  account: "<your-account-name>"
  globalAccount: "<your-global-account-name>"
  enableServerViz: "true"
  netVizImage: appdynamics/machine-agent-netviz:latest
  netVizPort: 3892
  resources:
    limits:
      cpu: 500m
      memory: "1G"
    requests:
      cpu: 200m
      memory: "800M"
  imageWin: docker.io/appdynamics/machine-agent-analytics:20.6.0-win-ltsc2019
  nodeOS: all
```

Set `nodeOS: windows` or `nodeOS: linux` when the Windows and Linux `daemonsets` properties differ and must be defined in separate `infraviz.yaml` files. Depending on the value of `nodeOS`, the Cluster Agent Operator assigns a `nodeSelector` value to the Machine Agent that determines its placement.
For `nodeOS: windows`, it sets "`kubernetes.io/os`" to `windows`.
For `nodeOS: linux`, it sets `kubernetes.io/os` to `linux`.

To override this default behavior, specify a `nodeSelector` in `infraviz.yaml`. For example:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: appdynamics-infraviz
  namespace: appdynamics
---
apiVersion: appdynamics.com/v1alpha1
kind: InfraViz
metadata:
  name: appd-infraviz
  namespace: appdynamics
spec:
  # ... content removed for brevity
  nodeOS: windows
  imageWin: docker.io/appdynamics/machine-agent-analytics:20.6.0-win-ltsc2019
  nodeSelector:
    kubernetes.io/os: my-windows-node-selector
```

`InfraViz` Configuration Settings

To configure Infrastructure Visibility, you can modify these parameters in the `infraviz.yaml` file included with the download package. After changing the file, delete and re-create the `InfraViz` deployment to ensure the changes are applied.

Parameter	Description	Required/Optional	Default
`account`	AppDynamics account name	Required	N/A
`args`	List of command arguments	Optional	N/A
`controllerUrl`	URL of the AppDynamics Controller	Required	N/A
`enableContainerHostId`	Flag that determines how container names are derived; specify either `pod name` or `container id`.	Optional	`true`
`enableMasters`	By default, only Worker nodes are monitored. When set to `true`, Server Visibility is provided for Master nodes. For managed Kubernetes providers, the flag has no effect because the Master plane is not accessible.	Optional	`false`
`enableServerViz`	Enable Server Visibility	Optional	`true`
`env`	List environment variables	Optional	N/A
`eventServiceUrl`	Event Service Endpoint	Optional	N/A
`globalAccount`	Global account name	Required	N/A
`image`	Retrieves the most recent version of the Machine Agent image.	Optional	`appdynamics/machine-agent-analytics:latest`
`imagePullSecret`	Name of the pull secret image	Optional	N/A
`imageWin`	Machine Agent image for Windows nodes	Required when `nodeOS` is set to `all` or `windows`	N/A
`logLevel`	Level of logging verbosity. Valid options are: `info` or `debug`.	Optional	`info`
`metricsLimit`	Maximum number of metrics that the Machine Agent sends to the Controller.	Optional	N/A
`netVizImage`	Retrieves the most recent version of Network Agent image.	Optional	`appdynamics/machine-agent-netviz:latest`

`netVizPort`	When > 0, the Network Agent is deployed in a sidecar with the Machine Agent. By default, the Network Visibility Agent works with port `3892`.	Optional	`3892`
`nodeSelector`	OS specific label that identifies nodes for scheduling of the `daemonset` pods.	Optional	`all`
`priorityClassName`	Name of the priority class that determines priority when a pod needs to be evicted.	Optional	N/A
`propertyBag`	String with any other Machine Agent parameters	Optional	N/A
`proxyUrl`	URL of the proxy server (`protocol://domain:port`)	Optional	N/A
`proxyUser`	Proxy user credentials (`user@password`)	Optional	N/A
`resources`	Definitions of resources and limits for the Machine Agent	Optional	N/A
`resourcesNetViz`	Set resources for the Network Visibility (NetViz) container	Optional	**Request** • CPU: 100m • Memory: 150Mi **Limit** • CPU: 200m • Memory: 300Mi
`stdoutLogging`	Determines if logs are saved to a file or redirected to the Console.	Optional	`false`
`tolerations`	List of tolerations based on the taints that are associated with nodes.	Optional	N/A
`uniqueHostId`	Unique host ID in AppDynamics. Valid options are: `spec.nodeName`, `status.hostIP`.	Optional	`spec.nodeName`

Build the Cluster Agent Container Image

This page describes how to build a Cluster Agent container image or push a pre-built image to an internal registry.

This procedure is required when you:

- Need to build your own image to satisfy internal image policies, or
- Require that the pre-built Cluster Agent Image from Docker Hub, or the Red Hat Container Registry, is stored in an internal container registry. See Install the Cluster Agent.

Otherwise, you can reference the pre-built Cluster Agent Images on Docker Hub and the Red Hat Container Registry in the `cluster-agent.yaml` or `values.yaml` Helm file. This example uses the pre-built Cluster Agent Image 20.12.1 published to Docker Hub:

```
apiVersion: appdynamics.com/v1alpha1
kind: Clusteragent
metadata:
 name: k8s-cluster-agent
 namespace: appdynamics
spec:
 controllerUrl: "http://<account>.saas.appdynamics.com"
 account: "<account-name>"
 appName: "Cluster1"
 image: docker.io/appdynamics/cluster-agent:20.12.1
```

Build the Custom Cluster Agent Image

The download packages from the AppDynamics Download Portal include files you need to build and deploy the container image for the Cluster Agent.

Depending on the base Image that was used to build the Cluster Agent Image, there are two package types available on the portal:

- Alpine Linux-based base Image
- Rhel Linux 7-based base Image

The directory structure of the download packages for the Alpine and Rhel Linux is:

Build the Custom Cluster Agent Image

You can build the Rhel Linux 7-based Cluster Agent image on Red Hat Linux only. Red Hat Linux is a subscription-based system. You must subscribe to the Red Hat Linux system to use the packages manager such as `microdnf`, which is used in `Dockerfile-rhel`. Perform these tasks before running the docker build command for Rhel:

1. Use Red Hat Linux 7 (Rhel VM Box, Rhel EC2 instance, and so on)
2. To subscribe to the Red Hat Linux system, from a command-prompt enter:

```
$ subscription-manager register --username=<your-username> --password=<your-password>
```

3. To get the pool-ids, enter:

```
$ subscription-manager list --available
```

4. To attach the pool-id from step 3, enter:

```
$ subscription-manager attach --pool=<pool-id>
```

5. To enable repos, enter:

```
$ subscription-manager repos --enable=rhel-7-server-rpms
$ subscription-manager repos --enable=rhel-7-server-extras-rpms
$ subscription-manager repos --enable=rhel-7-server-optional-rpms
```

6. To install Docker on the Rhel machine, enter:

```
$ yum install docker device-mapper-libs device-mapper-event-lib
```

7. To start docker services, enter:

```
$ systemctl start docker.service
$ systemctl enable docker.service
```

8. To build the Rhel based Cluster Agent image using Docker, enter:

```
$ docker build -t docker.io/<organization>/cluster-agent:<Agent-Version> -f Dockerfile-rhel
```

See the Red Hat website under Using the docker command and service for more details.

These steps are also available at the official website of Red Hat under Using the docker command and service.

If you are unable to perform the preceding steps, you can use a pre-built Rhel-based Cluster Agent image mentioned in Use Pre-Built Cluster Agent Image. This pre-built option requires that you have Docker installed and Red Hat credentials. The pre-built image can run on any operating system. To get Red Hat credentials, create an account on the Red Hat portal.

1. Change directories into the `docker` directory:

```
$ cd docker
```

2. Build and tag a `docker` image using this syntax:

To use a specific Alpine Image version, set the version using `--build-arg version=<alpine-version>`.

If you do not specify an image version, then the build command takes the *latest* Alpine image version automatically.

For example, to build and tag the image version 20.8.0 using alpine version 3.11.6 on Docker Hub registry for the account name `johndoe`:

3. When the build is successful, verify that the image appears in your local Docker repository.
   Example output:
4. Push the image:

```
$ docker push <registryname>/<accountname>/cluster-agent:<Agent-version>
```

For example, pushing the image version 20.8.0 to Docker Hub registry for the account name Johndoe:

```
$ docker push docker.io/johndoe/cluster-agent:20.8.0
```

Use Pre-Built Cluster Agent Image

You can also use a pre-built Alpine-based Cluster Agent or Rhel-based Cluster Agent to an internal registry.

Alpine Cluster Agent Image

You can use the pre-built Alpine-based Cluster Agent image from the AppDynamics Docker Hub account.

1. Log in to your Docker Hub account:

```
$ docker login docker.io -u <your-username> -p <your-password>
```

2. Pull the image:

```
$ docker pull docker.io/appdynamics/cluster-agent:<Agent-version>
```

For example, to pull the Alpine based cluster agent image version 20.8.0:

```
$ docker pull docker.io/appdynamics/cluster-agent:20.8.0
```

To push this image to another registry or accounts:

1. Rename the image:

```
$ docker tag docker.io/appdynamics/cluster-agent:<Agent-version> <registryname>/<accountname>/cluster-
agent:<Agent-version>
```

For example:

```
$ docker tag docker.io/appdynamics/cluster-agent:20.8.0 docker.io/johndoe/cluster-agent:20.8.0
```

2. Push the image:

```
$ docker push <registryname>/<accountname>/cluster-agent:<Agent-version>
```

For example, to push the image version 20.8.0 to Docker Hub registry for the account name Johndoe:

```
$ docker push docker.io/johndoe/cluster-agent:20.8.0
```

Rhel Cluster Agent Image

Alternatively, you can use the pre-built Rhel based Cluster Agent Image from the Appdynamics Redhat Container Registry account.

1. Log in to Red Hat Registry using your Customer Portal credentials:

```
$ docker login registry.connect.redhat.com -u <your-username> -p <your-password>
```

2. Pull the image:

```
$ docker pull registry.connect.redhat.com/appdynamics/cluster-agent:<Agent-version>
```

For example, to pull the Rhel based Cluster Agent version 20.8.0:

```
$ docker pull registry.connect.redhat.com/appdynamics/cluster-agent:20.8.0
```

You can now use the Cluster Agent image.

To push this image to another registry, or to other accounts:

1. Rename the image:

```
$ docker tag registry.connect.redhat.com/appdynamics/cluster-agent:<Agent-version> <registryname>
/<accountname>/cluster-agent:<Agent-version>
```

For example:

```
$ docker tag registry.connect.redhat.com/appdynamics/cluster-agent:20.8.0 scan.connect.redhat.com
/johndoe/cluster-agent:20.8.0
```

2. Push the image:

```
$ docker push <registryname>/<accountname>/cluster-agent:<Agent-version>
```

For example, to push the image version 20.8.0 to Redhat Registry for account name `johndoe`:

```
$ docker push scan.connect.redhat.com/johndoe/cluster-agent:20.8.0
```

Set the Image in Cluster Agent YAML Specification

Make sure to reference the image in the `cluster-agent.yaml` `spec:`

```
apiVersion: appdynamics.com/v1alpha1
kind: Clusteragent
metadata:
  name: k8s-cluster-agent
  namespace: appdynamics
spec:
  controllerUrl: "http://<account>.saas.appdynamics.com"
  account: "<account-name>"
  appName: "Cluster1"
  image: <registryname>/<username>/cluster-agent:<Agent-version>
```

For assistance, contact AppDynamics Support.

Cluster Agent and the Operator Compatibility Matrix

The compatibility matrix describes which Operator (Kubernetes or OpenShift) version is required for each Cluster Agent version.

- To view the list of features for the latest agent, see Release Notes.
- To view the list of features for older agents, see Past Agent Releases.

AppDynamics recommends that you upgrade the Operator with the Cluster Agent upgrade. See Upgrade the Cluster Agent. Ensure that use the latest YAML configuration files for the Operator and Cluster Agent from the upgraded version because the Cluster Agent may support new or modified configuration. If you do not update the Operator or YAML configuration file, and the Cluster Agent is upgraded, then the Cluster Agent may not work properly.

This table lists the Operator version with its supported Cluster Agent version:

Operator (Kubernetes/ OpenShift)	Cluster Agent
0.6.9	21.5.0
0.6.8	21.3.1, 21.3.0
0.6.7	21.2.0
0.6.6	20.12.1
0.6.6	20.12.0
0.6.5	20.11.0
0.6.4	20.11.0
0.6.3	20.10.0
0.6.2	20.9.1, 20.9.0
0.6.1	20.9.1, 20.9.0
0.5.4	20.8.0
0.5.3	20.7.0
0.5.2	20.6.0

Auto-Instrument Applications with the Cluster Agent

This page describes how to auto-instrument Kubernetes applications running in a cluster where the Cluster Agent is deployed. See Install the Cluster Agent.

For available instrumentation options, see Container Installation Options. AppDynamics recommends using auto-instrumentation for the simplest operational experience.

Auto-instrumentation supports dynamically adding the required App Server Agents to these application types:

- Java
- .NET Core on Linux
- Node.js

Enable Auto-Instrumentation

To enable auto-instrumentation, you:

- Add additional configuration to the `cluster-agent.yaml` file (described on this page), or
- Add configuration to the Cluster Agent Helm Chart `values.yaml` file (see Install the Cluster Agent with Helm Charts), and upgrade the Cluster Agent.

When the Cluster Agent detects a supported application deployment and the deployment matches the configured auto-instrumentation rules, the Cluster Agent modifies the application's deployment spec using the Kubernetes API. The Cluster Agent attaches an init container with the AppDynamics .NET Core, Node.js, or Java Agent image to the deployment. When the application restarts, the required agent is copied into the application container. As a result, the application container references the AppDynamics agent (Node.js Agent, .NET Core on Linux Agent, or Java Agent) in an auto-instrumented application.

Requirements

- Cluster Agent >= 20.5. See Cluster Agent Requirements and Supported Environments.
- Installed the latest Cluster Agent and Operator versions in the cluster. The `cluster-agent-operator.yaml` will set up the permissions required by the Cluster Agent to perform auto-instrumentation. See Install the Cluster Agent.
- At least one application is deployed to the cluster that was not previously instrumented with the required AppDynamics agent.
- A Controller with sufficient agent licenses based on the number of applications that will be auto-instrumented.
- Ensure that you have sufficient cluster capacity to handle pod restarts. See Minimize the Impact of Pod Restarts.

Application-Specific Requirements

- Node.js >= 8.6
- Java
  - Java applications must support including the `-javaagent` argument in the Java command using an environment variable. By default, the Cluster Agent uses `JAVA_TOOL_OPTIONS`; however, you can change this using the `defaultEnv` property.
  - Based on Java Agent resource requirements, it may be necessary to adjust the configured memory requests or limits for the application pods (see Install the Java Agent).
- .NET Core on Linux and Node.js
  - Ensure that the application base image OS matches the App Server Agent base image OS (Linux versus Alpine). For example, if your .NET Core on Linux application uses an Ubuntu base image, then you must set the `imageInfo.image` tag to the Linux version. In this example, the image tag is `20.11.0-linux`.
    If the application used an Alpine Linux base image OS, then the tag would be `20.11.0-alpine`. See the AppDynamics Docker Hub page.

    ```
    apiVersion: appdynamics.com/v1alpha1
    kind: Clusteragent
    metadata:
      name: k8s-cluster-agent
      namespace: appdynamics
    spec:
      # content removed for brevity
      instrumentationRules:
        - namespaceRegex: dev
          language: dotnetcore
          appName: MyDotNetAppOnUbuntu
          imageInfo:
            image: "docker.io/appdynamics/dotnet-core-agent:20.11.0-linux"
            agentMountPath: /opt/appdynamics
    ```

    If the base images do not match, then the App Server Agent may not start successfully. See Validate Auto-Instrumentation. The Node.js Agent also has specific Node.js runtime requirements that could prevent the agent from starting. See Node.js Supported Environments.

- For .NET Core and Node.js applications which communicate with an on-premises Controller, see Using Auto-Instrumentation with an On-premises Controller.
- If transaction analytics data is required, then you must configure the `analyticsHost` and `analyticsPort` properties.

Auto-Instrumentation Configuration

1. Set up the Cluster Agent feature to remove deleted pods from the Controller **Tiers & Nodes** Dashboard. Re-create the `cluster-agent-secret` that was created in Install the Cluster Agent to include `api-user`, and set the value to a local user from the Controller with the `Administrator` role:

   ```
   kubectl -n appdynamics delete secret cluster-agent-secret
   kubectl -n appdynamics create secret generic cluster-agent-secret --from-literal=controller-key=<access-key> --from-literal=api-user="<username>@<customer>:<password>"
   ```

   The Cluster Agent uses the `api-user` to mark the associated node in the Controller as historical upon pod deletion.

2. Add auto-instrumentation configuration to the `cluster-agent.yaml` file or the Helm values `YAML`. The structure of the auto-instrumentation configuration is based on a set of default properties. The addition of one, or more `instrumentationRules`, can target specific application deployments and may override the default properties.

   ```
   apiVersion: appdynamics.com/v1alpha1
   kind: Clusteragent
   metadata:
     name: k8s-cluster-agent
     namespace: appdynamics
   spec:
     # cluster agent properties
     #...
     # required to enable auto-instrumentation
     instrumentationMethod: Env
     # default auto-instrumentation properties
     # may be overridden in an instrumentationRule
     defaultProperty1: value
     defaultProperty2: value
     # ...
     # one or more instrumentationRules
     instrumentationRules:
       - property1: value
         property2: value
       - property1: value
         property3: value
       # ...
   ```

   To perform these common tasks, configure the specified properties:

   - To set the namespaces that are in scope for auto-instrumentation, configure these properties:
     - `nsToInstrument`
     - `nsToInstrumentRegex`
   - (Optional) To filter and target the set of applications within a namespace that are in scope for auto-instrumentation, configure these properties:
     - `instrumentationMatchString`
     - `labelMatch`
   - To assign an application and tier name to an instrumented application, configure these properties: (See Options for Naming the AppDynamics Application)
     - `appNameStrategy`
     - `tierName` (optional, will use deployment name by default)
   - To instrument which containers are in a multi-container application (default is to instrument the first container), configure these properties:
     - `instrumentContainer`
     - `containerMatchString`
   - If transaction analytics is required for .NET and Node.js applications, configure these properties for the analytics host and port (not required for Java):
     - `analyticsHost`
     - `analyticsPort`

   For available configuration options, see Cluster Agent Configuration for Auto-Instrumentation and Cluster Agent Configuration for Instrumentation Rules.

3. After you save the configuration, apply or upgrade the Cluster Agent deployment. The related pods and containers restart based on the deployment rollout strategy associated with the applications.

To validate and troubleshoot auto-instrumentation, see Validate the Cluster Agent Installation.

- If an application deployment does not match the properties defined in `instrumentationRules`, then auto-instrumentation is not enabled.
- If an auto-instrumentation property is not defined as a default, or in an `instrumentationRules`, then the Cluster Agent uses the corresponding default value specified in Cluster Agent Configuration for Auto-Instrumentation. If there are no corresponding default values, then auto-instrumentation is not enabled.

Configuration Examples

This first example targets Java applications in the namespaces that match the `ecom.*` pattern. Each matching application will be instrumented with a `20.20.1` Java Agent and will report to the `Ecommerce` application in the AppDynamics Controller. By default, the tier name will be the name of the Kubernetes deployment.

```
apiVersion: appdynamics.com/v1alpha1
kind: Clusteragent
metadata:
  name: k8s-cluster-agent
  namespace: appdynamics
spec:
  appName: "<app-name>"
  controllerUrl: "<protocol>://<appdynamics-controller-host>:8080"
  account: "<account-name>"
  image: "docker.io/appdynamics/cluster-agent:20.12.1"
  serviceAccountName: appdynamics-cluster-agent
  nsToMonitorRegex: ecom.*
  #
  # auto-instrumentation config
  #
  instrumentationMethod: Env
  nsToInstrumentRegex: ecom.*
  defaultAppName: Ecommerce
  instrumentationRules:
    - language: java
      imageInfo:
        image: docker.io/appdynamics/java-agent:20.20.1
        agentMountPath: /opt/appdynamics
```

The second example targets namespaces that contain Java and .NET Core on Linux applications.
The example incorporates these advanced configurations:

- Uses multiple `instrumentationRules` to target Java versus .NET Core on Linux applications.
- Uses the `labelMatch` strategy to determine the agent type and associated agent image based on the value of the `framework` label in each application spec.
- Rather than assigning a Controller application name in the `YAML` file, the configuration uses `appNameStrategy: label` to assign an application name based on a label from the application deployment spec.
- For the Java applications, it uses `instrumentContainer: select` and `containerMatchString: .*` to instruct the Cluster Agent to auto-instrument each container in a multi-container deployment, rather than just the first container (default).

```
apiVersion: appdynamics.com/v1alpha1
kind: Clusteragent
metadata:
  name: k8s-cluster-agent
  namespace: appdynamics
spec:
  appName: "<app-name>"
  controllerUrl: "<protocol>://<appdynamics-controller-host>:8080"
  account: "<account-name>"
  image: "docker.io/appdynamics/cluster-agent:20.12.1"
  serviceAccountName: appdynamics-cluster-agent
  nsToMonitorRegex: ecom.*
  #
  # auto-instrumentation config
  #
  instrumentationMethod: Env
  nsToInstrumentRegex: stage
  appNameStrategy: label
  instrumentationRules:
    - namespaceRegex: stage
      language: dotnetcore
      labelMatch:
        - framework: dotnetcore
      appNameLabel: appName
      imageInfo:
        image: "docker.io/appdynamics/dotnet-core-agent:20.11.0-linux"
        agentMountPath: /opt/appdynamics
    - namespaceRegex: stage
      language: java
      labelMatch:
        - framework: java
      appNameLabel: appName
      instrumentContainer: select
      containerMatchString: .*
      imageInfo:
        image: "docker.io/appdynamics/java-agent:21.3.0"
        agentMountPath: /opt/appdynamics
```

The third example is a deployment spec that defines the `appName` and `framework` label for a matching .NET Core on Linux application, based on the auto-instrumentation configuration from the second example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dotnet-app
  labels:
    appName: my-app
    framework: dotnetcore
spec:
   # ...
```

AppDynamics Application Name Strategies

The Controller's **Application Dashboard** provides three application name strategies. Select a strategy by assigning the `appNameStrategy` property to one of these values:

- manual: Use the `defaultAppName` or `appName` parameters in the `cluster-agent.yaml` file to set the application name.
- label: Use a label from the application's deployment spec as the application name.
- namespace: Use the Kubernetes namespace as the application name.

Manual

By default, the `appNameStrategy` is `manual`, which uses the `defaultAppName` or `appName` parameter to set the application name.

- If `defaultAppName` is provided, then use it (unless overwritten in an instrumentation rule).
- If `appName` is provided in an instrumentation rule, then use it.

For example in this spec, `ECommerce` is the default application name applied to the `ecom` and `groceries` namespace, and `BookStore` is the application name applied to the `books` namespace.

```
apiVersion: appdynamics.com/v1alpha1
kind: Clusteragent
metadata:
  name: k8s-cluster-agent
  namespace: appdynamics
spec:
  appName: "<cluster-name>"
  # ...
  # auto-instrumentation config
  instrumentationMethod: Env
  nsToInstrumentRegex: ecom|books|groceries
  appNameStrategy: manual
  defaultAppName: ECommerce
  instrumentationRules:
    - namespaceRegex: books
      appName: BookStore
```

Label

This option uses the `label` parameter as the application name strategy. To use the label option, specify a value in the `appNameLabel` parameter. Th e `appNameLabel` value refers to a label specified in the application deployment spec.

- If `spec.appNameLabel` is provided, then use it (unless overwritten in an instrumentation rule).
- If `appNameLabel` is provided in an instrumentation rule, then use it.

For example in this spec, the deployment spec label `appname` is used to set the application name in the `ecom` and `groceries` namespaces, and the label `app` is used in the `books` namespace.

```
apiVersion: appdynamics.com/v1alpha1
kind: Clusteragent
metadata:
  name: k8s-cluster-agent
  namespace: appdynamics
spec:
  appName: "<cluster-name>"
  # ...
  # auto-instrumentation config
  instrumentationMethod: Env
  nsToInstrumentRegex: ecom|books|groceries
  appNameStrategy: label
  appNameLabel: appname
  instrumentationRules:
    - namespaceRegex: books
      appNameLabel: app
```

For an application deployed to the `ecom` or `groceries` namespaces that sets the label `appname` (shown in this deployment spec snippet), it reports to the `eCommerce` application in the Controller's **Application Dashboard**.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ecom-app
  labels:
    appname: eCommerce
spec:
...
```

Namespace

This option uses the `namespace` parameter as the application name strategy and allows you to use the namespace name where an application is deployed as the application name in the Controller's **Application Dashboard**.

In this spec, each application in `ecom`, `books` and `groceries` namespace uses the application name based on the namespace that it is deployed to.

```
apiVersion: appdynamics.com/v1alpha1
kind: Clusteragent
metadata:
  name: k8s-cluster-agent
  namespace: appdynamics
spec:
  appName: "<cluster-name>"
  # ...
  # auto-instrumentation config
  instrumentationMethod: Env
  nsToInstrumentRegex: ecom|books|groceries
  appNameStrategy: namespace
```

Cluster Agent Configuration for Auto-Instrumentation

These configuration parameters are supported in the `cluster-agent.yaml` file for auto-instrumentation. For additional configuration parameters supported by the Cluster Agent, see Configure the Cluster Agent.

Parameter Name	Default Value	Description
`appNameLabel`	N/A	The value of this label will be the AppDynamics application name.
`appNameStrategy`	manual	The option to specify a name for the AppDynamics application. You can specify any of these values for this parameter: • manual • namespace • label See Options for Naming the AppDynamics Application.
`defaultAnalyticsHost`	N/A	The hostname of the Analytics Agent. This parameter is required if you require the Node.js Agent or the .NET Core Agent to send the default transaction data to the Analytics Agent. The default value is applied to all the instrumented resources unless it is overwritten by the `instrumentationRules` configuration.
`defaultAnalyticsPort`	N/A	The listening port for Analytics Agent. For example, if the Analytics Agent is listening on port 9090, then the value of this parameter is `9090`. The default value is applied to all the instrumented resources unless it is overwritten by the `instrumentationRules` configuration. This parameter is required along with `defaultAnalyticsHost` if you require the Node.js or the .NET Core Agent to send the default transaction data to the Analytics Agent.
`defaultAnalyticsSslEnabled`	N/A	This value is based on whether the Analytics Agent port is SSL enabled. If the port is SSL enabled, specify the value as `true`; else, specify as `false`. This parameter is required with `defaultAnalyticsPort` and `defaultAnalyticsHost`, if you require the Node.js or the .NET Core Agent, to send the default transaction data to the Analytics Agent. The default value is applied to all the instrumented resources unless it is overwritten by the `instrumentationRules` configuration.
`defaultAppName`	""	(Required) Application name used by the agent to report to the Controller.
`defaultCustomConfig`	N/A	This parameter is specific to Java applications. You can add any custom system property if your application framework requires any specific configuration for instrumentation. This value gets appended to the `env or defaultEnv` variable as configured with other Java Agent properties.

`defaultContainerMatchString`	N/A	This is a regex value to choose the containers to instrument. This parameter requires you to use the `select` option within the `instrumentContainer` (specified in the instrumentation rules). When the `select` option is used with this parameter, Cluster Agent instruments the containers that match the regex value.
`defaultEnv`	`JAVA_TOOL_OPTIONS`	This parameter is specific to Java applications. Environment variable to which the `-javaagent` argument and App Agent system properties are added. You can override this to use any other environment best suited for the deployment

`defaultInstrumentationLabelMatch`	`[]`	Specific deployment labels marked for instrumentation.
		
		This parameter accepts a list of `key-value` pairs to instrument.
		
		You must match a minimum of one label for instrumentation.
		
		For example:
		
		`defaultInstrumentationLabelMatch:`
		`- label1: value1`
		
		`- label1: value2`
		
		`- label2: value2`
		
		For example, if only `label1: value2` matches, the instrumentation works as expected.

`defaultInstrumentMatchString`	`.*`	Names of deployments targeted for instrumentation. This parameter is used as the default value for the `matchstring` parameter that is specified within instrumentationRules. If `matchstring` is not specified, then Cluster Agent uses this parameter value. This parameter accepts deployment names as a regular expression or regex. If there are multiple deployments to instrument, you can separate names with a '`|`' without spaces. By default, this parameter instruments all deployments configured by `nsToInstrumentRegex`. Therefore, if there is no value specified for `matchString` in the instrumentation rules and if you do not specify any value for this parameter, then Cluster Agent instruments every deployment.

`imageInfo`	`java:`
	`  image: "docker.io/appdynamics/java-  agent:latest"`
	`  agentMountPath: "/opt/appdynamics"`
	`  imagePullPolicy: "IfNotPresent"`
	
	`dotnetcore:`
	`  image: "docker.io/appdynamics/dotnet-core-agent:latest"`
	`  agentMountPath: /opt/appdynamics`
	
	`imagePullPolicy: "IfNotPresent"`
	
	`nodejs:`
	`  image: "docker.io/appdynamics/nodejs-agent:20.8.0-stretch-slimv14"`
	`  agentMountPath: /opt/appdynamics`
	`  imagePullPolicy: "IfNotPresent"`
		The Docker repository from where the Node.js Agent, .NET Core for Linux, and Java Agent is pulled.
		
		Supported values are:
		
		• `image`: Location of the agent image, along with its `tag/version`
		• `agentMountPath`: Location of image artifacts in the image file system. The default is `/opt/appdynamics`.
		You must change this configuration if the path is different.
		• `imagePullPolicy`: The pull policy required for the agent's docker image. You can choose one of the following pull policy based on your requirement:
		  ◦ `Always`
		  ◦ `IfNotPresent`
		  ◦ `Never`
		
		This parameter is used in the init containers that are added during auto-instrumentation.

`instrumentationMethod`	`None`	The instrumentation method used for instrumenting Apps. Supported values are: • `None`: Instrumentation is disabled. • `Env`: Attach the instrumentation properties to the container environment variables. If a value is not set, the instrumentation will not start.
`instrumentationRules`	`[ ]`	This is a mandatory parameter to enable auto-instrumentation. This includes the list of specific instrumentation rules. You can apply the rules to one or more namespaces, and can filter on deployment names and labels. Instrumentation rules are granular to support targeting specific deployments. See Cluster Agent Configuration for Instrumentation Rules.
`nsToInstrumentRegex`	`""`	Required. If you do not specify a value, auto-instrumentation will not work. Specify the namespaces to be instrumented as a regex. If there are multiple namespaces to instrument, separate namespaces using "|" without spaces. By default, namespaces are not instrumented.
numberOfTaskWorkers	2	To configure the rate limit for the number of deployments that are auto-instrumented at the same time. Increasing this value may lead to a larger number of concurrent pod restarts in the cluster.

`netvizInfo`	`bciEnabled: true`
	
	`port: 3892`
		To configure the Network Visibility App Agent, `netviz`, you must deploy a Network Agent separately (in addition to setting this parameter).
		See Install Infrastructure Visibility with the Cluster Agent Operator on how to install the Network Agent.
		
		By default, `netviz` is enabled. In the absence of a Network Agent, this property does not have any impact.
		You can enable or disable the `netviz` Agent per your requirements.
		
		Supported values are:
		
		1. `bciEnabled`: Boolean flag denoting whether `netviz` is enabled or not.
		2. `port`: The port the Network Agent is listening on (default is `3892`). Override this value only when running the Network Agent on a port other than the default port.

`runAsGroup`	`0`	If you configured the application container as a non-root user, provide the group ID (GID) of the corresponding group. `runAsGroup` is used to set the appropriate file permission on the agent artifacts. `The default runAsGroup` value is applied to all the instrumented resources, unless it is overwritten by the `instrumentationRules` configuration.
`runAsUser`	`0`	If you configured the application container as a non-root user, provide the user ID (UID) of the corresponding user. `runAsUser` is used to set the appropriate file permission on the agent artifacts. The default `runAsUser` value is applied to all the instrumented resources, unless it is overwritten by `instrumentationRules` configuration.

`resourcesToInstrument`	`Deployment`	Cluster Agent instruments the resources that are listed in this parameter. The default value is Deployment. If you need to instrument StatefulSets, add `StatefulSet` to the list.
		
		For example, to instrument Deployments and StatefulSets, configure:
		
		`resourcesToInstrument:`
		
		`- Deployment`
		`- StatefulSet`

Cluster Agent Configuration for Instrumentation Rules

The Cluster Agent uses a combination of `namespaceRegex`, `matchString,` and `labelMatch` for selecting an instrumentation rule on a first match basis. If the rules do not match, then the configuration uses the default values that were specified by these parameters:

- `nsToInstrumentRegex`
- `defaultInstrumentationLabelMatch`
- `defaultInstrumentMatchString`

For example, instrumentation rule `-matchString "<string>"` is added to the Cluster Agent configuration file:

```
instrumentationRules:
    - matchString: "<string>"
```

If the name of the deployment matches this string, then the remaining parameters under `instrumentationRules` are applied. If the string does not match, auto-instrumentation stops. If the parameters are not defined within the rules, then the Cluster Agent defaults to the parameters `nsToInstrumentRegex, defaultInstrumentationLabelMatch`, and `defaultInstrumentMatchString` to determine the instrumentation rules.

Parameter Name	Default Value	Description
`analyticsHost`	N/A	The hostname of the Analytics Agent. This parameter is required if you require the Node.js Agent or the .NET Core Agent to send the default transaction data to the Analytics Agent.
`analyticsPort`	N/A	The listening port for Analytics Agent. For example, if the Analytics Agent is listening on port 9090, then the value of this parameter is `9090`. This parameter is required along with `defaultAnalyticsHost` if you require the Node.js or the .NET Core Agent to send the default transaction data to the Analytics Agent.

`analyticsSslEnabled`	N/A	This value is based on whether the Analytics Agent port is SSL enabled or not. If the port is not SSL enabled, specify the value as `false`. This parameter is required along with `defaultAnalyticsPort` and `defaultAnalyticsHost` if you require the Node.js or the .NET Core Agent to send the default transaction data to the Analytics Agent.
`containerMatchString`	N/A	This is a regex value to choose the containers with the name that satisfies the value. This parameter requires you to use the `select` option within `defaultInstrumentContainer` or `instrumentContainer`. When the `select` option is used along with this parameter, Cluster Agent instruments the containers that match the regex value This parameter overrides the default value specified for `defaultContainerMatchString`.
`customAgentConfigSource`	N/A	This parameter provides an option to use the custom configuration of the instrumenting agents through ConfigMaps. This parameter requires that you create the required ConfigMaps in the Cluster Agent namespace. This parameter is dynamically configurable from the Cluster Agent YAML file. The changes that you make in the YAML file are updated to all the instrumented agents without restarting the application. Similarly, the changes that you make to the configuration in the ConfigMap is updated to all the instrumented agents without restarting the application. • `configMapName`: Specify the name of the ConfigMap. This allows the agent to use the same custom ConfigMap to update all the instrumented agent namespace. • `subDir`: (Required for Java Agent) Specify the relative path where the ConfigMaps are mounted. For example: `/ver20.8.0.3686/conf` , where `20.8.0.3686` is the version of the Java Agent. This directory may differ based on the Java Agent version. See Example 6. The absolute mount path for the ConfigMap is `<agent home path>/subDir` This `ConfigMap` replaces any ConfigMap of the same name in the target application's namespace. If you need to remove a ConfigMap file (used in the rules) from your deployment, then you must first remove this parameter from the Cluster Agent YAML file, and then remove the ConfigMap from the Cluster Agent's namespace.
`instrumentContainer`	`first`	This parameter provides an option to choose the container that must be instrumented. You can specify any of the following values: • `first`: This is the default value. When you specify this value, Cluster Agent instruments the first container in the pod. • `all`: When you specify this value, Cluster Agent instruments all the containers in the pod. • `select`: When you specify this value, Cluster Agent instruments only those containers with the name that matches a regex specified in the `defaultContainerMatchString` parameter or the `containerMatchString` parameter.
`language`	N/A	The language of the application to be instrumented. These languages are supported: • `dotnetcore` • `java` • `nodejs`
`matchString`	N/A	Regular expression to match on deployment name on which the rule applies. If you do not specify a value for this parameter, then the Cluster Agent uses the value specified in the `defaultInstrumentMatchString`, and instruments all the deployments that satisfy the value.

`labelMatch`	{}	A list of `key-value` pairs of labels to include in this rule. It is sufficient to match any one of the labels.
		
		For example:
		`labelMatch:`
		`-label1: value1`
		`-label1: value2`
		`-label2: value2`
		If only `label1: value2` matches, then instrumentation works as expected.

`appName`	`<defaultAppName>`	Application name used by the Java Agent to report to the Controller. This overrides `defaultAppName`. If no value is provided, then the configured `defaultAppName` is used.
`appNameLabel`	N/A	The value of this label is the AppDynamics application name.

`customAgentConfig`	N/A	This parameter is specific to Java applications. You can add any custom system property if your application framework requires any specific configuration for instrumentation. This value takes precedence over the default value specified in the `defaultCustomConfig` parameter. This value is appended to the `env` or `defaultEnv` variable as configured with other Java Agent properties.
`tierName`	""	Tier name used by the Java Agent to report to the Controller. If no tier name is provided, then the deployment name is used as the default.
`env`	""	This parameter is specific to Java applications. Environment variable to which the App Agent system properties will be added. When specified, this overrides `defaultEnv` for the deployments matching this instrumentation rule. If none are provided, it defaults to the `defaultEnv` (if configured), or to the default `env JAVA_TOOL_OPTIONS` (if not configured at default cluster level).

`imageInfo`	`image: "docker.io/appdynamics/java-agent:latest" agentMountPath: "/opt/appdynamics" imagePullPolicy: "IfNotPresent"`
	
	`dotnetcore: image: "docker.io/appdynamics/dotnet-core-agent:latest" agentMountPath: /opt/appdynamics imagePullPolicy: "IfNotPresent"`
	
	`nodejs: image: "docker.io/appdynamics/nodejs-agent:20.8.0-stretch-slimv14" agentMountPath: /opt/appdynamics imagePullPolicy: "IfNotPresent"`
		Location of the agent image. You can select one of these parameters:
		• `image`: Location of the agent image, and its tag/version.
		• `agentMountPath`: Location of image artifacts in the image file system. The default is `/opt/appdynamics`. This configuration is required only if the path differs from the default path.
		• `imagePullPolicy`: The pull policy required for the agent's Docker image. The default is `IfNotPresent`. You can choose one of these pull policies based on your requirement:
		  ◦ `Always`
		  ◦ `IfNotPresent`
		  ◦ `Never`
		This parameter is used in the init containers that are added during auto-instrumentation.
		
		The default value is `IfNotPresent`.
		
		For the specific language mentioned in this rule, this overrides `image-info` for the deployments matching this instrumentation rule.
		
		You must configure this if you want to override the default cluster-level configuration, and use a custom agent version for this specific rule selection.

`netvizInfo`	`bciEnabled: true`
	
	`port: 3892`
		To configure the Network Visibility App Agent, `netviz`, you must deploy a Network Agent separately (in addition to setting this parameter).
		See Install Infrastructure Visibility with the Cluster Agent Operator to install the Network Agent.
		
		By default, `netviz` is enabled. This property has no impact without a Network Agent.
		You can enable or disable the `netviz` Agent based on your requirements.
		
		Supported values are:
		
		1. `bciEnabled`: Boolean flag denoting whether `netviz` is enabled or not.
		2. `port`: The port the Network Agent is listening on (default is `3892`). Override this value only when running the Network Agent on a port other than the default port.

`runAsGroup`	`0`	If you configured the application container as a non-root user, provide the `groupId` of the corresponding group. This sets the appropriate file permission on the agent artifacts. This value is applied to all the instrumented resources. Add this parameter, if you require to override the default value of `runAsGroup` that is configured for default instrumentation, or if you require a specific value for the resources that satisfy this rule.
`runAsUser`	`0`	If you configured the application container as a non-root user, it provides the `userId` of the corresponding user. This sets the appropriate file permission on the agent artifacts. This value is applied to all the instrumented resources. Add this parameter, if you require to override the default value of `runAsUser` that is configured for default instrumentation, or if you require a specific value for the resources that satisfy this rule.

Configuration Examples

Example 1: Instrument all the deployments in the `ecom` namespace. The Java Agents running in each pod will register to the Controller with the `Ecommerce` application, and the tier names will default to the Kubernetes deployment names.

Example 2: Instrument only those deployments with names starting with `Payment` and with the label `module=payment` in any namespace.

Example 3: Instrument all deployments in namespaces `ecom` and `books`. For applications in the `books` namespace, use the `BookStore` application name in the Controller and a specific version of the Java Agent.

Example 4: Override the configuration in Example 3 for the `groceries` namespace to use a different Controller application name, and the `JAVA_OPTS` environment variable to inject the `-javaagent` Java argument.

Example 5: Instrument a Node.js application in namespace `books` and a .NET Core Linux application in namespace `groceries`. Also, the Node.js application must send the transaction data to the Analytics Agent.

Example 6: Instrument the containers with the names that start with `str` for the Java application in namespace `ecom`. Also, use the custom agent configurations in `controller-info.xml` and `app-agent-config.xml` configuration files that are present in the `controllerConf` ConfigMap along with the `log4j2.xml` file for logging settings of the agent present in `logConf` ConfigMap. These ConfigMaps are present in the Cluster Agent's namespace.

Example 7: Instrument all the deployments in the `groceries` namespace and instrument only those deployments with names starting with `Payment` in the `books` and `ecom` namespace.

Use Auto-Instrumentation with an On-Premises Controller

- The .NET Core and Node.js applications support only certificates signed by CA, not the self-signed certificates. If your Controller is using a self-signed certificate, only auto-instrumentation for Java applications is supported.
- To use the custom SSL certificate, ensure to configure the Cluster Agent uses SSL for on-premises Controllers. This certificate is used by the instrumentation agents. See Configure the Cluster Agent.
- For Node.js applications, you must update the certificate path in the `shim.js` file, and use this ConfigMap within the `customAgentConfigSource` parameter:
  `certificateFile: "/opt/appdynamics-nodejs/custom-ssl.pem"`
  For more information about this parameter, see Cluster Agent Configuration for Instrumentation Rule.

Minimize the Impact of Pod Restarts

When auto-instrumentation is enabled, the related pods restart based on the deployment rollout strategy associated with the deployments. To accommodate pods restarts, you may need to increase memory and CPU quotas associated with the impacted namespaces. To reduce the impact of restarting a large number of pods, the Cluster Agent (by default) allows only two concurrent auto-instrumentation tasks. The subsequent deployments (`resourcesToInstrument`) are auto-instrumented after the rollout of an instrumented deployment. However, you can configure the parameter, `numberOfTaskWorkers`, to specify the number of concurrent auto-instrumentation tasks based on your cluster's requirements.

Validate Auto-Instrumentation

This page describes how to validate and troubleshoot Cluster Agent auto-instrumentation.

Prerequisites

- Cluster Agent is reporting to the Controller. See Validate the Cluster Agent Installation.
- If you upgraded the Cluster Agent, you followed the Upgrade the Cluster Agent procedure and used a compatible set of images and YAML files.

Validate Auto-Instrumentation

1. After applying auto-instrumentation, confirm that the application is reporting to the Controller using the application name assigned based on the name strategy you chose. See Auto-Instrument Applications with the Cluster Agent.

2. Verify that the application pods targeted for auto-instrumentation have been recreated. As auto-instrumentation is being applied, use `kubectl get pods` with the `-w` flag to print out updates to pod status in real-time. A successfully auto-instrumented application shows that the app pod was recreated, and the correct init container was applied and copied the App Server Agent to the application container.

```
kubectl -n <app ns> get pods -w
NAME                          READY   STATUS            RESTARTS   AGE
dotnet-app-85c7d66557-s8hzm   1/1     Running           0          3m11s
dotnet-app-6c45b6d4f-fpp75    0/1     Pending           0          0s
dotnet-app-6c45b6d4f-fpp75    0/1     Init:0/1          0          0s
dotnet-app-6c45b6d4f-fpp75    0/1     PodInitializing   0          5s
dotnet-app-6c45b6d4f-fpp75    1/1     Running           0          6s
dotnet-app-85c7d66557-s8hzm   1/1     Terminating       0          20m\
```

   You can also use `kubectl get pods -o yaml` to check whether the application spec was updated to include the init container and the state of the init container:

```
kubectl -n <app-ns> get pod <app-pod> -o yaml
 ...
 initContainers:
  - command:
    - cp
    - -r
    - /opt/appdynamics/.
    - /opt/appdynamics-java
    image: docker.io/appdynamics/java-agent:latest
  ...
  initContainerStatuses:
  - containerID: docker://8bb892f322e5a043866d038631392a2272b143e54c8c431b3590312729043eb9
    image: appdynamics/java-agent:20.9.0
    imageID: docker-pullable://appdynamics/java-agent@sha256:
077ac1c4f761914c1742f22b2f591a37a127713e3e96968e9e570747f7ba6134
    ...
    state:
      terminated:
        containerID: docker://8bb892f322e5a043866d038631392a2272b143e54c8c431b3590312729043eb9
        exitCode: 0
        finishedAt: "2021-02-03T22:39:25Z"
        reason: Completed
```

   > The pod annotation may display `APPD_POD_INSTRUMENTATION_STATE` as `failed` for Node.js and .NET Core (Linux) applications even when the instrumentation is successful.

Troubleshoot Auto-Instrumentation When Not Applied

If the application pod was not recreated, and an init container was not applied, there may be an issue with the namespace and application matching rules used by the Cluster Agent.

1. First, confirm that the Cluster Agent is using the latest auto-instrumentation configuration.

```
kubectl -n appdynamics get cm instrumentation-config -o yaml
```

If the YAML output does not reflect the latest auto-instrumentation configuration, then validate the Cluster Agent YAML configuration, and apply or upgrade the Cluster Agent:

2. If the pods are still not recreated, then enable `DEBUG` logging in the Cluster Agent configuration:
3. Apply or upgrade the Cluster Agent to update the `DEBUG` logging configuration:
4. After you update the Cluster Agent logging configuration, tail the logs to search for messages indicating that the Cluster Agent has detected the application, and considers the application in scope for auto-instrumentation:

```
# apply grep filter to see instrumentation related messages:
kubectl -n appdynamics logs <cluster-agent-pod> -f | grep -E 'instrumentationconfig.go|deploymenthandler.
go'
```

However, if output displays (similar to this example) for the application you want to auto-instrument, then the Cluster Agent does not consider the application in scope for auto-instrumentation:

```
[DEBUG]: 2021-03-30 21:22:09 - instrumentationconfig.go:660 - No matching rule found for Deployment
dotnet-app in namespace stage with labels map[appName:jah-stage framework:dotnetcore]
[DEBUG]: 2021-03-30 21:22:09 - instrumentationconfig.go:249 - Instrumentation state for Deployment
dotnet-app in namespace stage with labels map[appName:jah-stage framework:dotnetcore] is false
```

Review your auto-instrumentation configuration to determine any necessary updates and save the changes. Then, delete and re-create the Cluster Agent as described previously. For example, if you overwrote the namespace configuration in an `instrumentationRule` using `namespaceRegex`, be sure that you use a value that is included in `nsToInstrumentRegex` (`dev` in the example):

```
apiVersion: appdynamics.com/v1alpha1
kind: Clusteragent
metadata:
  name: k8s-cluster-agent
  namespace: appdynamics
spec:
  # content removed for brevity
  nsToInstrumentRegex: stage|dev
  instrumentationRules:
    - namespaceRegex: dev
```

After you update the configuration and the Cluster Agent has been recreated, your output should be similar to this example indicating that the Cluster Agent recognizes the application as in scope for auto-instrumentation:

```
[DEBUG]: 2021-03-30 21:22:10 - instrumentationconfig.go:645 - rule stage matches Deployment spring-boot-
multicontainer in namespace stage with labels map[appName:jah-stage acme.com/framework:java]
[DEBUG]: 2021-03-30 21:22:10 - instrumentationconfig.go:656 - Found a matching rule {stage  map[acme.com
/framework:[java]]   java select .*  JAVA_TOOL_OPTIONS map[agent-mount-path:/opt/appdynamics image:
docker.io/appdynamics/java-agent:21.3.0 image-pull-policy:IfNotPresent] map[bci-enabled:true port:3892]
0 0 appName  0 false []} for Deployment spring-boot-multicontainer in namespace stage with labels map
[appName:jah-stage acme.com/framework:java]
[DEBUG]: 2021-03-30 21:22:10 - instrumentationconfig.go:249 - Instrumentation state for Deployment
spring-boot-multicontainer in namespace stage with labels map[appName:jah-stage acme.com/framework:java]
is true
[DEBUG]: 2021-03-30 21:22:10 - deploymenthandler.go:312 - Added instrument task to queue stage/spring-
boot-multicontainer
```

Troubleshoot Auto-Instrumentation When it Fails to Complete

If the application pod was re-created but the instrumented application is not reporting to the Controller, then auto-instrumentation did not complete successfully, or the App Server Agent failed to register with the Controller.

1. If the application pods are being restarted, but the init container is not completing successfully, then check for events in the application namespace using the **Cluster Agent Events** dashboard or `kubectl get events`. Some common issues that may prevent the init container from completing are: image pull or resource quotas issues.

```
kubectl -n <app ns> get events
```

2. Verify that the application deployment spec was updated with an init container, and if the status information includes any errors:

```
kubectl -n <app-ns> get pod <app-pod> -o yaml
```

3. Start a shell inside the application container to confirm the auto-instrumentation results. Check if the App Server Agent environment variables are set correctly and if the logs are free of errors.

```
kubectl -n <app-ns> exec -it <app-pod> /bin/sh
# Issue these commands in the container
env | grep APPD

# For Java app
env | grep JAVA_TOOL_OPTIONS # or value of defaultEnv
ls /opt/appdynamics-java
cat /opt/appdynamics-java/ver<version>/logs/<node-name>/agent.<date>.log

# For .NET Core app
ls /opt/appdynamics-dotnetcore
cat /tmp/appd/dotnet/1_agent_0.log

# For Node.js app
ls /opt/appdynamics-nodejs
cat /tmp/appd/<id>/appd_node_agent_<date>.log
```

4. If you are instrumenting a Node.js or .NET Core application, and the auto-instrumentation files have been copied to the app container, but the App Server Agent logs do not exist, or errors regarding binary incompatibilities display, then you may have a mismatch between the app image and the App Server Agent Image (referenced in the `image` property of the auto-instrumentation configuration).
For .NET Core, the image OS versions must match (Linux for example).
For Node.js, the image OS versions must match, as well as the Node.js version. To confirm, check the image tags, Dockerfiles, or exec-ing into the app container:

```
$ kubectl -n <app-ns> exec -it <app-pod> /bin/sh
cat /etc/os-release   # (or /etc/issue)
NAME="Alpine Linux"...
# for Node.js app
node -v
```

Uninstall Auto-Instrumentation from an Instrumented Application

This page describes how to uninstall instrumentation changes from an application that has been auto-instrumented using the Cluster Agent.

Uninstall Auto-Instrumentation

1. To remove the application from auto-instrumentation scope, update `cluster-agent.yaml` by using the `nsToInstrument`, `nsToInstrumentRegex`, `instrumentationMatchString` and `labelMatch` properties (depending on which properties you initially used).
   To completely disable auto-instrumentation, set the `instrumentationMethod` to `None`. See Auto-Instrument Applications with the Cluster Agent .
2. To update the auto-instrumentation configuration, apply or upgrade the Cluster Agent.
3. Delete and re-create the application deployment.

Configure App Agents to Correlate with Cluster Agent

This page describes how to configure APM container correlation with the Cluster Agent when manually instrumenting Kubernetes applications. See Instrument Kubernetes Applications Manually.

When you establish APM correlation, the application containers monitored by the Cluster Agent display in the Controller under the **Application /Container** view. The **Cluster Agent Pod** Dashboard also provides a link to the **APM Node** Dashboard in the **Pod Details** page.

> This configuration is required only when you use init containers or Dockerfiles to manually instrument applications in a cluster where the Cluster Agent is running. When you use Cluster Agent auto-instrumentation, APM container correlation is processed automatically. See Auto-Instrument Applications with the Cluster Agent.

APM container correlation uses the `UNIQUE_HOST_ID` (or `APPDYNAMICS_AGENT_UNIQUE_HOST_ID`) property of specific language agents. For examples of how to set this property, see the language-specific container installation page:

- Install the Java Agent in Containers
- Install the .NET Core Agent for Linux in Containers
- Install the Node.js Agent in Containers

The table lists the commands required to assign the `UNIQUE_HOST_ID` value based on runtime, platform, and App Agent type:

Platform	Container Runtime	Command for Container ID
Kubernetes	Docker	`UNIQUE_HOST_ID=$(sed -rn '1s#.*/##; 1s/(.{12}).*/\\1/p' /proc/self/cgroup)`
OpenShift 3.10 and 3.11	Docker	`UNIQUE_HOST_ID=$(sed -rn '1s#.*/##; 1s/docker-(.{12}).*/\\1/p' /proc/self/cgroup)`
OpenShift 4.x	CRI-O	`UNIQUE_HOST_ID=$(cat /proc/self/cgroup | head -1 | awk -F '/' ' {print $NF}' | awk -F '-' '{print $2}' | cut -c 1-12) ; if [ -z $UNIQUE_HOST_ID]; then UNIQUE_HOST_ID=$(cat /proc/self/cgroup | head -1 | awk -F '/' '{print $NF} ' | cut -c 1-12) ; fi ;`

Kubernetes with Docker

Java APM correlation works out-of-the-box because the Agent retrieves the `UNIQUE_HOST_ID` value automatically. For other Agents with the `UNIQUE_HOST_ID` property, you must set the property using this command: `UNIQUE_HOST_ID=$(sed -rn '1s#.*/##; 1s/-(.{12}).*/\\1/p' /proc/self/cgroup)`

For all language Agents with a `uniqueHostId` property, set the property using this method when building the App image:

```
UNIQUE_HOST_ID=$(sed -rn '1s#.*/##; 1s/(.{12}).*/\\1/p' /proc/self/cgroup)
JAVA_OPTS="$JAVA_OPTS -Dappdynamics.agent.uniqueHostId=$UNIQUE_HOST_ID"
```

If you cannot change the application image, you can assign the `UNIQUE_HOST_ID` in the Kubernetes deployment specification by modifying the `command` attribute:

```
command: ["/bin/sh"]
args: ["-c", "UNIQUE_HOST_ID=$(sed -rn '1s#.*/##; 1s/(.{12}).*/\\1/p' /proc/self/cgroup) && java -
Dappdynamics.agent.uniqueHostId=$UNIQUE_HOST_ID $JAVA_OPTS -jar /java-services.jar"]
```

This approach implies that the entry point of the application is known. In the example, the original entry point is:

```
java $JAVA_OPTS -jar /java-services.jar
```

OpenShift 3.10 and 3.11 with Docker Runtime

For all language Agents with the `UNIQUE_HOST_ID` property in OpenShift v3.10 and v3.11 with Docker runtime, set the property using this method when building the App image:

```
UNIQUE_HOST_ID=$(sed -rn '1s#.*/##; 1s/docker-(.{12}).*/\\1/p' /proc/self/cgroup)
```

If you cannot change the application image, you can assign the `UNIQUE_HOST_ID` in the OpenShift v3.10 or v3.11 deployment specification by modifying the `command` attribute:

```
command: ["/bin/sh"]
args: ["-c", "UNIQUE_HOST_ID=$(sed -rn '1s#.*/##; 1s/docker-(.{12}).*/\\1/p' /proc/self/cgroup) && java -
Dappdynamics.agent.uniqueHostId=$UNIQUE_HOST_ID $JAVA_OPTS -jar /java-services.jar"]
```

This approach implies that the entry point of the application is known. In the example, the original entry point is:

```
java $JAVA_OPTS -jar /java-services.jar
```

OpenShift 4.x with CRI-O Runtime

For all language Agents with the `UNIQUE_HOST_ID` property in OpenShift v4.x with CRI-O runtime, set the property using this method when building the App image:

```
UNIQUE_HOST_ID=$(cat /proc/self/cgroup | head -1 | awk -F '/' '
{print $NF}' | awk -F '-' '{print $2}' | cut -c 1-12) ; if [ -z $UNIQUE_HOST_ID ]; then UNIQUE_HOST_ID=$(cat
/proc/self/cgroup | head -1 | awk -F '/' '{print $NF}' | cut -c 1-12) ; fi ;
```

If you cannot change the application image, you can assign the `UNIQUE_HOST_ID` in the OpenShift v4.x deployment specification by modifying the `command` attribute:

```
command: ["/bin/sh"]
args: ["-c", "UNIQUE_HOST_ID=$(cat /proc/self/cgroup | head -1 | awk -F '/' '{print $NF}' | awk -F '-'
'{print $2}' | cut -c 1-12) ; if [ -z $UNIQUE_HOST_ID ]; then UNIQUE_HOST_ID=$(cat /proc/self/cgroup | head
-1 | awk -F '/' '{print $NF}' | cut -c 1-12) ; fi ;  java -Dappdynamics.agent.uniqueHostId=$UNIQUE_HOST_ID
$JAVA_OPTS -jar /java-services.jar"]
```

This approach implies that the entry point of the application is known. In the example, the original entry point is:

```
java $JAVA_OPTS -jar /java-services.jar
```

Use the Cluster Agent

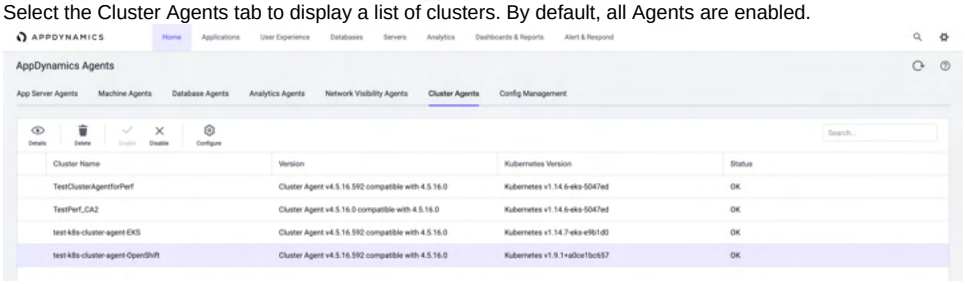
You administer the Cluster Agent from the AppDynamics Controller where you can enable, disable, configure, and delete the Cluster Agent. You can also change which namespaces to monitor.

Any configuration change you make in the Controller also applies to the Cluster Agent. These changes take a couple of minutes, and the status displays with a progress indicator.

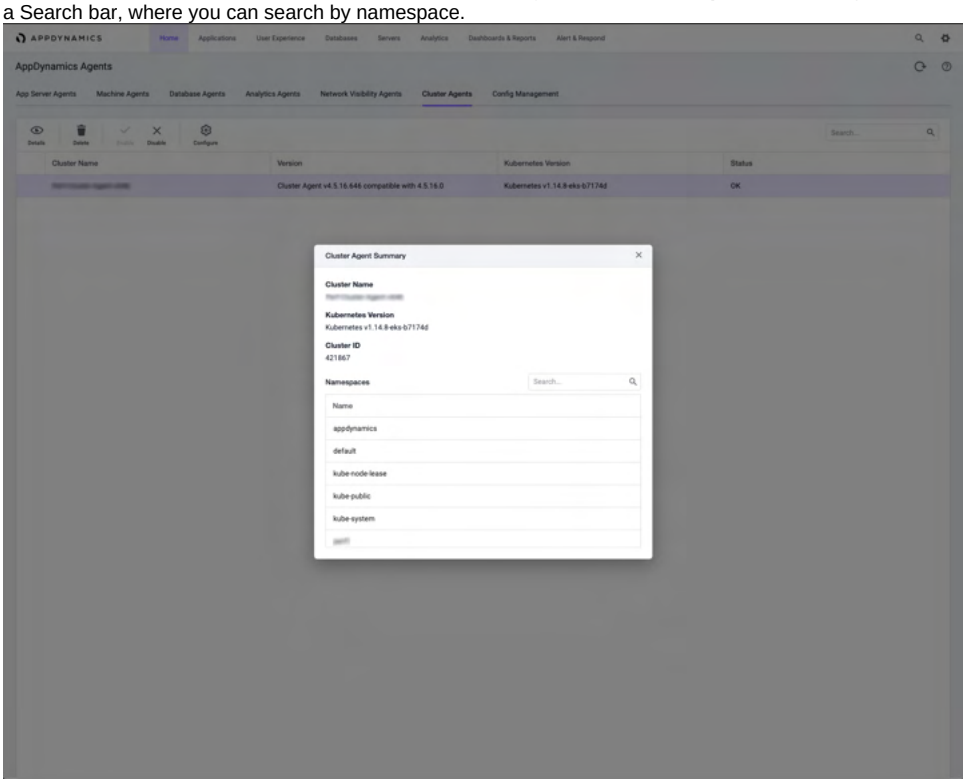
View the Cluster Agent Summary

From the **Controller UI** panel:

1. In the upper-right corner, click Settings **> AppDynamics Agents**.
2. Select the Cluster Agents tab to display a list of clusters. By default, all Agents are enabled.



3. Double-click a Cluster Agent. A dialog appears showing a **Cluster Agent Summary** of your current settings. The summary card shows the **Cluster Name**, **Kubernetes Version**, **Cluster ID**, and currently monitored **Namespaces**. To the right of the **Namespaces** table is a Search bar, where you can search by namespace.

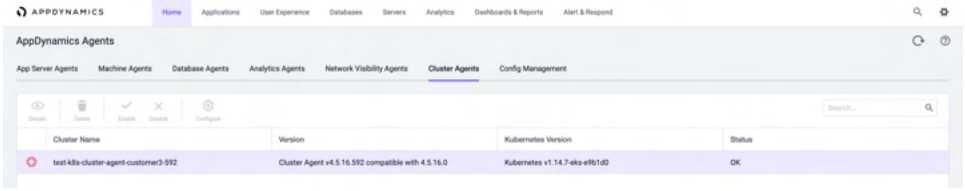


4. Click **X** to close the Cluster Agent Summary card.

Disable the Cluster Agent

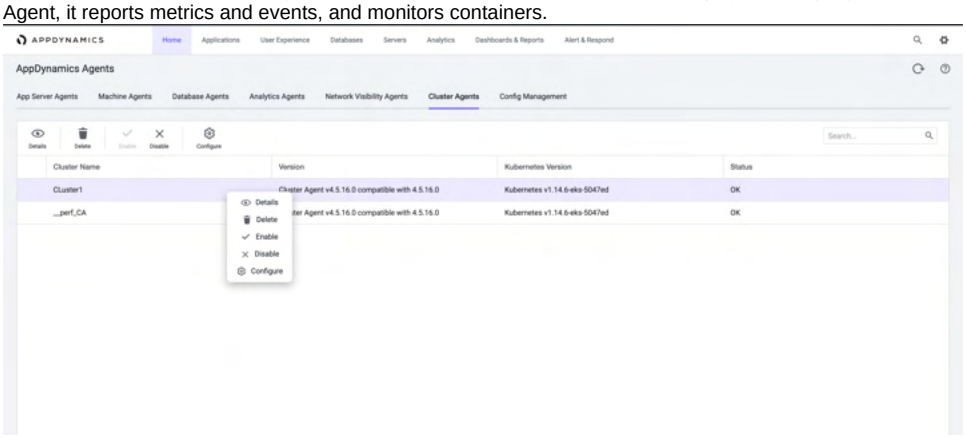
1. Select a Cluster Agent from the list.

2. Click Disable ✕. You can also select the Cluster name, right-click, and select **Disable**. A dialog appears confirming that you disabled the agent successfully. These changes take a couple of minutes, and the status displays with a progress indicator. When you disable the Cluster Agent, it does not report metrics or events, nor monitor pods or containers. You can re-enable a disabled Cluster Agent.



Enable the Cluster Agent

1. Select a Cluster Agent from the list.
2. Click Enable ✓. You can also select the Cluster name, right-click, and select **Enable**. A dialog appears confirming that you enabled the agent successfully. These changes take a couple of minutes, and the status displays with a progress indicator. When you enable the Cluster Agent, it reports metrics and events, and monitors containers.



Edit Namespaces

You can modify the namespaces by using either `nsToMonitor` or `nsToMonitorRegex` fields.

These are common namespace behavior scenarios with the `nsToMonitor` field:

- After the Cluster Agent's initial registration, the namespaces are retrieved from the Cluster Agent `YAML` file.
- After initial registration, if the Cluster Agent restarts and the Controller is running, the Controller UI's user namespace settings take precedence.
- After initial registration, if the Controller restarts and the Cluster Agent is running, any previous changes made to the namespaces in the Controller UI are preserved.
- If both the Cluster Agent and the Controller restart, the namespaces are retrieved from the `YAML` file.

The following are common namespace behavior scenarios with `nsToMonitorRegex` field:

> To use the `nsToMonitorRegex` field, ensure that you are using Controller >= 20.10 and Agent >= 20.9.

- After initial registration, at every subsequent minute the agent checks for the `nsToMonitorRegex` field to match the namespaces. If there is `nsToExcludeRegex` mentioned, those namespaces are excluded from the monitored namespaces.
- If a namespace configuration is to be monitored or ignored from the Controller, it takes precedence over the agent configuration (Cluster Agent YAML file). Therefore, if a namespace satisfies the regular expression that is configured for `nsToExcludeRegex`, but the Controller configuration includes that namespace under monitored namespaces, then the namespace is monitored.
  To exclude this namespace from being monitored, you must delete the namespace in the Controller configuration. In this scenario, where there is a conflict between the agent configuration and the Controller configuration, the addition or removal of the relevant namespace from the Cluster Agent configuration does not have any effect.
- If the Cluster Agent is restarted, the `nsToMonitorRegex` and `nsToExcludeRegex` fields are read from the agent configuration file along with the namespaces configured in the Controller.
- If the Controller is restarted, the monitored and ignored namespaces included in the Controller configuration are retained and reapplied along with the agent configuration.
- If both the Cluster Agent and the Controller are restarted at the same time, the `nsToMonitorRegex` and `nsToExcludeRegex` fields are read from the agent configuration file. The namespaces to be monitored are determined based on the agent configuration.
- If a new namespace matching the configured regex is added, the namespace is monitored without any additional configuration. Similarly, when a namespace does not match the configured regex, it is not monitored.

You can modify namespaces in a Cluster by:

- Updating Namespaces in the Cluster Agent Configuration File - Modify the `nsToMonitor` or the `nsToMonitorRegex` field in the `cluster-agent.yaml` file before deploying the Cluster Agent. Namespaces mentioned in the `nsToMonitor` field are considered only during initial registration. You can update the `nsToMonitorRegex` field in the `cluster-agent.yaml` file even after the registration.
  You can specify regular expressions (regex) for the namespaces that you want to monitor by using the `nsToMonitorRegex` field. If you do not require the Cluster Agent to monitor some of the namespaces that match the regex specified in the `nsToMonitorRegex` field, then you can specify `nsToExcludeRegex` along with `nsToMonitorRegex`. For example, if Cluster Agent must monitor all the namespaces except the namespace that has the name starting with exclude, specify `nsToMonitorRegex` as `.*` and `nsToExcludeRegex` as `^exclude`.
- Editing Namespaces in the User Interface - Edit namespaces after you deploy the Cluster Agent. Once deployed, the cluster is listed under AppDynamics Agents/Cluster Agents. Open the AppDynamics browser-based configuration user interface and edit namespaces.

Update Namespaces in the Cluster Agent Configuration File

To update namespaces in a cluster, use the `nsToMonitor` or the `nsToMonitorRegex` field in the `cluster-agent.yaml` before deploying the Cluster Agent.

Updating namespaces using the `nsToMonitor` field in the `cluster-agent.yaml` file does not change your currently monitored namespaces. However, updating namespaces using the nsToMonitorRegex changes the currently monitored namespaces. The scenarios are mentioned at Edit Namespaces.

If the Cluster Agent pod unexpectedly restarts, the namespaces stored in the Controller take precedence and are preserved. See Troubleshoot the Cluster Agent.

To update namespaces:

1. Open the `cluster-agent.yaml` file in a text editor.
2. Locate the `nsToMonitor` field:

   ```
   nsToMonitor:
       - "default"
       - "appdynamics"
   ```

   Or
   To specify namespaces based on regular expressions, include the `nsToMonitorRegex` field:

   ```
   nsToMonitorRegex: <regular expression>
   nsToExcludeRegex: <regular expression> #This is an optional field
   ```
3. Verify the list of namespaces. If it is not correct, add or remove namespaces in the `cluster-agent.yaml` file.
4. Open a command prompt. To apply the namespaces changes, enter:

   ```
   kubectl apply -f cluster-agent.yaml
   ```
5. To verify that the AppDynamics Operator registered the monitored namespace value, enter:
   When using the `nsToMonitor` field:

   ```
   kubectl -n appdynamics describe cm cluster-agent-config | grep
   APPDYNAMICS_CLUSTER_MONITORED_NAMESPACES -A 2
   ```

   When using the `nsToMonitorRegex` field:

   ```
   kubectl-app describe cm cluster-agent-mon
   ```

This is the expected output from the `nsToMonitor` example:

```
APPDYNAMICS_CLUSTER_MONITORED_NAMESPACES:
----
default
```

If you use the `nsToMonitorRegex` field, the default value of `nsToMonitor` is ignored.

Edit Namespaces in the User Interface

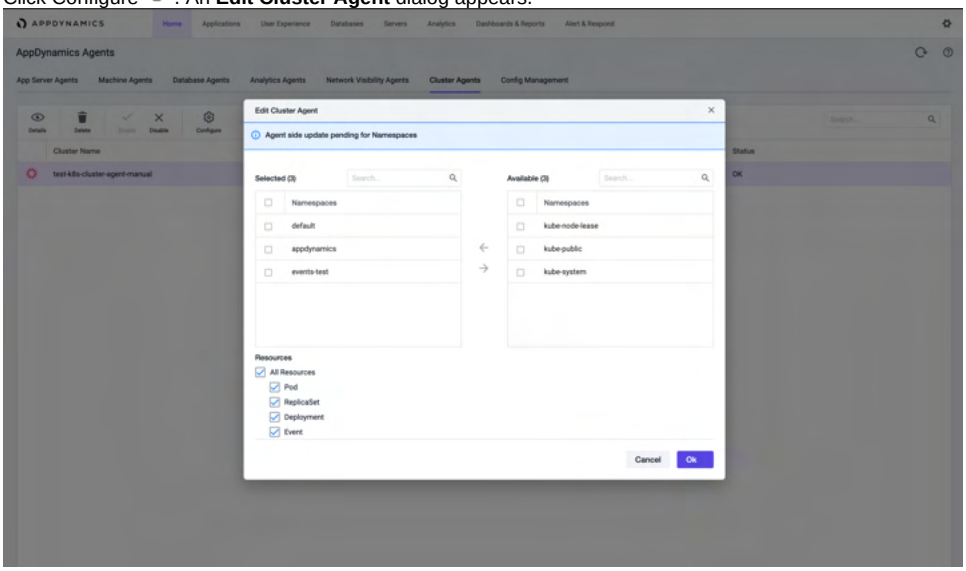
Once you have deployed the Cluster Agent, the **Selected** namespaces column displays the currently monitored namespaces.

To edit a namespace from the cluster, you must have administrator permissions:

To edit a namespace:

1. Double-click a cluster from the list.

2. Click Configure. An **Edit Cluster Agent** dialog appears.



3. To modify namespaces, select a namespace by checking the corresponding box.
4. Once the desired namespace box is checked, click the arrow to move it to either the **Selected** namespace table or the **Available** namespace table.
5. Click **OK** to close the dialog box. These changes take a couple of minutes and the status displays in a banner notification.

Updating namespaces in the `cluster-agent.yaml` file does not change your currently monitored namespaces. It assures that if the Cluster Agent pod unexpectedly restarts, that your namespaces are preserved. See Troubleshoot the Cluster Agent.

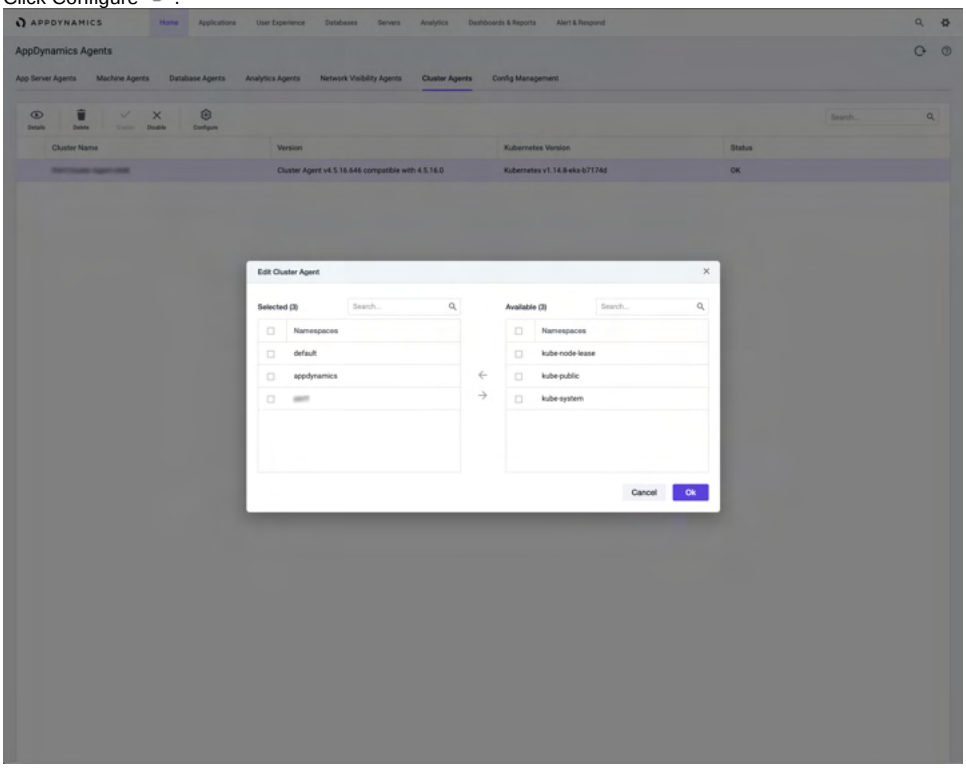
Add or Remove Namespaces

To add or remove a namespace from the cluster, you must have administrator permissions:

To add or remove a namespace from the cluster:

1. Double-click a cluster from the list.

2. Click Configure .



3. To modify namespaces, select a namespace by checking the corresponding box. Once the desired namespace box is checked, click the arrow to move it to either the **Selected** namespace table, or the **Available** namespace table.
4. Click **OK** to close the dialog box. These changes take a couple of minutes and the status displays with a progress indicator..

Delete a Monitored Namespace

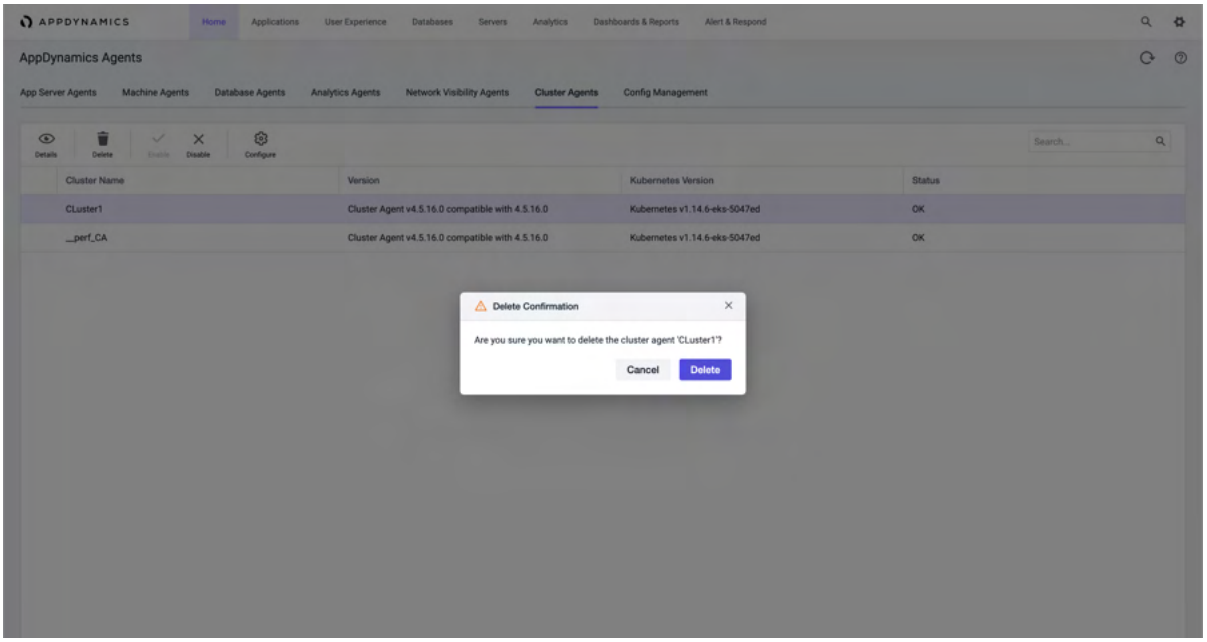
To delete a Cluster Agent monitored namespace:

1. Double-click a cluster from the list.
2. Click Configure . The **Edit Cluster Agent** dialog appears.
3. Select a namespace by checking the corresponding box.
4. Click the arrow to move to either the **Available** namespace, or to the **Selected** namespace table.
5. Delete the namespace from your Kubernetes cluster.

Delete the Cluster Agent

To delete a Cluster Agent:

1. Select the Cluster Agent name that you want to delete.
2. Click Delete . A dialog asks you to confirm that you want to delete the Cluster Agent.
3. Click **Delete**. If you delete a Cluster Agent, you will lose all historical data.


APPDYNAMICS
Home
Applications
User Experience
Databases
Servers
Analytics
Dashboards & Reports
Alert & Respond
AppDynamics Agents
App Server Agents
Machine Agents
Database Agents
Analytics Agents
Network Visibility Agents
Cluster Agents
Config Management
Details
Delete
Enable
Disable
Configure
Search...
Cluster Name
Version
Kubernetes Version
Status
CLuster1
Cluster Agent v4.5.16.0 compatible with 4.5.16.0
Kubernetes v1.14.6-eks-5047ed
OK
__perf_CA
Cluster Agent v4.5.16.0 compatible with 4.5.16.0
Kubernetes v1.14.6-eks-5047ed
OK
Delete Confirmation
Are you sure you want to delete the cluster agent 'CLuster1'?
Cancel
Delete

Cluster Metrics

The **Cluster Agent Dashboard** metrics derive from the Kubernetes API, and they report information for the clusters and pods. For any defined set of namespaces, the Cluster Agent reports events on these Kubernetes and hardware resources.

AppDynamics monitors cluster health and Kubernetes objects for:

- Cluster Summary Metrics
- CPU
- DaemonSets
- Deployments
- Endpoints
- Jobs
- Memory
- Nodes
- Pods
- PVC
- ReplicaSets
- StatefulSets
- Services
- Storage Quota

Cluster Summary Metrics

Metric Name	Description	UI Location	Metric Path
Error events count	Number of error events	**Dashboard > Errors**	Hardware Resources|Cluster|Error events count
Evicted pods count	Number of evicted pods	**Pods > Evicted**	Hardware Resources|Cluster|Evicted pods count
Eviction threats count	Number of events that represent pod evictions	**Dashboard > Errors**	Hardware Resources|Cluster|Eviction threats count
Image pull errors	Number of image pull errors	**Dashboard > Issues > Image Issues**	Hardware Resources|Cluster|Image pull errors
Image pulls	Number of image pulls	**Dashboard > Issues > Image Issues**	Hardware Resources|Cluster|Image pulls
Info events count	Number of informational events	**Dashboard > Errors**	Hardware Resources|Cluster|Info events count
Pod errors	Number of errors related to pods	**Dashboard > Issues > Pod Issues**	Hardware Resources|Cluster|Pod errors
Pod Kills	Number of pods that were killed	**Inventory > Pods > Pod Kills**	Hardware Resources|Cluster|Pod Kills
Pod restarts	Number of times the pods restarted	**Dashboard > Issues > Pod Issues**	Hardware Resources|Cluster|Pod restarts
Pods Scaledowns	Count of scaledowns; you can scale down your deployments and replica sets.	**Inventory > Pods > Scaledowns**	Hardware Resources|Cluster|Pods Scaledowns
Pods count	Total count of pods	**Inventory > Pods > Phases > Normal**	Hardware Resources|Cluster|Pods count
Pods failed	Number of failed pods	**Pods > Failed**	Hardware Resources|Cluster|Pods failed
Pods pending	Number of pods in a pending state. Pending status normally indicates an issue. See the Kubernetes documentation.	**Pods > Pending**	Hardware Resources|Cluster|Pods pending
Pods running	Number of pods in a running state	**Pods > Running**	Hardware Resources|Cluster|Pods running
Pods succeeded	Number of pods in Succeeded phase	**Dashboard > Pods By Phase**	Hardware Resources|Cluster|Pods succeeded
Pods unknown	Number of pods in Unknown state	**Dashboard > Pods By Phase**	Hardware Resources|Cluster|Pods unknown
Pods with Missing Dependencies - Config Maps and Secrets	If a pod is dependent on any Config Maps & Secrets, then those dependencies are missing.	**Inventory > Pods > Missing Dependencies - Config Maps and Secrets**	Hardware Resources|Cluster|Pods With Missing Dependencies - Config Maps And Secrets (Pod Metrics for Inventory tab)

Pods with Missing Dependencies - Services	If a pod is dependent on any Services, then those dependencies are missing.	**Inventory > Pods > Missing Dependencies - Services**	Hardware Resources|Cluster|Pods With Missing Dependencies (Pod Metrics for Inventory Tab)
Pods with No Limits	Number of pods with no limits (on CPU/memory) set. If you specified limits on any pod that you are starting, this metric indicates how many pods do not have a limit defined (Displays in the Inventory tab, under Pod Metrics).	**Inventory > Pods > No Limits**	Hardware Resources|Cluster|Pods With No Limits
Pods With No Liveness Probe	Number of pods with no liveness probe. If you configured a probe in Kubernetes to monitor liveness, the values display in the Inventory tab, under Pod Metrics.	**Inventory > Pods > No Probes -Liveness**	Hardware Resources|Cluster|Pods With No Liveness Probe
Pods With No Readiness Probe	Number of pods with no readiness probe. If you configured a probe in Kubernetes to monitor readiness, the values display in the Inventory tab, under Pod Metrics.	**Inventory > Pods > No Probes -Readiness**	Hardware Resources|Cluster|Pods With No Readiness Probe
Privileged Pods	Number of privileged pods that run with root access (Displays in the Inventory tab, under Pod Metrics).	**Inventory > Pods > Privileged**	Hardware Resources|Cluster|Privileged Pods
Storage errors	Overall number of errors related to storage for the cluster.	**Inventory > Pod Metrics**	Hardware Resources|Cluster|Storage errors
Storage quota violations	Number of storage quota violations; if someone exceeds that quota.	**Inventory > Pod Metrics**	Hardware Resources|Cluster|Storage quota violations

CPU

CPU Capacity

Metric Name	Description	UI Location	Metric Path
Total (MilliCores)	Total CPU capacity for the cluster in MilliCores	**Cluster Capacity > CPU**	Hardware Resources|Cluster|CPU|Capacity|Total (MilliCores)
Used (MilliCores)	CPU capacity already used by the cluster in MilliCores	**Cluster Capacity > CPU**	Hardware Resources|Cluster|CPU|Capacity|Used (MilliCores)

CPU Quota

Metric Name	Description	UI Location	Metric Path
Limit Used (%)	Percentage of CPU limit quota used	**Dashboard > Quotas > CPU Limit**	Hardware Resources|Cluster|CPU|Quota|Limit Used (%)
Limit Used (MilliCores)	MilliCores value for CPU limit quota used	**Dashboard > Quotas > CPU Limit**	Hardware Resources|Cluster|CPU|Quota|Limit Used (MilliCores)
Request Used (%)	Percentage of CPU request quota used	**Dashboard > Quotas > CPU Request**	Hardware Resources|Cluster|CPU|Quota|Request Used (%)
Request Used (MilliCores)	MilliCores value for CPU request quota used	**Dashboard > Quotas > CPU Request**	Hardware Resources|Cluster|CPU|Quota|Request Used (Millicores)

CPU Utilization

Metric Name	Description	UI Location	Metric Path
Limit (MilliCores)	Limit of CPU which can be used by the pods. Only the pods belonging to monitored namespaces are used to calculate this metric.	**Dashboard > Utilization > CPU**	Hardware Resources|Cluster|CPU|Utilization|Limit (MilliCores)
Request (MilliCores)	MilliCore value of CPU for which all the pods in monitored namespaces have requested.	**Dashboard > Utilization > CPU**	Hardware Resources|Cluster|CPU|Utilization|Request (MilliCores)
Used (MilliCores)	Actual CPU which the pods from monitored namespaces are currently using.	**Dashboard > Utilization > CPU**	Hardware Resources|Cluster|CPU|Utilization|Used (MilliCores)

DaemonSets

Metric Name	Description	UI Location	Metric Path

Count	Number of daemon sets that exist	**Inventory > Objects > DaemonSets > (Count)**	HardwareResources|Cluster|DaemonSets|Count
Nodes Available	Number of nodes that are running and available on the cluster	**Inventory > Objects > DaemonSets > Available**	HardwareResources|Cluster|DaemonSets|Nodes Available
Nodes MissScheduled	Number of nodes that are running, but should not be running	**Inventory > Objects > DaemonSets > MissScheduled**	HardwareResources|Cluster|DaemonSets|Nodes MissScheduled
Nodes Unavailable	Number of nodes that should be running, but are not running	**Inventory > Objects > DaemonSets > Unavailable**	HardwareResources|Cluster|DaemonSets|Nodes Unavailable

Deployments

Metric Name	Description	UI Location	Metric Path
Count	Number of deployments that exist in the cluster	**Inventory > Objects > Deployments > (Count)**	HardwareResources|Cluster|Deployments|Count
Replicas	Number of pod replicas in the cluster that are not in a terminated state	**Inventory > Objects > Deployments > Available**	HardwareResources|Cluster|Deployments|Replicas
Replicas Unavailable	Total number of unavailable pod replicas across all deployments in the cluster	**Inventory > Objects > Deployments > Unavailable**	HardwareResources|Cluster|Deployments|ReplicasUnavailable

Endpoints

Metric Name	Description	UI Location	Metric Path
Count	Number of endpoints in the cluster	**Inventory > Services > Endpoints > Count**	HardwareResources|Cluster|Endpoints|Count
Not Ready Address	Total number of not ready addresses for all the endpoints in the cluster	**Inventory > Services > Endpoints without ready IP**	HardwareResources|Cluster|Endpoints|Not Ready Address
Orphans	Total number of endpoints in the cluster which do not have any ready, nor any not ready addresses	**Inventory > Services > Orphan Endpoints with no IP**	HardwareResources|Cluster|Endpoints|Orphans
Ready Address	Total number of ready addresses for all the endpoints in the cluster	**Inventory > Services > Endpoints**	HardwareResources|Cluster|Endpoints|Ready Address

Jobs

Metric Name	Description	UI Location	Metric Path
Count	Total number of jobs in the cluster	**Inventory > Objects > Jobs > (Count)**	Hardware Resources|Cluster|Jobs|Count
Pods Active	Total number of active pods for all the jobs in the cluster	**Inventory > Objects > Jobs > Active**	Hardware Resources|Cluster|Jobs|Pods Active
Pods Failed	Total number of pods which reached phase Failed for all the jobs in the cluster	**Inventory > Objects > Jobs > Failed**	Hardware Resources|Cluster|Jobs|Pods Failed
Pods Succeeded	Total number of pods which reached phase Succeeded for all the jobs in the cluster	**Inventory > Objects > Jobs > Succeeded**	Hardware Resources|Cluster|Jobs|Pods Succeeded

Memory

Memory Capacity

Metric Name	Description	UI Location	Metric Path
Total (MB)	Total Memory capacity for the cluster in MBs	**Dashboard > Cluster Capacity > Memory**	Hardware Resources|Cluster|Memory|Capacity|Total (MB)
Used (MB)	Memory capacity already used by the cluster in MBs	**Dashboard > Cluster Capacity > Memory**	Hardware Resources|Cluster|Memory|Capacity|Used (MB)

Memory Quota

Metric Name	Description	UI Location	Metric Path
Limit Used (%)	Percentage of Memory limit quota used	**Dashboard > Quotas > Memory Limit**	Hardware Resources|Cluster|Memory|Quota|Limit Used (%)
Limit Used (MB)	MB value for Memory limit quota used	**Dashboard > Quotas > Memory Limit**	Hardware Resources|Cluster|Memory|Quota|Limit Used (MB)
Request Used (%)	Percentage of Memory request quota used	**Dashboard > Quotas > Memory Request**	Hardware Resources|Cluster|Memory|Quota|Request Used (%)
Request Used (MB)	MB value for Memory request quota used	**Dashboard > Quotas > Memory Request**	Hardware Resources|Cluster|Memory|Quota|Request Used (MB)

Memory Utilization

Metric Name	Description	UI Location	Metric Path
Limit (MB)	Limit of Memory which can be used by the pods. Only the pods belonging to monitored namespaces are used to calculate this metric.	**Dashboard > Utilization > Memory**	Hardware Resources|Cluster|Memory|Utilization|Limit (MB)
Request (MB)	MB value of Memory for which all the pods in monitored namespaces have requested.	**Dashboard > Utilization > Memory**	Hardware Resources|Cluster|Memory|Utilization|Request (MB)
Used (MB)	Actual Memory which the pods from monitored namespaces are currently using.	**Dashboard > Utilization > Memory**	Hardware Resources|Cluster|Memory|Utilization|Used (MB)

Nodes

Metric Name	Description	UI Location	Metric Path
Master Count	Number of master nodes in the cluster	**Inventory > Masters**	Hardware Resources|Cluster|Nodes|Master Count
Worker Count	Number of worker nodes in the cluster	**Inventory > Workers**	Hardware Resources|Cluster|Nodes|Worker Count
Memory Pressure Count	Number of nodes that are under memory pressure in the cluster	**Inventory > Memory Pressure**	Hardware Resources|Cluster|Nodes|Memory Pressure Count
Disk Pressure Count	Number of nodes that are under disk pressure in the cluster	**Inventory > Disk Pressure**	Hardware Resources|Cluster|Nodes|Disk Pressure Count

Pods

Pods Capacity

Metric Name	Description	UI Location	Metric Path
Total Count	Total number of pods that a cluster can support	**Pods > Total Count**	Hardware Resources|Cluster|Pods|Capacity|Total Count
Used Count	Number of pods already created in the cluster	**Pods > Count**	Hardware Resources|Cluster|Pods|Capacity|Used Count

PVC

PVC Quota

Metric Name	Description	UI Location	Metric Path
Used	PVC quota already being used in the cluster (count)	**Dashboard > Quotas > PVC**	Hardware Resources|Cluster|PVC|Quota|Used
Used %	Percentage of PVC quota already being used in the cluster	**Dashboard > Quotas > PVC**	Hardware Resources|Cluster|PVC|Quota|Used (%)

PVC Utilization

Metric Name	Description	UI Location	Metric Path
Capacity (MB)	Total PVC available for the pods in the monitored namespaces	**Dashboard > Utilization > PVCs**	Hardware Resources|Cluster|PVC|Utilization|Capacity (MB)

Request (MB)	Value for PVC requested by pods in monitored namespaces	**Dashboard > Utilization > PVCs**	Hardware Resources|Cluster|PVC|Utilization|Request (MB)

ReplicaSets

Metric Name	Description	UI Location	Metric Path
Count	Number of replica set resources in the cluster	**Inventory > Objects > ReplicaSets > Count**	Hardware Resources|Cluster|Count
Replicas	Total number of replicas for all the replica sets in the cluster	**Inventory > Objects > ReplicaSets > Count**	Hardware Resources|Cluster|ReplicaSets|Replicas
Replicas Available	Total number of available replicas for all the replica sets in the cluster	**Inventory > Objects > ReplicaSets > Available**	Hardware Resources|Cluster|ReplicaSets|Replicas Available
Replicas Unavailable	Total number of unavailable replicas for all the replica sets in the cluster	**Inventory > Objects > ReplicaSets > Unavailable**	Hardware Resources|Cluster|ReplicaSets|Replicas Unavailable

Services

Metric Name	Description	UI Location	Metric Path
Count	Total number of Kubernetes Services running in the cluster	**Inventory > Services > Services**	Hardware Resources|Cluster|Services|Count

StatefulSets

Metric Name	Description	UI Location	Metric Path
Count	Number of statefulsets in monitored namespaces	**Inventory > Objects > StatefulSets > (Count)**	Hardware Resources|Cluster|StatefulSets|Count
Replicas Ready	Number of replicas in a ready state across all statefulsets in monitored namespaces	**Inventory > Objects > StatefulSets > Replicas Ready**	Hardware Resources|Cluster|StatefulSets|Replicas Ready
Replicas Desired	Number of replicas across all statefulsets in monitored namespaces which are specified as desired in statefulset spec	N/A	Hardware Resources|Cluster|StatefulSets|Replicas Desired
Replicas Not Ready	Number of replicas across all statefulsets in monitored namespaces which are not ready and are yet to be created or started	**Inventory > Objects > StatefulSets > Replicas Not Ready**	Hardware Resources|Cluster|StatefulSets|Replicas Not Ready
Collisions	Number of hash collisions for statefulsets across all namespaces monitored	N/A	Hardware Resources|Cluster|StatefulSets|Collisions

Storage Quota

Metric Name	Description	UI Location	Metric Path
Used (MB)	Storage quota used by the cluster in MB	**Dashboard > Quotas > Storage**	Hardware Resources|Cluster|Storage|Quota|Used (MB)
Used (%)	Percentage of storage quota used by the cluster	**Dashboard > Quotas > Storage**	Hardware Resources|Cluster|Storage|Quota|Used (%)

Monitor Cluster Health

The **Cluster Dashboard** provides visibility into your cluster's health to quickly determine any impact on performance. Each dashboard indicator provides a different aspect of the cluster performance.

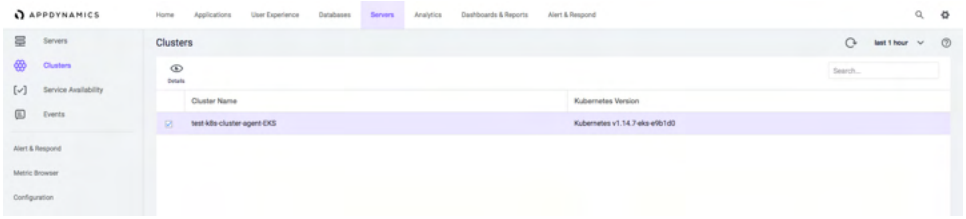
Important Notes

- The **Cluster Inventory Dashboard** may display 0 (zero) for Master Nodes; this is expected behavior. Unlike on-premises clusters, cloud providers develop and release at a different pace compared to the Kubernetes project and evolve independently. This master node, or a number of master nodes, in Amazon EKS, AKS, and other cloud-based environments or clusters, hide the master node of the cluster. The reported cluster results are similar to the results generated by the `kubectl get nodes` command. To verify your reported results, enter the `kubectl get nodes` command to review node information, and if master nodes are shown as zero.
- Cluster-level resource utilization metrics are the sum of resources consumed by each of the pods. These metric values may reach hundreds of thousand because the Cluster Agent reports sums for individual pods.
- Pod utilization metrics are the sum of each of the containers running within a pod.
- Pod count and pod state (running, pending, evicted, and failed pods) metrics show real-time values (not historical values averaged over a designated period of time). Whatever pods are running "now", are reported.

Clusters Dashboard

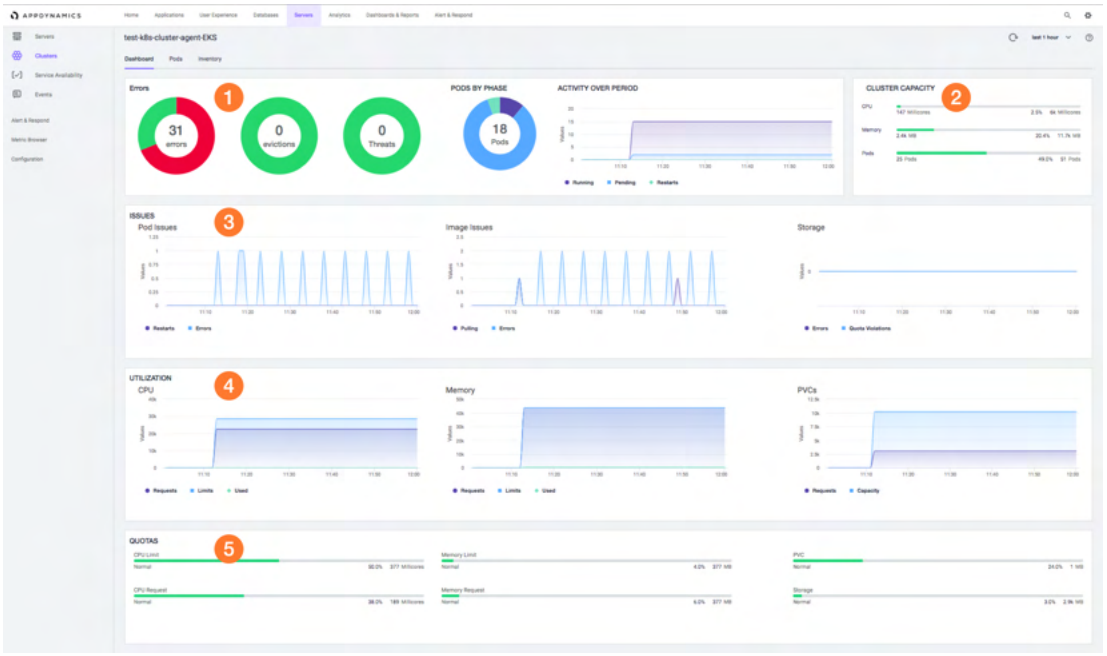
To access your cluster from the Controller:

1. Select **Servers > Clusters > Cluster Name**.



2. Select a Cluster Agent and double-click. The cluster interface displays the Dashboard, Pods, Inventory, and Events tabs.

Dashboard Tab



- **Errors card**: 1 Displays pie chart graphs for the monitored namespaces in each cluster:
  - **Errors**: Number of **Errors** (Error events count), **Evictions** (Evicted pods count), and **Threats** (Eviction threats count) for individual p ods.
  - **Pods by phase**: Number of pods that are in various states: **Failed**, **Pending**, **Running**, **Succeeded**, and **Unknown**.
  - **Activity over period**: Time-series chart that shows the number of pods in **Running**, and **Pending** states for a given time period.
- **Cluster Capacity**: 2 Score bar displays **CPU**, **Memory**, and **Pods**. A green line indicates the capacity usage. The Dashboard indicates the percent usage of **CPU**, **Memory** and **Pod** capacity of the cluster which you can use to plan the resource capacity for this cluster.

- **Issues card:** 3 Displays:
  - **Pod issues**: When a Cluster Agent observes pod restarts and errors.
  - **Image issues**: Image pulling and errors.
  - **Storage**: Storage capacity issues such as errors and quota violations.
- **Utilization card:** 4 Displays:
  - **CPU**: Requests, limits and used.
  - **Memory**: Requests, limits and used.
  - **PVCs**: Requests and capacity.
- **Quotas card**: 5 Displays % utilization of resources relative to the respective quotas. The Agent tracks these resources:
  - Percentage of CPU Limit Quota Used.
  - Memory Limit of Quota Used.
  - Percentage of PVC Quota Used.
  - Percentage of CPU Request Quota Used.
  - Percentage of Memory Request Quota Used.
  - Percentage of Storage Quota Used.

Numbers are cumulative for the entire cluster. Use these indicators to track the availability of specific resources based on the imposed quotas and use in cluster capacity planning.

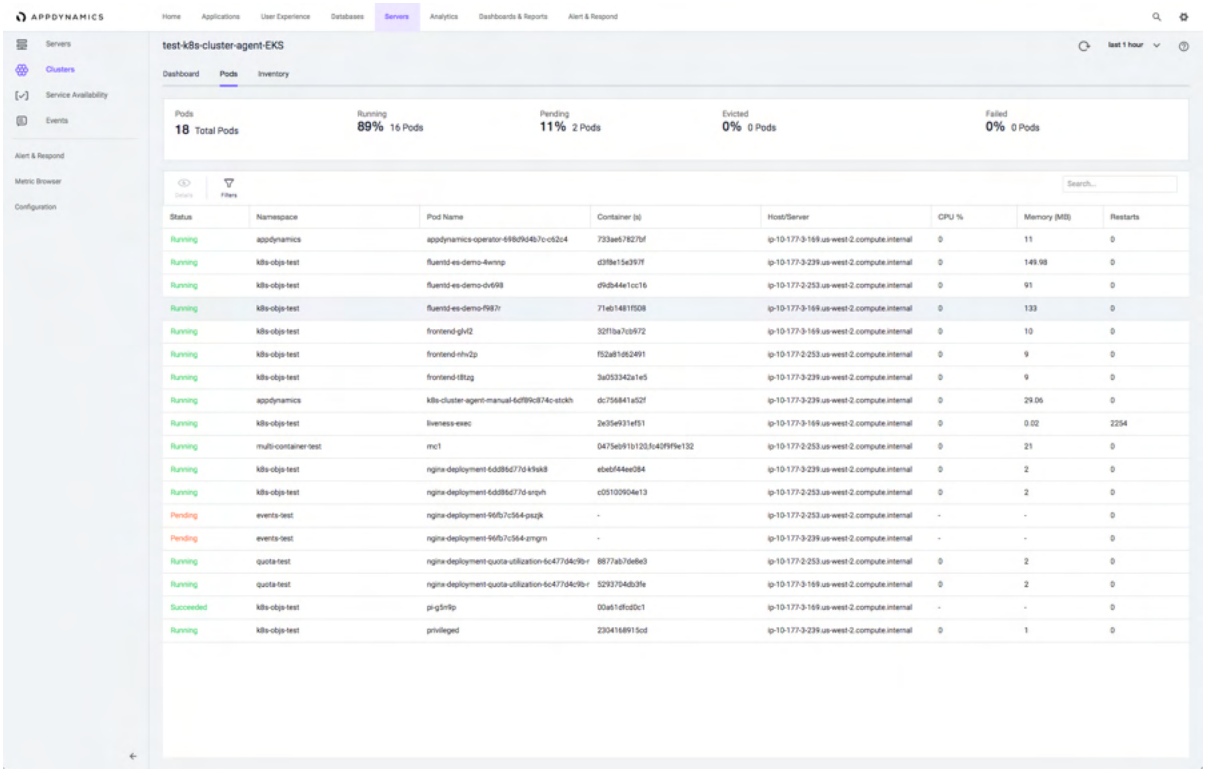
Pods Tab

The Pods tab displays pods in various states and shares a high-level summary of their status. This example shows pods running in Amazon EKS. All pods are displayed based on their registered **Namespace** and **Pod Name**.

> Terminated pods continue to display in the Pods list until they have been purged from the AppDynamics Controller, however their metrics are not updated. Purging occurs automatically at regular intervals. See Controller Settings for the Cluster Agent.

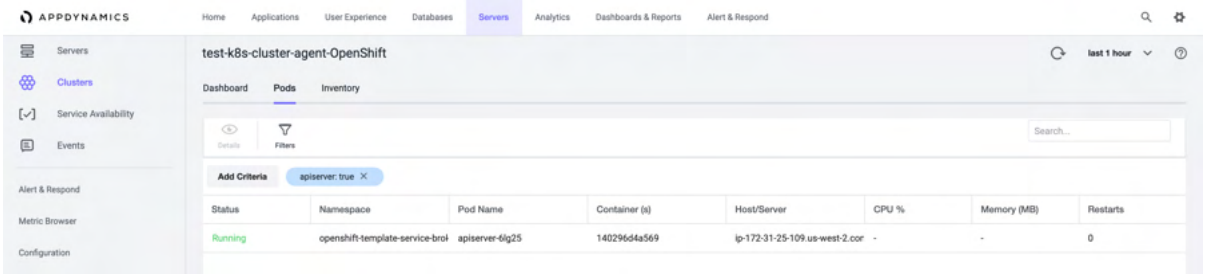
The top card displays a summary of the monitored pods and their status in each cluster:

- **Total Pods**: Total number of pods in the monitored cluster.
- **Running**: Percentage and number of pods in a running state.
- **Pending**: Percentage and number of pods in a pending state. Pending status normally indicates an issue. See Kubernetes documentation.
- **Evicted**: Percentage and number of evicted pods.
- **Failed**: Percentage and number of failed pods.

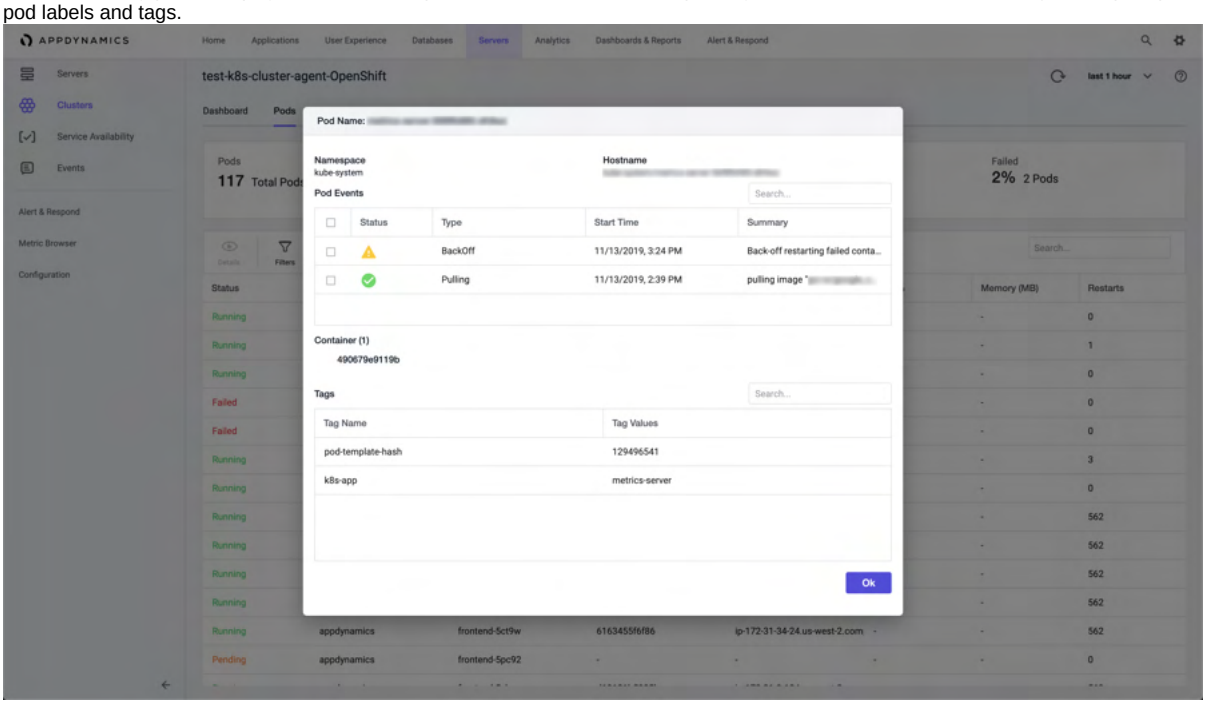


You can search based on **Namespace** or **Pod Name**.

You can further filter based on pod tags and labels:



Double-click any pod to display its **Pod Details** panel. From the **Pod Details** panel, you can review containers running in that pod, pod events, and pod labels and tags.



The Cluster Agent automatically detects a pod's:

- Status
- Namespace
- Pod name
- Container IDs
- Host or server
- CPU % (sum of running containers within the pod)
- Memory MB (sum of running containers within the pod)
- Restarts

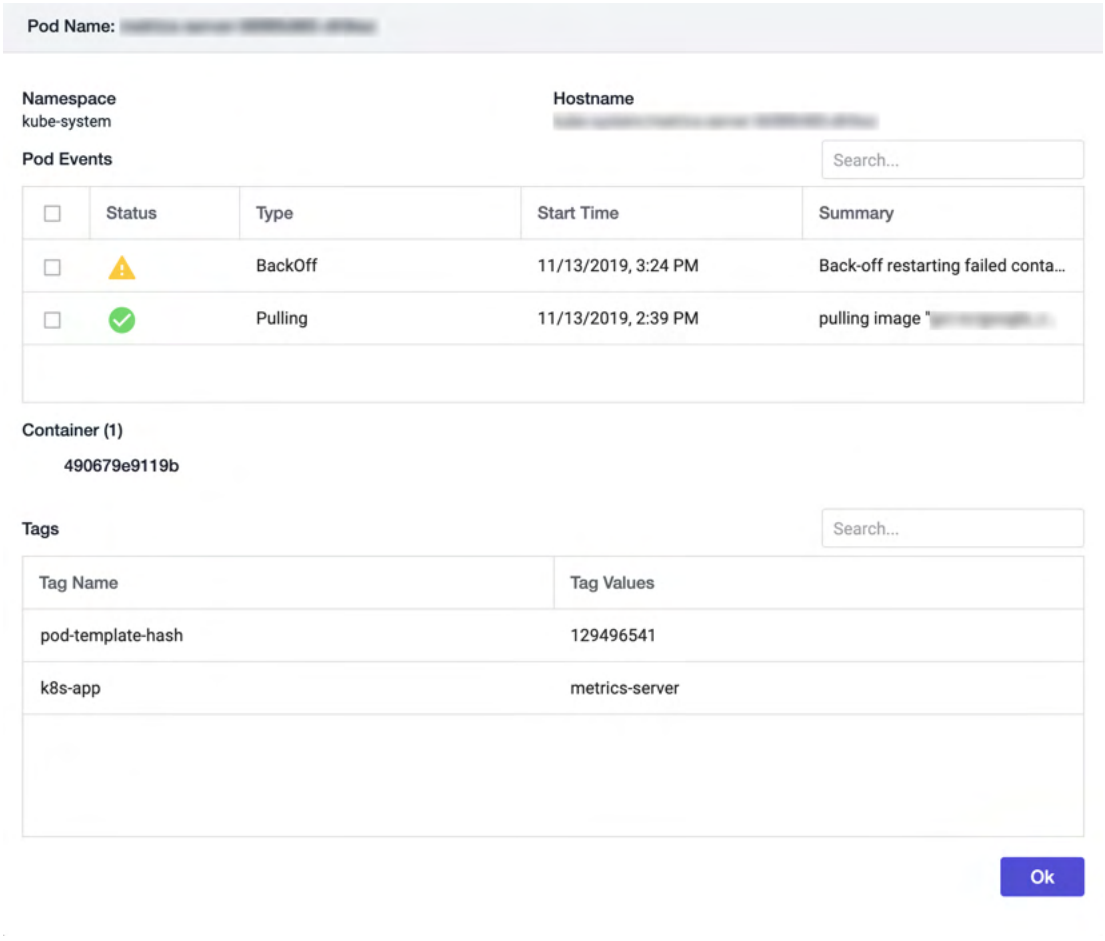
Pods are deleted for these reasons:

- When the namespace is un-monitored.
- When pods are deleted from the cluster, except for the failed pods.
- When the pod count exceeds the pod limit.
- If the pod is blocklisted.

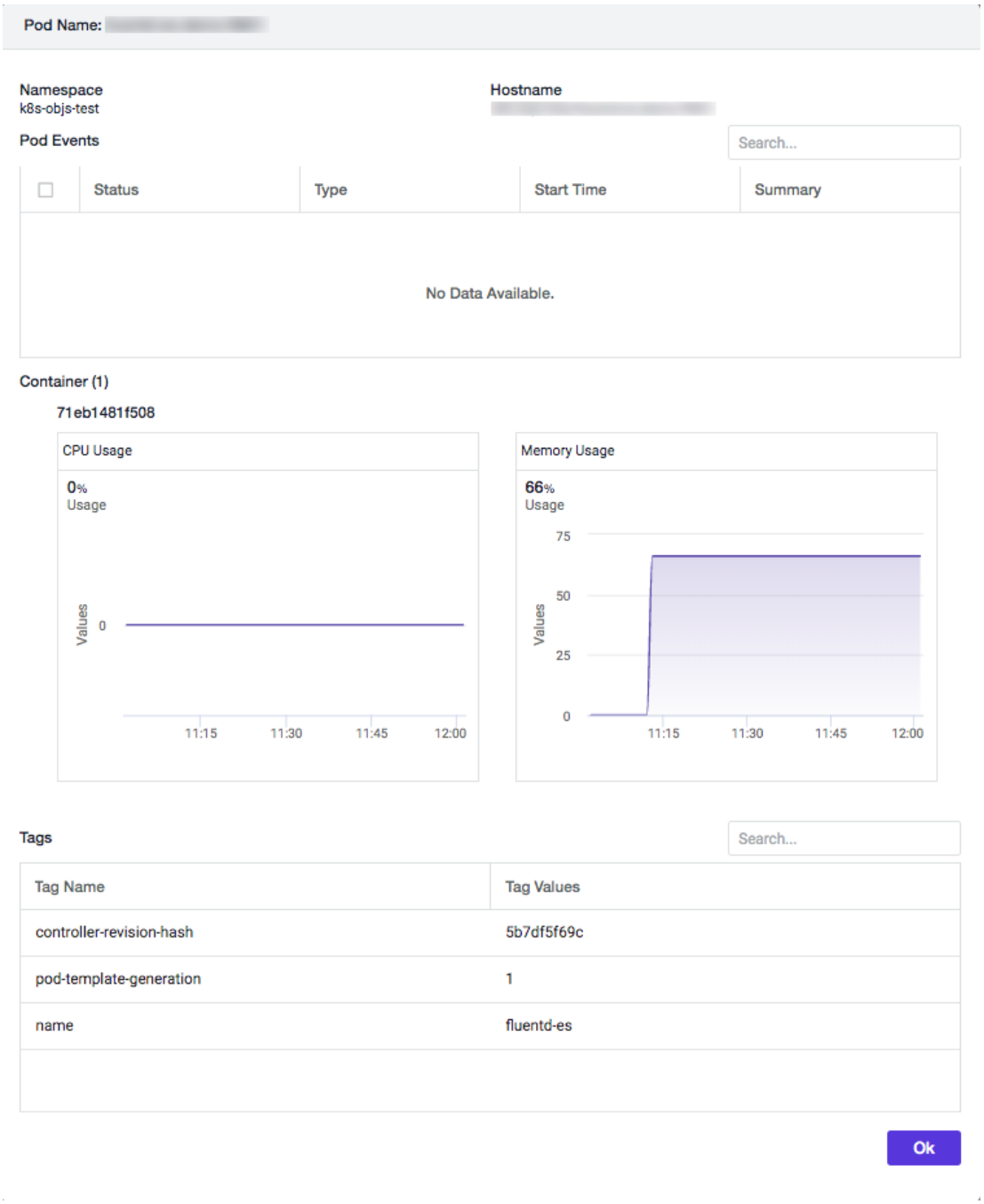
Pod Details Panel

The **Pod Details** panel displays:

- **Namespace**: In which namespace the pod is running.
- **Hostname**: Namespace or pod name.
- **Pod Events**: List of the most-recent events generated from the `kubectl describe pods` command.
- **Container** (count): List of running containers displayed by Container ID in this pod. Click each container to display individual container metrics.
- **Tags:** Kubernetes labels provided to this pod.

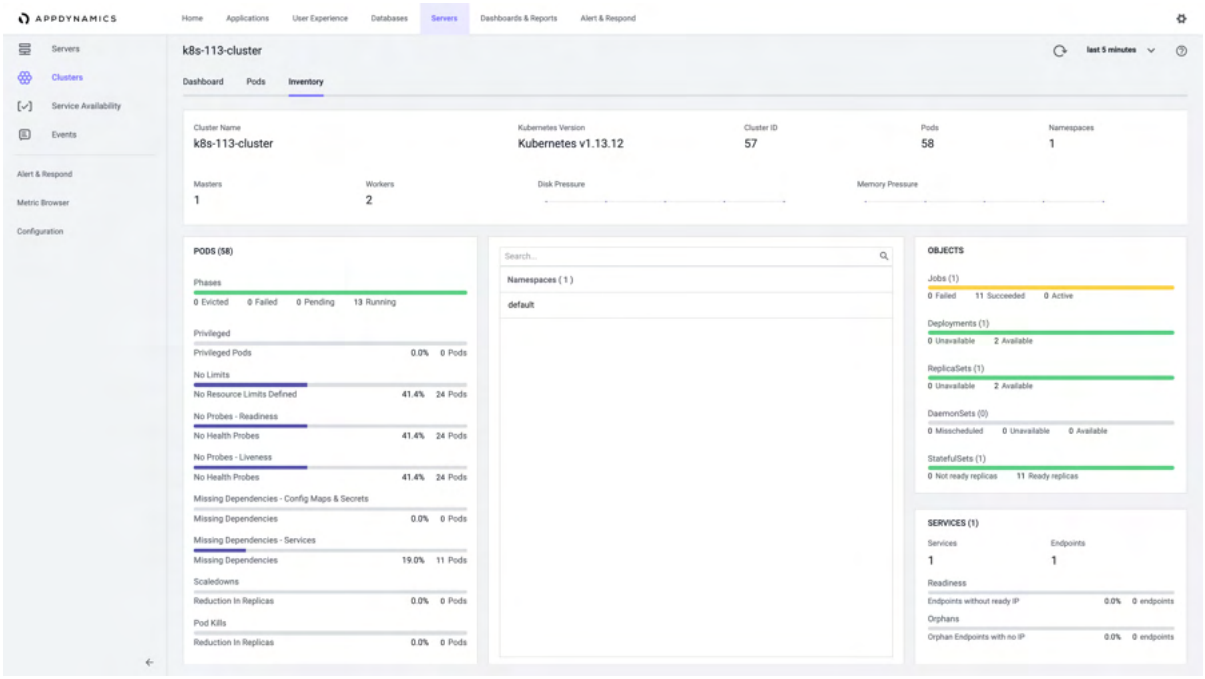


If you click the **Container ID**, it expands to display two container metrics: CPU and Memory Usage.



Inventory Tab

The Inventory tab displays a high-level snapshot or inventory view of your cluster. It displays the contents of the cluster which enables you to troubleshoot applications running in the cluster.



The Cluster card displays:

- **Cluster Name**: Name of the cluster.
- **Kubernetes Version**: Kubernetes version running on the cluster.
- **Cluster ID**: ID of the Cluster Agent.
- **Pods**: Number of monitored pods.
- **Namespaces**: Number of monitored namespaces.
- **Masters**: Number of master nodes.
- **Workers**: Number of worker nodes.
- **Disk Pressure**: Snippet of disk pressure trend in the cluster.
- **Memory Pressure**: Snippet of memory pressure trend in the cluster.

The Pods card displays:

- **Pods by Phases**: Whether a pod is running, evicted, pending or failed. This number is the same count as the pods in a "running" state in the Pods tab.
- **Privileged Pods**: The pods run as root.
- **No Limits**: You can specify limits to any pod that you are starting. The No Limits metric indicates how many pods do not have a limit defined.
- **No Readiness Probe**: If you configured a probe in Kubernetes to monitor readiness, the values display here.
- **No Liveness Probe**: If you configured a probe in Kubernetes to monitor liveness, the values display here.
- **Missing Dependencies - Config Maps & Secrets**: If a pod is dependent on any Config Maps & Secrets, then those dependencies are missing.
- **Missing Dependencies - Services**: If a pod is dependent on any Services, then those dependencies are missing.
- **Scaledowns**: Count of scaledowns of your deployments and replicasets.
- **Pod Kills**: Number of pods that were killed.

Namespaces: List of namespaces that you can search from the Search bar

The Objects card displays:

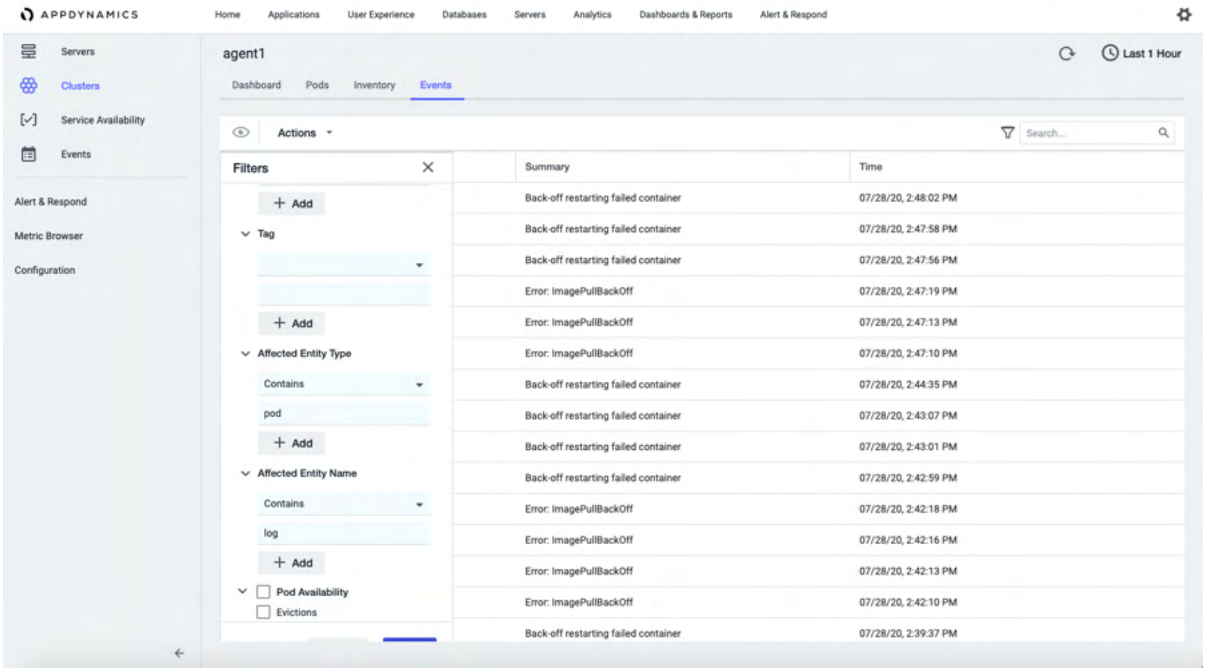
- **Jobs**: Total number of jobs in the monitored namespaces, and the number of failed, active, and succeeded job counts.
- **Deployments**: Number of deployments in monitored namespaces, and the total available and unavailable replica counts.
- **ReplicaSets**: Number of replicasets in monitored namespaces, and the total available and unavailable replica counts.
- **DaemonSets**: Number of daemonsets in monitored namespaces, and the total available, unavailable, and mis-scheduled pod counts.
- **StatefulSets**: Number of statefulsets in monitored namespaces, and the total ready and not ready replica counts.

The Services card displays the health of the entire cluster:

- **Services**: Total number of services being monitored.
- **Endpoints**: Total number of endpoints being monitored.
- **Readiness**: Number of endpoints without a ready IP address.
- **Orphans**: Number of orphaned endpoints with no IP address.

Events Tab

The Events tab displays the cluster events that are specific to the selected cluster. You can use **Filters** to view any specific event.


APPDYNAMICS
Home
Applications
User Experience
Databases
Servers
Analytics
Dashboards & Reports
Alert & Respond
Servers
Clusters
Service Availability
Events
Alert & Respond
Metric Browser
Configuration
agent1
Last 1 Hour
Dashboard
Pods
Inventory
Events
Actions
Search...
Filters
Add
Tag
Add
Affected Entity Type
Contains
pod
Add
Affected Entity Name
Contains
log
Add
Pod Availability
Evictions
Summary
Time
Back-off restarting failed container 07/28/20, 2:48:02 PM
Back-off restarting failed container 07/28/20, 2:47:58 PM
Back-off restarting failed container 07/28/20, 2:47:56 PM
Error: ImagePullBackOff 07/28/20, 2:47:19 PM
Error: ImagePullBackOff 07/28/20, 2:47:13 PM
Error: ImagePullBackOff 07/28/20, 2:47:10 PM
Back-off restarting failed container 07/28/20, 2:44:35 PM
Back-off restarting failed container 07/28/20, 2:43:07 PM
Back-off restarting failed container 07/28/20, 2:43:01 PM
Back-off restarting failed container 07/28/20, 2:42:59 PM
Error: ImagePullBackOff 07/28/20, 2:42:18 PM
Error: ImagePullBackOff 07/28/20, 2:42:16 PM
Error: ImagePullBackOff 07/28/20, 2:42:13 PM
Error: ImagePullBackOff 07/28/20, 2:42:10 PM
Back-off restarting failed container 07/28/20, 2:39:37 PM

Monitor Cluster Events

There are two event types in Kubernetes:

- Normal events
- Warning events

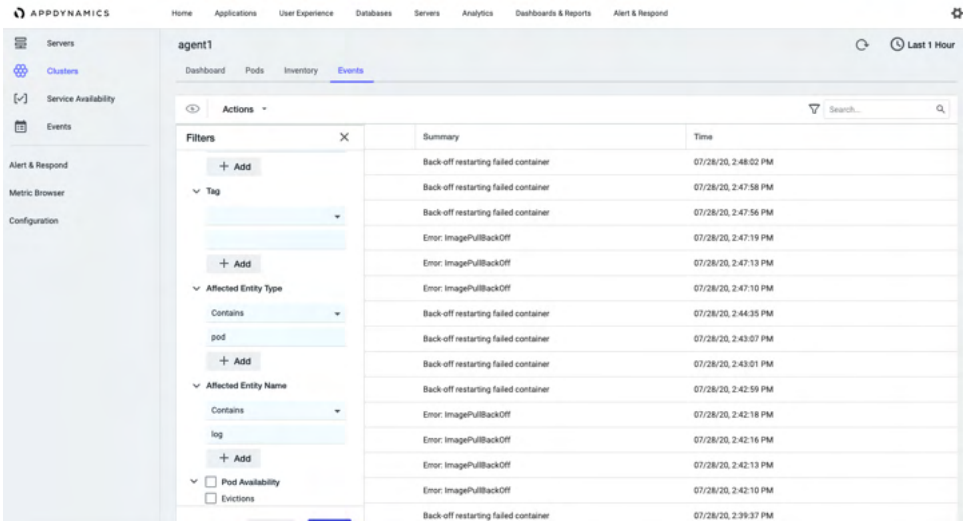
See Application Introspection and Debugging for more information on Kubernetes events. AppDynamics reports a subset of these as Cluster Events. They are comprised of Kubernetes warning events and important informational notices concerning state changes in the cluster. An example of a state change is a pod transitioning from a pending to a running state.

> This document contains links to Kubernetes documentation. AppDynamics makes no representation as to the accuracy of Kubernetes documentation because Kubernetes controls its own documentation.

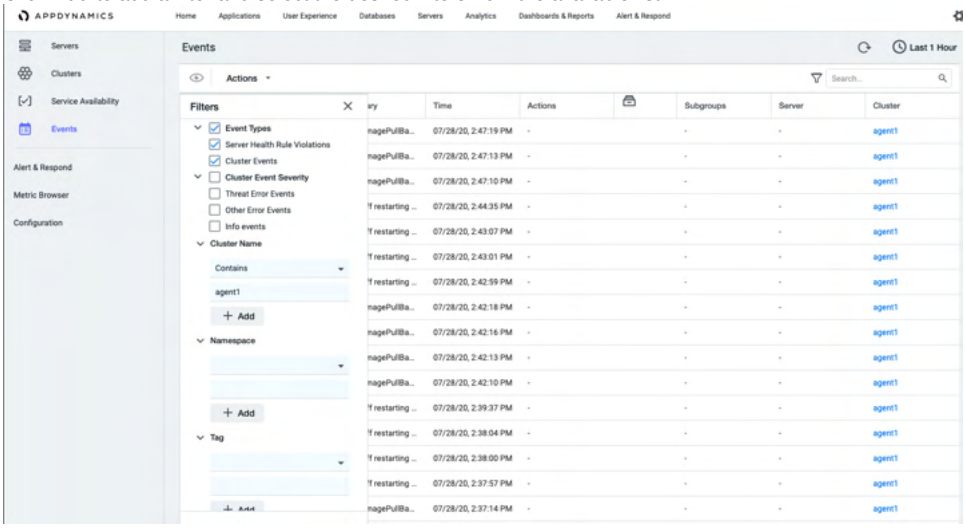
A **Cluster Event** of type warning displays a yellow warning icon. When a Pod, Replicaset, or Deployment gets added, updated, or deleted, otherwise known as a state change, the Cluster Agent reports it as an Info type event and displays a blue info icon.

To access Cluster Event data:

1. Click the **Servers** tab.
2. On the navigation bar, select **Events**. A panel displays a list of events that have been reported from both servers and clusters. To check for specific events, you can use the **Filters** option.

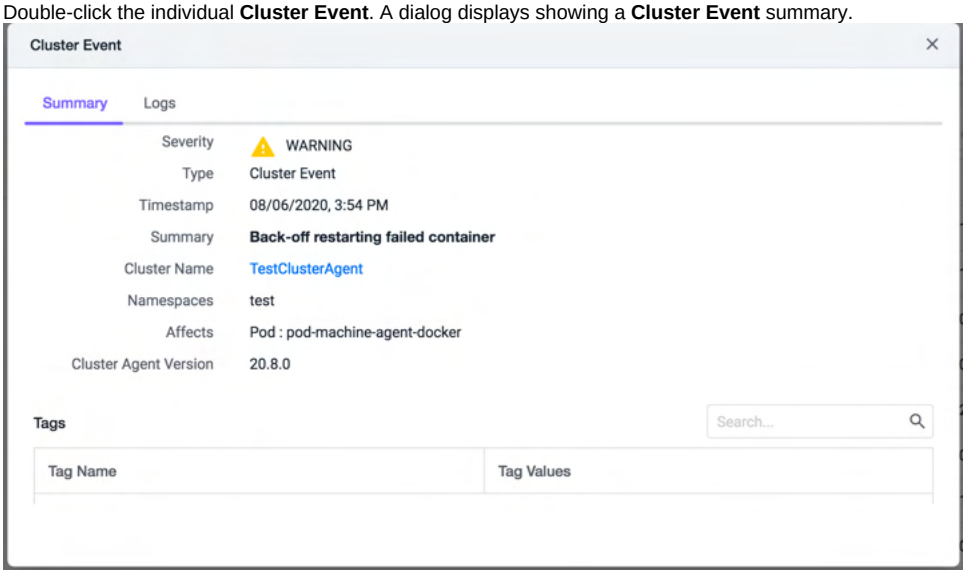


3. Click **Add** to add a filter and select the desired filters from the available list.



4. Click **Apply**. The list of events is filtered to show only the selected events.

5. Double-click the individual **Cluster Event**. A dialog displays showing a **Cluster Event** summary.



   The **Logs** tab displays the log details. See Enable Log Collection for Failing Pods.
6. Click **X** to close the dialog.

View Container Information

Related pages:

- Container Metrics
- Administer the Cluster Agent
- Hardware Resources Metrics

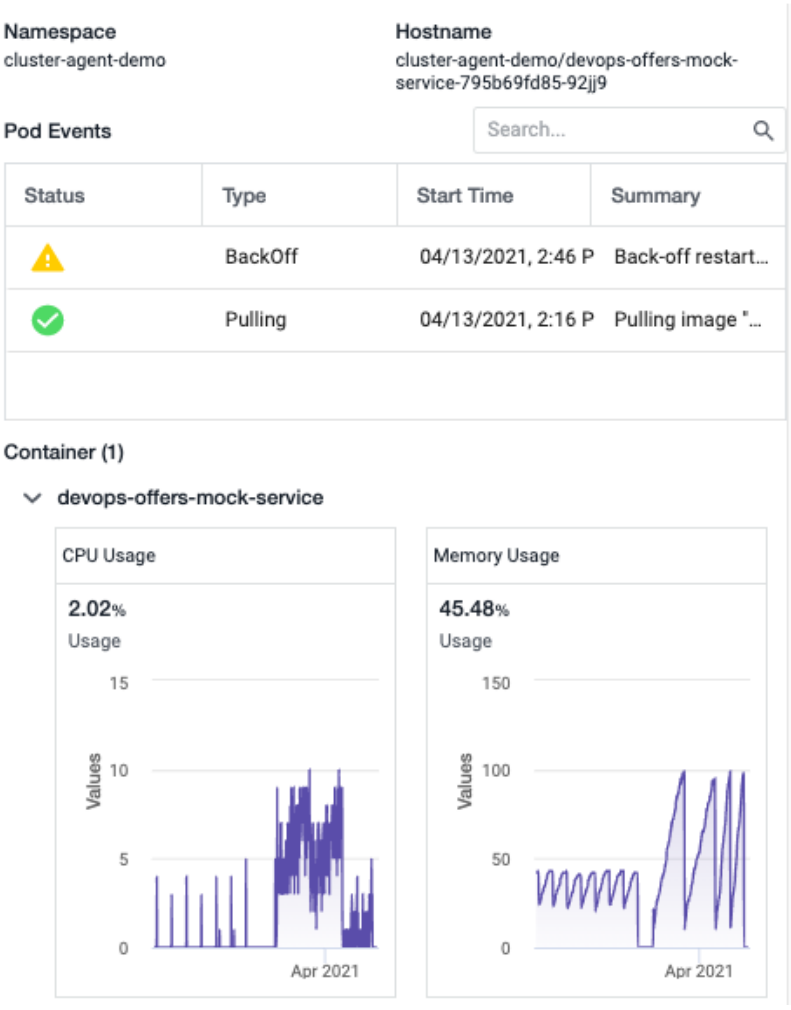
The AppDyamics Cluster Agent runs within your Kubernetes cluster. Metrics reported for containers are taken from the Kubernetes Metrics Server. Users need to install the metrics-server on their cluster to see container metrics. Once deployed, it discovers nodes in your cluster and starts collecting metrics from the Kubelet API. The Containers Dashboard shows the resource utilization metrics of your containerized applications.

The Containers page lists all containers to be monitored that are used by an application that is registered to the Controller.

View Container Metrics

To view Container metrics:

1. Navigate to **Servers** > **Clusters**.
2. Double-click a cluster to view the cluster agent **Dashboard**.
3. Navigate to the **Pods** tab.
4. Double-click a pod to view the pod details and the containers that are running in the pod.
   A list of Container names is displayed.
5. Click a **Container Name** to expand and view the metrics for that specific container.



Measure Resource Usage

Two metrics are reported per container:

- **CPU Usage**: Hardware Resources|CPU|%Busy
- **Memory Usage**: Hardware Resources|Memory|Used %

CPU

Name	Description
%Busy	Percentage of time the CPU was busy processing system or user requests from this container; this metric includes CPU Stolen time. This %Busy value can exceed 100%. For example, if a system has 16 cores the %Busy can go to 1600%.

Memory

Name	Description
Used %	Memory utilization by the container as a percentage of the memory hard limit.

Administer the Cluster Agent

You can use several settings in the Controller to administer the Cluster Agent.

Plan the Cluster Agent Deployment

If you have more than one Cluster Agent on your Controller, you must start your Cluster Agents in a staggered fashion. This procedure ensures that the Controller responds optimally. Use the table to determine the stagger start time for each cluster. Based on the cluster size, it may take a few minutes for the Cluster Agent to fully register with the Controller.

Plan capacity for Clusters

Capacity planning is important when preparing to deploy the Cluster Agent. The Cluster Agent and the Machine Agent share the capacity of the same Controller. The total number of pods and containers in your cluster is equal to the number of Machine Agents slots that are used. For example, if you have 250 pods and 250 containers, that is equivalent to using 500 Machine Agents in the Controller. See Controller Sizing.

Controller Settings for the Cluster Agent

These properties are recommended for a Controller supporting the Cluster Agent. Certain Server Visibility properties will apply to all instances of Server Visibility and the Cluster Agent.

Property	Description
`sim.cluster.agent.limit`	Maximum number of Cluster Agents per account. Configurable at the account level. For example, if you set the `sim.cluster.agent.limit` to a value of five, and you have ten Cluster Agents reporting, then the Controller randomly selects a subset of Clu Agents. These random Cluster Agents will not be registered and reported. If the configured limit is reached, any new Cluster Age will not be registered.
`sim.cluster.pod.limit`	Maximum number of pods per Cluster Agent per account. Configurable at the account level. If the number of pods exceeds the configured limit, the Cluster Agent randomly selects a set of pods that will not register or report.
`sim.machines.registrations.maxPerSecondPerAccount`	Maximum number of machine registrations allowed per account, per second. This property applies to all instances of Server Visib and the Cluster Agent.
`sim.cluster.container.limit`	Controller level limit for the number of containers per cluster. This is used if there is no account-level configuration. To overwrite a account-level, add an additional property to the corresponding account. This property only affects the Cluster Agent.
`sim.machines.offline.toStaleTimeoutMillis`	Length of time in milliseconds before an offline machine is considered stale, and eligible for purging. This property applies to all instances of Server Visibility and the Cluster Agent.
`sim.machines.registrations.update.frequency`	Machine's timestamp update frequency in milliseconds. This property applies to all instances of Server Visibility and the Cluster Agent.
`sim.cluster.events.maxPerMinute`	Controller level limit for the number of cluster events that are allowed per minute. This property only affects the Cluster Agent.
`sim.cluster.monitoring.enabled`	This property enables the Cluster Agent on the Controller. If set to false, all Cluster Agents stop reporting metrics and component but still register every minute. This property only affects the Cluster Agent.

See Controller Settings for Server Visibility.

To avoid the risk of purging live entities, ensure that the `sim.machines.offline.toStaleTimeoutMillis` value is always greater than the `sim.machines.registrations.update.frequency` value.

On-Premises Customers

For on-premises customers, to modify the Cluster Agent properties for your account:

1. Log in to the Administration Console with the administrator root password.
2. Select **Controller Settings**.
3. In the **Controller Settings** list, search for the setting.

4. In the **Value** field, enter the recommended value.
5. Click **Save**.
6. Log out of the Administrator Console.

SaaS Customers

For SaaS customers, create an AppDynamics Support ticket to have the Cluster Agent properties updated for your account.

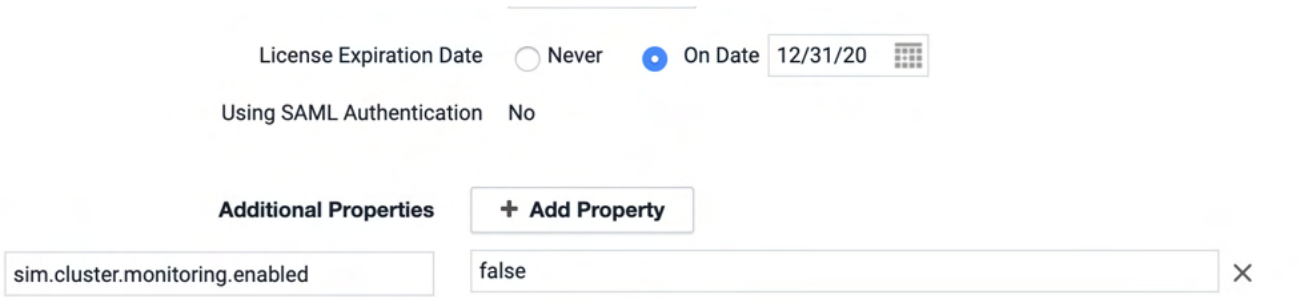
Control Cluster Agent Reporting

You can stop and resume the Cluster Agent from reporting metrics and registering containers and pods. You can do this on a per-account basis, or on a Controller-wide basis.

Stop a Cluster Agent by Account

To stop the Cluster Agent on the Controller:

1. Log in to the Administration Console with the administrator root password.
2. Select **Account Settings**.
3. In the accounts list, double-click the account for which you want to configure the Cluster Agent.
4. In the **Additional Properties** pane, click **+Add Property**.
5. In the left field, enter "sim.cluster.monitoring.enabled".
6. In the right field, enter "false".
7. Click **OK** to save the property.
8. Log out of the Administrator Console.



Start a Cluster Agent by Account

To start the Cluster Agent on the Controller:

1. Log in to the Administration Console with the administrator root password.
2. Select **Account Settings**.
3. In the accounts list, double-click the account for which you want to configure the Cluster Agent.
4. Locate the sim.cluster.monitoring.enable property, and in the right field, enter "true".
5. Click **OK** to save the property.
6. Log out of the Administrator Console.
7. If the Cluster Agent has been stopped using the "sim.cluster.monitoring.enabled" setting, it does not resume reporting metrics and registering pods and containers until you enable the Cluster Agent using the User Interface. See Use the Cluster Agent.

Stop all Agents on the Controller

To stop all Cluster Agents on the Controller:

1. Log in to the Administration Console with the administrator root password.
2. Select **Controller Settings**.
3. In the **Controller Settings** list, search for the "sim.cluster.monitoring.enabled" setting.
4. In the right field, set the value to "false".
5. Click **Save**.
6. Log out of the Administrator Console.

Start all Agents on the Controller

To start all Cluster Agents on the Controller:

1. Log in to the Administration Console with the administrator root password.
2. Select **Controller Settings**.
3. In the **Controller Settings** list, search for the "sim.cluster.monitoring.enabled" setting.

4. In the right field, set the value to "true".
5. Click **Save**.
6. Log out of the Administrator Console.
7. If Cluster Agents have been stopped using the "sim.cluster.monitoring.enabled" setting, they do not resume reporting metrics and registering pods and containers until they are enabled through the User Interface. See Use the Cluster Agent.

Enable Log Collection for Failing Pods

This page describes how to enable and use the Cluster Agent feature to collect logs for failing pods. This feature is automatically enabled on SaaS Controllers. You must enable on-premises Controllers described on this page.

When enabled, the Cluster Agent automatically collects logs for restarted or failed pods. The pod failure logs capture the status of the `CrashLoopBackoff` events that display the `Back-off restarting failed container` message. This message usually indicates that Kubernetes started a container but then the container quickly exits. If this occurs, Kubernetes attempts to restart the container. You can use these log messages to determine why the pod is exiting, whether due to a liveness probe, or some other issue.

Minimum Requirements

The Cluster Agent log collection feature requires:

- Cluster Agent Agent >= 20.7
- Controller >= 20.10.0

Requirements to Enable On-Premises Controllers

- Enable logging by using the Cluster Agent configuration setting, `sim.cluster.logs.capture.enabled`.
- Update the full file path to store the logs using the Controller setting, `sim.cluster.logs.root.file.path`. The logs are stored in the specified file location.
- Ensure you have 50 GB hard disk space on the file system.
- When updating the file path, ensure that you move previously collected logs from the old path to the new path to view them on the Controller UI.

Controller Settings

The following table lists the log settings required for an on-premises environment:

> For SaaS, the Technical Support team updates the configuration properties. Therefore, to configure the logs for SaaS, contact the support team by creating an AppDynamics Support ticket.

Setting	Definition	Location	Default
`sim.cluster.logs.bucket.name`	The folder name to store the pod logs.	Controller Administration Console with root user permission (**Controller Settings**)	`kubernetes-log-snippets`
`sim.cluster.logs.capture.enabled`	An option to enable or disable log capturing. This option is disabled by default.	Controller Administration Console with root user permission You can configure this setting at both account level (**Account Settings**) and Controller level (**Controller Settings**)	true
`sim.cluster.logs.expiration.in.days`	The number of days after which the logs get cleared from the storage.	Controller Administration Console with root user permission (**Controller Settings**)	`10`
`sim.cluster.logs.root.file.path`	The complete root file path to store the logs. Modifying the path results in storing the logs in the new path but the older logs will remain in the older location.	Controller Administration Console with root user permission (**Controller Settings**)	`/opt/appdynamics`
`sim.cluster.failed.pod.limit`	The number of historical pods beyond this value is purged.	Controller Administration Console with root user permission (**Controller Settings**)	`1000`

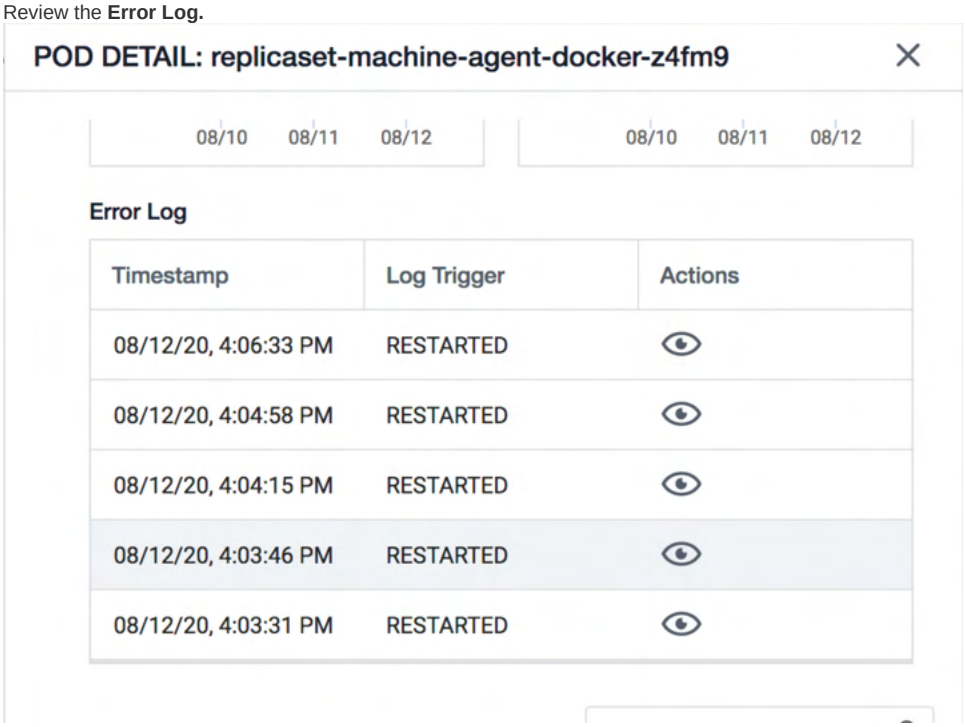
Use Cluster Agent Log Collection

You can view the logs on the **Pod Details** page and the **Cluster Event** page.

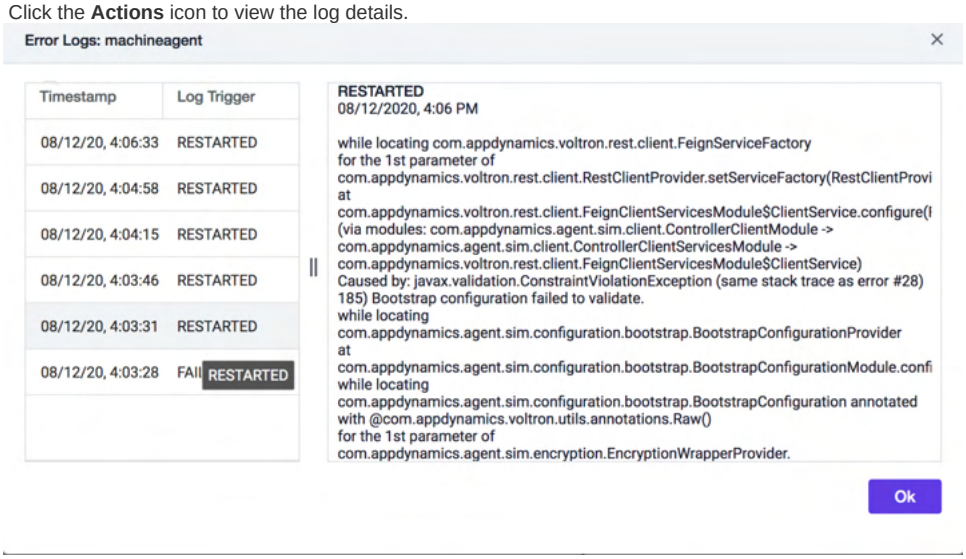
Pod Details Page

1. Select **Servers > Clusters > Cluster Name**.
2. Select a Cluster Agent and double-click. The cluster interface displays the **Dashboard**, the **Pods**, the **Inventory**, and the **Events** tabs.
3. Click **Pods**.
4. Double-click any pod to display its **Pod Details** panel.

5. Review the **Error Log.**



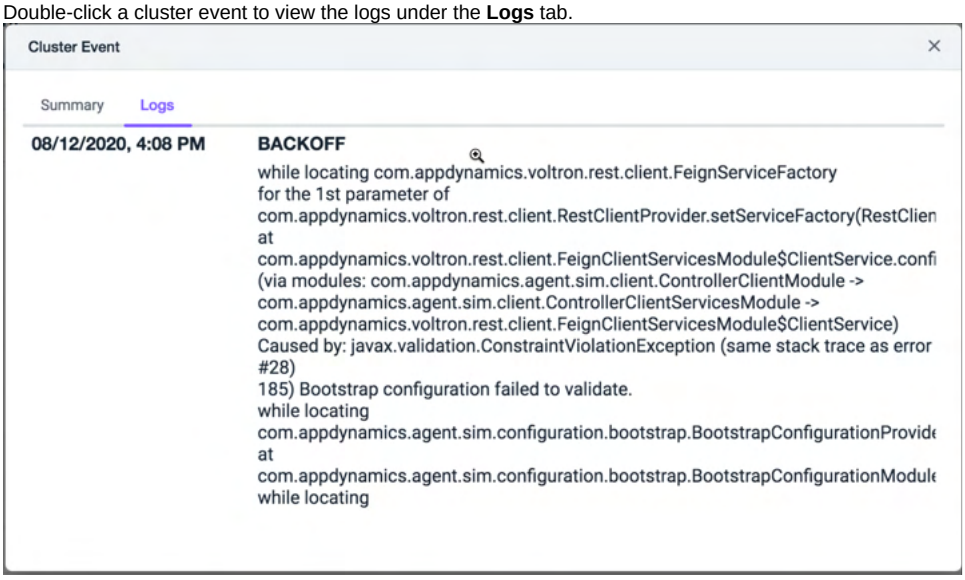
6. Click the **Actions** icon to view the log details.



Cluster Events Page

1. Click the **Servers** tab.
2. On the left navigation bar, select **Events** to see a list of events that have been reported from both servers and clusters.
   or
   Select **Clusters >** `<Clustername>` **> Details > Events**

3. Double-click a cluster event to view the logs under the **Logs** tab.



Upgrade the Cluster Agent

This page describes how to upgrade the Cluster Agent using:

- Kubernetes CLI, or
- Cluster Agent Helm Chart

See Install the Cluster Agent.

Upgrade the Cluster Agent Using Kubernetes CLI

1. Download the target version of the Cluster Agent bundle from the AppDynamics Downloads portal.
2. Unzip the bundle and `cd` to the unzipped folder:

```
unzip appdynamics-cluster-agent-alpine-linux-<version>.zip
cd appdynamics-cluster-agent-alpine-linux-<version>
```

3. Update the `cluster-agent.yaml` to include any previously applied configuration.
4. Update the image tag to add the version of the downloaded Cluster Agent bundle (for example, 20.12.1):

```
apiVersion: appdynamics.com/v1alpha1
kind: Clusteragent
metadata:
  name: k8s-cluster-agent
  namespace: appdynamics
spec:
  appName: "<app-name>"
  controllerUrl: "http://<appdynamics-controller-host>:8080"
  account: "<account-name>"
  # docker image info
  image: "docker.io/appdynamics/cluster-agent:20.12.1"
  serviceAccountName: appdynamics-cluster-agent
```

5. Delete and re-create the Operator and Cluster Agent.

```
kubectl delete -f cluster-agent.yaml
kubectl delete -f cluster-agent-operator.yaml
kubectl create -f cluster-agent-operator.yaml
kubectl create -f cluster-agent.yaml
```

Upgrade the Cluster Agent Using the Helm Chart

The latest Cluster Agent Helm Chart is available at `appdynamics-charts/cluster-agent`. Before you can upgrade to the latest Cluster Agent, you must first uninstall the existing Helm Chart and then re-install it.

1. Use `helm get` and `helm show` to determine if an upgrade is required to install the most recent Cluster Agent images from `appdynamics-charts/cluster-agent`:

```
# which images are currently installed by my release?
helm get all "<my-cluster-agent-helm-release>" -n appdynamics | grep -E 'agentTag|operatorTag'
  agentTag: 20.10.0
  operatorTag: 0.6.3

# which images are available to install?
helm show values appdynamics-charts/cluster-agent | grep -E 'agentTag|operatorTag'
  agentTag: 20.11.0
  operatorTag: 0.6.5
```

2. To perform the upgrade, uninstall and re-install the Cluster Agent Helm Chart using the same namespace from the previous installation:

```
helm uninstall "<my-cluster-agent-helm-release>" --namespace appdynamics
helm install -f ./ca1-values.yaml "<my-cluster-agent-helm-release>" appdynamics-charts/cluster-agent --
namespace=appdynamics
```

Uninstall the Cluster Agent

This page describes how to uninstall the Cluster Agent using:

- Kubernetes CLI, or
- Cluster Agent Helm Chart

See Install the Cluster Agent.

Uninstall the Cluster Agent Using Kubernetes CLI

Delete the Cluster Agent Operator and Cluster Agent using the YAML files:

```
kubectl delete -f cluster-agent.yaml
kubectl delete -f cluster-agent-operator.yaml
```

Uninstall the Cluster Agent Using the Helm Chart

Uninstall the Cluster Agent Helm Chart using the namespace that you used for the installation:

```
helm uninstall "<my-cluster-agent-helm-release>" --namespace appdynamics
```

Cloud Native Visualization

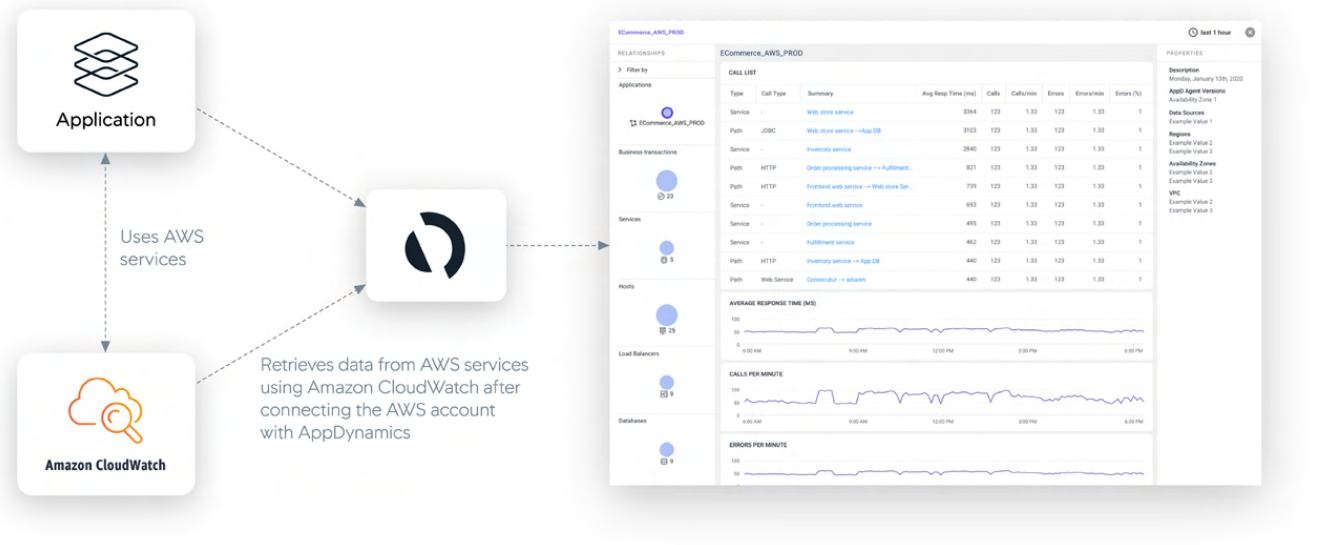
Deployment Support

SaaS

As part of the Cloud Native launch program, early adopters will have access to the Cloud Native Visualization functionality on a trial basis. AppDynamics reserves the right to charge separately for Cloud Native Visualization after the initial trial period.

Cloud Native Visualization introduces the AIOps platform to visualize the interactions and relationships between objects in your application infrastructure. This capability improves collaboration and unifies the efforts of different engineering functions by creating a real-time single source of truth for monitoring your application environment.

This illustration shows a high-level overview Cloud Native Visualization:



Additionally, Cloud Native Visualization also introduces an improved AppDynamics user interface that includes Relationships maps, Interactions maps, and related metrics. By correlating application metrics and interactions to cloud infrastructure in a unified view, AppDynamics Cloud Native Visualization enables you to:

- Automatically discover services associated with a cloud platform account and ingest the relevant cloud platform metrics and metadata.
- Navigate cloud platform services in a holistic application landscape that automatically correlates the Application Performance Monitoring (APM) data with cloud platform data.
- Surface insights on the impact of cloud provider services on an application's performance.
- Compare key performance metadata and visualize data flow based on the application entities and interactions.
- Interact with a set of data points on the time-series graph to capture specific values tracked by both AppDynamics APM and Amazon CloudWatch.
- Explore the dynamic, data-flow based application object page and interactions views containing application services.
- Drill down to the cloud infrastructure layer to understand how two respective topologies intersect in the context of application services.
- View your application infrastructure data, service, and business transactions in one application landscape. Determine what infrastructure exists and where it is located.

Get Started

Cloud Native Visualization is available to all existing AppDynamics SaaS customers. To get started, contact your AppDynamics account representative.

The AppDynamics User Interface and Cloud Native Visualization

AppDynamics introduces an intuitive user interface for Cloud Native Visualization that enables you to monitor your applications using a simplified approach.

You can now view entities and data, hosted by Amazon Web Services (AWS) in Cloud Native Visualization by connecting your Amazon account to AppDynamics. See Supported AWS Services.

The differences between the existing AppDynamics user interface and the redesigned user interface within Cloud Native Visualization are:

Data Model and Terminology

With Cloud Native Visualization, AppDynamics has introduced a new data model to replace the app, tier, and node hierarchy (see Tiers and Nodes). T he redesigned approach includes these concepts:

- Relationships maps
- Interactions maps
- Services (formerly tiers)
- Service Instances (formerly nodes)
- Object pages

> Traditional AppDynamics elements such as flow maps, dashboards, metrics, and health rules are still available to a Cloud Native Visualization user. However, to interact with these elements, you must return to the existing AppDynamics user interface. The Cloud Native entities do not display in the existing AppDynamics user interface.

Concepts

When Cloud Native Visualization is enabled, you access the new AppDynamics user interface from the existing Controller user interface through a pin . If a pin does not exist in your user interface, then you have not configured the Controller properly for this feature. See Monitor Cloud Native Applications.

Three main elements of the Cloud Native Visualization feature are:

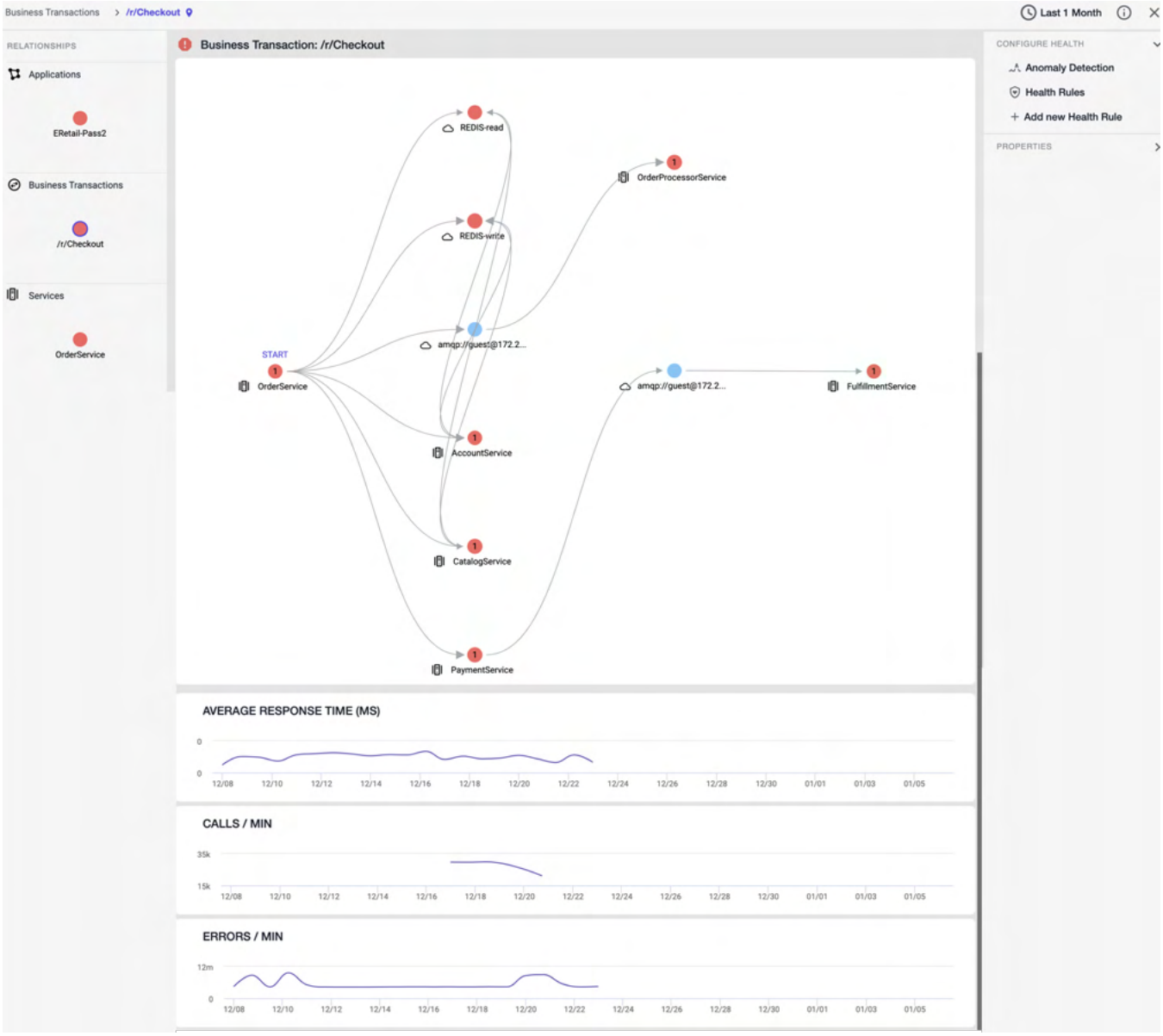
- Object page—Shows information in the context of an object or entity. This page is specific to the type of entity the user has selected, such as a service, business transaction, or database.
- Interactions map—Displays a visualization of how data is flowing in the system between different entities such as application services, databases, and load balancers where the application is the object in focus.
- Relationships map—Allows you to understand how Cloud Native entities are correlated to APM entities in one page. Additionally, it provides a view of the static topology of your hybrid APM and Cloud Platform application landscape.

Both the Interactions and Relationships maps enable you to click an entity and drill down to view more detailed information about the entity. See Cloud Native Visualization UI Overview.

Object Page

An Object page is a view that corresponds to an entity type, such as **Services**, **Applications**, or **Hosts**. Cloud Native Visualization models customer systems as a set of entities, relationships, interactions, and paths between those entities.

If an entity is unhealthy, Cloud Native Visualization depicts the entity in orange color or red color depending on the health state of the entity (`Warning` or `Critical`) as shown in this image:



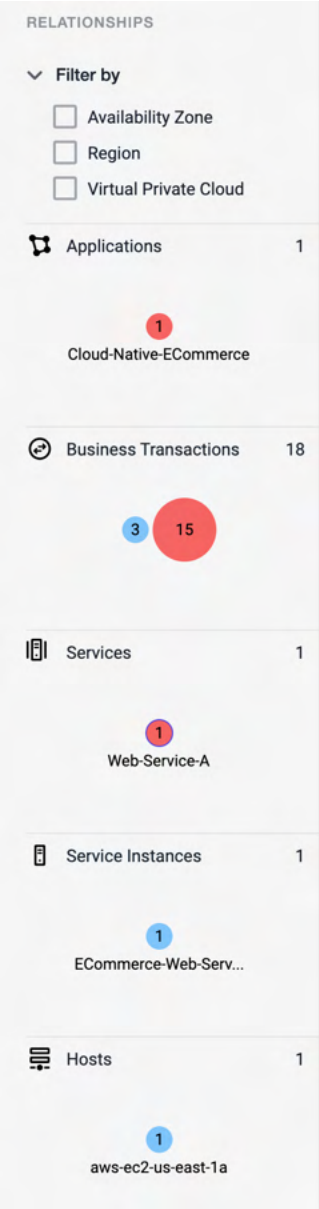
You can click the unhealthy entity to drill down and determine the affected business transactions, services, and service instances. You can view the associated health rules and modify them based on the performance data. In addition, you can use anomaly detection to automatically determine anomalies with the entities. This helps initiate remediation actions quickly. See Configure Entity Health.

Relationships Map

The Relationships map displays the relationships between an application and other service entities. The flexible visualization displays upstream and downstream connections to the type and number of entities and the health status of the objects in focus.

Objects are categorized as healthy or unhealthy using color-coded group circlets to depict health status. This allows you to quickly identify unhealthy entities and initiate remediation actions to resolve any critical issues. See Monitor the Health Status of Entities.

- Blue—indicates that the entities are healthy.
- Orange—indicates a health warning.
- Red—indicates that the entities have critical health issues.



Interactions Map

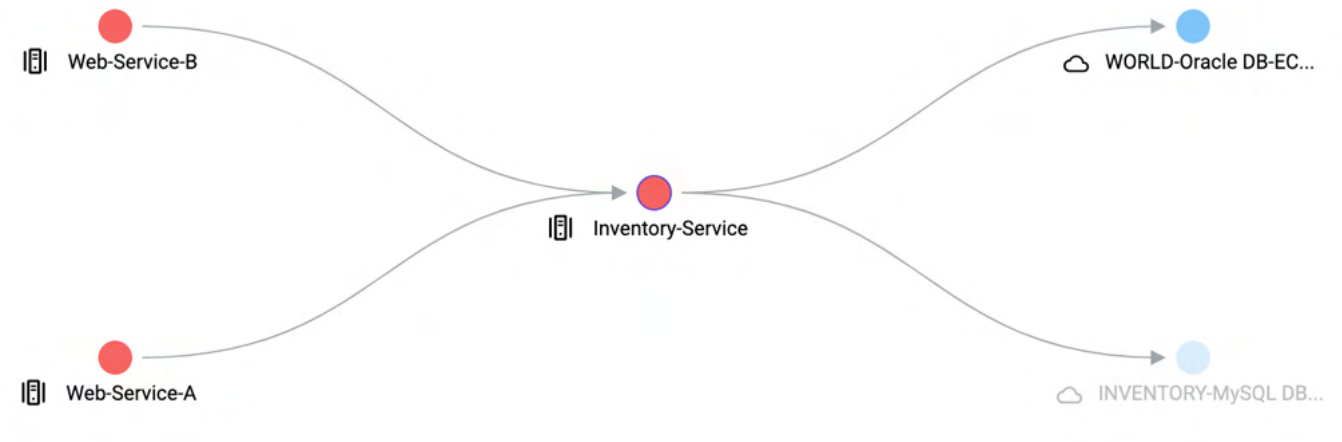
Interactions maps present a dynamic visual representation of entities and their interactions based on a specific context. You can use the **Filter by** optio ns in the Relationships map to refine the services that are part of the interaction in focus. The services that are not part of the filter display opaquely.

In this example, the **Inventory-Service** object has associated upstream and downstream objects with a **Virtual Private Cloud** filter selected in the Relationships map.

- A **Web-Service-B service** instance (unhealthy)
- A **Web-Service-A service** instance (unhealthy)

And downstream objects:

- A **WORLD-Oracle DB-ECOMM** instance
- An **INVENTORY-MySQL DB-ECOMM** instance that does not reside in the same VPC as the WORLD-Oracle DB



When an unhealthy entity is reported in the Relationships map or in the **List** view, you can drill down these entities and view details in the Interactions map to determine the affected business transactions, services or service instances. See Monitor the Health Status of Entities.

Supported AWS Services

Cloud Native Visualization supports these Amazon Web Services (AWS) services:

- Amazon Elastic Compute Cloud (EC2)
- Amazon Elastic Block Store (EBS)
- Amazon Relational Database Service (RDS) including the six types of database engines:
  - Amazon Aurora
  - PostgreSQL
  - MySQL
  - MariaDB
  - Oracle
  - Microsoft SQL Server
- Amazon Elastic Load Balancers (ELB) including three types of load balancers:
  - Classic Load Balancers (CLB)
  - Network Load Balancers (NLB)
  - Application Load Balancers (ALB)
- Virtual Private Cloud (VPC)

> Additional AWS services and support for other cloud platforms will be added in future releases.

Metrics in Cloud Native Visualization

For information about metrics, see Cloud Native Considerations on the Cloud Native Visualization Metrics Overview page.

AWS CloudWatch Specific API Rate Limits per Service

Cloud Native Visualization uses the Amazon CloudWatch APIs to provide data. AWS imposes hard rate limits on many of the AWS Service APIs. Connecting your AWS Account to AppDynamics may cause an increase in the usage and billing of these specific CloudWatch Service APIs. These hard rate limits also impact how quickly you reach your rate limit. These rate limits often display as a `503` error: `Request limit exceeded`. See API Request Throttling in Amazon's documentation.

Cost

AppDynamics makes metadata requests every fifteen (15) minutes and metric requests every five (5) minutes. The number of metrics is dependent on the instance type. See Monitoring your usage and costs in Amazon's documentation.

Cloud Native Visualization Requirements

As part of the Cloud Native launch program, early adopters will have access to the Cloud Native Visualization functionality on a trial basis. AppDynamics reserves the right to charge separately for Cloud Native Visualization after the initial trial period.

Cloud Native Visualization simplifies the integration of service data from cloud platform providers. By visualizing relationships and dependencies, this feature helps you find the cause of performance issues in your applications or the related cloud platform services.

Cloud Native Visualization requires configuration performed by AppDynamics.

Before You Begin

Before you start, you must consider these system requirements and permissions.

AppDynamics Components

Cloud Native Visualization requires:

- SaaS Controller >= 20.10.0.
- Java Agent >= 20.9.0, .NET, and Node.js Agents >= 20.10.0 are required for Amazon EC2 to APM node correlation.
- Sufficient APM licenses to use the Java, .NET, or Node.js Agents.
- For the best experience, Java Agents should be configured to use `service-instance` (node) name reuse. See Use Node Name Reuse for Java Agents in Cloud Native Visualization. For details regarding Java Agent version compatibility, see Java Supported Environments.

You need permission to view at least one application, which can be accomplished by giving permission to view all applications or specific applications. See Configure Cloud Native Visualization Permissions and Create and Manage Custom Roles to set up View Infrastructure Entities account-level permission.

AWS Instance Metadata Permissions

If you are using an AppDynamics agent, the Java, .NET, or Node.js agent must have access to the AWS instance metadata. Some Amazon EC2 instances have metadata access restrictions. If access is restricted, you must configure firewall pass-through rules to allow metadata access for the application instance where the agent is running. See Retrieving instance metadata in Amazon's documentation.

Set Up Cloud Native Visualization

You must connect AWS services to AppDynamics and set up Cloud Native Visualization permissions. See Set Up Cloud Native Visualization.

This document contains links to Amazon Web Services (AWS) documentation. Amazon manages all linked Amazon documentation, therefore AppDynamics makes no representation as to the accuracy of such documentation.

Next Steps

Set Up Cloud Native Visualization.

Set Up Cloud Native Visualization

As part of the Cloud Native launch program, early adopters will have access to the Cloud Native Visualization functionality on a trial basis. AppDynamics reserves the right to charge separately for Cloud Native Visualization after the initial trial period.

This page describes how to set up and use AppDynamics Cloud Native Visualization. Make sure that you:

1. Review the Cloud Native Visualization Requirements
2. Connect Amazon Web Services to AppDynamics
3. Configure Cloud Native Visualization Permissions

Cloud Native Visualization requires configuration performed by AppDynamics. The configurations have been rolled out gradually to SaaS Controllers since October 28, 2020.

Connect Amazon Web Services to AppDynamics

As part of the Cloud Native launch program, early adopters will have access to the Cloud Native Visualization functionality on a trial basis. AppDynamics reserves the right to charge separately for Cloud Native Visualization after the initial trial period.

This page explains how to connect your AppDynamics account with Amazon Web Services (AWS) using Amazon CloudWatch to display Cloud Native Visualization. Cloud Native Visualization is available in SaaS Controller >= 20.10.0, see Cloud Native Visualization Requirements.

This page contains links to Amazon Web Services (AWS) documentation. AppDynamics makes no representation as to the accuracy of Amazon documentation because Amazon controls its own documentation.

AppDynamics supports connecting multiple AWS accounts using either of these methods:

- Use Role Delegation - Role delegation grants additional permissions to a new or existing role
- Use Access Key Credentials - Access key credentials to assign a user a new pair of access keys

Set Up Role Delegation

To set up role delegation, you must configure settings on your Controller and AWS account:

1. Create a Role Name and Copy External ID in the Controller
2. Create an AWS IAM Service Policy
3. Create an AWS IAM Role
4. Connect your AWS IAM Service Role to AppDynamics

Create a Role Name and Copy External ID in the Controller

To connect your AppDynamics Controller:

1. Go to https://accounts.appdynamics.com/subscriptions.
2. Sign in to your account.
3. Select the Controller that you want to connect to Amazon CloudWatch.
4. Navigate to **Home > Cloud Platform**.
5. Under **Cloud Platform**, select **Amazon CloudWatch**.
6. Select **Role Delegation**.
7. Enter your Amazon credentials:
    a. Enter a **Connection Name**, for example: **AppDynamicsMonitoring**.
    b. Enter your organization's **AWS Account ID** without the dashes.
    c. Enter an **AWS Role Name**. For example: **AppDynamicsMonitoringRole**.

        The AWS Role Name cannot contain spaces or special characters.

    d. Under **External ID**, select **Copy**.

        Make sure you save this External ID because you will need to enter it later in your AWS Identity and Access Management (IAM) console. If you close this window without capturing the External ID, you will have to generate a new ID for your **App DynamicsMonitoring** account.

8. Leave this browser window open. Open another separate browser window, and log in to your AWS Account.

Create an AWS IAM Service Policy

In the AWS IAM Management Console, you can attach permissions policies using either JSON or the AWS UI.

It is recommended to use JSON because it contains the minimum required permissions.

If you prefer to attach permissions policies manually, you can use the AWS UI. See Creating IAM Policies - AWS Identity and Access Management.

Attach Permissions Policies Using JSON

Navigate to the AWS Management Console and open the IAM Console.

1. Select **Create policy**.
2. Select **JSON**.
3. Copy this code:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "elasticloadbalancing:DescribeLoadBalancers",
                "ec2:DescribeInstances",
                "cloudwatch:GetMetricData",
                "ec2:DescribeVpcs",
                "ec2:DescribeRegions",
                "ec2:DescribeVolumes",
                "elasticloadbalancing:DescribeTargetHealth",
                "rds:DescribeDBInstances",
                "elasticloadbalancing:DescribeTargetGroups",
                "ec2:DescribeSubnets",
                "cloudwatch:ListMetrics",
                "rds:DescribeDBClusters"
            ],
            "Resource": "*"
        }
    ]
}
```

4. Paste the code under the JSON tab.
5. Select **Review policy**.
6. Enter a name for the policy, such as **AppDynamicsMonitoringPolicy**, and add a description (optional).
7. Click **Create policy**. A message displays: **AppDynamicsMonitoringPolicy** has been created.

Create an AWS IAM Role

1. In the Identity and Access Management (IAM) console left navigation pane, select **Roles**.
2. Select **Create role**.
3. Select **Another AWS account**.
4. For **Account ID**, enter AppDynamics Account ID: **932519523259**.
5. For **Options**, select **Require external ID**.
    a. Navigate to your AppDynamics Controller browser panel.
    b. Copy the **External ID**, if you have not already done so.
6. Return to the AWS Management Console browser panel.
7. Paste or enter the **External ID** generated by the AppDynamics Controller, and leave the **Require MFA** option disabled.
8. Select **Next: Permissions**.
9. Under **Attach permissions policies**, search for and select **AppDynamicsMonitoringPolicy**.
10. Select **Next: Tags** (optional).
11. Select **Next: Review**.
12. Enter a name for the role, such as **AppDynamicsMonitoringRole**, and add a description (optional).
13. Select **Create role**. A message displays: **The role AppDynamicsMonitoring role has been created**.

Connect Your AWS IAM Service Role to AppDynamics

1. Return to the Controller browser panel. The AppDynamicsMonitoring connection uses the **AppDynamicsMonitoringRole** to connect.
2. Ensure that the AWS role name in the Controller matches the role name with the permissions policy created in the AWS IAM Management Console. In the example, the name used is **AppDynamicsMonitoringRole.**
3. Select **Connect** to initiate the AppDynamics data connection with Amazon CloudWatch.
4. The **Home > Cloud Platform > Integration Status** displays the connection status of your Amazon CloudWatch integration.
5. Before navigating to the Network Dashboard, confirm that the integration status of your AWS account displays **Connected**.

Use Access Key Credentials

To integrate AppDynamics with Amazon CloudWatch:

1. Create an AWS IAM User within your AWS account.
2. Connect your AWS IAM User to AppDynamics and attach read-only policies to limit the permissions granted to the user.

Create an AWS IAM User

1. Navigate and sign in to the AWS Management Console.
2. Open the IAM Console.
3. In the Identity and Access Management (IAM) left navigation pane, select **Users**.
4. Select **Add user**. If you are unable to add a user, see Access Management.
5. Enter the **User name:** `appD_monitoring_user`, or a user name of your choice.
6. Under **Select AWS access type**, select **Programmatic access**.
7. Select **Next: Permissions**.
8. Under **Grant permissions**, select **Attach existing policies directly**.
9. Select appropriate policies for the `appD_monitoring_user` monitoring account. AppDynamics recommends `ReadOnlyAccess` policies. Additionally, you can select custom policies specific to the active resource that you want to ingest Amazon CloudWatch metrics. See Access Management and Example IAM Identity-Based Policies.
   For example, these polices are selected:
   - `ElasticLoadBalancingReadOnly`
   - `AmazonRDSReadOnlyAccess`
   - `AmazonEC2ReadOnlyAccess`
   - `CloudWatchReadOnlyAccess`

   You can optionally set the permissions boundary.
10. Select **Next: Tags** (optional).
11. Select **Next: Review**.
12. Select **Create user**.
13. Note the **Access Key ID** and the **Secret Access Key** for the user, or click **Download .csv**.

You will need the keys to add Amazon CloudWatch to your AppDynamics account. If you navigate away from this panel without capturing these keys, you cannot access them again. You will have to create a new **`appD_monitoring_user`** monitoring account. See Creating IAM Users (Console).

Connect Your AWS IAM User Account to AppDynamics

To connect your AWS IAM user account to AppDynamics:

1. Go to https://accounts.appdynamics.com/subscriptions.
2. Sign in to your account.
3. Select the Controller that you want to connect to Amazon CloudWatch.
4. Navigate to **Home > Cloud Platform**.
5. Under **Connect a Cloud Platform**, select **Amazon CloudWatch**.
6. Select **Access Key Credentials**.
7. Enter a Connection Name.
8. Enter your Amazon **appD_monitoring_user** Access Key ID.
9. Enter your Amazon **appD_monitoring_user** Secret Access Key.
10. Select **Connect** to initiate the AppDynamics data connection with Amazon CloudWatch.
11. Before navigating to the Cloud Native Visualization user interface, select **Home > Cloud Platform > Integration Status** to confirm that the integration status of your AWS account displays **Connected**.

After you finish enabling AWS, you are redirected to the Applications tab to review the information.

Next Steps

- (Optional) Administer Cloud Native Visualization
- Configure Cloud Native Visualization Permissions

Administer Cloud Native Visualization

As part of the Cloud Native launch program, early adopters will have access to the Cloud Native Visualization functionality on a trial basis. AppDynamics reserves the right to charge separately for Cloud Native Visualization after the initial trial period.

This page describes how to edit and delete your AppDynamics account with Amazon Web Services (AWS). If you modify these settings, Cloud Native Visualization may not correctly display all Amazon CloudWatch API entities and metrics.

Edit an Amazon CloudWatch Connection

To edit an existing connection, an account administrator must perform these steps:

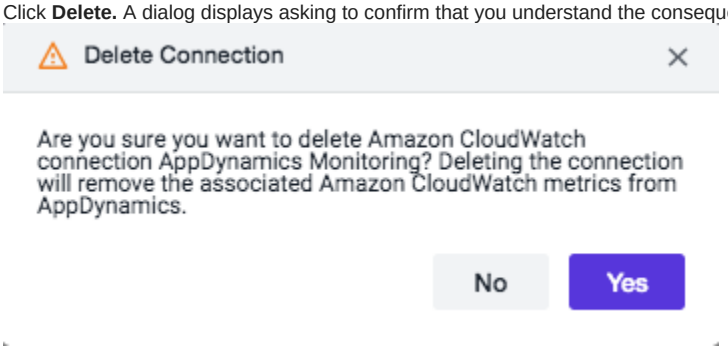
1. Go to https://accounts.appdynamics.com/subscriptions.
2. Sign in to your account.
3. Select the Controller that you want to connect to Amazon CloudWatch.
4. Navigate to **Home > Cloud Platform** to display a list of connections.
5. Select the connection name to edit.
6. Click .
7. Click **Connect**.

Confirm that the **Integration Status** of your Amazon account shows success before navigating to Cloud Native Visualization. To verify metrics are flowing in your AppDynamics account, navigate to the desired Object page.

Delete an AWS CloudWatch Connection

To delete an existing connection, an account administrator must perform these steps:

1. Go to https://accounts.appdynamics.com/subscriptions.
2. Sign in to your account.
3. Select the Controller that you want to connect to Amazon CloudWatch.
4. Navigate to the **Home > Cloud Platform** tab to display a list of connections.
5. Select the connection name to delete.
6. Click **Delete.** A dialog displays asking to confirm that you understand the consequences of deleting the connection.



7. Click **Yes**. The Amazon account is no longer monitored.

Next Steps

Configure Cloud Native Visualization Permissions

Configure Cloud Native Visualization Permissions

As part of the Cloud Native launch program, early adopters will have access to the Cloud Native Visualization functionality on a trial basis. AppDynamics reserves the right to charge separately for Cloud Native Visualization after the initial trial period.

This page describes how to configure Role-Based Access Control (RBAC) to provide administrative user permissions for configuring Cloud Native Visualization. You can also grant non-administrative users permission to use or view Cloud Native Visualization.

To use or view Cloud Native Visualization:

1. Set up a **View Infrastructure Entities** account-level permission.
2. Set permission to view all applications or specific applications, see Cloud Native Visualization Requirements and Create and Manage Custom Roles.

To configure access and roles, you must either be the AppDynamics Account owner or have Administrator permissions.

Set up View Infrastructure Entities Permission

To use Cloud Native Visualization, you need **View Infrastructure Entities** account-level permission and at least one of the following:

- Any default role
- Any application-level permission
- Any dashboard permission

View Infrastructure Entities permission is not added to any default role. See Create and Manage Custom Roles.

You must set up **View Infrastructure Entities** permissions; otherwise databases and load balancers will not appear correctly in Cloud Native Visualization.

Create and Assign Role-Based Access to Cloud Native Visualization

Step 1: Create a Role to Access Cloud Native Visualization

To enable Cloud Native Visualization on a Controller:

1. Log in to the AppDynamics Controller as a user with Administrator permissions.
2. Click  .
3. Select **Administration**.
4. Select **Roles**.
5. Click **Create**.
6. Enter a role name, for example, **Cloud Native Visualization User**.
7. Enter **Description** (optional).
8. Select **Account**.
9. Click + **Add**. All Account permissions display.
10. Click the **View Infrastructure Entities** checkbox.
11. Click **Done**.

Step 2: Assign Users to the Cloud Native Visualization User Role

Using the role you defined previously, for example, **Cloud Native Visualization User**:

1. Go to **User and Groups with this Role.**
2. Click + **Add**.
3. Select the Users or Groups that you would like to grant this role.
4. Click **Done**.
5. Click **Save**.
   A security warning window displays.
6. Click **Accept** to save the role. A **Saved Successfully** message displays.
7. Refresh your browser to reflect the changes.

Next Steps

Monitor Cloud Native Applications.

Monitor Cloud Native Applications

As part of the Cloud Native launch program, early adopters will have access to the Cloud Native Visualization functionality on a trial basis. AppDynamics reserves the right to charge separately for Cloud Native Visualization after the initial trial period.

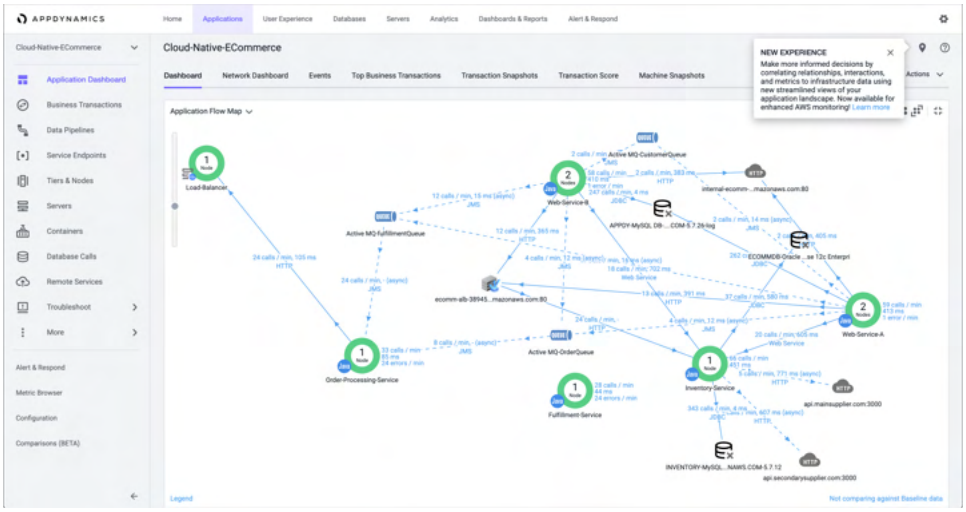
This page describes how to access, exit, and explore Cloud Native Visualization.

Access Cloud Native Visualization

Cloud Native Visualization access depends on the prerequisite configuration. See Set Up Cloud Native Visualization.

To access Cloud Native Visualization:

1. Go to https://accounts.appdynamics.com/subscriptions.
2. Sign in to your account.
3. Select the Controller that you connected to Amazon CloudWatch.
4. Go to **Applications** and select the desired application.
5. From the **Application Flow Map > Dashboard**, a **NEW EXPERIENCE** pin is introduced on the upper right corner of the AppDynamics application flow map page. You can explore your application's landscape using this new user interface.



6. Click the pin to enter Cloud Native Visualization. A Cloud Native Visualization **Applications** page displays.

Exit Cloud Native Visualization

To exit Cloud Native Visualization and return to the existing AppDynamics Controller interface, click **X** on the top bar of the upper-right corner of the page.

Next Steps

Cloud Native Visualization UI Overview.

Cloud Native Visualization UI Overview

As part of the Cloud Native launch program, early adopters will have access to the Cloud Native Visualization functionality on a trial basis. AppDynamics reserves the right to charge separately for Cloud Native Visualization after the initial trial period.

This page describes the new Cloud Native Visualization User Interface (UI) features that you can use to monitor and analyze your holistic application landscape. With Cloud Native Visualization, AppDynamics has introduced a new data model with concepts such as Relationships maps, Interactions maps, and Object pages that display Application Performance Monitoring (APM) and AWS performance data for these objects.

New terms are introduced that replace old terminology:

- Services (formerly tiers)
- Service Instances (formerly nodes)

See Tiers and Nodes.

Services (Tiers)

The app, tier, and node model has been at the heart of AppDynamics APM since its inception. Cloud Native Visualization slightly modifies this model. In particular, what was formerly known as tiers are now called **Services;** nodes are now called **Service Instances**.

Service Instances (Nodes)

In contrast to a node, the service instance concept represents the process lifecycle of a single application process, with an explicit lifetime. However for a service/node, however, the data model changes are more extensive than simple renaming. See Use Node Name Reuse for Java Agents in Cloud Native Visualization.

Cloud Native Visualization User Interface

Cloud Native Visualization is a series of dependency graphs. The Relationships map and Interactions map display a visual representation of relationships between objects within a business application. Objects are visual representations of the components of your application. In the Relationships map, you can click on each entity to learn more about its unique properties and origin. You can view these entities in the Interactions map to understand how data flows among the application components. As monitored systems become increasingly complicated with multiple entities and finer granularity, it becomes challenging for users to manage such complex systems. Cloud Native Visualization delivers a unified experience of data exploration and navigation through Object pages.

Navigate Object Pages

An Object page is a view that corresponds to an object type, such as service instances, databases, or hosts. Cloud Native Visualization uses object pages to model systems as a set of entities, relationships, and paths between those entities. Object pages can represent aggregations of components, such as services (tiers), which are an aggregation of several service instances (nodes) and can also represent flows, such as business transactions. O bject pages correlate the most important information for a given object on one page.

For each object type, Cloud Native Visualization object pages display both:

- **Lists** view—Displays multiple objects of a particular entity type. The **Lists** view enables a user to click on a specific object to see a **Details** view.
- **Details** view—Displays one instance of an entity. Click on a specific object in a list to access the **Details** view.

Object Page Components

Top Bar:

- Browsing History Path
- Time Range Menu
- Close

Left Panel:

- Filter By
- Relationships Map

Main Content Area:

- Interactions Map
- Details View
- Lists View

Right Panel:

- Configure Health

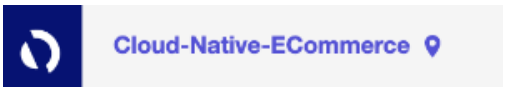
- Properties Panel

Top Bar

The Cloud Native Visualization top bar consists of a clickable Browsing History Path, a Time Range Menu, and a Close button **X** to exit the new UI experience.

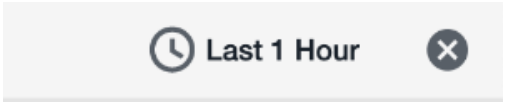
Browsing History Path

The Browsing History Path specifies the trail of recent navigation. You can click on names in the path to navigate to previous pages.



Time Range Menu

The Time Range Menu selects what data displays throughout the Cloud Native Visualization UI. The Time Range Menu does not have preconfigured time ranges. To create time ranges you must use the existing Controller user interface. After you create the time range, you can return to the new Cloud Native Visualization UI. When you select a new time range from the menu, it remains the active range as you navigate to other pages in the UI.



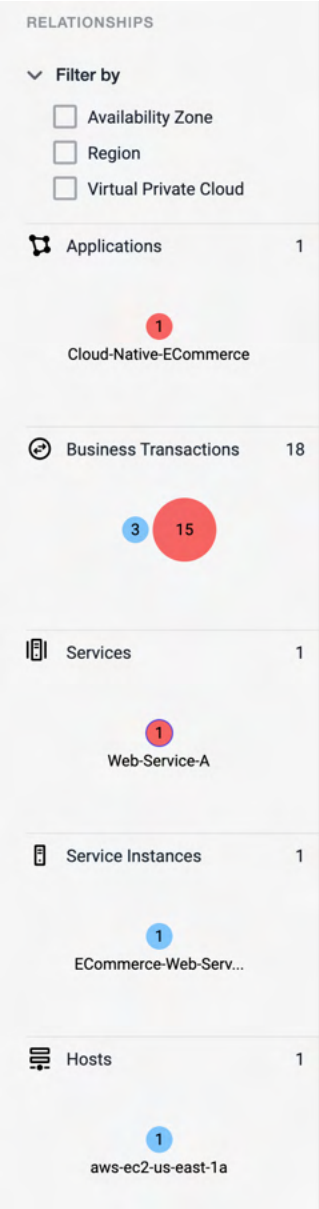
Close

The Close button **X** enables you to exit Cloud Native Visualization and return to the existing AppDynamics Controller user interface.

Left Panel

The Cloud Native Visualization left panel consists of a **Filter by** feature and the **Relationships** map. Objects are categorized as healthy or unhealthy using color-coded group circlets to depict health status for quick health issue identification. See Monitor the Health Status of Entities.

- Blue—indicates that the entities are healthy.
- Orange—indicates a health warning.
- Red—indicates that the entities have critical health issues.



Filter by

The **Filter by** feature in Cloud Native Visualization enables you to filter by zones, regions, or VPCs and is available for all entities that display on the **O bject** page, **Relationships** map, and **Interactions** map.

After you apply a filter, each section updates with a subset of corresponding business transactions, services, and so on, based on the object's context. The **Interactions** map displays services that are not part of the focused filter opaquely.

The available filters are:

- **Availability Zone (AZ)**: An Availability Zone (within a region) offers one or more unique data centers with redundant power, networking, and connectivity in a region to ensure resilience.
- **Region**: Regions are physical locations where data centers are geo-located to help businesses determine the correct geography for data compliance, service availability, and pricing.
- **Virtual Private Cloud**: VPCs contain multiple availability zones.

See Amazon VPC for On-Premises Network Engineers – Part 1.

> You can apply only one filter at a time. The **Filter by** selection does not persist through multiple selections. If you click on an entity, the filter is reset.

Relationships Map

The **Relationships** map enables you to navigate the vertical layers of entities that make up your application. At the highest level is **Applications**, then below Applications are **Business Transactions**, then **Services**, **Services Instances**, **Hosts**, **Databases**, and other relevant associated infrastructure entities. The entities are categorized as healthy and unhealthy based on their health status and display as group circlets. A blue-colored group circlet indicates that the entities are healthy, while a red-colored group circlet indicates that the entities are unhealthy.

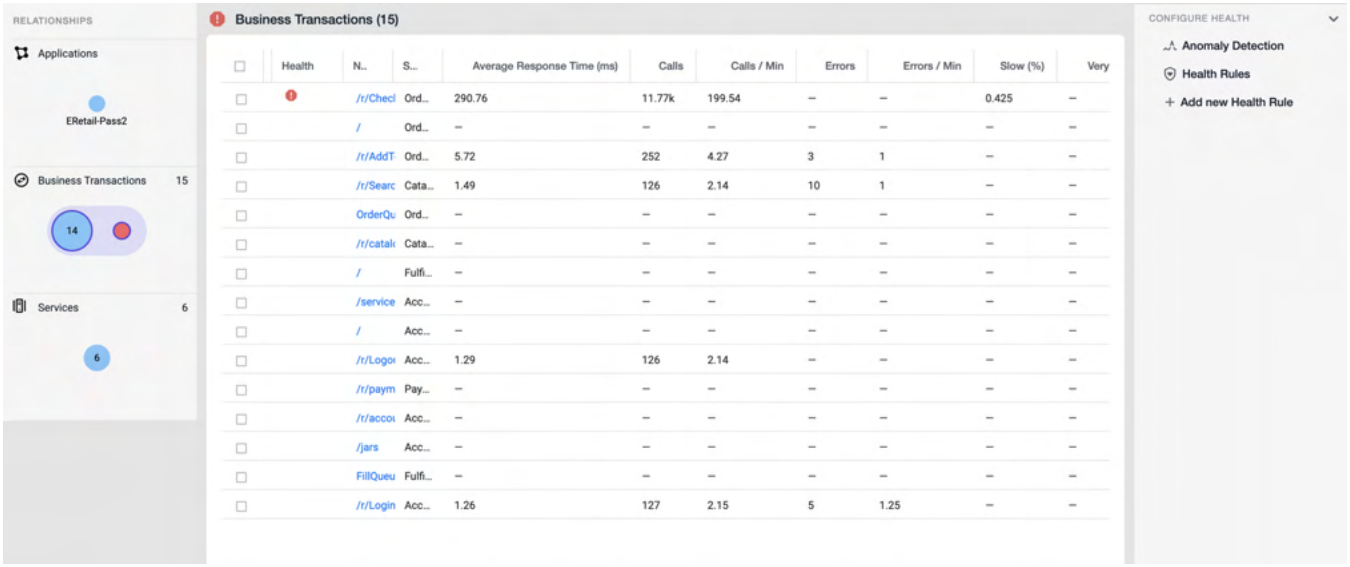
In the **Relationships** map, if you click any object, content is updated within the context of that object. Different types of entities display in the **Relationships** map depending on the entity you are viewing. For example, **Service Instances** do not display on the **Relationships** map when looking at **Applications**. **Service Instances** display when you reach the level of **Services**. If hosts and databases are not associated with an application, then those layers do not display.

Main Content Area

The middle panel is the main content area and consists of the **Lists** view, **Interactions** map, and **Details** view.

List View

The **List** view displays in the main content area. The **List** view displays a list of multiple instances of one entity type. In the following example, the entity type is **Business Transactions**, and there are six (6) **Business Transactions** associated with the **CloudMon-Staging** entity.



The **List** view changes based on the object in focus. It presents different columns and metrics depending on what object is selected. The **Health** column indicates the status of the entities. A dark red icon indicates that the entity is in a `Critical` state, while an orange icon indicates that the entity is in a `Warning` state. If the entity is healthy, then no icon displays in the **Health** column. In this example, the **Business Transactions Object** page displays Key Performance Indicator (KPI) metrics for each column.

You can select entities in the List view to view associated health rules or enable anomaly detection. See Configure Entity Health.

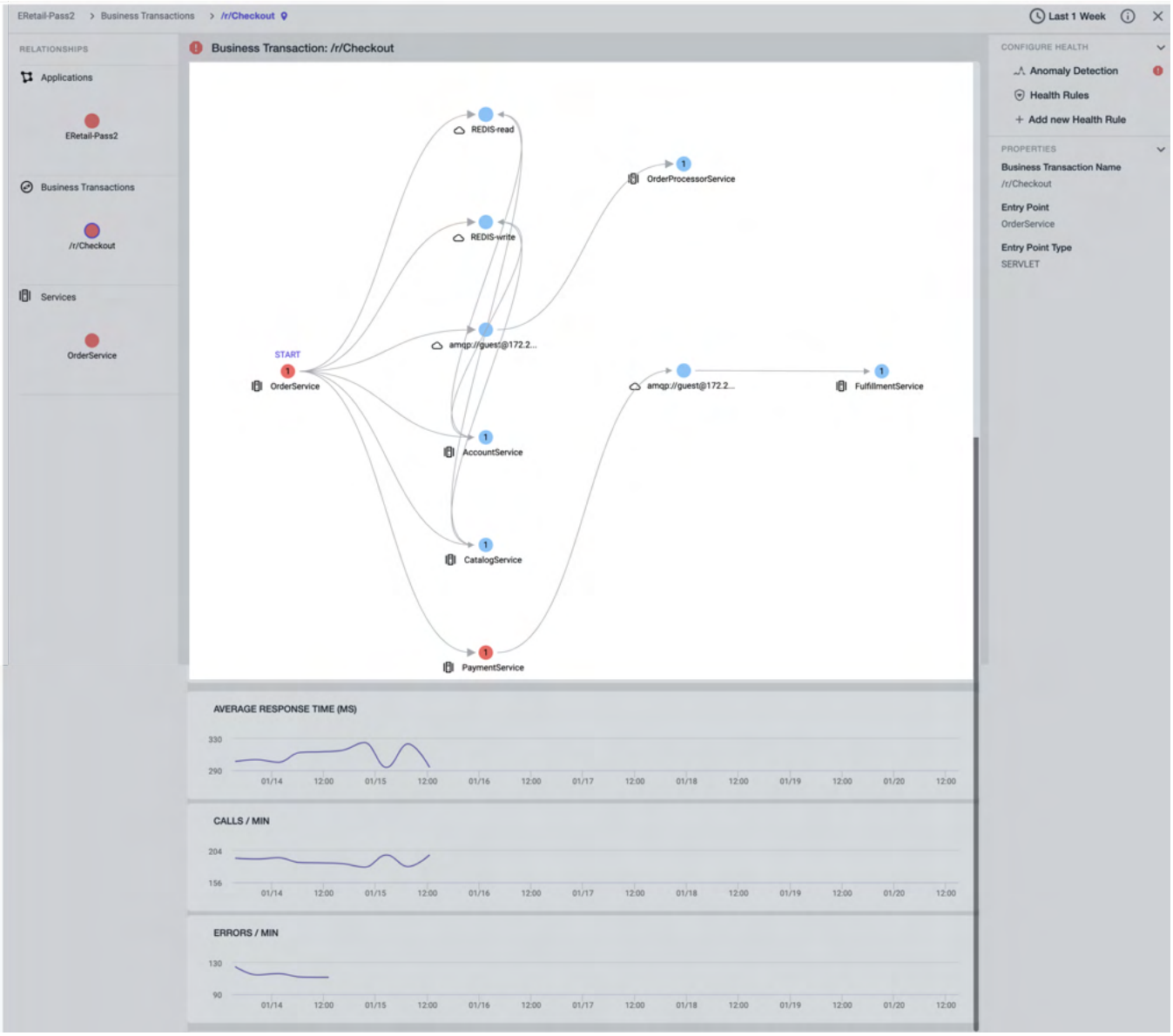
Interactions Map

The **Interactions** map displays in the main content area. An **Interactions** map is a graph that represents how services, databases, hosts, and other associated infrastructure entities within a business transaction, interact with each other. You can measure the entities' performance and track the origin and the destination of these interactions to trace a sequence of requests through your application. The **Interactions** map is only available in these three views:

- Business Transactions Details
- Services Details
- Service Instances Details

The **Interactions** map indicates the health status of the entities it displays. You can use the **Interactions** map to drill down an unhealthy entity to determine the affected services or service instances. Currently, the **Applications** page (Details view) only displays an Interactions list, not an Interactions map.

The **Configure Health** options available in the right panel help monitor the health and performance of the selected entity. You can configure health rules or use anomaly detection for the entity. The health and performance status of the entity appears adjacent to the **Configure Health** options. In the following example, the interaction map depicts the interactions for `/r/Checkout` business transaction. The critical icon ( ) adjacent to **Anomaly Detection** in the right panel indicates that anomaly detection is enabled for `/r/Checkout` business transaction and an anomaly is detected with `critical` severity.



Details View

The **Details** view displays one instance of an object's details. The **Details** view shows Key Performance Indicator (KPI) metric trends based on the object page context. In this example, these metrics display for the **Applications** page:

- **AVERAGE RESPONSE TIME (MS)**
- **CALLS / MIN**
- **ERRORS / MIN**

Right Panel

The Cloud Native Visualization right panel consists of these sections:

- **Configure Health**
- **Properties**

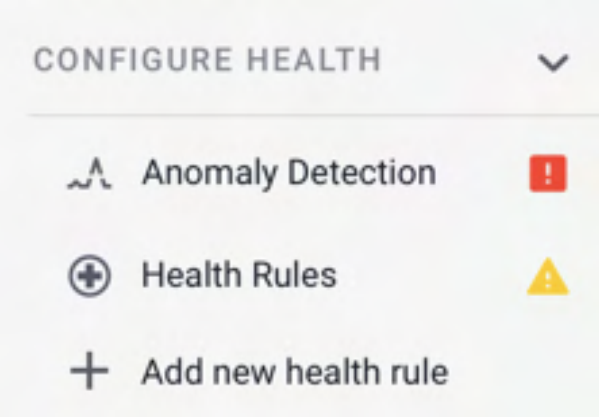
Configure Health

The **Configure Health** section provides options that help monitor the performance and the health of entities.

- **Anomaly Detection**—Automatically determines whether the selected entities (business transactions) in your application perform normally. It takes 48 hours for Anomaly Detection to model train the entities to monitor the performance. Model training is done to determine the normal operating range of parameters of an entity and report anomalies if the parameters vary from the established normal range. The **Anomaly Detection** view lists the entity names and indicates the model training status of the entities. The **Health** column in this view indicates the performance status of the entities as Critical state (icon), Warning state (icon) or Unknown (icon). If no icon is displayed in the column, the performance of the entity is in the normal operating range.
- **Health Rules**—Displays all health rules associated with the selected entities. The **Health Rules** view lists the health rule name, monitored objects, and indicates if the health rule is enabled. You can click the monitored objects in the **Health Rules** view to display a list of all the

monitored objects in the **Configure Health** section in the right panel. The **Health** column indicates the health status of the monitored entities as Critical state (icon) or Warning state (icon). If no icon displays in the column, the entity is healthy. You can create, modify, delete, or enable and disable a health rule based on the performance data.

- **Add new Health Rule**—Displays the **New Health Rule** dialog. Depending on the entities you select, some fields in this dialog are pre-populated with values. You can configure the other parameters to create a new health rule.



See Configure Entity Health.

Properties

Each object type has its own set of properties. **Properties** display the attributes of each instance of an object, such as the name and corresponding metadata. This area is context-sensitive; it displays information and tools relevant to the user context. From the **Lists** views, you can select a single instance of an object to view the **Properties** in the right panel. From the **Details** views, **Properties** display for the object in focus.

PROPERTIES

Instance ID
i-024b6b7c1982b12b7

Launch Time
2020-07-29T23:24:10Z

State Name
RUNNING

Monitoring State
DISABLED

State Transition Reason
-

Instance Type
t2.2xlarge

CPU Options Core Count
8

Public DNS Name
-

Public IP Address
3.133.122.178

VPC ID
vpc-09fa913cdb2bdc3d5

Region
us-east-2

Availability Zone
us-east-2a

Subnet ID
subnet-0836819a286407c49

Root Device Name
/dev/sda1

Applications Page

As part of the Cloud Native launch program, early adopters will have access to the Cloud Native Visualization functionality on a trial basis. AppDynamics reserves the right to charge separately for Cloud Native Visualization after the initial trial period.

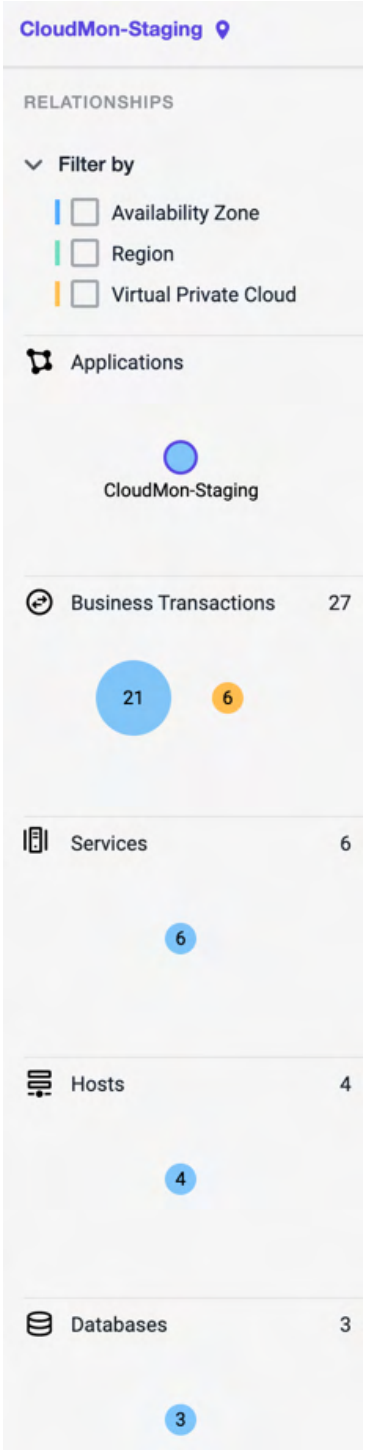
When you access a Cloud Native Visualization **Applications** page, the service application displays as the object of focus. An Application's **Object** page has four main components:

- Relationships map
- Interactions list
- Metrics
- Configure Health

Relationships Map

On the **Applications** page, in this example **CloudMon-Staging**, the **Relationships** map lists the objects related to the application and the count of each specific entity that comprises the service:

- **Applications**
- **Business Transactions (27)**
- **Services (6)**
- **Hosts (4)**
- **Databases (3)**

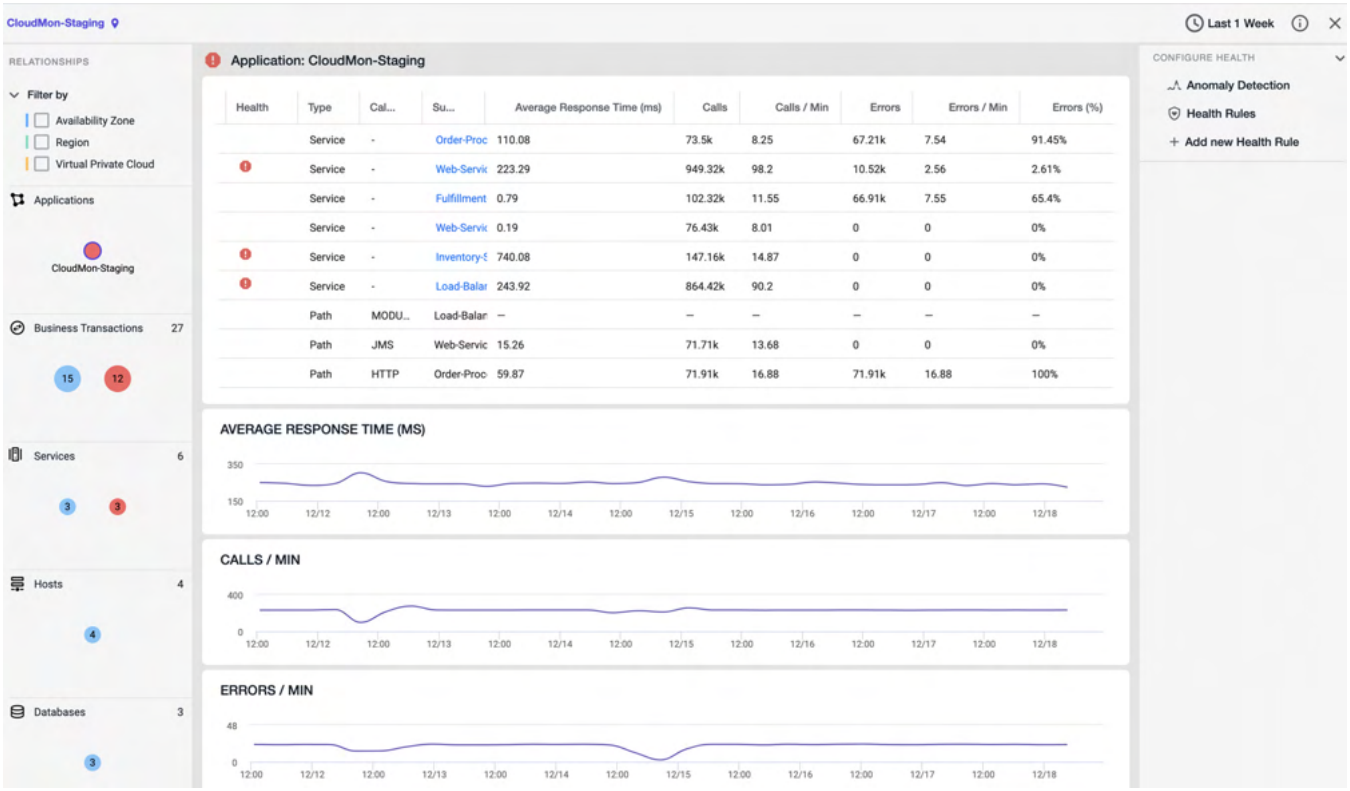


Health Status of the Entities

The **Relationships** map displays the health status of the listed entities. The entities are categorized as healthy or unhealthy based on their health status, and display as group circlets. A blue-colored group circlet indicates that the entities are healthy, while an orange-colored or a red-colored group circlet indicates that the entities are unhealthy. See Monitor the Health of Entities.

Interactions List

The Interactions list displays in the main content area. This list displays all of the associated objects and paths between object entities and related Key Performance Information (KPI) for each. It also indicates the status of the entities in the **Health** column.



The **Applications** page Key Performance Indicator (KPI) metrics are displayed for each column in a data grid.

Name	Description
Health	The **Health** column displays icons that indicate the health status of the entities: •  —Indicates that the entity is in a `Critical` state. You can click the entity to drill down and determine the affected business transactions, services, and service instances. •  —Indicates that the entity is in a `Warning` state. You can click the entity to drill down and determine the affected business transactions, services, and service instances. • No icon—Indicates that the entity is healthy.
Type	A call graph can be one of these types: • Full call graphs—Capture the entire call sequence for the business transaction invocation. In this case, call graphs exist for each monitored service instance involved in the processing of the business transaction. Periodically collected and diagnostic snapshots are always full call graphs. • Partial call graphs—Represent the subset of the call sequence for processing a business transaction, typically from the point at which the transaction has been determined to be slow or contain errors. Partial call graphs are identified as such in the transaction snapshot.
Call Type	A call graph in a transaction snapshot shows business transaction processing information on a particular service that participated in the business transaction. A call graph lists the methods in a call stack and provides information about each call.
Summary	Name of the associated entity.
Average Response Time (ms)	Response time-averaged over the time range in milliseconds; aggregated per application.
Calls	Total number of calls.
Calls / min	Average requests (calls) per minute (Load).
Errors	Total number of errors.
Errors / min	Number of errors per minute.

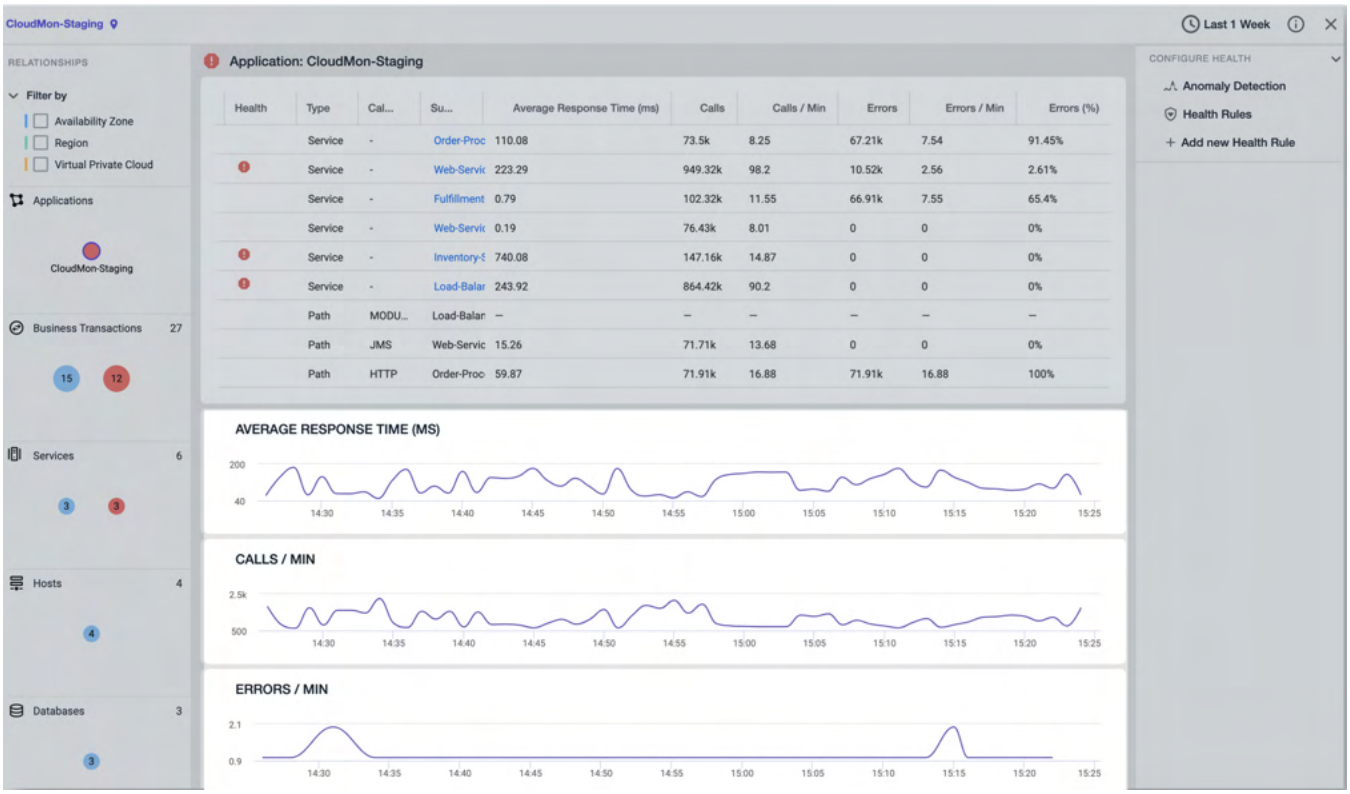
Errors (%)	Percentage of errors per minute.

When you navigate from an **Object** page to an Interactions list through the Relationships map, the Relationships map does not change. The Interactions list retains the context of the entity that you are viewing.

Metrics

The **Metrics** section displays three key application metrics:

- **Average Response Time (ms)**
- **Calls Per Minute (load)**
- **Errors Per Minute**

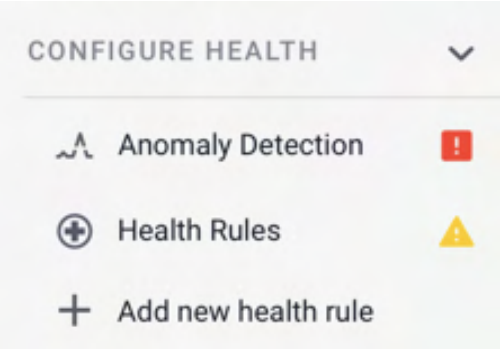


The cards display details in expandable rows. The data in these rows roll up information from all service instances (nodes) within a group.

Configure Health

The **Configure Health** section provides options that help monitor the performance and the health of entities.

- **Anomaly Detection**—Automatically determines whether the selected entities (business transactions) in your application perform normally. It takes 48 hours for Anomaly Detection to model train the entities to monitor the performance. Model training is done to determine the normal operating range of parameters of an entity and report anomalies if the parameters vary from the established normal range. The **Anomaly Detection** view lists the entity names and indicates the model training status of the entities. The **Health** column in this view indicates the performance status of the entities as Critical state (icon), Warning state (icon) or Unknown (icon). If no icon is displayed in the column, the performance of the entity is in the normal operating range.
- **Health Rules**—Displays all health rules associated with the selected entities. The **Health Rules** view lists the health rule name, monitored objects, and indicates if the health rule is enabled. You can click the monitored objects in the **Health Rules** view to display a list of all the monitored objects in the **Configure Health** section in the right panel. The **Health** column indicates the health status of the monitored entities as Critical state (icon) or Warning state (icon). If no icon displays in the column, the entity is healthy. You can create, modify, delete, or enable and disable a health rule based on the performance data.
- **Add new Health Rule**—Displays the **New Health Rule** dialog. Depending on the entities you select, some fields in this dialog are pre-populated with values. You can configure the other parameters to create a new health rule.



See Configure Entity Health.

Monitor the Health of Entities

As part of the Cloud Native launch program, early adopters will have access to the Cloud Native Visualization functionality on a trial basis. AppDynamics reserves the right to charge separately for Cloud Native Visualization after the initial trial period.

Cloud Native Visualization groups entities in the **Relationships** map, **Interactions** map, and the **List** view based on their health status. All entities are grouped by color-coded health status, into healthy and unhealthy categories. This table shows how the color of the group circlet indicates the health status of entities:

Entity/Object Icon	Health Status Description
	Twenty entities are operating within normal parameters and are healthy. No alerts are reported for these entities.
	Twelve entities are unhealthy, in `Critical` state.
	Seven entities are unhealthy, in `Warning` state.

View the Health Status of Entities

The **Relationships** map displays the health status of entities. This helps you identify and focus on those entities (out of the thousands in your account) that need attention. You can initiate remediation actions to quickly resolve any critical issues.

Group circlets indicating healthy and unhealthy entities display in the **Relationships** map. You can click the circlets to view the health status details of all entities.

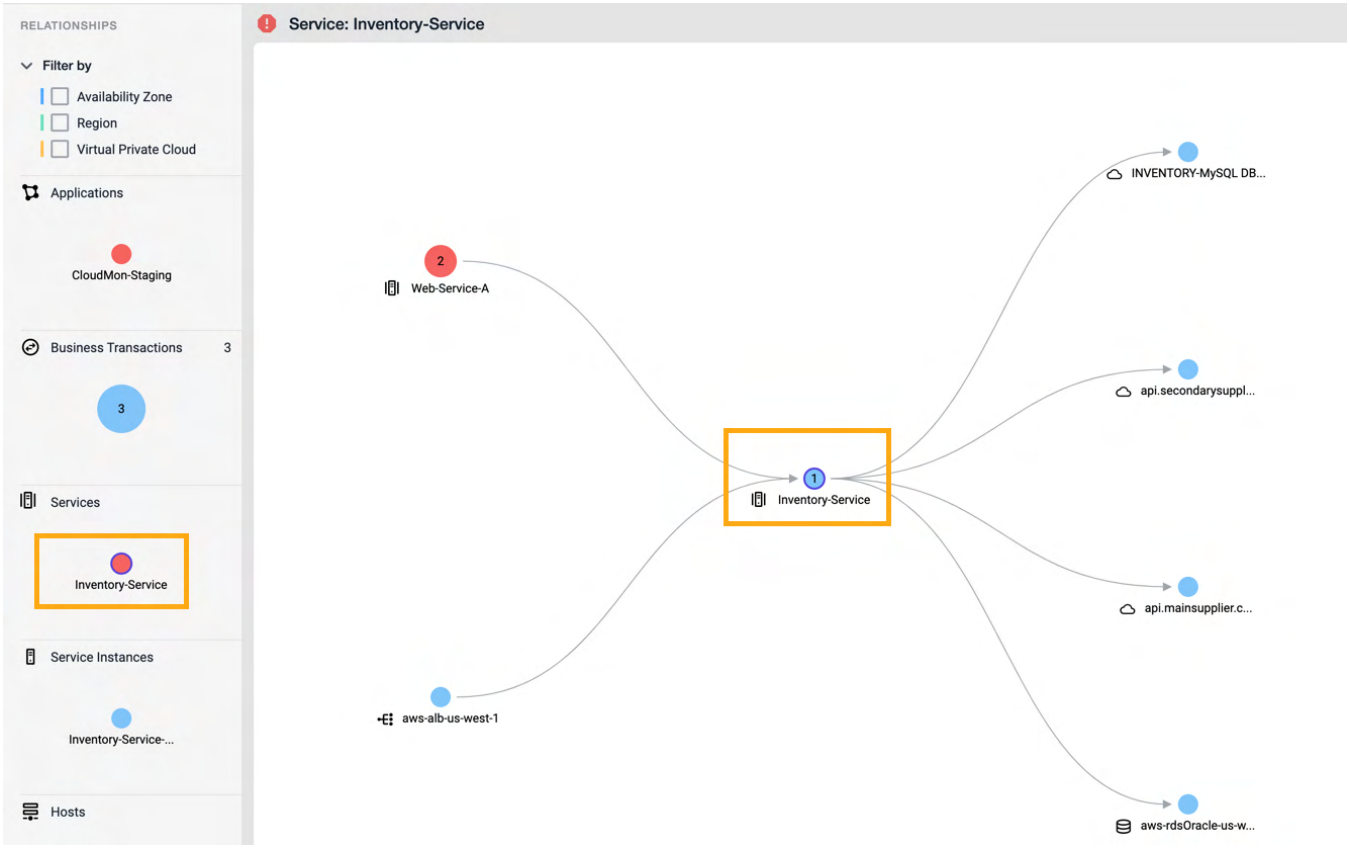
After you click the entity groups in the **Relationships** map, a list of entities associated with the parent entity and its corresponding details display in the **List** view. The **Health** column displays icons that indicate the health of the entities.

Health Icon	Description
	Indicates that the entity is in `Critical` state. You can click the entity to drill down and determine the affected business transactions, services, and service instances.
	Indicates that the entity is in `Warning` state. You can click the entity to drill down and determine the affected business transactions, services, and service instances.
No icon	Indicates that the entity is healthy.

Determine the Affected Entities

An unhealthy group circlet in the **Relationships** map displays the cumulative health status of all comprising entities. If one of the many services, or service instances within the object is unhealthy, the overall status is considered to be unhealthy.

In the following example, the status of **Inventory Service** in the **Relationships** map is unhealthy. However, **Inventory Service Instance** is healthy as shown in the **Interaction** map. This is because, though the parent entities are unhealthy, some of their child entities (service instances) remain healthy.



To determine the affected services or service instances:

1. Click the **Last 1Hour** dropdown list on the top right corner of the page and select a time period to retrieve data.
2. In the **List** view, click the hyperlinked name of the unhealthy entity. All associated entities (including services and service instances) display on the **Interaction** map.

   > The properties of the entity you select on the **Interaction** map display on the **Properties** pane on the right.

3. Note the affected entities.
4. Click each of the affected entities to determine the affected services or service instances.

View Performance Status of Affected Entities

To monitor the performance of the selected entities using anomaly detection:

1. In the **List** view, select the affected entities.
2. Click the **Anomaly Detection** option in the right panel. The **Anomaly Detection** view replaces the **List** view. It takes 48 hours for anomaly detection to start reporting performance data.
3. To return to the **List** view, close the **Anomaly Detection** view.

See Anomaly Detection.

View Health Rules Associated with Affected Entities

To view the health rules associated with the entities.

1. In the **List** view, select the affected entities.
2. Click the **Health Rules** option in the right panel. The **Health Rules** view replaces the **List** view. This view lists all the health rules that you have configured to monitor the selected entities.
3. If you want to view the entities that the health rule monitors, click **Applied on <> Objects** in the **Monitored Objects** column. A list of monitored entities displays in the right panel with the health status indicator icons.
4. If you want to edit the health rule, click the health rule name. The **Edit Health Rule** dialog displays. Make necessary updates and save the health rule.
5. To return to the **List** view, close the **Health Rules** view.

See Health Rules.

Configure Entity Health

As part of the Cloud Native launch program, early adopters will have access to the Cloud Native Visualization functionality on a trial basis. AppDynamics reserves the right to charge separately for Cloud Native Visualization after the initial trial period.

The right panel on the **Cloud Native Visualization Object** page provides these options to configure the health rules and monitor the performance of the entities:

- Anomaly Detection
- Health Rules
- Add new Health Rule

To configure these options, select:

- a single entity or multiple entities from the **List** view, or
- an entity from the **Interaction** map

When you select an option, the **Configure Health** view replaces the **List** view or the **Interactions** map content area.

Anomaly Detection

Anomaly Detection uses machine learning capabilities to continuously monitor latency, errors, and throughput of business transactions to reduce the Mean Time to Detect (MTTD) when an anomaly occurs. It takes 48 hours for anomaly detection to become available. During that time, the machine learning model trains on the business transactions in your application. Model training determines the normal operating range of parameters of an entity and report anomalies if the parameters vary from the established normal range.

When you configure anomaly detection for business transaction entities, the **Anomaly Detection** view replaces the **List** view content area. This view lists the entities, their health status, and model training status.

The **Health** column in this view indicates the health status of the business transactions as follows:

-  —The entity has an anomaly and is in `Critical` state.
-  —The entity has an anomaly and is in `Warning` state.
-  —The health status is not yet available for various reasons such as the model training is in progress, or there is not enough data.
- no icon—The performance of the entity is in the normal operating range.

The **Model Training** column on the right displays the training statuses as follows:

- **In Training**—Model training is in progress for the business transaction. It takes 48 hours for anomaly detection to start reporting data.
- **Ready**—Model training is complete.
- **Not available**—Model training cannot proceed because there is not enough data for the select business transaction.

Model training data is refreshed every hour.

View Anomaly Detection Status

To view Anomaly Detection status for entities in the **List** view:

1. Select the required entities. You can select all the entities by checking the select all check box.
2. Click **Anomaly Detection** in the **Configure Health** section of the right panel. **Anomaly Detection** view replaces the **List** view content area.
3. Check the **Health** column to determine if any entities are unhealthy. If any of the entities are reported unhealthy, you can initiate remediation steps.

Health Rules

Cloud Native Visualization lists all the health rules configured to monitor the entities in your application. When you select the **Health Rules** option from the right panel, the **Health Rules** view replaces the **List** view content area. The total number of health rules configured for the selected entities is indicated in brackets. This view lists the health rule name, its description, the objects it monitors and also indicates if the health rule is enabled. When you click the monitored objects in the **Health Rules** view, a list of all the monitored objects displays in the **Configure Health** section in the right panel.

Health Rule Operations

The **Health Rules** view allows you to perform various CRUD operations. You can also enable or disable the evaluation of health rules. This table describes the health rule operations:

Operation	Description

+	Creates a new health rule for the selected entity.
	Deletes an existing health rule. ⓘ You can delete only one health rule at a time.
Monitoring	Enables or disables the evaluation of the health rule for the selected health rules or all the health rules. ⓘ If you disable the evaluation of all the health rules, you must confirm the action.
	Refreshes the list of health rules.
Filter 	Searches for health rules with the given keyword.

Create a New Health Rule

You can create a health rule to monitor a single entity or multiple entities.

ⓘ You can create a maximum of 50 health rules per account.

1. From the **List** view, select the entities to be monitored by the health rule you configure.

   ⓘ You can configure a maximum of 25000 entities to be monitored by a health rule. If you select more than 25000 entities, the health rule is not evaluated.

2. In the Cloud Native Visualization right panel, click **Add New Health Rule**. The **New Health Rule** dialog displays.

   Alternately, if you are already in the **Health Rules** view, click + (add new health rule icon).

   ⓘ The entities selected in the **List** view are pre-selected in the **New Health Rule** > **Select Object Type** dialog.

3. In the **Select Object Type** dialog, configure the object details and filter criteria:

   a. Select an object from the **Object Type** dropdown list to be monitored by the health rule.

      ⓘ If the list of health rules is lengthy, use the search box available on the top right corner to search for the required health rule.

   b. To apply filters to your selection:

      i. In the **FILTER BY PARENT OBJECT TYPE** area, select a parent object from the **Parent Object** dropdown.
      ii. To specify a filter condition:
         1. Specify a **Tag Key**.
         2. Select an operator.
         3. Define a value to be matched in the **Tag Value** field.
      iii. To add a related parent object to the filter, click **Add related parent object type** and select an object.

   c. Define the health rule evaluation scope:

      i. To trigger health rule violation alerts based on the performance of an individual object, for example, service instance, select **Apply to each <object> individually**.
      ii. To trigger health rule violation alerts based on the performance of a group of objects, and grouped by their parent object, for example, service instances grouped by a service:
         1. Select **Apply to groups of monitored objects**.
         2. Select an option from the **group objects by** dropdown.
         3. Select a parent object from the adjacent dropdown.

4. Click **Next**. The **Define Conditions** dialog displays.

> ⚠ Ensure that the configuration data you enter is accurate before you configure conditions. If you make changes to the object filter or evaluation scope after you configure conditions, the conditions are reset.

5. Define a condition or a set of conditions to evaluate the performance of the selected objects or group of objects:
    a. In the **Critical Criteria** tab, select a value from the **Use data from last <> min(s)** dropdown. The list displays numbers between 1 and 120 minutes. The data collected during this time interval is the minimum amount of data required to evaluate a health rule and determine if there is a health rule violation.

    > ⓘ If you define a persistence threshold for the health rule condition, ensure that you define an evaluation time frame of 30 minutes or less.

    b. Click **+ Add Condition** to add a new condition component. A row defining the condition displays.
    c. Configure the Condition as required.
    d. To add more conditions, repeat steps b and c.
    e. To evaluate multiple conditions within a health rule, use a boolean expression.
    f. To copy the critical criteria configuration to warning criteria, click **Copy from Critical Criteria**. You can modify the warning criteria if required.
6. Click **Next**. The **Display Health** dialog displays.
7. To mark the unhealthy status of the objects on dashboards:
    a. If you selected a single object to monitor in the health rule evaluation scope, select the **Mark <> as unhealthy** check box.
    b. If you selected a group of objects to monitor in the health rule evaluation scope, select the **Mark <> as unhealthy** check box and enter a % value. The dashboards report the unhealthy status of the group of objects only if % of objects exceeds the given value.
8. Click **Next**. The **Overview** dialog displays.
    a. Enter a name for the health rule. The health rule is enabled by default.
    b. In the **Wait Time after Violation** field, enter the number of minutes to wait (if the health status remains unchanged) before re-evaluating the health rule. See Health Rule Wait Time After Violation.
9. Click **Save**.

Configure a Condition

1. In the first field of the condition row, enter a name for the condition. This name is used in the generated notification text and in the AppDynamics Console to identify the violation.
2. From the dropdown below the **Add Condition** button, define metrics to evaluate the condition. Select:
    - a. From the adjacent dropdown, select a metric qualifier from the following options:

Qualifier Type	Description
Minimum	The minimum value reported across the configured evaluation time frame.
Maximum	The maximum value reported across the configured evaluation time frame.
Value	The arithmetic average of all metric values reported across the configured evaluation time frame. This value is based on the type of metric.
Sum	The sum of all the metric values reported across the configured evaluation time frame.

    b. From the adjacent dropdown list, select a metric to monitor.

    c. From the dropdown after the metric, select comparison type to evaluate the metric.

    - To limit the effect of the health rule to conditions during which the metric is within a defined range—standard deviations or percentages—from the baseline, select **Within Baseline**. To limit the effect of the health rule to when the metric is not within that defined range, select **Not Within Baseline**. Then, select the baseline to use, the numeric qualifier of the unit of evaluation, and the unit of evaluation. For example:

      ```
      Within Baseline of the Default Baseline by 3 Baseline Standard Deviations
      ```

    - To compare the metric with a static literal value, select **< Specific value** or **> Specific Value** from the menu, then enter the specific value in the text field. For example:

      ```
      Value of Errors per Minute > 100
      ```

    - To compare the metric with a baseline, select **< Baseline** or **> Baseline** from the dropdown, and then select the baseline to use, the numeric qualifier of the unit of evaluation, and the unit of evaluation. You can use the Baseline Standard Deviation or Baseline Percentage as the unit of evaluation. For example:

```
Maximum of Average Response Time is > Baseline of the Daily Trend by 3 Baseline Standard Deviations
```

See Dynamic Baselines for information about the baseline options.

> **Baseline Percentages**
>
> The *baseline percentage* is the percentage above or below the established baseline at which the condition is triggered. For example, if you have a baseline value of 850 and you have defined a baseline percentage of > 1%, then the condition is true if the value is > `[850+(850x0.01)]` or `859`.
>
> To prevent health rule violations from being triggered when the sample sets are too small, these rules are not evaluated if the load—the number of times the value has been measured—is less than 1000. For example, if you specify a very brief time slice, the rule may not violate even if the conditions are met, because the load is not large enough.

- or

a. Select the **Metric Expression** option from the dropdown and click **Add Expression**. The **Metric Expression** dialog displays that allows you to construct a mathematical expression to use as a metric.
b. Click **Add Variable** to add a variable.
c. In the **Variable Name** field enter a name for the variable.
d. From the dropdown, select a metric qualifier from the following options:

Qualifier Type	Description
Minimum	The minimum value reported across the configured evaluation time frame.
Maximum	The maximum value reported across the configured evaluation time frame.
Value	The arithmetic average of all metric values reported across the configured evaluation time frame. This value is based on the type of metric.
Sum	The sum of all the metric values reported across the configured evaluation time frame.

e. Select a metric from the adjacent dropdown.

> **Health Rule Evaluation Condition**
>
> A health rule is not evaluated if any metric in the expression has a null value. This is to avoid erroneous evaluations as shown in the following examples:

Expression	Null Value	Evaluation
`a-b-c`	a	`entire expression is evaluated negative`
a/b	b	`the number 'a' is divided by zero, evaluates to an error`
`a*b`	a or b	`entire expression is evaluated as zero`

f. Repeat steps 1 through 4 for each metric that you use in the expression.
You can remove a variable by clicking **X**.
g. In the **Expression** section, build the expression by clicking **Insert Variable** to insert variables you created and appropriate mathematical signs.
h. When the expression is complete, click **Submit**.
i. To compare the metric with a static literal value, select **< Specific value** or **> Specific Value** from the menu, and then enter the specific value in the text field.

3. If you want the condition to evaluate to `True` whenever a configured metric does not return any data during the evaluation time frame, expand **Advanced Settings** and check the **Evaluate to true on no data** option.
This option does not affect the evaluation of unknown when there is not enough data for the rule to evaluate. For example, if the health rule is configured to evaluate the last 30 minutes of data and a new node is added, the condition evaluates to unknown for the first 30 minutes even if the **Evaluate to true on no data** box is checked.
4. To define a Persistence Threshold for the condition to reduce false alerts:

a. Select **Trigger only when a violation occurs __ times in the last __ min(s)**.
b. Define the number of times metric performance data should exceed the defined threshold to constitute a violation.
c. If required, adjust the evaluation time frame by setting an alternate evaluation time frame.

> You can define a Persistence Threshold for a condition only if you have defined an evaluation time frame of 30 minutes or less.

Create a Custom Boolean Expression

Once you define all the conditions required for the health rule, create a custom boolean expression:

1. In the Critical Criteria tab, select **CUSTOM** from the **If <> of the following conditions are met** dropdown.
2. Enter a combination of conditions using `AND` and/or `OR` operators. For example, `(A OR B) AND C`.

Update a Health Rule

To update a health rule:

1. In the **Health Rules** view, click the health rule name. The **Edit Health Rule** wizard displays.
2. Make the necessary updates as the wizard guides you through the procedure.
3. Click **Save** to apply the updates.

This video describes the configurable entity health features in Cloud Native Visualization.

Sorry, your browser doesn't support embedded videos.

Cloud Native Visualization Metrics Overview

As part of the Cloud Native launch program, early adopters will have access to the Cloud Native Visualization functionality on a trial basis. AppDynamics reserves the right to charge separately for Cloud Native Visualization after the initial trial period.

This page provides an overview of Cloud Native Visualization metrics in AppDynamics. Cloud Native Visualization provides a variety of graphs and tools that visualize metric data so that you can effectively analyze your application's performance. It also enables you to extend and adapt metrics in specific ways.

AppDynamics allows you to create metrics based on various domain-related data collected from applications, services, and endpoints. These metrics can be used for monitoring the health and performance of the entity as well as correlating different entities and domain data.

What is a Metric

Metrics are numerical measurements of an entity, or interactions measured or sampled over a period of time, typically with a fixed frequency. Because of this, Metrics are represented through time-series visualization. Metrics are inherently compressible and are summarized both through an entity's relationships and through an entity's attributes at any given time.

For information about variable granularity for the AWS monitoring, see Variable Ingestion Granularity for AWS Detailed Monitoring.

Metrics Data Model

AppDynamics Measurements are registered by a domain or feature in a type system. APM, Server, Network, Database are domain examples, and have measurements such as "Response Time" or "Message Size". A measurement has certain properties, and each measurement belongs to a category and content type as described in the Measurement Types table. A measurement packet can also have elements that belong to different content types, depending on whether measurements were ingested at read-time or post-aggregation.

Measurement Types

Measurement types determine how data is aggregated. There are five categories of measurement types:

- Counter
- Gauge
- Meter
- Monotonic
- Rate

The Content types at ingestion or post-aggregation are:

- Number: A single number that defines a measurement value at any particular timestamp. For example: `CPU = 65`.
- Distribution: A typical distribution of a measurement value for a particular timestamp will encompass `SUM, MIN, MAX, CURRENT, COUNT, GROUP_COUNT` fields.

For a measurement type that belongs to a category, aggregations performed by the AppDynamics platform are predefined.

Category	Description	Content Types at Ingestion	Content Types Post-aggregation	Default Consumption	Example	The Expression for Observed Value Computation
Counter	Measures discrete counts that need to be summed up for consumption.	• Number • Distribution	Distribution	The cumulative total of the count.	• Number of Slow Calls • Request Count • Event Count	`SUM`
Gauge	Measures, observes, or samples something that is constantly changing and is consumed as a per-instance value.	• Number • Distribution	Distribution	Typical Observed Sample per instance.	• CPU Idle %, • Memory Utilization % • Queue Length • Lag	`CURRENT / GROUP_COUNT`
Meter	Measures discrete metric values over a period.	• Number • Distribution	Distribution	Average value on a per-period basis.	• Response Time • Execution Time • Request Size • Message Size	`SUM/COUNT`

Monotonic	Measures monotonically increasing metric values.	• Number • Distribution	Distribution	Sum of all the monotonic counts.	• Total bytes consumed since the beginning of an hour • Total bytes received since the beginning of the day	`CURRENT`
Rate	Measures discrete counts, but is typically consumed as average per-period.	• Number • Distribution	Distribution	Average of the total count on a per-period basis.	• Calls per Minute • Errors per Minute	`(SUM * GROUP_COUNT) / COUNT`

Observed Value Computation Metrics results include the following fields:

Name	Definition
`CURRENT`	The last value of the metric in the time unit. This is taken based on the latest time for time roll ups.
`COUNT`	The number of values (or samples) recorded in the time unit. Varies based on metric type, such as CPM or ART.
`GROUP_COUNT`	The number of service instances (nodes) that participated in the rollup.
`SUM`	The sum of all the values (or samples) recorded in the time unit.

Observed Value

The Observed Value is the default value that is computed and displayed on the user interface. This table shows formulas for the observed value for **Cluster rollup** based on **Individual** and **Collective** clusters. The **Time Roll-up Type** displays how the metrics in the **Controller** map to the metrics in Cloud Native Visualization.

Time Rollup Type	Cluster rollup - Individual	Cluster rollup - Collective
Average	`SUM/COUNT`	`(SUM * GROUP_COUNT) / (COUNT *DEFAULT_GROUP_COUNT_MULTIPLIER(100))`
Sum	`(SUM / GROUP_COUNT) * DEFAULT_GROUP_COUNT_MULTIPLIER(100)`	`SUM`
Current	`(CURRENT / GROUP_COUNT) * DEFAULT_GROUP_COUNT_MULTIPLIER(100)`	`CURRENT`

Metric Time Series

This table illustrates how Metric Data Resolution over Time is displayed in the Cloud Native Visualization user interface:

> These are the initial numbers and are subject to change.

Query Time Range	Ingestion Granularity	Response Granularity	Max Number of Data points Per Metric
(<) 1 day	1 Min	1 Min	1440
(<) 1 day	5 Min	5 Min	288
2 days	1 Min	5 Min	1440
2 days	5 Min	5 Min	576
3 days to 8 days	1 Min, 5 Min	5 Min	864
3 months - 1 Year	1 Min, 5 Min	1440 Min (1 day)	365
8 days	1 Min, 5 Min	10 Min	1152
9 days - 3 Months	1 Min, 5 Min	60 Min	72

About Metric Visualization Tools

Cloud Native Visualization offers an Object page, which is the main interface that allows you to inspect metrics, even though metric data appears throughout the Controller. The Relationships and Interactions maps display metric values collected over a selected time period. In the browser, you can visualize and APM and Amazon CloudWatch metrics to compare and analyze patterns.

For understanding how AppDynamics ingests the AWS detailed monitoring data for the metrics with variable granularity, see Variable Ingestion Granularity for AWS Detailed Monitoring.

Variable Ingestion Granularity for AWS Detailed Monitoring

As part of the Cloud Native launch program, early adopters will have access to the Cloud Native Visualization functionality on a trial basis. AppDynamics reserves the right to charge separately for Cloud Native Visualization after the initial trial period.

AppDynamics Cloud Native Visualization uses the Amazon CloudWatch API to collect metrics related to AWS Cloud Computing Services. When working with the AppDynamics platform, consider:

- AWS offers 5-minute and 1-minute granularity for metrics. You can change the granularity for a subset of the AWS CloudWatch metrics in a few AWS services from 1-minute to 5-minute granularity and vice versa.
  For example, Amazon EC2 allows customers to configure detailed monitoring, offered at 1-minute granularity and basic monitoring, offered at 5-minute granularity on a subset of EC2 metrics. The AppDynamics platform presents such metrics with variable granularity at a minimum granularity of five minutes. Other metrics that are either at 1-minute or 5-minute granularities will be presented at their respective granularities.
  An administrator can enable detailed monitoring on a specific EC2 instance in AWS. In a scenario where an administrator enables detailed monitoring on a specific EC2 instance in AWS for debugging, CloudWatch starts reporting data for this instance at a 1 minute granularity. After the administrator debugs the issue, the administrator turns off Detailed monitoring. CloudWatch now reports data every 5 minutes because of the default Basic monitoring.
  With Cloud Native >= 1.1, you can view this data on the Cloud Native Visualization interface with both 1-minute and 5-minute granularity.
  For example, when a AWS user is monitoring 10 EC2 instances (i-01 to i-10).

  > When a 1-minute granularity metric reports for a time range 10:00 to 10:01, the timestamp is 10:00 and granularity is 1 minute. However, a 5-minute granularity metric reported for time-range 10:05 to 10:10, the timestamp is 10:05 and granularity is 5 minutes.

  For the time period between 10:00 am to 10:15 am, all EC2s report at Basic monitoring (5-minute granularity). Therefore, there are three 5 minute data points timestamped at 10:00, 10:05, and 10:10. Between 10:15 and 10:20, the user changes the monitoring mode for two instances: i-05 and i-07 to Detailed monitoring (1-minute granularity). During this period, still, one 5-minute data point timestamped 10:15 is observed. Again, the user changes the AWS monitoring mode back to Basic monitoring between 10:20-10:45 for the two instances: i-05 and i-07. Here, 5-minute data points are timestamped: 10:20, 10:25, 10:30, 10:35, 10:40 for each instance with Basic Monitoring enabled. The following 1-minute data points are observed for the two instances i-05 and i-07:

  - 10:20+ (timestamped 10:20, 10:21, 10:22, 10:23, 10:24)
  - 10:25+ (timestamped 10:25, 10:26, 10:27, 10:28, 10:29)
  - 10:30+ (timestamped 10:30, 10:31, 10:32, 10:33, 10:34)
  - 10:35+ (timestamped 10:35, 10:36, 10:37, 10:38, 10:39)
  - 10:40+ (timestamped 10:40, 10:41, 10:42, 10:43, 10:44)

  When all monitoring returns to Basic monitoring, the platform switches to capture one 5-minute data point for each metric at 5-minute granularity.
- The AppDynamics platform floor metrics offered at a 5-minute granularity rounds up to the nearest 5th-minute timestamp.
  As an example, a 5-minute granular metric with a timestamp in AWS CloudWatch of "2020-08-20T07:47:00+00:00" and value 200 will have a timestamp of "2020-08-20T07:45:00+00:00" with value 200.
- AWS makes CloudWatch metrics available with some delays of up to 15 minutes. Therefore, you may encounter delays in metric availability on AppDynamics Cloud Native Visualization. Although AWS might update Amazon CloudWatch metrics after they are made available, AppDynamics uses the metric values that were made available at the time of reading.

  Due to these reasons, you may observe discrepancies between AWS CloudWatch metrics and those presented by AppDynamics Cloud Native Visualization.

See Cloud Native Visualization Metrics Overview.

Use Node Name Reuse for Java Agents in Cloud Native Visualization

As part of the Cloud Native launch program, early adopters will have access to the Cloud Native Visualization functionality on a trial basis. AppDynamics reserves the right to charge separately for Cloud Native Visualization after the initial trial period.

The app, tier, node model has been at the heart of AppDynamics APM since its inception. Cloud Native Visualization slightly modifies this model. In particular, what was formerly known as tiers are now called **Services**, and the former nodes are now called **Service Instances**.

A tier represents a set of identical nodes, often load-balanced, so renaming tiers as services better reflects the intent of the construct. However in the case of the named concept node, the implications of the data model change are more complicated than simply renaming node to service instance.

Nodes

Each named node in AppDynamics APM represents a placeholder into which metrics are published when an instrumented application process is started with that node name. The node name identifies the "reporting slot". This model works well for cloud scale-up and scale-down use cases where some bounded elastic capacity is provided and provisioned on a fixed set of infrastructure resources.

This model does not work well with purely ephemeral nodes. Purely ephemeral nodes operate in a space where the whole environment from the operating system up the stack to the application code is spun up once, and only once, and is destroyed at the end of its working life. This is the definition of an ephemeral node, it is not meant to be static, nor long-lived.

To ease node naming issues in such ephemeral use cases, AppDynamics provides capabilities for node name reuse. Node name reuse marks nodes as "historical", allowing uniquely named placeholders to be re-used by different nodes over time. See Historical and Disconnected Nodes.

Service Instances

In contrast to a node, the service instance concept represents the process lifecycle of one application process, with an explicit lifetime. To maintain the proper service instance lifecycle in the Cloud Native Visualization application model a new node naming mechanism has been provided in the Java Agent. This naming mechanism allows service instances to be populated with data gathered from the traditional AppDynamics agents. The AppDynamics system automatically generates a node name for each process as it starts and clears the associated node history from the Controller database once it shuts down. This approach allows the Controller to track the lifecycle of each process while avoiding a build-up of historical nodes over time. Each node maps to a unique service instance with its own startup and shutdown time in the cloud native environment.

Dynamic Environment Considerations

An AppDynamics app agent operates in a dynamic environment much the same way it does in a traditional data center. The auto-naming behavior that defines a node name in the `controller-info.xml` file is optional. The node name can be arbitrary and is implemented for backward compatibility. This helps you understand when a node was started and when it stopped.

This table outlines the comparison of the ephemeral service instance (node) feature with the service instance (node) name reuse feature:

Ephemeral Service Instance (Node)	Service Instance (Node) Re-use and Mark as Historical
Pros • Node history is better retained in Cloud Native Visualization because the service instance (node) names are unique across container instance restarts. • Each service instance (node) or JVM is guaranteed to receive a unique service instance (node) name. • No need for a shutdown hook, JVM can be killed without shutdown hook invocation.	**Pros** This is the existing AppDynamics functionality and previously recommended approach.
Cons Historical data relating to churned ephemeral nodes is purged from the controller and can only be viewed for retrospective failure analysis in the Cloud Native Visualization.	**Cons** • This method is slated for deprecation. • Relies on a shutdown hook that must be invoked. If the shutdown hook is not invoked, the service instance (node) is not purged from the Controller. Nodes are very often forcibly terminated in container environments, causing nodes to not be purged. • Cloud Native service instance entities can incorrectly represent multiple different processes sharing a node name.

Configure the Agent for Dynamic Environments with Ephemeral Service Instance (Node) Auto Naming

You can configure the Java Agent to automatically name service instances (nodes) based on the platform. For automatic service instance naming to work, you must specify an application name and service (tier) name.

The ephemeral service instance (node) property is useful for monitoring environments where there are many JVMs with a short life span. When set to true, the service instance (node) name format is shown as:

```
<Hostname>-<Date time>-<5-character random string>
```

The property `-Dappdynamics.agent.node.use.as.ephemeral=true` (or the equivalent environment variable/XML tag) registers the service instance (node) as ephemeral.

Set this property to true to mark service instances (nodes) as ephemeral. You do not need to specify the service instance (node) name in `controller-info.xml` (<node-name></node-name> or as a system property (`-Dappdynamics.agent.nodeName`).

An example of how the service instance name is displayed:

```
jbossserver.local-2019-10-07 12:13:27.000132-JrQdU
```

The property for the Java Agent is as shown:

System Property: `-Dappdynamics.agent.node.use.as.ephemeral=true`

Environment Variable: `APPDYNAMICS_JAVA_AGENT_EPHEMERAL_NODE`

in controller-info.xml: `<use-as-ephemeral-node/>`

Analytics

You can use Analytics with the APM, Browser RUM, Mobile RUM, and Browser Synthetic Monitoring product modules for:

- Transaction Analytics
- Log Analytics
- Browser Analytics
- Mobile Analytics
- Browser Synthetic Analytics
- Connected Devices Analytics

Analytics extracts the data, generates baselines and dashboards, and provides perspective beyond traditional APM by enabling real-time analysis of business performance correlated with your application performance.

Installation and Administration

- Deploy Analytics Without the Analytics Agent
- Deploy Analytics With the Analytics Agent
- Analytics and Data Security
- Upgrade Analytics Agent
- Upgrade Platform Components
- Troubleshoot Analytics Issues

Analytics Data Collection

- Collect Log Analytics Data
- Collect Transaction Analytics Data
- Business Transaction and Log Correlation
- Collect Business Data From SQL Calls

Using Analytics

- Business Journeys
- Experience Level Management
- Search Analytics Data
- Visualize Analytics Data

Extensibility and Reference

- Create Analytics Metrics From Scheduled Queries
- Analytics Events API
- ADQL Reference

Overview of Analytics

This page describes Analytics. Analytics is built on the AppDynamics Application Performance Management (APM) platform, which includes the Events Service, the unstructured document store for the platform.

Analytics can answer business-oriented questions such as:

- How many users experienced failed checkout transactions in the last 24 hours?
- How much revenue was lost because of these failures?
- How is the lost revenue distributed across different product categories?
- What is your revenue for the day for a geographical region?
- What was the revenue impact, by product category, associated with the two marketing campaigns we ran last week?

The exact components you need to install depend upon your environment and your requirements.

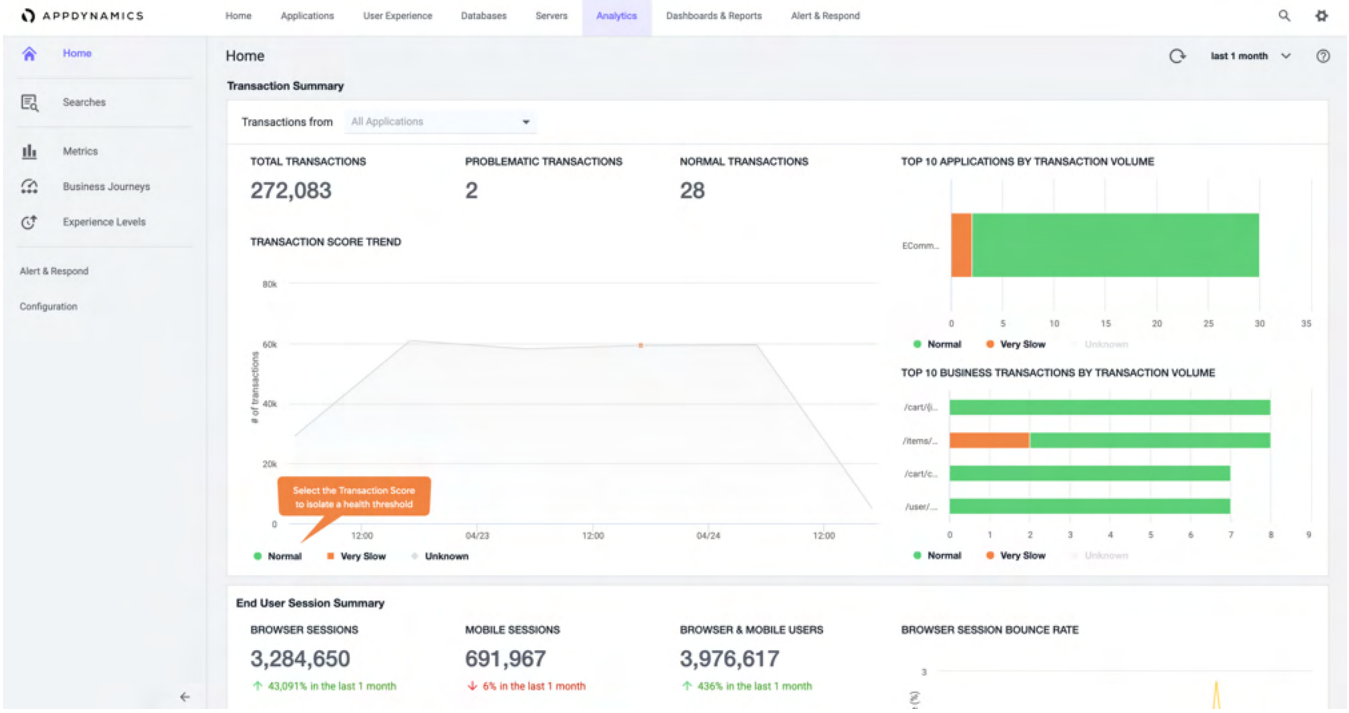
- To plan your deployment, see Deploy Analytics With the Analytics Agent.
- If you are upgrading from a previous version, see Upgrade Analytics Agent.
- If you have been using job files for Log Analytics and want to switch to using source rules (part of Configuration Log Management), see Migrate Log Analytics Job Files to Source Rules.
- To view Analytics Agents connected to your Controller, see View Analytics Agent Health.

Analytics Home Page

The **Analytics Home** page consolidates data from the transaction, browser, and mobile events. The **Home** page automatically generates **Transaction** and **End User Monitoring Summary** panels through queries that aggregate data into widgets.

To view the different widgets on the home page, you need the appropriate licenses and access. See Restrictions and Security.

You can access the AppDynamics home page by clicking the **Home** icon on the left navigation pane in **Analytics**. You can either use the left navigation pane, or click **Home** on the right pane to navigate to the Analytics modules (**Searches**, **Metrics**, **Business Journeys**, **Experience Levels**, **Alert & Respond**, and **Configuration**).



Home Page Data

AppDynamics Query Language (ADQL) queries are used to generate the data for each home page widget. You cannot access or edit the ADQL queries.

When you select a time range, each query automatically includes the time filter for the widgets. These widgets refresh every two minutes.

Transaction Summary

The **Transaction Summary** panel aggregates the Analytics data from applications and business transactions. **Transaction Summary** widgets are categorized by Transaction Threshold from normal to error. The **Transaction Summary** panel contains six widgets:

- **Total Transactions** - Count of all transactions.

- **Problematic Transactions** - Count of transactions that are not within normal threshold boundaries.
- **Normal Transactions -** Count of transactions that are within the normal threshold boundaries.
- **Top 10 Applications by Transaction Volume** - Graphical view of applications.
- **Top 10 Business Transactions by Transaction Volume** - Graphical view of transactions.
- **Transaction Score Trend** - Total number of transactions for all applications.

End User Session Summary

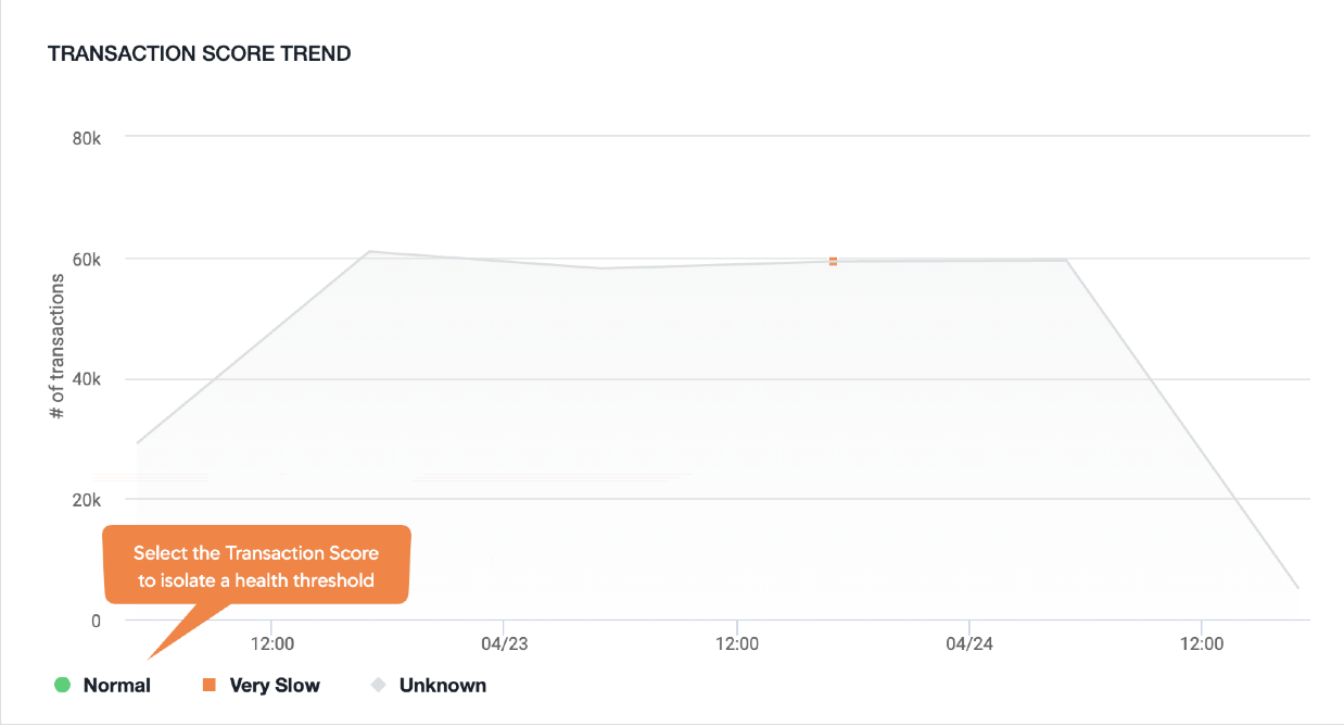
The **End User Session Summary** panel displays browser and mobile session data in widgets. The **End User Session Summary** panel contains six widgets:

- **Browser Sessions** - Count of unique browser sessions.
- **Mobile Sessions** - Count of unique mobile sessions.
- **Browser & Mobile Users** - Count of browser and mobile users.
- **Browser Session Bounce Rate** - Rate of bounces per total sessions.
  A bounce is a browser session where the page count is one. Bounces occur when a user opens a webpage and leaves the site without visiting additional pages.
- **Top Page Referrers by Session Volume** - Website visited by users before accessing your website. The referrer is an optional component of the Events Service. If you have not configured referrers, this widget does not contain data.
- **Session Trend** - Total number of sessions separated by browser and mobile sessions.

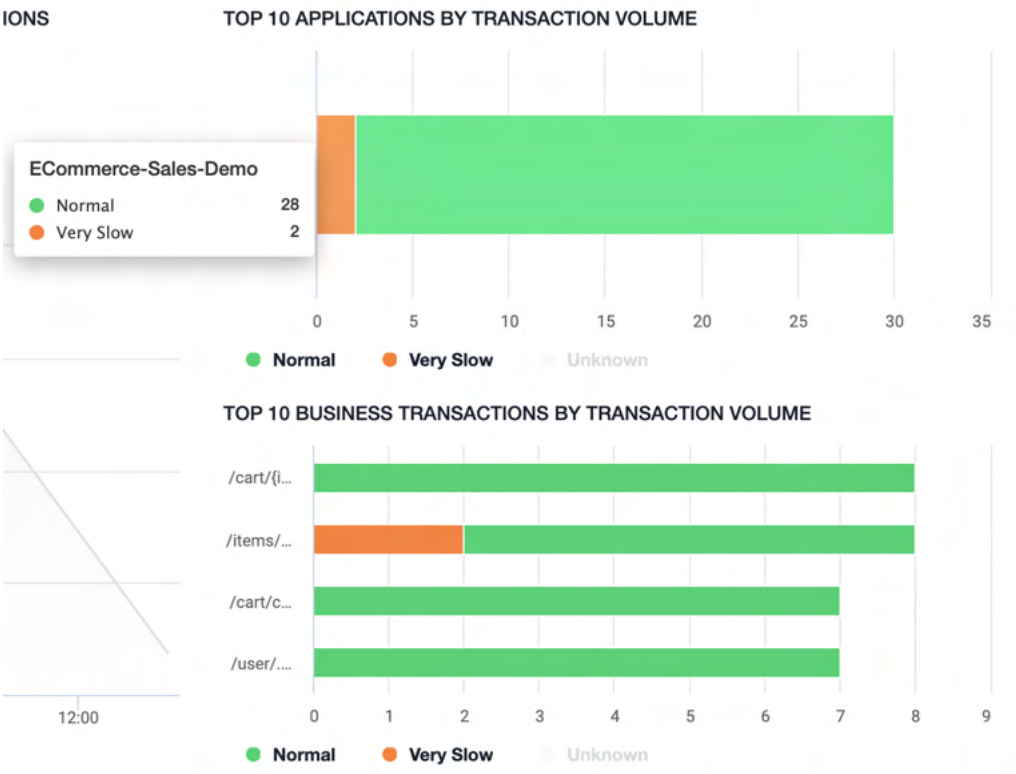
Options for Home Page Data

Transaction Score Options

You can select the transaction score in each chart to isolate metrics. For example, selecting **Normal** in **Transaction Score Trend** removes other transaction scores:

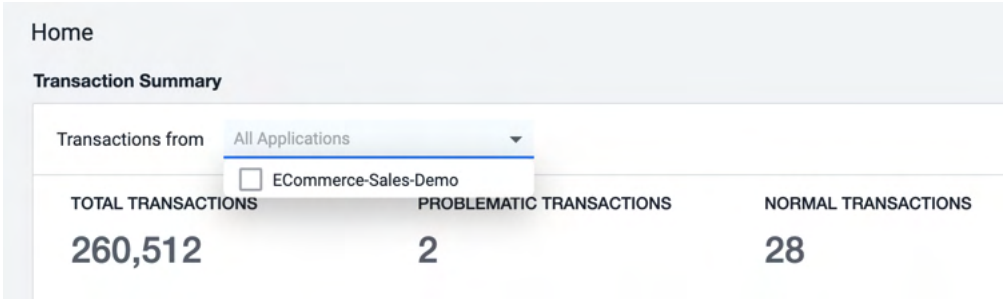


Next to the **Transaction Score Trend** are two horizontal bar graphs that display the top ten applications and business transactions by transaction volume. Hover over the graphs to see a breakdown of additional transaction information.



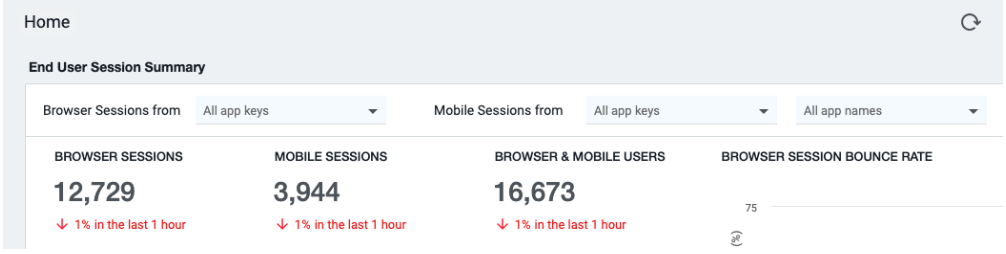
Transaction Filter Options

You can filter the view of each panel to display specific applications or application keys. Click the dropdown arrow at the top of each panel and select the desired fields.



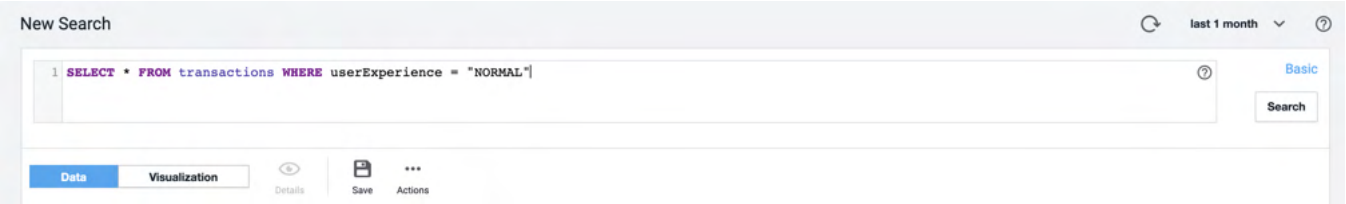
End User Monitoring Filter Options

You can filter browser sessions and mobile sessions to display application keys. For mobile sessions, selecting specific application names further filters the application keys.



Drill Down Options

You can drill down into each metric to open the ADQL search that generates the metric. For example, selecting **Normal Transactions** automatically displays the ADQL Query for normal transactions.



Restrictions and Security

If your **Analytics Home** page does not display data on widgets, there is either an error in retrieving data, or no data is available in the Events Service.

If panels or widgets are missing from your home page, confirm that you have sufficient licenses and have access to Analytics data.

Licenses

To view data on the **Transaction Summary** panel, you need:

- At least one application enabled for Analytics
- Active Transaction Analytics licenses

If your Transaction Analytics license expires, Analytics stops ingesting data for the **Analytics Home** page. The **Transaction Summary** panel still displays your existing data, but does not ingest any new data.

To view data in the **End User Session Summary** panel, you need:

- Browser Analytics and/or Mobile Analytics enabled. See Analytics Data Sources.
- Active EUM licenses to Mobile and/or Browser Real User Monitoring.

Some widgets may not be available depending on your End User Monitoring (EUM) licenses. For example, if you have a license to Mobile RUM only, your Analytics Home page will not contain browser-specific widgets (browser sessions, bounce rate, and referrers). Aggregated widgets such as Browser & Mobile Users and Session Trend only show the data to which you have an EUM license.

RBAC

Analytics enforces role-based access control (RBAC). As a result of RBAC, your **Analytics Home** page displays only the data to which you have access. See Analytics and Data Security.

In the **End User Session Summary** panel, numeric widgets and the **Session Trend** chart requires access to both Browser and Mobile Real User M these widgets, you must have access to **Browser Requests** and **Mobile Requests and Sessions** events.

ADQL Reference

The AppDynamics Query Language (ADQL) is a query language for searching available data in Analytics.

ADQL is similar to the SELECT statement in the prevalent Structure Query Language (SQL) but is designed solely for AppDynamics Analytics data. Just like the SELECT command in SQL, ADQL allows you to specify the data source (such as transaction events, EUM events, and logs), a list of fields to retrieve, and conditions for selecting events from the analytics data store.

With ADQL, you can construct queries in these environments:

- Analytics Search panel in the Controller with **Advanced Mode**.
- Analytics Query Events APIs in the body of the query request.

To simplify entering your queries in Analytics Search with **Advanced Mode**, the ADQL auto-complete feature displays keywords, a list of available fields, and functions based on the stage of your typed search. See Creating Advanced Searches for Analytics.

ADQL Typographical Conventions and Symbols

ADQL reference topics use custom typographical conventions and symbols to describe the syntax for ADQL queries.

Convention	Name	Usage
BOLD	Bold typeface	Bold typeface indicates ADQL keywords and operators, such as **SELECT**, **FROM**, **AND** and so on.
field_name and numeric_field_name	N/A	The terms field_name and numeric_field_name represent field names that appear in your analytics data. Field names depend on the event type. See ADQL Data.
`Monospace` typeface	Monospace typeface	Function names, such as count are indicated by `monospace` typeface.
event_type	N/A	The term event_type represents the available event types collected from your applications. Valid values are: • transactions • logs • browser requests • mobile requests • mobile crash reports • analytics custom events • synthetic sessions
Italic typeface	*Italics*	*Italics* represent variables or placeholders. You supply the actual value.
[ ]	Square brackets	Terms in square brackets are optional. Nested square brackets indicate that the term inside is optional and can only be included if the optional term in the outer square brackets is included.
|	Pipe	The pipe character separates alternate items to indicate one of the items may be included. You can read it as "or" in a syntax statement.
{ }	Curly braces	Curly braces are used to group items to disambiguate the definition of items.
( )	Parentheses	Parentheses are used to indicate the order of operation in an expression.
...	Ellipsis	An ellipsis indicates the preceding term may be repeated.

ADQL Queries

This page describes ADQL query syntax. ADQL uses the **SELECT** statement and filtering statements, otherwise referred to as queries, to return sets of data that you can add to **Custom Dashboards**.

ADQL search queries have a 5,000 character limit and reserve keywords in certain field names or milestones.

Reserved ADQL Keywords
SELECT
FROM
WHERE
ORDERBY
AND
OR
EXISTS
NOT
LIKE
REGEX
COMPARE
LIST
IN
BETWEEN
COUNT
DISTINCT
MIN
MAX
STATS
SUM
AVG
STDDEV
FILTER
PERCENTILE
RANGE
SERIES
SUBSTRING
INDEXOF
LENGTH
TRIM
CONCAT
NOW

The query syntax is:

```
SELECT * | {[DISTINCT] expression [AS alias] [, expression [AS alias]]...}
FROM event_type
[WHERE condition_expression]
[SINCE timevalue [UNTIL timevalue]]
[HAVING condition_expression]
[ORDER BY {field_name | alias} [ASC | DESC] [, {field_name | alias} [ASC | DESC]]...]
[LIMIT integer [, integer]...]
```

While the **SELECT** and **FROM** clauses are required, all other clauses are optional. Enter spaces around each keyword.

For **GROUP BY** functionality, see GROUP BY.

SELECT Clause

The ADQL event query begins with a **SELECT** clause, which can take one or more arguments.

SELECT * | {[**DISTINCT**] field_expression [**AS** alias] [, field_expression [**AS** alias]]...}

- A single asterisk * in the **SELECT** clause returns the list of fields for each event.
- To return specific fields for each event, specify a comma-separated list of fields. Optionally, each field_expression can be aliased to a label using the **AS** construct.
- Use the **DISTINCT** qualifier to return unique combinations of the specified field values. If no limit specified, the top ten results by count are returned. See LIMIT Clause.
  DISTINCT can not be applied to a wild card **SELECT** query, such as

  ```
  SELECT DISTINCT *
  ```

- String values must be surrounded by either single quotes or double quotes, such as 'hello' and "hello".

The field_expression can take several forms. See ADQL Expressions.

Query Examples with SELECT Clause

Query	Result
SELECT `count`(transactionName) **FROM** transactions **WHERE** application='*yourAppName*'	Returns a count of business transactions for the specified application.
SELECT * **FROM** logs **WHERE** sourceType='*yourLogFile*' **AND** message **LIKE** '*GET*'	Returns log events containing a specified keyword.
SELECT * **FROM** transactions **WHERE** application='*yourAppName*' **AND** transactionName='*yourBTName*'	Returns all fields for business transactions in the specified application matching the specified transaction name.
SELECT DISTINCT transactionName **FROM** transactions **WHERE** application='*yourAppName*'	Returns a list of unique transaction names from the specified application.
SELECT *customerName* **AS** "First And Last Name", *age* **AS** "Years Old", ...	Returns the specified fields relabeled as specified in the query statement. In this query, *customerName* and *age* are user-defined fields.

AS Clause

Use the **AS** clause to rename a field to a label of your choice. The label is also referred to as an alias.

field_expression **AS** alias

Returns the field with the specified alias. The alias may be any valid identifier (a letter followed by any number of letter or digit characters) or a value in quotes. If the value of the alias contains spaces or special characters, it must be in quotes. See ADQL Expressions.

AS Clause Example

SELECT eventTimestamp **AS** 'Time Stamp' **FROM** transactions **WHERE** application='*yourAppName*'

FROM Clause

Use the **FROM** clause to specify the event type. For example:

FROM event_type

Event Types

Event types describe the source of the analytics data, such as business transactions (transactions) or browser requests. Each event type has a UI field name and an Events Service internal name. The Events Service internal name is used in ADQL queries.

This table lists the event types with their UI names and Events Service internal names.

Event Type UI Label	Events Service Internal Name
Transactions	transactions
Logs	logs
Browser Requests	browser_records
Browser Sessions	session_records
Mobile Requests	mobile_snapshots
Mobile Sessions	mobile_session_records
Mobile Crash Reports	mobile_crash_reports
Synthetic Sessions	synth_session_records

FROM Clause Examples

For this result	Query
Return all fields for browser requests	**SELECT** * **FROM** browser_records **WHERE** appkey='E2E-AAB-AUM'
Return all fields for the business transactions in your application	**SELECT** * **FROM** transactions **WHERE** application='*yourAppName'*
Return all fields for the logs specified by "*yourLogFile*"	**SELECT** * **FROM** logs **WHERE** sourceType='*yourLogFile*'

WHERE Clause

Overview

Use the **WHERE** clause to specify one or more condition expressions separated by logical operators such as **AND**, **OR.**

The **WHERE** clause uses the following syntax:

```
[WHERE condition_expression]
```

See ADQL Expressions for a description of the condition expression syntax. Condition expressions use Comparison Operators.

> AppDynamics recommends that you do not use periods (.) in the field name, otherwise, the syntax will break.

Examples

To filter logs based on error or warning log level, use a query similar to the following:

```
SELECT * FROM logs WHERE sourceType='yourLogFile' AND (logLevel='ERROR' OR logLevel='WARN')
```

GROUP BY

ADQL does not have an explicit **GROUP BY** clause. However, queries containing field names in the **SELECT** clause behave as **GROUP BY** statements if the query contains metric functions or bucketing functions.

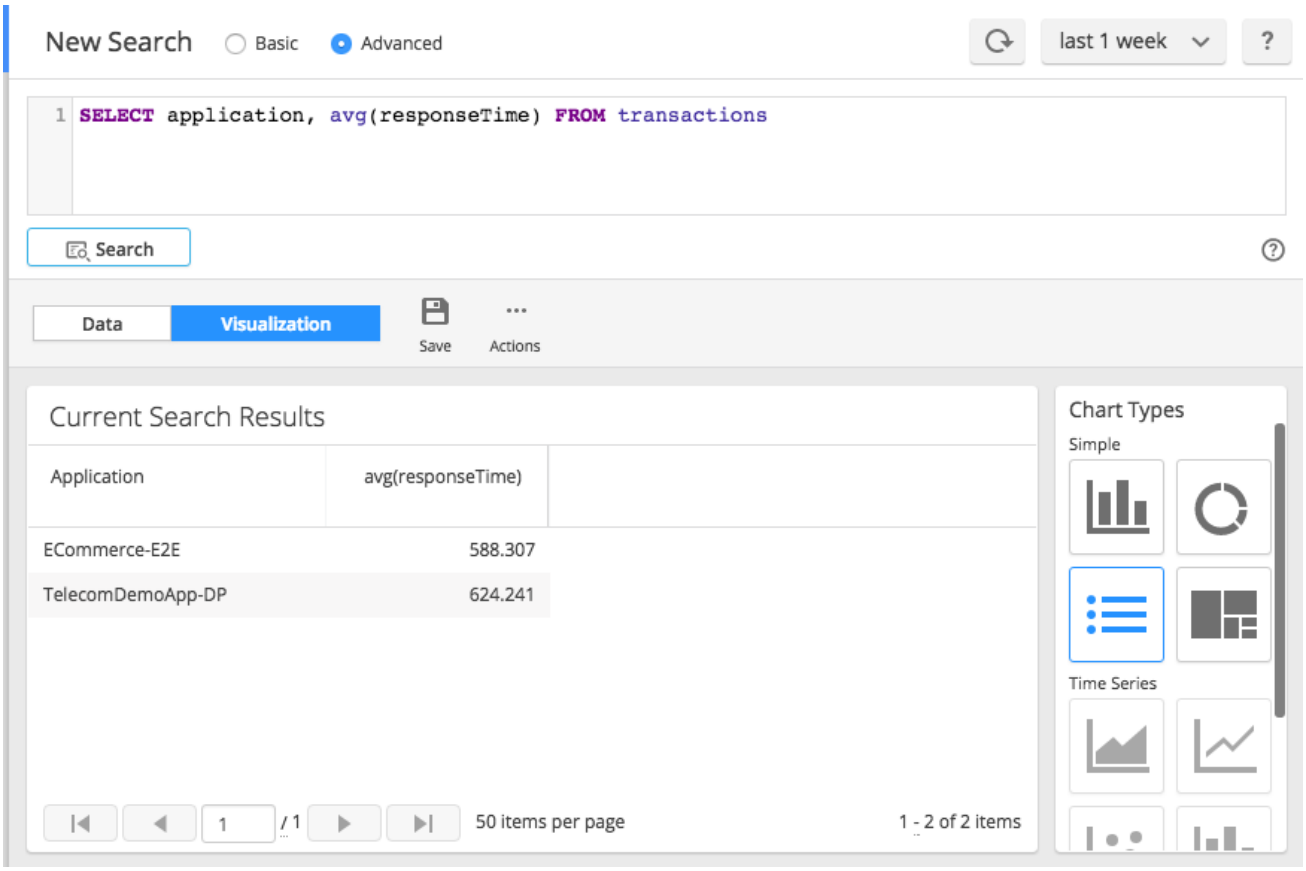
Group by queries are limited to containing up to five terms. Terms can be either fields (which use an implicit **GROUP BY**) or bucketing functions, such as `series`.

Aggregation Examples

In this example, the query groups the `avg(responseTime)` aggregation based on the non-aggregation field, application. So the result contains the average response time calculated for each application.

SELECT application, `avg(responseTime)` **FROM** transactions

Sample results:

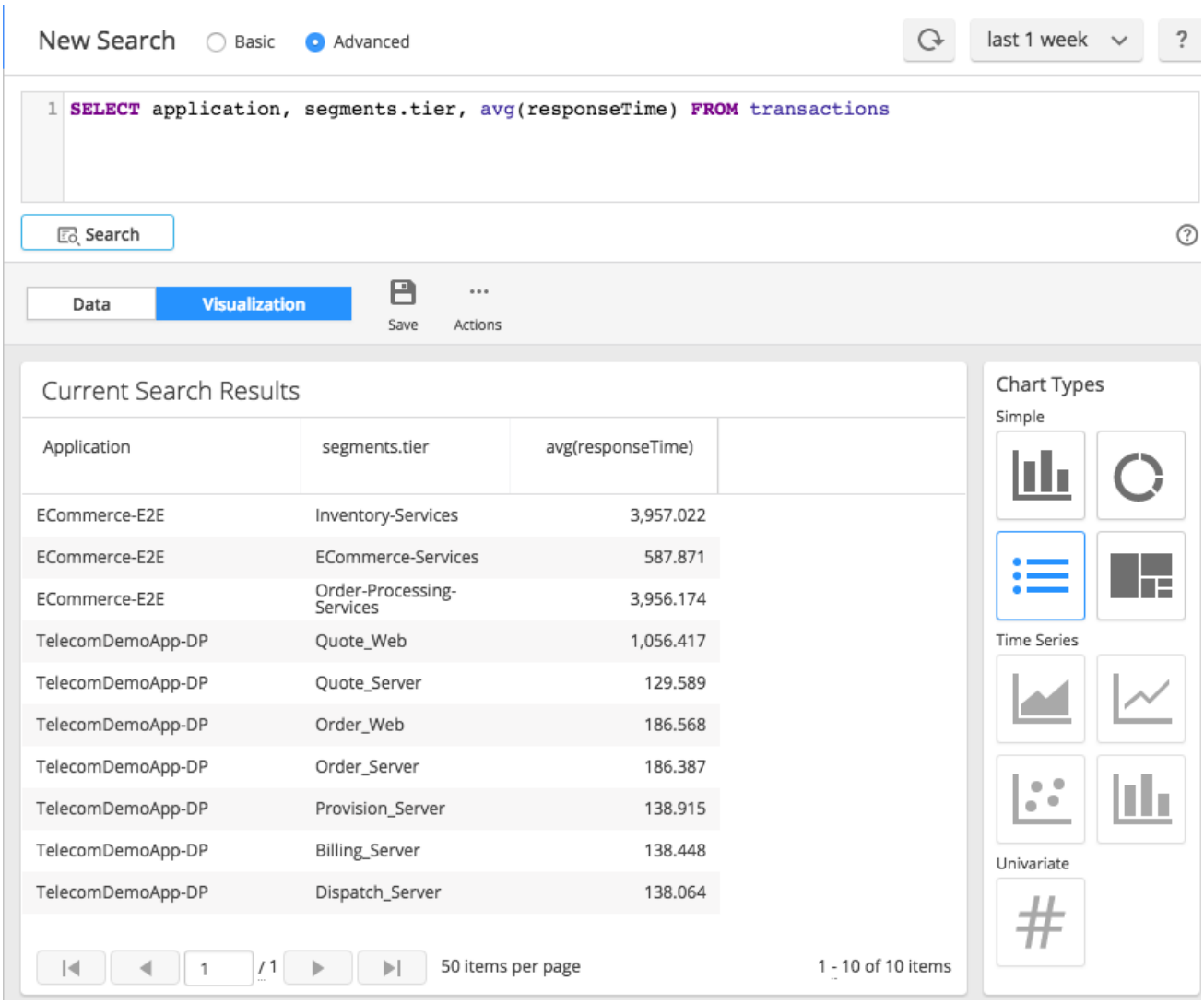


The implicit grouping can be extended by using multiple non-aggregation fields preceding an aggregation field. For example:

SELECT application, segments.tier, `avg(responseTime)` **FROM** transactions

By adding tier as a second non-aggregation field, the query groups the `avg(responseTime)` aggregation based on the non-aggregation fields, application, and tierName. Because there are multiple grouping fields, the second non-aggregation field, tier name, is sub-grouped under the application grouping. So each combination of application and tier name together forms a group over which the average response time is calculated.

Sample results:



For SQL users, these two statements produce similar results in their respective environments:

- ADQL query: **SELECT** application, segments.tier, `avg`(responseTime) **FROM** transactions
- SQL query: **SELECT** application, segments.tier, `avg`(responseTime) **FROM** transactions GROUP BY applicationName, segments.tier

Bucketing Function Example

Bucketing functions such as `series` and `range` define groups over which other aggregations can be calculated. The buckets are specified by the parameters of the respective functions and form groups over which subsequent aggregations are calculated.

For example, the following query calculates the average response time over two-minute intervals (or buckets):

SELECT `series`(eventTimestamp, '2m'), `avg`(responseTime)

Sample results:

```
1 SELECT series(eventTimestamp,'2m'), avg(responseTime) FROM transactions WHERE application = 'ECommerce-E2E'
```

Search

Data | Visualization | Save | Actions

series(eventTimest...	avg(responseTime)
08/22/16 3:40:00 PM	46.172
08/22/16 3:42:00 PM	56.328
08/22/16 3:44:00 PM	44.537
08/22/16 3:46:00 PM	50.068
08/22/16 3:48:00 PM	50.157
08/22/16 3:50:00 PM	43.767
08/22/16 3:52:00 PM	51.759
08/22/16 3:54:00 PM	37.514

You can also combine an aggregation with a bucketing function and implicit **GROUP BY**. The following query returns the average response time grouped by tier over the series of six-hour intervals.

SELECT `series`(eventTimestamp, '6h'), segments.tier, `avg`(responseTime) **FROM** transactions

Sample Results:

```
1 SELECT series(eventTimestamp, '6h'), segments.tier, avg(responseTime) FROM transactions
```

Search

Data | Visualization

Save | Actions

Current Search Results

series(eventTimest...	segments.tier	avg(responseTime)
08/22/16 11:00:00 AM	ECommerce-Services	41.889
08/22/16 11:00:00 AM	Inventory-Services	251.444
08/22/16 11:00:00 AM	Order-Processing-Services	251.444
08/22/16 5:00:00 PM	Inventory-Services	4,009.542
08/22/16 5:00:00 PM	ECommerce-Services	588.622
08/22/16 5:00:00 PM	Order-Processing-Services	4,009.542
08/22/16 11:00:00 PM	ECommerce-Services	542.803
08/22/16 11:00:00 PM	Inventory-Services	5,665.215
08/22/16 11:00:00 PM	Order-Processing-Services	5,665.215
08/23/16 5:00:00 AM	ECommerce-Services	9.984
08/23/16 11:00:00 AM	ECommerce-Services	10.580

ORDER BY Clause

The **ORDER BY** clause enables you to sort the query results by one or more fields. The syntax is:

[**ORDER BY** {field_name | alias} [**ASC** | **DESC**] [, {field_name | `alias`} [**ASC** | **DESC**]]...]

Rules governing ordering include:

- Sort order, ascending (**ASC**) or descending (**DESC**) is applied per field. If not specified, the ascending (**ASC**) order is the default.
- **ORDER BY** is supported on fields or aliases specified in the **SELECT** clause.
- Aliases used in the **ORDER BY** clause should be surrounded with back quotes if they contain spaces or other special characters.
- **ORDER BY** works on single-value metric functions and single-value math expressions. **ORDER BY** can not be used with percentile or stats.
- **ORDER BY** on nested fields or metric aggregations that operate on nested fields is not supported. A nested field is any field that has a "." in the name, such as `segments.errorList.errorType` or `btdata.estimatedtime.`

Ordering functionality for metric and bucketing functions includes:

- **ORDER BY** can be used to order the buckets on a bucketed field in a bucketing function. For example, see the "Series aggregations - Ordering based on the keys of the series buckets." in the table below.
- **ORDER BY** can be used on metric functions that follow bucketing functions, including filtered metric functions. Ordering by the metric value orders the resulting buckets. The metric function must be aliased and referenced by its alias such as:

```
SELECT transactionName, avg(responseTime) AS averageResponseTime FROM transactions ORDER BY
averageResponseTime
```

- Compound orderings can be specified for functions that provide implicit **GROUP BY** functionality.
- A single ordering can be defined for the series function, no ordering is allowed for the range function.
- **ORDER BY** can accept math expressions on fields, but not math expressions on metric functions. If the value of a math expression is null it is ordered at the end of the results in ascending order. The following is an invalid query:
SELECT transactionName, `max`(discountValue) / `max`(cartTotal) **AS** ratio **FROM** transactions **ORDER BY** ratio

ORDER BY Clause Examples

For This Result	Query
An ordering of queries with no aggregations	**SELECT** field_name **FROM** event_type **ORDER BY** field_name **ASC**
Group By - Ordering by the lexicographic ordering of the terms. The fields included in the **ORDER BY** clause must be in the same order as in the query itself.	**SELECT** field_name, `count`(*) **FROM** event_type **ORDER BY** field_name **DESC**
Returns a sorted set of timestamp ranges	**SELECT** `series`(eventTimestamp, 10m), `count`(*) **FROM** transactions **ORDER BY** eventTimestamp **DESC**
Series aggregations - Ordering based on the keys of the series buckets.	**SELECT** `series`(salary, 20000), `count`(salary) **FROM** event_type **ORDER BY** salary **DESC**
Date series aggregations - Ordering based on the date value of the keys of the series buckets	**SELECT** `series`(eventTimestamp, '1h'), `count`(eventTimestamp) **FROM** transactions **ORDER BY** eventTimestamp
Order by an alias.	**SELECT** responseTime **AS** RT **FROM** transactions **ORDER BY** RT
Order by an aliased math expression.	**SELECT** discountValue / cartTotal **AS** ratio **FROM** transactions **ORDER BY** ratio

HAVING Clause

Use the **HAVING** clause on the output produced by another aggregation. The **HAVING** clause operates on an expression to determine the intervals to be shown in the response. The metric specified in the expression must be numeric and the expression must return a boolean value for the **HAVING** clause to work.

Additionally, the **HAVING** clause must be referenced by an alias in the query as follows:

```
SELECT customerName, count(*) as Requests, avg(responseTime) as ResponseTime FROM transactions HAVING Requests
> 10000
```

where `Requests` is the alias referenced in the query.

Syntax

```
SELECT selectItems
   FROM relation
   WHERE where=booleanExpression
   SINCE statement
   HAVING havingClause
   ORDER BY sortItems
   LIMIT limits
  ;
```

where `havingClause` is a boolean expression. The **WHERE**, **ORDER BY**, and **LIMIT** clauses and SINCE `statement` are optional.

HAVING Clause Examples

HAVING clause examples with different filters:

Simple Comparison Filter

```
SELECT transactionName, avg(responseTime) as AVRT FROM type HAVING AVRT > 0
```

Simple Range Filter

```
SELECT transactionName, avg(responseTime) as AVRT FROM type HAVING AVRT BETWEEN [10, 90)
```

Simple List Filter

```
SELECT transactionName, avg(responseTime) as AVRT FROM type HAVING AVRT IN (20, 30, 40, 60)
```

Compound Filter

```
SELECT transactionName, avg(responseTime) as AVRT FROM type HAVING AVRT > 20 AND AVRT < 90
SELECT transactionName, avg(responseTime) as AVRT FROM type HAVING AVRT < 20 OR AVRT > 90
SELECT transactionName, avg(responseTime) as AVRT FROM type HAVING AVRT > 20 AND AVRT < 90 AND AVRT IN (70, 80)
```

Not Filter

```
SELECT transactionName, avg(responseTime) as AVRT FROM type HAVING NOT AVRT > 50
```

Not Compound Filter

```
SELECT transactionName, avg(responseTime) as AVRT FROM type HAVING NOT (AVRT > 20 AND AVRT < 90)
```

Multi-Metric Filter

```
SELECT transactionName, avg(responseTime) as AVRT, max(responseTime) as MAXRT FROM type HAVING NOT (AVRT > 20
AND AVRT < 90) AND MAXRT > 130
```

Limitations

The **HAVING** clause is not supported for:

- Nested fields in Elasticsearch
- Multi-metric functions, such as `percentiles` and `stdev`
- Filtered metric functions (filter function)
- Invalid queries:

Unsuppported Conditions/Filters	Queries
Like	**SELECT** transactionName, avg(responseTime) as AVRT **FROM** transactions **HAVING** AVRT LIKE '10'
Exists	**SELECT** transactionName, avg(responseTime) as AVRT **FROM** transactions **HAVING** AVRT IS NOT NULL
Not Aliased	**SELECT** avg(responseTime) as avtr **FROM** type **HAVING** avg(responseTime) > 10 **SELECT** txName, count(*), node, count(*) **FROM** transactions **HAVING** count(*) > 10
Nested	**SELECT** transactionId, AVG(segments.numCalls) as av_numCalls, MIN(segments.numCalls), MAX(segments.numCalls), SUM(segments.numCalls) **FROM** transactions **HAVING** av_numCalls > 2
Filter Function	**SELECT** appraisalrating, filter(min(salary), where salary > 60000) as val **FROM** transactions **HAVING** val > 60000
Multi-Valued Metric Functions	Percentile: **SELECT** appraisalrating, percentile(salary, 94) as per_sal **FROM** transactions **HAVING** per_sal > 2 ORDER By appraisalrating Standard Deviation: **SELECT** appraisalrating, stdev(salary) as stdev_sal **FROM** transactions **HAVING** stdev_sal > 2 ORDER By appraisalrating
Missing Parent Aggregation	**SELECT** avg(responseTime) as AVRT **FROM** transactions HAVING AVRT > 2

SINCE...UNTIL Clause

Use the **SINCE**... **UNTIL** clause on a time range in a query. If the **UNTIL** portion is omitted, the query matches all the events that occurred after the time specified in the **SINCE** portion. A query can contain both a **SINCE**... **UNTIL** clause and a **WHERE** clause. In such cases, **WHERE** argument comes before **SINCE... UNTIL** and is applied in that order. The time range specified in the clause overrides any other time limits applied to the query, including the time range selection in the UI.

Syntax

```
SINCE timevalue [UNTIL timevalue]
```

where `timevalue` is a combination of an integer and (optionally) a time unit. If `timevalue` is omitted the value is interpreted as a Unix epoch timestamp in milliseconds.

Unit Symbols	Description
[m | minute | minutes]	minute
[h | hour | hours]	hour
[d | day | days]	day
[w | week | weeks]	week (7 days)

Examples

Query	Result
SELECT * FROM transactions SINCE 1 h UNTIL 15 m	All transactions that occurred between 1 hour and 15 minutes ago
SELECT * FROM transactions SINCE 1 h	All transactions that occurred after one hour ago
SELECT * FROM transactions SINCE 1502147143000 UNTIL 1502148043000	All transactions that occurred between the Unix timestamps 1502147143000 and 1502148043000

LIMIT Clause

The **LIMIT** clause enforces a limit on the returned search results. **LIMIT** works differently for non-aggregation and aggregation queries.

The syntax is:

[**LIMIT** integer [, integer]...]

An integer is a sequence of digits. If **ORDER BY** is not specified in the query, the results sort in descending timestamp order. The Analytics Search UI caps the number of search results at 10000 records.

Non-Aggregation Queries

Specify **LIMIT** *N* for non-aggregation queries to return the first N documents.

SELECT * **FROM** transactions **LIMIT** 10

For non-aggregation queries, using the Analytics Query API, can return a maximum of 10,000 results. If a limit value higher than 10,000 is specified, the value is overridden with the maximum value.

Aggregation Queries

For aggregation queries a list of values is allowed in the **LIMIT** clause. Each value applies to one "group by" term or bucketing function in the query. A "group by" term is a field name in an aggregation query. In the following example, *field1* and *field3* are group by terms, and `series`(*field2*, '1m') is a bucketing function:

SELECT *field1*, `series`(*field2*, '1m'), *field3*, `count`(*) ...

Bucketing functions consume a value from the **LIMIT** clause list of values, but that value is ignored because it doesn't apply to such functions. Because the default limit is 10, if there are more bucketing functions than user-specified limits, the **LIMIT** clause will use 10 buckets. For example, if there two limit values specified, but three functions or terms that could consume the limit values, the last one defaults to a limit of 10.

The maximum number of buckets is 1000 for the first group by term or bucketing function. Subsequent terms are limited to 100 buckets. A larger value is overridden and replaced with the maximums. Aggregation queries can return a maximum of 10000 results.

REST API usage for limits is slightly different. See Analytics Events API.

Aggregation Examples

Query	Result
SELECT transactionName, `avg`(responseTime) **FROM** transactions	No limit specified, shows a maximum of top 10 results by count.
SELECT DISTINCT requestGUID **FROM** transactions	No limit specified, shows the top 10 results by count.
SELECT transactionName, `avg`(responseTime) **FROM** transactions **LIMIT** 50	Returns the top 50 results by count.
SELECT application, transactionName, `avg`(responseTime) **FROM** transactions **LIMIT** 500, 50	Returns the top 500 applications by count. Within each of those buckets, the top 50 transactionNames by count.
SELECT application, transactionName, `avg`(responseTime) **FROM** transactions **LIMIT** 2000, 500	Because of the enforced limits, this query returns the top 1000 applications by count. Within each of those buckets, the top 100 transactionNames by count.
SELECT application, transactionName, `avg`(responseTime) **FROM** transactions **LIMIT** 20	Returns the top 20 applications by count. within each of those buckets the top 10 (using the default) transactionNames by count.
SELECT application, `series`(eventTimestamp, '1m'), transactionName, `avg`(responseTime) **FROM** transactions **LIMIT** 20, 25, 30	Returns the top 20 applications by count. Within those a date-time series with all the data. Within each date-time series bucket, shows the top 30 transactionNames by count. The value 25 is applied to the series function, but because series is a bucketing function, it does not actually use the limit.

SELECT application, `series`(eventTimestamp, '1m'), transactionName, `avg`(responseTime) **FROM** transactions **LIMIT** 20, 25	Returns the top 20 applications by count. Within those, shows a date-time series with all the data. Within each date-time series bucket the top 10 (using the default) transactionNames by count. The value 25 is applied to the series function, but because series is a bucketing function, it does not actually use the limit.
SELECT `series`(eventTimestamp, '1m'), `avg`(responseTime) **FROM** transactions **LIMIT** 20	**LIMIT** doesn't apply to the functions, only to the group by terms, therefore in this query, the limit value has no effect and the query returns all the data in the series function.

ADQL Expressions

ADQL uses two primary categories of expressions in ADQL queries:

- Field expressions used in the **SELECT** clause
- Condition expressions used in the **WHERE** clause

To build an expression, use field names, functions, numbers, and operators indicated in the described syntax.

Syntax of field_expression (SELECT clause)

- {field_name | function | numeric_literal}
- (expression): surround the expression with parentheses to denote the order of operation in the query
- Math expressions such as:
  - unary_operator expression
  - expression binary_operator expression
- A condition_expression such as [**NOT**] expression **COMPARISON_OPERATOR** value can also be an alias used in a field_expression in the **SELECT** clause.
  For example:

```
SELECT field_name AS myField... WHERE myField > 2
```

Syntax of condition_expression (WHERE clause)

- [**NOT**] expression **COMPARISON_OPERATOR** value
- (condition_expression)
- condition_expression **LOGICAL_OPERATOR** condition_expression

The field expression syntax in the **SELECT** clause can also contain functions. The syntax of a function_expression is: `function`(field_name, arguments, ...)

Syntax Element	Description
field_name or numeric_field_name	The name of a field in the specified event type. Field names require back quotes when the name contains spaces or special characters. For example, the field name, `full name`, is surrounded by back quotes because it contains a space. See Special Characters for the full list. Use the Events Service internal names in your ADQL queries. The list of default fields and their internal names are described under ADQL Data.
function	See Analytics Functions.
numeric_literal	A sequence of digits, optionally including a decimal point ". " or sign {+ | -}. Example: 2, -4, 3.14
unary_operator	The - **unary** operator negates the value of the operand.
binary_operator	Arithmetic operators: {*, /, %, +, -}. See Math Expressions.
COMPARISON_OPERATOR	A set of operators that compare values. See Comparison Operators for the full list.
LOGICAL_OPERATOR	See Logical Operators.
value	The syntax for a value = {numeric_literal | string | boolean}. Values are text, numbers, dates or boolean values used to compare with the value in the specified field_name. The data type of the value must match the type of the specified field. Text strings require quotes. Numbers, dates, boolean values (true or false) and null do not need quotes. Quotes can be either single or double quotes.

Instead of specifying that the expression applies to a particular array element, the expression can apply to any array element.

Fields with Spaces or Special Characters

If the Events Service internal name contains a special character, then the name must be surrounded by back quotes when it is used in a field expression in the **WHERE** clause. For a field name such as `segments.httpData.headers.User-Agent`, the ADQL query must use `segments.httpData.headers.`User-Agent``.

For example:

```
SELECT segments.httpData.headers.`User-Agent`, count(segments.httpData.headers.`User-Agent`) FROM transactions
WHERE segments.httpData.url = 'http://localhost:29990/packages'
```

The most likely situation in which spaces and other special characters may be used in naming are when field names are added with data collectors.

Special Characters

Special characters in ADQL are:

- equals : '='
- not equals : '<>' or '!='
- less than : '<'
- less than or equals : '<='
- greater than : '>'
- greater than or equals : '>='
- plus: '+'
- minus: '-'
- asterisk: '*'
- slash: '/'
- percent: '%'
- concatenation: '||'
- question mark: '?'
- open parens: '('
- open square bracket/brace: '['
- close parens: ')'
- close square bracket/brace: ']'

Analytics Functions

Functions serve as arguments in the **SELECT** clause. Function types in ADQL include:

- Metric functions
- Bucketing functions
- String functions
- Funnel functions
- Data type conversion functions
- Functions that modify other functions

For many functions, only numeric fields are valid. "All" data types means *integer, floating point, datetime,* and *string.*

Functions ignore null values, such as those resulting from integer division by zero or empty data fields.

To add a **WHERE** clause to a query, use the query box in the data tab of the Controller Analytics search UI. When you perform the search, the visualization tab under the table widget shows the results of any metric functions.

Each section below describes one type of function.

Metric Functions

Metric functions:

- Perform a calculation on a set of values for a field.
- Return a result such as a count or sum, or a set of results such as percentile.
- Can only be in the **SELECT** clause.
- Should always follow a field name if one exists—otherwise, not all visualizations may work.

The distinctcount() function is accurate for up to 3000 unique return values. For over 3000 unique return values, distinctcount() is approximate due to performance optimizations.

Function	Returns	Valid Field Type (s)	ADQL Query Examples
`avg(numeric_field_name)`	Average value for a numeric field.	*integer, floating point, datetime*	**SELECT** `avg`(btdata.estimatedtime) **FROM** browser_records **WHERE** appkey = '*your_appkey*'
`count (field_name)`	A count of events.	*all* The form count(*) is also valid.	**SELECT** `count`(orderID) **AS** '*Sales*' **FROM** logs **WHERE** method='*GET*' The following query returns the top 10 IP addresses sending HTTP requests to an application. **SELECT** ip, `count`(*) **FROM** web_session_records **WHERE** appkey = "*yourApp*" LIMIT 10
`distinctcount (field_name)`	A count of the number of unique values recorded for a field.	*all*	**SELECT** `distinctcount`(field_name) **FROM** logs
`max(numeric_field_name)`	The maximum recorded value of a numeric field.	*integer, floating point, datetime* Specify one field only.	**SELECT** `max`(numeric_field_name) **FROM** transactions **SELECT** nodeName, `max`(eventtimestamp) **FROM** logs
`min(numeric_field_name)`	The minimum recorded value of a numeric field.	*integer, floating point, datetime* Specify one field only.	**SELECT** `min`(numeric_field_name) **FROM** transactions
`stdev(numeric_field_name)`	The population standard deviation of a numeric field.	i*nteger, floating point*	**SELECT** `stdev`(numeric_field_name)**FROM** transactions

`sum(numeric_field_name)`	The sum of the values of a numeric field.	*integer, floating point, datetime* Specify one field only.	**SELECT** `sum`(numeric_field_name) **FROM** transactions
`percentile(numeric_field_name, percent)`	The specified percentile values between 0 and 100.	*integer, floating point, datetime* Specify one or more percentiles.	**SELECT** `percentile`(responseTime, 50, 75, 90, 95) **FROM** transactions
`stats(numeric_field_name)`	Statistics for a field: `count`, `min`, `max`, `average`, and `sum`.	*integer, floating point, datetime*	**SELECT** `stats`(numeric_fieldName) **FROM** transactions **WHERE** application='*yourApp*'
`totalResultCount()`	The total count of events.	Does not accept a field.	This query first selects all error transactions for the "ECommerce" application. It then calculates, for each unique transaction name, the ratio of instances of that transaction name to the total number of error transactions. (The assumption is that any given transaction name can appear in multiple error transactions.) **SELECT** `transactionName`, `count ( * ) / totalResultCount()` **FROM** transactions **WHERE** application = 'ECommerce' **AND** userExperience = 'ERROR'

Bucketing Functions

Bucketing functions operate on a field, and group data into buckets. Often, the data in the buckets are further aggregated by another function. For example, this query creates buckets that group an app's transactions by the hour, and then count how many transactions are in each bucket (hour):

SELECT `series`(eventTimestamp, '1h'), `count`(*) **FROM** transactions **WHERE** application='*yourAppName*'

To include a larger data set in your results, use the time range selector in the upper right corner of the Controller Analytics search UI.

Dates are different from datetime values. Units from weeks down to milliseconds are supported:

- `'1w'` for one week
- `'10d'` for ten days
- `'2h'` for two hours
- `'5m'` for five minutes
- `'36s'` for thirty-six seconds
- `'800ms'` for eight hundred milliseconds

When you use dates to define buckets, think about how many buckets your definition produces. The number of buckets for series() cannot exceed 2000, and much smaller numbers of buckets are the norm.

Function	Returns	Valid Field Type(s)	ADQL Query Examples
`range(field_name, n|(n1, n2), n3|(n4, n5), ...)`	One or more buckets.	*integer, floating point, dates* Use a single value `x` or pairs of values `(x,y)`. See below for explanation of how to define buckets.	This range query: **SELECT** `range`(segments.transactionTime, 0, 10, 20) **FROM** transactions Results in the following buckets: [-Inf, 0], [0, 10], [10, 20], [20, +Inf]

`series(field_name, interval_size)`
 	
`series(field_name, interval_size, lower_extended_bound, upper_extended_bound, [offset],[rangeStyleBucketKeys], [strictEndpoints])`
 	
If you specify bounds, then all buckets are returned, empty or not.
 	
If you do not specify bounds, then:
• If you are using the UI, the UI inserts default bounds.
• If you are using the API, then only non-empty buckets are returned.
	A series of buckets based on the provided interval.	*integer, floating point, dates*
		
		For numeric fields, specify the interval as an integer.
		
		See below for explanation of optional keyword arguments `offset`, `rangeStyleBucketKeys`, and `strictEndpoints`.
			SELECT `series`(metrics.`Application Server Time (ms)`,10), `count`(*) **FROM** browser_records **WHERE** appkey='*yourApp Key*'
			
			SELECT `series`(eventTimestamp, '1h'), `count`(*) **FROM** transactions **WHERE** application='*yourAppName*'

Defining Buckets for range()

Every bucket is defined as bounded by two single values, where the bucket includes the first value and excludes the second.

Pairs of values `(x, y)` define an explicit bucket from `x`, inclusive, to `y`, exclusive.

A single value `x` implicitly defines a bucket according to rules that depend on where the value is placed among the arguments to `range()`.

- Rules for using a single value to define a bucket:
    a. Immediately after the field (that is, as the second argument): The bucket starts at negative infinity.
    b. After another single value: The bucket starts at the previous argument, inclusive, and extends to `x`, exclusive.
    c. After a pair of values, and before another single value: The bucket starts at `x`, inclusive, and extends to the following argument, exclusive.
    d. At the end of the parameter list: The bucket ends at positive infinity. This is the one exception to the assumption that `x` is the second of two values.
    e. Between two pairs of values: No bucket can be defined, and the value is ignored.

Range Query Example

Here is a range query that exercises most of the rules above:

SELECT `range`(segments.transactionTime, 0, (10, 20), 30, (40, 50), 60, 70) **FROM** transactions

The query produces the following buckets:

- [-Inf, 0] because rule *a* applies to 0
- [10, 20] because (10, 20) defines that bucket explicitly
- no bucket for 30 because rule *e* applies so 30 is ignored
- [40, 50] because (40, 50) defines that bucket explicitly
- [60, 70] because rule *c* applies to 60
- [70, +Inf] because rule *d* applies to 70

Optional keyword arguments for `series()`: `offset`, `rangeStyleBucketKeys`, and `strictEndpoints`

Series supports the following optional keyword arguments.

`offset`

Changes the start value of each series from 0 by the specified positive (+) or negative offset (-) duration, such as 1h for an hour, or 1d for a day. The values `alignStart` and `alignEnd` align buckets to the start and end of the extended bounds arguments respectively. The function must have extended bounds specified to use these special values. Defaults to 0. Accepts both integer and string values.

Examples:

SELECT `series`(responseTime, 1, 1, 10, `offset`="*alignStart*"), `count`(*) **FROM** transactions

SELECT `series`(responseTime, 100, 1, 1000, offset=1), `count`(*) **FROM** transactions

`rangeStyleBucketKeys`

Converts the start and end values in a numeric series to "bucketStart - bucketEnd" format as a string type. For example, "0-10", "10-20", and so on. This output this argument returns is similar to what returned by the range function. The series function must be acting on a numeric type to use this keyword argument. Defaults to false. Accepts a boolean only.

Example:

SELECT `series`(responseTime, 100, 1, 1000, `rangeStyleBucketKeys`=*true*), `count`(*) **FROM** transactions

`strictEndpoints`

Adds a filter on the query starting at the minimum extended bound and extending to the maximum extended bound. This ensures that no buckets outside of the extended bounds are present in the results. Defaults to false. Accepts a boolean only. Extended bound forces the histogram aggregation to start building intervals on a specific minimum value and expands the interval up to a given maximum value.

Example:

SELECT `series`(responseTime, 2, 1, 5, strictEndpoints=true, rangeStyleBucketKeys=true), `count`(*) **FROM** transactions

Selecting the Top *n* Results for Time Series Queries

Deployment Support

Top *n* results is available for SaaS deployments only.
SELECT `series`(eventTimestamp, '1m'), transactionName, `count`(*) **FROM** transaction **LIMIT** 1, 12The series function can compute the global top *n* over an entire period. To set a top *n* limit, add a **LIMIT** clause to the end of your query:

This example limits the query to the top 12 results of transactionName. The value 1 is applied to the `series` function, but because series is a bucketing function, it does not actually use the limit. You must include a value for the `series` function.

The global top *n* limit does not apply when queries:

- Contain a selection before the series function
  SELECT application, `series`(eventTimestamp, '1m'), transactionName, `...`
- Have additional bucketing functions
  SELECT `series`(eventTimestamp, '1m'), `range`(responseTime, 0, 60, 120), transactionName, `...`
- Include a **HAVING** clause
  SELECT `series`(eventTimestamp, '1m'), transactionName, `avg`(responseTime), ... **HAVING** a >15

String Functions

ADQL string manipulation functions:

- Resemble their SQL counterparts.
- Are case-sensitive.
- Return null if any argument is null.
- Use 1-based, not 0-based, indexing.
  - For example, the `indexOf()` function gives the index of the first character of a string as 1, not 0.
  - As a result, 0 is not a valid value for any integer argument

String functions can transform any event data that resides in the Analytics Events Service. One important use case is doing ad hoc data transformation when the structure, serialization, or format in which data has been collected needs to be changed.

For AppDynamics < 4.5.2, fixing incorrectly-collected data requires adjusting collection settings.

You can use string functions to:

- Hone in on a `substring()` by working with string `length()` and/or the `indexOf()` characters in a string
  - For example, you can extract the domain name from a URL
- Combine multiple strings using the `concat()` operator
  - For example, you can combine values from multiple fields into a single value

- `trim()` whitespace from the beginnings and ends of fields
- Pair the output of the `substring()` function, item by item, with the output of another function

  - This implicit aggregation is the ADQL equivalent of GROUP BY in SQL
  - See the Implicit Aggregation Example:

Function	Returns	Valid Field Type(s)	ADQL Query Examples

`concat(inputString1, string inputString2, ... inputStringN)`	A single string which combines the input strings.	*string*
		
		Specify two or more strings.
			Function:
			`concat('foo', 'bar')`
			Result:
			`'foobar'`

`indexOf(inputString, substring, [occurrence])`	1-based start index of `substring` in `string`.
	
	If `substring` is not found, returns 0.
	
	If `occurrence` is supplied as *n*, returns index of nth occurrence of `substring`.
	
	If `occurrence` is supplied as -*n*, returns index of nth occurrence of `substring` counting backwards from the end of `inputString`.
		string for `inputString`, `substring`
		
		positive or negative *int* for `occurrence`
			Function:
			`indexOf('www.wikipedia.org', '.', 1)`
			Result:
			4
			
			Function:
			`indexOf('www.wikipedia.org', '.', -1)`
			Result:
			14
			
			Function (example of chaining and combining functions):
			SELECT `substring('www.wikipedia.org', indexOf('www.wikipedia.org', '.', 1)+1, indexOf('www.wikipedia.org', '.', -1) - indexOf('www.wikipedia.org', '.', 1) -1)` **FROM** transactions
			Result:
			`'wikipedia'`

`length(inputString)`	The number of characters including whitespace of the input string.	*string*	Function: `length(' foo bar ')` Result: 9

`substring(inputString, startIndex, numChars)`	The substring of `inputString` starting at (1-based) `startIndex`.
	
	If `startIndex` is negative, starts that number of characters from the end of the string.
	
	If `numChars` is supplied, returns substring of that length; if `numChars` runs past end of string, returns only up to end of string.
	
	If `startIndex` or `numChars` is invalid, returns an empty string.
		string for `inputString`
		
		positive or negative *int* for `startIndex`
		
		positive *int* for `numChars`
			This example extract the domain name from a URL.
			Function:
			`substring('https://example.com/home.htm', 9, indexOf('https://example.com/home.htm','/',3) - 9)`
			Result:
			`'example.com'`

`trim(inputString)`	A copy of the input string with leading and trailing whitespace removed.	*string*	Function:
			`trim(' foo bar ')`
			Result:
			`'foo bar'`

Implicit Aggregation Example

Consider this data:

Application ID	responseTime (ms)
SJC-001	500
SJC-002	600

SJC-003	700
LAX-001	200
LAX-002	300
LAX-003	400

The following ADQL query pairs each item from the output of `substring()` with a corresponding item from the output of `avg()`:

SELECT `substring(application, 1, 3)`, `avg(responseTime)` FROM transactions

The results are:

SJC, 600

LAX, 300

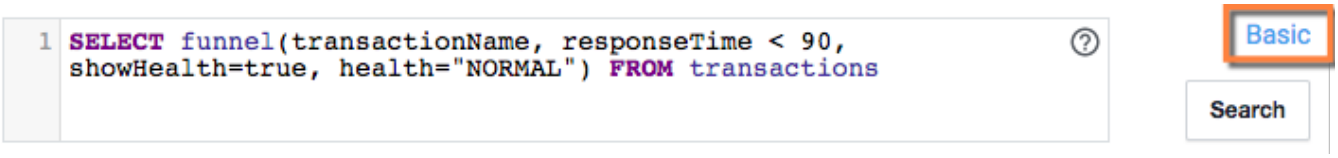
Notice how ADQL uses `substring()` to govern the groupings to which the second function, `avg()`, is applied.

Funnel Functions

You can create funnel widgets from an ADQL query using the funnel function. This query example creates a funnel widget of normal transactions:

```
SELECT funnel(transactionName, responseTime < 90, showHealth=true, health="NORMAL") FROM transactions
```

To see the funnel widget, enter your query and select **Search**. After you see the search results, select **Basic** at the top right of the query:



See AppDynamics Community.

Data Type Conversion Functions

> Data type conversion functions are only available for SaaS deployments.

Data type conversion functions can be used in the **SELECT** or **WHERE** clauses. Below are the available data type conversion functions. For `formatString` requirements, see Joda's `DateTimeFormat` documentation in the Classes section.

Function	Returns	ADQL Query Example
`toDate(string value, [string formatString])`	A string value formatted as a datetime.	**SELECT** `toDate(field_name, "MM/dd/yyyy")` **FROM** `transactions` *Returns* 2019-01-15T00:00:00.000Z for a `field_name` with string value "01/15/2019"
`toDate(int value)`	An integer value formatted as a datetime.	**SELECT** `toDate(12341234)` **FROM** `transactions` *Returns* Dec 31 1969 19:25:41 GMT-0800
`toString(datetime value, [string formatString])`	A datetime value formatted as a string.	**SELECT** `toString(field_name, "MMM dd, yyyy")` **FROM** `transactions` *Returns* "Jan 15, 2019" for a `field_name` with datetime value 2019-01-15T00:00:00.000Z

`toString(float/int/bool value)`	A value formatted as a string.	**SELECT** `toString(123.123)` **FROM**`transactions` *Returns* "123.123"
`toInt(datetime/float/string/bool value)`	An integer value, formatted as follows: • `datetime`- converts to milliseconds, • `float`- truncates • `string`- parses to floats and truncates to an integer • `bool`- converts to 0/1	**SELECT** `toInt(123.4)` **FROM**`transactions` *Returns* 123
`toFloat(int/string value)`	An integer or string value formatted as a float.	**SELECT** `toFloat(123123)` **FROM**`transactions` *Returns* 123123.0
`round(float value, int decimalPlaces)`	A value rounded off to `decimalPlaces`. `decimalPlaces` must be a non-negative integer.	**SELECT** `round(123.123123, 3)` **FROM**transactions *Returns* 123.123
`ifNull(object value, object replacementValue)`	A value if the value is non-null. If the value is null, returns `replacementValue`.	**SELECT** `ifNull(field_name, 10)`**FROM** `transactions` *Returns* 10 if the value of `field_name` is null *Returns* Value of `field_name` if the value is not null

⚠ For data type conversion functions (except `ifNull`), if the value of `field_name` returns null, the converted data type will also return null.

Use mathematical and `now( )` functions on datetimes

You can perform addition and subtraction mathematical functions on datetime and long values, for example:

```
SELECT toDate(field_name, "MMM, dd, yyyy") - toDate(field_name,"MMM, dd, yyyy") FROM transactions
```

The function will return a datetime value. Multiplication, division, and modulus mathematical functions are not supported. Addition and subtraction functions cannot be combined with floats.

You can use the `now( )` function to add or subtract the server's current time from a value, for example:

```
SELECT toInt(now() - field_name) FROM transactions
```

The above function determines how long ago `field_name` was created and converts the value to an integer.

Functions That Modify Other Functions

Function	Returns	Valid Field Type(s)	ADQL Query Examples

filter(metric_function (field_expression), ...) [**WHERE**] condition_expression)	Aggregations for specific subsets of data, computed by applying a filter to the input of a single metric function.	The metric function must be one of the following that returns a single value. {`count	avg	max	min	sum	distinctcount`}  The second argument has the same syntax as a **WHERE** clause and optionally begins with the **WHERE** keyword, for example:  **SELECT** filter(`avg`(responseTime), **WHERE** responseTime > 1) **FROM** transactions **WHERE** application = "Travel"	filter(`sum`(numeric_field_name), numeric_field_name > 100) **SELECT** 100.0 * filter(`count`(*), field_name = "value") / `count`(*) as "%" **FROM** transactions **WHERE** application = "*yourApp*" AND transactionName = "*yourValue*"

Combining Different Types of Functions

Here are some special cases to consider when combining functions of different types:

- The maximum accuracy of distinctcount() is reduced if it follows a bucketing function. Nested `distinctcount()` queries trade accuracy for memory savings, because the previous function establishes buckets, and distinctcount() creates sets of buckets within each of the previous function's buckets, and Elasticsearch automatically makes that tradeoff when that happens
    - For example, the first of the two example queries below returns less accurate results than the second:
        - `SELECT transactionName, distinctcount(userId) FROM transactions`
        - `SELECT distinctcount(userId) FROM transactions`
- The return value of `stdev(numeric_field_name)` cannot be used to sort or filter results.

Analyzed Fields

Selected fields in the analytics data are "analyzed". Analysis consists of tokenizing a block of text into individual terms suitable for use in an inverted index and normalizing these terms into a standard form to improve their searchability. The ADQL Comparison Operators behave differently for analyzed and non-analyzed fields. To construct queries on analyzed fields, you need to understand some concepts about how the tokens are built.

Uppercase letters in analyzed fields are all converted to lowercase to build tokens.

Delimiters and other factors (such as CamelCase terms) affect how strings are tokenized. For a string such as "myname@company.com", "@" is a delimiter, therefore myname and company.com are two separate tokens. Additionally, company.com is two separate tokens. Note that all non-alphanumeric characters are delimiters. A term that uses CamelCase, such as `VicePresident`, is tokenized into separate tokens based on recognition of the CamelCase nature of the term resulting in the tokens: `Vice` and `President` as well as `VicePresident`.

For an example string such as: `<VicePresident:SalesAndMarketing> - EMEAAustraliaUSA94107`, the tokens generated include the following:

- vicepresident
- vice
- president
- salesandmarketing
- sales
- and
- marketing
- emeaaustraliausa94107
- emeaaustraliausa
- emea
- australia
- usa
- 94107

Analytics Analyzed Fields

The analyzed fields in analytics events are:

- **Logs**: Message
- **Transactions**: Errors and Error Detail
- **Mobile**: stacktrace

Queries on Analyzed Fields

Full-text search is supported on analyzed fields, including the message field for logs, using the **LIKE** operator. See Comparison Operators.

On analyzed fields, the **REGEXP** operator matches exactly only the analyzed and processed tokens, so you cannot query across the complete message.

Consider this log message:

```
[2016-06-09 06:07:50,118] [INFO ] [org.springframework.jms.listener.DefaultMessageListenerContainer#0-1] [com.
appdynamics.provision.OrderMessageListener] [AD_REQUEST_GUID[2b8ce807-986c-45f9-a5b4-2fe5a6fd90f3]] Received a
message to process the order Order_3138 for the user myname@company.com
```

In this log message, myname and company.com are two separate tokens because "@" is a delimiter. To search a log message like this for results based on the email address requires searching across tokens.

A query such as the following using **REGEXP**, will **fail** because myname and company.com are two separate tokens.

```
SELECT FROM logs WHERE appName = 'yourAppName' AND sourceType = 'yourLogFile' AND message REGEXP
‘myname@company.*
```

The **LIKE** operator is not affected by delimiters, so an alternative query using **LIKE** operator is a better choice.

```
SELECT FROM logs WHERE appName = 'yourAppName' AND sourceType = 'yourLogFile' AND message LIKE ‘myname@company'
```

You can also use wildcards in the query because they work across tokens.

```
SELECT FROM logs WHERE appName = 'yourAppName' AND sourceType = 'yourLogFile' AND message LIKE ‘myname@company*'
```

Example Queries for Analyzed Fields

Searching the following analytics log events:

```
Jun 7 11:27:45 appd sshd[1032]: Illegal user test from 110.49.183.11
Jun 7 11:27:46 appd sshd[1032]: Failed password for illegal user test from 110.49.183.11 port 9218 ssh2
Jun 7 11:27:46 appd sshd[1032]: error: Could not get shadow information for NOUSER
```

Note the results from the following queries:

Query	Results
SELECT * **FROM** logs **WHERE** sourceType='*yourLogFile'* **AND** message **REGEXP** 'illegal.+user'	Does not match any log events in the sample because the query string spans across multiple tokens. Use LIKE for an instance like this.
SELECT * **FROM** logs **WHERE** sourceType='*yourLogFile'* **AND** message **REGEXP** 'illegal.*'	Matches the first two log events
SELECT * **FROM** logs **WHERE** sourceType='*yourLogFile'* **AND** message **REGEXP** 'Failed*'	Does not match any log events in the sample because the token has only lowercase 'failed'

To search for string : javaIOException

Query
SELECT * **FROM** transactions **WHERE** application = '*yourApp*' **AND** segments.errorList.errorCode **REGEXP** 'java[a-z][a-z][a-z][a-z][a-z][a-z][a-z][a-z][a-z][a-z][a-z]'

Math Expressions

Math expressions are field expressions that use arithmetic operators. These expressions can be used in ADQL queries in both the **SELECT** and **WHERE** clauses.

field_expression **BINARY_OPERATORS** field_expression

Arithmetic Operators

Math operators are valid on integer and float data fields, constants (for example, 3.14), and on functions that return a single integer or float value, with the exception of % (modulo). The operators `*, / and %` have higher precedence than + and `-`. Parentheses can be used to control the order of operations, a unary minus (`-`) is also supported.

Operator	Name	Description
*	asterisk	Multiplication
/	slash	Division of two integer types performed as integer division. Integer division by zero returns null.
%	modulo	Calculates the reminder after the division of two integers. Valid only for integers.
+	plus	Addition. Requires a space before a numeric literal.
-	minus	Subtraction. Requires a space before a numeric literal. Can also be used as unary_operator.

Math Expression Usage

Math expressions can be used in several ways.

Query	Result
SELECT regionId % 4, `count`(*) **FROM** transactions **WHERE** application = "*yourApp*"	Math expression is used as a **GROUP BY** term in an implicit group by query
SELECT field_name1, (field_name2 + field_name3) * 3.14 **FROM** transactions **WHERE** application = "*yourApp*"	Math expression is used as a composite field.
SELECT `sum`(field_name1) / `count`(field_name2) **FROM** transactions **WHERE** application = "*yourApp*" **SELECT** filter(`sum`(field_name1), **WHERE** field2 **IS NOT NULL**) / `count`(field_name3) **FROM** ...	Math expression used on an aggregation function.
SELECT `avg`(field1 + field2) **FROM** event_type **WHERE** ... **SELECT** `series`(field1 + field2, 1) **FROM** event_type **WHERE** ...	Math expression used within an aggregation function.
SELECT * **FROM** transactions **WHERE** (responseTime * cartValue) > 100 **SELECT** * **FROM** transactions **WHERE** (regionId % 3) **IN** (1, 2) **SELECT** * **FROM** transactions **WHERE** (responseTime * cartValue) **BETWEEN** (100, 200)	Math expression used in a **WHERE** clause with comparison operators.
SELECT segments.userData.itemTitle, (segments.userData.itemPrice / 100) **AS** *Price* **FROM** transactions **WHERE** application = 'ECommerce 2.0' **AND** transactionName = '/product_details.xhtml' **AND** Price > 20	Convert a value in cents to dollars. This query converts segments.userData.itemPrice from cents to dollars and shows results where the price is greater than $20.
SELECT transactionName, (responseTime / 1000) **AS** responseSeconds, userExperience, requestGUID **FROM** transactions **WHERE** application = "ECommerce 2.0" **AND** responseSeconds > 5	Convert a value such as average response time to seconds instead of milliseconds and view each transaction that is greater than X seconds.

Usage Notes

If any field or intermediate expression result for an expression is null, the entire expression will be null.

Math expressions in the **WHERE** clause are evaluated per record, not in aggregate.

Fields and metric functions can not be combined in the same expression, for example, a query for the following type is *not* valid:

SELECT responseTime + sum(cartValue) **FROM** ...

Mathematical operations that operate on every event (as opposed to on aggregation functions) will reduce query performance. Avoid statements such as:

WHERE (responseTime - 200) > 800 and use a construct such as **WHERE** responseTime > 1000 instead.

Comparison Operators

Comparison operators, such as =, !=, <, >, **LIKE**, and **IN**, can be used in condition_expressions of the **WHERE** clause in the ADQL query statement.

This page describes the comparison operators that you can use in the condition_expression syntax.

Operators

Comparisons on strings are case-insensitive.

Operator	Description	Supported Usage
=	Equals: True if the value of the specified field_name equals the specified value in the expression. String comparisons using the equals operator are case-sensitive.	non-analyzed fields
!=	Not equals: True if the value in the specified field_name does not equal the specified value.	non-analyzed fields
>	Greater than: True if the value in the specified field_name is greater than the specified value.	non-analyzed fields of t ype int, float and date
>=	Greater than or equal: True if the value in the specified field_name is greater than or equal to the specified value.	non-analyzed fields of t ype int, float and date
<	Less than: True if the value in the specified field_name is less than the specified value.	non-analyzed fields of t ype int, float and date
<=	Less than or equals: True if the value in the specified field_name is less than or equal to the specified value.	non-analyzed fields of t ype int, float and date
IN	Finds records where the value of a field equals any one of the specified values in a list of string or numeric values. The values for **IN** must be in parentheses. String values must be surrounded by single quotes. Wild card usage within the list of values provided to the **IN** operator is supported. For example: `SELECT transactionName, percentile(segments.transactionTime, 90) FROM transactions WHERE application = 'prd-analytics' AND transactionName IN ('api_v2./v2/events/* /search','api_v1./v1/events/*/search')`	non-analyzed fields only. Invalid for analyzed fields.
LIKE	The **LIKE** operator performs a full-text search that analyzes the string value and field value. If the string value contains multiple terms, separated by spaces, it is processed as an **AND** condition of the terms. Use **LIKE** to search values in analyzed fields. The **LIKE** operator supports the use of wild cards. **LIKE** is not supported for queries on mixed case strings. For example, querying against a log message that contains "pEer" will not match the search string 'peer', because "pEer" is tokenized by the search analyzer into two separate tokens - "p" and "eer").	for full-text search of analyzed fields
BETWEEN	Finds values of a field based on an inclusive or exclusive value range. For an inclusive range search, use square brackets [ ]. For an exclusive value range, use parentheses ( ).	Supported for data types int, float, and datetime (*New in 4.4.1*).
IS NULL	Finds records where a field does not contain a value.	non-analyzed fields and analyzed fields
IS NOT NULL	Finds records where a field contains a value.	analyzed and non-analyzed fields
NOT BETWEEN	Finds records where a field does not contain a value in an inclusive or exclusive value range. The reverse of **BE TWEEN**.	non-analyzed fields
NOT LIKE	The reverse of **LIKE**.	analyzed fields
NOT IN	Finds records where the value of a field does not match any value in a list of string or numeric values. The values for **NOT IN** must be in parentheses, and string values must be surrounded by single quotes. There is also a logical operator **NOT**, which is unrelated to this comparison operator.	non-analyzed fields

REGEXP	The **REGEXP** operator applies a regular expression pattern match of a string to the pattern passed as an argument. See REGEXP Operator for details of usage. This is an advanced feature and requires knowledge of regular expression patterns.	non-analyzed fields For analyzed fields: • Use lowercase input strings. • Do not use to search across tokens
* ?	The wild card operators. Use of wild cards is supported on queries of string fields. The asterisk, '*', character matches zero or more characters and the question mark, '?', character matches a single character. The wild card operator works similarly when used with the equals "=" or **LIKE** operator. The wild card operator is case sensitive for non-analyzed fields.	analyzed fields and non-analyzed fields

Examples

Query	Result
SELECT * **FROM** mobile_snapshots **WHERE** appkey='*your_appkey*' **AND** numeric_field_name **IN** (1419, 613, 748)	Returns all fields for mobile snapshots for the specified appkey where the value of the specified numeric field matches one of the values in parentheses.
SELECT count(*) **FROM** browser_records **WHERE** appkey='*your_appkey*' **AND** field_name **IN** ('string1','string2', 'string3')	Returns a count of browser records for the specified appkey where the value of the specified field matches one of the specified strings.
SELECT ip **FROM** browser_records **WHERE** ip **LIKE** "10.134.*"	By filtering on IP address range using wild card operator, finds records in the specified IP address range.
SELECT * **FROM** logs **WHERE** message **LIKE** 'DEBUG Adding operation'	Matches the sample log line "DEBUG com.appdynamics.operations. PublishTestService - Adding to bulk publish operation."
SELECT * **FROM** logs **WHERE** message **LIKE** 'error'	Finds all the log events containing the word 'Error' in any field
SELECT * **FROM** transactions **WHERE** application='*yourAppName*' **AND** cartTotal **BETWEEN** [10, 50]	Inclusive range: Returns all transactions with a cart total between 10 and 50, including 10 and 50.
SELECT * **FROM** transactions **WHERE** application='yourAppName' **AND** cartTotal **BETWEEN** (10, 50)	Exclusive range: Returns all transactions with cart total between 10 and 50, excluding 10 and 50.

Logical Operators

Logical operators can be used in the **WHERE** clause in an ADQL query to make compound condition expressions. These operators are **AND**, **OR**, and **NOT**.

Condition Expression Syntax

(condition_expression **LOGICAL_OPERATOR** condition_expression)

Examples

Expression	Result
(field_expression_1 **AND** field_expression_2)	Evaluates as true if both field_expressions are true.
(field_expression_1 **OR** field_expression_2)	Evaluates as true if either field_expression is true.
(field_expression_1 **AND NOT** field_expression_2)	Evaluates are true if field_expression_1 is true and field_expression_2 is false.

REGEXP Operator

This is an advanced feature and requires knowledge of regular expression patterns.

The regular expression operator, **REGEXP**, can be used in the **WHERE** clause to handle complex matching queries. This operator applies a regular expression pattern match of a string to the pattern passed as an argument. ADQL uses the Lucene regular expression engine to analyze the **REGEXP** exp ression. The search results for queries that use the **REGEXP** operator are dependent not only on regular expression syntax and rules but also on whether you are searching analyzed or non-analyzed fields. To use **REGEXP** on analyzed fields, see Analyzed Fields.

The **REGEXP** operator can only match with the lowercased tokens in the analyzed fields. To search for an uppercase string in an analyzed field using a wild card or **REGEXP**, you need to input a lowercase string. For example info, not INFO.

Allowed Characters

Any Unicode character may be used in the regular expression pattern, but certain characters are reserved and must be escaped. Any reserved character can be escaped with a backslash `"\*"` including a literal backslash character:`"\\"`.

The standard reserved characters are:

```
. ? + * | { } [ ] ( ) " \
```

For query performance reasons, before using any reserved character in a **REGEXP** pattern, you need to specify the first three characters of the string explicitly, such as specifying 123 before the brackets [0-9] in the following example:

SELECT * **FROM** logs **WHERE** sourceType='*yourLogFile*' **AND** id **REGEXP** '123 [0-9]'

A query such as the following is invalid.

SELECT * **FROM** logs **WHERE** sourceType='*yourLogFile*' **AND** id **REGEXP** '[0-9]'

Example Operations, Strings, and Associated Patterns

Supported Operation	Description	String	Pattern
Match any character	A period "." can be used to represent any character.	abcde	abc..
One or more	A plus sign "+" can be used to repeat the preceding shortest pattern one or more times.	aaabb	aaab+
Zero or more	An asterisk "*" can be used to match the preceding shortest pattern zero or more times.	aaaabbbcc	aaaab*c*
Zero or one	A question mark "?" makes the preceding shortest pattern optional. It matches zero or one times.	aaabbbc	aaa?b+c?
Min-to-max	Curly brackets "{}" can be used to specify a minimum and (optionally) a maximum number of times the preceding shortest pattern can repeat. Allowed forms: {5} # repeat exactly 5 times {2,5} # repeat at least twice and at most 5 times	aaaabbbcc	aaaab{3}c {2} aaaab{2,4} c{2,4}
Grouping	Parentheses "( )" can be used to form sub-patterns.	abababab	abab(ab)*
Alternation	The pipe symbol " | " acts as an OR operator. The match will succeed if the pattern on either the left-hand side OR the right-hand side matches.	aaabbb	aaa (ccc|bbb)

ADQL Data

This page describes data that ADQL captures in Analytics.

Analytics Type	Description
Transaction Analytics	Transaction Analytics is defined as event data collected by the Java and .NET Agents for every instance of a business transaction. Every instance of a business transaction that passes through a tier is defined as an event. You can view the raw data for any transaction instance that occurs within the data retention time. Transaction Analytics enables you to analyze the business impact of every event and observe how customers use your application.
Browser, Mobile, and IoT Analytics	Browser, Mobile, and IoT Analytics provide event data collected by the End User Monitoring (EUM) JavaScript Agent, Mobile, and IoT Monitoring SDKs.
Log Analytics	Log Analytics provides event data collected from log files.
Synthetic Sessions Analytics	Synthetic Sessions Analytics provides events data on sessions collected by Browser Synthetic Monitoring.

See AppDynamics Analytics Events API.

Analytics Transaction Data

Each instance of a business transaction passing through a single tier is an event, so the data associated with a single execution of a distributed business transaction (for example, Checkout) is stored as a series of events in Analytics. The data consists of the default transaction data and additional data from data collectors configured for that transaction. This data is organized and stored per business transaction.

Default Transaction Data

Event Type: transactions

Key (event type identifier): application

The following table lists the data collected by default for each instance of the business transactions that are enabled for Analytics. The table also lists the Events Service internal name. When accessing the data using ADQL queries, you need to use the Events Service internal name in the query.

UI Field Name	Description	Events Service Internal Name
Application	Application name	`application`
Business Transaction	Business transaction name	`transactionName`
Error	Error details (deprecated) This field is replaced by the more granular fields: Error Code, Error Detail, and Error Type.	`deprecated`
Error Code	The value of this field depends on the Error Type. For example, if Error Type is an HTTP_Error_code, then the value is a string representing the corresponding HTTP error code such as 404, 503 and so on.	`segments.errorList.errorCode`
Error Detail	This is what the "Error" field represented in 4.1. Primarily the error message	`segments.errorList.errorDetail`
Error Type	Classification of errors into buckets done internally by the Agent. For example, the error type could be HTTP_ERROR_CODE, LOGGER_MESSAGE, THROWABLE and so on.	`segments.errorList.errorType`
Node	Node name	`segments.node`
Tier	Tier name	`segments.tier`
request GUID	GUID for this specific user request as assigned by AppDynamics	`requestGUID`
Response Time (ms)	Business transaction response time for this request in milliseconds	`responseTime`
Timestamp	Time the event occurred in the application	`eventTimestamp`
User Experience	Indicates if the transaction was marked as Normal, Slow, Very Slow, Stall, or Error	`userExperience`
Exit calls	Details of database and remote service calls. View Details flow map. Note that for partial snapshots, exit call information captured in call graphs can differ between the Analytics and APM UI. Each UI presents an appropriate level of contextual visibility while preserving low overhead. An APM snapshot, for example, may capture only the exit calls that are most pertinent to transaction performance.	Not Applicable

Custom HTTP Request Data

When HTTP data collectors are configured, the following information can be collected:

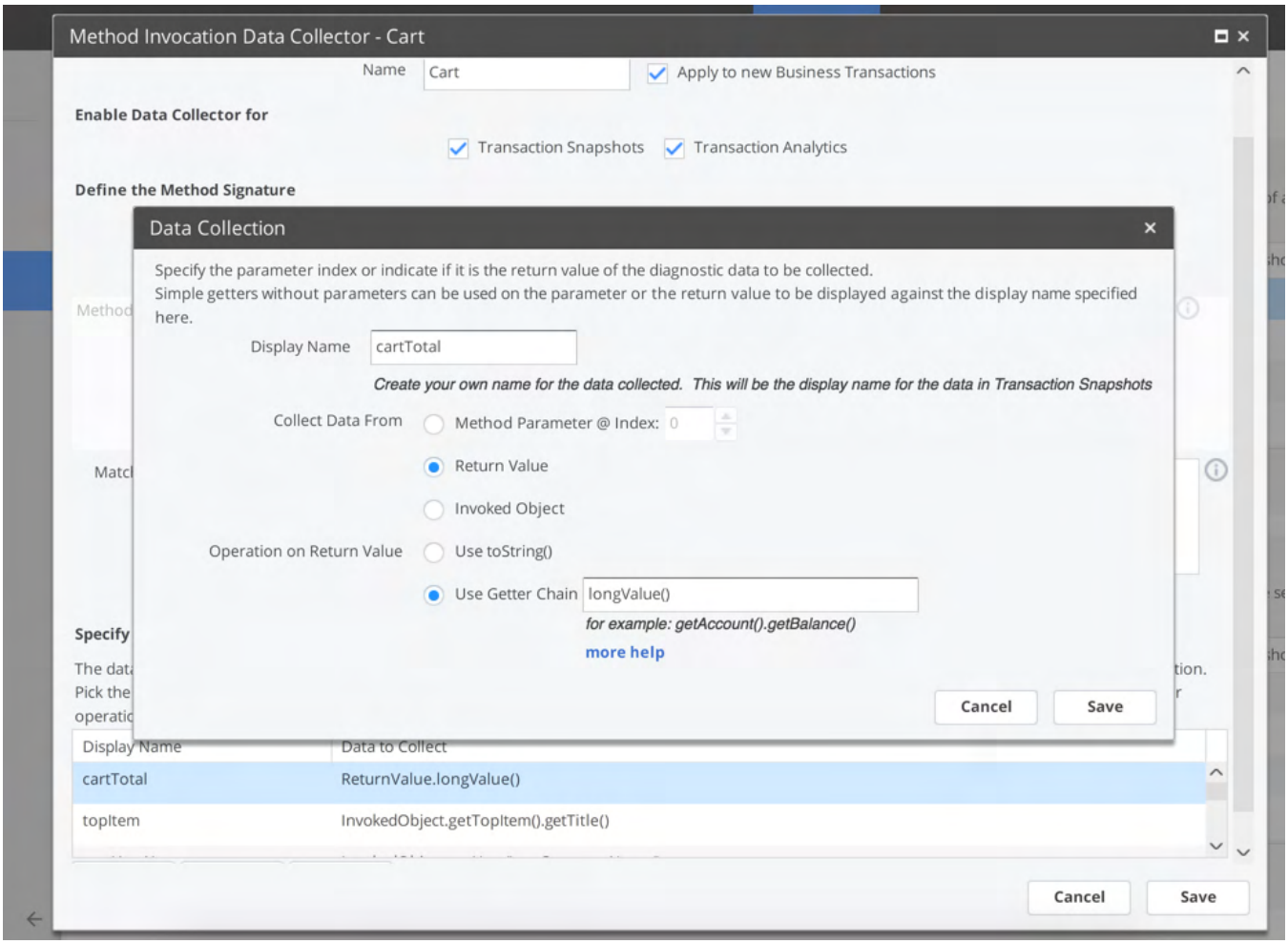
Field	Events Service Database Name
cookies	`segments.httpData.cookies`
headers	`segments.httpData.headers`
parameters	`segments.httpData.parameters`
principal	`segments.httpData.principal`
session ID	`segments.httpData.sessionId`
session Objects	`segments.httpData.sessionObjects`
URL	`segments.httpData.url`
URI path segments: segment0-*n*	`segments.httpData.uriPathSegments`

Analytics uses the configured display name as the field name for the HTTP parameters. In Controller versions prior to 4.3, Analytics displayed the data using the actual parameter name.

Custom Method Data

Custom data collected as specified in the method invocation data collector configuration. See Data Collectors.

To specify a custom data field when using the Analytics query APIs, prefix the field name with `segments.userData`. For example, if your field is cartTotal, such as shown in the following screenshot, the database name is `segments.userData.cartTotal`.



Deprecated Fields

AppDynamics deprecated the `Request Experience` and `Transaction Time` for Analytics Agents >= 4.2.x. Both fields are replaced with new fields:

Deprecated Field	Description	New Field
Request Experience	Indicates if the transaction was marked as Normal, Slow, Very Slow, Stall, or Error	User Experience
Transaction Time	Business transaction response time for this request in milliseconds	Response Time

Analytics Log Data

Log Analytics collects from log files depends on the source of the log file and the pattern that you specify to structure the data in the log with. Every log entry is an event in the Log Analytics event stream.

Log Analytics Data

Log Analytics only supports the UTF-8 encoding format.

Event Type: logs

Key (event type identifier): sourceType

These fields are captured by default; you can configure and capture optional data, but these fields are always present:

UI Field Name	Description	Events Service Internal Name
pickupTimestamp	The timestamp when the Java Agent picked up the event and sent it to the Analytics Agent.	`pickupTimestamp`
Message	The message body of the log event.	`message`
host	IP address or host name where the event was generated.	`host`
source	Location of the logs, usually a path or directory such as `/tomcat/logs`.	`source`
sourceType	The kind of log file, such as `apache-httpserver-access-log`.	`sourceType`
Timestamp	Timestamp of the log event.	`eventTimestamp`
Extracted Fields	Fields that were extracted using the Controller UI in previous versions appear in the Extracted Fields list. See Collect Log Analytics Data.	Varies

You can optionally configure these fields:

Optional Fields	Description
nodeName	Name of the node where the log event occurred.
tierName	Name of the tier where the log event occurred.
appName	Application name where the log event occurred.

Analytics Custom Events Data

Data collected for your Analytics custom events depends on the schema you create and manage.

The only default field is `eventTimestamp`. If custom events are used to configure Business Journeys, this field is mandatory. All other fields are defined by your schema.

If you are exporting data as a CSV file, a single quotation mark (`'`) prepends cells that begin with `=`, `+`, `–`, `@`, or `"`.

See Analytics Events API.

Analytics Browser Requests Data

Browser Analytics provides details about each browser user request. The data collected includes information about pages, performance metrics, location of your users, browser and device data, errors, and any custom data that you configured in your Browser RUM configuration.

This page lists the field names available for browser request Analytics data, along with the Events Service internal name. When accessing the data using ADQL queries, you need to use the Events Service internal name in the query. See Browser RUM Metrics.

Event Type: browser_records

Key (event type identifier): appkey

UI Field Name	Events Service Internal Name
Page Info Fields	
Agent ID	`agentid`
appkey	`appkey`
Client GUID	`cguid`
IP address	`ip`
Page Experience	`pageexperience`
Page Name	`pagename`
Page Type	`pagetype`
Parent Page URL	`pageparenturl` ⚠ The `pageparenturl` is not available by default. On-premises customers need to run a special script on the EUM Server. SaaS customers need to file a ticket requesting AppDynamics to do this for them. See the Community article.
Page Referrer	`referrer` See the Community article.
URL	`pageurl`
Performance Time Fields	
Application Server Time	`metrics.`Application Server Time (ms)``
DOM Ready Time	`metrics.`DOM Ready Time (ms)``
Domain Lookup Time	`metrics.`Domain Lookup Time (ms)``
End User Response Time	`metrics.`End User Response Time (ms)``
Estimated BT Time	`btdata.estimatedtime`
First Byte Time	`metrics.`First Byte Time (ms)``
Server Connection Time	`metrics.`Server Connection Time (ms)``
SSL Handshake Time	`metrics.`SSL Handshake Time (ms)``

Location Fields	
City	`geocity`
Country	`geocountry`
Region	`georegion`
Browser and Device Data Fields	
Browser	`browser`
Browser Version	`browserversion`
Device	`device`
Device OS	`deviceos`
Errors Field	
AJAX Error	`ajaxerror`
Error Type	`errortype`
Script Error Field	
Line Number	`scripterrordata.linenumber`
Message	`scripterrordata.message`
Origin	`scripterrordata.origin`
timestamp	`scripterrordata.timestamp`
Custom User Data Field	

your_field_name	`userdata.your_field_name`
	`userdataBoolean.your_field_name`
	`userdataDate.your_field_name`
	`userdataLong.your_field_name`
	`userdataDouble.your_field_name`
	
	Note: if your field name contains spaces or any reserved characters, the field name must be surrounded by backquotes. For example, if you define a custom field called "Item Purchased", the Events Service field name is `userdata.\`Item Purchased\``

Analytics Browser Sessions Data

This page describes the field names available for Analytics Browser Sessions data, along with the Events Service internal name. When accessing the data using ADQL queries, you need to use the Events Service internal name. You can view the Browser Sessions using the Analytics Query API. This page describes fields that are specific to Browser Sessions.

Event Type: web_session_records

Key (event type identifier): appkey

UI Field Name	Events Service Internal Name
Session Fields	
Browser	`browser`
Browser Version	`browserversion`
City	`geocity`
Closed	closed
Country	`geocountry`
Device	`device`
Device OS	`deviceos`
IP Address	`ip`
Page View Count	`pageviews`
Region	`georegion`
Session Duration (ms)	`metrics.durationMS`
Session Start Time	`startTimeMS`
User Experience	`experience`
Page View Fields	
Client GUID	`browserRecords.cguid`
Page Error Type	`browserRecords.errortype`
Page Experience	`browserRecords.pageexperience`
RUM Page Name	`browserRecords.pagename`
Custom User Data	

your_field_name	`userdata.your_field_name`
	`userdataBoolean.your_field_name`
	`userdataDate.your_field_name`
	`userdataLong.your_field_name`
	`userdataDouble.your_field_name`
	
	If your field name contains spaces or any reserved characters, the field name must be surrounded by backquotes. For example, if you define a custom field called "Item Purchased", the Events Service field name is `userdata.`Item Purchased`.`

Analytics Mobile Crash Reports Data

This page describes the field names available for Analytics Mobile Crash Reports data, along with the Events Service internal name. When accessing the data using ADQL queries, you need to use the Events Service internal name in the query.

Event Type: mobile_crash_reports

Key (event type identifier): appkey

UI Field Name	Events Service Internal Name
App Crash Time	`appcrashtimestamp`
App Key	`appkey`
Carrier	`carrier`
Connection Type	`connectiontype` For example: "wifi", "4g", "cell"
Country	`geocountry`
Crash Group ID	`groupid`
Crash Id	`crashid`
Crashed Function	`crashfunction` For example: "-[MTFirstViewController segFault413]", "com.appd.examples.android.instrumentation.MainActivity.d42" This value depends on your application.
Device / Manufacturer	`devicemanufacturer` For example: "Samsung", "Apple"
Mobile App Name	`mobileappname`
Mobile App Version	`mobileappversion`
Model	`devicemodel` For example: "iPhone 5" "iPad 2 WIFI"
OS Version	`osversion` For example: "iOS 6.0", "Android 4.2"
Platform	`platform` For example: "Android", "iOS"
Region	`georegion`
User Data	

your_field_name	`userdata.your_field_name`
	`userdataBoolean.your_field_name`
	`userdataDate.your_field_name`
	`userdataLong.your_field_name`
	`userdataDouble.your_field_name`
	
	If your field name contains spaces or any reserved characters, the field name must be surrounded by backquotes. For example, if you define a custom field called "Item Purchased", the Events Service field name is `userdata.`Item Purchased``

Analytics Mobile Requests Data

Mobile Analytics provides details about the performance of your mobile apps as experienced by your end users. The data collected includes information about the mobile app names and versions, network requests, performance times, locations, carrier and device data, errors, and any custom data that you configured in your Mobile RUM configuration.

This page describes the field names available for mobile requests Analytics data, along with the Events Service internal name. See Mobile RUM Metrics. W hen accessing the data using ADQL queries, you need to use the Events Service internal name in the query.

Event Type: mobile_snapshots

Key (event type identifier): appkey

UI Field Name	Events Service Internal Name
Mobile App Info Fields	
agentid	`agentid`
agentversion	`agentversion`
appkey	`appkey`
cguid	`cguid`
Mobile App Name	`mobileappname` For example: "com.appdynamics.eum.test.apps.E2E-PictureSharingApp"
Mobile App Version	`mobileappversion`
Platform	`platform`
Network Request Fields	
Background	`happenedinbackground`
Experience	`networkrequestexperience`
HTTP Status Code	`httpstatuscode`
Network Request Name	`networkrequestname`
Network Request Type	`networkrequesttype`
Request Content Length	`requestContentLength`
Response Content Length	`responseContentLength`
Performance Time Fields	
Estimated BT Time	`btdata.estimatedtime`
Network Request Time	`networkrequesttime`
Location Fields	
Country	`geocountry`
Region	`georegion`
Carrier and Device Data Fields	
Carrier	`carrier`

Connection Type	`connectiontype` For example: "wifi", "4g", "cell"
Device / Manufacturer	`devicename` For example: "Samsung", "Apple"
Model	`devicemodel` For example: "iPhone 5" "iPad 2 WIFI"
OS Version	`osversion` For example: "iOS 6.0", "Android 4.2"
Error Fields	
Network Type Error	`networkerror`
Custom User Data Fields	

your_field_name	`userdata.your_field_name`
	`userdataBoolean.your_field_name`
	`userdataDate.your_field_name`
	`userdataLong.your_field_name`
	`userdataDouble.your_field_name`
	
	If your field name contains spaces or any reserved characters, the field name must be surrounded by backquotes. For example, if you define a custom field called "Item Purchased", the Events Service field name is `userdata.`Item Purchased``

Analytics Mobile Sessions Data

This page describes the field names available for Analytics Mobile Sessions data, along with the Events Service internal name. When accessing the data using ADQL queries, you need to use the Events Service internal name in the query.

Event Type: mobile_session_records

Key (event type identifier): appkey

UI Field Name	Events Service Internal Name
Mobile Session Fields	
Mobile App Name	`mobileappname` example: "com.appdynamics.eum.test.apps.E2E-PictureSharingApp"
Carrier	`carrier` For example: "AT&T", "Verizon"
Closed	`closed`
Connection Type	`connectiontype` For example: "cell", "3g"
Country	`geocountry`
Device / Manufacturer	`devicename`
Duration	`metrics.durationMS`
IP Address	`ip`
Mobile App Version	`mobileappversion`
Model	`devicemodel` For example: "iPad 2 WIFI", "Kindle Fire"
OS Version	`osversion` For example: "iOS 5.1", "Android 4.2"
Region	`georegion`
Session Start Time	`startTimeMS`
Session Status	`closed` Note "closed" is a Boolean field. True means the session is closed. False means the session is still going on.
Event Count Fields	
Breadcrumb Count	`metrics.breadcrumbcount`
Crash Count	`metrics.crashcount`
Custom Metric Count	`metrics.custommetriccount`
Event Time Stamp	`eventTimestamp`
Info Point Count	`metrics.infopointcount`

Network Request Count	`metrics.networkrequestcount`
Timer Count	`metrics.timerCount`
UI Event Count	`metrics.uiCount`
User Data Count	`metrics.userdatacount`
Crash Fields	
App Crash Time	`crash.appcrashtimestamp`
Crash Exception	`crash.crashexception` For example: "java.lang.NullPointerException"
Crash File	`crash.crashfile` For example, "MainActivity.java"
Crash Group ID	`crash.groupid`
Crash Line Number	`crash.crashlinenumber`
Error Description	`error.description`
Symbolicated or Deobfuscate	`crash.symbolicateddeobfuscated` For example: false
User Data	

your_field_name	`userdata.your_field_name`
	`userdataBoolean.your_field_name`
	`userdataDate.your_field_name`
	`userdataLong.your_field_name`
	`userdataDouble.your_field_name`
	
	If your field name contains spaces or any reserved characters, the field name must be surrounded by backquotes. For example, if you define a custom field called "Item Purchased", the Events Service field name is `userdata.`Item Purchased``

Breadcrumb Fields	
Breadcrumb Text	`breadcrumb.bctext`
Network Request Fields	
Connection Type	`networkrequest.connectiontype`
End Time	`networkrequest.endTimeMS`
Experience	`networkrequest.networkrequestexperience`
HTTP Status Code	`networkrequest.httpstatuscode`
IP Address	`networkrequest.ip`
Network Error	`networkrequest.networkerror`
Network Request Name	`networkrequest.networkrequestname`
Network Request Time	`networkrequest.networkrequesttime`

Infopoint Fields	
Duration (ms)	`infopoint.duration`
infopname	`infopoint.infopname`
Is Dynamic	`infopoint.isdynamic`
UI Event Fields	
Activity	`ui.activity`
Class	`ui.uiclass` For example: "UITableView"
Label	`ui.uilabel` For example: "IOS TableCell"
Root View	`ui.rootview` For example: "SettingsScreen"
UI Event	`ui.uievent` For example: "Button Pressed", "Root View Change", "Table Cell Selected"

Analytics Mobile Non-Fatal Issues Data

This page describes the field names available for Analytics Mobile Non-Fatal Issues data, along with the Events Service internal name. When accessing the data using ADQL queries, you need to use the Events Service internal name in the query.

Event Type: mobile_non_fatal_issue_records

UI Field Name	Events Service Internal Name
Agent Info Fields	
Agent ID	`agentid`
Agent Version	`agentversion`
App Info Fields	
App Key	`appkey`
Mobile App Name	`mobileappname` For example: "com.appdynamics.eum.test.apps.E2E-PictureSharingApp"
Mobile App Version	`mobileappversion`
Device Info Fields	
Carrier	`carrier` For example: "AT&T", "Verizon"
Connection Type	`connectiontype` For example: "cell", "3g"
Device Manufacturer	`devicemanufacturer` For example: "Samsung"
Device Model	`devicemodel` For example: "Galaxy S5"
Device Name	`devicename`
IP Address	`ip`
jailbroken	`jailbroken`
OS Version	`osversion` For example: "5.1", "4.2"
Platform	`platform` For example: "iOS", "Android"
Code Issues	
description	`description` For example: "App Not Responding in Utils.formatData"
filename	`filename`
groupid	`groupid`
Line Number	`linenumber`
severity	`severity` For example: "Info", "warning"

symbolicateddeobfuscated	`symbolicateddeobfuscated` For example: false
symbolname	`symbolname` For example: "MainActivy.onResume"
pickupTimestamp	`pickupTimestamp`
Timestamp	`timestamp` For example: "03/15/18 5:32:33 PM"
Custom User Data Fields	

your_field_name	`userdata.your_field_name`
	`userdataBoolean.your_field_name`
	`userdataDate.your_field_name`
	`userdataLong.your_field_name`
	`userdataDouble.your_field_name`
	
	If your field name contains spaces or any reserved characters, the field name must be surrounded by backquotes. For example, if you define a custom field called "Item Purchased", the Events Service field name is `userdata.`Item Purchased``

Event Fields	
eventid	`eventid`
eventindex	`eventindex`
timestamp	`eventTimestamp`
eventtype	`eventtype` For example: "ANR", "Error"
Location Fields	
City	`geocity`
Country	`geocountry`
Region	`georegion` For example: "California", "Sichuan"

Analytics Synthetic Sessions Data

This page lists the field names available for Analytics Browser Synthetic Sessions data, along with the Events Service internal name. When accessing the data using ADQL queries, you need to use the Events Service internal name in the query. For details on the meaning of each metric, see Browser Synthetic Metrics.

Event Type: `synth_session_records`

Key (event type identifier): Job Name (scheduleName)

UI Field Name	Events Service Internal Name
Session Fields	
Browser	`browser` For example: "Internet Explorer", "Chrome"
Browser Version	`browserversion` For example: "52.0.2743.116", "11.0.9600.18124"
City	`geocity`
Closed	`closed`
Country	`geocountry`
Devices	`device` The computer form factor of the device. For example: "Computer" or "Mobile & Tables"
Device Model	`devicemodel` The model of the device. For example: "Galaxy S8" or "iPhone 7"
Device OS	deviceos The operating system of the device. For example: "Windows" or "Android"
Duration (sec)	`metrics.`Session Duration (ms)``
End Time	`endTimeMS`
Error Type	`failureType` For example: "Timed out", "Test script crashed"
Job Name	`scheduleName` The name of the synthetic job.
Region	`georegion`
Result	`success`
Schedule ID	`measurementSpec.scheduleId`
Session Type	`sessionType` For example: "WEB_DRIVER", "URL"
Session Visual Time (sec)	`metrics:`Session Visual Time (ms)``
Start Time	`startTimeMS`
Status	`status`
Timestamp	`eventTimestamp`
Page View Fields	
Page Error Type	`browserRecords.pageerror`
RUM Page Name	`browserRecords.pagename`
URL	`pageurl`

Analytics Connected Device Data

Connected Device Data provides details about each IoT connected device. The data collected includes information about devices, network requests and error events, and any custom properties and events that you instrument in your IoT application.

This page describes the required field names for Analytics Connected Device data, along with the Events Service internal name. When accessing the data using ADQL queries, you need to use the Events Service internal name in the query. Because IoT Monitoring largely depends on custom data, your connected device Analytics data will most likely include many more field names than given below.

Event Type: iot_records

Key (event type identifier): appkey

UI Field Name	Events Service Internal Name
Fields	
App Key	`appkey`
deviceguid	`deviceguid`
devicetype	`devicetype`
Timestamp	`eventTimestamp`
eventtype	`eventtype`
customevent Fields	
eventsummary	`customevent.eventsummary`
networkrequestevent Fields	
duration	`networkrequestevent.duration`
networkerror	`networkrequestevent.networkerror`
statuscode	`networkrequestevent.statuscode`
URL	`networkrequestevent.url`
errorevent Field	
Resource Name	`errorevent.name`

Analytics Data Sources

Related pages:

- ADQL Data

This page describes how Analytics can collect data from many sources in your AppDynamics APM Platform.

Transaction Analytics Data Sources

Transaction Analytics data is collected by the AppDynamics Java, .NET, PHP and Node.js App Agents. Collecting Transaction Analytics data requires no change to your application code. You enable analytics on the app server agents and Controller you already use.

Once you have enabled analytics on your application, you can collect and analyze several kinds of data:

- Default performance data collected by the app agents about your application's business transactions.
- HTTP-based data collected by HTTP data collectors.
- Custom data collected by method invocation data collectors.
- Business data collected from parameterized SQL calls.

See Collect Transaction Analytics Data.

Log Data Sources

You can collect Log Analytics data from many types of log files, including instrumented and non-instrumented applications as well as infrastructure. Log Analytics can be used as a standalone component. You can search and analyze log data just as you do transaction data. Log Analytics works by default with the syslog (log4j) format and can be configured for other log formats, including GZIP files (log files ending in .gz). You can capture other log formats by setting up regular-expression-based mapping. See Collect Log Analytics Data.

You can view correlated Log and Transaction Analytics Data. By configuring business transaction GUID injection, you can see logs that are related to specific business transaction requests. See Business Transaction and Log Correlation.

End User Monitoring Data Sources

Analytics can also use data collected by Browser Real User Monitoring, Mobile Real User Monitoring and IoT Monitoring. Analytics makes the data available in a more flexible search format. If you have enabled Browser RUM, Mobile RUM or IoT Monitoring *and* enabled Analytics, you see these event types on the Analytics search page.

For details on how analytics extends the capability of EUM, see the section "*Browser Analyze versus Browser Request Analytics*" in Browser RUM Analyze.

Browser Synthetic Data Sources

Sessions data captured through Browser Synthetic Analytics is available in Analytics.

Custom Event Sources

You can add custom events for Analytics using the AppDynamics Analytics Events API.

Deploy Analytics With the Analytics Agent

Related pages:

- Monitor JVMs
- Monitor .NET Nodes

This page describes the deployment options for Analytics, which includes the Events Service.

Transaction Analytics Only

You need an Analytics Agent to capture Transaction Analytics for:

- Java Agents <= 4.5.15
- Any version of .NET, Node.js, and PHP Agents
- Controller <= 4.5.15

To configure Transaction Analytics:

- Download and install either the Standalone Machine Agent or the Analytics Agent (standalone binary) on each machine. You can also use a shared Analytics Agent instance.
- Enable Analytics for applications in the Controller.

Log Analytics Only

To configure Log Analytics:

- Install either the Standalone Machine Agent or the Analytics Agent (standalone binary) on the machine.
- Enable Analytics for log sources in the Controller.

Transaction and Log Analytics

To configure Transaction and Log Analytics:

- Install either the Standalone Machine Agent or the Analytics Agent (standalone binary) on the machine.
- Enable Analytics for applications and log sources in the Controller.

SaaS/On-premises Analytics Configuration

When you deploy Analytics with the Analytics Agent, the configuration requirements differ based on your deployment.

- SaaS: The Events Service is provided as part of the SaaS service. AppDynamics stores the data and hosts the server components of the system for you. You need to install only the Analytics Agent components of the system described in Agent Side Components.
- On-premises: You host the components yourself, storing all data on-premises. You need to install the Agent Side Components, Controller Deployment Components, and Events Service Deployment Components. While the on-premises deployment involves additional setup and administration, you are able to retain the Analytics data in your own data center.

Encrypt Data on the Analytics Agent

You can encrypt any data on the Analytics Agent using `secure://<your-encrypted-credentials>`. You can encrypt data in the Analytics Agent properties file or in System Properties.

The following example demonstrates how to encrypt `http.event.accessKey` in the Analytics Agent properties file.

```
http.event.accessKey=secure://-001-24-Dr9FQGC179o4vPnuljnx8A==ZGVw/P4OONvpUidIhJ2u78FpRVVW8fbgr8J1HBHXwnE=
ad.secure.credential.store.filename=/opt/appdynamics/secretKeyStore
ad.secure.credential.store.password=s_gsnwR6+LDch8JBf1RamiBoWfMvjjipkrtJMZXAYEkw8=
```

See Encrypt Agent Credentials.

Separate Host for the Analytics Agent

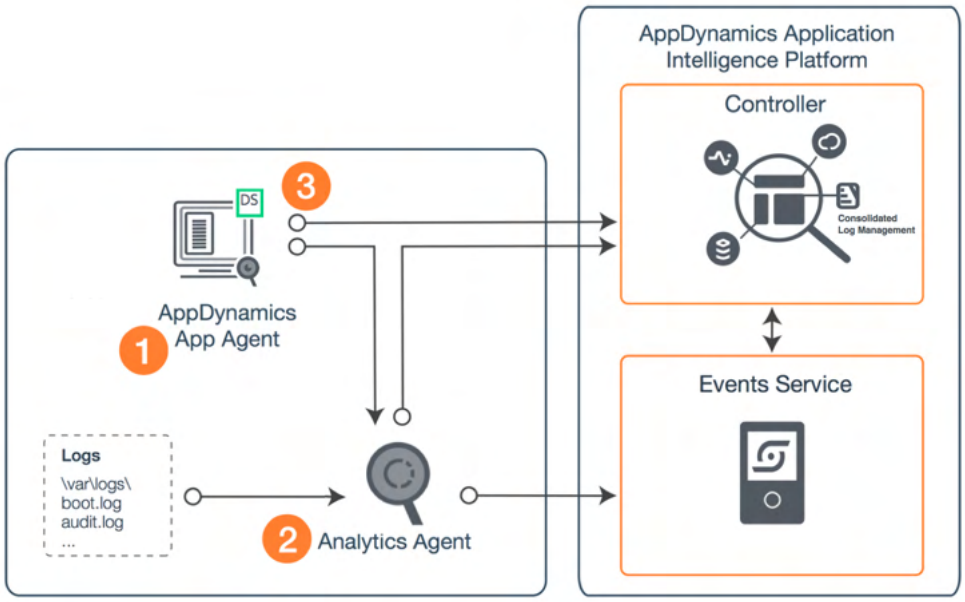
If your Analytics Agent is on a host separate from the monitored application, see Remote Analytics Agent Sizing.

Analytics Agent Components

This page describes Analytics Agent architecture.

Analytics Agent-Side Components

For both SaaS and on-premises Analytics, you need to deploy (or enable) agent-side Analytics components in your application environment.



Legend: Agent Components

1 AppDynamics App Agent: Analytics relies upon the same app server agents that AppDynamics APM uses. If you use AppDynamics APM, you likely already have these deployed to your monitored applications.

2 Analytics Agent: The Analytics Agent collects data from one or more app server agents and sends it to the Events Service. It also reads and transmits log data from log files from the local machine. The Analytics Agent is available as a standalone binary or as part of the Standalone Machine Agent download.

- As part of the Standalone Machine Agent download, the Analytics Agent runs as an extension (also called a monitor).
- A standalone binary distribution of the Analytics Agent is available for Windows environments or other environments that do not have the Standalone Machine Agent installed.

3 Analytics Dynamic Service: The Analytics Dynamic Service extends your app server agent functionality to collect and forward data to the Analytics Agent. It is built into the Java and .NET Agents but is not enabled by default. No additional download is needed.

Browser and Mobile Analytics

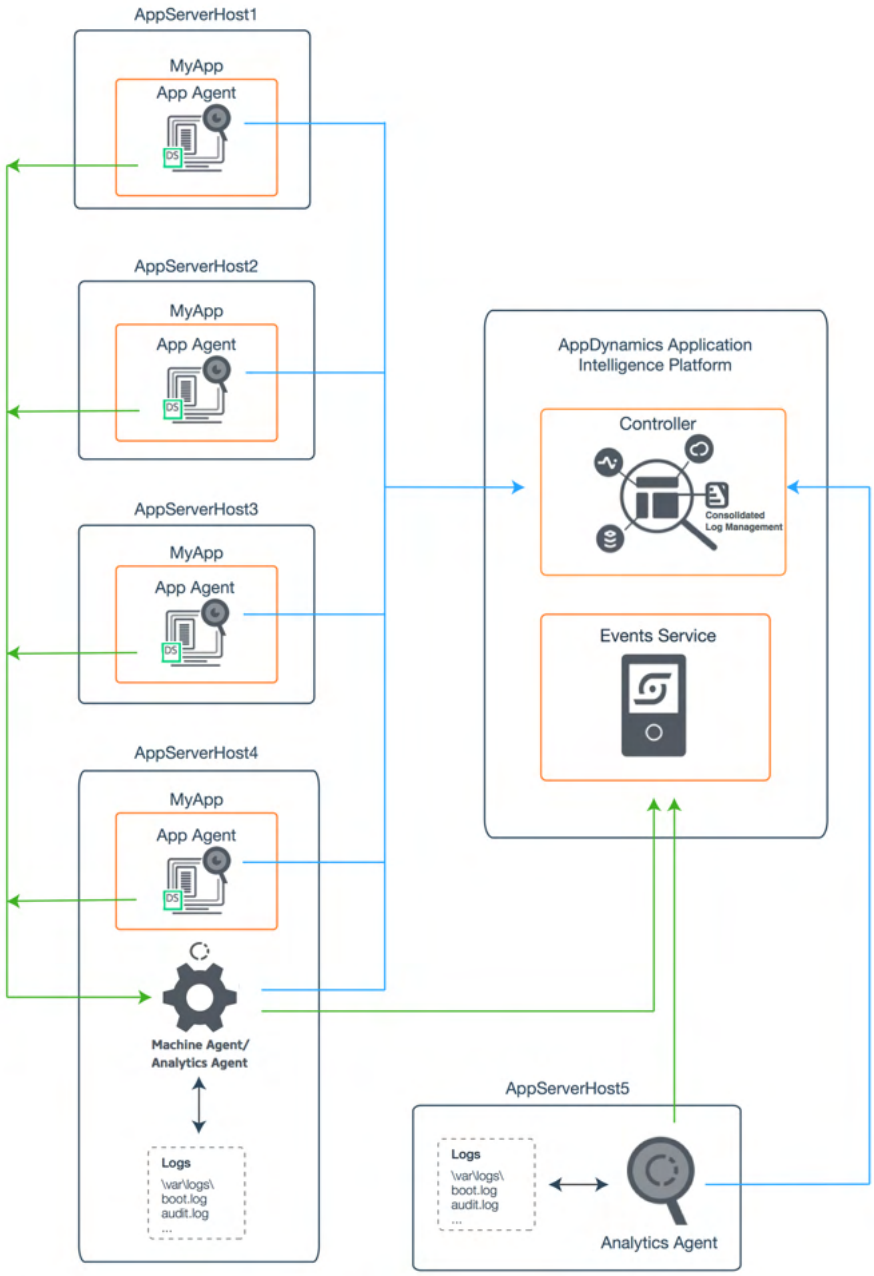
To see Analytics data from Browser and Mobile Real User Monitoring, you need to enable those components and enable Analytics. See EUM Server Deployment.

Deploying Analytics Agents to Multiple Nodes

A real-world scenario is unlikely to consist of a single monitored node, as shown in the diagram above. It usually consists of many applications deployed over many hosts.

While the APM app server agents continue to send data to the Controller in the normal way, the Analytics Dynamic Service sends its data to the Analytics Agent. This agent runs in a separate JVM process in the local environment or network, either as part of the Standalone Machine Agent or on its own.

There must be at least one Analytics Agent in the monitored environment, although multiple app server agents collecting only transaction data can share a single Analytics Agent, as shown in the diagram below. However, each machine where you want to gather log data must have its own Analytics Agent instance.

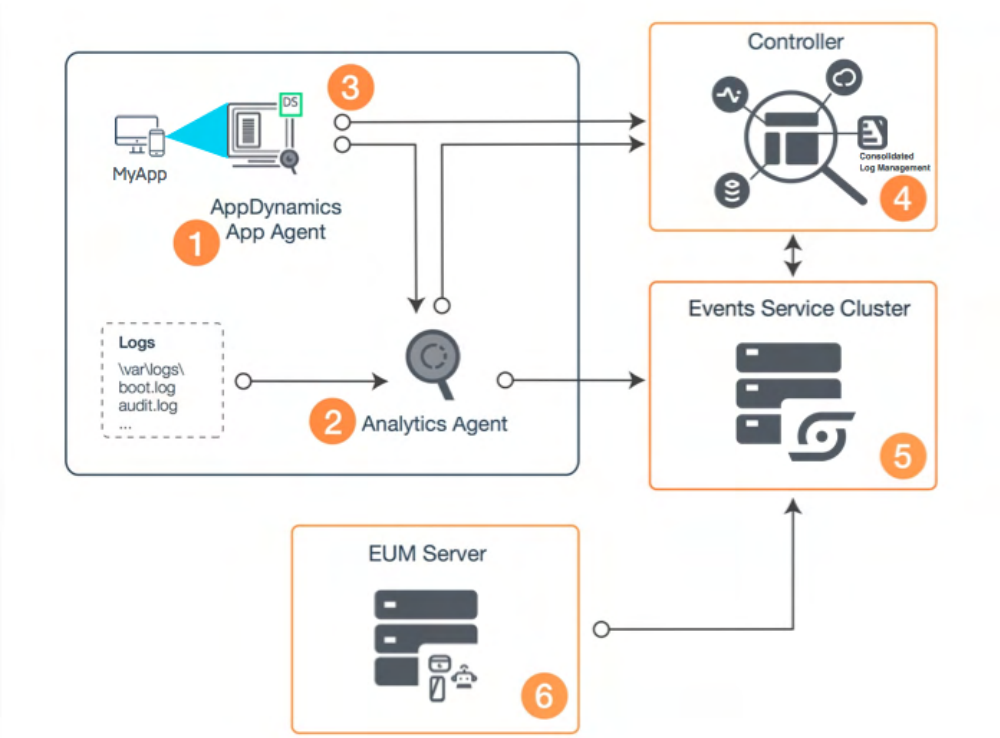


In this example, AppServerHost1-3 are collecting only Transaction data, so only app server agents are required. Each one connects to the Analytics Agent on AppServerHost4, where log information is also being collected. The Analytics Agents could also reside on the same machines as each app server agents. On AppServerHost5, only log data is being collected, so an Analytics Agent is also necessary there.

Server Side Components for On-premises

To set up AppDynamics Analytics on-premises, you also need to install the server parts of the system, the AppDynamics Controller, and the Events Service.

The following figure depicts the basic components of an on-premises deployment:



Legend: Server Components

> If you are using SaaS EUM, you must use the SaaS Events Service. If you are using the on-premises EUM Server, you must use an on-premises instance of the Events Service.

4 AppDynamics Controller: The heart of an AppDynamics deployment, the Controller processes and presents the information gathered by the agents.

5 Events Service: The unstructured document store, it gathers and stores data from the Analytics, Database, and Network Visibility Agents and, if you have End User Monitoring, from 6, the EUM Server. It allows the Controller UI to run queries on that data.

Install Agent-Side Components

On this page:

Related pages:

- Analytics Agent Components

This page provides an overview of how to configure Application Analytics components to collect business transaction data.

In most Application Analytics deployment scenarios, you must enable the Analytics Agent-side components. The agents are distributed as system-based operations or standalone bundles in the AppDynamics Downloads Portal.

If your app server agent supports Agentless Analytics, such as the Java Agent (4.5.15 and later) and .NET Agent (20.10 and later), you do not need to install the standalone Analytics Agent or Machine Agent extension for Transaction Analytics data collection. See Deploy Analytics Without the Analytics Agent for more information.

Prepare to Enable the Agents

Review the deployment options, agent-side component architecture, and components you will need to install in Deploy Analytics With the Analytics Agent.

Analytics Agent

The Analytics Agent collects and sends data from the AppDynamics App Agent and log files to the Events Service. The Analytics Agent is not enabled by default. You need to download and enable the desired bundle using a method of your choice.

To use Application Analytics:

1. Obtain a separate Application Analytics license.
2. Enable the Analytics Agent.
3. Point to the Event Service. See Revise the Analytics Agent Properties File for agent-specific configuration and Event Service Deployment for the Event Service.
4. Enable Analytics on the Controller.

AppDynamics App Agent

To access Analytics and collect transaction data from an application, you must deploy a supported app server agent version in all environments, such as the Node.js Agent or the PHP Agent. If you already use AppDynamics APM, agents may already be installed in your environment.

Install the Standalone Analytics Agent on Windows

As described in Analytics Deployment Options, the Analytics Agent can run on the same host or on a different host from the app server agent, depending on your use case.

The AppDynamics Downloads Portal provides the following distribution archives:

- Standalone Analytics Agent (no JRE)
- Analytics Agent with JRE 1.8 for both 32-bit and 64-bit Windows machines

Revise the Analytics Agent Properties File

The Analytics Agent properties file determine how your Analytics Agent communicate with other components, what type of data it gathers from the monitored application, and how the components of the Application Analytics deployment and the Controller authenticate to each other.

Modify the default configuration and set the desired values for the agent properties in the properties file. The location of this file depends on your exact deployment scenario.

1. Open the `<analytics-agent-home>/conf/analytics-agent.properties` file with a text editor. The Analytics Agent on Windows requires double backslash for paths.
   For example:

```
conf.dir=C:\\AppD\\analytics-agent\\conf
ad.dw.log.path=C:\\AppD\\analytics-agent\\logs
```

2. To run the Analytics Agent as a Windows service, modify the following properties:
    - `ad.dw.log.path=<analytics-agent-home>\\logs`
    - `conf.dir=<analytics-agent-home>\\conf`
    - ad.jvm.options.name=analytics-agent.vmoptions
    - `ad.jvm.heap.min=512m`
    - `ad.jvm.heap.max=1g`
3. Follow the instructions to enable the standalone Analytics Agent.
4. To install the Analytics Agent on Windows, run the `install` command:

```
bin\analytics-agent.exe service-install
```

Start and Stop the Analytics Agent

To start/stop the Analytics Agent on Windows, run the following commands in the Windows Services menu.

To start the agent, run:

```
bin\analytics-agent.exe service-start
```

To stop the agent, run:

```
bin\analytics-agent.exe service-stop
```

Uninstall the Analytics Agent

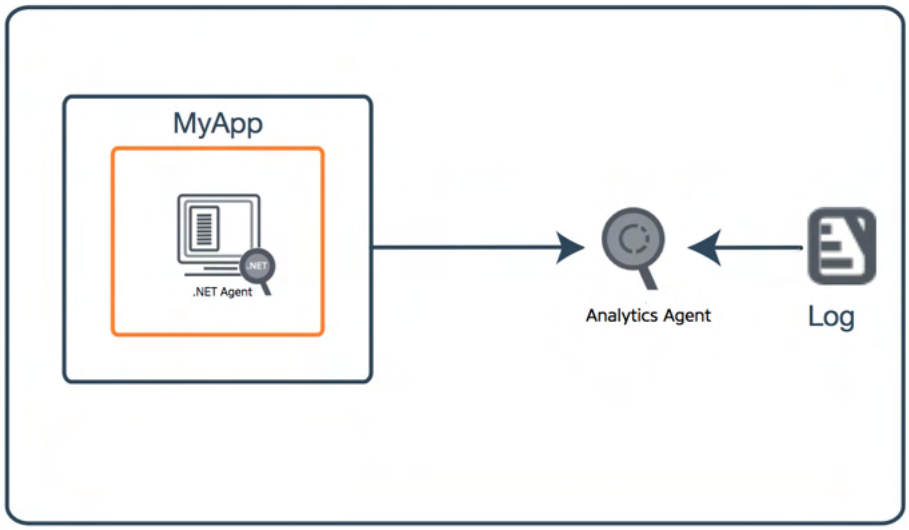
To uninstall the Analytics Agent, run the `.exe` file with the `uninstall` command:

```
bin\analytics-agent.exe service-uninstall
```

Enable Analytics Agent for a Local App Server Agent

This section assumes that you are installing the Analytics Agent on the same host as your app server agent.

In this deployment, the Analytics Agent reads and transmits log data in log files from the local machine. The app server agent transmits data from the monitored application to the Analytics Agent.

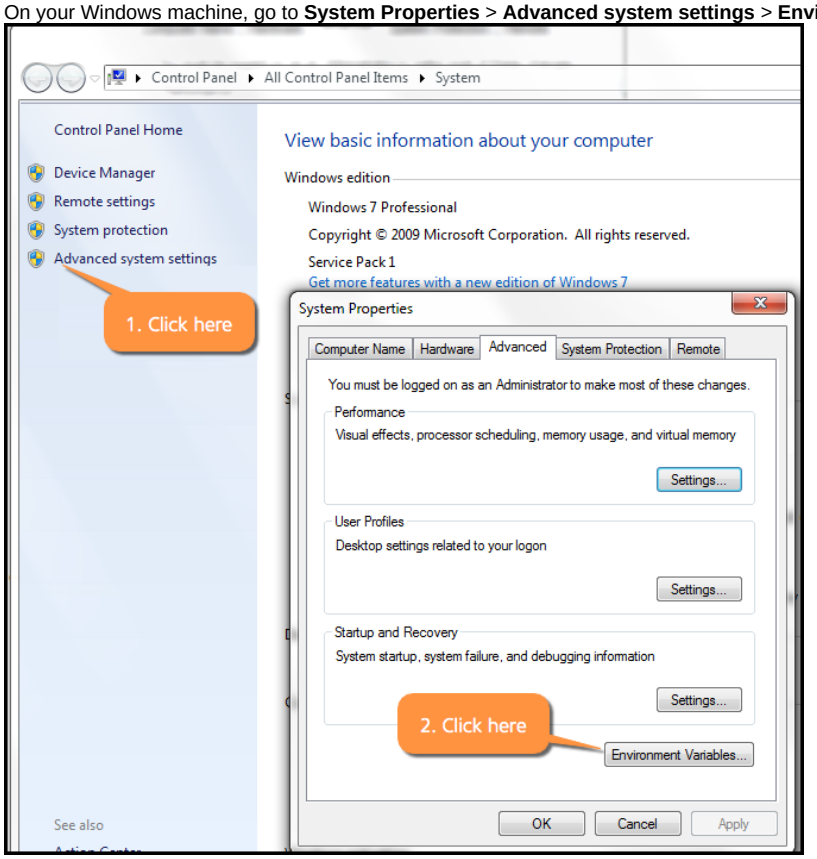


Install the Analytics Agent with the `.exe` and `analytics-agent.exe` files found in the *<analytics-agent-home>*`/bin` directory. Because the Analytics Agent is written in Java, you will run the agent in a Java Virtual Machine (JVM). The Analytics Agent runs as a Windows service.

Enable the App Server Agent for a Remote Analytics Agent

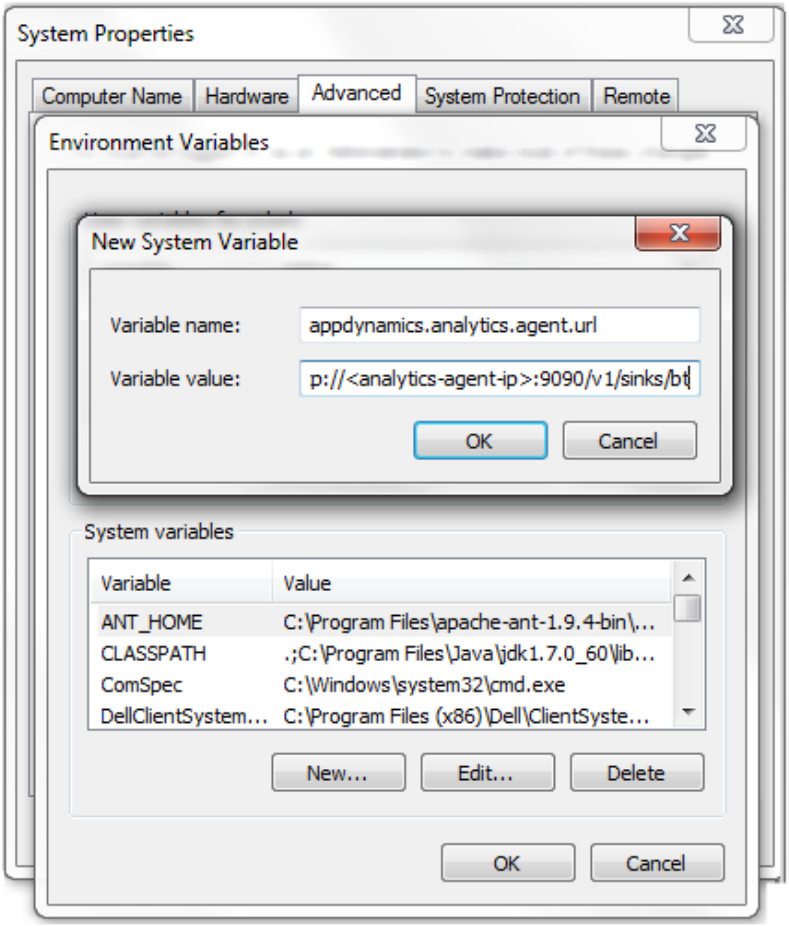
To specify the location of the remote Analytics Agent with an environment variable named `appdynamics.analytics.agent.url` on Windows:

1. On your Windows machine, go to **System Properties** > **Advanced system settings** > **Environment Variables**.



2. Under **System Variables**, click **New**. AppDynamics recommends a system environment variable approach as opposed to a user environment variable approach; in the environment variable approach, a user must have the same permissions as the user with all the instrumented applications running.

3. Set the value for the `appdynamics.analytics.agent.url` system variable to `http://<analytics-agent-ip>:9090/v2/sinks/bt`. Replace `<analytics-agent-ip>` with the hostname of the Analytics Agent for your environment.



4. Restart the applicable application or process where you want the new environment variables to take effect. While it is not required that you restart your machine, the parent process that invokes the monitored process must be restarted. For `w3wp` on Windows, restart IIS by running `iisreset` after the changes are made to the environment variables.

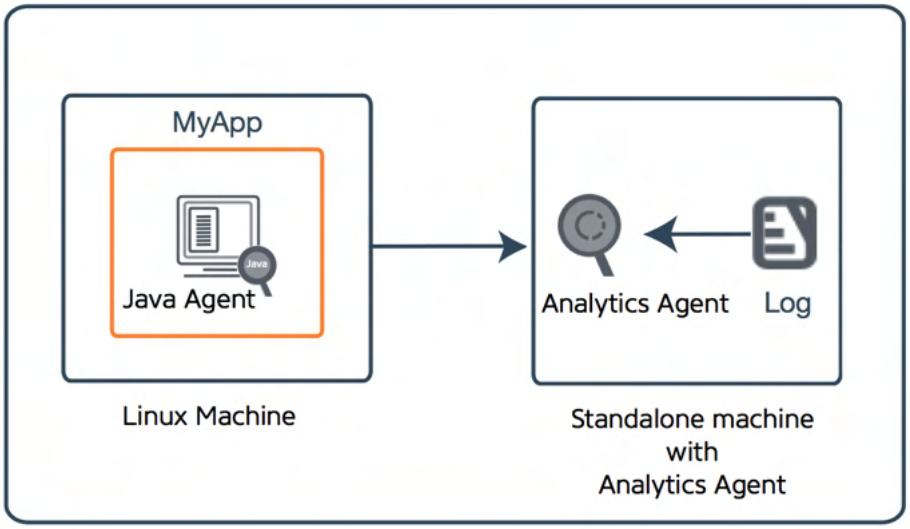
Install the Standalone Analytics Agent on Linux

As described in Analytics Deployment Options, the Analytics Agent can run on the same host or on a different host from the app server agent, depending on your use case.

The AppDynamics Downloads Portal provides the following distribution archives:

- Standalone Analytics Agent (no JRE)

The default settings of the app server agent typically assume that the Analytics Agent is on the same host and uses the default port. If your Analytics Agent is on a host separate from the app server agent, as shown in the diagram below, or you have changed the default port, you need to specify the new host and port values for the app server agent.



Revise the Analytics Agent Properties File

The Analytics Agent properties file determine how your Analytics Agent communicate with other components, what type of data it gathers from the monitored application, and how the components of the Application Analytics deployment and the Controller authenticate to each other.

Modify the default configuration and set the desired values for the agent properties in the properties file. The location of this file depends on your exact deployment scenario.

1. Open the `<analytics-agent-home>/conf/analytics-agent.properties` file with a text editor.
2. Follow the instructions to enable the standalone Analytics Agent.

Start and Stop the Analytics Agent

You can start and stop the Analytics Agent directly in the command line.

To start the agent, run:

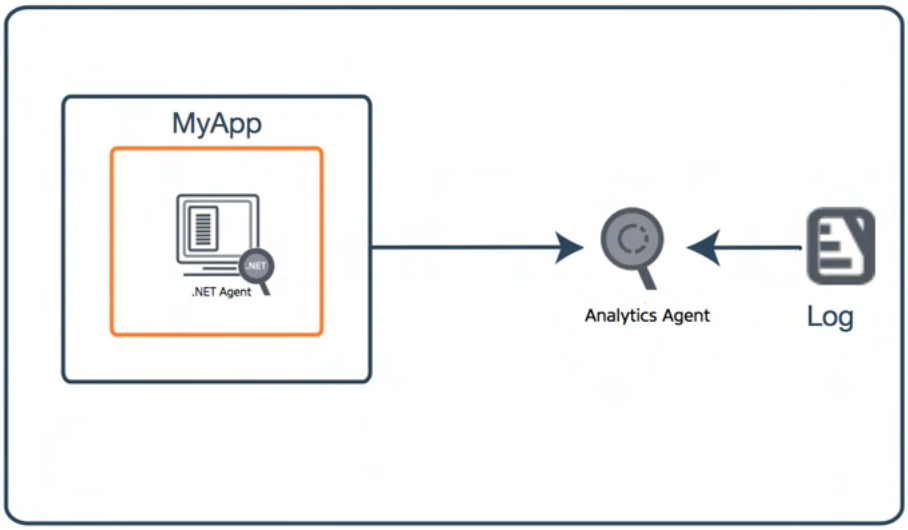
```
<analytics agent executable> start
```

To stop the agent, run:

```
<analytics agent executable> stop
```

Run the Analytics Agent From a Read-Only Filesystem

In this deployment, the Analytics Agent is installed and run from a read-only filesystem. To do so, you need to direct the Analytics Agent to write its log files to a writable partition.



By creating a directory for the Analytics Agent in a writable file system and symlinks for the content of the Analytics Agent in the read-only file system, you will have a writable top-level directory for the Analytics Agent and a writable logs directory under the top-level directory.

To run the Analytics Agent from a separate writable directory using artifacts from a read-only directory:

1. Provide necessary permissions to the read-only Analytics Agent directory.

```
# analytics-agent-readonly refers to the read-only analytics agent.
chmod -R 555 /tmp/analytics-agent-readonly/
```

2. Create a directory for the Analytics Agent in a writable file system.

```
mkdir /tmp/analytics-agent-writable
chmod 755 analytics-agent-writable/
cd /tmp/analytics-agent-writable
```

3. Create symlinks to the contents of the Analytics Agent in the read-only file system.

```
ln -s /tmp/analytics-agent-readonly/bin/ bin
ln -s /tmp/analytics-agent-readonly/lib/ lib
ln -s /tmp/analytics-agent-readonly/monitor.xml monitor.xml
```

4. Create a writable `conf` directory under that top-level directory.

```
cd /tmp/analytics-agent-writable
mkdir conf
chmod 755 conf/
cd conf
```

5. Create symlinks to the contents of the `conf` directory.

```
ln -s /tmp/analytics-agent-readonly/conf/analytics-agent.* .
ln -s /tmp/analytics-agent-readonly/conf/job/ job
ln -s /tmp/analytics-agent-readonly/conf/grok/ grok

cd /tmp/analytics-agent-writable
```

6. Create logs directory under the `analytics-agent-writable` directory.
7. Run the Analytics Agent from the `writable` directory.

```
cd /tmp/analytics-agent-writable
nohup bin/analytics-agent.sh start &
```

8. The resulting directories look similar to:

```
/tmp/
dr-xr-xr-x  6 ec2-user ec2-user 4096 Jul 17 20:45 analytics-agent-readonly
drwxr-xr-x  4 ec2-user ec2-user 4096 Jul 26 20:17 analytics-agent-writable

 /tmp/analytics-agent-writable/
total 24
drwxr-xr-x . . . 4096 Jul 27 19:41 .
drwxrwxr-x  . . .  4096 Jul 27 19:37 ..
lrwxrwxrwx 1 . . .   50 Jul 27 19:38 bin -> /tmp/analytics-agent-readonly/bin/
drwxr-xr-x . . .  4096 Jul 27 19:41 conf
lrwxrwxrwx . . .  50 Jul 27 19:38 lib -> /tmp/analytics-agent-readonly/lib/
drwxrwxr-x . . . 4096 Jul 27 19:41 logs
lrwxrwxrwx . . .   57 Jul 27 19:38 monitor.xml -> /tmp/analytics-agent-readonly/monitor.xml


conf/
total 20
drwxr-xr-x . . . 4096 Jul 27 19:41 .
drwxr-xr-x . . . 4096 Jul 27 19:41 ..
lrwxrwxrwx . . .   77 Jul 27 19:39 analytics-agent.properties -> /tmp/analytics-agent-readonly/conf
/analytics-agent.properties
lrwxrwxrwx . . .   76 Jul 27 19:39 analytics-agent.vmoptions -> /tmp/analytics-agent-readonly/conf
/analytics-agent.vmoptions
lrwxrwxrwx . . .  56 Jul 27 19:40 grok -> /tmp/analytics-agent-readonly/conf/grok/
lrwxrwxrwx . . .  55 Jul 27 19:39 job -> /tmp/analytics-agent-readonly/conf/job/
drwxrwxr-x . . . 4096 Jul 27 21:32 watermark
```

Enable the App Server Agent for a Remote Analytics Agent

For most configurations, install the Analytics Agent on the same machine as the app server agent. In certain setups, such as installing an Analytics Agent extension, you will need a separate machine.

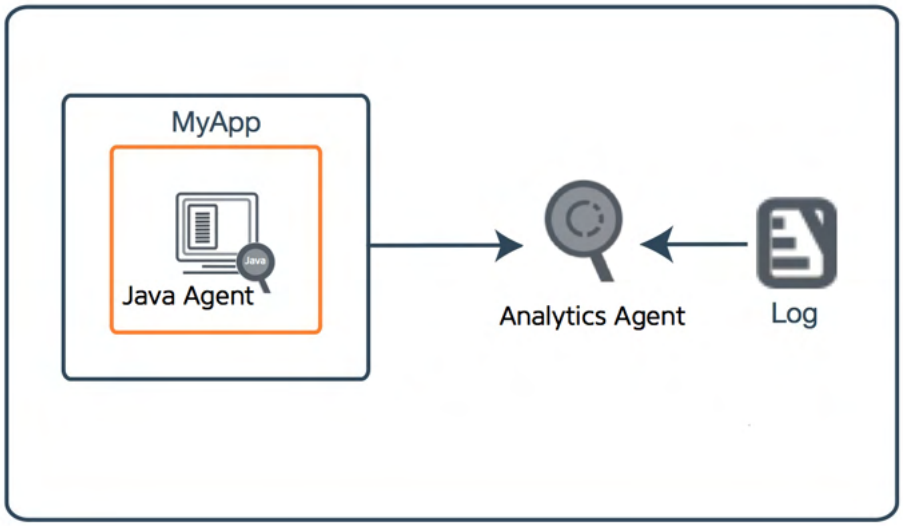
1. Specify the location of the remote Analytics Agent with a `-D` parameter. The argument is passed to the monitored application as follows:

```
-Dappdynamics.analytics.agent.url=http://<analytics-agent-ip>:9090/v2/sinks/bt
```

2. Replace `<analytics-agent-ip>` with the hostname of the Analytics Agent for your environment.

Enable the Standalone Analytics Agent

In this deployment, the Analytics Agent and app server agent runs on the same machine. The Analytics Agent reads and transmits log data from log files on the host machine. The app server agent transmits data from the monitored application to the Analytics Agent.



For environments that do not have a Machine Agent installed, install the Analytics Agent as a separate binary, `analytics-agent.sh`.

1. Unzip the Analytics Agent distribution archive to the installation directory on each target host.
   When you unzip this archive, you get three directories:
   - `bin`: contains the binary file for Linux, Solaris, and OSX
   - `lib`: contains all the jar files that need to be in the classpath
   - `conf`: contains all the configuration files, such as the properties and `vmoptions` files
2. Open the following file to configure connectivity from the Analytics Agent to the Events Service.

   ```
   <analytics-agent-home>/conf/analytics-agent.properties
   ```

3. In the `analytics-agent.properties` file, do the following:

   - Change the default URL and, if necessary, the port number to connect the Events Service by modifying the `http.event.endpoint` value.
     For example:

     ```
     http.event.endpoint=http://<events_service_host:events_service_port>
     ```

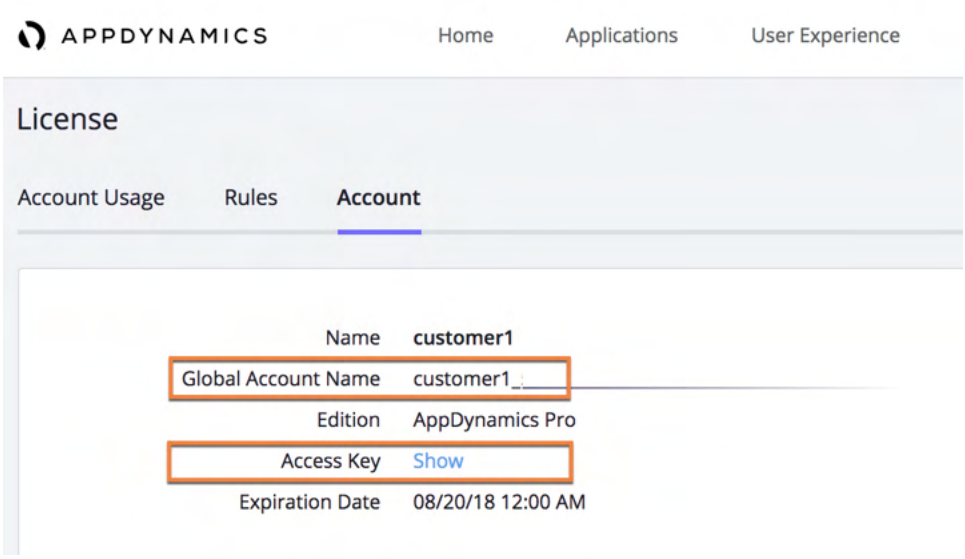
     For SaaS-based installations, the host and port are:

     - `https://analytics.api.appdynamics.com:443` (North America)
     - `https://fra-ana-api.saas.appdynamics.com:443` (Europe)
     - `https://sydn-ana-api.saas.appdynamics.com:443` (APAC)

     If your firewall rules use IP addresses, review firewall considerations in Troubleshoot Analytics Agent Issues. For on-premises installations, use any host and port that you have configured. In clustered environments, this is often a load balancer.
   - Change the default URL and the port number to point to the Controller by modifying the `ad.controller.url` property.
     For example:

     ```
     ad.controller.url=http://<application_server_host_name>:<http-listener-port>
     ```

     This is the URL and the port number you use to access the Controller UI.
4. The **Global Account Name** and **Access Key** values are found under **View License** in the Controller.



5. Configure the account and account key for which the agent should publish the business transaction data. Use the **Global Account Name** and **Access Key** values you have collected earlier. For example:

```
# The global_account_name in the Controller for this analytics data, similar to the following:
http.event.accountName=<customer1_74678b04-8a71-40ef-acaf-xxxxxxxxxxxx>
# Replace this value with the access key of the account name configured above.
http.event.accessKey=<3d58aba2-xxx-xxx>
```

   The `http.event.accountName` property specifies the **Global Account Name** of the account. The `http.event.accessKey` property specifies the **Access Key**, which provides an authentication mechanism between the Controller and the components of the Application Analytics deployment. The Controller installation process generates the **Access Key** value.
   For SaaS-based installations, set the `http.event.name` property to **Name** of the account. The default value is `customer1`. If the property is not configured correctly, the Analytics Agent will not be able to authenticate. The resulting errors are stored in the `analytics-agent.log` file.
6. If you are collecting log information on this host, you need to:
   - Configure your log sources. See Collect Log Analytics Data for details.
   - Update additional properties. See Configure Properties for Consolidated Log Management.
7. Save and close the file.
8. Start the Analytics Agent by running the following command.

```
bin\analytics-agent.sh start
```

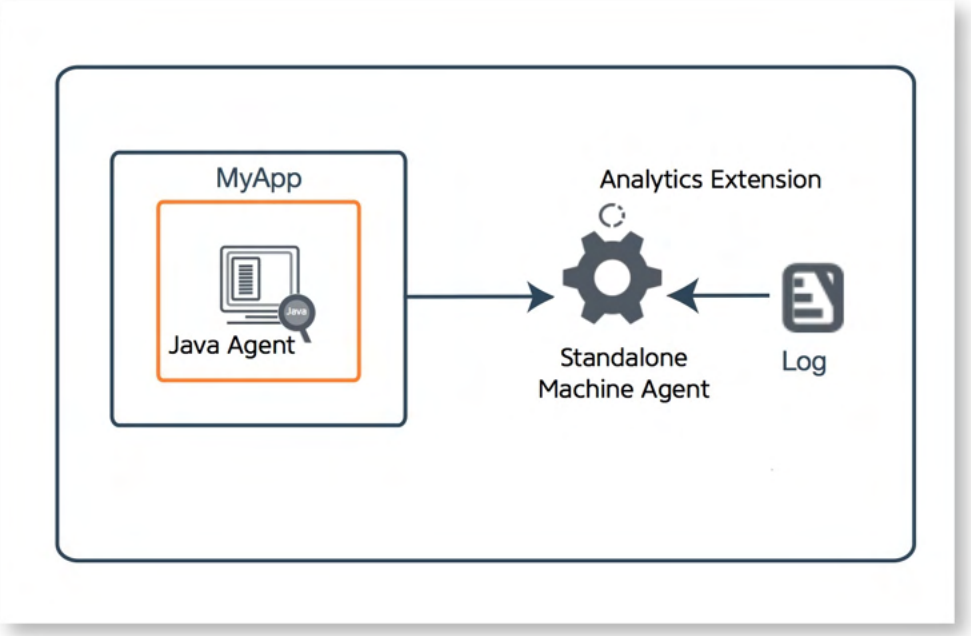
Enable the Analytics Agent as an Extension to the Standalone Machine Agent

This section describes how to configure the Analytics Agent as an extension to the Standalone Machine Agent and run as a Machine Agent monitor.

The AppDynamics Downloads Portal provides the following distribution archives:

- Machine Agent Bundle - 64-bit osx (zip)
- Machine Agent Bundle - for both 32-bit and 64-bit Linux machines (rpm)
- Machine Agent Bundle - for both 32-bit and 64-bit Linux machines (zip)
- Machine Agent Bundle - for 64-bit solaris (zip)
- Machine Agent Bundle - for 64-bit solaris-sparcv9 (zip)
- Machine Agent Bundle - for 64-bit solaris-x64 (zip)
- Machine Agent Bundle - for both 32-bit and 64-bit Windows machines (zip)

If the Machine Agent and app server agent are running on the same machine, installing a standalone Analytics Agent is not necessary. In environments with the standalone Machine Agent already running, you can enable and run the Analytics Agent as an extension.



Confirm that JRE 8 or later is installed on the host running the standalone Machine Agent. If the required version of JRE is not available on the host, then the Analytics Agent cannot be enabled.

1. On the host running the standalone Machine Agent, open `<machine-agent-home>/monitors/analytics-agent/monitor.xml` with a text editor.
2. Set the enabled tag to `true` and save the file when you are finished.

```
<monitor>
    <name>AppDynamics Analytics Agent</name>
    <type>managed</type>
    <!-- Enabling this requires JRE 8 or higher  -->
    <enabled>true</enabled>
...
```

3. Open the following file to configure connectivity from the Analytics Agent to the Events Service:

```
<machine-agent-home>/monitors/analytics-agent/conf/analytics-agent.properties
```

4. Follow the instructions to enable the standalone Analytics Agent.
5. If the Machine Agent is already running at this point, restart it to pick up the changes in the configuration.

> To connect to the Events Service through a proxy server, see Connect the Agent to the Events Service through a Proxy.

Tune Analytics Agent

This section describes the post-installation configuration of the Analytics Agent: configuring Analytics Agent to connect to an Event Service with a proxy server, collecting log files using Consolidated Log Management (CLM), setting resource usage limit, and changing JVM options.

Connect to the Events Service through a Proxy

If the Analytics Agent needs to connect to the Events Service through a proxy server:

1. Open *<analytics-agent-home>*`\conf\analytics-agent.properties` with a text editor.
2. Add this information:

```
# optional proxy properties
http.event.proxyHost=<your proxy host>
http.event.proxyPort=<your proxy port>
http.event.proxyUsername=<your proxy username, if authentication is required>
http.event.proxyPassword=<your proxy password, if authentication is required>
```

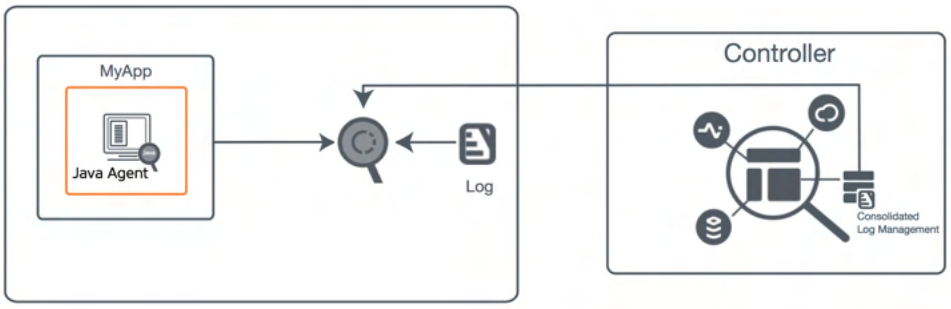
3. Save and close the file.

Configure Analytics Agent for Consolidated Log Management

If you want your Analytics Agent to work with CLM you need to configure a few properties in the Analytics Agent properties file.

Provide the Controller URL, customer name, and name of your Analytics Agent in the `analytics-agent.properties` file. The customer name is found under **View License** in the Controller.

If you are using CLM to configure your log files with source rules, you will need to provide these properties with the correct values.



The default values are the following:

```
# Format should be http://<host>:<port>
ad.controller.url=http://localhost:8090

# The customer name field from the AppDynamics license page.
http.event.name=customer1

# This is the friendly agent name that will show up in the controller when the agent registers and syncs
configuration.
ad.agent.name=analytics-agent1
```

Change Java Virtual Machine Options

If you need to change any JVM startup options, modify `<analytics-agent-home>/conf/analytics-agent.vmoptions` with a text editor.

The `vmoptions` file name is read from the `ad.jvm.options.name=analytics-agent.vmoptions` properties file. If you change the `vmoptions` file name, you will need to change the `ad.jvm.options.name` property as well.

If the Analytics Agent Windows service is installed and you need to change the properties file or the `vmoptions` file, you need to uninstall the service and reinstall it for the changes to take effect.

Configure Resource Usage Limit

To limit resource usage, you can enable the default limit on the number of job for a single Analytics Agent.

This limit can be overridden but is not recommended without a thorough understanding of the potential impact on resource usages, such as CPU usage, disk, and network I/O.

The property is `ad.max.enabled.jobs` and is found in the *<analytics-agent-home>*`/conf/analytics-agent.properties` file. By default, `ad.max.enabled.jobs` is set to 20.

Verify Analytics Agent Status

To verify that the Analytics Agent has started, look for the following entry in the App Agent log file: `Started [Analytics] collector`.

To connect to the Events Service through a proxy server, see Connect the Agent to the Events Service through a Proxy.

Troubleshooting Tips

- Ensure that the properties in `analytics-agent.properties` are properly set. See Enabled Job Files Limit.
- JRE version is >= 1.7 and the `JAVA_HOME` variable is set in the environment.
- All properties in `analytics-agent/conf/analytics-agent.vmoptions` are compatible with the JRE.

Install Agent-Side Components in Kubernetes

This page describes the deployment options for Transaction Analytics and Log Analytics instrumented with AppDynamics app server agents in Kubernetes applications. See Install Agent-Side Components.

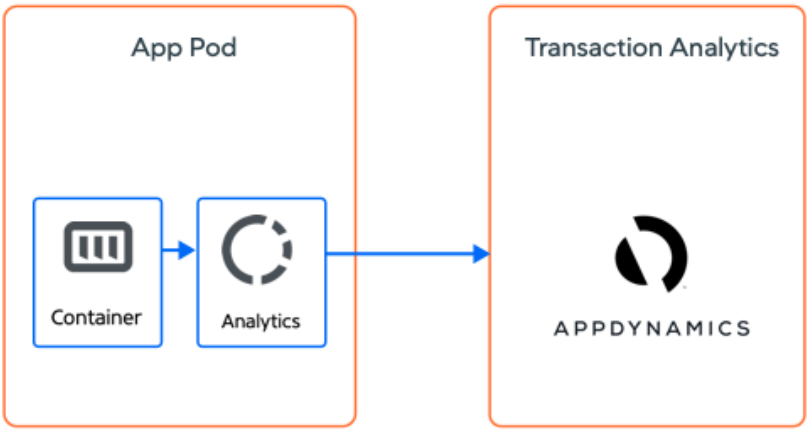
> Transaction Analytics and Log Analytics require that an Analytics Agent is deployed with an app server agent. The Java Agent >= 4.5.15 or the .NET Agent >= 20.10 does not require deploying an Analytics Agent for Transaction Analytics. See Deploy Analytics Without the Analytics Agent.

Transaction Analytics

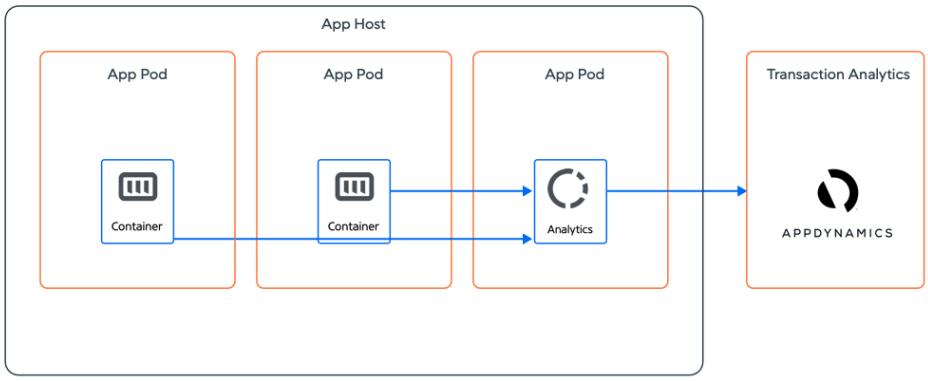
The Analytics Agent acts as a proxy between the app server agent and the Events Service. See Deploy Analytics With the Analytics Agent.

There are two deployment options for the Analytics Agent to support Transaction Analytics on a Kubernetes application.

1. A sidecar to the application container.



   In this model, an Analytics Agent container is added to each application pod and will start/stop with the application container.
2. A shared agent where a single Analytics Agent is deployed on each Kubernetes worker node. Each pod on the node will use that Analytics Agent to communicate with the Events Service.



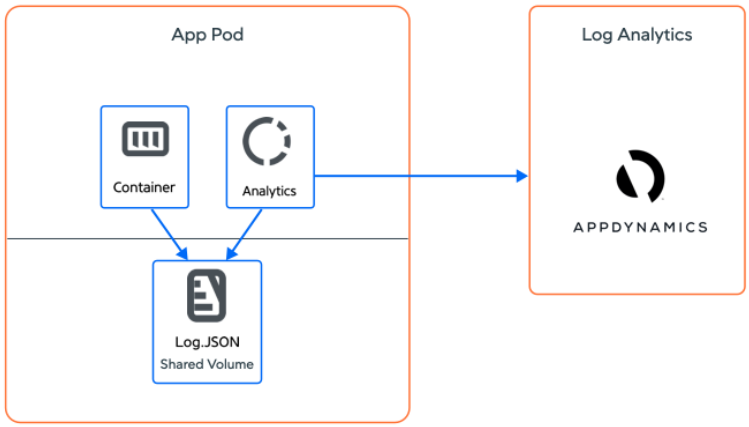
In this model, the Analytics Agent is deployed as a Daemonset.

Log Analytics

Once deployed, the Analytics Agent has access to the application's logs and can send log data to the Events Service.

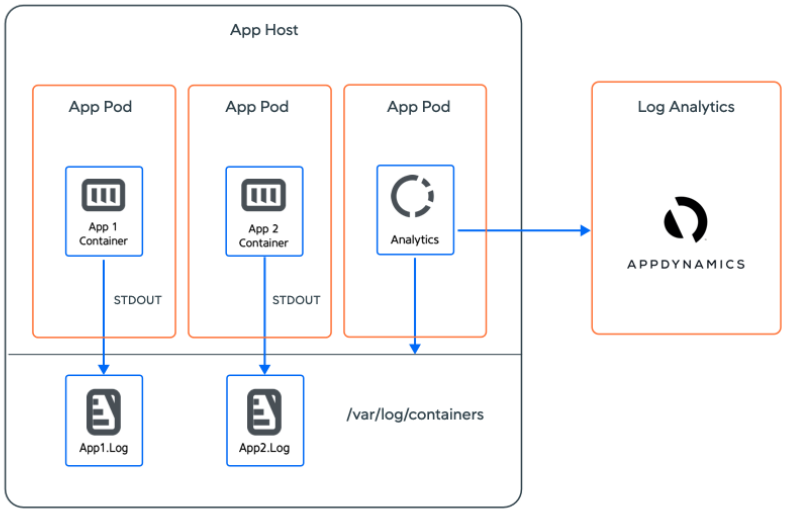
There are three deployment options for the Analytics Agent to support Log Analytics on a Kubernetes application.

1. A sidecar to the application container.



In this model, an Analytics Agent container is added to each application pod and will start/stop with the application container. The Analytics Agent and application container are configured to share a volume where the application logs are written.

2. If the application bypasses the container filesystem and emits log data to `STDOUT` and `STDERR`, the Analytics Agent can be deployed on each Kubernetes worker node. The Analytics Agent can access the log output for every application container on the worker node's file system, stored by Kubernetes under `/var/log/containers` as a unique file per container.



In this model, the Analytics Agent is deployed as a Daemonset.

> For some Kubernetes distributions such as OpenShift, the Analytics Agent will require elevated permissions to access the files under `/var/log/containers`.

3. If a syslog provider is available in the Kubernetes cluster, the Analytics Agent can be deployed to receive syslog messages with TCP transport. A single Analytics Agent instance is required per syslog provider. See Collect Log Analytics Data from Syslog Messages.

For Transaction and Log Analytics, the sidecar approach is simpler to deploy, but consumes more cluster resources because it requires one additional container per application pod. The shared agent approach adds another deployment object to manage, but can significantly reduce the overall resource consumption for a cluster.

Example Configurations to Deploy the Analytics Agent

The following deployment specs are specific examples of how to implement the deployment options explained above. In addition, see Install the .NET Agent for Linux in Containers and Install the Node.js Agent in Containers for best practices on how to set the Analytics Agent host, port, and SSL environment variables.

Transaction Analytics: Deployment Spec Using A Sidecar

The following deployment spec defines two containers, the application container `flight-services`, which uses an image instrumented with an app server agent, and the Analytics Agent container `appd-analytics-agent`, which uses the Analytics Agent from Docker Hub, `docker.io/appdynamics/analytics-agent:latest`.

The `appd-analytics-agent` container leverages a ConfigMap and Secret to configure the Events Service credentials required by the Analytics Agent, including the account access key and global account name. See Install Agent-Side Components.

As a sidecar, the Analytics Agent is available at `localhost` and uses the default port 9090. The app server agent will connect automatically and no additional configuration is required.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flight-services
spec:
  selector:
    matchLabels:
      name: flight-services
  replicas: 1
  template:
    metadata:
      labels:
        name: flight-services
    spec:
      containers:
      - name: flight-services
        image: sashaz/ad-air-nodejs-services-analytics:latest
        imagePullPolicy: IfNotPresent
        envFrom:
          - configMapRef:
              name: controller-info
        env:
        - name: APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY
          valueFrom:
            secretKeyRef:
              key: appd-key
              name: appd-secret
        - name: APPDYNAMICS_AGENT_TIER_NAME
          value: flight-services
        ports:
        - containerPort: 8080
          protocol: TCP
        restartPolicy: Always
      - name: appd-analytics-agent
        envFrom:
        - configMapRef:
            name: controller-info
        env:
        - name: APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY
          valueFrom:
            secretKeyRef:
              key: appd-key
              name: appd-secret
        - name: APPDYNAMICS_EVENTS_API_URL
          valueFrom:
            configMapKeyRef:
              key: EVENT_ENDPOINT
              name: controller-info
        - name: APPDYNAMICS_GLOBAL_ACCOUNT_NAME
          valueFrom:
            configMapKeyRef:
              key: FULL_ACCOUNT_NAME
              name: controller-info
        image: docker.io/appdynamics/analytics-agent:latest
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9090
          protocol: TCP
        resources:
          limits:
            cpu: 200m
            memory: 900M
          requests:
            cpu: 100m
            memory: 600M
...
```

The full spec can be found in the Flight Services YAML File. The `controller-info` ConfigMap can be found in the Controller Info YAML File. The command to create `appd-secret` can be found in Secret.

Transaction Analytics: Deployment Specs Using A Shared Analytics Agent

The following deployment spec is for the same `flight-services` application, but instead of using a sidecar, it references a shared Analytics Agent deployed separately as a Daemonset. The `flight-services` container sets the agent environment variables `APPDYNAMICS_ANALYTICS_HOST` and `APPDYNAMICS_ANALYTICS_PORT` to the `analytics-proxy` service for the shared Analytics Agent defined in the example below.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flight-services
spec:
  selector:
    matchLabels:
      name: flight-services
  replicas: 1
  template:
    metadata:
      labels:
        name: flight-services
    spec:
      containers:
      - name: flight-services
        image: sashaz/ad-air-nodejs-services-analytics:latest
        imagePullPolicy: IfNotPresent
        envFrom:
          - configMapRef:
              name: controller-info
        env:
        - name: APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY
          valueFrom:
            secretKeyRef:
              key: appd-key
              name: appd-secret
        - name: APPDYNAMICS_AGENT_TIER_NAME
          value: flight-services
        - name: APPDYNAMICS_ANALYTICS_HOST
          value: analytics-proxy
        - name: APPDYNAMICS_ANALYTICS_PORT
          value: "9090"
        ports:
        - containerPort: 8080
          protocol: TCP
      restartPolicy: Always
...
```

The full spec can be found in the Flight Services YAML File. Use this spec in conjunction with the following deployment spec.

In the `analytics-agent.yaml` file below, the shared Analytics Agent is deployed as a Daemonset. The file also defines a service `appd-infra-agent-service` that publishes an endpoint in the namespace where the shared Analytics Agent can be reached.

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: appd-infra-agent
spec:
  selector:
    matchLabels:
      name: appd-infra-agent
  template:
    metadata:
      labels:
        name: appd-infra-agent
    spec:
      serviceAccountName: appdynamics-infraviz
      containers:
      - name: appd-analytics-agent
        envFrom:
        - configMapRef:
```

```
            name: controller-info
        env:
        - name: APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY
          valueFrom:
            secretKeyRef:
              key: appd-key
              name: appd-secret
        - name: APPDYNAMICS_EVENTS_API_URL
          valueFrom:
            configMapKeyRef:
              key: EVENT_ENDPOINT
              name: controller-info
        - name: APPDYNAMICS_GLOBAL_ACCOUNT_NAME
          valueFrom:
            configMapKeyRef:
              key: FULL_ACCOUNT_NAME
              name: controller-info
        image: docker.io/appdynamics/analytics-agent:latest
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9090
          protocol: TCP
        resources:
          limits:
            cpu: 200m
            memory: 900M
          requests:
            cpu: 100m
            memory: 600M
        volumeMounts:
        - name: ma-log-volume
          mountPath: /opt/appdynamics/conf/logging/log4j.xml
          subPath: log4j.xml
        - mountPath: /hostroot
          name: hostroot
          readOnly: true
      restartPolicy: Always
      volumes:
      - name: ma-log-volume
        configMap:
          name: ma-log-config
      - name: hostroot
        hostPath:
          path: /
          type: Directory
---
apiVersion: v1
kind: Service
metadata:
  name: appd-infra-agent-service
spec:
  selector:
    name: appd-infra-agent
  ports:
  - name: "9090"
    port: 9090
    targetPort: 9090
status:
  loadBalancer: {}
```

The full deployment spec can be found in the Machine Agent YAML File. The `appdynamics-infraviz` service account is defined in the RBAC YAML File . The ma-log-config ConfigMap is defined in the Machine Agent Log Config File.

A best practice is to deploy the shared Analytics Agent in a dedicated namespace (typically `appdynamics`) separate from the namespaces used by applications.

```
$ kubectl -n appdynamics apply -f analytics-agent.yaml
```

To provide access to the shared Analytics Agent from an application namespace:

1. An `ExternalName` service is required to map a service name (`analytics-proxy` in the example) to the DNS name of `appd-infra-agent-service` created previously:

```
kind: Service
apiVersion: v1
metadata:
  name: analytics-proxy
spec:
  type: ExternalName
  externalName: appd-infra-agent-service.appdynamics.svc.cluster.local
  ports:
  - port: 9090
    targetPort: 9090
```

2. Create this service in each application namespace where an App Server Agent is deployed:

```
$ kubectl -n <app namespace> apply -f analytics-proxy.yaml
```

3. Note that `analytics-proxy` is the value of `APPDYNAMICS_ANALYTICS_HOST` used in the `flight-services` deployment spec.

```
- name: APPDYNAMICS_ANALYTICS_HOST
  value: analytics-proxy
```

Log Analytics: Deployment Spec Using A Side Car

The following deployment spec snippet is for a Java application that defines an application container, `client-api`, and an Analytics Agent container, `appd-analytics-agent`, that acts as a sidecar to the application container. An init container, `appd-agent-attach`, is also defined, but the related definitions are removed to simplify the example.

A shared volume, `appd-volume`, is mounted to the application container and Analytics Agent container using the mount path `/opt/appdlogs`. The Java application is configured to write its logs to this path and the Analytics Agent is configured to read the logs from this path and send them to the Events Service.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    name: client-api
  name: client-api
spec:
  selector:
    matchLabels:
      name: client-api
  template:
    metadata:
      labels:
        name: client-api
    spec:
      containers:
      - name: client-api
        envFrom:
        - configMapRef:
            name: agent-config
        env:
        - name: APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY
          valueFrom:
            secretKeyRef:
              key: appd-key
              name: appd-secret
        - name: JAVA_OPTS
          ...
        image: sashaz/java-services:v5
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 8080
          protocol: TCP
```

```
        resources: {}
        volumeMounts:
        - mountPath: /opt/appdlogs
          name: appd-volume
          ...
      - name: appd-analytics-agent
        env:
        - name: APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY
          valueFrom:
            secretKeyRef:
              key: appd-key
              name: appd-secret
        envFrom:
        - configMapRef:
            name: agent-config
        image: docker.io/appdynamics/analytics-agent:latest
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9090
          protocol: TCP
        resources:
          limits:
            cpu: 200m
            memory: 900M
          requests:
            cpu: 100m
            memory: 600M
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
        volumeMounts:
        - mountPath: /opt/appdlogs
          name: appd-volume
      dnsPolicy: ClusterFirst
      initContainers:
      - name: appd-agent-attach
      ...
      restartPolicy: Always
      schedulerName: default-scheduler
      serviceAccountName: appd-account
      volumes:
      - emptyDir: {}
        name: appd-volume
...
```

The full spec can be found in the Java App YAML File.

Log Analytics: Deployment Spec For Shared Analytics Agent (STDOUT/STDERR Support)

The following deployment spec supports the use case where application containers are emitting logs to `STDOUT` and `STDERR`, not the application container filesystem.

Since Kubernetes writes the container logs to the host under `/var/log/containers`, the Analytics Agent can read them there. The Analytics Agent is deployed as a Daemonset. A volume `varlog` is defined with access to the host path `/var/log/containers` and mounted to the Analytics Agent container, `appd-analytics-agent`. The Analytics Agent is configured to read the container-specific logs written to `/var/log/containers`. See Configure Log Analytics Using Source Rules.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: appdynamics-loganalytics
  namespace: appdynamics
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  labels:
    name: appd-analytics
  name: appd-analytics
spec:
  selector:
    matchLabels:
      name: appd-analytics
  template:
    metadata:
      labels:
        name: appd-analytics
    spec:
      nodeSelector:
        kubernetes.io/os: linux
      containers:
      - name: appd-analytics-agent
        env:
        - name: APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY
          valueFrom:
            secretKeyRef:
              key: controller-key
              name: appd-secret
        envFrom:
        - configMapRef:
            name: agent-config
        image: docker.io/appdynamics/analytics-agent:log-20.6.0
        imagePullPolicy: Always
        ports:
        - containerPort: 9090
          protocol: TCP
        - containerPort: 5144
          hostPort: 5144
          protocol: TCP
        resources:
          limits:
            cpu: 300m
            memory: 900M
          requests:
            cpu: 200m
            memory: 800M
        volumeMounts:
        - name: varlog
          mountPath: /var/log
          readOnly: true
        - name: dockerlog
          mountPath: /var/lib/docker/containers
          readOnly: true
      restartPolicy: Always
      serviceAccountName: appdynamics-loganalytics
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
      - name: dockerlog
        hostPath:
          path: /var/lib/docker/containers
```

The full spec can be found in the Log Analytics YAML File. An OpenShift example can be found in the Log Analytics OpenShift YAML File.

Upgrade Analytics Agent

This page describes how to update the Analytics Agent standalone binary and includes some steps that are relevant only if you are collecting Log Analytics data.

If you are running the Analytics Agent through the Standalone Machine Agent, see Upgrade the Machine Agent for instructions.

> You should maintain a copy of the controller.xml file in addition to the `analytics-agent.properties`, job folder, and watermark folder.
>
> Preserving the watermark file and the job files is only necessary when you are collecting log data for analytics. The watermark file preserves information about the number of bytes read from the log sources by the Analytics Agent and is relevant only when Log Analytics is enabled. Job files contain the configuration for <= 4.3 log sources.

File Locations

After unzipping the Analytics Agent standalone binary, the file locations are:

- property file: *<analytics-agent-home>*`/conf/analytics-agent.properties`
- job files: *<analytics-agent-home>*`/conf/job`
- watermark file: *<analytics-agent-home>*`/conf/watermark`

Running Analytics Agent via the Standalone Machine Agent, the file locations are:

- properties file: *<machine-agent-home>*`/monitors/analytics-agent/conf/analytics-agent.properties`
- job file: *<machine-agent-home>*`/monitors/analytics-agent/conf/job`
- watermark file: *<machine-agent-home>*`/monitors/analytics-agent/conf/watermark`

Upgrade Procedure for Analytics Agent (Standalone Binary)

Use these steps to upgrade your Analytics Agent (standalone binary) to 4.5.

1. Stop the Analytics Agent.
2. Download the 4.5 Analytics Agent binary from the download site.
3. Extract the new zip. Be sure to extract into a new empty directory or confirm that the old directories are empty before extracting the new version. Otherwise, you may encounter errors starting the Analytics Agent.
4. (Windows only) If the Analytics Agent is installed as a Windows Service, you need to manually uninstall the old service before installing the 4.5 version.
5. Save a copy of your old analytics-agent.properties so you can copy values to the new analytics-agent.properties file.
6. Modify the new `analytics-agent.properties` file with in accordance with the steps here: Install Agent-Side Components, referring to your previous properties as needed.
7. If you have been collecting log analytics data:
    a. Copy the old watermark file to the corresponding location in the new *<analytics-agent-home>* directory structure.
    b. Copy your pre-existing job files to the corresponding location in the new *<analytics-agent-home>* directory structure.
8. Start the new Analytics Agent.
   You can observe that the agent starts the enabled job file, and starts tailing the logs from the last read position in the watermark file. Usually, a log message is displayed stating the start of the tailing from last read location. It matches with the watermark file. If at time printing log message gets out of sync, it can be manually validated through the UI that all logs have been tailed, and no duplicate logs have been tailed.

To move from job files to the new Consolidated Log Management source rules, see Migrate Log Analytics Job Files to Source Rules.

Analytics and Data Security

Related pages:

- Manage Field Visibility
- Manage API Keys
- Create and Manage Custom Roles

Application Analytics provides role-based access control (RBAC) to enable you to control and protect your data. The Analytics permissions enable you to provide granular, controlled access to features and analytics data in your environment. This topic outlines the available permissions.

Roles and Permissions

A predefined role called Analytics Administrator is provided with preset permissions for all Analytics-specific permissions.

You can clone predefined roles as a starting point for creating your own customized roles, but you should not assume the cloned roles have all of the permissions of the predefined role. In some cases, there may be hidden permissions, so you should add or remove permissions as needed for your customized role to ensure that you get the RBAC result you need.

To create roles and assign users to roles for Analytics, users need both the **Analytics Administrator** and the **Account Owner** role.

See Transaction Analytics Permissions.

Default Application

To ensure that your users have access to the full Analytics functionality they need, be sure to give them the view permission on the Default Application. This permission is given to all predefined admin roles and to the read-only role, **Applications & Dashboards Viewer**. If you create new roles for analytics, you need to grant this permission in addition to specific analytics permissions.

When you create a new role, this permission is not automatically given to that role. You can add the read-only role to new users. Alternatively, navigate to **A dministration** > **Applications** and select the **View** permission in the Default Application.

Analytics Administration UI Tabs

Access the Analytics tab in the Controller Administration UI when creating new roles to see the available permissions.

General Permissions

This section describes permissions that control access to analytics features:

- Manage Fields
- Manage APIs
- Manage Metrics
- Configure Log Analytics with Source Rules
- Manage Business Journeys
- Manage Experience Levels

Search Permissions

All the saved searches created in Analytics appear in this section. The admin role can assign View, Edit, and Delete permissions on a search by search level. The admin can create and save specific searches needed by various roles in your organization. In addition to enabling the permissions, View, Edit, and Delete, for a specific saved search, the admin must also enable access to the related application or log source type for the search. Otherwise, the data access level won't be granted to the role.

To create a new saved search, the user must have the **Can Create a Search** permission.

View permission: Assigning only the View permission for a saved search means users can not edit or delete the search.

Edit permission: Assigning Edit permission for a saved search means users can modify the filter criteria and save back to the same search.

Delete permission: Assigning Delete permission for a saved search means users can delete the search.

Event Type Permissions

Transactions Permissions: Enables the admin user to assign permissions for viewing application transaction data. Permissions can be granted to view all transaction data for the account or on an application-by-application basis. When creating new roles, remember that granting permissions to view transaction analytics data does not automatically grant permissions to see all application data associated with a specific transaction analytics record.

You need to grant at least read-only permissions to the application to enable the user to see associated transaction snapshot data such as flow maps.

Log Permissions: Enables the admin user to assign permissions for viewing log data. Permissions can be granted to view log data for all source types configured for the account or on a source by source basis.

Browser Requests Permissions: Enables the admin user to assign permissions for viewing browser request data. Permissions can be granted to view data for all applications for the account or for specific applications.

Mobile Requests Permissions: Enables the admin user to assign permissions for viewing mobile requests and crash report data. Permissions can be granted to view data for all applications for the account or for specific applications.

Custom Analytics Events: Enables the admin user to assign permissions for querying custom analytics events data. Permissions can be granted to view data for all custom analytics events or for specific applications.

Synthetic Permissions: Enables the admin user to assign permissions for querying Synthetic events data. Permissions can be granted to view data for all applications for the account or for specific applications.

Manage API Keys

Related pages:

- Create and Manage Custom Roles

This page describes how to manage REST API Keys.

The Analytics Events REST API provides three independent endpoints for these responsibilities:

- Schema Management
- Publish Events
- Query Events

API Keys provide a secure authentication mechanism for a caller to prove identity when using these public REST APIs.

Because call sites can vary across your infrastructure, departments, and geographic regions, these credentials could be widely distributed and hard to control. Publishing or querying events from different call sites, therefore, require fine-grained control over the keys and the operations allowed for a particular key. You may also need to revoke these keys on demand if they are found to be compromised.

For these reasons, managing access for the Analytics Events API uses API keys. Your organization can create multiple API keys, manage and distribute them based on your own internal policies. For instance, each department or geographic region may be assigned its own API key for distribution and control management. If an API key is compromised, it can be deleted and a new key created without other undesirable side effects.

API keys are only valid for calling the public Analytics Events REST API and can't be used to access the AppDynamics Controller or data in any other way. See Analytics Events API.

Permissions

API keys are used only for Analytics Events REST APIs. When you create an API key, you can specify read and write permissions for each API key.

Users with the **Manage API** permission can do the following actions:

- Add—Creates an API key
- Disable—Deactivates the calls using that key temporarily, but does not remove the key
- Enable—Re-enables a deactivated key
- Delete—Removes the key permanently

It may take up to 15 minutes for a key that has been deleted or deactivated to be detected and all operations using that key to be rejected.

When you create an API key, you see permissions for the various event types (logs, transactions, browser, and mobile) in our platform. By using these permissions you can limit or grant access to specific event types for each specific API key.

Examples:

Select **Publish all Custom Events** when you create their API key to generate an API key for your partners allowing them to only publish custom events.

Under **Log Permissions**, select **apache** as the **source type** to generate an API key to allow various users to only query Apache log data.

Once an API Key is created, you cannot change the permissions.

You can grant permissions for API Keys as follows:
Custom Analytics Events

- Manage Schema— Enables creation of schemas using the Analytics Events Schema Management APIs.
- Query Custom Events—Enables you to query all Analytics event types (with the appropriate permissions).
- Publish Custom Events—Enables you to publish Analytics custom events using the Analytics Events Publish APIs.

Transactions—Query all transactions or specific applications.

Logs—Query all logs or specific source types.

Browser Requests—Query all browser requests or specific applications.

Mobile Requests—Query all mobile requests or specific applications.

Synthetic Requests Permissions—Query all synthetic requests data or specific applications.

Connected Devices Permissions—Query all connected device requests data or specific applications.

If you have deployed EUM such that you are using an on-premises Events Service for transaction and log analytics data, and the SaaS Events Service for your EUM data, you can not query the browser or mobile request data using the Analytics API.

Create API Keys

1. Navigate to **Analytics > Configuration** > **API Keys**. You can view existing keys and access actions to manage the keys.
2. Click **+Add** to view the configuration panel.
3. Add a **Name** and **Description**.

   Do not click **Create** until you finish selecting all the necessary permissions for your use case, including any necessary analytics data permissions. You cannot change the permissions once create the key. You can only edit the description and whether the key is active or inactive.

4. Expand each permission section to select the permissions for this key.
5. Click **Create**.
6. Copy and save the key.
7. Select the checkbox indicating you have copied the key and click **Done**.

   You cannot retrieve the key once you dismiss this dialog.

Manage Field Visibility

Related pages:

- Create and Manage Custom Roles
- Analytics and Data Security

This page describes how to manage the visibility of fields in your analytics data. This capability enables you to prevent fields from being viewed by others in your organization.

You can set the visibility of the fields in most of the analytics event types with the exception of custom events. Hiding fields in custom events is not supported.

Permissions

The Manage Fields permission is required to manage field visibility. If a role has the Manage Fields permission, it means all the users with that role can manage all the fields they have access to. They can unhide the fields or see which fields have been hidden. For instance, if you have Manage Fields permission and View permission for the business transactions in the Ecommerce App, then you can only see and manage fields for that application.

Manage Fields

1. From the Analytics Search panel, click  next to each component grouping to open the Manage Fields panel.
2. Use the Manage Fields panel to show or hide fields. The list of fields varies based on the event type selected for the search.

Delete Extracted Fields

For >= 4.3, you may have two types of extracted fields, fields created in 4.2 where the Controller version has been upgraded to a higher version and fields extracted from logs with log source rules.

For fields created in 4.2, you can delete the fields:

1. Hover over the field and click on the view icon that appears.
2. Click on **Delete** in the popup.

For fields being extracted from logs using log source rules, you can delete the fields by editing the source rules. See Field Extraction for Source Rules.

Manage Business Journeys and Experience Levels

This page describes how to manage the visibility of business journeys and experience level management (XLM) with role-based access control (RBAC). Business journey and XLMRBAC is required to restrict access to sensitive information among your organization. If a user attempts to access restricted features, a message appears warning the user that access is denied.

Only the analytics admin can grant permissions to business journeys and XLM, as well as with their respective searches, event types, and sources. The admin authorizes access on an individual basis by assigning each user to the applicable role.

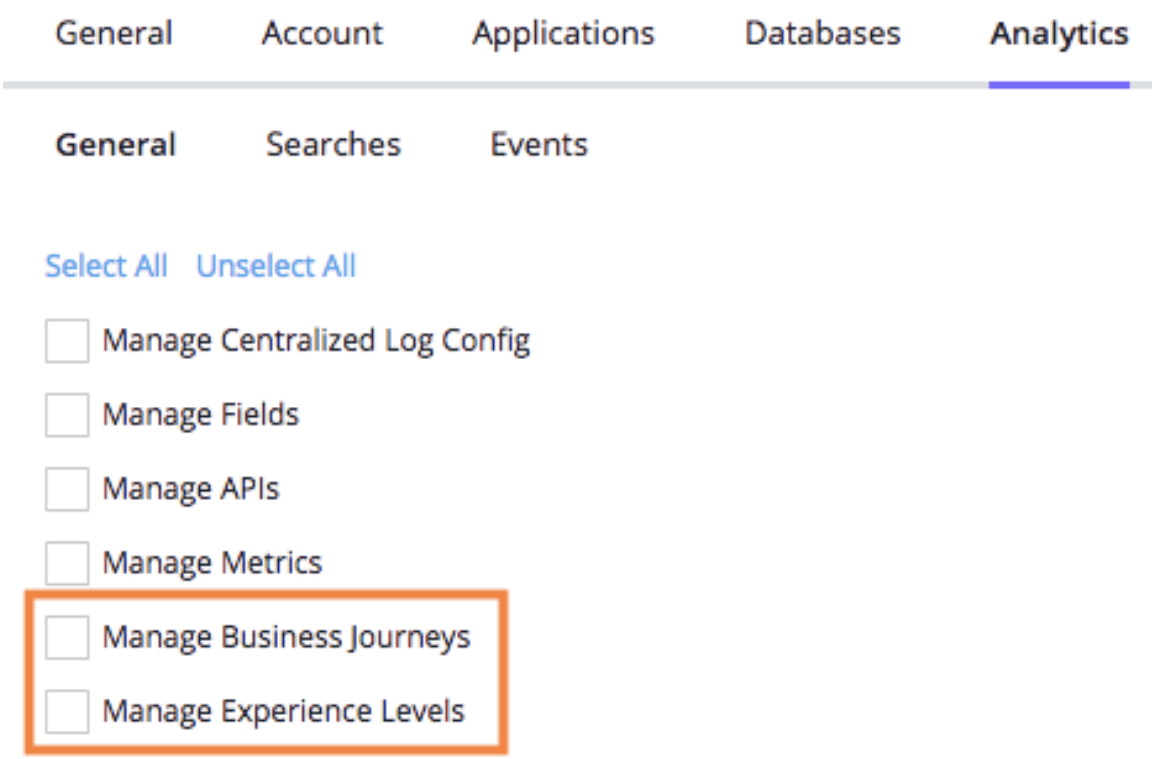
Creating Business Journey and XLM Roles

In the Controller UI, navigate to **Settings** > **Administration** > **Roles**.

The analytics admin can view and edit existing roles in the Analytics tab, which contains three sections: General, Searches, and Events. Select **+Create** to make a new role.

General Permissions

In the General tab, the analytics admin authorizes access to business journeys and/or XLM. This is required for the user to perform any actions on either business journeys or XLM. Check **Manage Business Journeys** and/or **Manage Experience Levels**:



With general permission, the user has feature level access to business journeys and/or XLM. However, the user cannot view the definition of existing events or saved searches, or create new events, with only general permission. See the below sections to enable these permissions.

> A user can access business journey data through searches without general permission to business journeys, as long as the user's role permits access to the underlying event type(s) used in business journey definition.
>
> In this way, you can restrict access to define business journeys while also allowing a user to query the data. Ensure the user has all necessary access to event type(s) in the Events tab and leave **Manage Business Journeys** unchecked.

Search Permissions

There are two search types in Analytics, drag and drop and query language. The analytics admin grants access to both types in the Searches tab.

Choose **Can Create a Search** to access both search types. Choose **+Add** to access specific saved searches.

Create search permission is required in order for a user to perform these operations:

- Save a search
- Create a metric
- Create a visual widget

Search access is dependent on event type access. A user has access to create and save searches only for the event type(s) and source(s) assigned in the user's role.

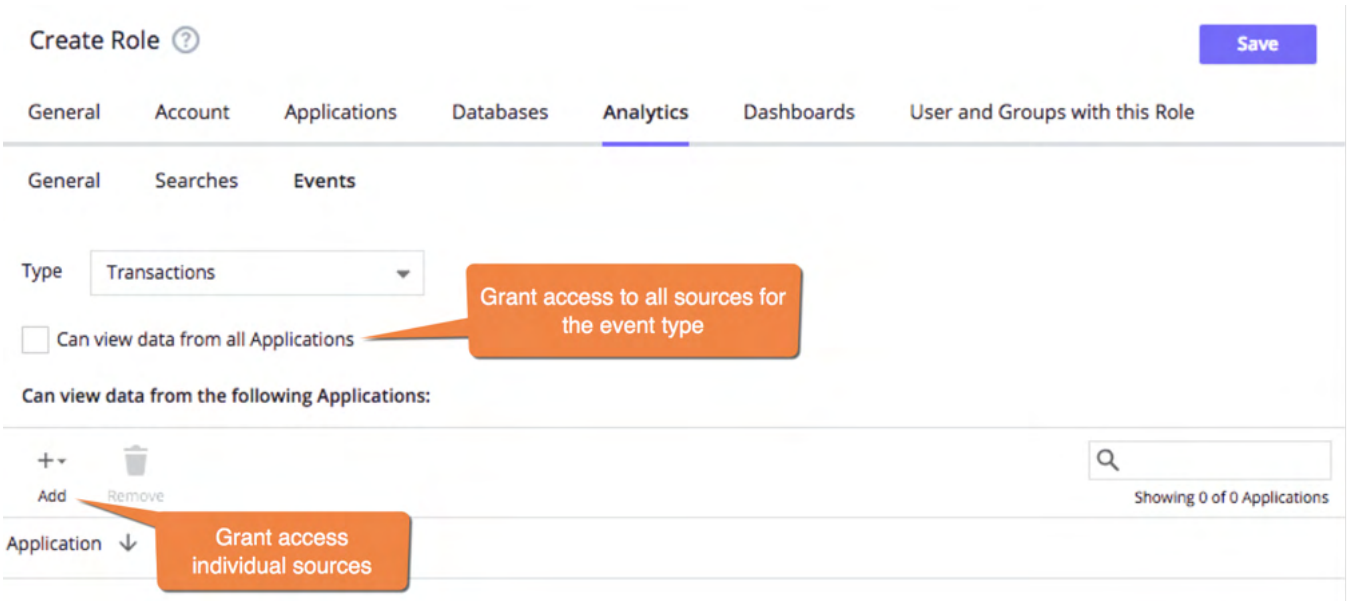
Event Type and Source Permissions

The section describes how to assign event type and source access. Granting permission to business journeys and/or XLM does not provide a user blanket access to all reports. Users have access only to the exact event type(s) and source(s) granted by the analytics admin. For example, transactions are an event type, and their source is applications. Therefore, if a user requires access to particular applications, the analytics admin must select transactions as well as each necessary application for the user.

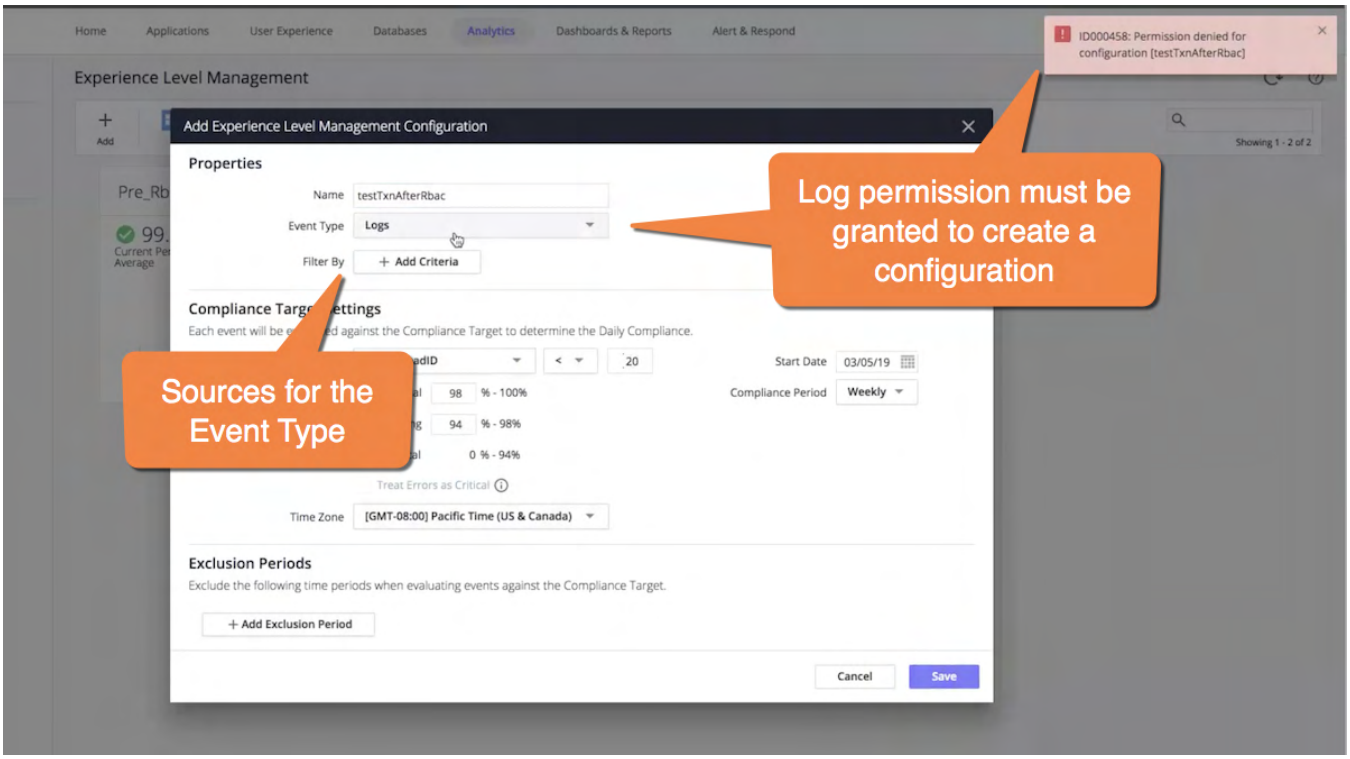
This table lists the available event types and their corresponding sources:

Event Type	Source
Transaction	Application
Log	Source Type
Browser Records	App Key
Mobile Records and Sessions	App Key
Synthetic Session	App Key
Connected Devices	App Key
Custom Analytics Events	Particular user-defined event type

You can specify exact access in the Events tab. Select the event type, then specify access for the relevant source. Check "**Can View Data from all** `[source]`**"** to grant blanket access to each source. To select individual sources, click **+Add**.



With the appropriate event type and source permissions, the user can access existing reports as well as create new reports. For example, a user with permissions to Transactions and all applications can now create a new configuration for any application. However, if this user attempts to create a configuration for Log events, the configuration does not save and an error message appears in the UI:



For XLM configurations, the **Filter By** field indicates the source(s) of the given **Event Type**. Users with limited permission to sources need to add their permitted source(s) as filter criteria. If you do not specify filter criteria, you must have permission to all sources for the event type to create the configuration.

Dashboard Access

Grant permission to dashboards in the Dashboard tab. In this tab, the admin specifies if a user can create dashboards and time permissions, as well as custom permissions.

If an Analytics dashboard contains data from business journeys and/or XLM, access to the dashboard depends on the above permissions. For example, a role that allows its users to create a dashboard, but contains no permission to business journeys, does not permit the user to access a business journey-related dashboard. Ensure that roles have the necessary permission to access Analytics data and general dashboard access.

View Analytics Agent Health

This page describes how to access the health list of Analytics Agents.

You can view Analytics Agents connected to your Controller in **Settings > AppDynamics Agent > Analytics Agents**. You can also view agent status and health information for the Log Analytics and Transaction Analytics pipelines.

When a new Analytics Agent starts and connects to the Controller, it registers with the Controller. Once the agent is registered, the agent requests configuration information every five minutes. As part of this request, the Analytics Agent reports health status information to the Controller. A healthy pipeline means analytics data is flowing from the Analytics Agent to the Events Service successfully.

Condition	Transaction Pipeline Health	Log Pipeline Health
Analytics Agent does not contact the Controller for more than five minutes.	Not connected	Not connected
No business transaction analytics data is flowing and there are no errors in the connection.	Connection is healthy, no data flowing	Not Applicable
Business transaction analytics data is flowing and the Agent is publishing to the Events Service successfully with no errors.	Healthy	Not Applicable
Business transaction analytics data is flowing, but the Agent is experiencing errors while publishing transactions to the Events Service.	Unhealthy	Not Applicable
No log analytics data is flowing and there are no errors in the connection.	Not Applicable	Connection is healthy, no data flowing
Log analytics data is flowing and the Agent is publishing to the Events Service successfully with no errors.	Not Applicable	Healthy
Log analytics data is flowing, but the Agent is experiencing errors while publishing transactions to the Events Service.	Not Applicable	Unhealthy

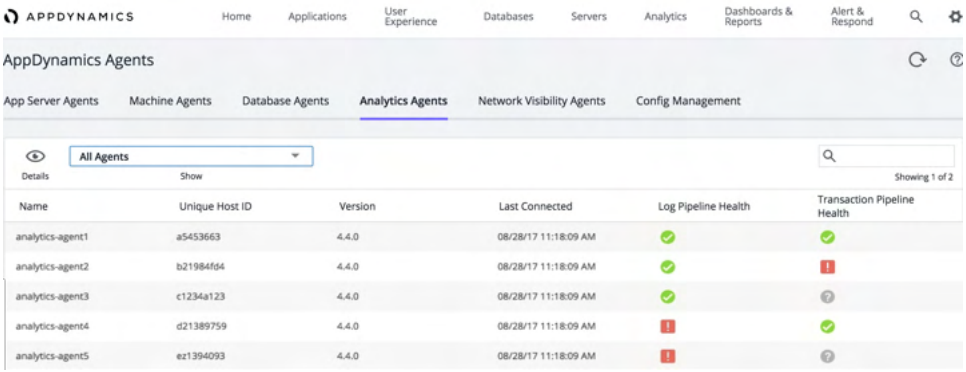
Permissions

To view the Analytics Agents list, you must belong to a role with the Account level **Administer users, groups, roles, authentication, etc. View Lic**
AppDynamics Agents. Use Agent Download Wizard permission. The default Account Owner and Analytics Administrator roles have this permissi

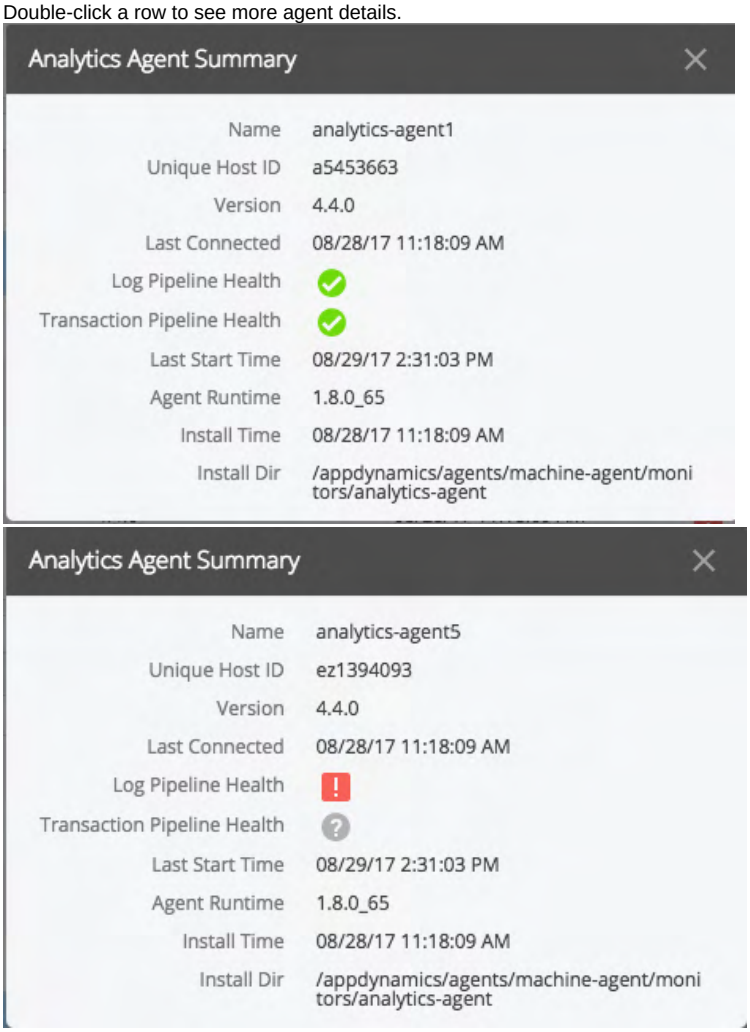
Access Analytics Agent Health List

To access Analytics Agents health view, navigate to **Settings > AppDynamics Agents > Analytics Agents**.

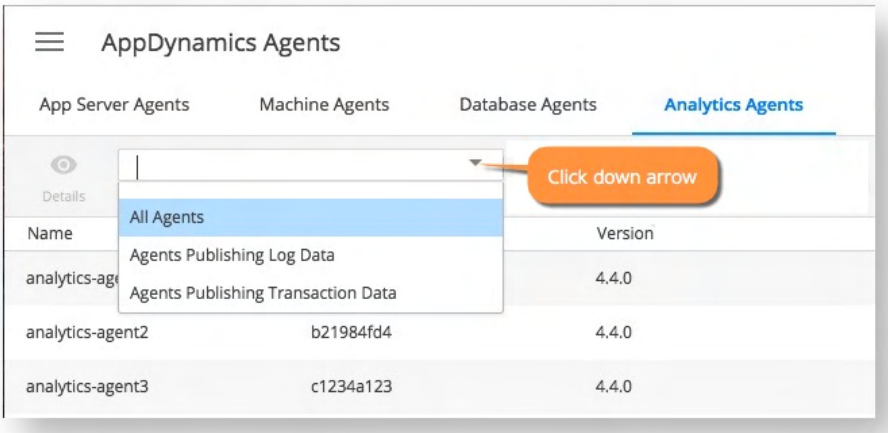
1. View information for connected Analytics Agents.



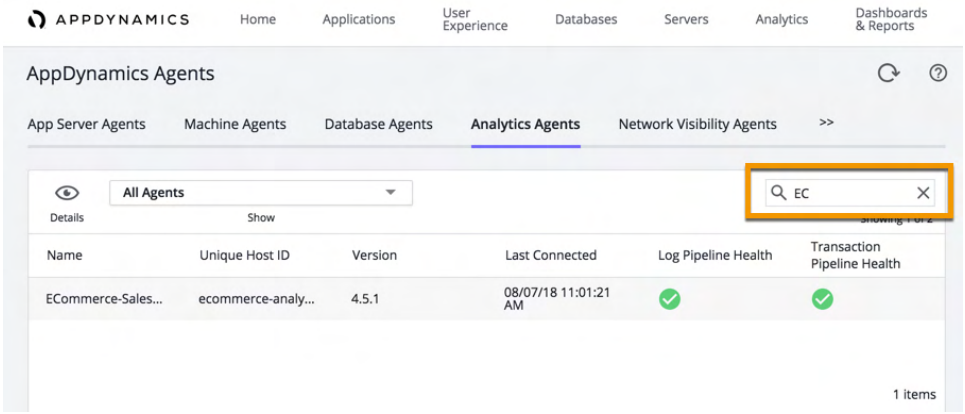
2. Double-click a row to see more agent details.



3. View Analytics Agents by event type (log data and business transaction data).



4. Perform a search on the list of registered Analytics Agents.



Analytics Licenses

Related Pages:

- License Management
- License Entitlements and Restrictions

This page describes how to interpret the details of your Analytics license. The information is split between Infrastructure-based Licensing and Agent-based Licensing. To see a complete list of licenses and packages, see License Entitlements and Restrictions.

Licensing for Analytics

Analytics licenses differ depending on what licenses you purchase. Under the Infrastructure-based Licensing model, Analytics licenses are part of monitoring packages, except for Log Analytics, which is a seperate license. Under the Agent-based model, each Analytics agent is licensed separately. Learn more about Licensing.

Remote Analytics Agent Sizing

Related pages:

- Deployment Options and Scenarios
- Install Agent-Side Components
- Event Service Sizing and Capacity

This page describes how to estimate the hardware requirements for a remote Analytics Agent deployment where a single Analytics Agent is aggregating transaction events from multiple APM Application Agents. This page describes transaction analytics only.

You should consider these two questions when sending analytics data to a remote Analytics Agent:

- How many APM agents can report to one remote Analytics Agent?
- What are the machine requirements for hosting the remote Analytics Agent?

Do not extract the numbers on this page to size for Log Analytics because you must install the Analytics Agent on the local machine to capture Log Analytics.

Analytics Agent Sizing Based on Event Volume

Based on our testing, the volume of events being sent to the Analytics Agent is the limiting factor in determining how many APM agents can report to one remote Analytics Agent.

The tests were conducted on virtual hardware and programmatically generated workload. Real-world workloads may vary. To best estimate your hardware sizing requirements, carefully consider the traffic patterns in your application and test in a test environment that closely resembles your production application and user activity.

Calculate Analytics Event Volume

One business transaction can traverse many tiers. In each tier, one business transaction traverses one node. One node produces one request per business transaction when the transaction is synchronous. For async transactions, multiple events may be generated by a node for a single request. One request equals one analytics event. To calculate how many events a business transaction generates, you need to count the number of tiers/nodes that are sending data into the Analytics Agent.

You can estimate the number of events using the following formula:

> One business transaction generates events at a rate = **calls per minute** times the **number of tiers** reporting analytics data for the business transaction.

In simple terms: #events for one business transaction = calls per minute times # of tiers.

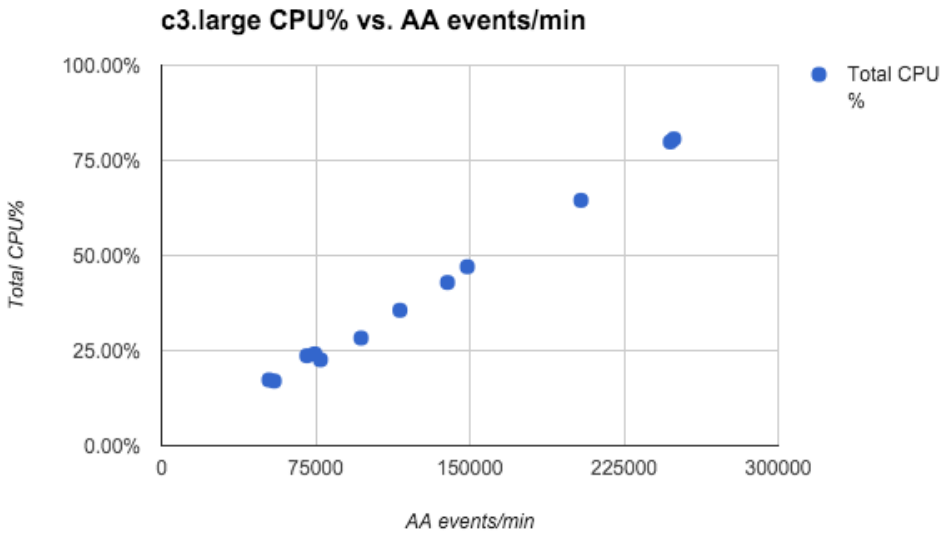
Characteristics of the Amazon EC2 Instance Types

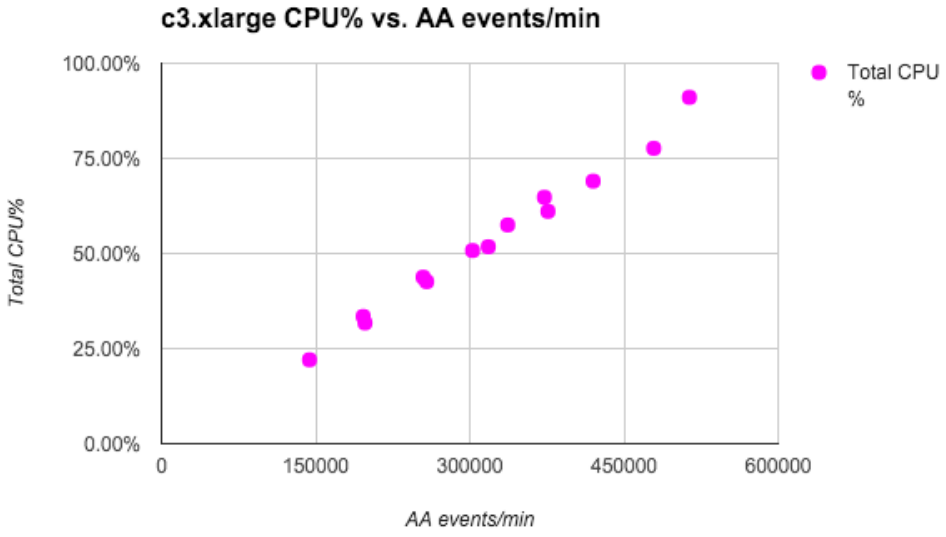
For complete information on Amazon EC2 instance types, see https://aws.amazon.com/ec2/instance-types/.

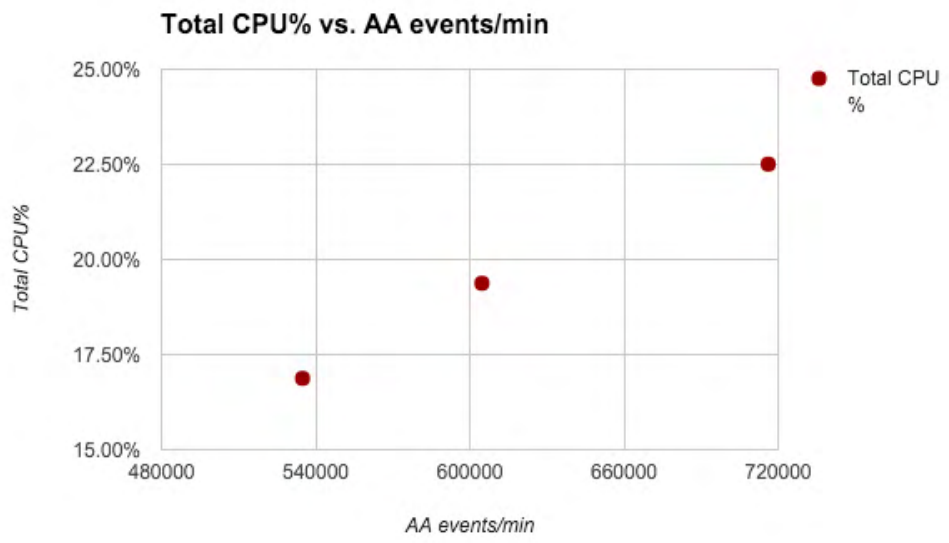
The testing was performed using c3.large, c3.xlarge, and c4.4xlarge.

Model	vCPU	Mem (GiB)	SSD Storage (GB)
c3.large	2	3.75	2 x 16
c3.xlarge	4	7.5	2 x 40
c4.4xlarge	16	30	EBS-Only

Test Results


c3.large CPU% vs. AA events/min
100.00%
75.00%
50.00%
25.00%
0.00%
Total CPU%
0
75000
150000
225000
300000
AA events/min
Total CPU %


c3.xlarge CPU% vs. AA events/min
100.00%
75.00%
50.00%
25.00%
0.00%
Total CPU%
0
150000
300000
450000
600000
AA events/min
Total CPU %



Raw Data

Analytics Agent Host Machine	Analytics Agent events/min	Total CPU%	JVM Heap (Mb)
c3.large	52313	17%	468
c3.large	54831	17%	470
c3.large	70746	24%	475
c3.large	74541	24%	477
c3.large	77344	23%	487
c3.large	97074	28%	512
c3.large	115999	36%	519
c3.large	139143	43%	526
c3.large	148782	47%	587
c3.large	204073	65%	527
c3.large	247543	80%	624
c3.large	249261	81%	637
c3.xlarge	196288	33%	518
c3.xlarge	254586	44%	727
c3.xlarge	302689	51%	497
c3.xlarge	336879	58%	913
c3.xlarge	372515	65%	1024
c3.xlarge	513598	91%	922
c3.xlarge	478954	78%	922
c3.xlarge	420000	69%	979
c3.xlarge	376034	61%	1024
c3.xlarge	318000	52%	1024
c3.xlarge	258000	43%	1024
c3.xlarge	198000	32%	1024
c3.xlarge	144000	22%	1024
c4.4xlarge	534900	17%	552
c4.4xlarge	604725	19%	841

c4.4xlarge	716141	23%	1024

Enable SSL for the Analytics Agent

Related pages:

- Controller SSL and Certificates

This page describes how to configure the AppDynamics Analytics Agent to connect to your app server agents using SSL.

1. Run the `keytool` command to create a new key pair for the Analytics Agent in the keystore. Refer to the Oracle documentation for more details on using keytool. The following command creates a keystore if it doesn't exist and generates the public/private key pair:

   ```
   keytool -genkeypair -alias analytics-agent -keystore aa-keystore.jks -validity 1825
   ```

   Follow the on-screen instructions to configure the certificate. This generates a self-signed certificate in the keystore. The next step generates a signing request for the certificate. Note the following:

   a. For the first and last name, enter the domain name where the Analytics Agent is running.
      The domain name used in the `appdynamics.analytics.agent.url` property must match the Common Name (CN) of the certificate used by the Analytics Agent. For example, if the URL was `https://localhost:9090/v2/sinks/bt` then the CN of the certificate should be `localhost`. If the names don't match then the client marks the certificate as invalid and the HTTPS connection is not established".
   b. Enter a secure password for the key.
      This command creates a key pair with a validity of 1825 days (5 years). Replace 1825 with the validity period appropriate for your environment.
2. Generate a certificate signing request for the certificate you created as follows:

   ```
   keytool -certreq -alias analytics-agent -keystore aa-keystore.jks -file AppDynamics.csr
   ```

3. Submit the certificate signing request file generated by the command (`AppDynamics.csr` in the example command) to your Certificate Authority (CA) of choice.
4. When it's ready, the CA returns the signed certificate and any root and intermediary certificates required for the trust chain. The response from the CA should include any special instructions for importing the certificate if needed. If the CA supplies the certificate in text format, copy and paste the text into a text file.
5. Import the signed certificate:

   ```
   keytool -import -trustcacerts -alias analytics-agent -file mycert.cer -keystore aa-keystore.jks
   ```

   This command assumes the certificate is located in a file named `mycert.cer`.
6. If you see the error "Failed to establish chain from reply", install the issuing CA's root and any intermediate certificates into the keystore. The root CA chain establishes the validity of the CA signature on your certificate. Although most common root CA chains are included in the bundled JVM's trust store, you may need to import additional root certificates, such as certificates belonging to a private CA. To do so:

   ```
   keytool -import -alias [Any_alias] -file <path_to_root_or_intermediate_cert> -keystore <controller_home>
   /appserver/glassfish/domains/domain1/config/aa-keystore.jks
   ```

7. When done importing the certificate chain, try importing the signed certificate again.
8. Update the following properties in the analytics-agent.properties file:
   a. `ad.dw.https.enabled`=true to enable the HTTPS connector on the Analytics Agent. The HTTPS connection is exposed on the port defined by the `ad.dw.http.port` property.
   b. `ad.dw.https.keyStorePath`= absolute path to the Java keystore that contains the Analytics Agent public and private key.
   c. `ad.dw.https.keyStorePassword`= Java keystore password.
   d. `ad.dw.https.trustStorePath`=absolute path to the truststore that establishes the chain of trust for the Analytics Agent public key certificate.
   e. `ad.dw.https.trustStorePassword`=the truststore password.
   f. `ad.dw.https.certAlias`= alias of the public key certificate stored in the Java Key Store.
9. Start the Analytics Agent.
   The HTTPS connection should now be exposed.
10. Confirm that the app server agent is configured to trust the Analytics Agent certificate. See "*Enable SSL between the Java Agent and the Analytics Agent*" on the Enable SSL for the Java Agent page.

Analytics Agent Logging

Related pages:

- Troubleshoot Analytics Issues

By default, the AppDynamics Analytics Agent writes log files to the <*analytics_agent_home*>/logs directory. See Agent Log Files for how the logs are organized into sets that roll over.

Log File Sizes

Each log can grow to 64 MB in size before being compressed and renamed. A maximum of ten compressed files can be generated.

The default pattern for agent log naming is <*filename*>-<*date_compressed*>.log.gz as shown in this table:

Log File Name	Definition
analytics-agent.log	Current log
analytics-agent-2018-12-11.log.gz	Previous log
analytics-agent-2018-12-10.log.gz.log	Second previous log

Logging Level for the Analytics Agent

The default logging level for most log files is INFO. Higher logging levels consume more disk space.

You can control the logging level for the Analytics Agent by changing the value of the ad.dw.log.level parameter in the analytics-agent.properties file.

For example, to set the log level to DEBUG:

```
ad.dw.log.level=DEBUG
```

Deploy Analytics Without the Analytics Agent

Deployment Support

The Analytics Agent is not required to capture Transaction Analytics data. This page describes the requirements to use the "agentless" configuration of Analytics.

In this context, "agentless" refers to the configuration of Transaction Analytics without an Analytics Agent. An app server agent, such as Java or .NET, is still required to use this feature.

Agentless Analytics

To capture Transaction Analytics for Java or .NET Windows applications, you can configure Transaction Analytics without a dedicated Analytics Agent.

In this deployment model, Analytics retrieves data directly from your app agent. The data is reported to the Events Service and then the Controller. No separate Analytics Agent is required.

> If you upgrade the Java Agent to >= 4.5.15, and use Transaction Analytics, Agentless Analytics is automatically enabled. If you upgrade the .NET Agent to >= 20.10, and use Transaction Analytics, Agentless Analytics is automatically disabled; you can enable this feature in the Controller or with a node property. See Enable Agentless Analytics.
>
> You may notice unusual behavior (such as gaps in Analytics data) related to two known issues:
>
> - High CPU usage (in the Java Agent version 4.5.19 and below)
> - Blocked firewall requests
>
> See Troubleshoot Analytics Issues.

Requirements

Make sure you meet the following minimum requirements:

- Sufficient Transaction Analytics Licenses
- Java Agent >= 4.5.15
- .NET Agent >= 20.10
- Controller >= 4.5.16 (SaaS or on-premises)

To use Agentless Analytics, see this compatibility matrix for your agent deployment requirements.

Connect App Agents to the Events Service

See the following sections to ensure that your deployment can connect to the Events Service:

- Identify Internal Events Service Endpoints
- Connection Ports
- Enable SSL
- Proxy Configuration

Identify Internal Events Service Endpoints

The Controller uses the connection URL configured in `Admin.jsp` to the Events Service. Agentless Analytics also uses this URL. See Connect to the Events Service.

Connection Ports

If you have network policies that may prevent this connection, such as firewalls, you need to configure the policies to connect with the Events Service. See Troubleshoot Analytics Issues.

Enable SSL

If you use a custom trust store, see Enable SSL for the Java Agent or Enable SSL for the .NET Agent.

Proxy Configuration

Enable Agentless Analytics

You can enable Agentless Transaction Analytics at the Controller, node/tier, and agent level.

Disable Agentless Analytics

You can disable Agentless Transaction Analytics at the Controller, node/tier, and agent level.

> **Existing Analytics Agents**
>
> Once you upgrade to the minimum Controller and agent versions, existing Analytics Agents stops reporting data. Agentless Analytics does not duplicate data from the Analytics Agent. AppDynamics recommends when you see **Transaction Analytics** in the Controller, you can uninstall the Analytics Agent unless you are using Log Analytics.

Configure Analytics

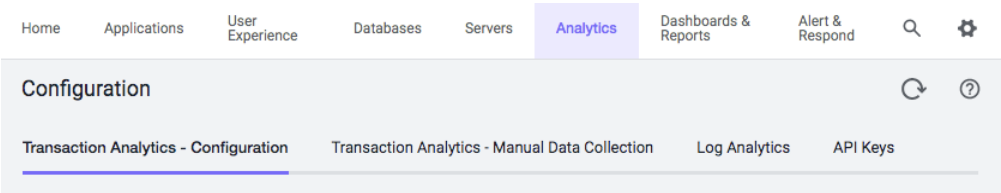
To generate information for Analytics, such as data collectors and log analytics, you need to configure the system to collect the applicable data.

Access Analytics in the Controller UI

In the Controller UI, navigate to **Analytics** > **Configuration**.

The Analytics configuration options include:

- **Transaction Analytics - Configuration**: Enable Analytics for specific applications and business transactions.
- **Transaction Analytics - Manual Data Collection**: Configure data collectors. Analytics supports three types of data collectors: method invocation data collector (MIDC), HTTP, and SQL.
- **Log Analytics**: Configure source rules to collect data for log analytics.
- **API Keys**: Create API authentication keys for users of the Analytics Events API.



End User Monitoring Data

Analytics monitors End User Monitoring (EUM) products. For SaaS deployments, Browser Request Analytics and Mobile Request Analytics, alternate views of EUM data, do not require configuration. If you are using an on-premises Events Service and an on-premises EUM Server, you need to configure the EUM Server to send data to the Events Service. See EUM Server Deployment.

Collect Transaction Analytics Data

Works with:

This page describes how to configure Transaction Analytics for Java and .NET applications. See Configure Transaction Analytics for Node.js and PHP Applications.

Before You Begin

If you have Java Agent >= 4.5.15, .NET Agent >= 20.10, and Controller >= 4.5.16, you do not need a local Analytics Agent to configure Transaction Analytics. See Deploy Analytics Without the Analytics Agent.

All other app agents and Controller versions require an Analytics Agent. See the appropriate documentation to ensure you have installed and configured the required components.

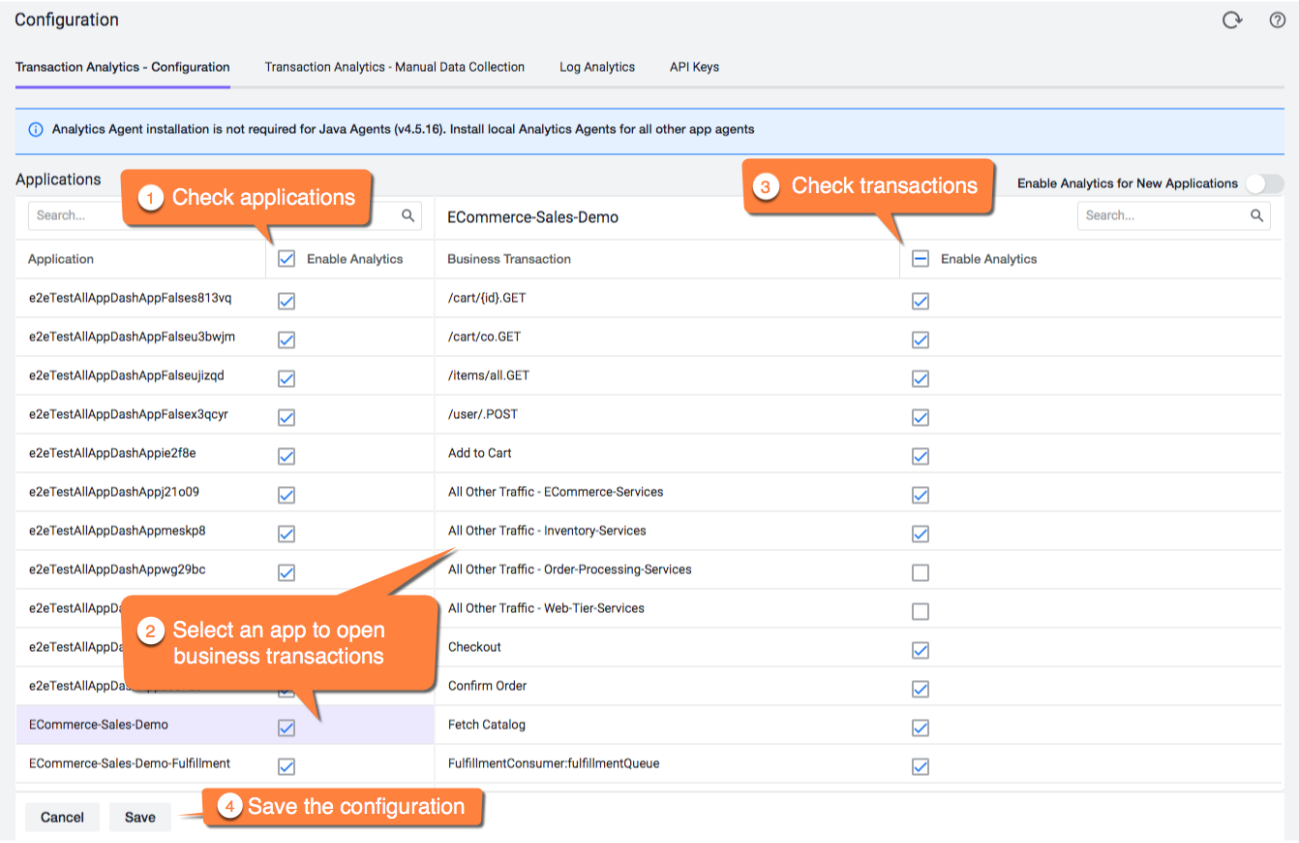
- SaaS deployments: Install Agent-Side Components
- On-premises deployments: Custom Install and Events Service Deployment

Configure Transaction Analytics

You need to configure Transaction Analytics for each desired application. Configuring Transaction Analytics consists of selecting the application and the specific business transactions that you want to analyze.

To configure Transaction Analytics:

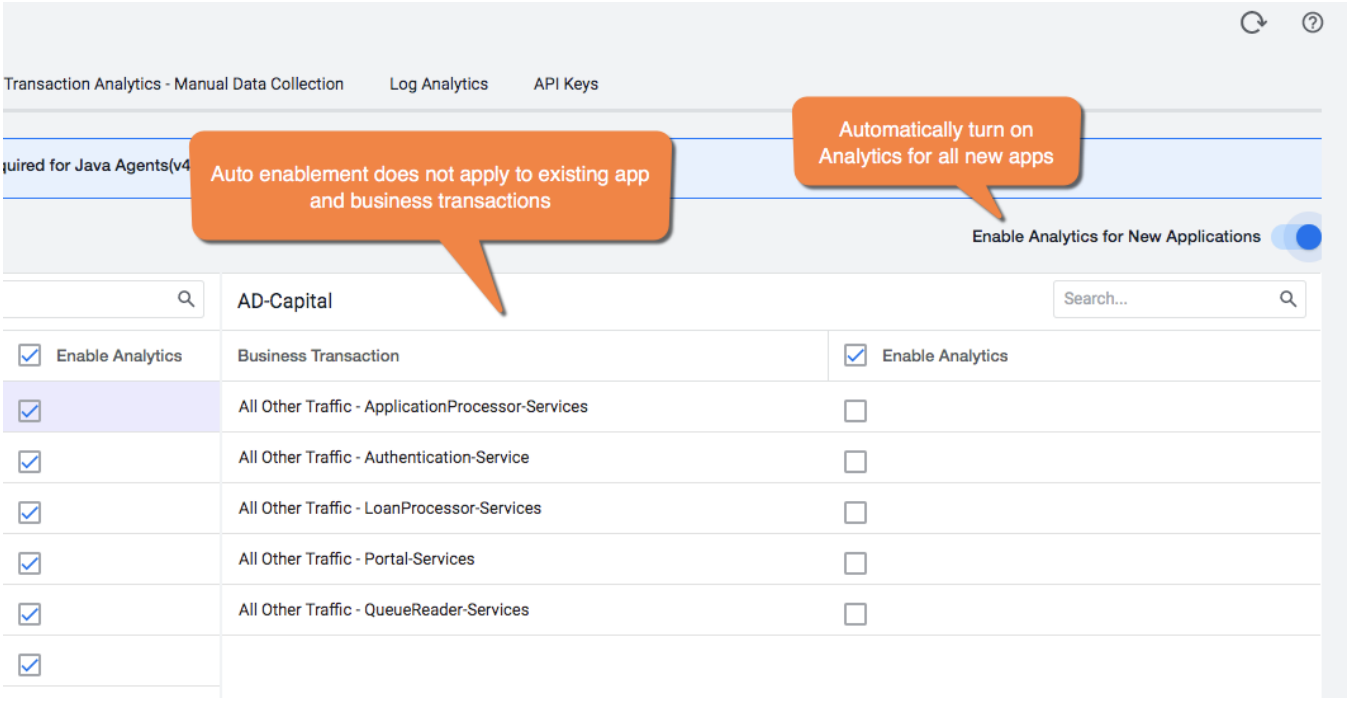
1. In the Controller UI, from the top navigation bar, select **Analytics**.
2. Select **Configuration** from the left panel.
3. Go to the Transaction Analytics - Configuration tab.
4. Check the application(s) from the left panel. You can select or deselect all applications by checking **Enable Analytics**. Find specific applications using the search box above **Applications**.
5. Select an application to open the business transactions in the right panel. By default, all business transactions, except All Other Traffic**,** are selected. You can deselect any business transactions you do not want to monitor. Find specific business transactions using the search box in the top right of the panel.
6. Click **Save**.



Automatically Enable Transaction Analytics

You can enable Transaction Analytics automatically for all applications >= 4.3.x. Click the toggle next to **Enable Analytics for New Applications**. This option configures Transaction Analytics for all new applications and all of each application's business transactions, except business transactions categorized as All Other Traffic.

Enable Analytics for New Applications applies to applications created after you select the feature. You need to manually configure Analytics for any applications and business transactions created before selecting **Enabled Analytics for New Applications**.



> You can configure Transaction Analytics for older agents manually. Go to **Transaction Analytics - Configuration** and locate the desired application. Uncheck and recheck the app, then select your business transaction(s).

Click the toggle again to turn off **Enable Analytics for New Applications**. The toggle becomes grey when the feature is turned off.

Edit Transaction Analytics Configuration

You can add or remove applications and business transactions from Transaction Analytics at any time.

To make changes to your Transaction Analytics configuration, select or deselect the desired applications and business transactions in **Transaction Analytics - Configuration**.

When you revert changes to Transaction Analytics, AppDynamics saves the previous configuration state. For example, you enable Transaction Analytics for an application and one specific business transaction. You later disable Transaction Analytics for the application, then re-enable it at a later date. When you re-enable the application, AppDynamics automatically selects the business transaction you originally selected.

Security Restrictions

The Analytics Administrator has complete access to all applications in Transaction Analytics. All other users can only view applications to which they have access based on RBAC permissions.

> The Analytics Dynamic Service loads Transaction Analytics for enabled applications after your JVM instance runs for at least two minutes.

Configure Recommended Data Collectors

Related pages:

- Configure Manual Data Collectors

This page describes Recommended Data Collectors in AppDynamics.

Recommended Data Collectors are potential Method Invocation Data Collectors (MIDC) suggested by Analytics. Recommended Data Collectors reduces the number of steps required to configure data collectors.

When you configure data collectors manually, provide the class and method parameters for each transaction that you want to monitor. Recommended Data Collectors automatically collects the event data from the Java Agent and publishes it to the Event Service. You can browse this data (transactions, classes, methods) and search for specific terms in the Controller.

> ⚠ Recommended Data Collectors does not support HTTP and SQL data collectors.

Before You Begin

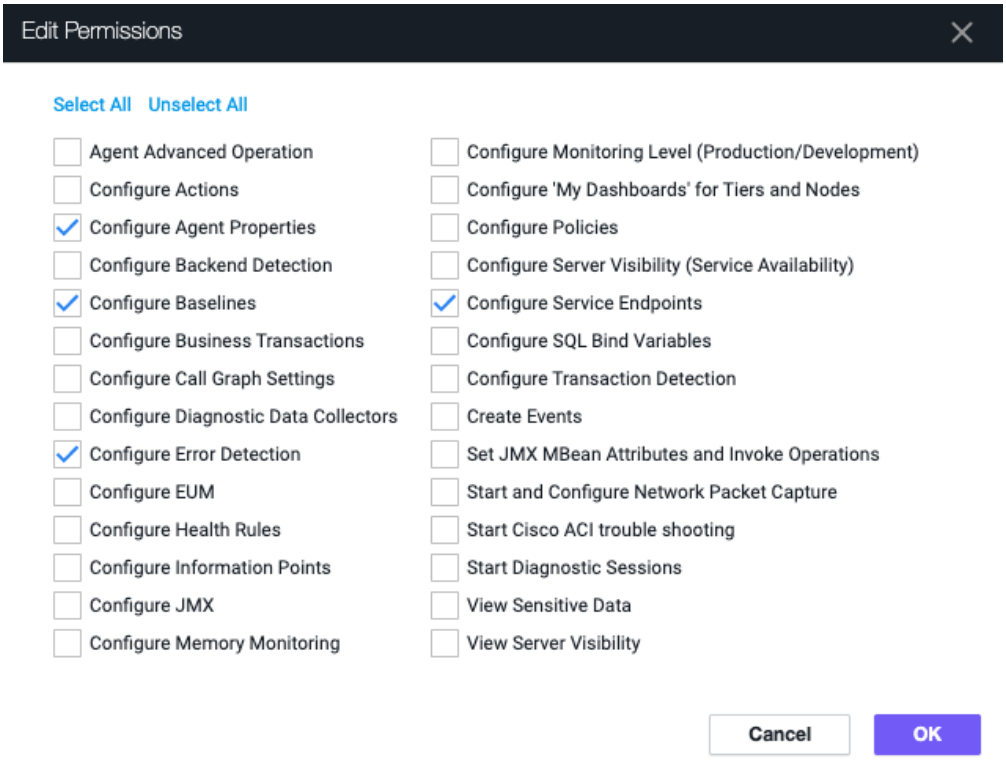
To use Recommended Data Collectors, ensure that your setup meets these requirements.

You need the **Configure Diagnostic Data Collectors** permission to access Recommended Data Collectors. The Analytics Administrator does not h default.

To view application data for Recommended Data Collectors, you need view access to the application and its transactions.

- For application permissions, configure user settings in **Administration** > **Roles** > **Applications**.
- For transaction permissions, configure user settings in **Administration** > **Roles** > **Analytics** > **Events**.

See Analytics and Data Security.



AppDynamics Components

- SaaS Controller >= 20.6.0
- Java Agent >= 4.5.19

Agentless Analytics

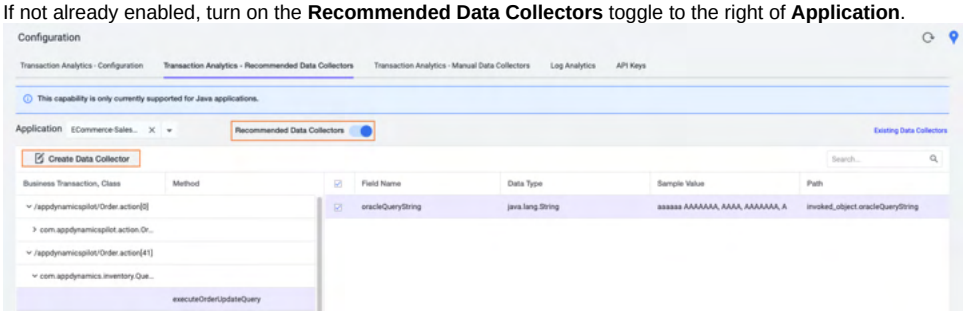
Recommended Data Collectors require Agentless Transaction Analytics. When you use the minimum supported versions of the Controller and Java Agent, Agentless Analytics is automatically enabled. See Deploy Analytics Without an Analytics Agent to ensure that your deployment works properly.

You must enable Agentless Analytics to use this feature. If you turn off Agentless Analytics, you cannot use Recommended Data Collectors.

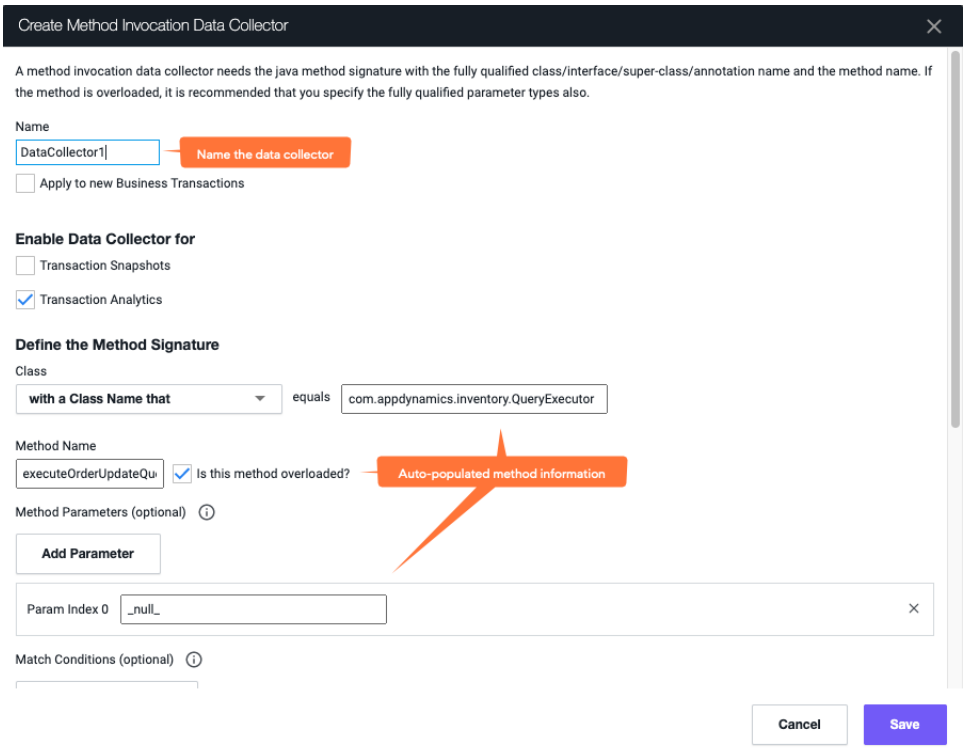
Create Recommended Data Collectors

To use Recommended Data Collectors:

1. In the Controller, select **Analytics** > **Configuration** > **Transaction Analytics - Recommended Data Collectors**.
2. Select an application from the **Application** dropdown menu. A list of business transactions appears in the left pane.
3. If not already enabled, turn on the **Recommended Data Collectors** toggle to the right of **Application**.



4. Select the dropdown arrow next to the desired business transaction name. A list of available classes appears in the left pane.
5. Select the desired class. A list of available methods appears.
6. Select the desired method.
7. Additional data appears in the right pane. Select the desired field name.
8. Above the business transaction list in the left pane, click **Create Data Collector**.
9. Enter a data collector name and confirm that the method signature information is correct. If you select multiple fields, enter a unique name for each data collector in the dialog.

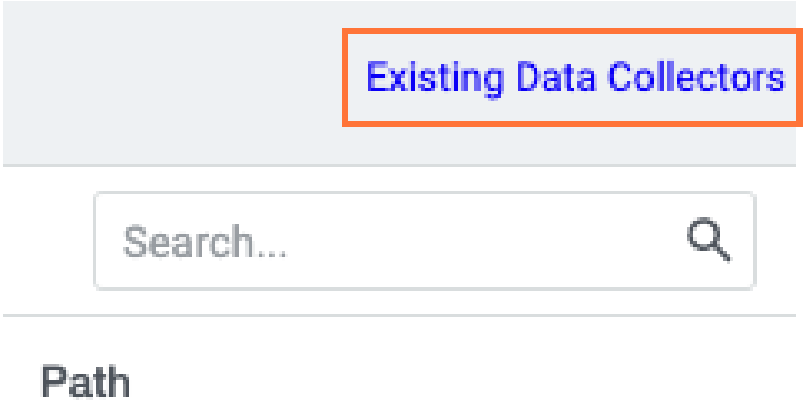


> To define a MIDC with method overloading in **Recommended Data Collectors**, you must re-select the **Is this method overloaded?** checkbox and redefine the parameters in **Manual Data Collectors** > **Edit Method Invocation Data Collector** > **Define the Method Signature**.

10. Click **Save**.

View Existing Data Collectors

Select **Existing Data Collectors** to view the created MIDCs. The **Transaction Analytics - Manual Data Collectors** tab appears.



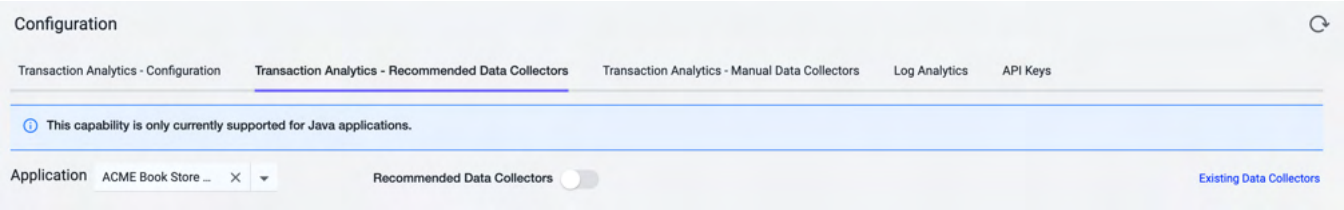
Recommended Data Collectors UI Overview

UI Term	Description
Business Transaction	The name of the business transaction(s) for the selected application. Business transaction names appear as: The relative path and business transaction ID in brackets []. For example, `/portal/Submit Application[65]`. If the Java Agent does not identify a relative path, the name is the business transaction ID only. For example, `Business Transaction 1`.
Class	The fully qualified class name of the selected business transaction.
Method	The method name of the selected class. Overloaded methods contain identifying data to distinguish each method. For example, if a class calls `method1` multiple times with different return values, the return value is displayed next to the method name: • `int method1()` • `void method1()` Variations in the method parameters are displayed in the parentheses: • `method1(str, ...)` • `method1(int, ...)`
Field Name	The name of the field of the selected method. The field name can be a parameter, return value, or invoked object.
Data Type	Type of the field name.

Sample Value	Shows an example field value. Special characters (`- , . @`) are not obfuscated. The following names are obfuscated for security: • Boolean: `aaaaaa` • Enums: the string value is pulled and obfuscated the same as upper and lower case letters • Lower case letters: a • Numbers: `0` • Upper case letters: A • All other characters: # In the case of arrays, collections, and maps, the values display a special format. • Array/Collection: `[Aaaa, ...]` • Map: `[{aaa=Aaaa}, ...]` The `Instance of <class_name>` value is displayed when the depth limit is obtained after introspecting the object. If you want to introspect further on this instance type, you can increase the introspection depth with the `analytics-runtime-introspection-depth` property. For example, in `method int getPrice(a)`, `a b c d e` where a has field b, b has field c, and so on. If the depth is 2, the value will be `Instance of c`. This case applies to arrays, collections, and maps. • Array/Collection: `[Instance of <class_name>, ...]` • Map: `[{Instance of <key_class_name>}={Instance of <value_class_name>}, ...]`
Path	Describes where the Java Agent has located the Recommended Data Collector in your code.

Disable Recommended Data Collectors for an Application

1. Go to **Transaction Analytics** > **Recommended Data Collectors**.
2. Select the application from the dropdown.
3. Turn off the **Recommended Data Collectors** toggle.



Troubleshoot Recommended Data Collectors Issues

Analytics Data Loss

When Recommended Data Collectors is enabled on an application with the Java Agent 4.5.19, restarting the agent may cause Analytics data loss. AppDynamics recommends that you upgrade to the Java Agent >= 20.3.

Recommended Data Collectors in the Controller UI

There are a few reasons why Recommended Data Collectors may not appear in the Controller UI:

- Auto-Refresh: In the Controller UI, data under the **Transaction Analytics - Recommended Data Collectors** tab does not refresh automatically. If you recently created an application and do not see it in the UI, refresh your browser.
- Deactivated Agentless Analytics: Recommended Data Collectors require Agentless Analytics. If a particular application is unavailable, confirm that Agentless Analytics is enabled for the tier or node.
- Permission: Recommended Data Collectors enforces RBAC. Without the **Configure Diagnostic Data Collectors** permission, you cannot create data collectors, but you can view suggested Recommended Data Collectors for the applications that you have access to.

Field Names Display Unavailable

Field names are not available for data captured on parameters, return values, and invoked objects, such as `int getNumber(intValue)`.

For example:

```
public class Calculator {
        public int getArea(int width, int height) {
        return width * height;
    }
}
```

Field Name	Explanation
Invoked Object	Refers to the object on which the method is invoked. In this example, the `Calculator` object is being invoked. The name of the `Calculator` object instance is not obtained, so **Invoked Object** displays **Unavailable**.
Parameters	Refers to the inputs to the method. In this example, the parameters are the width and the height. The parameter names are not obtained, so **Parameters** display **Unavailable**.
Return Value	Refers to the return value of the method. Return values do not have assigned names and **Return Value** displays **Unavailable**.

Customize Recommended Data Collectors

This page provides a list of system and node properties that customize Recommended Data Collectors.

Enable or Disable Recommended Data Collectors

You can enable or disable Recommended Data Collectors after restarting the Java Agent and setting the system properties as a JVM parameter. Recommended Data Collectors is enabled by default; the property `appdynamics.analytics.collection.feature.enabled` is optional.

System Property	Description	Default Value
`appdynamics.analytics.collection.feature.enabled`	Enable or disable Recommended Data Collectors on the Java Agent.	`true`

Customize Recommended Data Collectors

To customize Recommended Data Collectors, you can adjust the node properties in the Java Agent. See App Agent Node Properties.

Node Property	Description	Default Value
`analytics-collection-stack-traces-collected`	Number of stack traces collected per interval (time between stack trace collection limit resets).	`20`
`analytics-collection-stack-trace-collection-reset-interval`	Interval (in seconds) to reset stack trace collection limit.	`10`
`analytics-collection-special-characters`	List of characters which will not be sanitized from sample values. For example, `a^z` will not sanitize `a`, `^`, and `z`.	N/A
`analytics-collection-elements-scan-per-stack-trace`	Number of elements to scan per stack trace before stopping.	`20`

`analytics-collection-elements-retain-per-stack-trace`	Number of elements to collect per stack trace.
	
	For example, in this stack:
	
	`com.a`
	
	`-> com.b`
	
	`-> com.c`
	
	`-> com.d`
	
	`-> com.e`
	
	`-> com.f`
	
	`-> AppD exit point method`
	
	Elements `com.b` to `com.f` (bottom five) are analyzed by default.
		`5`

`analytics-collection-package-prefix`	Comma separated list of package names that will be treated preferably for suggestions.	N/A
`analytics-element-analysis-interval`	Interval (in seconds) for triggering element analysis.	`600`
`analytics-collection-reap-interval`	Interval (in seconds) to remove one suppressed collection instrumentation point. Removes the oldest suppressed point first.	`10`
`analytics-method-instrumentation`	Number of methods (unique class name/method name combinations) to instrument with Recommended Data Collectors. Overloaded variants are considered the same for this limit.	`10`

`analytics-maximum-num-rules-multiplier`	Average number of overloaded methods per instrumented method. The maximum number of instrumentation points is calculated with this formula: `analytics-method-instrumentation` * `analytics-maximum-num-rules-multiplier`.	`10`
`analytics-instrumentation-suppression`	Number of times Recommended Data Collectors is invoked and processed. For example, if `analytics-instrumentation-suppression` is set to 3 and a method is called four times, the fourth call does not collect data.	3
`analytics-runtime-attribute-value-length`	Maximum length of sample value string displayed.	32
`analytics-runtime-introspection-depth`	Level of depth in the object tree that Recommended Data Collectors extracts information.	2

Configure Manual Data Collectors

This page describes how to configure data collectors for Transaction Analytics.

There are three types of data collectors: HTTP, method invocation, and SQL. Data collectors enable you to add business context to a transaction. For example, a Method Invocation Data Collector (MIDC) can tell you if an order is slow based on method parameters.

Before You Begin

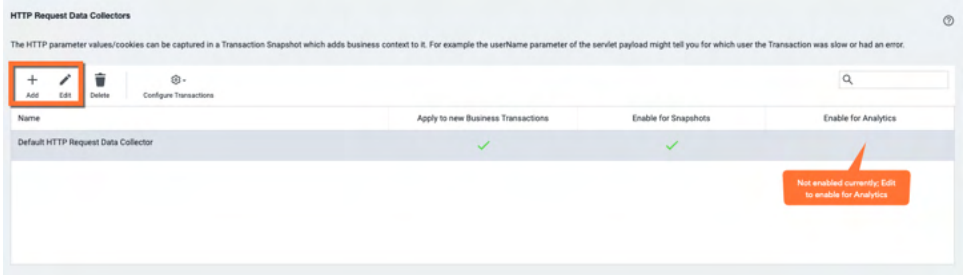
You must configure Transaction Analytics before you can configure data collectors. See Collect Transaction Analytics Data.

Configure HTTP Data Collectors

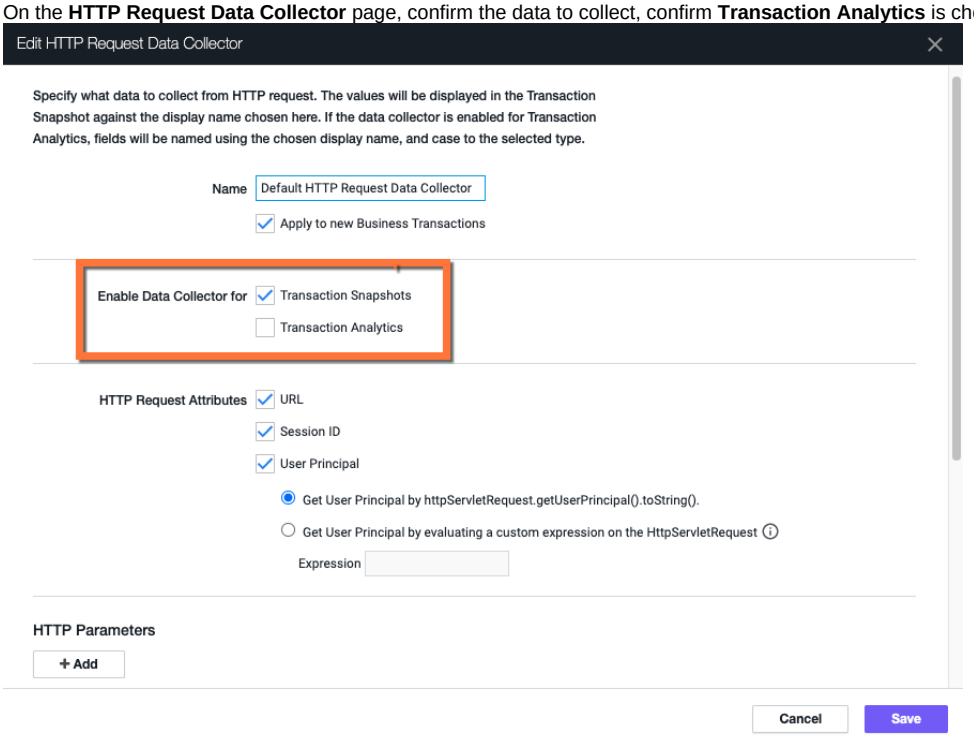
To collect HTTP request data, you can use the default HTTP Request Data Collector. You need to explicitly enable the collector for Analytics.

To configure HTTP Data Collectors for Analytics:

1. In the Controller UI, select **Analytics** > **Configuration** from the top navigation bar.
2. Go to the Transaction Analytics - Manual Data Collection tab.
3. Select the application of interest from the **Configure Analytics for Application** dropdown.
4. Navigate to **HTTP Request Data Collectors**. Select the HTTP Data Collector to enable and click **Edit** (or **Add** to create new collectors).



5. On the **HTTP Request Data Collector** page, confirm the data to collect, confirm **Transaction Analytics** is checked, and click **Save**.



6. Select **Configure Transactions** and confirm that the data collector is enabled on the appropriate business transactions. See Data Collectors.

Configure Method Invocation Data Collectors

You can also use the **Analytics** > **Configuration** page to enable existing or new Method Invocation Data Collectors. Open the **Method Invocation Data Collectors** panel. The process is essentially the same as described in Data Collectors.

- Check the **Transaction Analytics** check box to use this collector for Analytics.
- Check the **Configure Transactions** popup to confirm you have enabled the right Business Transactions.

- Check that the name of the MIDC does not exceed the 50 character limit.

Field Name Recommendations

While there are no requirements for MIDC field names, AppDynamics recommends using only alpha-numeric characters, hyphens, and underscores.

MIDC field names that start with special characters may cause application events to disappear from Analytics in the Controller. As a best practice, avoid the following characters in your MIDC field names:

- Dots
- Spaces
- Slashes
- Special characters (such as $, %, &)
- Regular expression characters

You can learn more about field name conventions and recommendations in Data Field Naming for Events Service >= 4.5.3.

> Changing the name of method invocation data collector will create a completely new field in Transaction Analytics. If you want to change the type of a data collector field, we recommend creating a new field instead to avoid conflicts with the old data.

Field Length Limit

The maximum number of characters allowed in a custom data field is 8,191.

Configure SQL Data Collectors

Transaction snapshots capture SQL queries. SQL data collectors provide a way to extract business data from parameters used in the SQL statements for use in analytics. See Collect Business Data From SQL Calls.

Enable Transaction Analytics Data Type Conversion for Data Collectors

> Transaction Analytics data type conversion requires Controller >= 4.5.6. For Java deployments, Transaction Analytics data type conversion is available for Java Agents >= 4.5.2. For .NET deployments, Transaction Analytics data type conversion is available for .NET Agents >= 4.5.5.

Data collectors determine field type (string, boolean, or number) at runtime. At times the data you collect may not be the type in which you want to use it. For example, you are not able to add metrics if any data collectors are field type string.

While field type cannot be changed, you can specify field type before runtime with Transaction Analytics data type conversion.

To enable Transaction Analytics data type conversion:

1. Log in to the administration console: `http:<controller-hostname>:8080/controller/admin.jsp or https:<controller-hostname>:45/controller/admin.jsp`.
2. On the **Controller Settings** page, change the value of `analytics.type.conversion.enabled` from `false` to true.
3. Click **Save**.

Data type conversion only appears when enabled in the **HTTP** and **Method Invocation Data Collector** configuration pages on your Controller. If this page was opened while you enabled data type conversion, you will need to refresh your page to see changes.

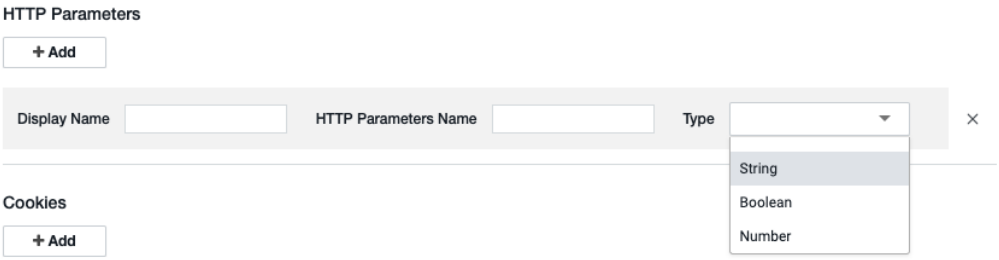
Data type conversion operates by the following rules:

- Available for new data collectors only.
- Select type manually for each data collector.
- Type cannot be changed once saved.

Transaction Analytics Data Type Conversion in HTTP Data Collectors

To use Transaction Analytics data type conversion, create a new HTTP parameter. You can add to an existing HTTP Request Data Collector or create a new one.

The process is essentially the same as creating an HTTP parameter. However, now that you've enabled Transaction Analytics data type conversion, you will see **Type** field with a dropdown.

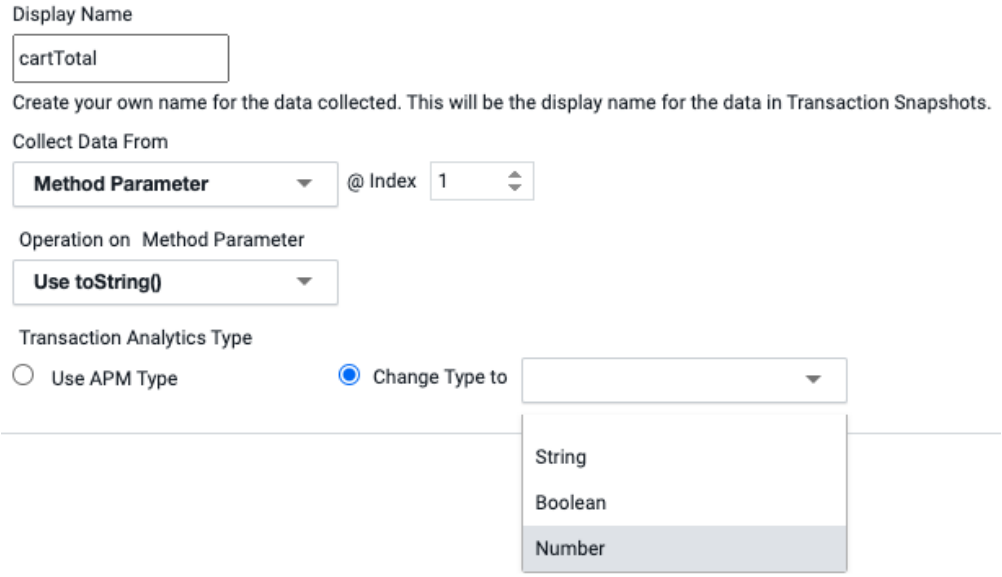


Select the desired field type for your HTTP parameter. If you do not want to specify a type, leave this field blank. The default type for HTTP data collectors is `string`.

Transaction Analytics Data Type Conversion in Method Invocation Data Collectors

To use Transaction Analytics data type conversion, create a new Method Invocation Data Collector (MIDC). You can add to an existing MIDC or create a new one.

The process is essentially the same as creating an MIDC. However, now that you've enabled Transaction Analytics data type conversion, you will see **Change Type to** field with a dropdown.



Select the desired field type for your MIDC.

Transaction Analytics Type defaults to **Use APM Type**, which determines type at runtime. If you do not want to specify a type, select **Use APM Type**.

Name Requirements For Transaction Analytics Data Type Conversion

Refer to these naming requirements when using Transaction Analytics data type conversion:

- Display Names must be unique within each data collector. Names are case sensitive.
- You can repeat Display Names across different data collector configurations, as long the field type is also the same.
- If you delete a field from a data collector and add it again with a different field type, you must rename the data collector.

Change Field Types

Field types cannot be changed once they have been saved to the data collector. To change the field type, we recommend to delete and recreate with a new name.

Configure Transaction Analytics for Node.js and PHP Applications

This page describes how to configure Transaction Analytics and Data Collectors for Node.js and PHP applications.

Before You Begin

Make sure you have installed and configured the components described in Installing Agent-Side Components and, for on-premises, Custom Install and Events Service Deployment before attempting to configure Transaction Analytics.

Configure Analytics for Node.js

Collect Default Node.js Transaction Data

To configure the Node.js Agent to send the default transaction data to the Analytics Agent, modify the require statement in your application.

Add these variables with the proper values to point to your Analytics Agent:

```
analytics: {
    host: <analyticsHostName>,
    port: <analyticsPort>,
    ssl: <true || false>
}
```

The Analytics Agent can be on the same host as the Node.js Agent or on a different host. See Install the Node.js Agent.

Configure Data Limits

You can also configure the Node.js Agent to set limits on data sent to the Analytics Agent with these properties correlated to environment variables.

Property Name	Description	Environment Variable
`analyticsMaxSegmentSizeInBytes`	The maximum size of a business transaction segment collected in an Analytics request. By default, the value is 0.1MB.	`APPDYNAMICS_ANALYTICS_MAX_SEGMENT_SIZE`
`analyticsMaxSegmentsPerRequest`	The maximum number of segments per Analytics request. By default, the value is 16.	`APPDYNAMICS_ANALYTICS_MAX_SEGMENTS_PER_REQUEST`
`analyticsMaxMessageSizeInBytes`	The size of a single request body sent to the Analytics Agent. By default, the value is 1MB.	`APPDYNAMICS_ANALYTICS_MAX_MESSAGE_SIZE`

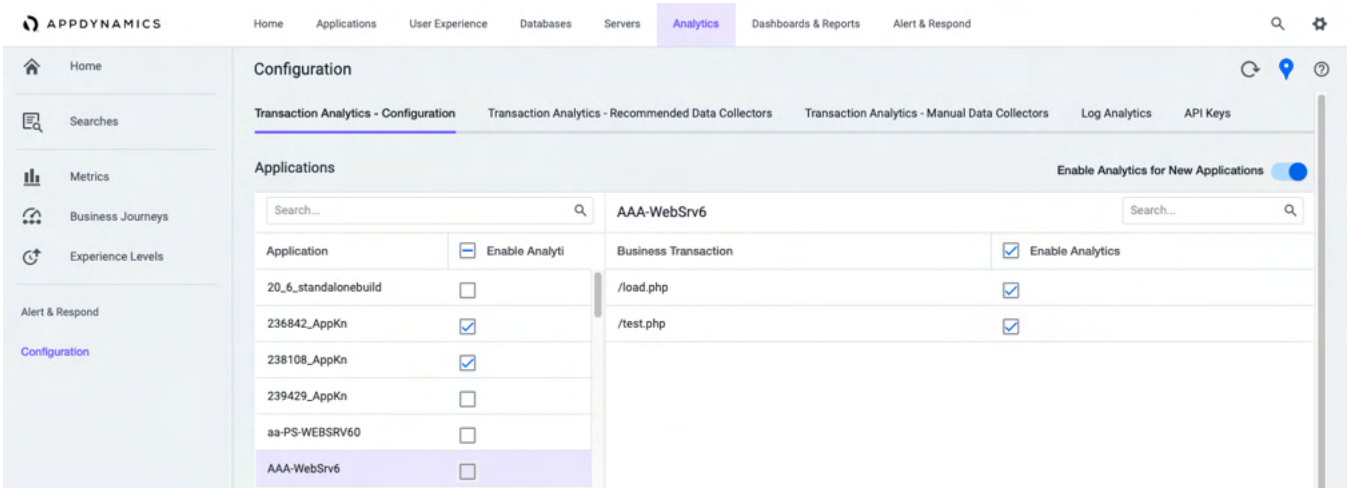
Your require statement should look similar to the following example where your Analytics Agent is on localhost and listening on Port 9090:

```
require ("appdynamics").profile({
    controllerHostName: '<Controller host name>',
    controllerPort: <Controller port number>,
    controllerSslEnabled: false, // Set to true if controllerPort is SSL
    accountName: '<AppDynamics account name>',
    accountAccessKey: '<AppDynamics account key>', // Required
    applicationName: '<Your application name>',
    debug: false,
    tierName: '<Choose a tier name>',
    nodeName: '<Choose a node name>', // The Controller appends the node name with a unique number
    analyticsMaxSegmentSizeInBytes: 20,
    analyticsMaxSegmentsPerRequest: 20,
    analyticsMaxMessageSizeInBytes: 100,
    analytics: {
        host: 'localhost',
        port: 9090 },
    logging: {
        'logfiles': [
        {'root_directory': '/tmp/appd', 'filename': 'echo_%N.log', 'level': 'TRACE', 'max_size': 5242880,
'max_files': 10 },
        { 'root_directory': '/tmp/appd', 'filename': 'protobuf_%N.log', 'level': 'TRACE', 'max_size': 5242880,
'max_files': 10, 'channel': 'protobuf' }]
            }
});
```

Collect Node.js Data Collectors

You can also collect additional data from your Node.js business transactions by using the Node.js API. To configure Transaction Analytics data collectors for Node.js, see `txn.addAnalyticsData()` in the Node.js Agent API Reference.

Configure Analytics for PHP Applications



1. In **Transaction Analytics - Configuration**, click the **Enable Analytics** checkbox to enable/disable Analytics on a PHP application.
2. To the right of **Business Transactions**, click the **Enable Analytics** checkbox to enable/disable Analytics on any Business Transaction(s) in the application.

> ⚠ Analytics data cannot be collected from PHP CLI programs.

To collect Transaction Analytics from the PHP Agent:

1. Configure the PHP Agent to send the default transaction data to the Analytics Agent. Modify the `.ini` file depending on the operating system under which your PHP Agent is installed.
   By default, your `.ini` file contains the following configuration where your Analytics Agent is on `localhost` and listens to Port 9090:

```
agent.analyticsHostName = localhost
agent.analyticsPort = 9090
```

Add the following variables with the proper values to point to your Analytics Agent:

```
agent.analyticsHostName = <analyticsHostName>
agent.analyticsPort = <analyticsPort>
```

The Analytics Agent can either be on the same host as the PHP Agent, or on a different host. For information about installing the PHP Agent, see Install the PHP Agent.

You must set the node level setting `analytics-dynamic-service-enabled=true` for reporting analytics.

2. HTTP/Method Invocation Data collectors for Analytics must be enabled for both transaction snapshots and Transaction Analytics to report data from the PHP Agent.
   In order to collect the method return value with an MIDC, the value must be assigned to a variable.

```
$ret = function()
```

If the return value is not stored in any variable, it displays as `null` in both snapshot and Analytics data.

Add Custom Fields to Transactions Using Java SDK

You can add custom data using the Java Agent SDK as an alternative to adding data collectors from the Controller UI.

The fields show up on Transaction Analytics under a section called "Custom Method Data".

Add the code similar the following to your application:

```
//include the app agent SDK packages

import com.appdynamics.apm.appagent.api.AgentDelegate;
import com.appdynamics.apm.appagent.api.DataScope;
import com.appdynamics.apm.appagent.api.IMetricAndEventReporter;

public void someMethod(String stringParam, int intParam, ComplexObject complexObjParam) {

    //Get the data reporter.
    IMetricAndEventReporter reporter = AgentDelegate.getMetricAndEventPublisher();

    //Define the scope for data collection.
    Set<DataScope> allScopes = new HashSet<DataScope>();
    allScopes.add(DataScope.ANALYTICS);
    allScopes.add(DataScope.SNAPSHOTS);

    Set<DataScope> analyticsScope = new HashSet<DataScope>();
    analyticsScope.add(DataScope.ANALYTICS);

    //Add data to different scopes based on the need.
    //Here both key1 and key2 are collected for both snapshots and analytics
    reporter.addSnapshotData("key1", stringParam, allScopes);
    reporter.addSnapshotData("key2", intParam, allScopes);

    //Here key3 is only collected for analytics.
    //Since the passed object is a complex java object and not a primitive type, toString() method is invoked
on it and the string value will
    //be collected.
    reporter.addSnapshotData("key3", complexObjParam, analyticsScope);

    .........
    Business logic.
```

The third parameter to the method is a set of enums, DataScope, that define the scope for collecting the custom data:

- To collect custom data only for snapshots, use DataScope.SNAPSHOTS as the third parameter.
- To collect the custom data for analytics, use DataScope.ANALYTICS as the third parameter.
- If custom data should be added to both snapshots and analytics then both the enums should be passed in a set as the third parameter.

Method details for `addSnapshotData()`

```
boolean addSnapshotData(java.lang.String key,
                        java.lang.Object value,
                        java.util.Set<DataScopes> dataScopes
```

Method Parameters

- key - key
- value - Primitive types can be passed as values and the types (int, boolean, etc) will be respected. Strings can be passed as values, however, note that if you pass composite objects then agent performs a .toString() operation on it.
- dataScopes - scope for the custom data collection:
    - Datascope.SNAPSHOTS
    - Datascope.ANALYTICS

Returns

true if the agent successfully collects the key-value pair for the given scope.

Collect Business Data From SQL Calls

Transaction snapshots capture SQL database calls. The SQL calls can contain useful business data. Analytics SQL data collectors are a way to collect the business data from SQL parameters for use in transaction analytics.

To configure an Analytics SQL data collector you need to know:

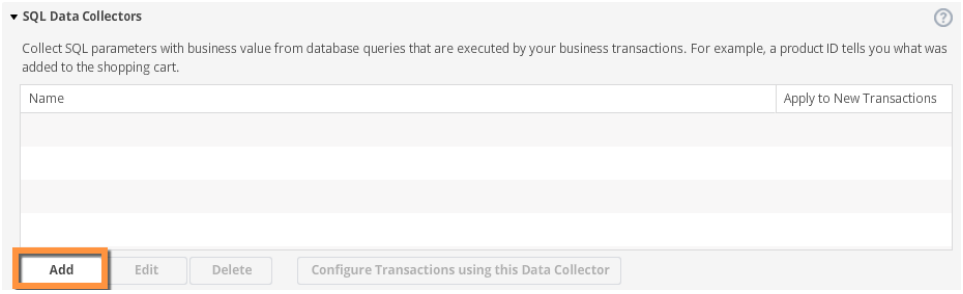
- Analytics-enabled application where the SQL call is executed.
- Database target of the SQL call.
- Specific SQL parameterized statement that contains the data of interest as a query parameter.
- Analytics-enabled business transactions making the database call.
- Parameters to be collected.

This feature is supported as:

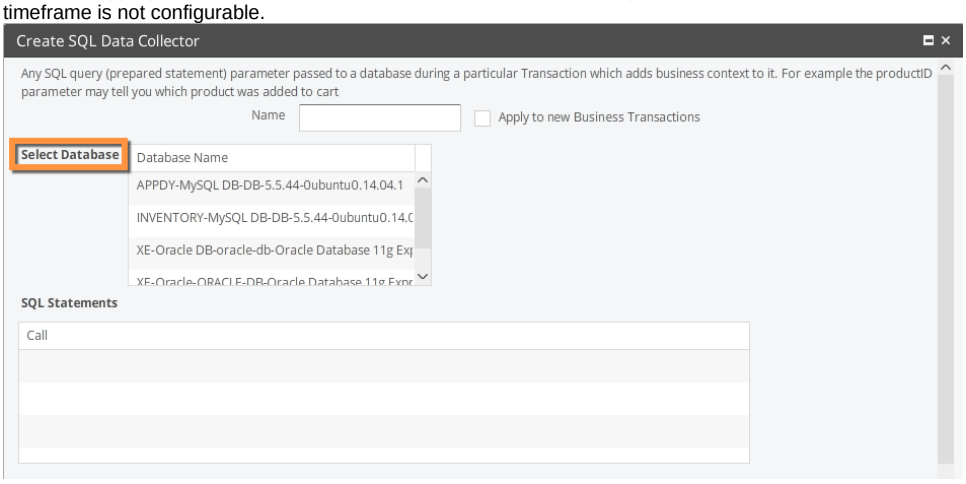
- The SQL data collector list shows the slowest database calls over the last 30 days. You can also create a SQL data collector directly from the **DB & Remote Service Calls** tab of a snapshot.
- Requires 4.3 Java Agent or 4.3 .NET Agent.
- Only prepared statements containing data of interest as binding variables can be used. Literal strings that are passed in can not be collected.
- There is a 500-character limit on the length of the SQL statement. Do not use any truncated queries that might appear in the SQL Statements list or a snapshot.
- The overall number of executions of SQL queries configured to collect Analytics data is limited to 10K. This is configurable using the `analytics-sql-cpm-limit` node property. See App Agent Node Properties.

Configure SQL Data Collectors From Analytics

1. In the Controller UI, from the top navigation bar, select **Analytics > Configuration**.
2. From the Transaction Analytics tab, select the application from the **Configure Analytics for Application** dropdown and confirm that analytics data collection is enabled.
3. Scroll down to expand the **SQL Data Collectors** section, and click **Add**.

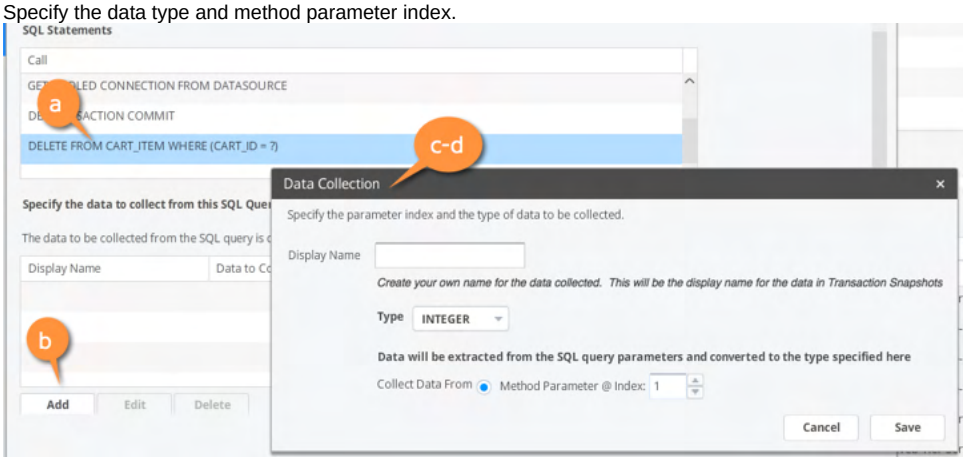


4. Name your data collector and indicate if the collector should apply to new business transactions.
5. Select the appropriate database. A list of available SQL statements displays showing the slowest database calls over the last 30 days. This timeframe is not configurable.



6. Define the data to collect.
    a. Select the SQL prepared statement containing the parameter that you want to capture for analytics.
    b. Click **Add** to specify the data to be collected.
    c. Type a display name for the data you are collecting.
    This name appears in the Analytics UI Fields list when the data is collected and passed to Analytics.

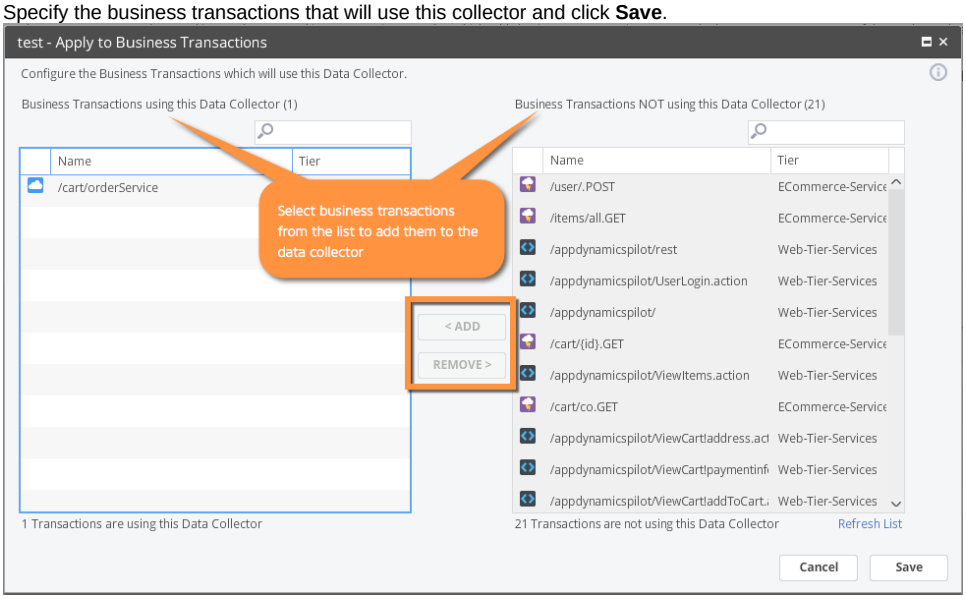
d. Specify the data type and method parameter index.



e. Click **Save**.

f. Click **Create SQL Data Collector**.

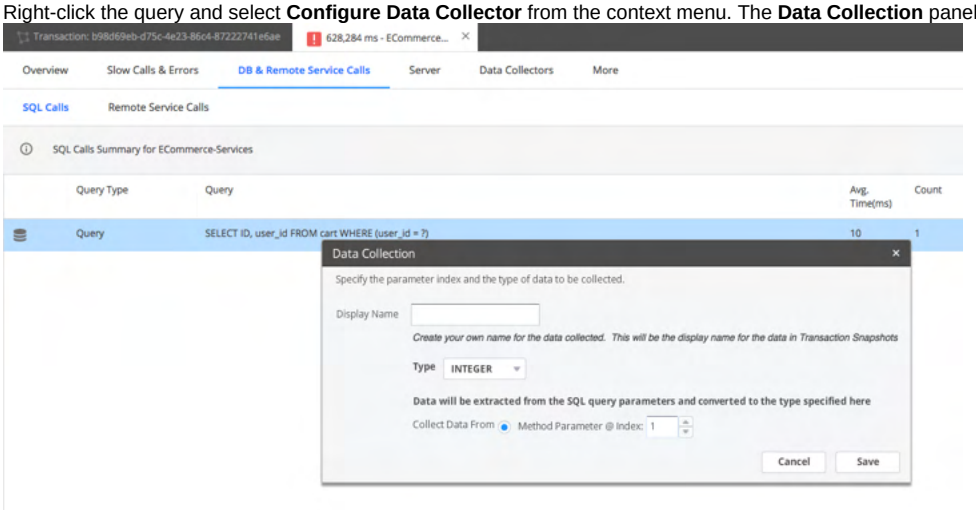
7. Specify the business transactions that will use this collector and click **Save**.



Configure SQL Data Collectors From Snapshots

This procedure provides a shortcut way to set up the configuration for an Analytics SQL data collector for a database call captured in an application transac tion snapshot. You need to find the transaction snapshot of interest and drill down to the node that contains the database call of interest.

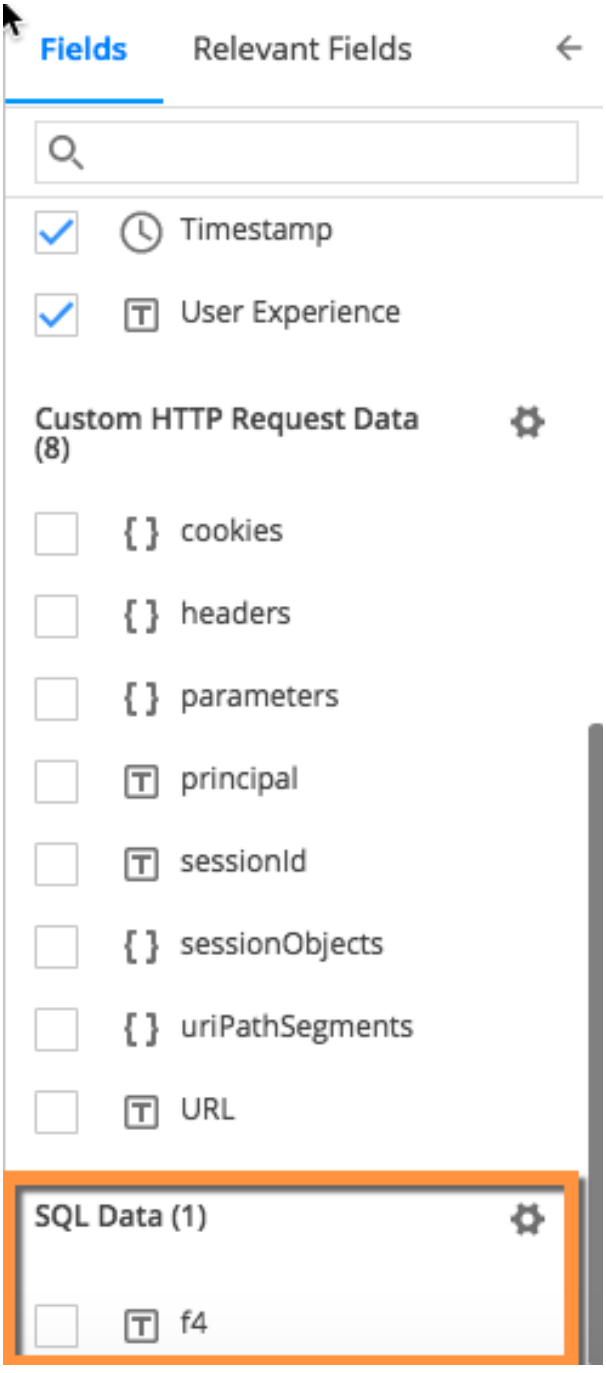
1. From the **DB & Remote Service Calls** tab of a snapshot containing the SQL call of interest, select the prepared statement that contains the data you want to collect for analytics.
2. Right-click the query and select **Configure Data Collector** from the context menu. The **Data Collection** panel displays.



3. Enter a display name for the data you are collecting.
4. Specify the type and the parameter to collect and click **Save**.
5. Select the business transactions from which to collect the data and click **Save**.

View SQL Data in Analytics

After the data is successfully collected for analytics, you see the fields in the **Fields** list in a section labeled SQL Data. This can take a few minutes depending on your application and the frequency of transactions executing the relevant SQL query.



Collect Log Analytics Data

Related pages:

- Configure Log Analytics Using Job Files
- Analytics Log Data

To capture and present log records as analytics data, you must configure one or more *log sources* for the Analytics Agent. The Analytics Agent uses the log source configuration to:

- Capture records from the log file
- Structure the log data according to your configuration
- Send the data to the Analytics Processor.

The Controller presents the Log Analytics data in the Analytics UI.

Before attempting to configure Log Analytics, confirm you have installed and configured the components described in Install Agent-Side Components and, for on-premises, Custom Install and Events Service Deployment.

Versions < 4.3 use *job files* to configure the log sources. You may continue to use job files that were created in previous versions. If you want to collect new log events into our platform, we recommend that you use the Centralized Log Management UI to define *source rules*. You may also find it useful to replace existing job file configurations with the new source rules so you can take advantage of new features introduced in 4.3. See Migrate Log Analytics Job Files to Source Rules.

To configure data collection for your log sources, see Configure Log Analytics Using Source Rules.

Manage Extracted Fields

In Controller < 4.3, you could extract fields from logs in the Controller UI. This is described on the Create Extracted Fields from Logs page. This option is not available >= 4.3. It is replaced entirely by Field Extraction in the Centralized Log Management UI. See Configure Log Analytics Using Source Rules for details.

Fields that were extracted by this mechanism in previous versions appear in the Extracted Fields list. Hovering over the field reveals a **View** icon. Click the **View** icon to delete the field or to see the configuration details:

- Regular expression used to extract the field
- Field name
- Field type
- Source type

Configure Log Analytics Using Source Rules

Related pages:

- Agent Scopes for Configuration Log Management
- Analytics Log Data
- Collect Log Analytics Data from Syslog Messages

The Centralized Log Management UI enables you to configure your log data sources using source rules.

Once you define source rules, you specify which Analytics Agents should use the rules by associating specific source rules with Agent Scopes. Agent Scopes are groups of Analytics Agents that you define. Using Agent Scopes simplifies the deployment of the log source configuration to multiple agents.

Log Analytics Source Rules

Analytics Agents collect logging information based on the log source rules you define in the Controller.

The primary function of a source rule is to specify the location and type of a log file, the pattern for capturing records from the log file, and the structure of the data of captured records. They can also specify field masking or sensitive data removal and manage time zones of captured records.

When the log source rules are enabled and associated with Agent Scopes, Analytics Agents automatically start collecting the configured logs as follows:

- Analytics Agents register with the Controller on startup
- Analytics Agents download log source rules to configure log collection (after registration)
- Log source rules are stored in the Controller data store and are configurable through the Centralized Log Management UI
- Analytics Agents start acting on log source rule changes within five minutes (this could be longer if there are any network communication issues)

Permissions

To configure log analytics source rules, you need the following permissions:

- Manage Centralized Log Config permission
- View access to the specific source types that you want to configure

See Analytics and Data Security.

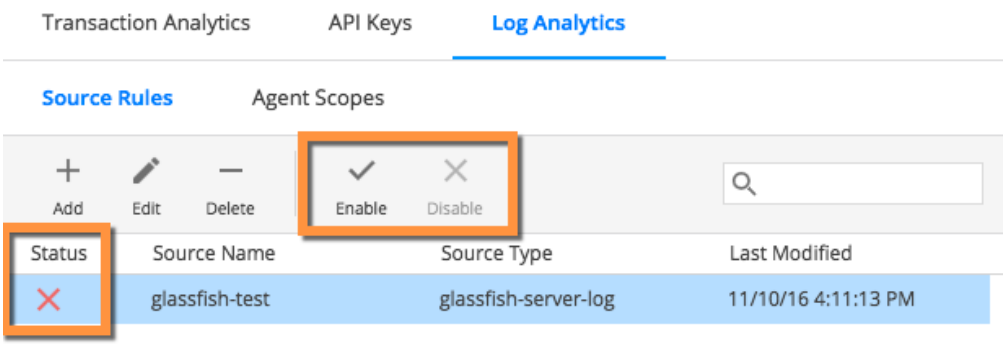
Analytics Agent Properties

If you are using Centralized Log Management (CLM), you need to configure additional properties in the `analytics-agent.properties` file. See information on configuring properties for consolidated log management in Install Agent-Side Components for details.

Managing Source Rules

When you create a source rule and save it, the rule is initially disabled. This is shown in the **Status** column on the source rule list page.

You can enable or disable a source rule by selecting it and choosing the corresponding option from the **Source Rules** menu, shown in the following screenshot:



You can also see which source rules use a specific agent scope. The Analytics Agent does not collect the log data until the source rule is enabled and assigned to an Agent Scope with active Analytics Agents.

Starting Point for Creating a Source Rule

You can create a source rule from any one of these starting points:

- AppDynamics template. Several templates for common log file formats are available.
- Existing source rule. You can use existing source rules as the starting point for new rules.
- New source rule. Start from scratch when your log file does not match one of the available templates.

Preview Extracted Log Data Using Sample Log Files

To improve validation of data collection and parsing of log messages, you can use a local log file with the Log Analytics Configuration UI to preview the field extraction that you want. Three types of field extraction are available.

- Grok patterns and key-value pair extraction
- Auto extraction using regular expressions
- Manual extraction using regular expressions

Best Practices for Source Rule Design

A few recommendations apply to making source rules.

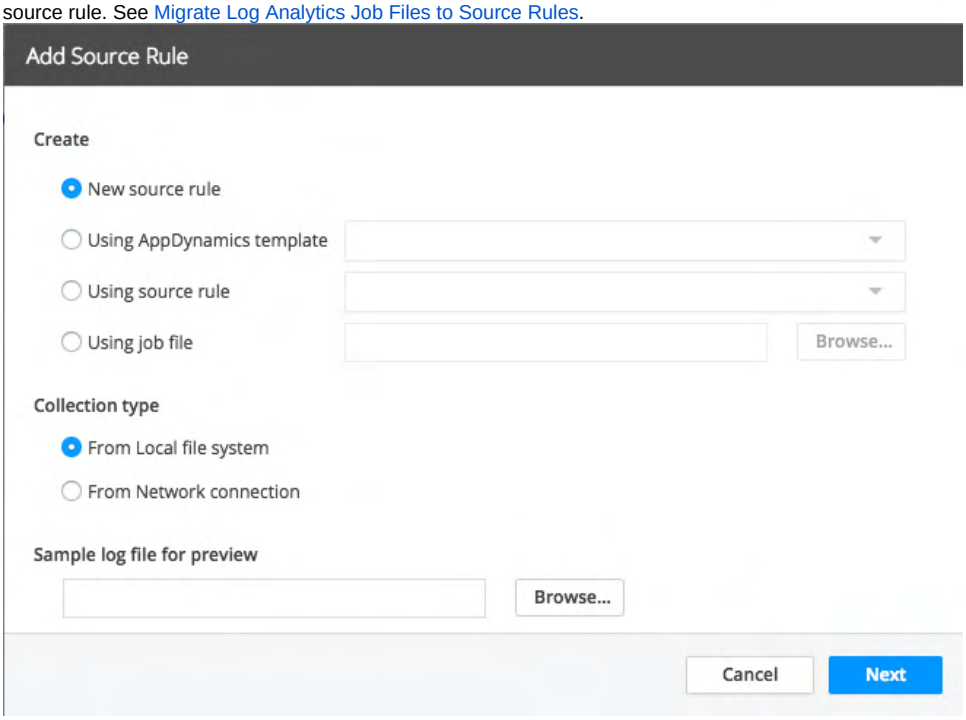
Use the simplest possible regular expressions and grok matching patterns possible. Do not make excessive use of wildcards or quantifiers as it can slow the responsiveness of the Controller UI. You can view examples of such greedy quantifiers on https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html.

If a regex pattern (including grok) is taking more than five seconds to match against a logline, extraction and further processing for those fields stop. If this occurs, some fields may be missing for that log line when viewed on the Controller. Other log lines are not impacted; however, this occurrence is often the result of an ineffective or faulty matching pattern in the first place, and processing is likely to take a long time for all log lines. This behavior is applicable to the dynamic Preview screens in the **Centralized Log Analytics Configuration** page as well.

The Controller limits the size of the records it retrieves to 32 KB by source rule in log analytics. This limit guards against excessive system resource burden, including potential resource burden that might be caused by excessive data collection due to faulty source rule patterns.

Create a Source Rule

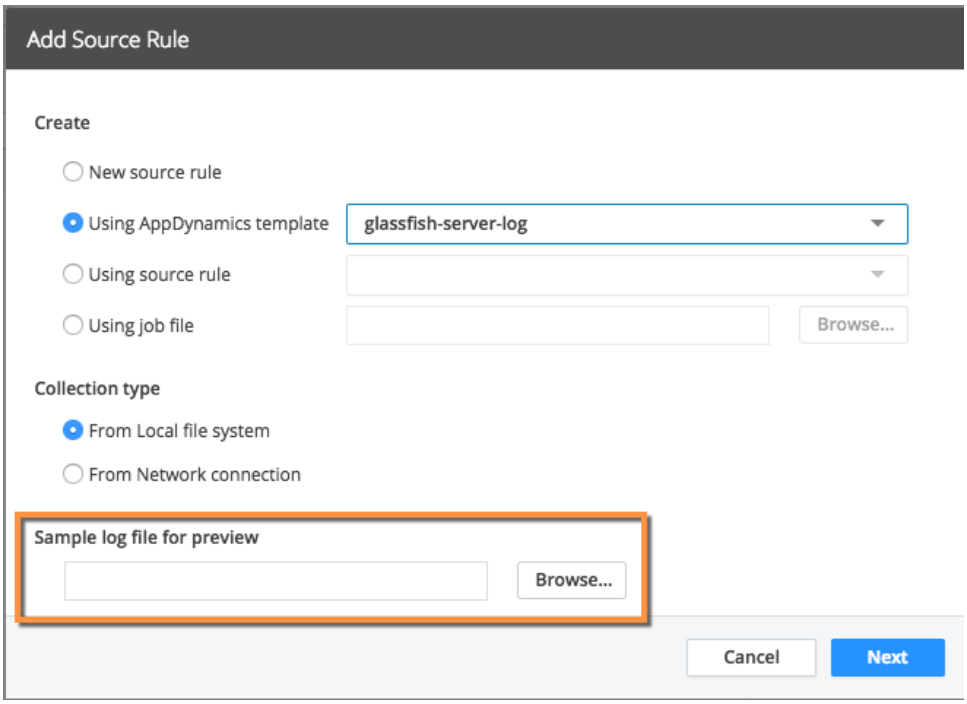
1. From the Controller top navigation bar, click **Analytics**.
2. From the left navigation panel, click **Configuration > Log Analytics**.
   You see two tabs, one for Source Rules and one for Agent Scopes.
3. From the Source Rules tab, click **+ Add .**
   You see the **Add Source Rule** panel.
4. In the **Add Source Rule** panel, select your starting point for the source rule. Note that you can also use a job file as the starting point for your source rule. See Migrate Log Analytics Job Files to Source Rules.



5. Use **Collection Type** to indicate if the source log file resides on the local filesystem or will be collected from a network connection.

> Collecting from a network connection is only used for a TCP source rule, which extracts log analytics fields from {{syslog}} messages over TCP. See Collect Log Analytics Data from Syslog Messages.

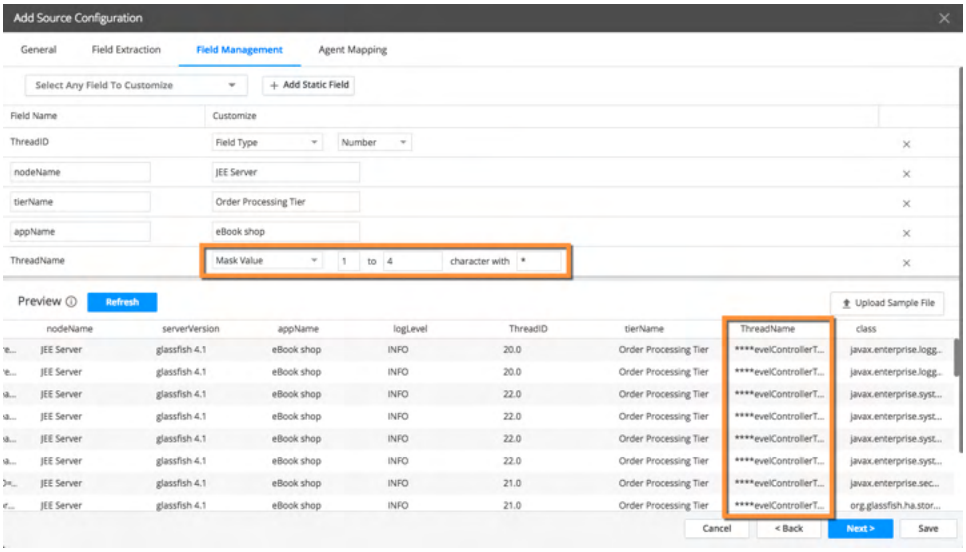
6. Use the **Browser** button to locate and specify a sample log file to preview the results of your configuration. You can also specify a sample file later in the configuration process during the field extraction step.



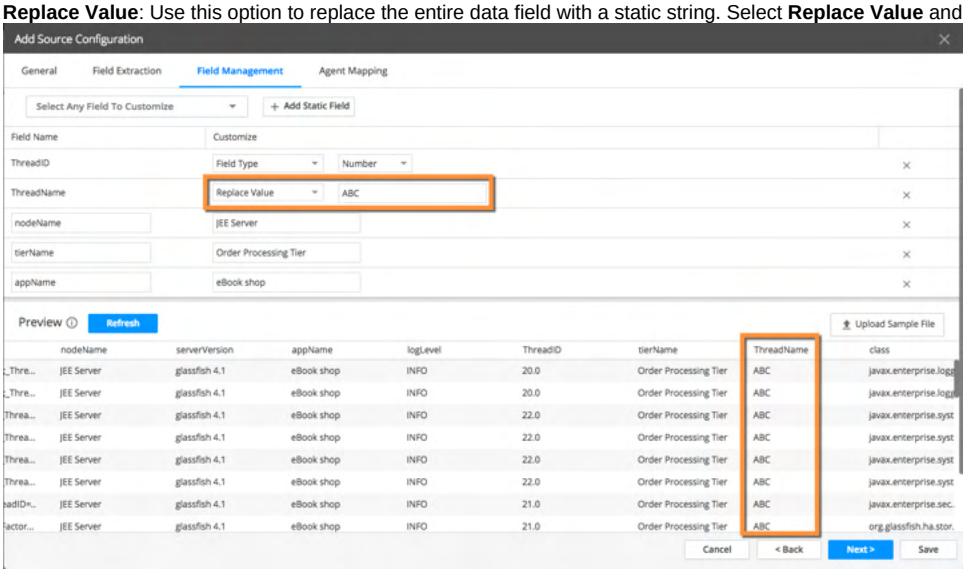
7. Click **Next** to see the **Add Source Configuration** page. Some fields may be prepopulated with data if you selected either an AppDynamics template or an existing source rule as your starting point. The four tabs are:
    - General
    - Field Extraction
    - Field Customization
    - Agent Mapping
8. On the General tab, name your rule, specify its location, timestamp handling, and other general characteristics. See General Configuration.
9. On the Field Extraction tab, configure the fields that you want to capture from your log file. For more details on the fields on this subtab, see Field Extraction. For a detailed procedure on using Auto Field Extraction, see Field Extraction for Source Rules.

    > Use the simplest possible regular expressions and grok matching patterns. Do not make excessive use of wildcards or quantifiers as it can slow the responsiveness of the Controller UI. Examples of expensive and greedy quantifiers can be found here https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html

10. On the Field Management tab, you can customize the handling of fields in a number of way including masking sensitive data, renaming fields, changing the data type and more. See Field Management.
    Mask Value: Use this option to mask values of sensitive data such as credit card numbers of social security numbers. Select **Mask Value** and enter the starting index and ending index (1 and 4 respectively for the example shown) and a character to use for masking in the display, such as an * asterisk.



Replace Value: Use this option to replace the entire data field with a static string. Select **Replace Value** and enter the string to use:



11. On the Agent Mapping tab, assign the source rule to specific Agent Scopes.
12. Click **Save** when you are finished. The source rule is saved in a disabled state.

Centralized Log Management UI Details

General Configuration Tab

Field	Description	Required
Source Name	Name of this source rule. This name must be unique. Appears in the list of defined source rules.	Yes
Source Type	Specifies the event type of log source file. This field is prepopulated when you start from an AppDynamics template or an existing source rule. If you are creating a new source rule from scratch, you can specify any value. This value is used to identify this specific type of log event and can be used in searching and filtering the collected log data. The value shows up in the field list for the log data.	Yes

Source File	The location and name of the log file to serve as a log source. The location must be on the same machine as the analytics-agent. You can use wild cards and you can specify whether to match files one level deep or all log files in the path directory structure. • Example for multi-level matching: path: /var/log/**/*.log This matches both /var/log/apache2/logs/error.log and /var/log/cassandra/system.log • Example for one level matching: path: /var/log/*/*.log This searches for .log files one level deep in the /var/log directory (matches on /var /log/cassandra/system.log but not on /var/log/apache2/logs/error.log).	Required when collection type is from the local filesystem.
Exclude Files	Exclude, or blocklist, files from the defined source rule(s). Input the relative path of the file (s) to exclude, using wildcards to exclude multiple files. • Example for a single file: **Source File**: /var/log/**/*.log **Exclude Files**: cassandra/system.log • Example for multiple files: **Source File**: /var/log/**/*.log **Exclude Files**: */system.log	No
TCP Port	Specifies the port for the analytics-agent to collect log files from a network connection. This field is not present if the collection type is **From local file system**. If no port number is provided, port 514 is used. Both the syslog utility and analytics-agent should have root access to send logs to port 514 (binding to ports less than 1024 requires root access).	Required when collection type is from a network connection.
Enable path extraction	This checkbox enables extraction of fields from the path name. Add a grok pattern in the text box. The syntax for a grok pattern is {%{SYNTAX:SEMANTIC}. For example, to extract `Admin Sever` from `/opt/apps/oracle/middleware/user_projects/domains /ouaf_domain1/servers/AdminServer/logs/AdminServer.log`, enter the following GROK pattern: servers/`%{DATA:servername}/%{GREEDYDATA},` where `%{DATA:server}` filters the server name and `%{GREEDYDATA}` filters rest of the message from the log path.	No
Start collecting from	Indicate where to begin tailing (collecting the log records). Options are: • From the beginning of the log file • From the end of the log file • A specific time range (in hours) - For example, if you set this to four hours, then when you enable the rule and analytics starts tailing the log data, only the last four hours of data is ingested. When tailing starts, the watermark state is maintained for that file. If the agent is now stopped and restarted, it will start tailing from where it left off for any old file. For new files, it will only tail the last four hours of data.	The default is to collect log records from the beginning of the file.
Override time zone	Use this to override the time zone for eventTimestamp in log events. If your log events don't have a time zone assigned and you are not overriding it, the host machine time zone is assigned to the log events. You cannot override the timezone in the pickupTimestamp field. If you override the time zone, you must provide a timestamp format. Time zone formats should conform to Joda-Time Available Time Zones, and the timestamp format should be created with the Joda-Time formatting system.	No
Override timestamp format	You can override the format if needed. Time zone formats should conform to Joda-Time A vailable Time Zones, and the timestamp format should be created with the Joda-Time formatting system.	Yes. The override timestamp format is needed to extract out time correctly from the log file. If you do not provide this field, the `ev entTimestamp` will be equal to `pickupTimestamp` despite log line having a timestamp.
Auto-Correct duplicate Timestamps	Enable this option to preserve the order of original log messages if the messages contain duplicate timestamps. This adds a counter to preserve the order.	No
Collect Gzip files	Enable this to also find gzip files in the specified path.	No

Field Extraction Tab

The following table describes the fields and actions on this tab. For a detailed procedure on using Auto or Manual Field Extraction, see Field Extraction for Source Rules.

Section /Field or Action	Description
Add Grok Pattern	
Message Pattern	This is the grok pattern used to extract fields from the log message. A pattern may be prepopulated when you use an AppDynamics template or an existing source rule as your starting point. You can add or remove grok patterns as needed.
Multiline Format	For log files that include log records that span multiple lines (spanning multiple line breaks), use this field to indicate how the individual records in the log file should be identified. You have two options: • **startsWith**: A simple prefix that matches the start of the multiline log record. • **regex**: A regular expression that matches the multiline log record. The multiline format is not supported when you are collecting log data from a network connection.
Extract Key-Value Pairs	
Field	Shows the field selected for key-value pair configuration.
Split	The delimiter used to separate the key from its value. In this example, "key=value": the split delimiter is the equal sign "=". Multiple comma-separated values can be added.
Separator	The delimiter used to separate out two key-value pairs. In this example, "key1=value1;key2=value2": the separator is the semi-colon ";". Multiple comma-separated values can be added.
Trim	A list of characters to remove from the starting and/or the end of the key/value before storing them. In this example, "_ThreadID_": you can specify the underscore "_" as the trim character to result in "ThreadID". Multiple comma-separated values can be added.
Include	A list of key names to capture from the "source". You must provide keys in the include field. If the include field is left blank no key-value pairs are collected.
Actions	
Upload Sample file	Browse to a local file to upload a sample log file.
Preview	Use to refresh the Preview grid to see the results of the specified field extraction pattern.
Auto Field Extraction	
Definer Sample	Select a message from the preview grid that is representative of the fields that you want to extract from the log records. You can select only one definer sample per source rule.
Refiner Samples	Refines the regular expression to include values that were not included in the original definer sample.
Counter Sample	This specifies something to ignore. Refines the regular expression to exclude values that were included by the original definer or refiner sample.
Manual Field Extraction	
Regular Expression	Add a regular expression to define the field you want to extract.
Field Type	Specify the type for the field.
Field Name	Automatically generated within the regular expression pattern. This name appears in the Fields list in the Analytics Search UI.
Actions	
Add Field	Use to add additional regular expressions for extracting more fields. You cannot add more than 20 fields.

Preview • All • Matching • Non-Matching	Use these buttons to filter the viewable results in the preview grid.
Upload Sample File	Upload a sample file from your local file system to use in the preview grid.

Field Management Tab

Field/Action	Description
Select Any Field to Customize	Select the field to add customizations. You can add multiple customizations to a single field.
Field Name	This column lists the fields that have customizations.
Customize	This column shows the specific customizations. For static fields, the display name is shown. Static fields cannot be further customized.
Mask Value	This customization option masks values in the collected data. Specify the starting and ending position in the data and the character to use as the masking value.
Replace Value	This customization option replaces the entire value of the field with a static string.
Rename	This customization option enables you to rename a field to a more recognizable display name.
Field Type	Use to change the data type of the field. For example, from string to number. Available types are String, Boolean, and Number. Note that after a source rule has been saved with a field of a specified data type, you cannot later change the data type of that field. Once the fields are indexed in the analytics database trying to specify a new type will cause a validation error.
Remove	This customization turns off data collection for the field. It can be reversed at a later time.
Add Static Field	This action allows you to add a static field to all the log events collected from this source. This field can then be used to search and filter the log data. For example, use this to add Tier, Node, and Application Name to the log data.

Field Extraction for Source Rules

This page describes how to use Auto and Manual Field Extraction to configure Log Analytics source rules. Field Extraction uses regular expressions to identify and format the fields for your Log Analytics data.

Auto Field Extraction enables you to upload a sample log file and select fields for extraction. The necessary regular expressions are automatically generated and highlighted in your sample messages. You can fine-tune the generated regular expressions using Refiner Sample and Counter Sample log messages.

Manual Field Extraction enables you to upload a sample log file and enter your own regular expressions to define fields and associate the fields with a data type.

Definitions

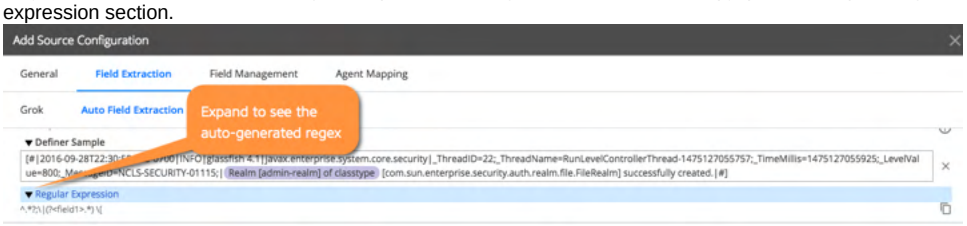
- Source Rule: a set of configuration settings for extracting analytics data from log files. You create source rules using the Centralized Log Management UI in the Controller. Source rules are stored in the Controller data store and periodically synced with the Analytics Agent. See Configure Log Analytics Using Source Rules.
- Sample File: a representative log file uploaded to the Controller that provides a way to test and fine-tune your log file source rules.
- Definer Sample: a specific log message selected from the sample log file used to define the fields that you want to extract from the log messages.
- Refiner Sample: an additional log message used to revise the auto-generated regular expression created in the Definer Sample step. The Refiner sample helps capture fields that were missed in the initial step.
- Counter Sample: an additional log message used to eliminate false positives while extracting the fields as defined in the Definer or Refiner steps.

Auto Field Extraction

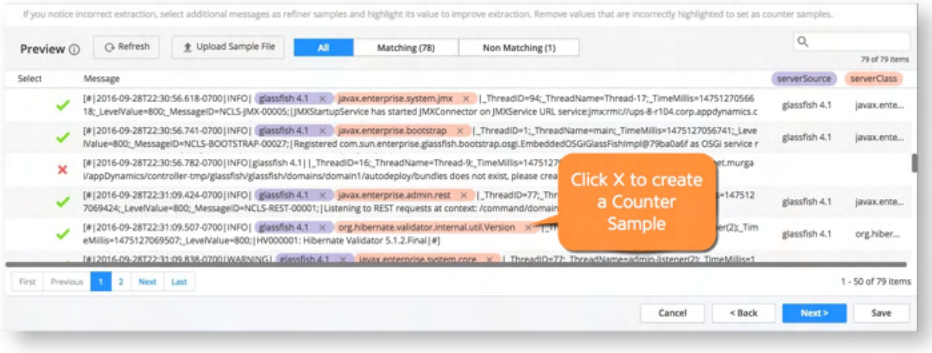
This section describes how to use Auto Field Extraction to extract fields from your log files.

To specify field extraction using a Definer Sample log message:

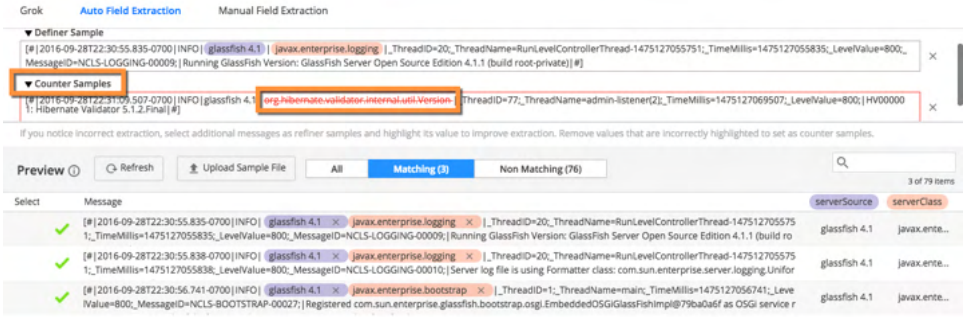
1. Using the Centralized Log Management UI, add a new source rule (or edit an existing source rule), and navigate to the Auto Field Extraction subtab within the Field Extraction tab. See the section "*Create a Source Rule*" in Configure Log Analytics Using Source Rules for detailed steps on accessing this UI.
2. If you do not see log messages in the grid, upload a sample file by clicking **Upload Sample File**.
   If you uploaded a sample file in a previous step in the configuration process or you started your new source rule using an existing source rule, you will see log messages in the grid.
3. Hover over any row on the grid and click **+Select as Definer Sample**.
   Choose a sample log message that contains the fields you want to extract from the log file. When you need to capture fields from dissimilar log messages, you can create more than one definer.
4. After you select the Definer Sample message, use the following steps to extract fields from the message. After selecting the Definer Sample and specifying the fields to extract, the preview grid displays rows with matching and non-matching results.
   a. Select text in the message. If you are defining key-value pairs, it is only necessary to highlight the key field.
      A popup appears where you can specify properties of the extracted field.
   b. Enter the field name, select the data type, and click **Extract**.
      The preview grid is updated to show matching rows with the selected text highlighted with a color.
   c. (Optional) To update the properties or delete the field, click the field you just created in the definer and the properties popup appears.
   d. Repeat steps a, b and c to create more fields.
      As you add additional fields, the regular expression changes. You can view or copy (but not edit) the regular expression in the regular expression section.



5. (Optional) Specify Refiner Samples: As you review the non-matching rows, if you notice a value that was not extracted and should have been extracted, then you can add that row as a Refiner Sample. A refiner sample is used to match the unmatched rows from the Definer step.
   a. On the grid, click **+Select as Refiner Sample.** You do not see this option unless you already selected the Definer Sample.
   b. Select the text.
   c. A popup appears. You can associate this portion of text with a field already created in the Definer step. When you extract, the regular expression changes and matching rows get updated in the Preview grid and will be shown as Matching. If necessary, you can add more than one Refiner Sample message.
6. (Optional) Specify Counter Samples: Counter samples eliminate false positives while extracting the fields. In the Preview grid, the highlighted values are shown along with an X button. Click the X next to a field to mark a value as a false positive (a Counter Sample).



The Counter Sample is shown below the Definer and Refiner Samples and the regular expression is updated to show the new set of matching and non-matching rows.



7. Click **Save** to save the fields to the source rule.
8. Continue to the other tabs of the Add Source Rule dialog to complete your source rule configuration.

Manual Field Extraction

You can specify your own regular expressions if needed using Manual Field Extraction.

1. Using the Centralized Log Management UI, add a new source rule (or edit an existing source rule), and navigate to the Manual Field Extraction subtab within the Field Extraction tab. See "*Create a Source Rule*" in Configure Log Analytics Using Source Rules for detailed steps on accessing this UI.
2. If you do not see log messages in the grid, upload a sample file by clicking **Upload Sample File**.
3. Click **Add Field**.
4. Click **Show Example** to review how to write regular expressions.
5. Type in your regular expression and associate it with a field type using the drop-down menu.
6. In the Preview grid, click Refresh.
   The preview grid is highlighted with the results of applying the regular expression to the sample log messages. You cannot add more than 20 fields.
7. (Optional) You can change the regular expression or the type and use **Refresh** to see the updated results.
8. Click **Save** to update your source rule.

Agent Scopes for Configuration Log Management

Related pages:

- Configure Log Analytics Using Source Rules

Agent Scopes are used to determine which agents use the CLM source rules. You can group your deployed Analytics Agents into scopes, then assign source rules to the scopes. While creating an agent scope, you are given an option to either choose agents from a static list or to add agents dynamically by specifying matching rules.

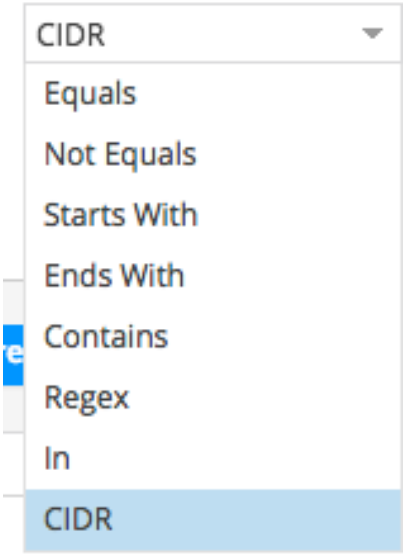
Static Agent Scopes

The Available Agents list is populated with the Analytics Agents reporting to your controller. You can use this list to add currently available agents to a scope.

Dynamic Agent Scopes

You can specify match conditions where registered Analytics Agents that match the rule are automatically added to, or excluded from, the scope. Analytics Agents that register with your Controller in the future, with characteristics that match the scope criteria, will also be added to the scope. You can match on agent name, unique host ID, or IP address.

The available operators for the matching conditions include regular expressions (regex) and Classless Inter-Domain Routing (CIDR) notation. CIDR is a compact notation for an IP address and its associated routing prefix.



The match conditions are additive, so exclusions will take precedence over inclusions. An agent must satisfy all conditions to be included. For example, an agent named *ABC* would not be included in the scope if you specify something similar to the following conditions:

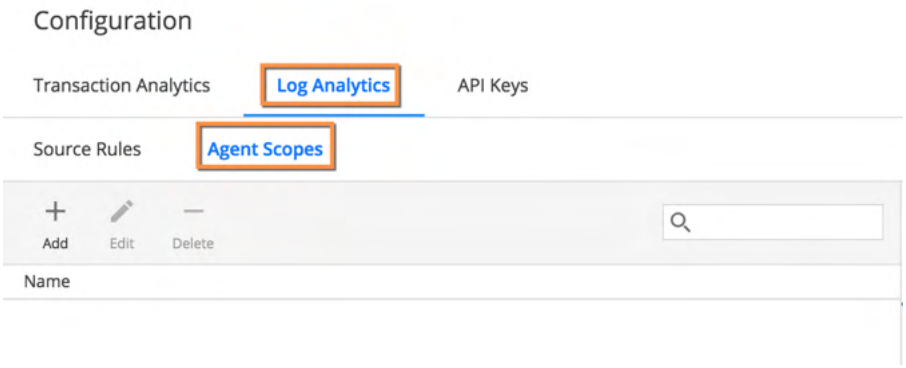
- include name equals '*ABC*'
- include name that ends with '*D*'

Define Agent Scopes

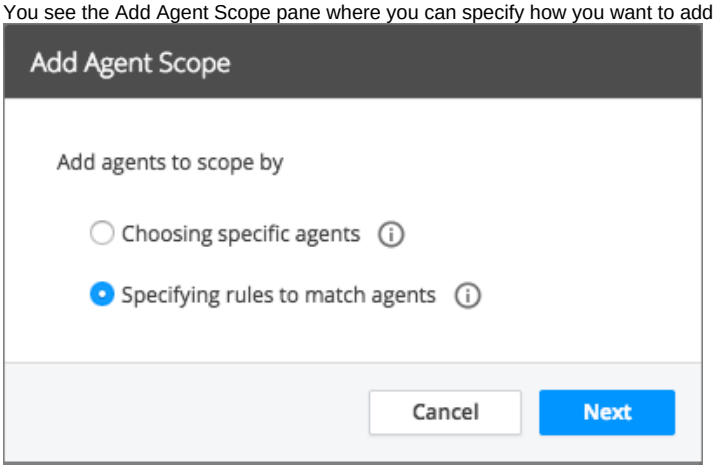
To add an Agent Scope:

1. From the Controller top navigation bar, click **Analytics**.

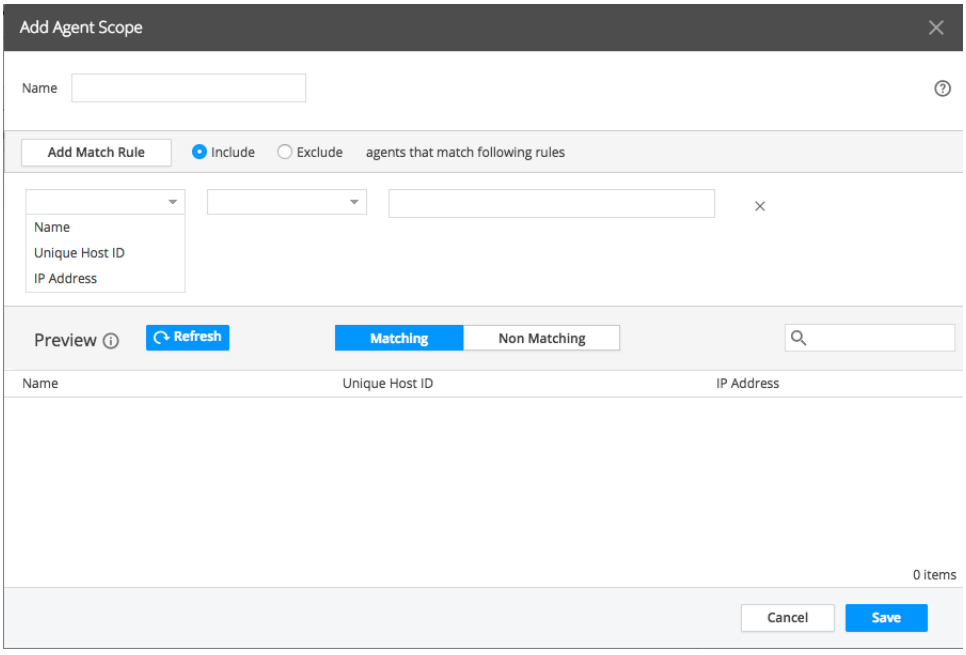
2. From the left navigation panel, click **Configuration > Log Analytics > Agent Scopes.**



3. **Click +Add.**
You see the Add Agent Scope pane where you can specify how you want to add agents: from a static list or dynamically by specifying match rules.



4. To add specific agents, select **Choosing specific agents** and move agents from the list of Available Agents to the list of Selected Agents to define the scope.
5. To add agents dynamically, select **Specifying rules to match agents** and specify your matching rules.



6. Name the **Agent Scope** and click **Save**.

Collect Log Analytics Data from Syslog Messages

Related pages:

- Configure Log Analytics Using Job Files
- Analytics Log Data

You can configure the Analytics Agent to receive syslog messages using TCP transport and standard syslog format adhering to RFC 5424. The Analytics Agent can listen on a TCP port for syslog messages coming from a remote source or from the same host on which the Analytics Agent is present.

To set this up:

- Configure your web server, such as Apache, to send logs to the Analytics Agent. There are two ways to direct the syslog data to the Analytics Agent:
  - Write the log messages to the usual log files and then read and forward to the agent from the log file. See Read syslog Messages From the Log File and Send to Analytics Agent.
  - Send the log data directly to the Analytics Agent without writing to a file first. See Send syslog Data Directly to the Analytics Agent.
- Configure the Analytics Agent to receive and parse the logs. There are two ways to configure syslog message collection for Log Analytics:
  - Job files (for <= 4.2), see Configure Log Analytics TCP Job File.
  - Source rules (for 4.3) Recommended for your new log file sources. See Configure Log Analytics TCP Source Rule.

Supported Environment Details

- Linux only
- Network protocol is TCP only
- A template for Apache Web Server syslog format is shipped with the Analytics Agent. You can bring any log data in through syslog over TCP as long as you provide the correct configuration to parse the log message.
- One job file per Analytics Agent can be enabled to receive syslog messages over TCP.

Both the syslog utility and analytics-agent should have root access if the port where the analytics-agent is listening for syslog messages is lower than 1024.

Send syslog Data Directly to the Analytics Agent

You can use the Linux Logger utility to direct your Apache access and error log messages to the syslog daemon directly. Using this method, the logs are not written to the usual access log and error log apache files or to the `/var/log/message` file. Instead, the log lines are sent directly to a syslog daemon which then forwards the data to the analytics-agent.

Configure:

1. Configure Apache to delegate logs to `/usr/bin/logger`.
   a. Locate and open `httpd.conf`, the Apache configuration file. This is typically located at `/etc/httpd/conf/`.
   b. Add a new CustomLog directive to send access logs and error logs to the syslog and comment out the line that sends to the access_logs.

```
#comment the following line to avoid logging to access_logs
#CustomLog logs/access_log combined

#Add a new CustomLog directive to send access logs and error logs to the syslog
CustomLog "|/usr/bin/logger -t httpd -p local6.info" combined
```

   This directive uses the logger utility to send messages with facility local6, tag 'httpd' and log format combined.
   The facility code specifies the type of program that is logging the message. Messages with different facilities may be handled differently.
   The tag 'httpd' in the httpd.conf directly relates to the program name in the rsyslog.conf (see next step) for filtering out which messages need to be sent. For example, there can be other programs writing to that particular port, but we only want to send the logs from the httpd program to analytics-agent.
2. Configure the rsyslog client.
   a. Edit `rsyslog.conf`, typically located at `/etc/`.
   b. Add the following lines above "RULES" or "var/log/messages" filter so Apache doesn't log to the `/var/log/message` file.

```
# log to analytics-agent
if $syslogfacility-text == 'local6' and $programname == 'httpd' then @@<analytics_agent_home>:514
# Prevent logging httpd to /var/log/messages
if $syslogfacility-text == 'local6' and $programname == 'httpd' then ~
```

   Make sure you replace 514 with the port where the analytics-agent is listening for syslog messages. This must match the port specified in the job file.
3. Restart apache and rsyslog, and look at /var/log/messages for any rsyslog errors.

Read syslog Messages From the Log File and Send to Analytics Agent

In this case, the Apache server writes logs to the usual access log and error log files, and the rsyslog daemon is configured to read from these log files and forward the log data to analytics agent. This method preserves the original log files. In this case, the access and the error log messages are logged in the `/var/log/message` file.

In this example, the rsyslog client is configured to read from a specific file and forward the message with facility local6 and severity level info over the specified port (default port is 514).

1. Locate and edit rsyslog.conf, typically located at `/etc/`.
2. In the begin forwarding section of the rsyslog.conf file, add the following lines:

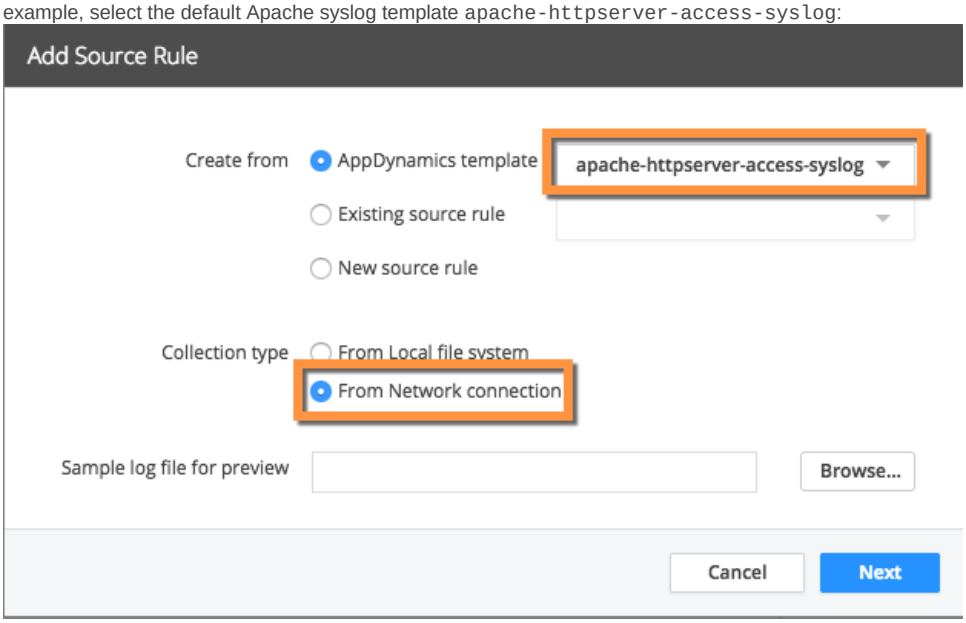
```
# add these lines in the begin forwarding section
$ModLoad imfile
$InputFileName /etc/httpd/logs/access_log << your file
$InputFileTag apache-access
$InputFileStateFile stat-apache-access
$InputFileSeverity info
$InputFileFacility local6
$InputRunFileMonitor
local6.info @@localhost:514
```

   $InputFileName: path to the log file you want to tail.
   local6.info: Use the Analytics Agent IP address if your analytics agent is not local to the controller. If needed, replace 514 with the port where the analytics-agent is listening for syslog messages. This must match the port specified in the job file or the source rule.
3. Restart rsyslog and review /var/log/messages for any rsyslog errors.

Configure Log Analytics TCP Source Rule

Using the Centralized Log Management UI, you can configure a source rule to extract log analytics fields from syslog messages over TCP.

1. Access the Centralized Log Management UI from your Controller by clicking **Analytics > Configuration > Log Analytics**.
2. On the Source Rules tab, click **+ Add.**
3. In the Add Source Rule panel, select the **Create from** source template and select **From Network Connection** as the collection type. For example, select the default Apache syslog template `apache-httpserver-access-syslog`:



   Several log format templates are shipped with the Analytics Agent. You can create a new source rule for any log format over syslog TCP as long as you configure it correctly.
4. Click **Next** to see the Add Source Configuration wizard.
5. Specify the collection details, such as the name of the source rule, source type and enter the TCP Port where the Analytics Agent is listening.
6. When you specify **From Network Connection** as the collection type, the grok pattern for the syslog header (which is appended to the log messages) is automatically added at the beginning of the grok Message Pattern:

```
%{SYSLOG5424PRI}%{SYSLOGBASE2}
```

7. Confirm that the value for Multiline Format is **None**.
8. Configure field extraction and field management as for any other source rule. See Configure Log Analytics Using Source Rules.

Configure Log Analytics TCP Job File

When selecting and configuring the port where the analytics agent will listen for the syslog data, make sure it does not conflict with anything else active in the network. If no port number is provided, port 514 is used. Both the syslog utility and analytics-agent should have root access to send logs to port 514 (binding to ports less than 1024 requires root access).

To allow the analytics-agent to listen at a port, specify the log file **source** property and associated parameters for **type**=syslog. For example, add the following to the appropriate job file:

```
source:
    type: syslog
    port: 514
    protocol: tcp
        numThreads: 1
```

A job file for apache commons is included in the analytics distribution at `/<analytics-agent-home>/conf/job/sample-apache-httpserver-access-syslog.job.`

Your job file should look similar to the following:

```
version: 2
enabled: true

source:
    type: syslog
    port: 514
    protocol: tcp
        numThreads: 5

fields:
   sourceType: apache-httpserver-access-syslog
   nodeName: Node1
   tierName: Tier1
   appName: App1

grok:
  patterns:
    - "%{SYSLOG5424PRI}%{SYSLOGBASE2} %{COMBINEDAPACHELOG}"

eventTimestamp:
   pattern: "dd/MMM/yyyy:HH:mm:ss Z"
```

Configure Log Analytics Using Job Files

Related pages:

- Analytics Log Data
- Configure Log Analytics Using Source Rules

This page describes how to configure log sources using job files. If you are configuring new log sources, we recommend you use the Centralized Log Management UI to define source rules.

To configure log analytics using job files:

1. Describe the Log Source in a Job File
2. Map the Log File Data to Analytics Fields
3. Verify Analytics Agent Properties

Describe the Log Source in a Job File

Each log source is represented by a job file. A job file is a configuration file that specifies the following:

- Location of the source log file
- Pattern for capturing records from the log file
- Pattern for structuring the data from the captured log records
- Other options for capturing records from the log source

To define a source, you create a job file (or modify one of the samples) in the Analytics Agent configuration directory. The Analytics Agent includes sample job files for Glassfish, OSX log, and others. The Analytics Agent can also collect and parse GZIP files - (log files ending in .gz).

The job files are located in the following directory:

- `<Analytics_Agent_Home>/conf/job/`

The agent reads the job files in the directory dynamically, so you can add job files in the directory without restarting the agent.

To configure a job file, use these configurable settings in the file:

- **enabled**: Determines whether this log source is active. To capture analytics data from this log source, set the value to true.
- **source**: Specifies the source of the logs.
    - **type**: Specifies the type of log source. There are two types, file, and syslog. Additional parameters depend on the value of **type**.
        - **file**: The location and name of the log file to serve as a log source. The location must be on the same machine as the analytics-agent. File has the following parameters:
            - **path**: Path to the directory where the log files reside. On Windows, the path should be provided as if on Unix environments such as:
            - Example: demo/logs
            - Example: C:/app/logs
            - **nameGlob**: A string to use to match on the log file name. You can use wild cards and you can specify whether to match files one level deep or all log files in the path directory structure. If the wild card starts the value of nameGlob, you must enclose the value in quotes. The matching patterns that are supported can be found here: http://java.boot.by/ocpjp7-upgrade/ch06s05.html under "glob".
            - Example for multi-level matching:
              path: /var/log
              nameGlob: '**/*.log'
              This matches both /var/log/apache2/logs/error.log and /var/log/cassandra/system.log
            - Example of one level matching:
              path: /var/log
              nameglob: '*/*.log'
              This searches for .log files one level deep in the /var/log directory (matches on /var/log/cassandra/system.log but not on /var/log/apache2/logs/error.log).
            - **startAtEnd**: If set to true allows tailing the file from the end.
        - **syslog**: See Collect Log Analytics Data from Syslog Messages.
    - **multiline**: For log files that include log records that span multiple lines (spanning multiple line breaks) configure the multiline property and indicate how the individual records in the log file should be identified. A typical example of a multiline log record is one that includes a Java exception. You can use one of two options with the multiline property to identify the lines for a multiline log record:
        - **startsWith**: A simple prefix that matches the start of the multiline log record.
          Example 1: To capture the following multiline log as one record:

          ```
          [#|2015-09-24T06:33:31.574-0700|INFO|glassfish3.1.2|com.appdynamics.METRICS.
          WRITE|_ThreadID=206;_ThreadName=Thread-2;|NODE PURGER Completed in 14 ms|#]
          ```

          You could use this:

          ```
          multiline:
             startsWith: "[#|"
          ```

Example 2: To capture the following multiline log as one record:

```
May 5, 2016 6:07:02 AM com.appdynamics.test.payments.plugin.testsource.testPaymentClient
deposit(pluginContext, financialTransaction, retry)
INFO: Source deposit requested amount:245.43000
May 5, 2016 6:07:02 AM com.appdynamics.test.payments.plugin.testsource.testPaymentClient
deposit(pluginContext, financialTransaction, retry)
INFO: Source deposit currency:USD
May 5, 2016 6:07:02 AM com.appdynamics.test.payments.plugin.testsource.testPaymentClient
deposit(pluginContext, financialTransaction, retry)
INFO: Source deposit payment method:Master Card
May 5, 2016 6:07:03 AM com.appdynamics.test.payments.plugin.testsource.testPaymentClient
deposit(pluginContext, financialTransaction, retry)
INFO: Source deposit decision:ACCEPT
May 5, 2016 6:07:03 AM com.appdynamics.test.payments.plugin.testsource.testPaymentClient
deposit(pluginContext, financialTransaction, retry)
INFO: Source deposit reason code:200
May 5, 2016 6:07:03 AM com.appdynamics.test.payments.plugin.testsource.testPaymentClient
deposit(pluginContext, financialTransaction, retry)
INFO: Source deposit orderId:7654
May 5, 2016 6:07:03 AM com.appdynamics.test.payments.plugin.testsource.testPaymentClient
deposit(pluginContext, financialTransaction, retry)
INFO: Source deposit piId:1234B
May 5, 2016 6:07:03 AM com.appdynamics.test.payments.plugin.testsource.testPaymentClient
deposit(pluginContext, financialTransaction, retry)
INFO: Source deposit referenceNumber:4620346703956702001
May 5, 2016 6:07:03 AM com.appdynamics.test.payments.plugin.testsource.testPaymentClient
deposit(pluginContext, financialTransaction, retry)
INFO: Source deposit extData requestToken:alf
/7wSR9PBh3zSMQ+++IYTRlk3Ys2DOxOmM5jWLRTyFJaSggCTyFJaSgjSB1/2v0wyTWRV0ekb0X
```

You could use this:

```
multiline:
   startsWith: "INFO: Source deposit requested amount"
```

- **regex**: A regular expression that matches the multiline log record.
  Example: To capture this multiline log as one record:

```
2016-06-01 16:28:21.035 WebContainer : 8 TRANSTART> ======================================
2016-06-01 16:28:21.036 WebContainer : 8 MERCHCFG > merchantID=appD, sendToProduction=true,
targetAPIVersion=1.00, keyFilename=(null), serverURL=(null), namespaceURI=(null),
enableLog=true, logDirectory=logs, logFilename=(null), logMaximumSize=10,
useHttpClient=false, timeout=130
2016-06-01 16:28:21.037 WebContainer : 8 REQUEST  >
merchantID=appD
davService_run=true
clientLibraryVersion=2.0.1
clientEnvironment=OS/400/V7R1M0/LINUX
application_id=ff22
exportService_addressWeight=medium
merchantReferenceCode=appD
clientLibrary=Java Basic
billTo_customer=xxxxxxxxx
billTo_finance=true
billTo_postalCode=94105
billTo_country=US
```

You can use the following regex configuration to identify the start of each "record":

```
multiline:
   regex: "^\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}\.\d{3} WebContainer : \d+ TRANSTART\u003E
======================================$"
```

The regex describes that the line should match 4 digits(2016) followed by followed by 2 digits (01), followed by a space, followed by 2 digit(16) ':' 2 digit (28) ':' 2 digits(21) '.' 3 digits(035) space followed by term 'WebContainer' space digit followed by term 'TRANSTART' folowed by '>'.

Whenever the Log tailer sees the matching pattern at the start of the line, it starts a new match and passes the previously collected data as one log record. Each matching pattern identifies when to start accumulating the log lines in a buffer until the matching pattern is found again at the beginning of another record.

If the particular format of a multiline log file does not permit reliable continuation line matching by regular expression, you may choose to use a single line format. For most types of logs, this would result in the capture of the majority of log records.

- **fields**: The fields are used to specify the context of the log data in the Controller UI, by application name, tier name, and so on. Specify the fields as free form, key-value pairs.
- **grok**: The grok parameter specifies the patterns by which the data in the unstructured log record is mapped to structured analytics fields. It associates a named grok expression (as defined in a .grok file that is bundled inside lib/analytics-shared-pipeline-core.jar) to a field in the data as structured by the agent. For example:

```
grok:
  patterns:
       - "\\[%{LOGLEVEL:logLevel}%{SPACE}\\]  \\[%{DATA:threadName}\\]  \\[%{JAVACLASS:class}\\]  %
{GREEDYDATA}"
       - "pattern 2"
       ...
```

In this case, the grok-pattern name LOGLEVEL is matched to an analytics data field named logLevel. The regular expression that is specified by the name LOGLEVEL is defined in the file `grok-patterns.grok` in the `grok` directory. See Specifying Grok Expressions.
Previous versions of Log Analytics used a single "pattern" rather than a pattern list. This mode is still supported for backward compatibility.
The log analytics grok processor does not allow underscores in the field names. For example, a field name "log_fields" will not work, Instead, use something like "logfields".

- **keyValue**: The keyValue parameter specifies how to parse the logs to identify key-value pairs with a user-defined delimiter. This enables you to configure the parsing for a message of the type "Key1 = Value1 Key2 = Value 2". See Specifying Key Value Pairs.
- **transform**: This parameter specifies how to change the type or alias name of any field extracted from the logs by grok or keyValue parameters.
- **eventTimestamp**: This setting defines the pattern for the timestamp associated with captured data.

Map the Log File Data to Analytics Fields

To specify how data in the unstructured log records should be mapped to structured analytics fields for log analytics, you provide the configuration in the job file. You can map unstructured log data in the following ways:

- grok patterns
- key value pairs
- transforms

Specify Grok Expressions

Grok is a way to define and use complex, nested regular expressions in an easy to read and use format. Regular expressions defining discrete elements in a log file are mapped to grok-pattern names, which can also be used to create more complex patterns.

Grok-pattern names for many of the common types of data found in logs are provided for you with the analytics agent. A list of basic grok-pattern names and their underlying structures are bundled inside `lib/analytics-shared-pipeline-core.jar`. You can list the grok files with a command such as the following:

```
unzip -l ./lib/analytics-shared-pipeline-core.jar|grep "\.grok"
```

To can view the definition of a grok file with a command such as the following:

```
unzip -p ./lib/analytics-shared-pipeline-core.jar grok/grok-patterns.grok
```

The `grok` directory also contains samples of more complex definitions customized for various common log types - `java.grok, mongodb.grok`, and so on. Additional grok patterns can be found here: https://grokdebug.herokuapp.com/patterns#.

Once the grok-pattern names are created, they are then associated in the jobs file with field identifiers that become the analytics keys.

The basic building block is `%{grok-pattern name:identifier}`, where `grok-pattern name` is the grok pattern that knows about the type of data in the log you want to fetch (based on a regex definition) and `identifier` is your identifier for the kind of data, which becomes the analytics key. So `%{IP:client}` would select an IP address in the log record and map it to the key `client.`

Custom Grok Patterns

Complex grok patterns can be created using nested basic patterns. For example, from the `mongodb.grok` file:

```
MONGO_LOG %{SYSLOGTIMESTAMP:timestamp} \[%{WORD:component}\] %{GREEDYDATA}
```

It is also possible to create entirely new patterns using regular expressions. For example, the following line from `java.grok` defines a grok pattern named JAVACLASS.

```
JAVACLASS (?:[a-zA-Z$_][a-zA-Z$_0-9]*\.)*[a-zA-Z$_][a-zA-Z$_0-9]*
```

Because JAVACLASS is defined in a .grok file in the `grok` directory can be used as if it were a basic grok pattern. In a jobs file, you can use the JAVACLASS pattern match as follows:

```
grok:
  pattern: ".... \[%{JAVACLASS:class}\\]
```

In this case, the field name as it appears in the Analytics UI would be "class". For a full example, see the following files:

- Job file: `<Analytics_Agent_Home>/conf/job/sample-analytics-log.job`
- Grok file: `java.grok` bundled inside `lib/analytics-shared-pipeline-core.jar`

Special Considerations for Backslashes

The job file is in YAML format, which treats the backslash as an escape character. Therefore, to include a literal backslash in the String pattern you need to escape the backslash with a second backslash. You can avoid the need to escape backslashes in the .job file grok pattern, by enclosing the grok pattern in single quotes instead of double quotes such as the following:

```
grok:
  patterns:
    - '\[%{DATESTAMP:TIME}%{SPACE}CET\]%{SPACE}%{NOTSPACE:appId}%{SPACE}%{NOTSPACE:appName}%{SPACE}%{NOTSPACE:
Severity}%{SPACE}%{NOTSPACE:messageId}:%{SPACE}%{GREEDYDATA}'
```

Numeric Fields

In Release 4.1.3, the grok definition syntax was enhanced to support three basic data types. When defining a pattern in the .grok file you can specify the data type as number, boolean, or string. If a Grok alias uses that grok definition in a .job file then the extracted field is stored as a number or boolean. Strings are the default. If the number or boolean conversion fails, then a log message appears in the agent's log file. No validations are performed as it is not possible to reverse engineer a regex reliably. These are pure runtime extractions and conversions.

Upgrade pre-4.1.3 Job Files

For 4.1.2 (or older) .job files in use that have fields that are unspecified or specified as NUMBER and now switch to the "type aware" files, the data inside Events Service will break. This is due to the type mapping. To avoid this, you need to modify the grok alias in your job files.
Examples:

```
Was:
grok:
  patterns:
    - "%{DATE:happenedAt},%{NUMBER:quantity}

Update job to:
grok:
  patterns:
    - "%{DATE:happenedAt},%{NUMBER:quantity_new}
```

```
Was:
grok:
  patterns:
    - "%{DATE:happenedAt},%{DATA:howMany}

Update job to:
grok:
  patterns:
    - "%{DATE:happenedAt},%{POSINT:howManyInt}
```

To Upgrade (Migrate) < 4.1.3 Job Files:

1. Stop analytics-agent.
2. Change .job files that use the enhanced grok patterns:

```
BOOL:boolean
INT:number
BASE10NUM:number
NUMBER:number
POSINT:number
NONNEGINT:number
```

   Change the grok alias so as not to conflict with the older aliases:

```
grok:
  patterns:
(Old)  - "%{DATE:quoteDate},%{NUMBER:open},%{NUMBER:high},%{NUMBER:low},%{NUMBER:close},%{NUMBER:
volume},%{NUMBER:adjClose}"

(New aliases)  - "%{DATE:quoteDate},%{NUMBER:openNum},%{NUMBER:highNum},%{NUMBER:lowNum},%{NUMBER:
closeNum},%{NUMBER:volumeNum},%{NUMBER:adjCloseNum}"
```

3. Start analytics-agent.

Specifying Key-Value Pairs

This section of the mapping configuration captures key-value pairs from fields specified by the `source` parameter. The values listed under source should refer to fields that were defined and captured by a grok pattern. For example, if you have a grok parameter that defines the following pattern "%{DATA:keyValuePairs}" then you can list the field "keyValuePairs" under the source parameter to capture any key-value pairs listed in the "keyValuePairs" string. If the `source` parameter is not specified then the agent attempts to extract key-value pairs from the entire log message. The result can be different than expected if the message contains more information than just key-value pairs.

The Key Value mapping contains these fields:

- **source**: A list of strings on which the keyValue filter should be applied. It is an optional field. If it is not provided the key-value pairs are parsed from the original log "message" string.
- **split**: The delimiter defined by the user to separate out the key from the value. In this example, key=value, the split delimiter between the key and the value is the equal sign =
- **separator**: The delimiter defined by the user to separate out two key-value pairs. In this example, key1=value1;key2=value2, the separator is the semi-colon ;
- **include**: A list of key names that the user wants to capture from the "source". If nothing is provided in "include" we capture all the key-value pairs.
- **trim**: A list of characters the user wants to remove from the starting and/or the end of the key/value before storing them.

The sample-glassfish-log.job file includes key-value pairs configuration. This file is found here: <*analytics_agent_home*>`/conf/job/`.

Key-Value Pairs Example

For a log file with the following entries:

> [#I2015-09-24T06:33:31:574-0700|INFO|glassfish3.1.2|com.appdynamics,METRICS.WRITE|_**ThreadID=200;_ThreadName=Thread-2;**|NODE PURGER Complete in 14 ms|#]
>
> [#I2015-09-24T06:33:46:541-0700|INFO|glassfish3.1.2|com.singularity.ee.controller.beans.license.LicenseUsageManagerBeanWRITE|_**ThreadID=202;_ThreadName=Thread-2**;|about to start persisting license usage data |#]

And the following grok pattern:

```
grok:
  patterns:
    - "\\[\\#\\|%{DATA}\\|%{LOGLEVEL:logLevel}\\|%{DATA:serverVersion}\\|%{JAVACLASS:class}\\|%{DATA:
keyValuePairs}\\|%{GREEDYDATA}"
```

The key-value parameter to extract the ThreadID and the ThreadName should look similar to the following:

```
keyValue:
  source:
        -"keyValuePairs"
  split: "="
  separator: ";"
  include:
        - "ThreadID"
        - "ThreadName"
  trim:
        - "_"
```

Specify Transform Parameters

This section of the mapping configuration enables you to change the type or alias name of any field previously extracted from the logs by your grok or key value configuration. The transform is applied after all fields have been captured from the log message. You can specify a list of field names, where you want to cast the value to a specific type or rename the field with an "alias".

The Transform mapping contains these fields:

- **field**: Specifies the name of the field to transform and can not be empty or null. If `field` is defined, either type or alias must be specified. If neither is specified, an error is written to the analytics-agent.log file.
- **alias**: The new name for the field.
- **type**: The value type for the field. Valid values are:
    - NUMBER
    - BOOLEAN
    - STRING - default is STRING

Verify Analytics Agent Properties

In addition to configuring the log source in the job file, you should verify the values in the `analytics-agent.properties` file found in the `conf` director y. Confirm these property values:

- `http.event.endpoint` should be the location of the Events Service.
    - For SaaS controllers the URL is one of the following:
        - `https://analytics.api.appdynamics.com:443` (North America)
        - `https://fra-ana-api.saas.appdynamics.com:443` (Europe)
        - `https://syd-ana-api.saas.appdynamics.com:443` (APAC)
    - For on-premises installations use whatever host and port you have configured. In clustered environments, this is often a load balancer.
- The `http.event.accountName` and `http.event.accessKey` settings should be set to the name and the key of the account in the Controller UI with which the logs should be associated. By default, they are set to the built-in account for a single tenancy Controller.
- The `pipeline.poll.dir` setting specifies where the log configuration files are located. This would not normally be changed unless you want to keep your files in a different location.
- `ad.controller.url` should match your AppDynamics controller URL and port.

Troubleshoot Logs

If log capture is working correctly, logs should start appearing in the Log tab in the Analytics UI. It can take some time for logs to start accumulating. Note the following troubleshooting points:

- If nothing appears in the log view, try searching over the past 24 hours.
- Timezone discrepancies between the logs and the local machine can cause log entries to be incorrectly excluded based on the selected timeframe in the Controller UI. To remediate, try setting the log files and system time to UTC or logging the timezone with the log message to verify.
- An inherent delay in indexing may result in the "last minute" view in the UI consistently yielding no logs. Increase the time range if you encounter this issue.

Troubleshoot Patterns

To help you troubleshoot the data extraction patterns in your job file, you can use the two debug REST endpoints in the Analytics Agent:

- http://<Analytics_Agent_host>:<Analytics_Agent_http_port>/debug/grok: For testing grok patterns
- http://<Analytics_Agent_host>:<Analytics_Agent_http_port>/debug/timestamp: For testing timestamp patterns

In the following examples, the Analytics Agent host is assumed to be `localhost` and the Analytics Agent port is assumed to be `9090`. To configure the port on your Agent, use the property `ad.dw.http.port` in `<Analytics_Agent_Home>/conf/analytics-agent.properties`.

The Grok Endpoint

The Grok tool works in two modes: extraction from a single line log and extraction from a multi-line log. To get a description of usage options:

```
curl -X GET http://localhost:9090/debug/grok
```

Single Line

In this mode, you pass in (as a POST request) a sample line from your log and the grok pattern you are testing, and you receive back the data you passed in organized as key/value pairs, where the keys are your identifiers.

```
curl -X POST http://localhost:9090/debug/grok --data-urlencode "logLine=LOG_LINE" --data-urlencode
"pattern=PATTERN"
```

For example, the input:

```
curl -X POST http://localhost:9090/debug/grok --data-urlencode "logLine=[2014-09-04T15:22:41,594Z]  [INFO ]
[main]  [o.e.j.server.handler.ContextHandler]  Started i.d.j.MutableServletContextHandler@2b3b527{/,null,
AVAILABLE}" --data-urlencode "pattern=\\[%{LOGLEVEL:logLevel}%{SPACE}\\]  \\[%{DATA:threadName}\\]  \\[%
{JAVACLASS:class}\\]  %{GREEDYDATA}"
```

would produce this output:

```
{
 threadName => main
 logLevel => INFO
 class => o.e.j.server.handler.ContextHandler
}
```

The input:

```
curl -X POST http://localhost:9090/debug/grok --data-urlencode "logLine=2010-05-
05,500.98,515.72,500.47,509.76,4566900,509.76" --data-urlencode "pattern=%{DATE:quoteDate},%{DATA:open},%
{DATA:high},%{DATA:low},%{DATA:close},%{DATA:volume},%{GREEDYDATA:adjClose}"
```

would produce this output:

```
{
 open => 500.98
 adjClose => 509.76
 volume => 4566900
 quoteDate => 10-05-05
 high => 515.72
 low => 500.47
 close => 509.76
}
```

Multi-line

The multi-line version uses a file stored on the local filesystem as the source input.

```
curl -X POST http://localhost:9090/debug/grok --data-urlencode "logLine=`cat FILE_NAME`" --data-urlencode
"pattern=PATTERN"
```

where FILE_NAME is the full path filename of the file that contains the multi-line log.

The Timestamp Endpoint

The timestamp tool extracts the timestamp from a log line in Unix epoch time.

To get a description of usage options:

```
curl -X GET http://localhost:9090/debug/timestamp
```

In this mode, you pass in (as a POST request) a sample line from your log and the timestamp pattern you are testing, and you receive back the timestamp contained within the log line.

```
curl -X POST http://localhost:9090/debug/timestamp --data-urlencode "logLine=LOG_LINE" --data-urlencode
"pattern=PATTERN"
```

For example, the input:

```
curl -X POST http://localhost:9090/debug/timestamp --data-urlencode "logLine=[2014-09-04T15:22:41,237Z]
[INFO ]  [main]  [io.dropwizard.server.ServerFactory]  Starting DemoMain" --data-urlencode "pattern=yyyy-MM-
dd'T'HH:mm:ss,SSSZ"
```

would produce this output Unix epoch time:

```
{
 eventTimestamp => 1409844161237
}
```

The input:

```
curl -X POST http://localhost:9090/debug/timestamp --data-urlencode "logLine=Nov 17, 2014 8:21:51 AM com.foo.
blitz.processor.core.hbase.coprocessor.endpoint.TimeRollupProcessEndpoint$HBaseDataFetcher callFoo1" --data-
urlencode "pattern=MMM d, yyyy h:mm:ss aa"
```

would produce this output Unix epoch time:

```
{
 eventTimestamp => 1416212511000
}
```

Sample Log Analytics Job Files

A number of sample job files are shipped with the Analytics Agent. These files can be found at `<analytics-agent-home>/conf/job`.

The sample job files include:

- Analytics logs: sample-analytics-log.job
- Analytics logs with requestGUID config example: sample-analytics-log-with-request-guid.job
- Apache access logs: sample-apache-httpserver-access-log.job
- Apache access logs with syslog config example: sample-apache-httpserver-access-sysloglog.job
- Apache error logs: sample-apache-httpserver-error-log.job
- Cassandra logs: sample-cassandra-log.job
- CouchDB logs: sample-couchdb-log.job
- Glassfish logs: sample-glassfish-log.job
- IIS logs: sample-iis-log.job
- Java Agent logs: sample-java-agent-log.job
- Jetty error logs: sample-jetty-error-log.job
- Jetty request logs: sample-jetty-request-log.job
- Log4J: sample-log4j.job
- MongoDB logs: sample-mongodb-log.job
- MySQL error logs: sample-mysql-error-log.job
- Nginx access logs: sample-nginx-access-log.job
- Nginx error logs: sample-nginx-error-log.job
- OS X system logs: sample-osx-system-log.job
- Postgres logs: sample-postgres-log.job
- Redis logs: sample-redis-log.job
- Stock quotes: sample-stock-quotes-csv.job
- WebLogic logs: sample-weblogic-log.job

Migrate Log Analytics Job Files to Source Rules

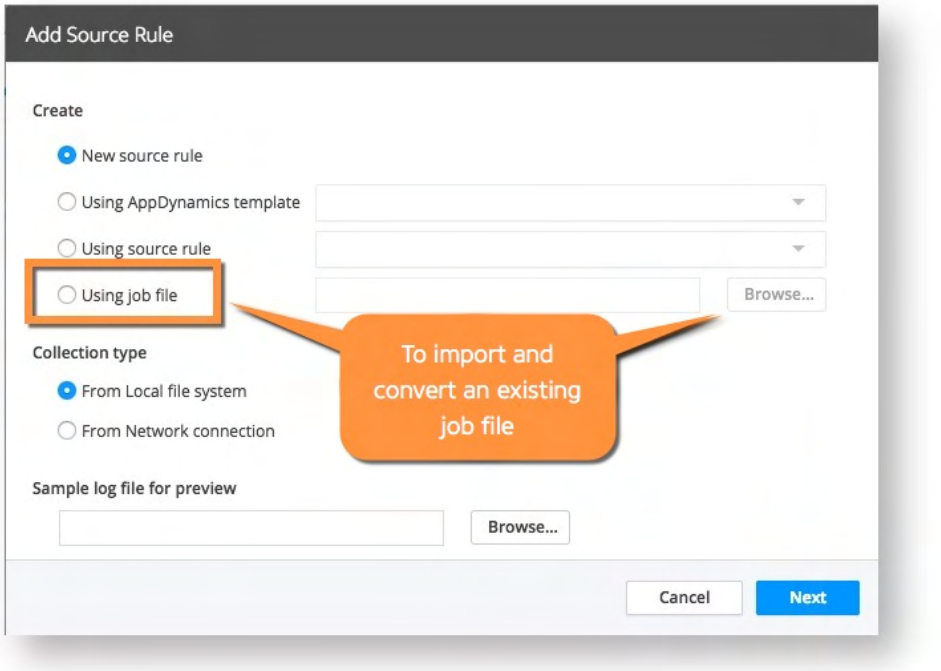
Related pages:

- Upgrade Analytics Agent

The Centralized Log Management UI enables you to configure your log data sources using source rules. Manually created job files are no longer needed.

We recommend that you convert your log source configuration from job files (job files were used for configuration < 4.3) to source rules.

1. Confirm that you have upgraded your Controller, Events Service, and Analytics Agent to 4.4. See these corresponding upgrade pages:
    - Upgrade the Controller Using the Enterprise Console
    - Upgrade the Events Service Using the Enterprise Console
    - Upgrade Analytics Agent
2. Use the Centralized Log Management UI to create source rules for your existing job files by importing the configuration from your old job file to a new source rule.



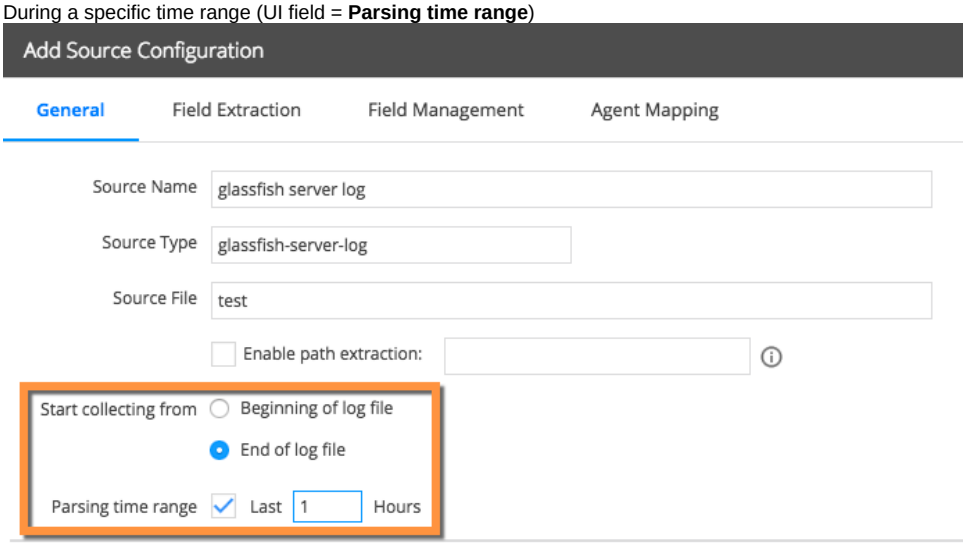
   Be sure to use either '**Parsing time range**' or '**End of log file**'. If you do not use this, you will double collect all the log files that have already been tailed by the job file. Read Timing Notes for more information to help you decide on your collection timing settings.
   When you save the source rules, they are in the disabled state by default.
3. Map the source rules to an Agent Scope.
4. Enable the new source rules through the UI.
5. Disable the job files that are actively collecting log analytics data. You do this by manually editing the job file and changing the `enabled` property to false. See Configure Log Analytics Using Job Files.
6. After you have migrated all your job file configurations to source rules, you can completely disable the use of job files by clicking "**Disable Field Extraction With Job Files**". This action can not be reversed and you will not be able to use job files after performing this action. Only trigger this action once you have completely moved to source rules created through the Centralized Log Management UI.

Timing Notes

By default, the controller communicates new configuration information to the Analytics Agent every five minutes. So it could be up to five minutes before the agent starts tailing your log file using the new source rule configuration. On the other hand, if you have both a job file *and* a source rule enabled for the same log data, the data is collected twice. To avoid this situation, you can use configuration settings to specify when to start collecting the log data:

- At the end of the file (UI field = **Start collecting from End of log file**)

- During a specific time range (UI field = **Parsing time range**)



See the section describing the configuration settings on the "*General Configuration Tab*" in Configure Log Analytics Using Source Rules.

Be sure to disable the job file once you have enabled the source rule to collect the Log Analytics data. You can verify that the new source rule is collecting data correctly by waiting until the new log records appear in the Analytics Search UI data grid. One way to distinguish between the log data collected by a job file or a source rule is to use a different source type in the source rule.

Business Transaction and Log Correlation

Works with:

This page provides instructions on the correlation between Business Transaction requests and logs.

When investigating the cause of slowdowns/outages in your business applications, the problem does not always originate in the application code. Any additional information from application or machine logs can be helpful in your DevOps teams' investigation. One way to see supporting data that impacts the business transaction is to analyze logs for that transaction.

By configuring business transactions for GUID Injection, you can correlate logs to specific business transaction requests. This can be helpful when you see slow transactions and the call graph does not give you enough information to get to the root cause. You can use this feature to get the full context related to a failed or slow transaction.

Correlating specific instances of your business transactions to the related logs works by injecting the same requestGUID of the business transaction into the associated logs through our Java Agent. This helps you to quickly find the relevant logs from multiple tiers and nodes for a specific business transaction.

Visibility for this correlation requires both Transaction Analytics and Log Analytics licenses. See License Entitlements and Restrictions.

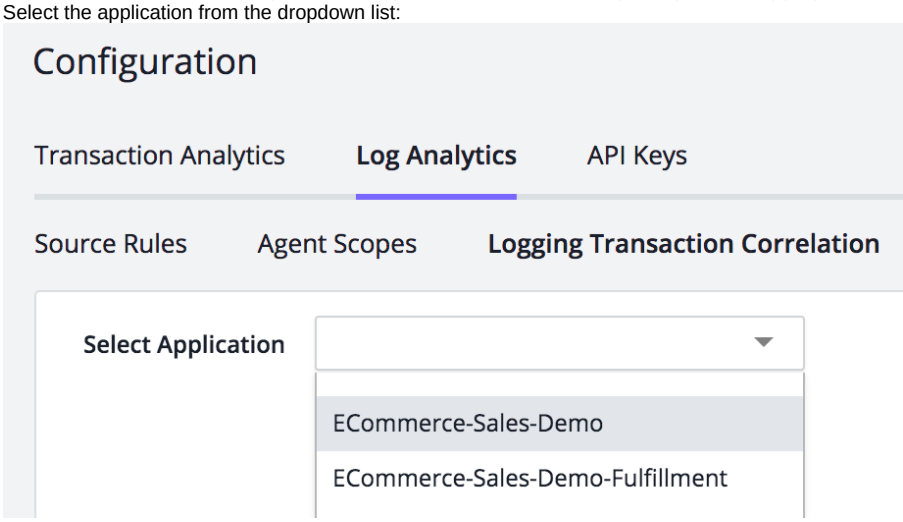
Configuring GUID Injection

This feature supports the following Java logging frameworks:

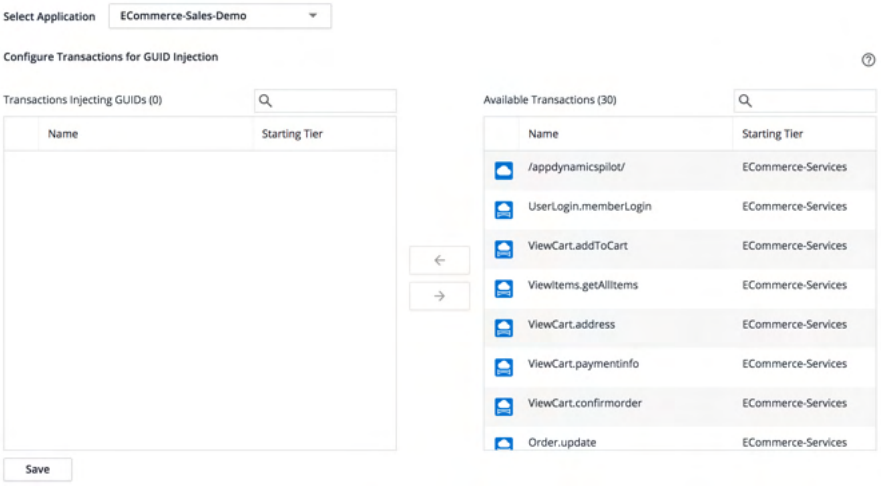
- Apache log4j versions 1.2.12 to 1.2.17 and 2.6.2 to 2.13.2.
- logback versions 1.0.0 to 1.1.3. Any version before 1.0.0 is not supported.

To enable transaction to log correlation, use the following steps. You must select the business transactions and specify the logging format.

1. Select business transactions for log correlation. This is how you specify to the Java Agent which business transactions you are interested in.
    a. In the Controller UI, click **Analytics** > **Configuration**. Then select **Log Analytics** > **Logging Transaction Correlation**.
    b. Select the application from the dropdown list:

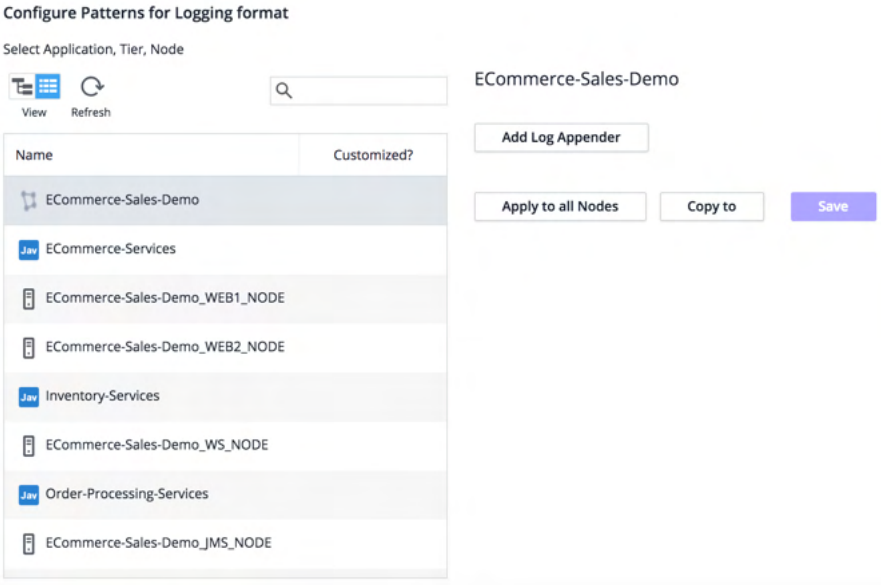


c. Scroll down to the section **Configure Transactions for GUID Injection**:

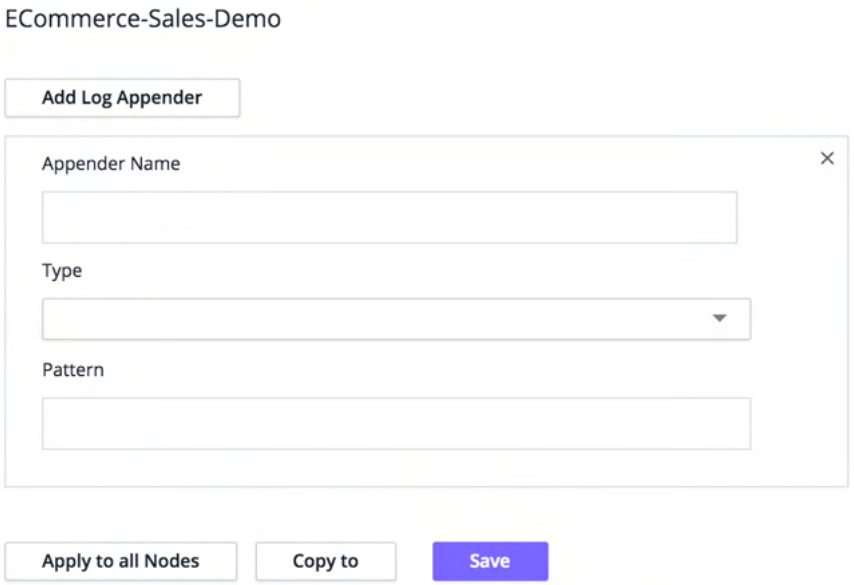


d. Add transactions from the right-hand list to the left-hand list and **Save**.
e. Proceed to the next step. You must also configure the logging patterns before you can see correlated logs.

2. Define how the injected information appears in the logs. You need to know the appender name and pattern for your application logging framework so you can properly configure this feature. In simple terms, a pattern is responsible for formatting a logging request and an appender takes care of output destination. You can configure this through the Controller UI as described here. You can also add the appender directly to your code. See Add Appender in Your Source Code.
   a. On the Analytics Configuration window, scroll to the section **Configure Patterns for Logging format.**
   b. Select application, tier, or node where you want to collect correlated logs.



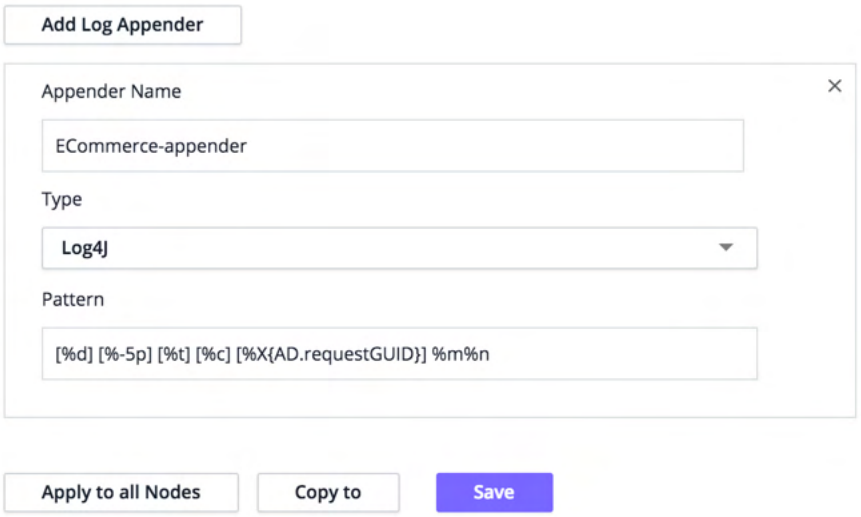
c. Enter the Appender name and choose the log framework.



d. Enter the pattern and the request GUID string. You can add the request GUID anyplace in the pattern. The request GUID must match the following exactly:

```
[%X{AD.requestGUID}]
```

For example, the following screenshot shows the standard log4j pattern plus the request GUID:



Detail of Log4j example with the request GUID string:

```
[%d]  [%-5p]  [%t]  [%c]  [%X{AD.requestGUID}]  %m%n
```

3. Restart the affected application to enable the logger to pick up the new logging configuration.

Add Appender in Your Source Code

If you have access to the source code for your application, you can also add the appender to the log4j.properties file directly. Here is an example of what that might look like with the request GUID in **BOLD**:

log4j.appender.order-file-appender=org.apache.log4j.FileAppender

log4j.appender.order-file-appender.File=logs/telecom-order.log

log4j.appender.order-file-appender.layout=org.apache.log4j.PatternLayout

log4j.appender.order-file-appender.layout.ConversionPattern=[%d] [%-5p] [%t] [%c] **[%X{AD.requestGUID}]** %m%n

log4j.logger.com.appdynamics.order=DEBUG, order-file-appender

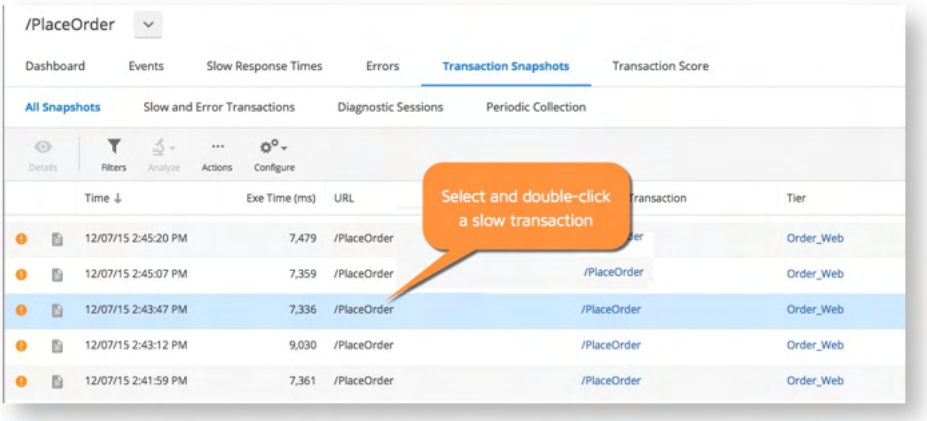
log4j.additivity.com.appdynamics.order=false

Viewing Correlated Logs and Transaction Data

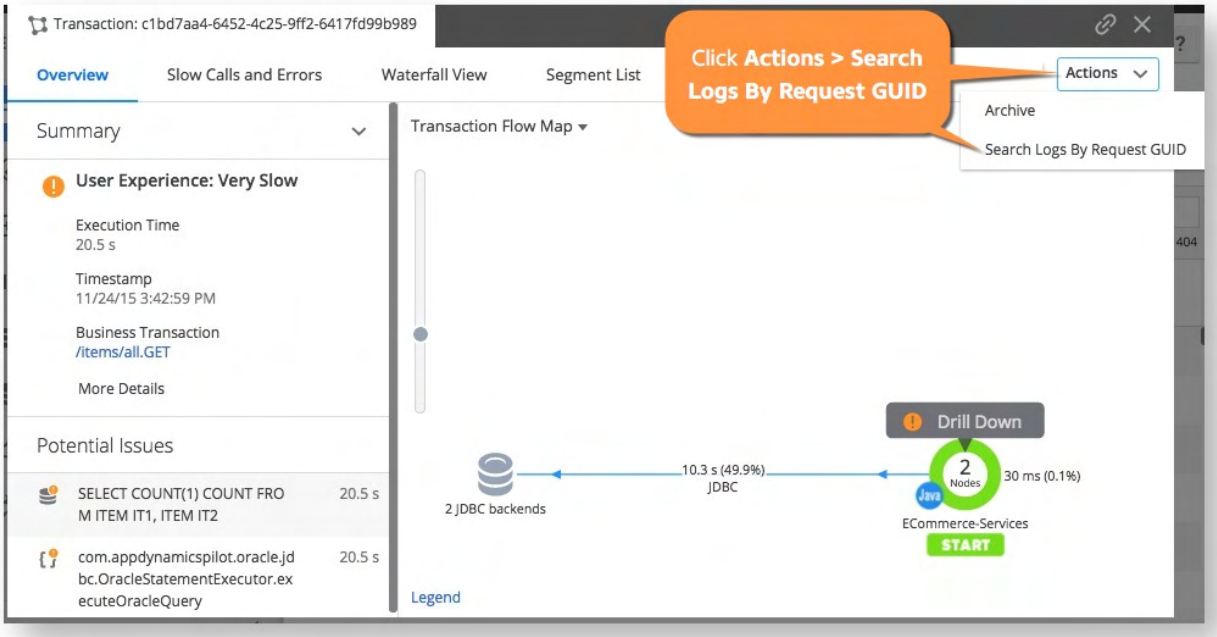
Once you have configured GUID injection, you can search the log files from a number of points in the Controller. Correlation only works when there are logs with the associated GUID in the given time range for an application where this feature is configured. The **Search Logs By Request GUID** button appears when there are logs for the snapshot request GUID in the snapshot time range.

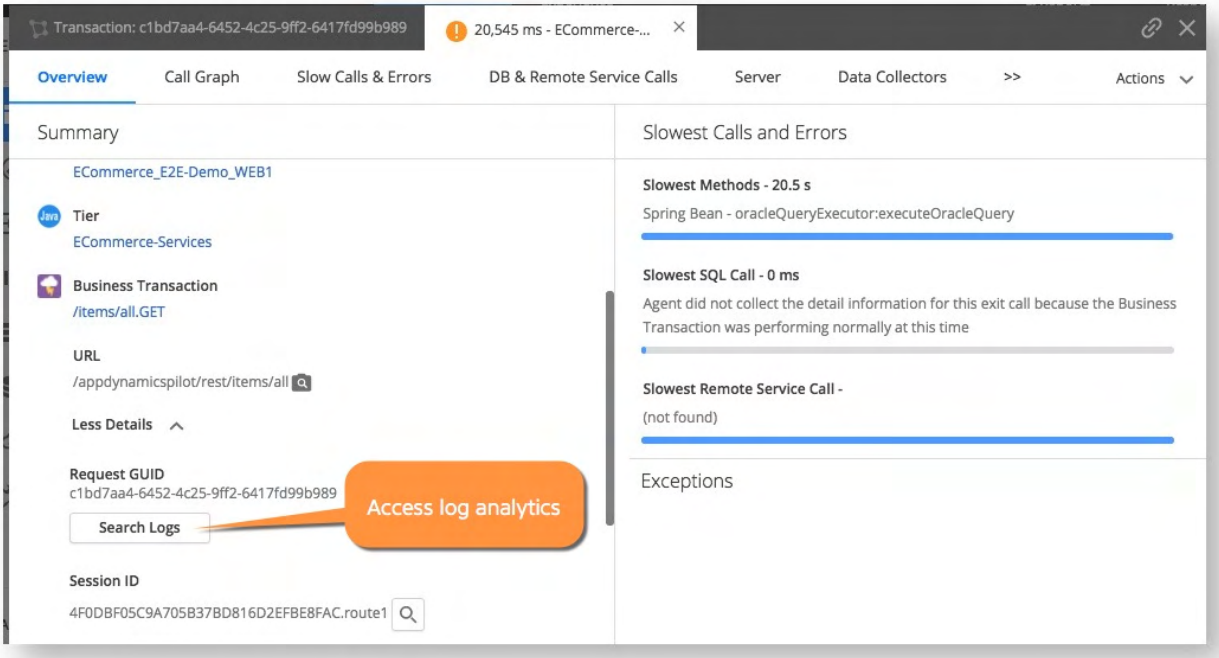
APM to Analytics: From a business transaction snapshot

1. Locate a slow transaction or other transaction that you want to troubleshoot.



2. Click **More Details** > **Search Logs** or use **Actions** > **Search Logs By Request GUID**.





From the log details in this example, you can see that the reason this transaction was slow is that the order processing queue was full and it took many retries before the order could complete.

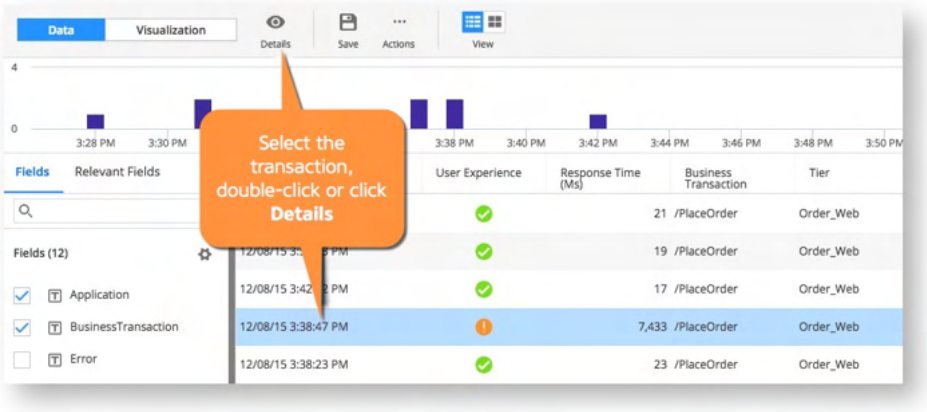
12/07/15 2:43:56 PM	[2015-12-07 22:43:54,593] [INFO] [org.springframework.jms.listener.DefaultMessageListenerContainer#0-1] [com.appdynamics.provision.OrderMessageListener] [AD_REQUEST_GUID[8f34fb97-b5fa-4f80-93a9-b56e0c75d222]] Received a message to process the order Order_506932 for the user timmy@lamelectrics.com	8f34fb97-b5fa-4f80-93a9-b56e0c75d222
12/07/15 2:43:56 PM	[2015-12-07 22:43:54,585] [INFO] [qtp1218204792-40] [com.appdynamics.order.OrderService] [AD_REQUEST_GUID[8f34fb97-b5fa-4f80-93a9-b56e0c75d222]] Sent the message to order processing queue, order Order_506932 and user timmy@lamelectrics.com	8f34fb97-b5fa-4f80-93a9-b56e0c75d222
12/07/15 2:43:54 PM	[2015-12-07 22:43:54,583] [WARN] [qtp1218204792-40] [com.appdynamics.order.OrderService] [AD_REQUEST_GUID[8f34fb97-b5fa-4f80-93a9-b56e0c75d222]] Order processing queue is full. Will retry [4] more times with [1] second interval before dropping the order	8f34fb97-b5fa-4f80-93a9-b56e0c75d222
12/07/15 2:43:54 PM	[2015-12-07 22:43:53,583] [WARN] [qtp1218204792-40] [com.appdynamics.order.OrderService] [AD_REQUEST_GUID[8f34fb97-b5fa-4f80-93a9-b56e0c75d222]] Order processing queue is full. Will retry [5] more times with [1] second interval before dropping the order	8f34fb97-b5fa-4f80-93a9-b56e0c75d222
12/07/15 2:43:53 PM	[2015-12-07 22:43:52,583] [WARN] [qtp1218204792-40] [com.appdynamics.order.OrderService] [AD_REQUEST_GUID[8f34fb97-b5fa-4f80-93a9-b56e0c75d222]] Order processing queue is full. Will retry [6] more times with [1] second interval before dropping the order	8f34fb97-b5fa-4f80-93a9-b56e0c75d222
12/07/15 2:43:52 PM	[2015-12-07 22:43:51,582] [WARN] [qtp1218204792-40] [com.appdynamics.order.OrderService] [AD_REQUEST_GUID[8f34fb97-b5fa-4f80-93a9-b56e0c75d222]] Order processing queue is full. Will retry [7] more times with [1] second interval before dropping the order	8f34fb97-b5fa-4f80-93a9-b56e0c75d222
12/07/15 2:43:51 PM	[2015-12-07 22:43:50,582] [WARN] [qtp1218204792-40] [com.appdynamics.order.OrderService] [AD_REQUEST_GUID[8f34fb97-b5fa-4f80-93a9-b56e0c75d222]] Order processing queue is full. Will retry [8] more times with [1] second interval before dropping the order	8f34fb97-b5fa-4f80-93a9-b56e0c75d222

1 / 1 100 items per page 1 - 12 of 12 items

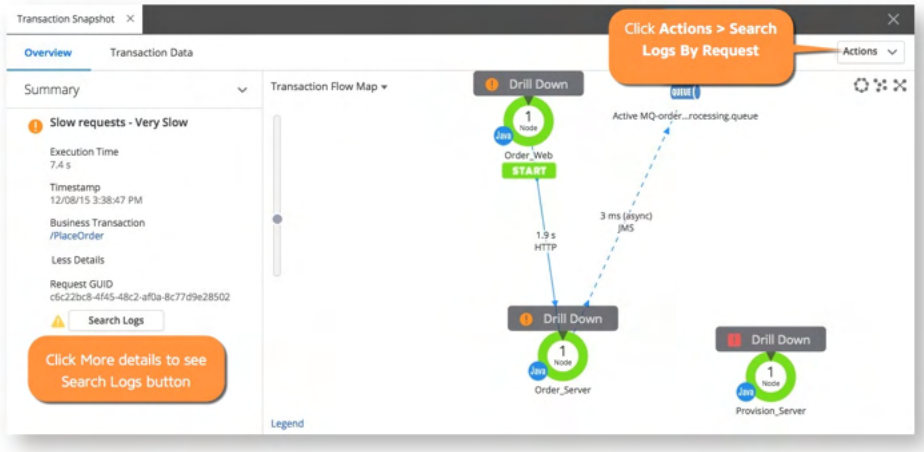
Transaction Analytics to Log Analytics

You can select relevant transactions from any transaction analytics search and see details.

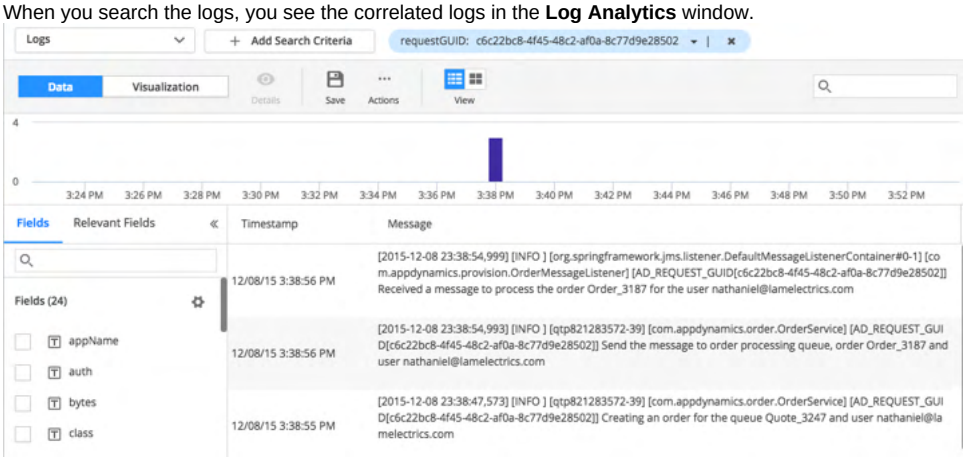
1. Select the transaction, then double-click or click **Details** in the **Action** toolbar.



2. On the **Overview** tab, you have two ways to search the logs by request GUID.



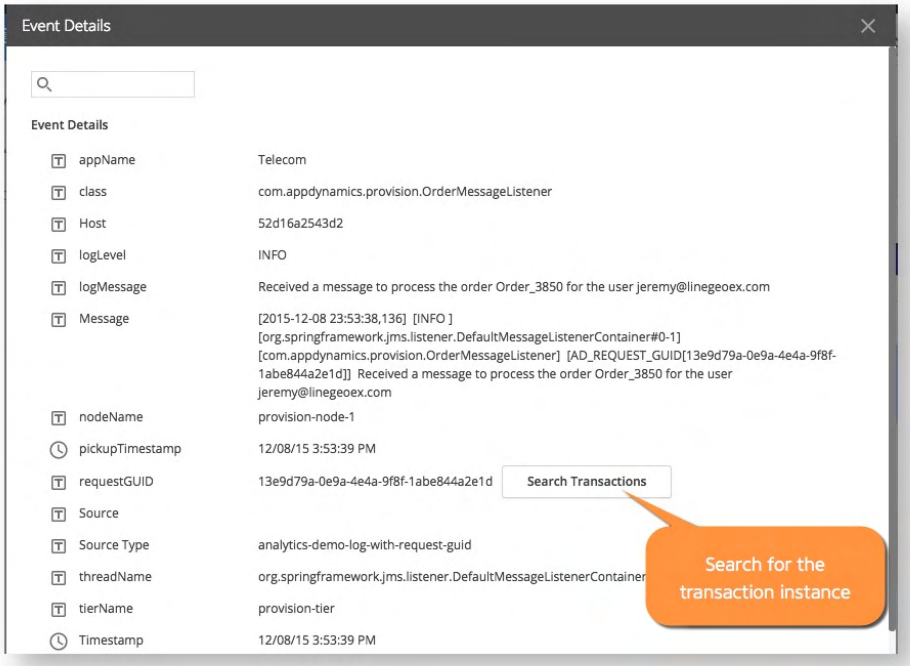
3. When you search the logs, you see the correlated logs in the **Log Analytics** window.



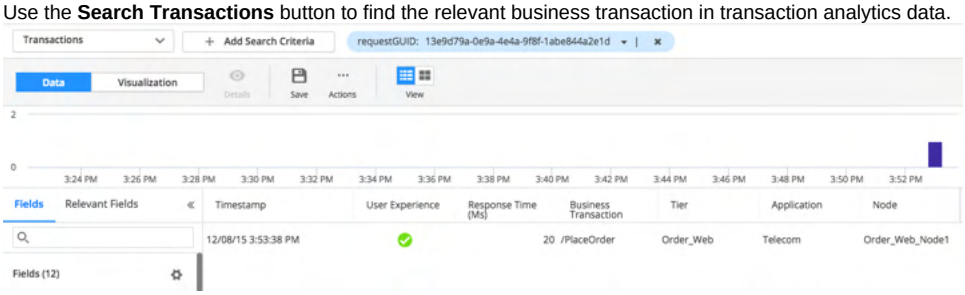
From Log Analytics to Transaction Analytics

You can select a log from any log analytics search and see the Event Details.

1. Select the log, then double-click or click **Details** in the **Action** toolbar. You can see the **Event Details** window.



2. Use the **Search Transactions** button to find the relevant business transaction in transaction analytics data.



Deploy Analytics in Kubernetes

This page describes deployment options for Transaction Analytics and Log Analytics instrumented with AppDynamics app server agents in Kubernetes applications.

Transaction Analytics (except for Java and .NET Agent) and Log Analytics require that an Analytics Agent is deployed with an app server agent.

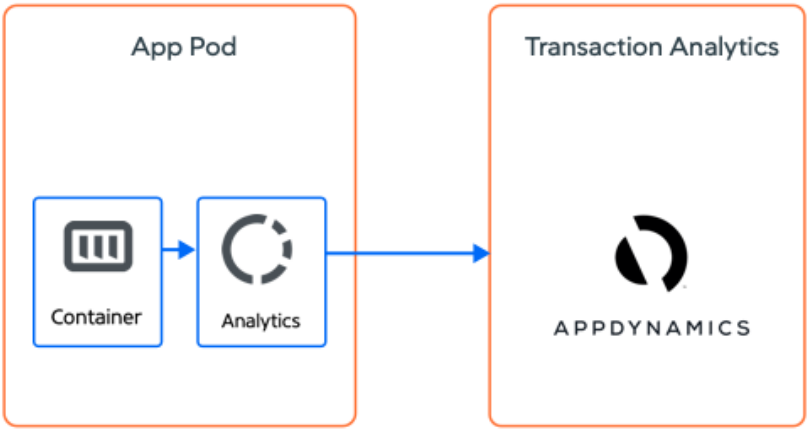
> For Transaction Analytics, the Java Agent >= 4.5.15 or the .NET Agent >= 20.10 supports "agentless" analytics, which does not require that an Analytics Agent is deployed. See Deploy Analytics Without the Analytics Agent.

Transaction Analytics

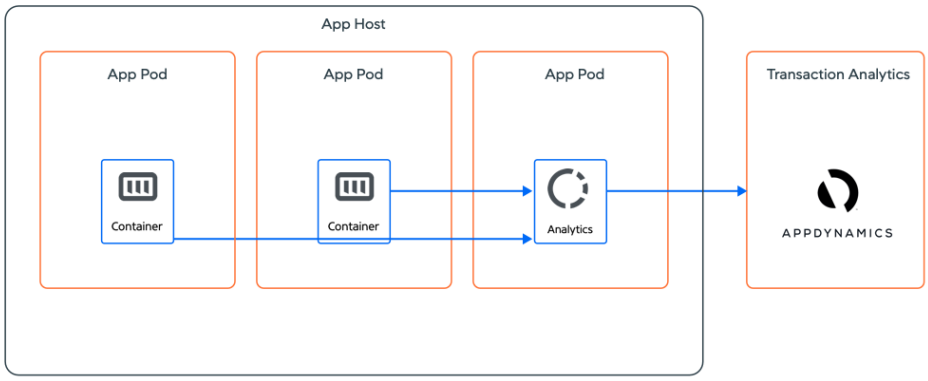
The Analytics Agent acts as a proxy between the app server agent and the Events Service. See Deploy Analytics With the Analytics Agent.

There are two deployment options for the Analytics Agent to support Transaction Analytics on a Kubernetes application.

1. A sidecar to the application container.



   In this model, an Analytics Agent container is added to each application pod and will start/stop with the application container.
2. A shared agent where a single Analytics Agent is deployed on each Kubernetes worker node. Each pod on the node will use that Analytics Agent to communicate with the Events Service.



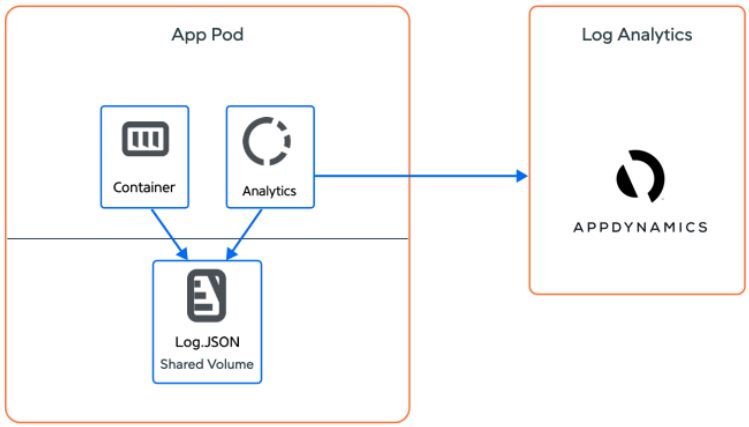
In this model, the Analytics Agent is deployed as a Daemonset.

Log Analytics

Once deployed, the Analytics Agent has access to the application's logs and can send log data to the Events Service.

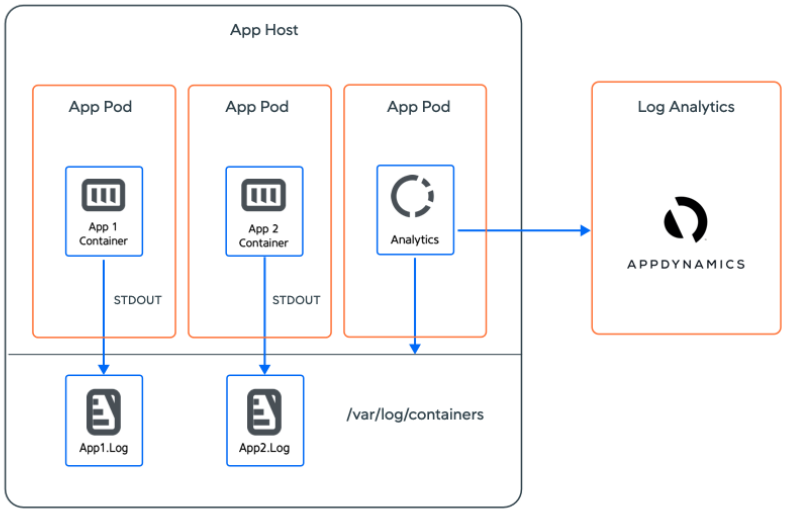
There are three deployment options for the Analytics Agent to support Log Analytics on a Kubernetes application.

1. A sidecar to the application container.



In this model, an Analytics Agent container is added to each application pod and will start/stop with the application container. The Analytics Agent and application container are configured to share a volume where the application logs are written.

2. If the application bypasses the container filesystem and emits log data to `STDOUT` and `STDERR`, the Analytics Agent can be deployed on each Kubernetes worker node. The Analytics Agent can access the log output for every application container on the worker node's file system, stored by Kubernetes under `/var/log/containers` as a unique file per container.



In this model, the Analytics Agent is deployed as a Daemonset.

> For some Kubernetes distributions such as OpenShift, the Analytics Agent will require elevated permissions to access the files under `/var/log/containers`.

3. If a syslog provider is available in the Kubernetes cluster, the Analytics Agent can be deployed to receive syslog messages with TCP transport. A single Analytics Agent instance is required per syslog provider. See Collect Log Analytics Data from Syslog Messages.

For Transaction and Log Analytics, the sidecar approach is simpler to deploy, but consumes more cluster resources because it requires one additional container per application pod. The shared agent approach adds another deployment object to manage, but can significantly reduce the overall resource consumption for a cluster.

Example Configurations to Deploy the Analytics Agent

The following deployment specs are specific examples of how to implement the deployment options explained above. In addition, see Install the .NET Agent for Linux in Containers and Install the Node.js Agent in Containers for best practices on how to set the Analytics Agent host, port, and SSL environment variables.

Transaction Analytics: Deployment Spec Using A Sidecar

The following deployment spec defines two containers, the application container `flight-services`, which uses an image instrumented with an app server agent, and the Analytics Agent container `appd-analytics-agent`, which uses the Analytics Agent from Docker Hub, d`ocker.io/appdynamics/analytics-agent:latest`.

The `appd-analytics-agent` container leverages a ConfigMap and Secret to configure the Events Service credentials required by the Analytics Agent, including the account access key and global account name. See Install Agent-Side Components.

As a sidecar, the Analytics Agent is available at `localhost` and uses the default port 9090. The app server agent will connect automatically and no additional configuration is required.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flight-services
spec:
  selector:
    matchLabels:
      name: flight-services
  replicas: 1
  template:
    metadata:
      labels:
        name: flight-services
    spec:
      containers:
      - name: flight-services
        image: sashaz/ad-air-nodejs-services-analytics:latest
        imagePullPolicy: IfNotPresent
        envFrom:
          - configMapRef:
              name: controller-info
        env:
        - name: APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY
          valueFrom:
            secretKeyRef:
              key: appd-key
              name: appd-secret
        - name: APPDYNAMICS_AGENT_TIER_NAME
          value: flight-services
        ports:
        - containerPort: 8080
          protocol: TCP
        restartPolicy: Always
      - name: appd-analytics-agent
        envFrom:
        - configMapRef:
            name: controller-info
        env:
        - name: APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY
          valueFrom:
            secretKeyRef:
              key: appd-key
              name: appd-secret
        - name: APPDYNAMICS_EVENTS_API_URL
          valueFrom:
            configMapKeyRef:
              key: EVENT_ENDPOINT
              name: controller-info
        - name: APPDYNAMICS_GLOBAL_ACCOUNT_NAME
          valueFrom:
            configMapKeyRef:
              key: FULL_ACCOUNT_NAME
              name: controller-info
        image: docker.io/appdynamics/analytics-agent:latest
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9090
          protocol: TCP
        resources:
          limits:
            cpu: 200m
            memory: 900M
          requests:
            cpu: 100m
            memory: 600M
...
```

The full spec can be found in the Flight Services YAML File. The `controller-info` ConfigMap can be found in the Controller Info YAML File. The command to create `appd-secret` can be found in Secret.

Transaction Analytics: Deployment Specs Using A Shared Analytics Agent

The following deployment spec is for the same `flight-services` application, but instead of using a sidecar, it references a shared Analytics Agent deployed separately as a Daemonset. The `flight-services` container sets the agent environment variables `APPDYNAMICS_ANALYTICS_HOST` and `APPDYNAMICS_ANALYTICS_PORT` to the `analytics-proxy` service for the shared Analytics Agent defined in the example below.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flight-services
spec:
  selector:
    matchLabels:
      name: flight-services
  replicas: 1
  template:
    metadata:
      labels:
        name: flight-services
    spec:
      containers:
      - name: flight-services
        image: sashaz/ad-air-nodejs-services-analytics:latest
        imagePullPolicy: IfNotPresent
        envFrom:
          - configMapRef:
              name: controller-info
        env:
        - name: APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY
          valueFrom:
            secretKeyRef:
              key: appd-key
              name: appd-secret
        - name: APPDYNAMICS_AGENT_TIER_NAME
          value: flight-services
        - name: APPDYNAMICS_ANALYTICS_HOST
          value: analytics-proxy
        - name: APPDYNAMICS_ANALYTICS_PORT
          value: "9090"
        ports:
        - containerPort: 8080
          protocol: TCP
      restartPolicy: Always
...
```

The full spec can be found in the Flight Services YAML File. Use this spec in conjunction with the following deployment spec.

In the `analytics-agent.yaml` file below, the shared Analytics Agent is deployed as a Daemonset. The file also defines a service `appd-infra-agent-service` that publishes an endpoint in the namespace where the shared Analytics Agent can be reached.

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: appd-infra-agent
spec:
  selector:
    matchLabels:
      name: appd-infra-agent
  template:
    metadata:
      labels:
        name: appd-infra-agent
    spec:
      serviceAccountName: appdynamics-infraviz
      containers:
      - name: appd-analytics-agent
        envFrom:
        - configMapRef:
```

```
            name: controller-info
        env:
        - name: APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY
          valueFrom:
            secretKeyRef:
              key: appd-key
              name: appd-secret
        - name: APPDYNAMICS_EVENTS_API_URL
          valueFrom:
            configMapKeyRef:
              key: EVENT_ENDPOINT
              name: controller-info
        - name: APPDYNAMICS_GLOBAL_ACCOUNT_NAME
          valueFrom:
            configMapKeyRef:
              key: FULL_ACCOUNT_NAME
              name: controller-info
        image: docker.io/appdynamics/analytics-agent:latest
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9090
          protocol: TCP
        resources:
          limits:
            cpu: 200m
            memory: 900M
          requests:
            cpu: 100m
            memory: 600M
        volumeMounts:
        - name: ma-log-volume
          mountPath: /opt/appdynamics/conf/logging/log4j.xml
          subPath: log4j.xml
        - mountPath: /hostroot
          name: hostroot
          readOnly: true
      restartPolicy: Always
      volumes:
      - name: ma-log-volume
        configMap:
          name: ma-log-config
      - name: hostroot
        hostPath:
          path: /
          type: Directory
---
apiVersion: v1
kind: Service
metadata:
  name: appd-infra-agent-service
spec:
  selector:
    name: appd-infra-agent
  ports:
  - name: "9090"
    port: 9090
    targetPort: 9090
status:
  loadBalancer: {}
```

The full deployment spec can be found in the Machine Agent YAML File. The `appdynamics-infraviz` service account is defined in the RBAC YAML File . The ma-log-config ConfigMap is defined in the Machine Agent Log Config File.

A best practice is to deploy the shared Analytics Agent in a dedicated namespace (typically `appdynamics`) separate from the namespaces used by applications.

```
$ kubectl -n appdynamics apply -f analytics-agent.yaml
```

To provide access to the shared Analytics Agent from an application namespace:

1. An `ExternalName` service is required to map a service name (`analytics-proxy` in the example) to the DNS name of `appd-infra-agent-service` created previously:

```
kind: Service
apiVersion: v1
metadata:
  name: analytics-proxy
spec:
  type: ExternalName
  externalName: appd-infra-agent-service.appdynamics.svc.cluster.local
  ports:
  - port: 9090
    targetPort: 9090
```

2. Create this service in each application namespace where an App Server Agent is deployed:

```
$ kubectl -n <app namespace> apply -f analytics-proxy.yaml
```

3. Note that `analytics-proxy` is the value of `APPDYNAMICS_ANALYTICS_HOST` used in the `flight-services` deployment spec.

```
- name: APPDYNAMICS_ANALYTICS_HOST
  value: analytics-proxy
```

Log Analytics: Deployment Spec Using A Side Car

The following deployment spec snippet is for a Java application that defines an application container, `client-api`, and an Analytics Agent container, `appd-analytics-agent`, that acts as a sidecar to the application container. An init container, `appd-agent-attach`, is also defined, but the related definitions are removed to simplify the example.

A shared volume, `appd-volume`, is mounted to the application container and Analytics Agent container using the mount path `/opt/appdlogs`. The Java application is configured to write its logs to this path and the Analytics Agent is configured to read the logs from this path and send them to the Events Service.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    name: client-api
  name: client-api
spec:
  selector:
    matchLabels:
      name: client-api
  template:
    metadata:
      labels:
        name: client-api
    spec:
      containers:
      - name: client-api
        envFrom:
        - configMapRef:
            name: agent-config
        env:
        - name: APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY
          valueFrom:
            secretKeyRef:
              key: appd-key
              name: appd-secret
        - name: JAVA_OPTS
          ...
        image: sashaz/java-services:v5
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 8080
          protocol: TCP
```

```
        resources: {}
        volumeMounts:
        - mountPath: /opt/appdlogs
          name: appd-volume
          ...
      - name: appd-analytics-agent
        env:
        - name: APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY
          valueFrom:
            secretKeyRef:
              key: appd-key
              name: appd-secret
        envFrom:
        - configMapRef:
            name: agent-config
        image: docker.io/appdynamics/analytics-agent:latest
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9090
          protocol: TCP
        resources:
          limits:
            cpu: 200m
            memory: 900M
          requests:
            cpu: 100m
            memory: 600M
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
        volumeMounts:
        - mountPath: /opt/appdlogs
          name: appd-volume
      dnsPolicy: ClusterFirst
      initContainers:
      - name: appd-agent-attach
      ...
      restartPolicy: Always
      schedulerName: default-scheduler
      serviceAccountName: appd-account
      volumes:
      - emptyDir: {}
        name: appd-volume
...
```

The full spec can be found in the Java App YAML File.

Log Analytics: Deployment Spec For Shared Analytics Agent (STDOUT/STDERR Support)

The following deployment spec supports the use case where application containers are emitting logs to `STDOUT` and `STDERR`, not the application container filesystem.

Since Kubernetes writes the container logs to the host under `/var/log/containers`, the Analytics Agent can read them there. The Analytics Agent is deployed as a Daemonset. A volume `varlog` is defined with access to the host path `/var/log/containers` and mounted to the Analytics Agent container, `appd-analytics-agent`. The Analytics Agent is configured to read the container-specific logs written to `/var/log/containers`. See Configure Log Analytics Using Source Rules.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: appdynamics-loganalytics
  namespace: appdynamics
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  labels:
    name: appd-analytics
  name: appd-analytics
spec:
  selector:
    matchLabels:
      name: appd-analytics
  template:
    metadata:
      labels:
        name: appd-analytics
    spec:
      nodeSelector:
        kubernetes.io/os: linux
      containers:
      - name: appd-analytics-agent
        env:
        - name: APPDYNAMICS_AGENT_ACCOUNT_ACCESS_KEY
          valueFrom:
            secretKeyRef:
              key: controller-key
              name: appd-secret
        envFrom:
        - configMapRef:
            name: agent-config
        image: docker.io/appdynamics/analytics-agent:log-20.6.0
        imagePullPolicy: Always
        ports:
        - containerPort: 9090
          protocol: TCP
        - containerPort: 5144
          hostPort: 5144
          protocol: TCP
        resources:
          limits:
            cpu: 300m
            memory: 900M
          requests:
            cpu: 200m
            memory: 800M
        volumeMounts:
        - name: varlog
          mountPath: /var/log
          readOnly: true
        - name: dockerlog
          mountPath: /var/lib/docker/containers
          readOnly: true
      restartPolicy: Always
      serviceAccountName: appdynamics-loganalytics
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
      - name: dockerlog
        hostPath:
          path: /var/lib/docker/containers
```

The full spec can be found in the Log Analytics YAML File. An OpenShift example can be found in the Log Analytics OpenShift YAML File.

Using Analytics Data

Related Pages:

- Configure Analytics
- Create Analytics Metrics From Scheduled Queries
- Analytics and Data Security

This describes how to use the Analytics data. You have several strategies available for locating and using analytics data of interest. Once you have identifie d data that is useful for your business, you can use visualization or other features to represent and monitor that data.

Analytics enables you to:

- Configure data sources to collect data for analytics.
- Create and visualize searches and add them to custom dashboards.
- Create API Keys for users of the Analytics Events API.
- Create and view metrics created from analytics searches.
- Define and monitor Business Journeys.
- Define Experience Level reports.
- Export data to CSV files.

Strategies for Locating Data of Interest

- Use a saved search by selecting one from the list. Use the actions in the action bar to manage named searches by saving, duplicating, and so on.
- Create searches for specific use cases. See Search Analytics Data.
- Focus on a specific time range by dragging your mouse across the event stream or by using the time range dropdown.
- Select various fields from the data, targeting the kind of data of interest to you, then scan the event list. Double-click any specific event to display more information.
- Examine Top 10 Values. Click on a field to see the top 10 values of that field in your filtered dataset. The results are presented as a count and percentage of all data within the specified time range for that field. These values provide immediate insights from your data without having any predefined rules or previous knowledge of the data. You can add a value to the search criteria bar by hovering over the value and clicking the **+** icon.
- Relevant Fields. This feature helps you find fields with a high relevance score. A high relevance score indicates these fields are significantly more common in your filtered results than in the entire data set and may be useful to investigate. See Investigate Using Relevant Fields for a suggested workflow.

Visualize Analytics Data

Once you have defined your data, use the Visualization tab to explore specific aspects of the data that interests you and drill down into the relationships you need to understand.

Each visualization type is a widget and widgets can be added/removed. See Visualize Analytics Data.

Search Analytics Data

You can search Analytics data in two modes:

- Basic mode, where you can add search criteria to filter your data and return a subset of available events. In Basic mode, you see the list of events that match your search criteria, along with a count of those events. This mode is referred to in the UI as **Drag and Drop Search**.
- Advanced Mode, where you can use the AppDynamics Query Language (ADQL). ADQL is a SQL-like query language that provides additional operators and functions to enable more complicated searches. This mode is referred to in the UI as **Query Language Search.**

For each mode, you see the analytics data sources that are licensed and enabled for your application. The applications, data sources, and other fields that you see depend on how the administrator has set the permissions for your role. When creating new roles, remember that granting permissions to view transaction analytics data does not automatically grant permissions to see all application data associated with a specific transaction analytics record. You need to grant at least read-only permissions to the application to enable the user to see associated transaction snapshot data such as flow maps.

Search Results Limits

Search results in the UI can return up to 1000 records. By default, each page displays 50 results at a time. In Advanced mode, you can explicitly set a LIMIT to see more than 50 results on a page. Setting a limit disables pagination and shows up to 1000 records on a single page. The total number of records for the query shows at the bottom right. The UI caps the number of search results at 1000 regardless of mode or the value of an explicit LIMIT. To return more than 1000 records, you must use the Analytics Events Query API.

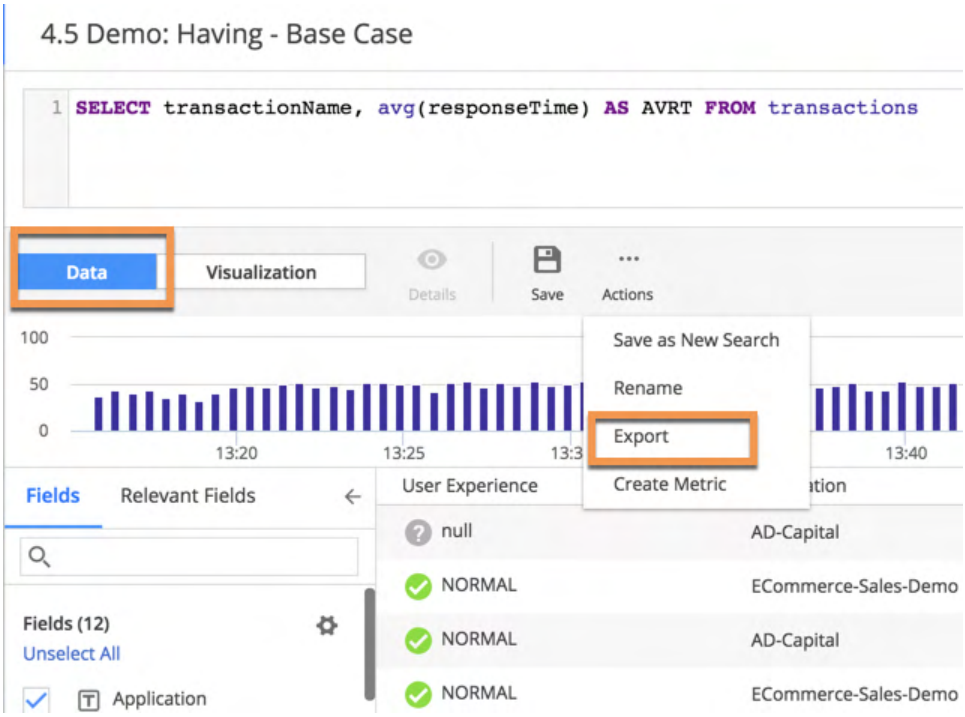
See LIMIT Clause.

Exporting Analytics Data

Analytics allows you to export data to a file in CSV format for external use.

Before you export a set of data, consider these guidelines:

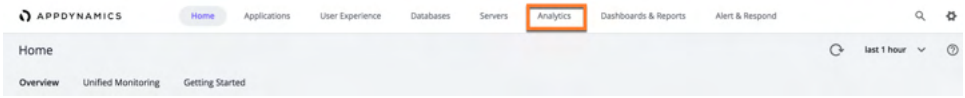
- The maximum number of records exported for a non-aggregation query is 65000. Use the LIMIT clause to restrict the records to a number below the upper bound. The LIMIT clause does not return any additional records that satisfy the export selection criteria.
- The number of records exported for an aggregation query by default is 10. If you want more than 10 records, use the LIMIT clause. You can export up to 1000 records by using the LIMIT clause.
- The CSV file format is RFC4180.
- Response from all valid ADQL queries can be exported. The method of writing ADQL queries remains the same.
- Only the fields that you specify in the SELECT clause are exported to the CSV file. ADQL query supersedes the UI selection. The fields selected to be displayed in the Data tab has no impact on the exported data even if the query includes `SELECT *`.
- Fields having nested representation are flattened and appear as separate fields in the CSV file. Additionally, the corresponding unflattened nested fields also appear in the exported records. Flattening fields refers to separating nested fields into individual fields.
- The supported datetime format is ISO, `yyyy-MM-dd'T'HH:mm:ssZZ`, without the millisecond value. The timezone is local to the browser from which export is requested. A timestamp field in a nested field is converted to the local timezone only in the flatten fields, not in the unflattened nested field. The format in the UI, that is `MM/DD/YYYY HH:MM:SS AM/PM`, is different from the format that you see in the exported data.
- The time required to export data varies with the size of the record you are trying to export. Exporting tends to slow down for larger set of records because executing the query on and retrieving a large data set takes longer to complete.



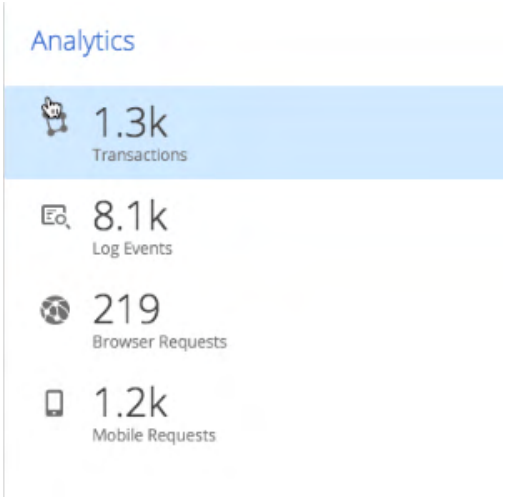
You can export metrics the Metric Browser and search results from the Searches screen. Use the **Export** option under **Actions** on the toolbar to do so. While you are on the Searches panel, use the **Data** mode to export your search result.

Access the Analytics Search UI

1. Navigate to the Analytics interface in the Controller UI by clicking **Analytics** in the top navigation bar.

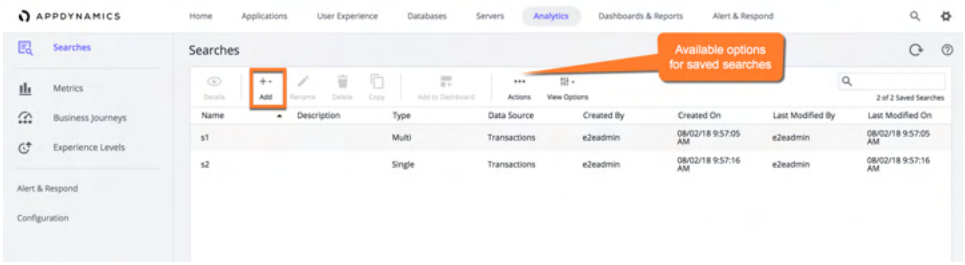


   An alternate way to access Analytics is from the Home page: select a data source to see a default search for that event type.

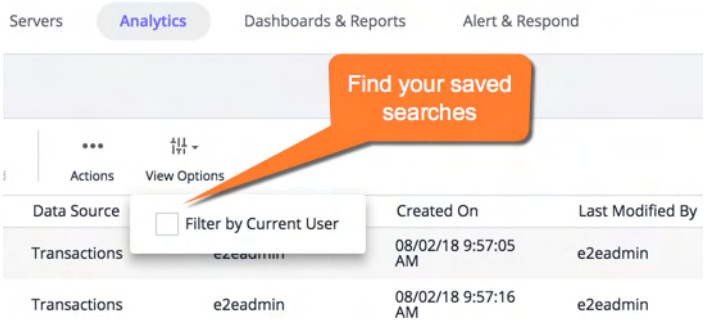


2. Click **Searches** to create new searches and see a list of previously saved searches.

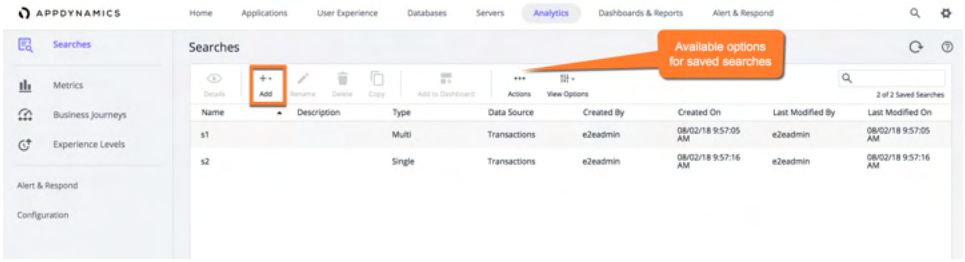
   The saved searches list displays any pre-existing, saved searches for which you have view permissions. The default sort order for the list is alphabetical by saved search name.



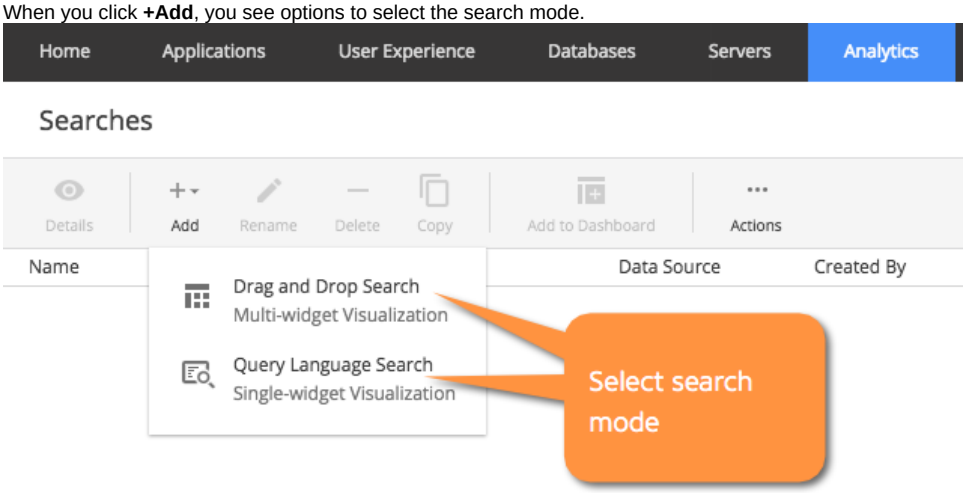
3. To view your saved searches, click **View Options** and select **Filter by Current User**.



4. Click **+Add** to create a new search or select an existing search to activate the actions on the action toolbar.



5. When you click **+Add**, you see options to select the search mode.



For details on the two search modes, see:

- Drag and Drop Search, see Create Basic Analytics Searches
- Query Language Search: see Create Advanced Analytics Searches

On the Search details page, you see two tabs for displaying the results for the Search.

- All event types share the same basic layout on the Data tab of the Search details page.
- Use the Data tab to search and filter the data that you want to review and analyze.
- Use the Visualization tab to create graphic representations based on that data.
- Search actions are available for each search. The actions vary according to the type of search and the tab you use. These actions are available:
  - Rename
  - Export (to CVS - only from the Data tab)
  - Create Metric
  - Add to Dashboard (only from the Visualization tab)

Basic and Advanced Search Comparison

This table provides a comparison of the functionality available in each search mode.

Functionality	Basic	Advanced
Drag and drop fields to the criteria search bar	yes	no
Type-ahead (auto-complete)	N/A	yes
Visualization canvas widget support	multiple widgets	one widget per search query
Grouping of events with the GROUP BY keyword	yes	yes
Wild cards with comma-separated text (Basic mode) or **IN** operator (Advanced mode)	no	no
Percentile Histogram widget	yes	no

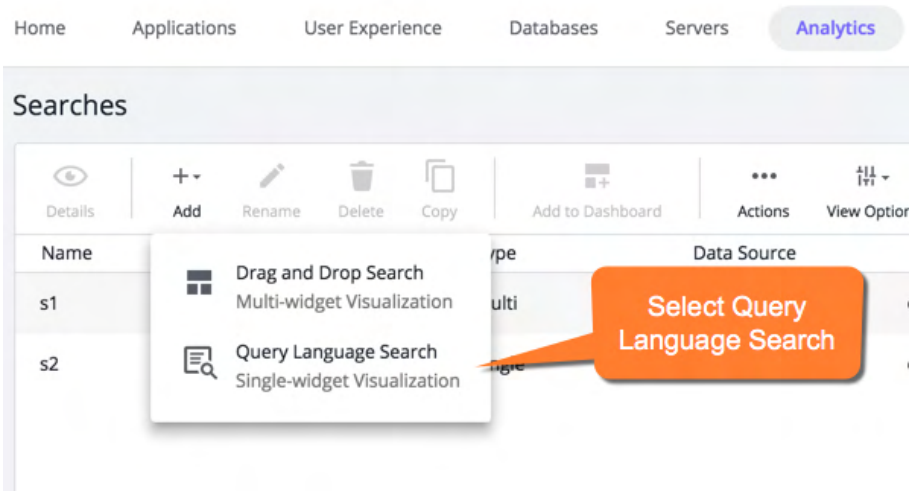
Create Advanced Analytics Searches

Related pages:

- Using Analytics Data
- ADQL Reference

This page describes how to use the AppDynamics Query Language (ADQL) to search analytics data using **Advanced** mode in the Analytics Search panel. If you have never used AppDynamics Analytics, you might want to start with basic mode before using Advanced mode for your searches.

Access Advanced mode by clicking **Add+** on the Analytics Search page. Select **Query Language Search**:



Construct ADQL Queries

Each ADQL event query begins with a **SELECT** statement that uses a **FROM** clause. Use the **FROM** clause to specify the data source you want to query. You can query one data source in each query.

Use the **SELECT** statement to further qualify the query using field_expressions and math_expressions. The **WHERE** clause adds conditions. For more details on specific ADQL syntax, keywords, functions, and operators, see the ADQL Reference.

When you click **Search** for queries other than select all fields (**SELECT** *), the results display on the visualization tab where the table widget accurately renders your query.

Use ADQL Auto-Complete

As you construct your query and type a space after a keyword, auto-complete offers suggestions for the next keyword or operator in the query. The display of auto-complete suggestions is based on the stage of your query and the event type you selected. Auto-complete offers suggestions for data values, such as event types and fields, when you type an equals sign "=" or other comparison operator after the event type name.

Note: The query box in the data tab is used only for adding **WHERE** clause differentiation. Aggregation functions are only available in the visualization tab under the table widget.

In advanced mode, when you type **SELECT**, the fields from the last specified event type display in the auto-complete list. Fields specific to a new event type display after you complete the **FROM** clause.

> Remember to type a space in front of the next keyword or you may see a validation error when executing the search or the search will be inexecutable.

Field Name Mapping

The field names in the auto-complete dropdown list correspond to UI labels and are obvious in most cases with minor differences in capitalization. When the field names have prepended database-related information, such as segments.errorList.errorType for Error Type or segments.httpData for HTTP data fields, they refer to data for individual segments of the transaction as it progresses from tier to tier. The segments fields do not represent data that refers to the overall business transaction instance. See ADQL Data for a list of fields, UI labels, and internal database names for each data source.

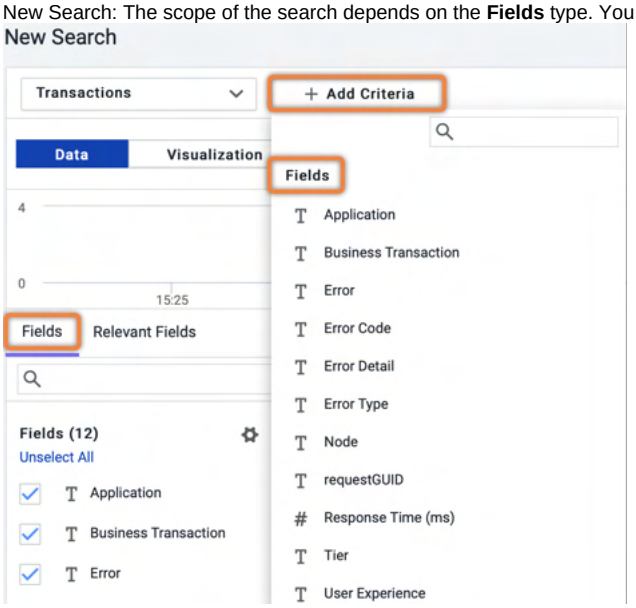
Create Basic Analytics Searches

This page describes UI elements available in the Analytics search panels and focuses on how to create searches in Basic mode. This mode is also called **D rag and Drop Search** in the UI and enables you to do multi-widget visualization of your search results. In this mode, you add search criteria based on the fields collected for the event type. Field search enables you to search for all values or a specific value of the field.

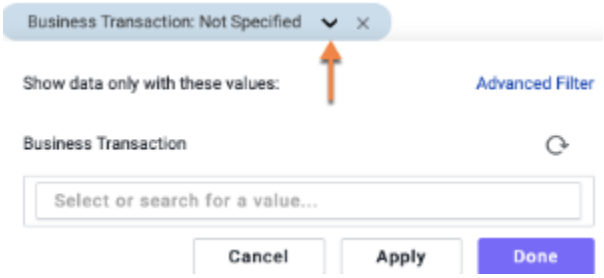
UI Elements

These UI elements are available to help construct your searches:

- New Search: The scope of the search depends on the **Fields** type. You can filter your search with the **+ Add Criteria** button.



- Field value dropdown: After you have selected a field, use the dropdown to add a list of values.



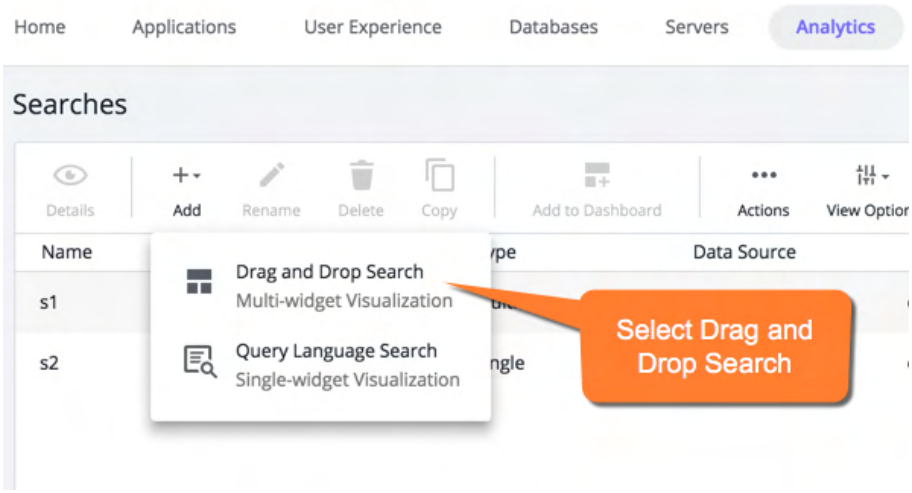
- Combination field selector and search: This field value selector combines a list of values with the ability to search for terms in the values list. It also has an **Advanced Filter** option.
- Text filter: Enables you to do an exact match on the term entered. Supports wild cards. See Search Options.

Basic Search

To create a search using a field:

1. From the Analytics Search list, click **+ Add**.

2. To use basic mode and multi-widget visualizations, select **Drag and Drop Search.**



   To use advanced mode (ADQL) and single-widget visualizations, see Create Advanced Analytics Searches.
3. On the Transactions tab, select the data source type that you want to search.
4. To add filters to your search, click **+ Add Criteria**.
   The field selector dropdown appears, so you can select the field you want to use.
5. Select a field. The field name appears in the search criteria line.
6. Click the dropdown next to the field name to open the field value selector. Select a value for the field.

   Fields such as application, business transaction, and node display a list of the values available. Some fields show a search box where you can enter criteria to filter the values of this field.

Search Options

This tables describes free text search and text filter behavior:

Type of entry	Analyzed	Non-analyzed
term	Find entries with this term. This search is case insensitive. For example, using the term "great" finds "great" and "AppDynamics is great".	Looks for an exact match and is case sensitive. For example, using the term "order" finds "order" but not "/PlaceOrder.
term1 term2	Returns results containing both *term1* and *term2* irrespective of the order the terms appear.	Terms are implicitly AND'd. Looks for an exact match and find results containing "*term1 term2"*.
term1, term2	This works the same as the space-separated list of terms.	Terms are implicitly OR'd. Looks for an exact match for either *term1* or *term2*.
*term** *term?*	Finds all fields where the value is this string plus one additional character <?> or multiple <*>. Because all non-alphanumeric fields are delimiters, the escape character has no effect.	Finds all fields where the value is this string plus one additional character <?> or multiple <*>. To search for a field containing a wild card character, escape with a "\".
NOT *term*	Use the **NOT** comparison operator to find everything except the term. NOT can be used with any of the entries in this table.	Use the **NOT** comparison operator to find everything except the term. NOT can be used with any of the entries in this table.

Free Text Search in Log Analytics

For log analytics, a free text search is available for searching the message field. This enables you to search for any keyword or string anywhere in the message field.

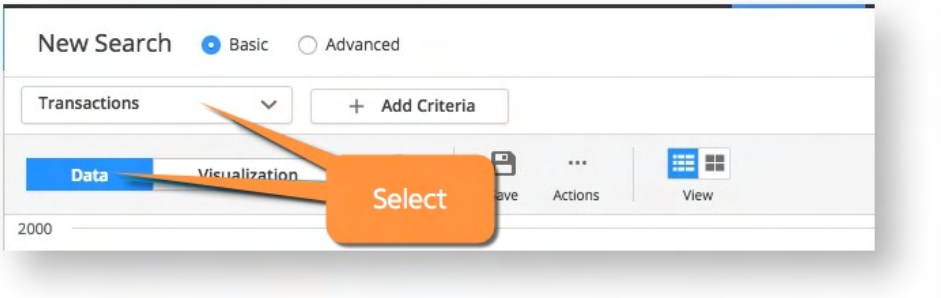
Investigate Using Relevant Fields

You can use Relevant Fields to find data fields that show a high Relevance Score. A high relevance score indicates these fields are significantly more common in your filtered dataset than in the entire dataset and may be particularly useful to analyze. Relevant field investigation may provide valuable insights and enable you to conduct better root cause analysis.

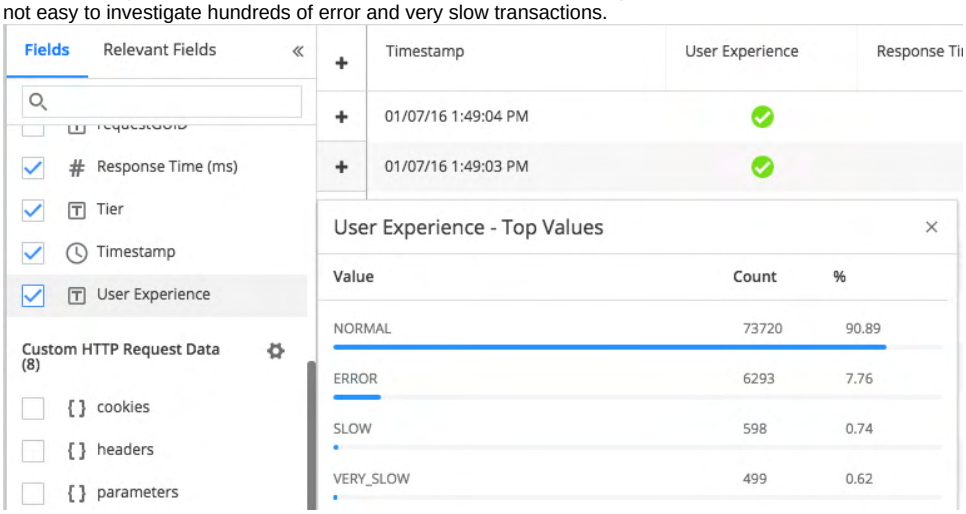
Relevant Fields investigation can be used with all event types.

As an example, you can use the following suggested workflow to investigate poor user experience.

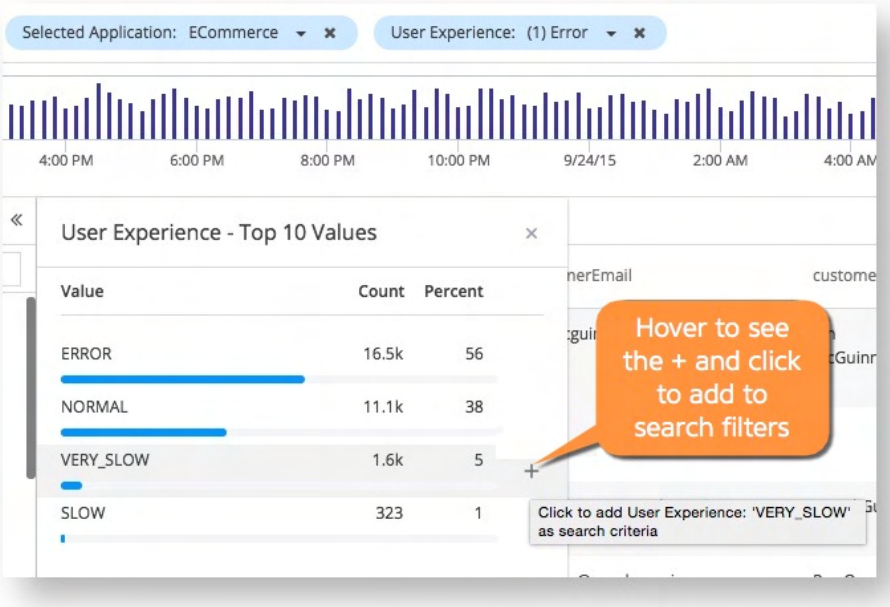
1. From the Analytics Search page, select the **Transactions** event type and the Data tab.



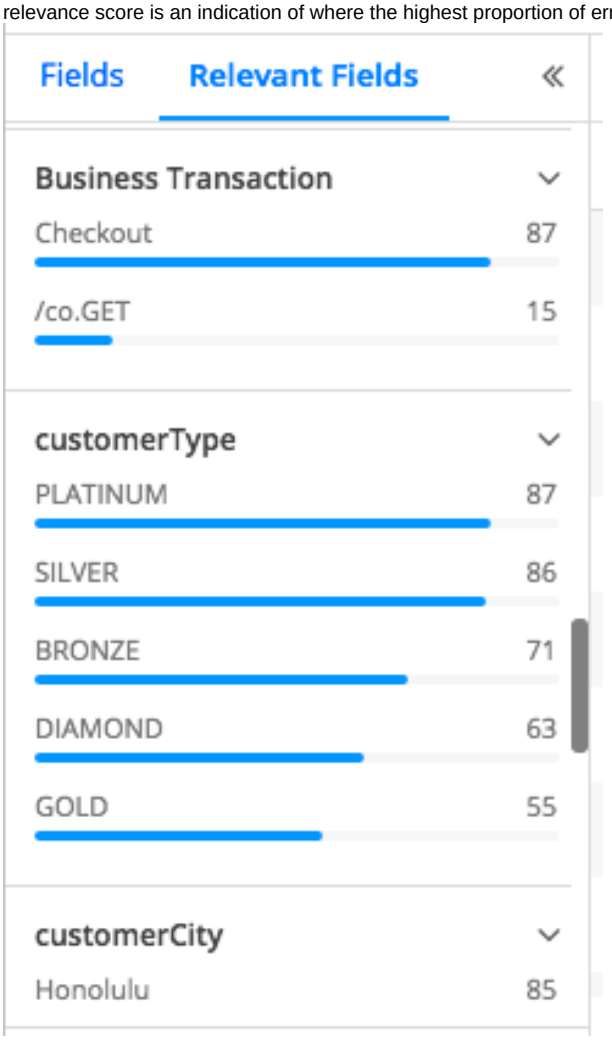
2. Click **Add Criteria** and select an application to investigate.
3. Click **User Experience** or another field in the **Fields** list to see the **Top Values**. If you see lots of error and very slow transactions, you realize it is not easy to investigate hundreds of error and very slow transactions.



4. Add **Error** and **Very Slow** values to your search criteria to narrow the total number of transactions to a smaller dataset. You can add the values quickly by hovering over the field in the **Top Values** list until the **+** sign appears at the right. Click the **+** to add to the search criteria. Now the focus is only on error/very slow transactions.



5. Click the **Relevant Fields** label and now you can see where to focus your investigation. In this example, you might start with the **Checkout** business transaction as the relevance score is 87. You can also see that **Platinum** and **Silver** customers have the two highest scores. The relevance score is an indication of where the highest proportion of error and very slow transactions are occurring.



6. Click a field in the **Relevant Field** list to see more details. You can continue to add relevant fields to your search criteria to keep narrowing the dataset. This process should enable you to find issues of significance for investigation.

Visualize Analytics Data

Related Pages:

- Create Basic Analytics Searches

After you have created a search for the Analytics data that you want to visualize, you can configure a widget to display the results of the search graphically.

Access the Analytics Widget Builders

You can access the Analytics visualization widgets from the Analytics Search UI or directly from **Dashboards & Reports**.

To access the Analytics widget builder from Analytics:

1. Select **Analytics > Searches** to view the saved searches list.
2. Double-click the search that you want to visualize or add a new search.

To access the Analytics widget builder from a Custom Dashboard:

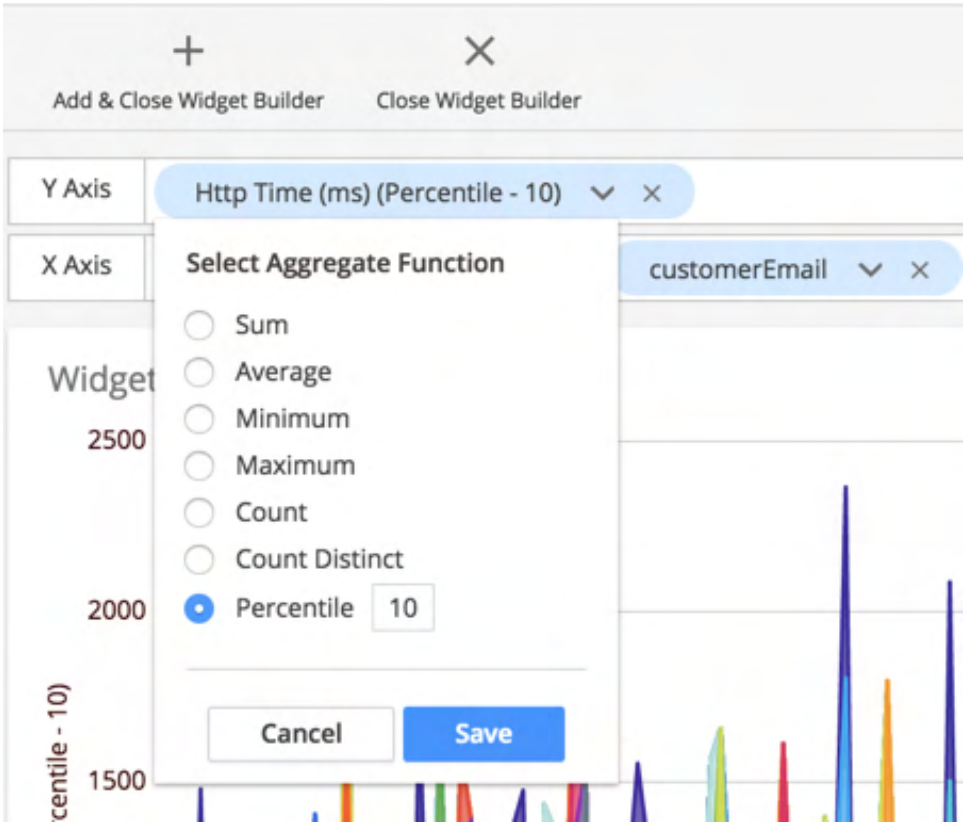
1. From the top navigation bar, click **Dashboards & Reports** to see the custom dashboard list.
2. Either double-click the dashboard where you want to add the Analytics widget or click **Create Dashboard** to create a new dashboard.
3. In the custom dashboard, click **Edit**, then **Add Widget**.
4. Click **Analytics**, **Saved Searches**, or **Experience Levels**.
5. Add Analytics widgets as needed to your dashboard.

Build a Widget

To use the Analytics widget builder:

1. Click the Visualization tab.
2. In the **Visualization** canvas, you can create widgets with one of the following modes:
    - In **Drag and Drop Search** (Basic) mode you can create multiple widgets on the canvas.
    - In **Query Language Search** (Advanced) mode you can generate one widget from your ADQL query at a time. In addition, when you use the Visualization tab in this mode, the order of functions and fields in the query matters. For all widgets (except table), the aggregate functions, such as count(*), avg(), sum(), percentile and so on, should always be preceded by a field. If the fields and functions in the query are not in this order, some visualizations do not work.

    See Conversion Analysis for an example of creating a conversion analysis using the funnel widget.
3. Click **Add Widget**, then select **Custom Widget Builder** or **Funnel Widget Builde**r depending on the type of widget that you want to create.
    In Basic mode, you can build custom widgets and drag fields to the canvas, which uses X and Y axes for configuration. After dragging a field to the axes, select a function from the drop-down next to the field name.
    In this example, the percentile function is applied to **HTTP Time (ms)**.



The funnel widget is configured differently from the other custom widgets, which use the Y and X axes as the basis of the configuration. A funnel widget is used to display data of progressively-staged proportions in a color-coded funnel.
You can add a title to the widget by over-writing **Widget Title** on the canvas.

4. After adding a widget and fields, you can right-click menu widget values to see other options. Select **Apply a global filter for *<field_value>*** to add a value to the existing search criteria.
   The supported boolean operators are **true**, **false**, **null**, and **is not null**. Use **is not null** to check both boolean and non-boolean fields, such as `lo an amount` or `IsReached`.
5. Click **Save & Close** to save the widget configuration.

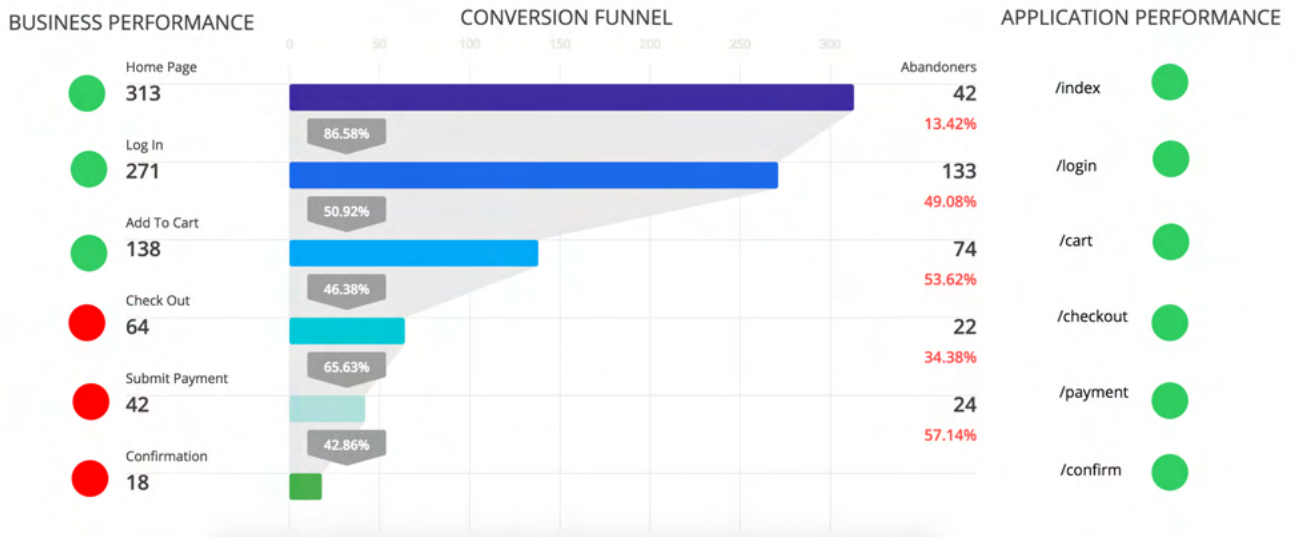
Conversion Analysis Using Funnel Widget

Funnel analysis include a series of events that lead towards a defined goal. For example, tracking user activity in a mobile app, a sale on an eCommerce website, online advertisements, and eventual purchase.

The funnel widget visualizes use cases such as:

- Retail application checkout drop-off rates from the point of adding items to the shopping cart to the actual purchase
- Business Journey drop-off rates from the point of applying for a loan to approving the loan
- Home mortgage loan approval drop-off rates from submitted application to approved applications
- Impact from performance issues by total sales, customer segment, location, and purchased products are gauged in a funnel with other health rule widgets

In this custom dashboard, the impact from performance issues by total sales, customer segment, location, and purchased products are gauged in a funnel with other health rule widgets.

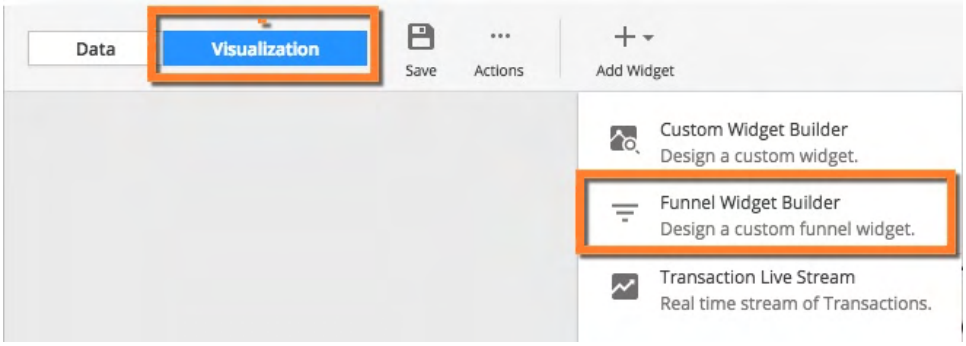


Funnels return the number of unique actors that successfully (or unsuccessfully) make it through a series of steps. Actors could mean users, devices, or any other identifiers that are meaningful to you. A funnel analysis reveals where a given flow loses the most actors. This helps identify areas for improvement, as well as the overall health of your business.

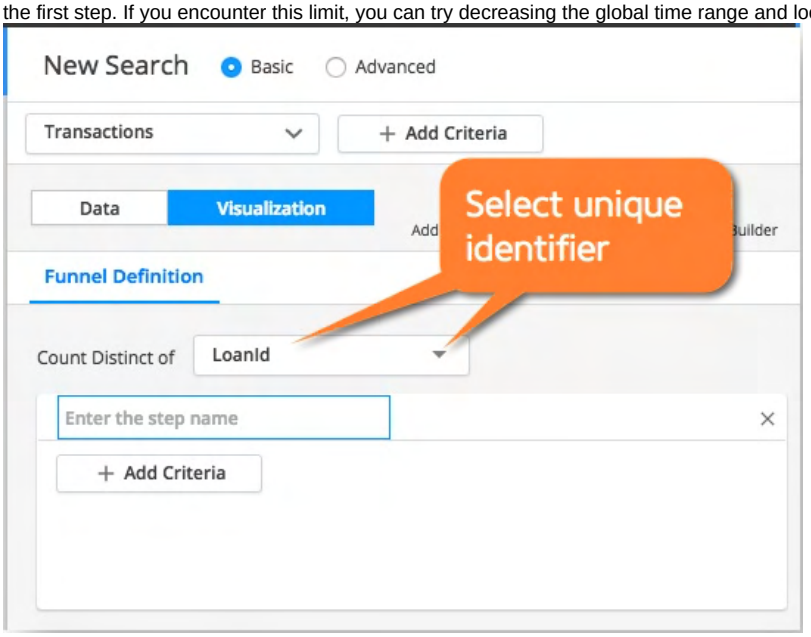
The first step in the funnel is the anchor step, and always represents 100% of the results. The filter criteria for the first step creates a set of unique values, and every subsequent step results in a subset of the values from the previous step and renders the corresponding drop-offs. The second and later steps describe the number of events that match the criteria for each step towards the goal. This number is typically less than 100%, but could be 100% if every event matches the subsequent steps.

Build a Funnel Widget

1. From the **Analytics Search** page using **Drag and Drop** (Basic) Mode, select the Visualization tab.
2. Click **Add Widget > Funnel Widget Builder**.

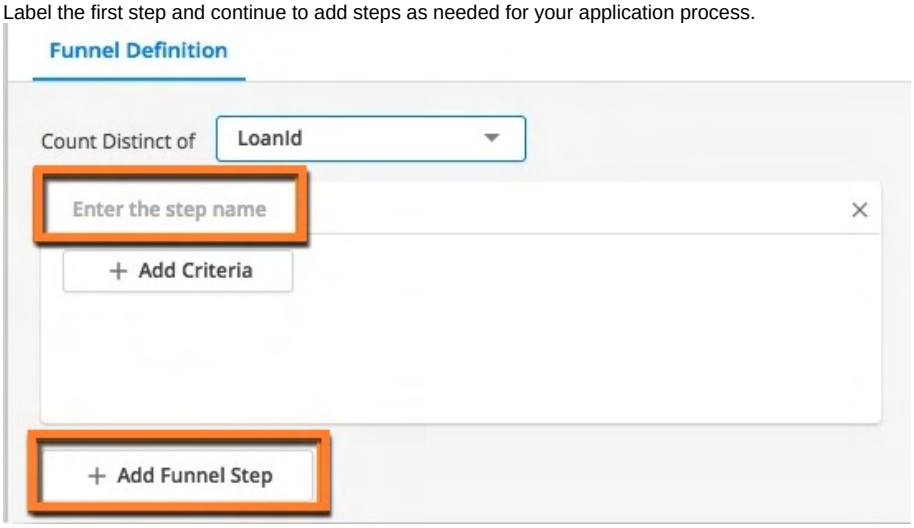


3. In the **Funnel Definition** pane, click the down arrow to select the unique field representing the items that you are counting. In this example, following the steps of a loan approval process, `LoanId` is used. The fields depend on your application. There is a limit of 50,000 unique values for the first step. If you encounter this limit, you can try decreasing the global time range and look at fewer values at a time.

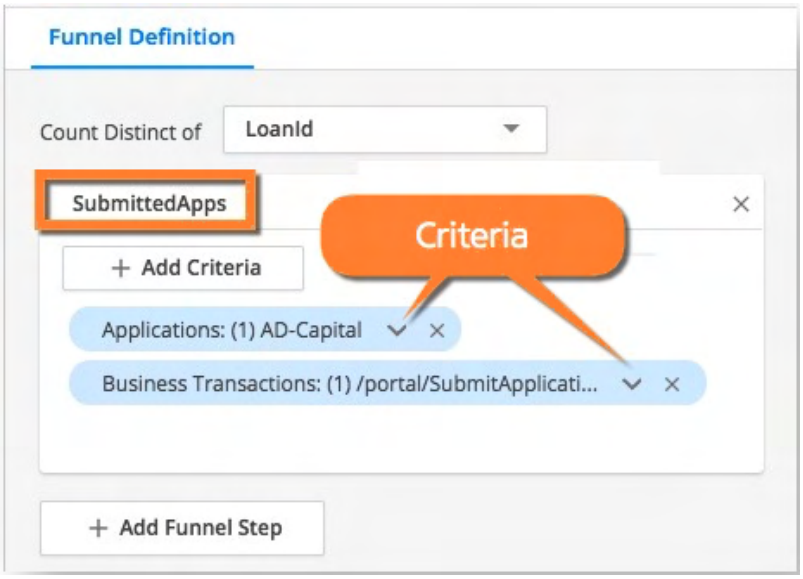


> Filtering on fields with null and empty values may cause more records in subsequent levels.

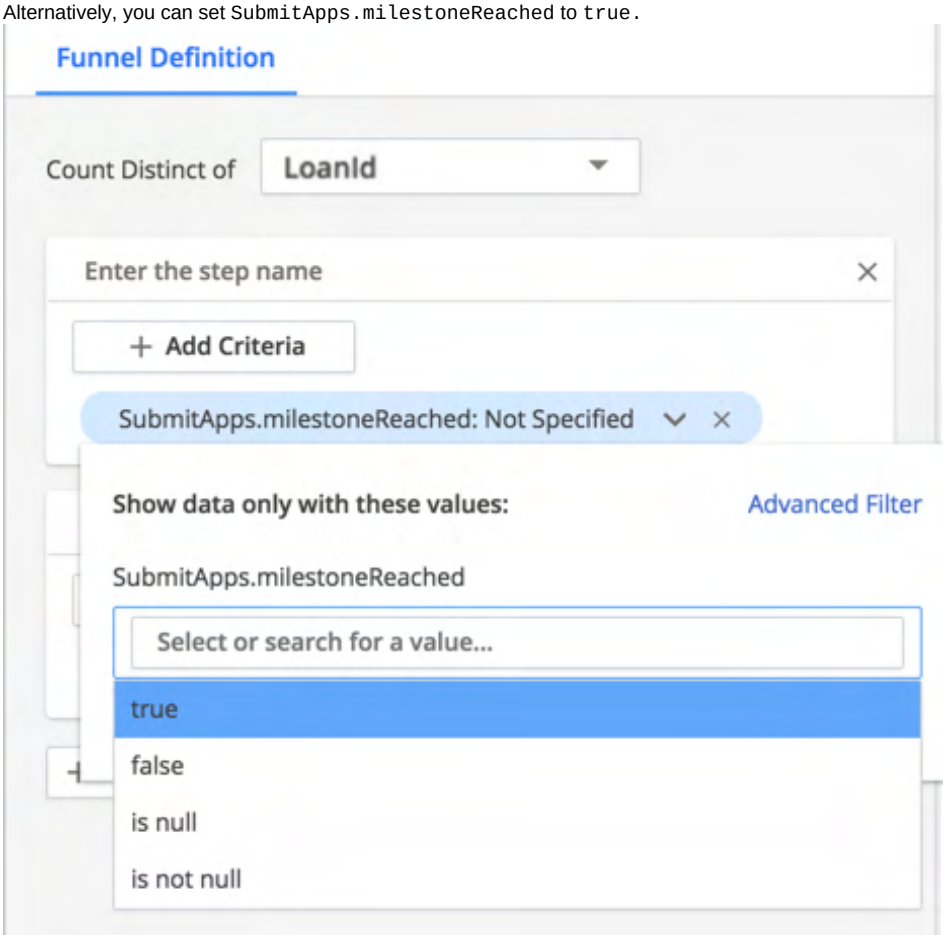
4. Label the first step and continue to add steps as needed for your application process.



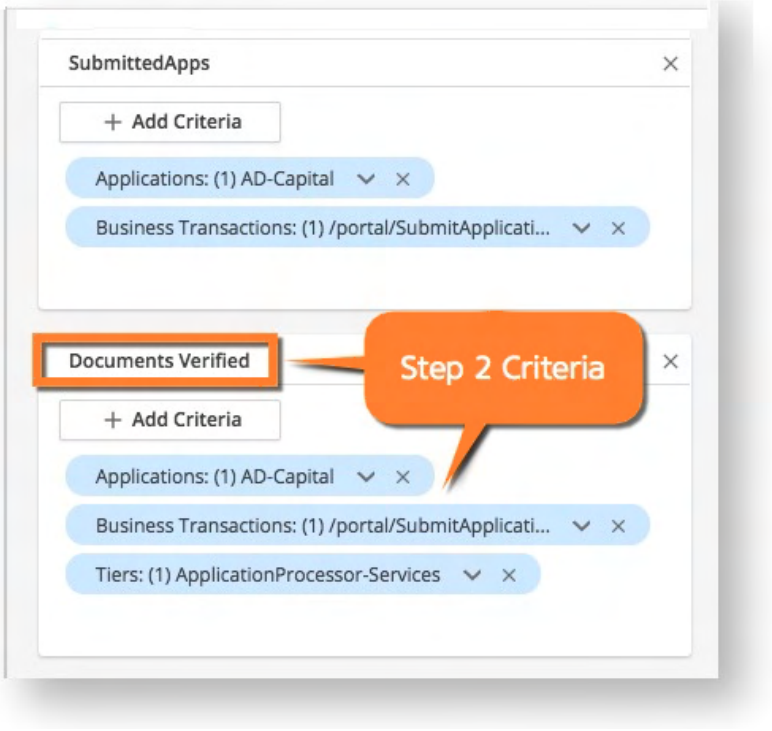
For this example, the first step represents the number of submitted loan applications. The search criteria include the AD-Capital application and the `/portal/SubmitApplication` business transaction.



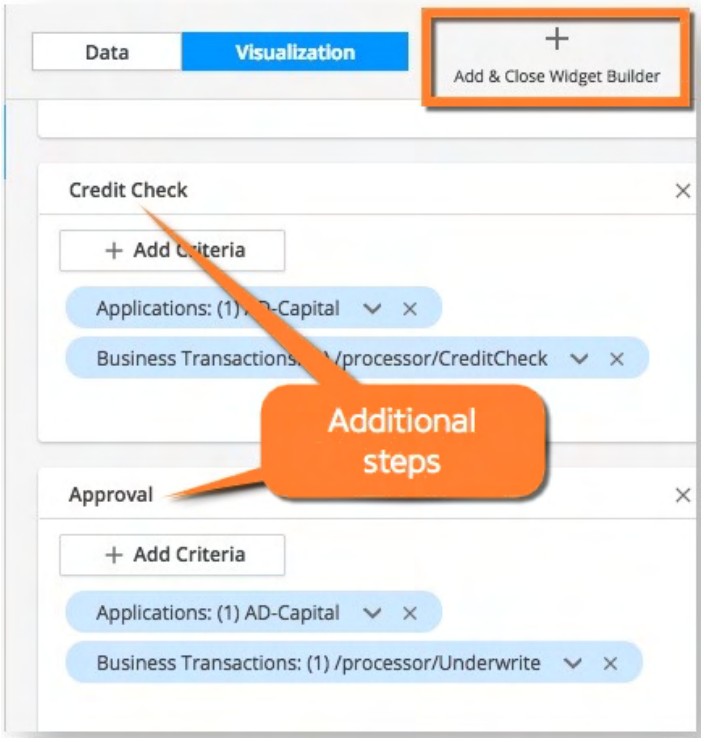
Alternatively, you can set `SubmitApps.milestoneReached` to `true`.



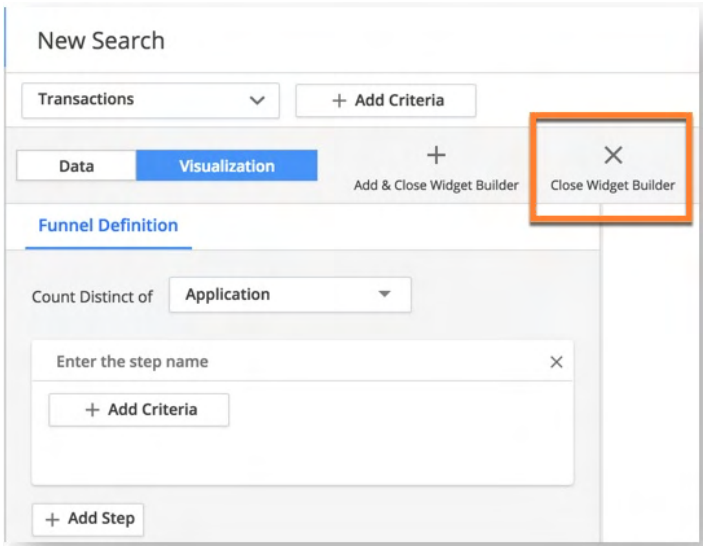
5. Continue to add and label your steps. This step represents the number of loan applications that reached the stage "Documents Verified."



   Alternatively, use the `milestoneReached` field and set it to true.
6. After you have added all the necessary steps, click **Add & Close Widget Builder** to save the completed funnel widget. This example contains two additional steps representing loans that reached the credit check and approval stages.

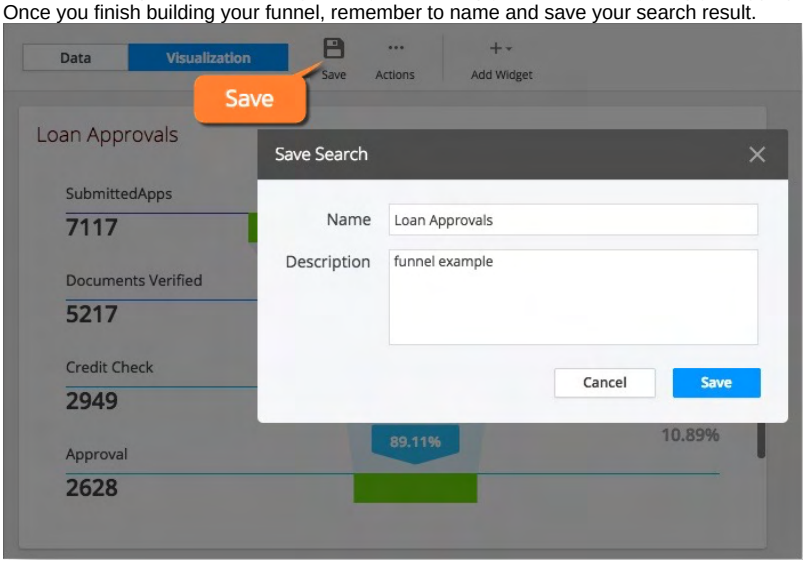


   If you want to cancel editing and revert to the previous version, click **Close Widget Builder**.

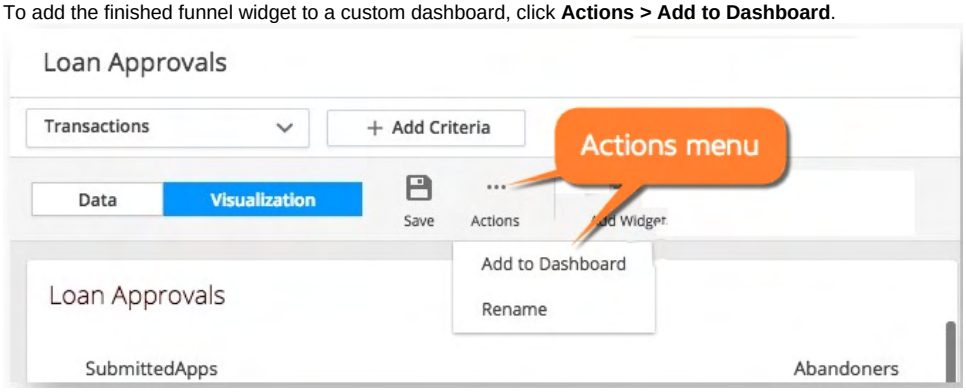


If the funnel steps need a rearrangement you can simply reorder them by simply dragging and dropping each step into appropriate positions.

7. Once you finish building your funnel, remember to name and save your search result.

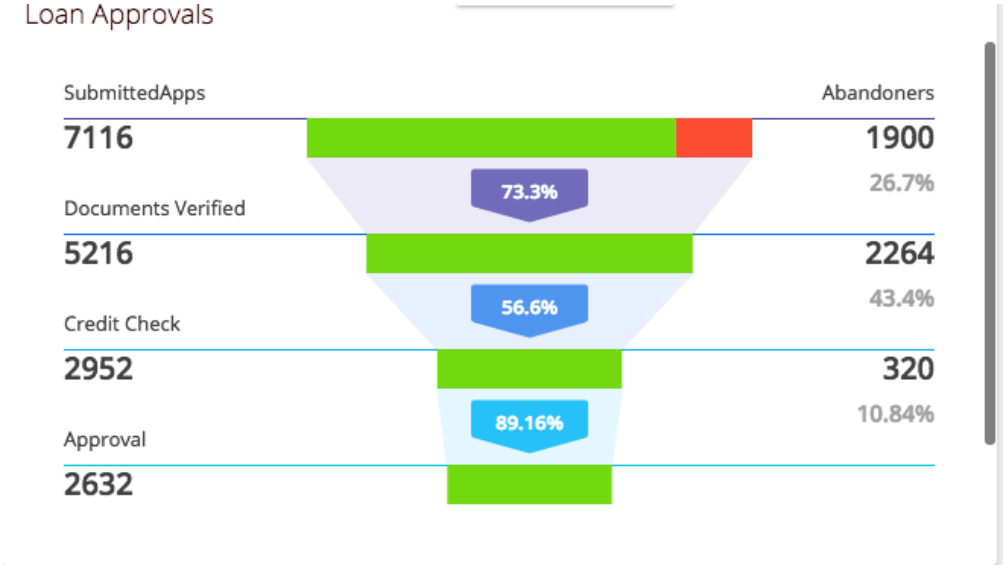


8. To add the finished funnel widget to a custom dashboard, click **Actions > Add to Dashboard**.



Use the Funnel to Troubleshoot

The finished funnel widget for the above example shows the drop-offs (Abandoners*)* in unique loan IDs from submission to approval. When creating the funnel, you can also select **Show Health** in the properties pane to display a health overlay that correlates application health data with the business conversion data. This helps to visualize performance for each step of the funnel widget.



The health overlay is supported for these events types with a user experience value:

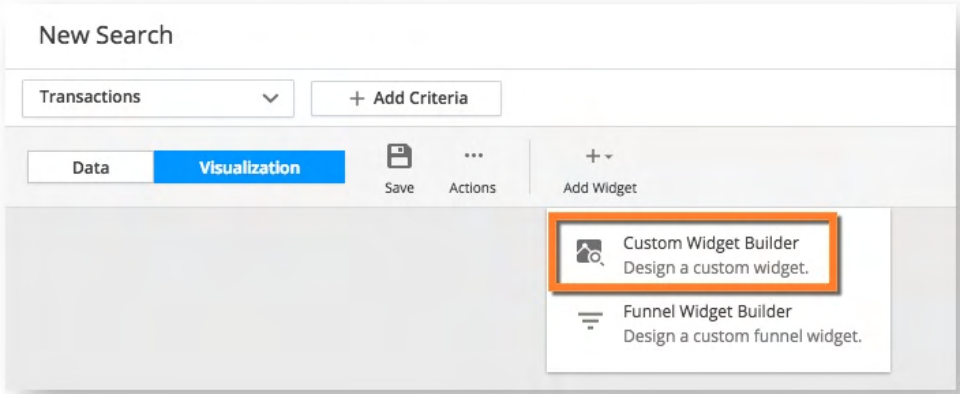
- Transaction Events – `userExperience`
- Browser Records – `pageexperience`
- Mobile Snapshots (Network Requests) – `networkrequestexperience`
- Web Session Records (Browser Sessions) – `experience`
- Mobile Session – `networkrequest.networkrequestexperience`

For enhanced troubleshooting, you can drill down to event data from any of the funnel steps by double-clicking the numbers on conversions and abandoners or by using Health Overlay and clicking on each health segment.

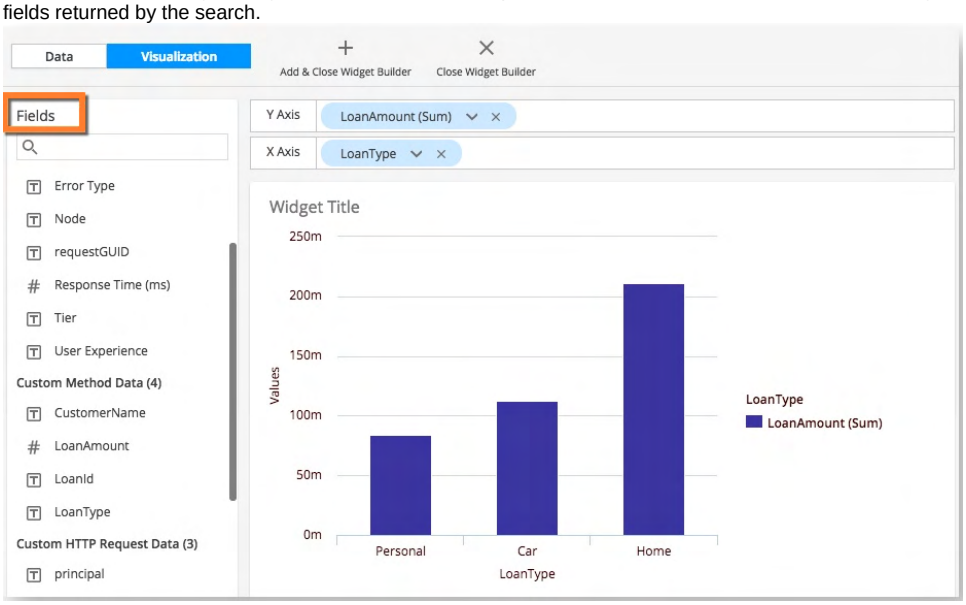
Configure Custom Widgets

To configure custom widgets for your analytics searches:

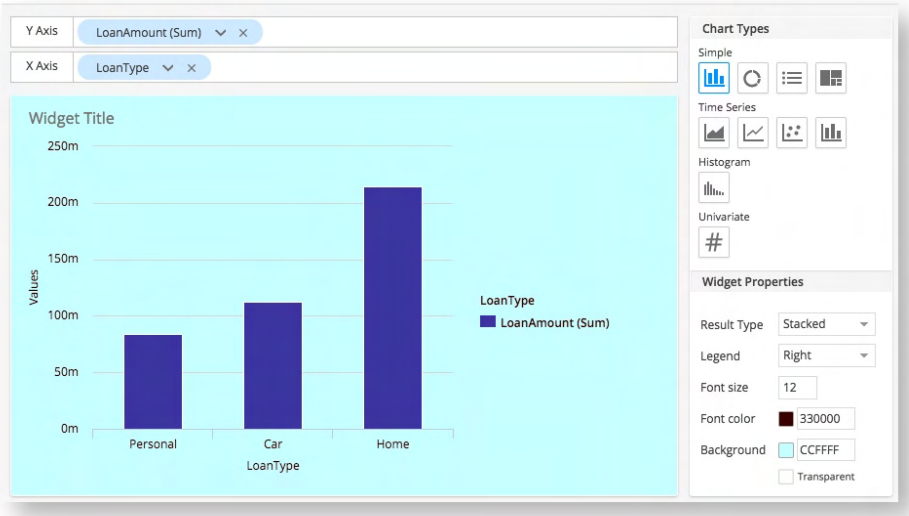
1. From the Analytics Search page using **Drag and Drop Search** (Basic) Mode, select the Visualization tab.
2. Click **Add Widget > Custom Widget Builder**.



3. To add a field to a widget, drag the field from the **Fields** panel on the left to the Y-axis or X-axis on the right. The available fields are based on the fields returned by the search.



4. From the **Chart Types** panel on the right, click the widget type to use for the display. The widget is previewed in the main panel. You can experiment previewing which widget type is best for visualizing a particular data set.



- Usage for the X and Y axes depends on the chart type. The UI prevents you from adding more fields than the widget supports. Different widgets have different limits for the maximum number of fields permitted on each axis.
- For the time series widgets (area, line, scatter, and histogram), you can use the **Resolution** field that appears on the X-axis to customize the frequency displayed if you do not want the widget to automatically select a default resolution based on the current time range. This time resolution field is not counted toward the field limit. The granularity of resolutions that can be chosen depends on the Global time range.

Custom Widget Builder Data Type and Field Restrictions

Widget Type	Data Type	Number of Fields on Y-axis	Number of Fields on X-axis
Column (simple)	any	zero	up to two
		one	up to two
		two or three	zero or one
Pie	any	one	up to two
Table	any	one	up to ten
Tree Map	any	one	up to two
Time Series (Area, Line, Points, Column)	any	zero	one
		one	one

		two or three	zero
Percentile Histogram (Not available in Advanced Mode)	numeric	zero	one
Numeric (Univariate)	any	one	zero

Visualize JavaScript Errors

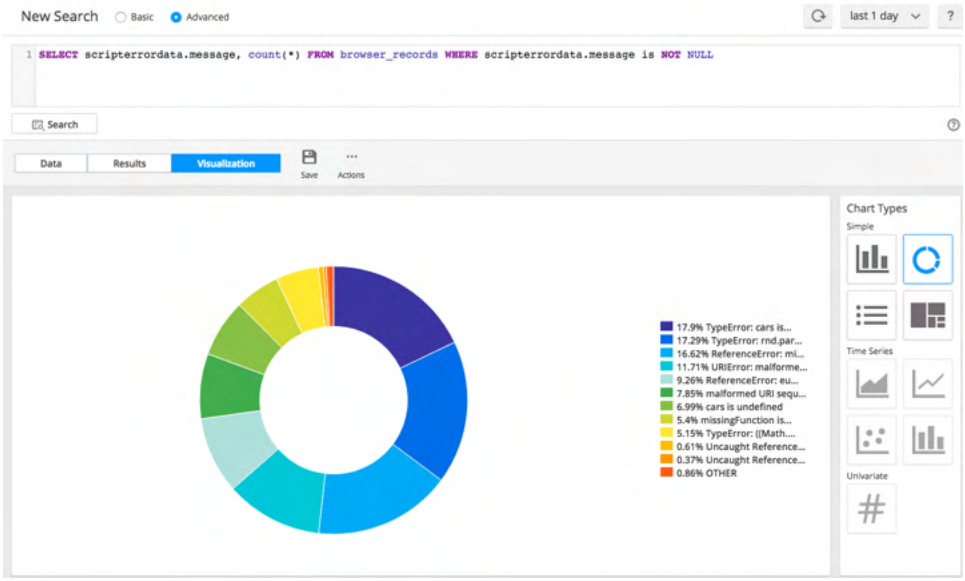
You can use Analytics to query your Browser Request event data and create a JavaScript exception and error summary dashboard.

This procedure assumes that you have configured and enabled the reporting of the JavaScript errors for Browser RUM and that JavaScript errors exist in the Events Service.

Using ADQL, you can search for the errors and then export a chart visualizing the results to a custom dashboard.

1. From the Analytics tab in the Controller UI, click **Add** to access the Analytics Search page.
2. Select Advanced mode and enter the following query:
   SELECT scripterrordata.message, count(*) **FROM** browser_records **WHERE** scripterrordata.message **IS NOT** NULL. The results are shown on the Visualization tab similar to the following:

3. Select the chart type you prefer to visualize the results.



4. From the **Actions** menu, click **Add to Dashboard** and either create a new dashboard or add it to your existing dashboard.
   Now you can customize the dashboard as needed with labels or other widgets and data.

You can also leverage additional fields such as scripterrordata.origin or pagename to group the errors together. The additional fields are:

- scripterrordata.linenumber
- scripterrordata.origin
- scripterrordata.timestamp

New Search ○ Basic ● Advanced

last 1 day

```
1 SELECT scripterrordata.linenumber, scripterrordata.message, scripterrordata.origin, count(*) FROM browser_records WHERE
scripterrordata.message is NOT NULL
```

Search

Data | Results | Visualization

Save Actions

scripterrordata.linenumber	scripterrordata.message	scripterrordata.origin	count(*)
24	TypeError: rnd.parseInt is not a function	http://127.0.0.1/eum-js-error.js	250
6	TypeError: cars is undefined	http://127.0.0.1/eum-js-error.js	244
6	Uncaught TypeError: Cannot read property '4' of undefined	http://127.0.0.1/eum-js-error.js	2
30	malformed URI sequence	http://127.0.0.1/eum-error.html	128
30	URIError: malformed URI sequence	http://127.0.0.1/eum-error.html	42
30	Uncaught URIError: URI malformed	http://127.0.0.1/eum-error.html	3
19	cars is undefined	http://127.0.0.1/eum-error.html	114
19	TypeError: cars is undefined	http://127.0.0.1/eum-error.html	48
19	Uncaught TypeError: Cannot read property '4' of undefined	http://127.0.0.1/eum-error.html	2
48	ReferenceError: eum is not defined	http://127.0.0.1/eum-error.html	151
48	Uncaught ReferenceError: eum is not defined	http://127.0.0.1/eum-error.html	6
29	ReferenceError: missingFunction is not defined	http://127.0.0.1/eum-js-error.js	148

1 / 1 50 items per page

1 - 25 of 25 items (1000 max)

Chart Types

Simple

Time Series

Univariate

Create Analytics Metrics From Scheduled Queries

Related pages:

- Using Analytics Data
- ADQL Reference
- Create and Manage Custom Roles

This page describes how to create analytics metrics from scheduled queries.

If you want to execute an analytics search repeatedly to monitor its value, you can create a metric from the search. The search will execute once per minute and report the results as a metric. You can create alerts on the metric in the usual way using Health Rules to trigger Policies and Actions. The analytics metric list is searchable.

You can create metrics for all Analytics event types - Browser, Mobile, Transaction, Log, and Custom.

Although the metrics calculated by scheduled queries may have decimal point precision, we only save and report whole numbers.

You can create a metric from an analytics search for the functions shown in this table. See Metric Data Resolution over Time.

Function	Metric Rollup Type	Example
count(* | field_name)	sum	SELECT count(activeMacUsers) FROM dummyTransactions
distinctcount (field_name)	average	SELECT distinctcount(activeMacUsers) FROM dummyTransactions
sum (numeric_field_name)	sum	SELECT sum(responseTime)where userExperience = "NORMAL" FROM transactions
avg (numeric_field_name)	average	SELECT avg(responseTime) FROM transactions
min (numeric_field_name)	average	SELECT min(responseTime) FROM transactions
max (numeric_field_name)	average	SELECT max(responseTime) FROM transactions
(exp1)/(exp2)	average	SELECT (count(activeMacUsers)) / (avg(activeWindowsUsers)) FROM dummyTransactions SELECT (avg(responseTime)*2)/((avg(segments.transactionTime)+3)/2) FROM transactions SELECT (sum(responseTime)*2)/(filter(sum(responseTime), where userExperience = "NORMAL") + 0.5) FROM transactions Each side of the expression must be enclosed in parenthesis.

You must have **Manage Metrics** permission and View access to Analytics to create metrics from analytics searches. See Analytics and Data Security and Transaction Analytics Permissions.

Create an Analytics Metric

1. Working in the **Analytics** > **Search** editor, set the search criteria to select the appropriate data.
2. Click **Actions** > **Create Metric** in the search action toolbar.
3. In the pop-up window, give your metric a name and a description.
   The name determines how the metric appears in the Metrics panel and the Metric Browser. Keep in mind:
   - Metrics are truncated to whole values. So values less than one are truncated to zero. To work around this, you can multiply a numeric_field_name by a factor of 10, 100, 1000, and so on, depending on the level of decimal accuracy you need. We recommend that you add this factor into the metric display name so other users can understand what the metric represents, for example, Display Name = Metric (Factor 1000). In cases where the value is a percentage, and you convert by multiplying by 100, you probably don't need to put the factor in the metric name because it would be implicit (Metric %).
   - Math operations are only supported inside the aggregation function. For example, `count(numeric_field_name * 10) from transactions.`

Metric Timestamps

Timestamp metrics always aggregate events from the current minute because metrics are created for events published in the last minute.

If you have specified a value for the `eventTimestamp` field, the timestamp value in the new metric uses an aggregation of events from the pickupTimestamp field.

For example, you set an `eventTimestamp` for an event on a previous day. You then create a metric for the event today. The metric does not use your explicit `eventTimestamp` because it is no longer from the current minute. The metric aggregates events using `pickupTimestamp`. This ensures that the metric timestamp is based on the current minute because `pickupTimestamp` cannot be explicitly changed.

See Analytics Events API.

Correlate Metrics from Multiple Events

When you create a query, AppDynamics produces a holistic metric. Because the queried data comes from different events, the Events Service has to retrieve this segmented data and stitch it together to produce a single metric. The Events Service waits for correlated events to arrive, where the maximum wait time depends on the event type:

- Mobile and Browser Session Events - two minutes.
- Transaction Events - ten seconds.

The wait time resets when the Events Service finds a correlated event.

If an event arrives after the maximum wait time, the event is not stitched together correctly in your query. For example in the following query `fieldA` and `fieldB` are separate but correlated events:

```
select count(*) from transactions where fieldA = "foo" and fieldB = "bar"
```

The Events Service finds `fieldA` and waits 10 seconds for a correlated event. If `fieldB` arrives 15 seconds after `fieldA`, the Events Services does not correlate both events. The count metric is now inaccurate because the events are not connected.

Monitor Analytics Metrics

For performance reasons, the scheduled queries used in analytics metrics can be disabled as follows:

- Queries are disabled if calling the Events Service results in any errors for ten consecutive times.
- Queries are not disabled if the Events Service is not reachable from the controller, such as during maintenance and upgrade windows.
- When the Controller is restarted, the queries which were previously disabled by the system due to consecutive failures are re-enabled.
- User-disabled queries remain disabled even after the system is restarted.

To monitor the metric:

1. Click **Metrics** in the left navigation bar.
2. Search, edit, enable, disable, or delete metrics from the **Metrics list**.

   ⚠ You can edit only the description of the selected metric.

3. To activate or deactivate metrics in a batch, select the corresponding rows in the grid and click the **Enable** or **Disable** button in the toolbar. Use Shift+ click or Command + click rows to select multiple rows.

   ⚠ If you see a status that says "Disabled due to repetitive failures", you can re-enable it by clicking **Enable**.

4. To see the metric in the Metric Browser, click the **Metric Browser**. For more details, see the section "*Metric Data Point Details*" in Metric Browser.

Business Journeys

Deployment Support

Related pages:

- Configure Business Journeys
- View Business Journeys
- Create an Example Business Journey

This page provides an overview of business journeys in AppDynamics.

Many industries have complex workflows and user journeys that span multiple transactions and event types, such as logs and End User Monitoring (EUM) data. These workflows typically manifest over long periods and cannot be measured using transactions alone. Business Journeys are a way to monitor and correlate the linear data flow across multiple event sources and track the total time for defined business workflows.

Typical multi-step workflows from different industries include:

- Payment transfers, credit card approval, and loan approval in financial services industries
- Cellphone activation and data recharge (pre-paid) in the Telco sector
- Insurance application through policy approval and insurance claims approval for insurance companies

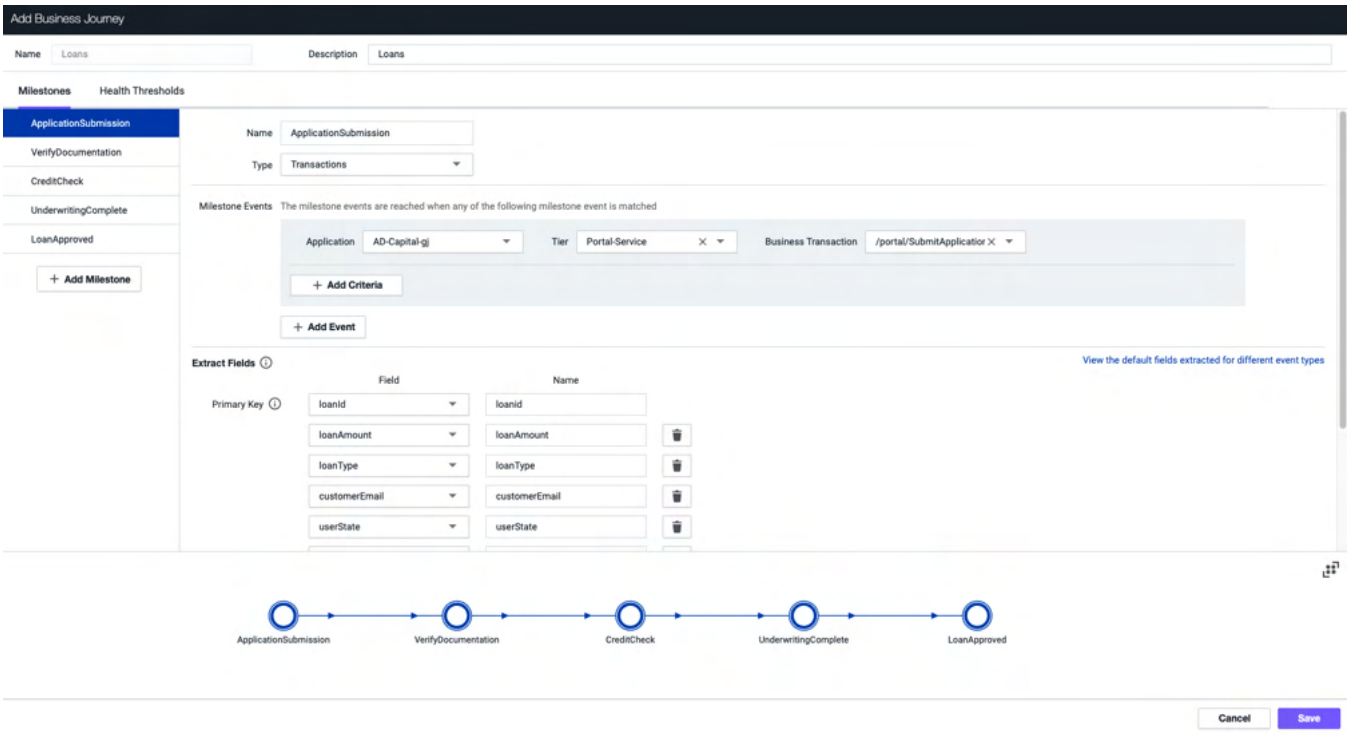
About Business Journeys

An AppDynamics Business Journey is a composite event type based on defined workflows. A Business Journey can include events from multiple analytics event types, such as logs, business transactions, custom events, and EUM data.

You will need at least one unit of the Transaction Analytics license for Business Journeys, even if none of the milestones use any of the Transaction

Business Journeys track linear processes. Branching, repetition, and loops are not allowed.

The following image depicts an example Business Journey definition.



You create a Business Journey, as shown above, by defining the following:

- Milestones and the events comprising the milestones. Milestones are the steps in your business workflow.
- Fields to be captured from each milestone event. A field indicates a category of information pertaining to the event.
- The primary key field that uniquely ties the events together. A primary key correlates the milestones for your Business Journey. The primary key value must be present in each milestone event.
- Additional fields that allow you to segment on different dimensions of the business workflow, such as loan types or payment amounts.
- Health thresholds for monitoring the Business Journey performance.

Application Analytics starts collecting the Business Journey events after you have defined and enabled a Business Journey. Analytics does not go back in time to collect Business Journey composite events. To define specific Business Journeys, you may need to configure the collection of additional fields from your source analytics events. See Data Prerequisites for Defining Business Journeys for more information.

Once the Business Journey events are being collected, you can view them from the Analytics Search UI in the same manner as other Analytics events. Because Business Journey events are, by their nature, potentially long-running processes, useful data might take some time to appear in the Analytics events list. Business Journey events are reported even when only one milestone is completed and the event is updated over time as subsequent events complete.

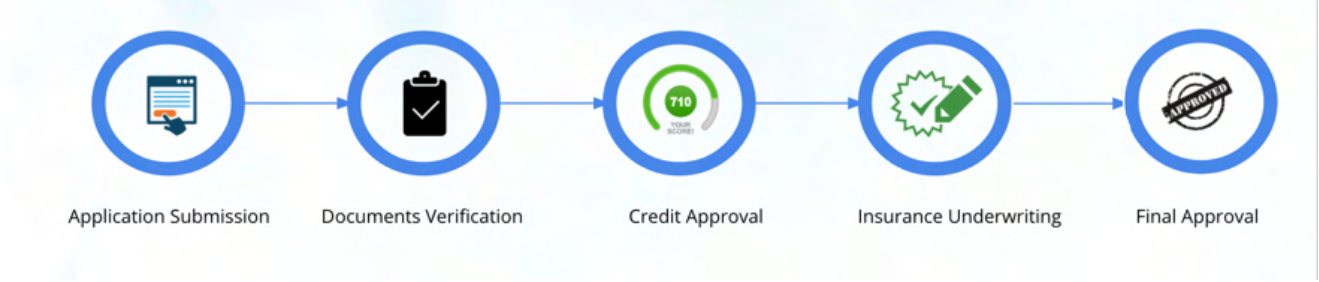
If you are already familiar with the fundamental concepts in Business Journeys clearly, proceed to either Configure Business Journeys or View Business Journeys as desired.

Business Journeys Milestones

A milestone is an event marking a significant stage in a business workflow. For example, in a loan application, the first milestone might be a user submitting a loan application. The second milestone might be document verification, followed by credit approval, insurance underwriting, and finally, loan approval. All milestones are mandatory; do not duplicate milestones.

The events in milestones are sequential. You need to create each milestone in the order that the event occurs. For example, in a loan Business Journey, a credit approval milestone cannot precede the loan application milestone.

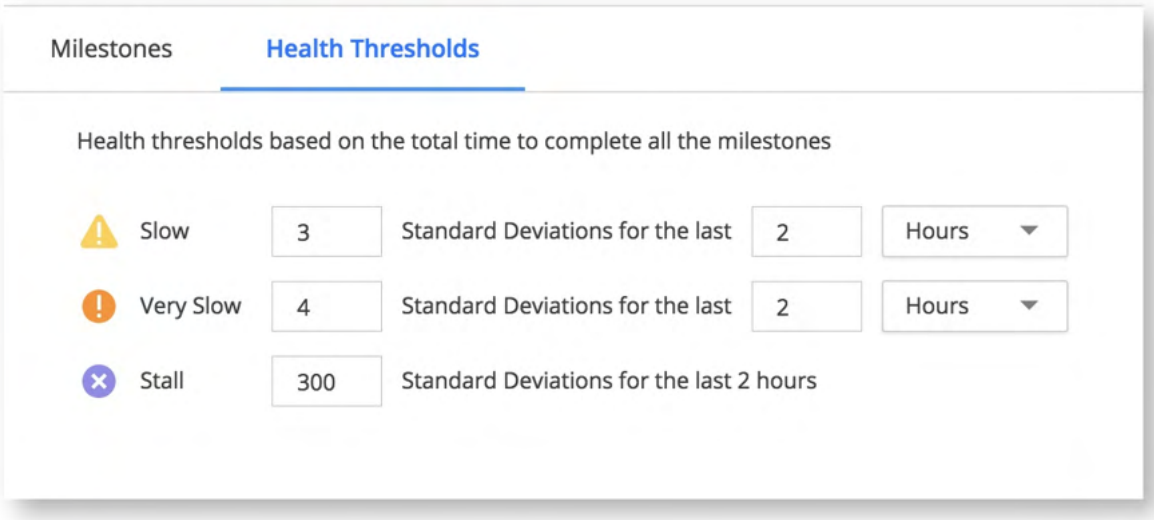
In each Business Journey milestone, you can add filters to specify the events that constitute a milestone and add fields to capture additional information about the workflow. You can use these events and fields to run ADQL queries and narrow down your searches.



As you create milestones, a flow map detailing the business workflow is automatically displayed in the Business Journey page. For more information, see Enable Flow Map.

Business Journeys Health Thresholds

A threshold is a boundary of acceptable or normal performance. Business Journey authoring provides default health thresholds against which it compares the performance of the business workflow. The health thresholds are based on the total time to complete all the milestones in your Business Journey. The threshold values are calculated by determining the standard deviation from the simple moving average over an interval of time. The default time interval is two hours. This means that if the average end-to-end time for the last two hours is N milliseconds, and if the Business Journey takes time equal to the standard deviation over N milliseconds (ms), it violates threshold.



Consider a simple moving average that is 1500 ms with a standard deviation of 100 ms. If you set the threshold to three, it means that the threshold is three times the standard deviation. In other words, a transaction that takes more than 1500 + (3*100) or 1800 ms violates the health threshold.

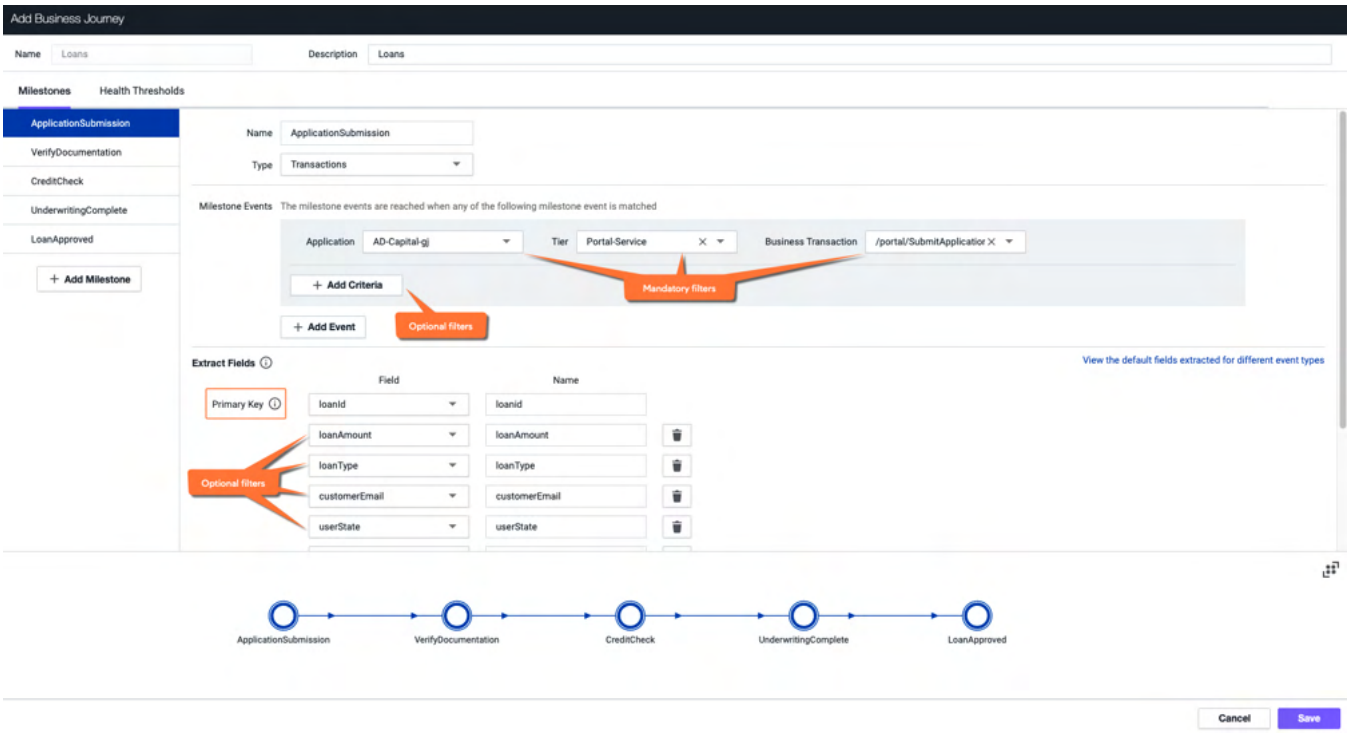
Before configuring the health threshold for a Business Journey definition, consider the following:

- A user experience (normal, slow, very slow, stall) is assigned for the Business Journey event when a value for `totalTime` is present and is greater than zero. Total time for the event could end up less than zero if the milestones are defined in an incorrect order. This can also happen even after the order is correctly defined, but the milestones are not in time order such that an early milestone has a future date compared to a later milestone.
- The total time for the Business Journey is calculated when both the first and the last milestones events have been reached. Therefore, the user experience cannot be determined and displayed until total time for the Business Journey is available.
- When a milestone encounters an error, for example, a transaction event where the `requestExperience` is Error, the user experience for the entire Business Journey event is marked as an Error. In this case, the total time is not available.
- Typically when a milestone has an error, the entire Business Journey workflow does not complete. However, if for some reason, all the milestones complete even when milestone errors exist, the total time for the Business Journey is available. However, in this case, we will retain overall user experience as an error and not change to Normal, Slow, and so on.
- After a Business Journey is initially enabled, Business Journey events are generated. The standard deviation is calculated every five minutes, so the first few Business Journey events will not have a user experience assigned to them.
- The minimum duration over which standard deviation is calculated is 30 minutes.

Filters and Fields

Filters allow you to extract event data for a select milestone based on selected criteria. For example, loan applications submitted in California. Filters define the scope of a milestone by limiting associated events. Fields extract additional relevant information from each milestone.

The mandatory filters are populated with the last 15 days of data. If no analytics data has been collected during the last 15 days, the drop-down menus for the mandatory fields will be empty. Besides the mandatory filters for each event type, Analytics allows you to optionally select additional filters by using the **Filter By** option. All the fields that are part of the filters and the additional filters that you optionally choose are automatically collected. You can use them to run ADQL queries later on.



A field indicates a category of information pertaining to the event. The **Extract Fields** section collects additional relevant information from each milestone. You can use data to query desired events and visualize matrices. The fields are categorized into mandatory and optional fields. The only mandatory field you must enter is the primary key. It is the unique identifier that ties milestones together.

In addition to the primary key, you can optionally add additional fields. These fields provide a context for your Business Journey definition. For example, loan amount, loan type, and customer name, customer id, and so on provide further details about a loan application. You can use these field values to slice and dice business transaction data later on. You must provide unique fields for each milestone. Validation fails if you create more than one milestone with the same set of fields or with the same set of mandatory and optional filters.

For each milestone, a field called `milestoneReached` is added to the associated event. The value is true if the milestone has been reached, otherwise it is null. You can use this field to query all events for which a milestone has been reached. You can also use `milestoneReached` to build a funnel widget that shows drop-offs for each milestone. The milestone name is appended to the field name as follows:

- *`milestone1_name`*`.milestoneReached`
- *`milestone2_name`*`.milestoneReached`

Likewise, the default fields and filters are also appended with and namespaced by milestone name.

At the Business Journey level, a field named `completed` tracks the status of the workflow using the boolean values. True represents a completed Business Journey, whereas False indicates that the journey is in progress or is incomplete. Use the Analytics Search UI or an ADQL query for the desired Business Journey with the query string `completed` to view the status of the associated events.

Milestone Timestamps

The `totalTime` and `timeTaken` fields represent the time it takes to reach a milestone. Timestamp fields are empty when the Business Journey is partially complete. As each milestone completes and the event data is captured in the Business Journey event, the total time is updated. The total time is calculated by subtracting the timestamp for the first event from the timestamp for the final event.

Timestamps are not exact fields. AppDynamics cannot calculate the exact time of events in a milestone because the event may occur at any point in method. For example, in the loan Business Journey above, the Credit Approval milestone may come from a log event. Your system may trigger a log at any point in the credit approval process. However, the start time of the Business Journey milestone can only be detected when the log event reaches AppDynamics. You may notice a slight difference in actual time due to this behavior.

Business Journey and Milestone Metrics

The table below lists the metrics that are calculated out of the box for Business Journeys and milestones, which are the only metrics supported by Business Journeys. You cannot use metric queries created from an ADQL search for Business Journeys.

	Metric	Description

Milestones	`Average time taken`	Average of time taken to reach the current milestone from the previous milestone. Not applicable for the first milestone.
	`Calls per minute`	Total number of Business Journeys that reached the milestone in the last one minute and started in the last 24 hours.
Business Journeys	`Average total time`	Average of time to reach to completion for all Business Journeys that started in last one day and completed in the last one minute. Calculated taking the average value of `totalTime` field stored in all Business Journey events.
	`Calls per minute`	Total number of Business Journeys that started in the last one minute. A Business Journey starts when AppDynamics detects the first milestone.

Mandatory Fields

Mandatory filters, mandatory fields, and fields extracted by default depend on the event type as shown in the following table.

Event Type	Mandatory Filters	Mandatory Fields	Fields extracted by Default

Transactions	Application Name
	Business Transaction Name
	Tier Name
		Primary Key	`eventTimestamp`
			`requestGuid`
			`requestExperience`

Logs	Source Type	Primary Key	`eventTimestamp`
			`logLevel`
			`logType`

Browser RUM	AppKey
	Page Name
	Page Type
		Primary Key	`eventTimestamp`
			`cguid`
			`pageexperience`

Mobile RUM	AppKey
	Network Request Name
		Primary Key	`eventTimestamp`
			`cguid`
			`networkrequestexperience`

Custom Events	Source Type	Primary Key	`eventTimestamp`

For custom events, you must include a date in the `eventTimestamp` field for the Business Journey to register the event. See Analytics Events API for more information.

Configure Business Journeys

Related pages:

- View Business Journeys
- Create an Example Business Journey

This page describes how to configure Business Journeys.

Before You Begin

Before creating Business Journeys, ensure that the feature is enabled in your environment. You also should review data prerequisites, validation rules, and limitations for names and Business Journey definitions.

Enable Business Journeys

By default, Business Journeys is enabled on SaaS environments. However, the feature is disabled for on-premises installations. To enable Business Journeys, an Administrator needs to configure certain properties in **Settings**.

Before starting the Event Service:

1. Open the `events-service-api-store.properties` file.
2. Change `ad.bizoutcome.enabled=false` to `true`.

The page displays a flow map detailing the business workflow across milestones. You can hide the flow map by changing the value of the `analytics.business.outcomes.flowmap.enabled` property to `false` in **Settings**.

Enable Flow Maps

As you create milestones, a flow map detailing the business workflow displays on the page. Selecting a Business Journey prompts a new window with the enabled flow map. This behavior is controlled by the `analytics.business.outcomes.flowmap.enabled` property in **Settings**. Setting its value to `true` indicates that flow maps are enabled for Business Journeys.

Data Prerequisites for Defining Business Journeys

Analytics creates Business Journey composite events by collecting events and fields that you have already captured as analytics data. In addition to built-in analytics data sources, custom event data can also be used to create Business Journeys. As you define the milestones in the Business Journey, you specify which events to use and which fields to extract from the events. You can extract fields collected by default or custom fields that you have configured for collection. You can also add milestones with custom events created from Analytics API Keys.

You need to know the starting event, ending event, and clearly-defined steps in between. To collect additional fields in the events being captured by Analytics, see:

- Analytics Custom Events Data
- Data Collectors
- Collect Business Data From SQL Calls
- Configure Log Analytics Using Source Rules

Restrictions and Caveats

- Business Journey names are case-insensitive and are stored as all lowercase. This means that two definitions named `application` and `Application` are considered a duplicate.
- The **Event Type** drop-down displays the lowercase definition name.
- Use the lowercase name for advanced queries.
- The primary key must be unique and not null. If the primary key holds null values, the underlying milestone events will not be stitched together to form a Business Journey event.
- Choose the primary key carefully; in particular, the cardinality of the values. The primary key is expected to uniquely identify and join milestone events together to form a meaningful business workflow. If the milestone events are misidentified by the primary key, inaccurate milestone events will join and produce faulty results.
- The primary key name cannot be changed unless the Business Journey definition is in the draft state.
- The Business Journey definition name cannot be changed after the definition is deployed. You can rename it while the definition is in the draft state.
- If a field related to a custom event is renamed, update the Business Journey definition created from the custom event for the renamed field to be correctly extracted.

Validation Rules

This section describes certain rules to validate Business Journey definition and naming convention for Business Journeys, milestones and extracted fields. You can save your work as a draft without invoking validation. When you are ready, you can use **Validate and Save** to check that your definition is acceptable. Saving the Business Journey definition does not enable the Business Journey. Data is not captured until you actually enable the Business Journey. See Business Journeys Life Cycle.

Validation Rules for Names

This section describes the validation rules for naming Business Journeys, milestones, and extracted fields. Validation rules are not applied when you save your Business Journey definition as a draft. Validation is only invoked when you use **Validate and Save**. The validation rules include the following:

- Names of Business Journeys, milestones, and extracted fields must have an alphanumeric string containing a-z, A-Z and 0-9. The only special character allowed is the underscore _.
- Special characters such as spaces, hyphens, dashes are not allowed in the names.
- Primary Key fields must have the same name (label) in all milestones in a Business Journey definition. The field name itself may vary in different event types.
- The primary key name cannot be changed unless the Business Journey definition is in the draft state. See Business Journeys Life Cycle for more information.
- The following reserved field names cannot be used to name milestones or fields in your Business Journey definition.
    - `pickupTimestamp`
    - `eventTimestamp`
    - `totalTime`
    - `userExperience`
- You cannot name milestones or field names with ADQL keywords. See ADQL Queries.

Validation Rules for Business Journey Definition

A Business Journey definition must contain:

- A minimum of two milestones.
- A unique set of filters. More than one milestone with the exact same set of required and optional filters is not allowed in the same definition.
- Unique milestone names.
- Extracted field names must be unique (except the Primary Key field).
- There must be one Primary Key field per milestone.
- Slow threshold duration must be greater than 30 minutes.

Create a Business Journeys Definition

1. In the Controller UI, navigate to **Analytics** > **Business Journeys**. A list of existing Business Journeys (if any) displays.
2. To define milestones, click **+ Add Milestone.**
3. Type a name.
4. From the **Type** dropdown, select the source of the analytics data, such as Transactions, Logs, Browser Requests, and Mobile Requests.
5. The **Milestone Events** section displays mandatory filters based on your selected data source. The fields vary based on your data source type.

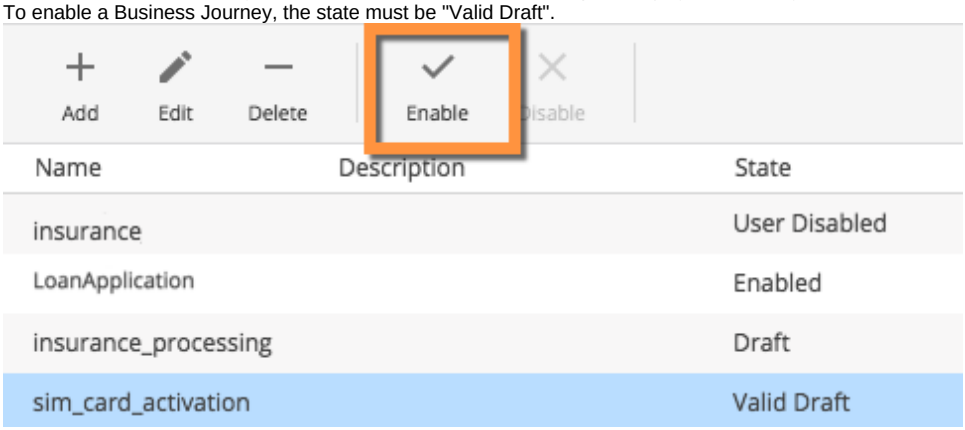
   For Transactions, specify the Application, Tier, and Business Transaction for each milestone you intend to create. For Analytics, the tier represents a Java Virtual Machine (JVM) service, such as an authentication service, in your application environment. Business Transaction is the cross-tier processing path representing the request for a service provided by the application.
6. You can optionally click **+ Add Criteria** to add additional filters based on your data source type.
7. You can define multiple milestone events of the same event type. Click **+ Add Event** to create another milestone event.
8. In the **Extract Fields** section, define your primary key field and name to specify which fields will be captured from the milestone event. You can click **Auto Fit** to auto-organize the milestone events or organize the events freely.

   Once you name the primary key, you can not change the name unless the definition is in the draft state. See Business Journey Life Cycle.
   The primary key field is not populated for subsequent milestones because the primary key might have different field names in the event type for different milestones.

   > Fields that are extracted by default in Business Journeys are not shown.

9. You can save the Business Journey with either:
    - **Save As Draft:** saves the definition in draft form. No validation is performed and you can return to the definition to complete it at a later time.
    - **Validate and Save**: Performs necessary validation. Use this option when you have completed all milestones. The definition is saved, but is not enabled.
10. In the **Health Thresholds** tab, you can specify the values that determine acceptable performance.
    See Business Journeys Health Thresholds to learn how the user experience value is calculated. When you finish creating the Business Journey definition, you are ready to enable it.

11. Select the Business Journey from the list and click **Enable**. The option is greyed out until you select a valid draft Business Journey from the list. To enable a Business Journey, the state must be "Valid Draft".



Business Journeys Life Cycle

This section provides information about the Business Journeys life cycle.

Draft

You might not have all the required details at the beginning and therefore, creating the definition may involve multiple temporary versions. Use **Save As Draft** to capture your initial, incomplete definition and place your Business Journey into a Draft state.

- No analytics data is processed and no Business Journey events are created in draft state, so there is nothing to query.
- The primary key can be changed when the Business Journey is in draft state.
- The draft state is less restrictive with respect to validations and supports most update operations.

Valid Draft

Once the definition has the required details, use **Validate and Save**, which transitions the definition into the Valid Draft state. You can still modify the definition in this state. Business Journey events are not captured until you enable the definition.

Enabled

When you are ready to process data, use the **Enable** action. This transitions the definition to the Enabled state. Most definitions will spend their lifetime in the Enabled state.

- Configuration changes are synced to the Analytics Servers every two minutes. Therefore, once you enable a Business Journey definition, there could be a delay of up to two minutes for events to be generated.
- Incoming events are processed, composite Business Journey events are created and stored. You can query for the Business Journey events.
- You can modify the definition in the following ways:
    - Add, rename, or delete fields for extraction (non-default fields)
    - Add, rename, or delete milestones

User Disabled

At some point, you may want to disable the Business Journey. The **Disable** action transitions the Business Journey definition to the User Disabled state. In this state, no incoming events are processed. Existing events can still be queried.

You can update the definition in the following ways:

- Add, rename or delete fields for extraction (non-default fields)
- Add, rename or delete milestones

Deleted

When you no longer need the Business Journey definition, use the **Delete** action. This is the final phase of the definition's lifecycle. In this state, no incoming events are processed for it. Existing Business Journey events are not available to query.

View Business Journeys

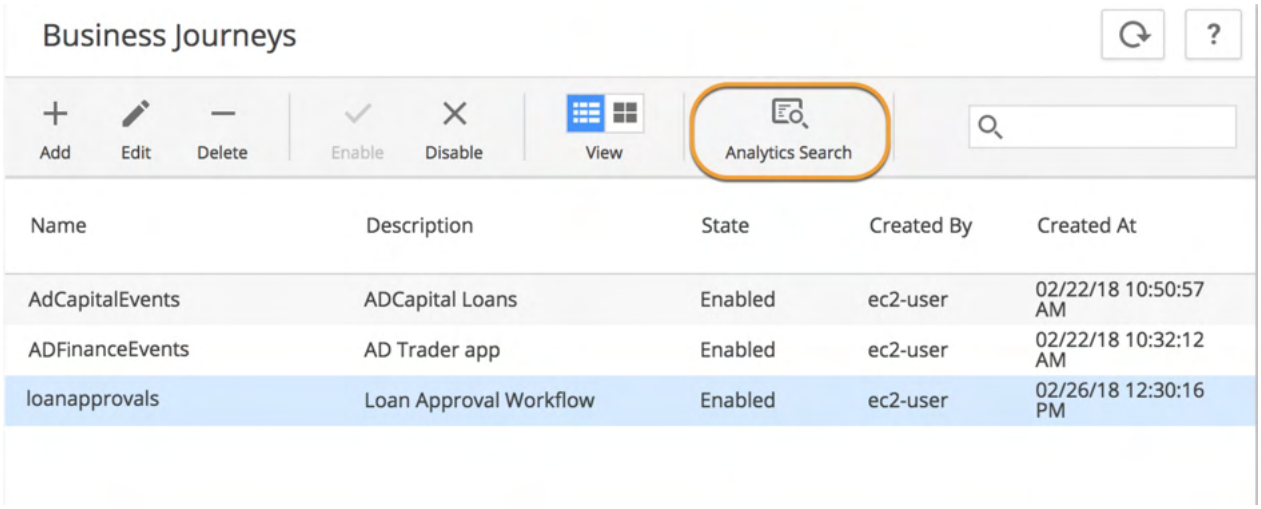
Related pages:

- Search Analytics Data
- ADQL Reference

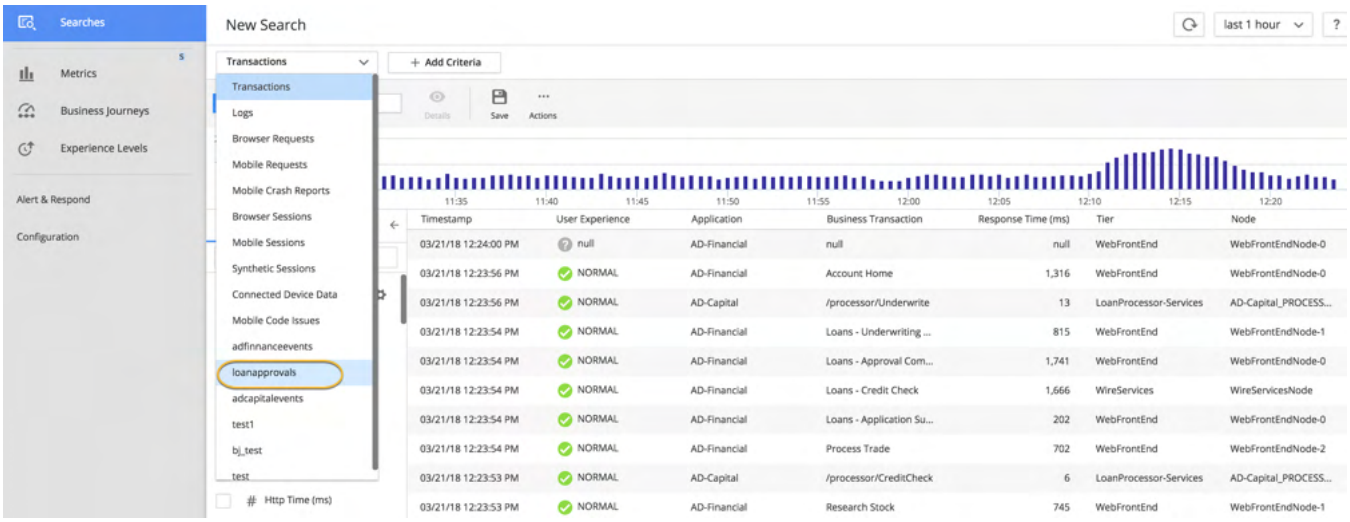
View Business Journey events from the Analytics Search UI in the same way as any other event type. Alternatively, use the **Analytics Search** button on the Business Journey UI.

Search for Business Journeys

Use the **Analytics Search** button in the Business Journeys UI for one-click access to the underlying events. Selecting an enabled Business Journey and clicking **Analytics Search** takes you directly to the Searches page in Basic mode. **Analytics Search** button is unavailable for definitions in Draft and Valid Draft states.

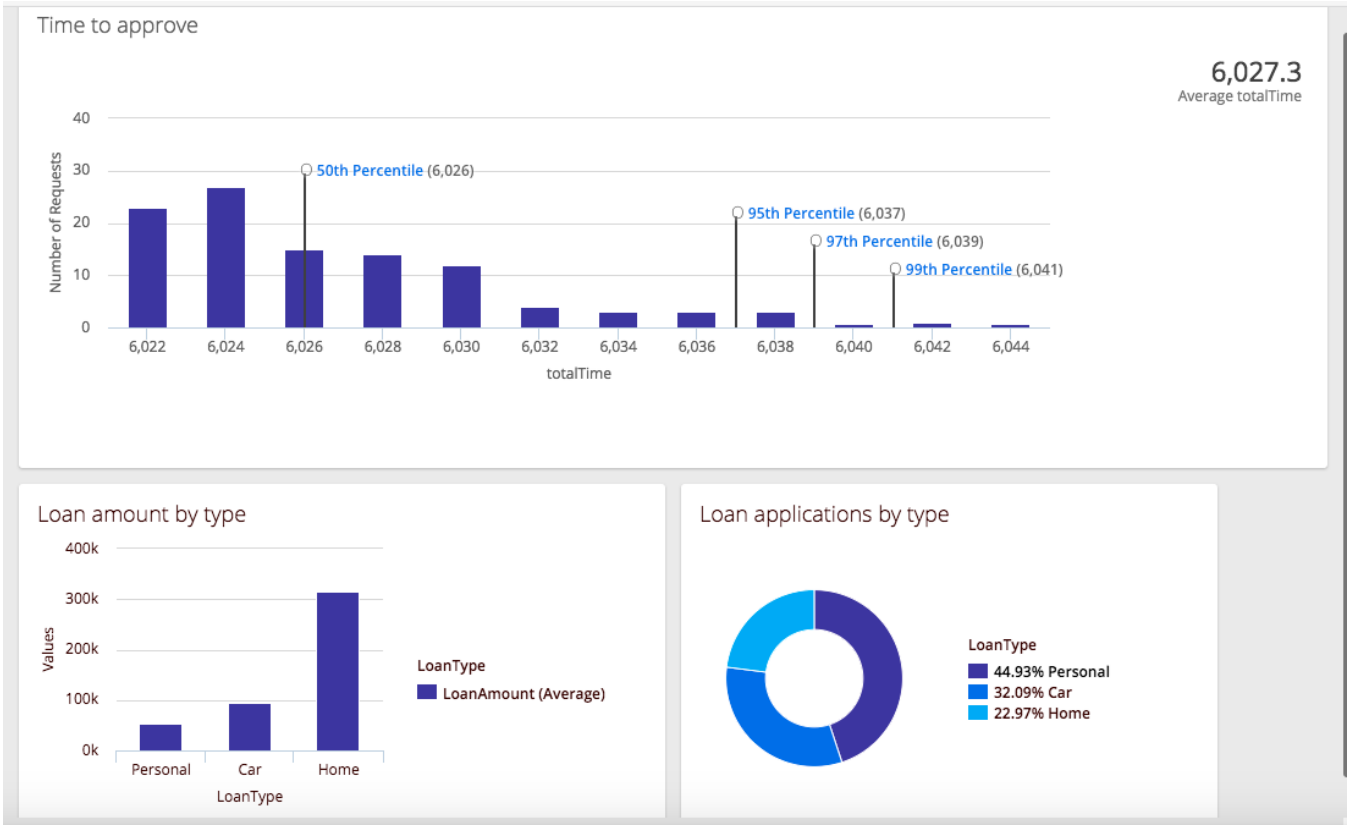


Alternatively, access the Analytics Search UI and use the event type dropdown to find your Business Journey event. The Business Journey name identifies the event type in the Search UI similar to custom events. In this example, it is "loanapprovals".



You can search the Business Journey data and create visualizations in the same fashion as for any other analytics event type.

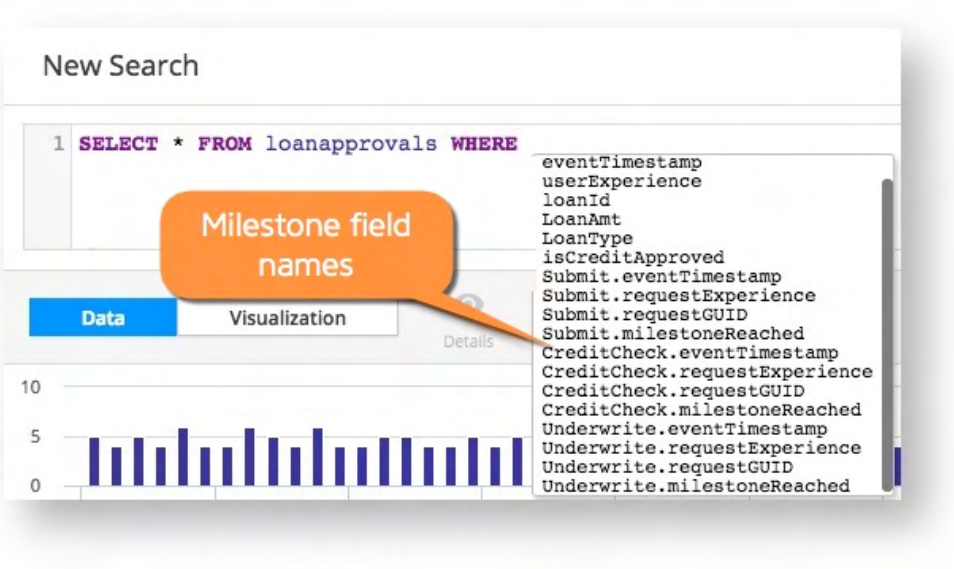
Business Journeys also gives you the ability to perform analytics, slice and dice data, and create widgets. For example, using the loan approval Business Journey, visualize percentile values of approval time, loan amount by type, and loan applications by type as follows:



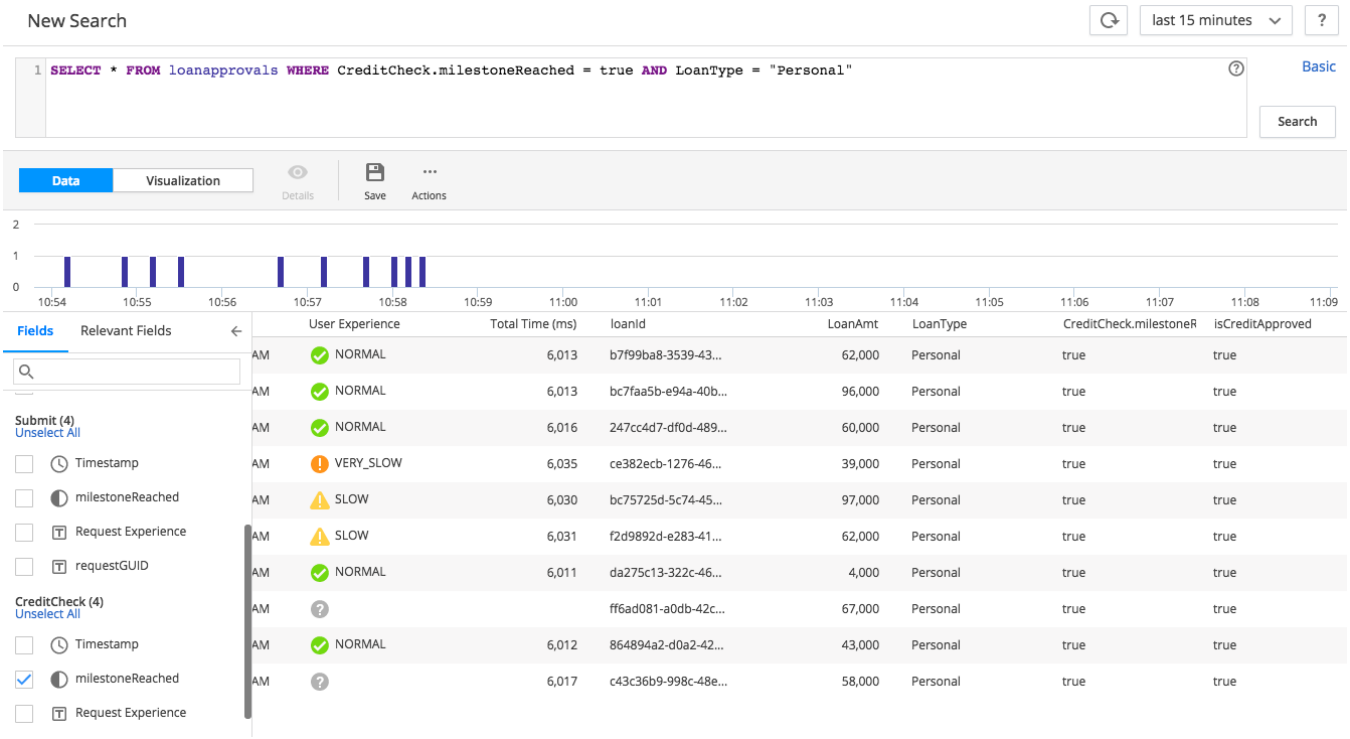
The 50th Percentile line indicates that 50% of the loan approvals occurred below this line. Similarly, each percentile line represents the corresponding percentage of loans approved below that line.

Business Journeys ADQL Queries

To use ADQL to query Business Journey events, you need to use the full namespaced field name for each milestone. The field names are constructed by appending the milestone name to the field. When constructing queries with ADQL in the Search UI, you must use the full name of the field. The following image shows constructing a sample ADQL query.



A sample ADQL query for the loanapproval Business Journey is:



Business Journeys Metrics

Analytics generates out-of-the-box metrics at the Business Journey definition and milestone levels. You can view these metrics in the Metric Browser. The metric values are reported per minute. For example, an observed value of 5 indicates that it's the count of the records received in the one-minute interval. If the number of records reported in the last five minute is 25, the Metric Browser reflects an approximate observed value, that is 5. To calculate metrics, Analytics adds two fields named `timeTaken` and `totalTime`.

The `timeTaken` field is added to the associated event for each milestone. This field represents the time taken for an event to reach a milestone from the previous milestone, and therefore, the first milestone will not have this value.

The `totalTime` field is added to the associated event for each Business Journey definition. `totalTime` represents the time for an event to cover all the milestones comprising the Business Journey. It is the time an event takes to reach the last milestone from the first one. If the first or the last milestone is missing, `totalTime` is not calculated for that event. For example, if the loan application is rejected in the CreditCheck stage, the workflow never reaches the approval stage. In this case, `totalTime` is not determined.

By using the above-mentioned fields, the following out-of-the-box metrics are generated.

Milestone Level

Average timeTaken: Average of the values measured for the `timeTaken` field. Evaluates the events that started in the last one day and reached the milestone in the last one minute.
Calls per minute: Number of events that started within the last one day and reached the selected milestone in the last one minute.

Definition level

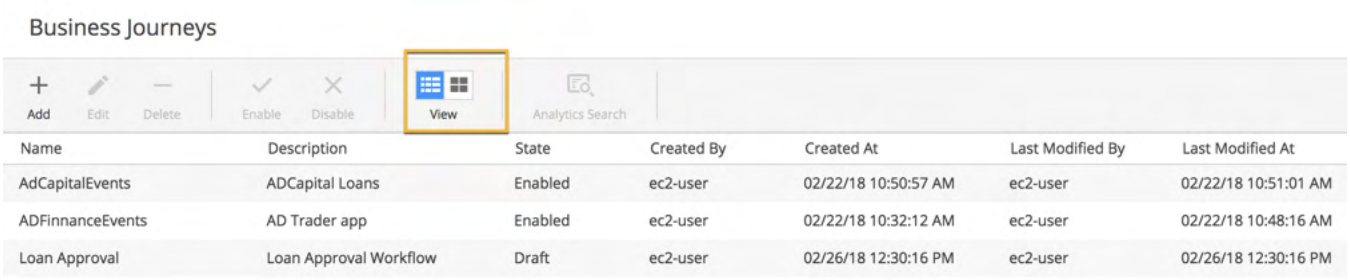
Average totalTime: Average of the values measured for the `totalTime` field. Evaluates the events that started in the last one day and reached the first milestone in the last one minute.
Calls per minute: Number of Business Journey events measured per minute. This number is equal to the calls per minute value of the first milestone. The value is calculated by evaluating the events that started in the last one day and reached the first milestone in the last one minute.

> The lookback period to generate out-of-the-box metrics is one day. The values for the metrics might not be accurate if events take more than a day to complete or to even reach the next milestone. This is due to the lack of sufficient event records for Analytics to generate correct values, leading to displaying incorrect values. For example, the average `totalTime` metrics will only be accurate if events finish within the lookback period, that is one day.

Note that the existing definitions will not have the out-of-the-box metrics created. You need to create new definitions to generate corresponding out-of-the-box metrics.

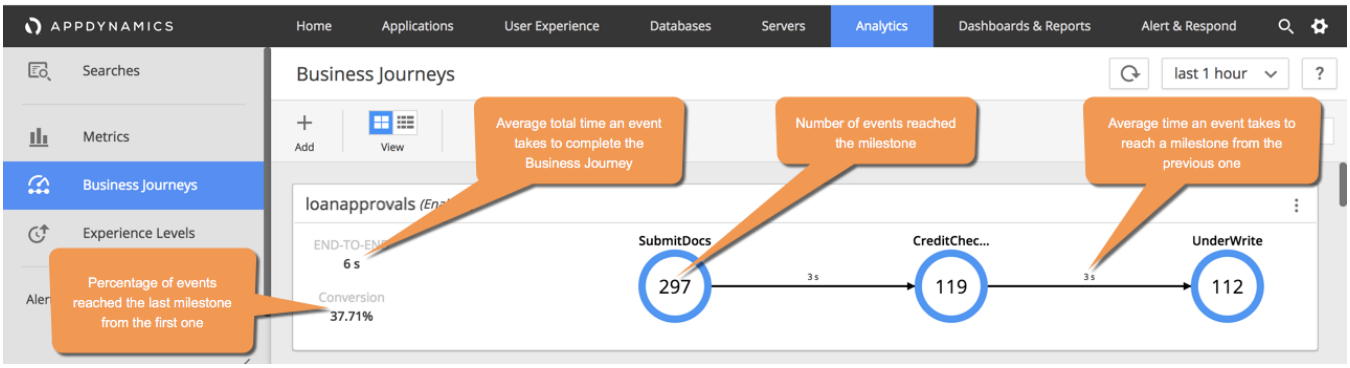
Business Journeys Dashboard

Analytics provides two Views for Business Journeys: Card View and Grid View. Business Journeys page defaults to the Grid View, but after switching to the Card View the UI makes it as the preference. The next time you log in, Business Journeys page defaults to the Card View. Use the **View** button on the toolbar to switch between the Views.



Name	Description	State	Created By	Created At	Last Modified By	Last Modified At
AdCapitalEvents	ADCapital Loans	Enabled	ec2-user	02/22/18 10:50:57 AM	ec2-user	02/22/18 10:51:01 AM
ADFinnanceEvents	AD Trader app	Enabled	ec2-user	02/22/18 10:32:12 AM	ec2-user	02/22/18 10:48:16 AM
Loan Approval	Loan Approval Workflow	Draft	ec2-user	02/26/18 12:30:16 PM	ec2-user	02/26/18 12:30:16 PM

The Card View displays an out-of-the-box dashboard for each Business Journey definition along with key metrics. The dashboard highlights the milestones associated with each Business Journey definitions and these metrics:



Dashboards for Business Journeys created prior to the release of Analytics 4.5 do not display the *Average time to reach the next milestone* metric. To workaround, edit the **Extract Field** section of one of the milestones and add the new field or remove an existing field. After the definition is saved Analytics starts generating the missing values. Within a few minutes, you should see values for *Average time to reach the next milestone* for the Business Journey you edited.

Time range menu determines what events are considered for calculating metrics. The events that are considered for calculating metrics are only those started in the selected time range.

- End-to-End: The average end-to-end time an event takes to complete the corresponding Business Journey. It is the average time taken by an event to reach the last milestone from the first one.
- Conversion: The conversion rate represents the percentage of events that reached the last milestone from the first one. It is calculated as (events reached the last milestone) * 100 / (events reached the first milestone).
- Average time to reach the next milestone: It is the average time an event takes to reach the next milestone from the current one. This value is represented by the number shown in between each milestone.
- Event count per milestone: It is the number of events that reached a particular milestone. This value is indicated by the number shown inside the circles representing each milestone.

Create Example Business Journeys

Related pages:

- Configure Business Journeys
- View Business Journeys

This page describes a loan application process, AD-Capital, to demonstrate the milestone workflow. Similar to a typical loan application process, the workflow in this Business Journey starts with application submission, followed by document verification, credit check, underwriting, and loan approval.

Milestones Data

The data for each milestone comes from different applications and event types, such as business transactions, logs, and end-user events.

In this workflow, the data associated with application submission comes from business transaction events while the document verification status comes from logs. Third-party service providers perform the credit check and underwriting. Then, the status is updated in logs and the loan approval status is updated in the transaction events.

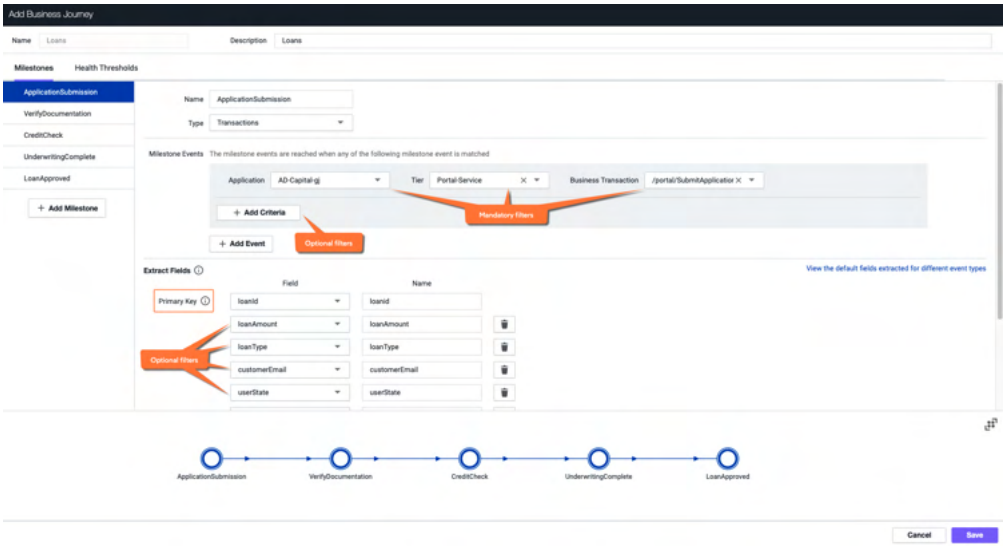
Extract Fields: Primary Key and Additional Fields

Choose a distinct primary key that uniquely identifies and ties together these independent milestones to represent loan application. In this example, `loanId` is used as the primary key. Enter `loanId` in the **Primary Key** field in each milestone. You must use the same data type as the primary key for all milestones in any given business journey.

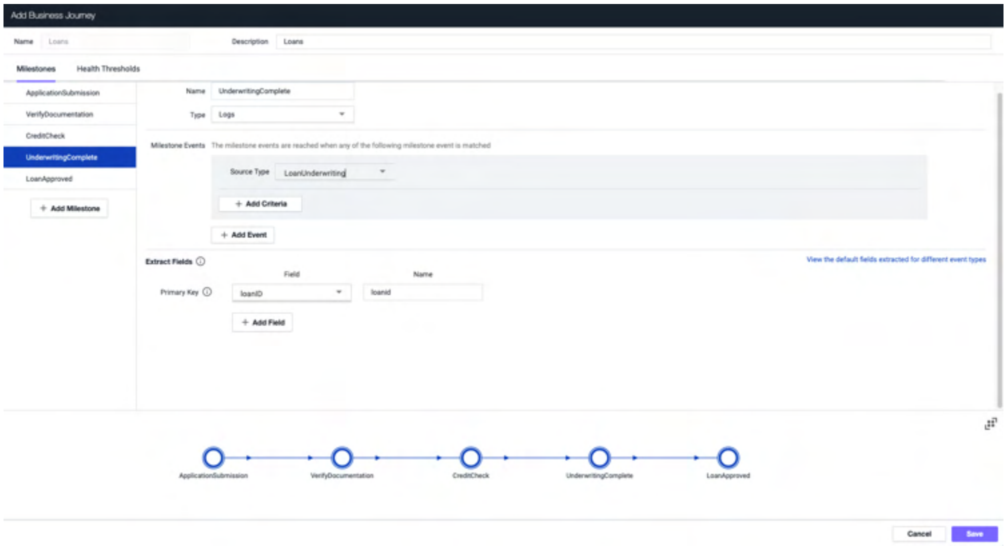
In addition to the default information collected by the Business Journey, such as event timestamp, you can extract additional business information, such as customer details, loan amount, loan type, bank name, and so on with the optional fields. These additional fields provide further context for your Business Journey definition. You can also use these fields to run ADQL queries.

Define Milestones

In the following example, you can define an application submission milestone by selecting an event type, primary key, and filters. The mandatory filters to limit the events for this milestones are **Type**, **Application**, **Tier**, and **Business Transaction**. The primary key is defined as `loanId`. Fields such as `loanA mount`, `loanType`, and `customerEmail`, are extracted in this milestone.



Similarly, the milestone for underwriting is defined with a different event type:



Since third-party service providers perform the underwriting, the status is updated in logs; **Logs** is considered the **Type** of event source. To define a milestone, click **+ Add Milestone**.

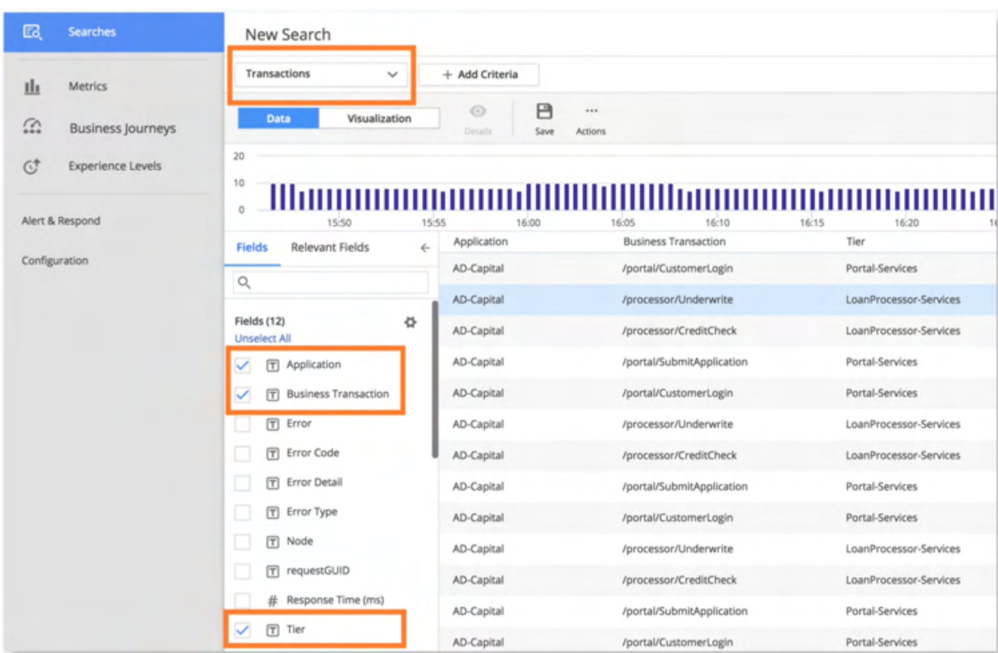
Occasionally the individual steps in a workflow, such as the application submission, can occur in different methods. You can differentiate these stages with unique filter conditions. For example, a user that submits an application through a browser performs the same action as a user that submits an application through mobile, but the filter conditions are uniquely defined. To track a milestone with several entry points, click **+ Add Event** and define your mandatory filters (application, tier, and business transaction), primary key, and optional filter conditions, including your custom fields.

You can only define the entry points for milestones if the entry points are of the same event type; if one entry point is in **Transaction** and the other is in **EUM**, you cannot track the entry points under the same milestone.

Define your remaining milestones, validate, and save the Business Journey. It takes a few minutes to display the generated records from the associated events.

Troubleshoot the Business Journey

If you think that the combination of selected fields and filters for each milestone is incorrect, you can validate with an ADQL query search or use the Analytics **Searches** widget:



You can see the valid Tiers and associated Business Transactions for AD-Capital in **Searches**, which displays all the valid values for your selection.

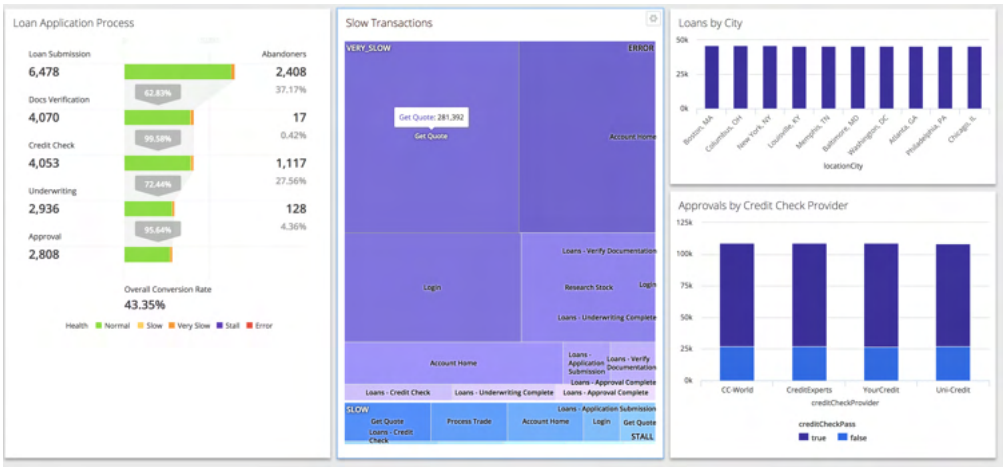
Alternatively, you can access **Tiers & Nodes** in the Controller for the AD-Capital application.

Run Analytics on the Business Journey

Use one of the search methods from **Searches** to view data, run Analytics, or visualize the performance of the Business Journey you just created.

View the Business Journey

You can view dashboards and widgets of the loan approval Business Journey in **Dashboards & Reports**. The visualization makes it easy to understand the performance of loan application workflow.



The funnel widget is located on the left side of the image. This widget represents the overall conversion rate, the health of participating events, and the number of abandoners at each critical step in the loan application process.

At the center of the dashboard, a widget illustrates slow business transactions at each step in the Business Journey.

The widget on the lower right side of the image compares credit check providers on the basis of credit approvals.

The widget on the upper right side displays the number of loans approved in different cities.

Experience Level Management

Related pages:

- View and Export XLM Compliance Data
- Configure Experience Level Management
- Migrate XLM Configurations Between Environments

The performance of business transactions and other types of events can affect user experience. Different levels of performance may correspond to service level agreements (SLAs), compliance targets, or compliance policies that your application must satisfy.

For example, an e-retailer might want to ensure that when customers add items to their carts, the transactions always complete within 100 milliseconds.

Experience Level Management (XLM) enables you to define experience levels. XLM shows you how your application is performing relative to those levels, in custom dashboards and in reports that you can view and export.

Working with XLM

Whether you speak of an experience level or a service level depends on whether your focus is user experience or SLAs, but you configure both through the same XLM UI. What this section says about experience levels applies equally to service levels.

You can define experience levels for any kind of data that meets the following criteria:

1. The data must be numerical.
2. The data must be individual (per event) values, not aggregates.

Since data collectors provide individual values, they are a good basis for experience levels as long as the values are numeric. By contrast, metrics and information points cannot be used as the basis for experience levels, because they provide aggregate values.

The following are all good bases for defining experience levels:

- Sales in dollars per week during the time an ad campaign is running
- Application response times calculated from business transaction events
- End-user response times calculated from RUM and synthetic events
- Custom analytics such as login times for platinum customers or item checkout response times for London customers

Compliance against the configured thresholds is calculated in daily intervals from the specified start date. The local time zone starting time for each day is converted to midnight of the equivalent Greenwich Mean Time (GMT) date. For example, *07/30* means *12:00 am GMT on 07/30*. The reporting job that calculates the XLM data runs every day at midnight and noon GMT. The results are updated the next time the reporting job runs.

XLM aggregates the results of reporting jobs into weekly or monthly periods, according to your choice of **Compliance Period** settings, including **Time Zone**

When viewing XLM data for a week or month, you can drill down into daily granularity. XLM also allows you to

- Create XLM reports in the time zone of your choice
- Export reports of however many reporting periods you choose, in CSV format
- Add your XLM report to a custom dashboard for periodic generation and delivery

You can export XLM configurations, and migrate them from one environment to another. You can also view and export the XLM Audit Trail, which automatically records XLM configurations and changes to them.

If you revise the configuration, the new configuration takes effect the next time the XLM reporting job runs. Past data is immutable and configuration changes do not affect the values.

To see which configuration was in effect at a particular moment in a reporting period, view the XLM Audit Trail.

Configuration Overview

In the XLM UI, you configure **Properties**, **Compliance Target Settings**, and optional **Exclusion Periods**.

- **Properties** specify what type of event to measure and what filters to apply.
- **Compliance Target Settings** specify the performance criteria you want XLM to report, through
    - the **Compliance Target**, which defines the desired level of performance for the properties
    - **Daily Target Thresholds** which define *Normal*, *Warning*, and *Critical* performance as percentages of the compliance target
    - **Treat Errors as Critical**, an option helpful for applications where apparently normal performance thresholds tend to conceal errors
- **Compliance Target Settings** also specify how you want your XLM reports structured in terms of time, through
    - the **Start Date**—the date you want the compliance calculation to begin
    - the **Compliance Period**, which
        - you can think of as the *reporting period*
        - can be weekly or monthly; choose one or the other according to your SLA requirements
    - the **Time Zone** of your choice
- **Exclusion Periods help you account for performance deviations caused by upgrades, maintenance, or usage patterns associated with weekends or holidays.**

Here is how our e-retailer example might look:

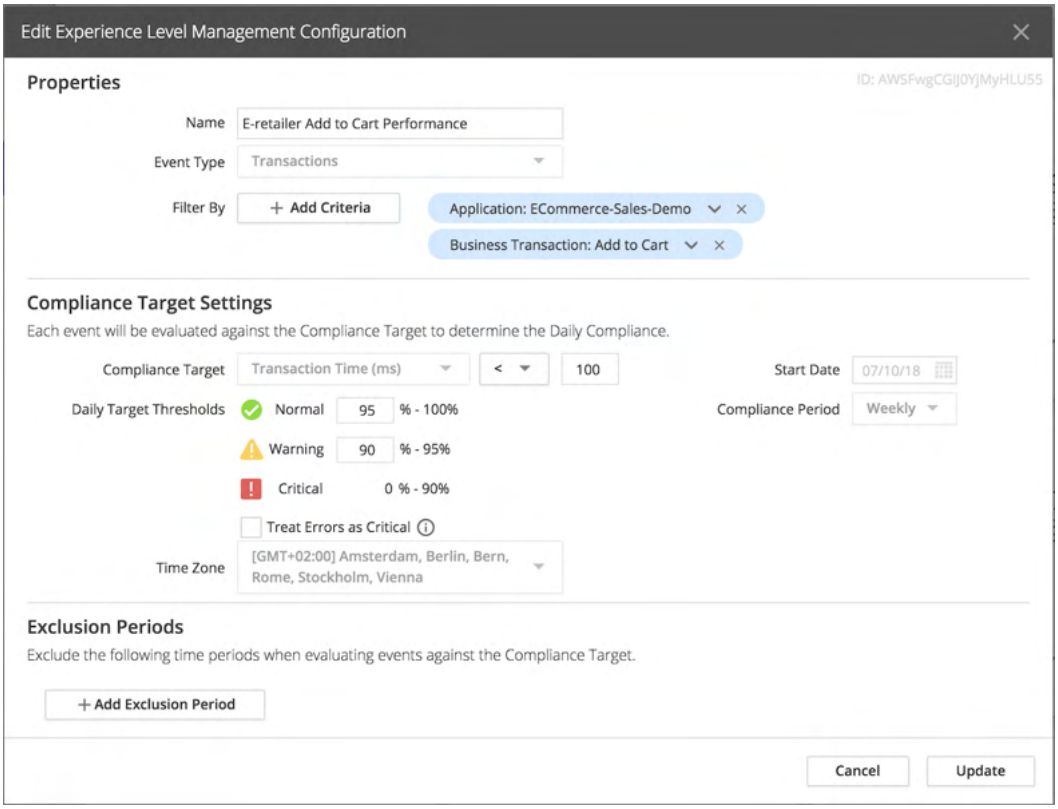
- The properties have *Transactions* as the event type, filtered to exclude all but an *Add to Cart* business transaction, for the relevant application
- The compliance target is *Response Time (ms)*, set to 100
- Values for daily target thresholds are 95% to 100% for *Normal*, 90% to 95% for *Warning*, and 0% to 90% for *Critical*
- The compliance period is *Weekly*
- The time zone is *GMT+02:00* for Amsterdam, where our hypothetical e-retailer has customers

Configured this way, XLM reports that performance is *Normal* for each week-long reporting period when the Add to Cart business transaction completes within 100 milliseconds greater than 95% of the time.

For periods when Add to Cart performance meets the target 95% of the time or less, but above 90% of the time, performance is at *Warning* level.

For periods when performance meets the target 90% of the time or less, performance is *Critical*.

This example configuration is illustrated below.



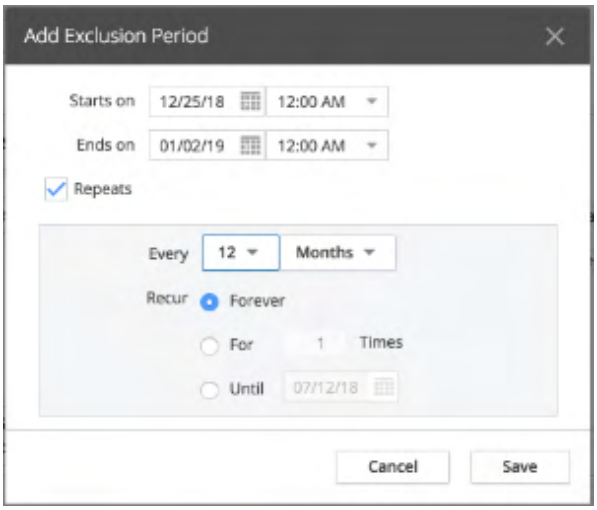
XLM Exclusion Periods

Exclusion periods are time periods which you want to exclude from an XLM report.

For example, scheduled upgrades and maintenance can disrupt performance, so you may not want those periods reflected in XLM reports. Likewise, you may want XLM reports to exclude weekends, holidays, or other times with atypical usage patterns.

XLM queries executed against your Analytics event data observe only the exclusion periods you explicitly configure in the XLM UI. You configure exclusion periods using the time zone of your choice. Under the hood, AppDynamics keeps track of the exclusion period in the corresponding GMT dates.

> XLM does not recognize exclusion and maintenance periods that you configure at the Controller level.

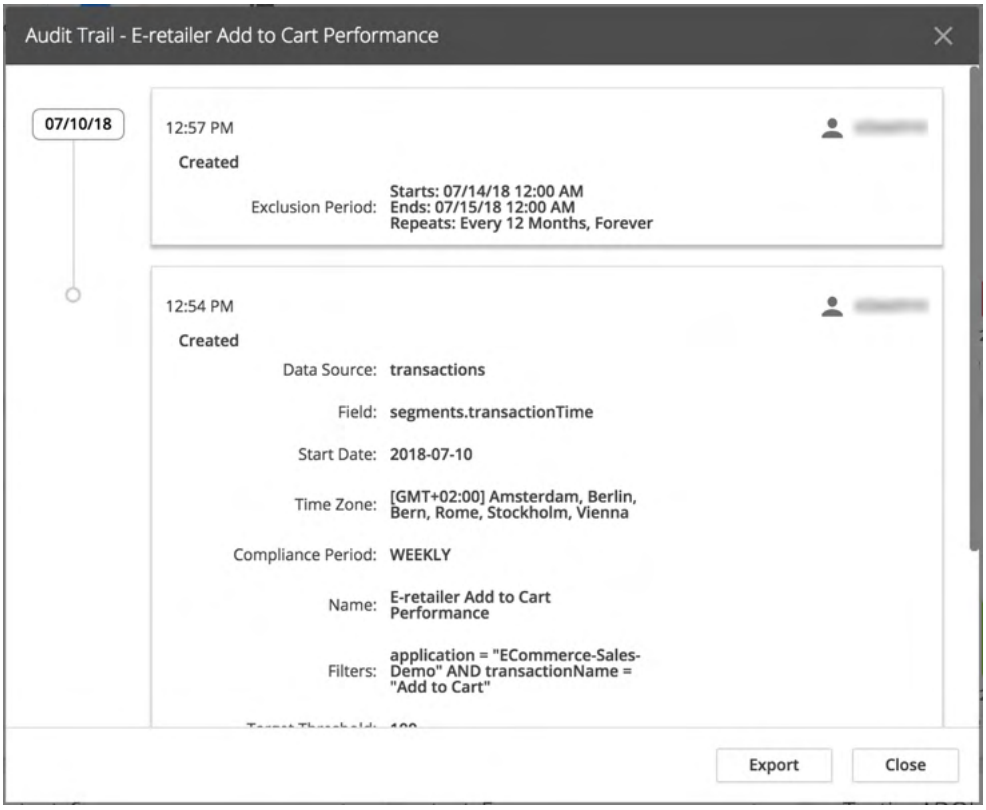


XLM Auditing

All changes to XLM configurations produce an immutable audit record called an XLM Audit Trail, which contains complete XLM configuration information. Since XLM audit reporting is separate from all other reporting in the AppDynamics Controller UI, the XLM Audit Trail is distinct from other available reports, such as the Controller Audit Report. You can export the XLM Audit Trail to a CSV file.

Each XLM configuration is limited to 1000 updates. If that limit is reached, the oldest audit records are dropped as new ones are added.

The XLM audit records contain information about what operations are performed and what fields are changed. The UI presents XLM audit records as a sequential timeline.



Configure Experience Level Management

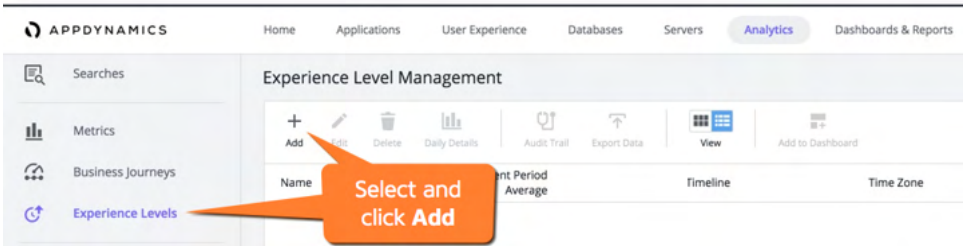
Related pages:

- View and Export XLM Compliance Data
- Migrate XLM Configurations Between Environments

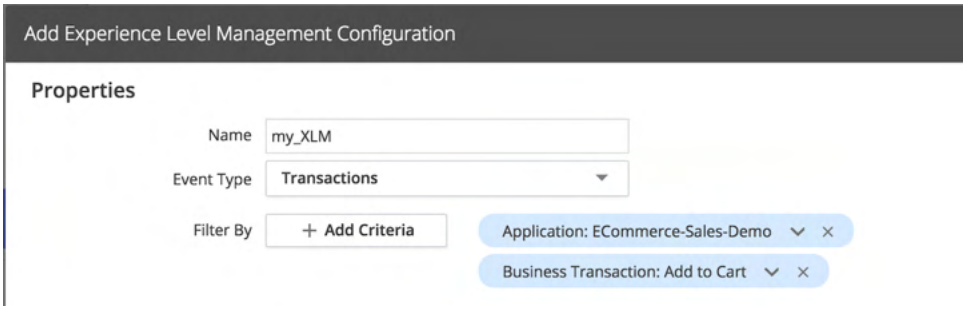
To configure Experience Level Management (XLM), you specify **Properties**, **Compliance Target Settings**, and optional **Exclusion Periods** as described earlier.

Configure an XLM report

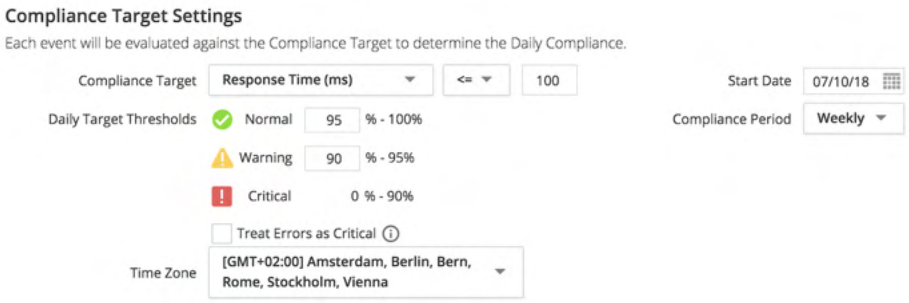
1. From the Controller UI, choose **Analytics > Experience Levels** and click **Add.**



2. In the **Properties** section, name your XLM report uniquely. Select the event type and any filters necessary for isolating the data that you want to evaluate.

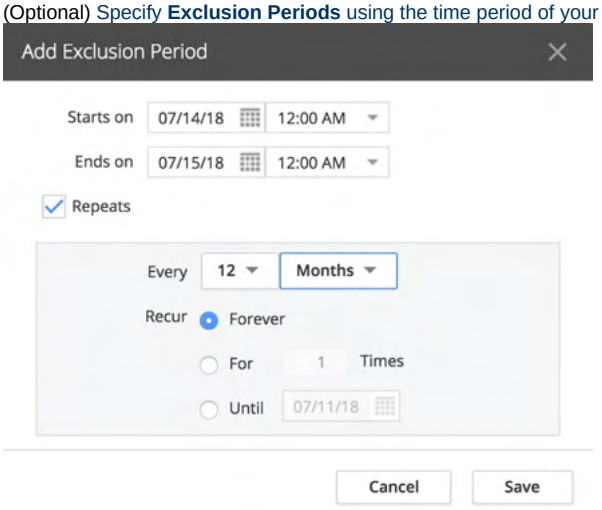


3. Select the desired field, target value, and daily target thresholds. Choose a time zone, report start date, and reporting period. Past dates are not allowed.



4. (Optional) Select **Treat Errors as Critical Failures.**

5. (Optional) Specify **Exclusion Periods** using the time period of your choice to denote periods during which data should be not collected.



6. Save the configuration. Data is collected and compliance is calculated in an automatic job that runs twice per day.

Treat Errors as Critical Failures

You can choose to treat events with errors as critical failures.

This enables you to identify instances where an event meets the compliance target but has a user experience status of Error.

For example, if you have defined Response Time of under 2000 milliseconds as the compliance target, an event with a response time of 100 milliseconds meets the compliance target, even if that event finished prematurely because of an error. In that case, XLM reporting that performance as Normal could be considered a false positive.

When you enable **Treat Errors as Critical Failures**, XLM will catch instances like this. Whether this option is optimal for your application is a subjective judgment for you to make.

Treat Errors as Critical Failures is only supported for event types that contain a user experience or similar field, as shown in the table below.

Event Type	Experience Field
Transaction Event	`userExperience`
Browser Record	`pageexperience`
Mobile Snapshot (Network Request)	`networkrequestexperience`
Browser Session	`pageexperience`
Mobile Session	`networkrequest.networkrequestexperience`

View and Export XLM Compliance Data

Related pages:

- Configure Experience Level Management
- Migrate XLM Configurations Between Environments

In the AppDynamics Controller UI, click **Analytics > Experience Level Management** to view or export XLM data.

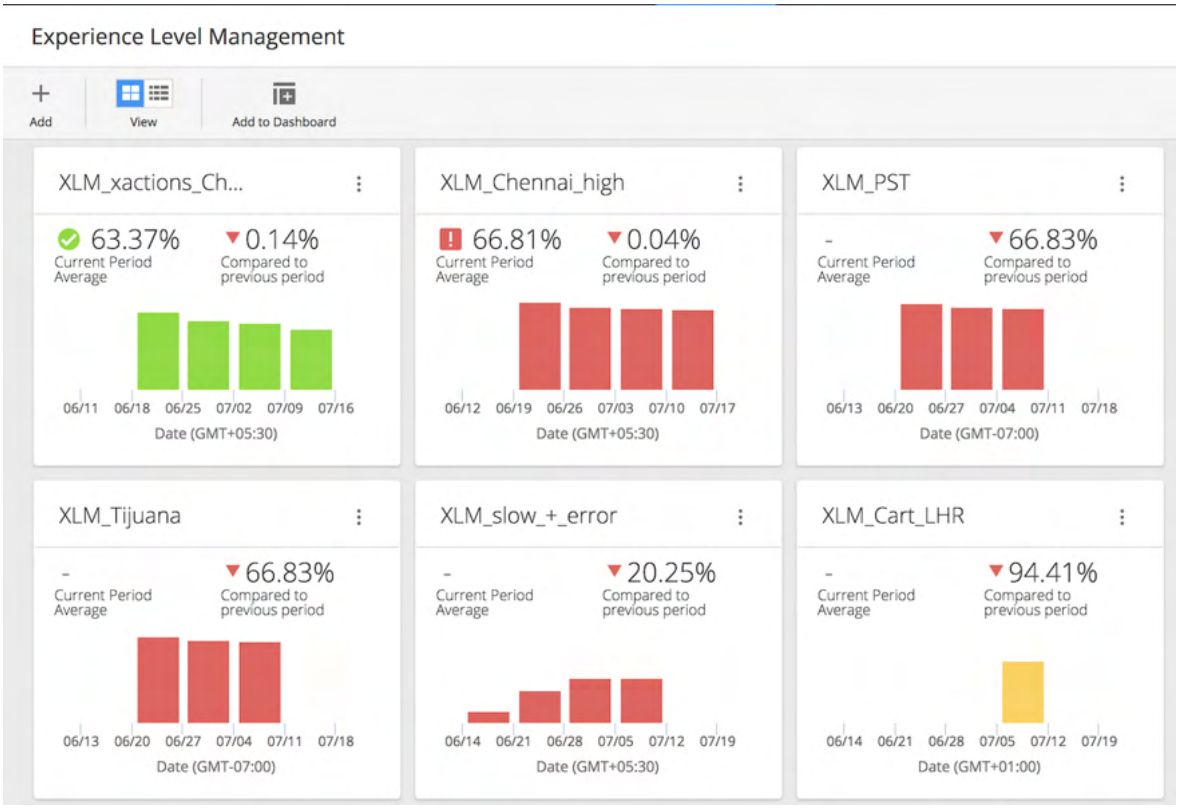
View XLM Data

You can view the last five reporting periods for your XLM data in a card or list view. To access data for earlier reporting periods, export the data as described below.

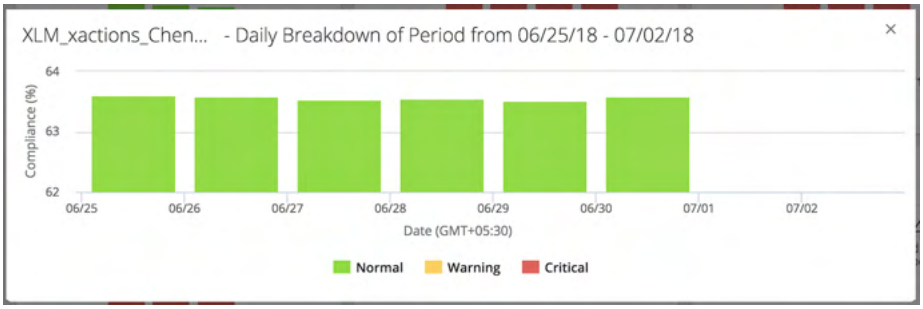
Timelines show dates in your local timezone and indicate how local time relates to Greenwich Mean Time. For example, if your local timezone is Pacific Standard Time, **Date (GMT-7:00)** appears beneath timelines.

Both views enable you to drill down from multiple reporting periods, to days, to individual events.

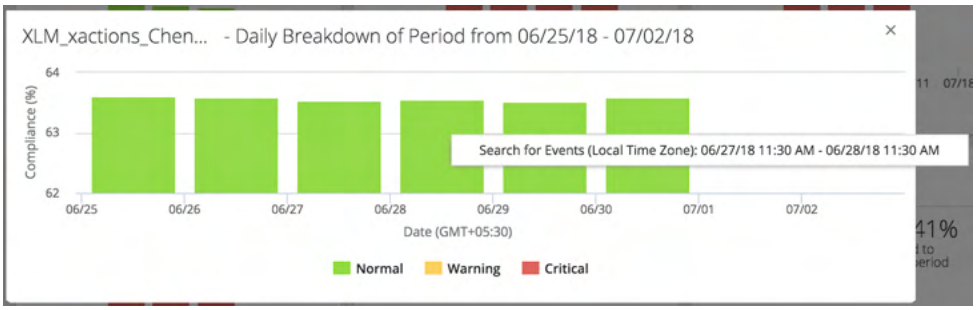
Card View



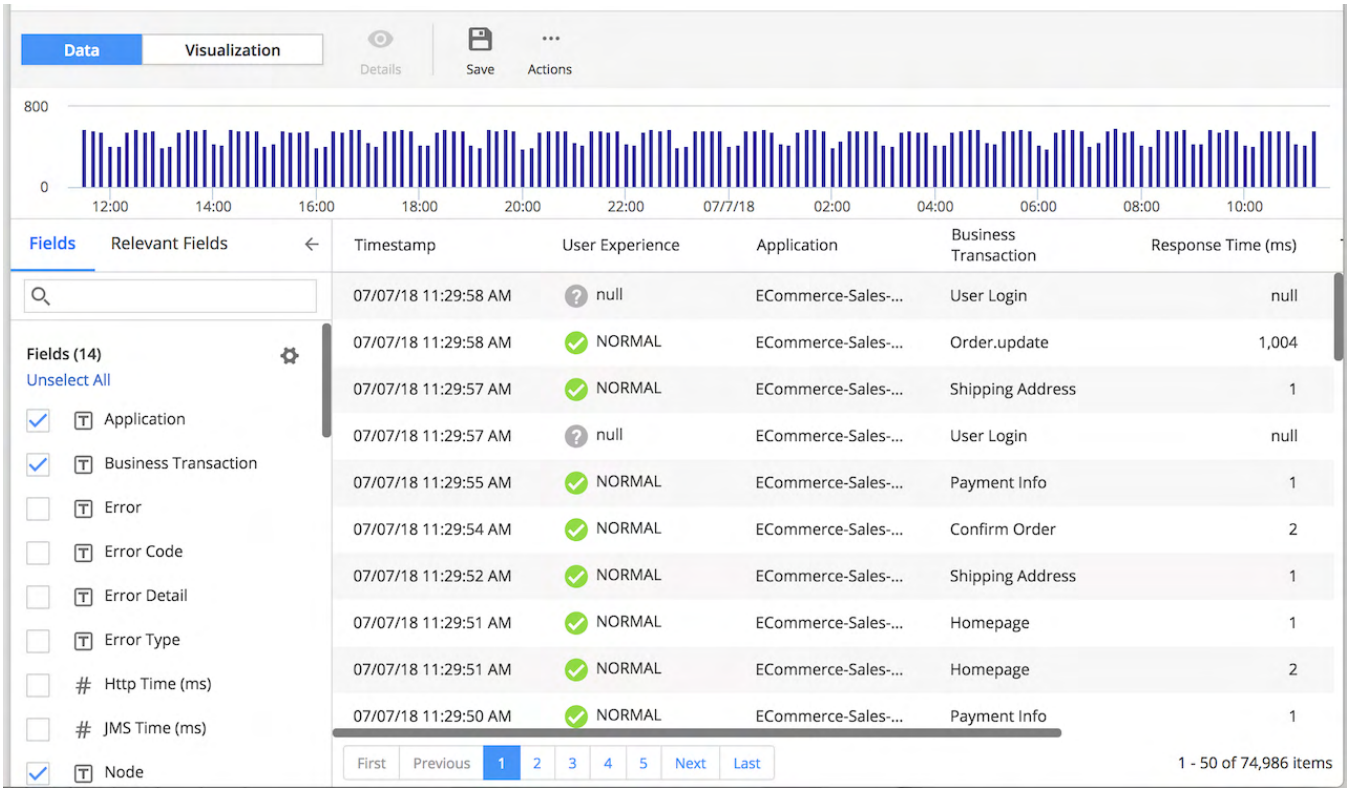
Click any reporting period for a daily breakdown.



Right-click any day to view associated events.



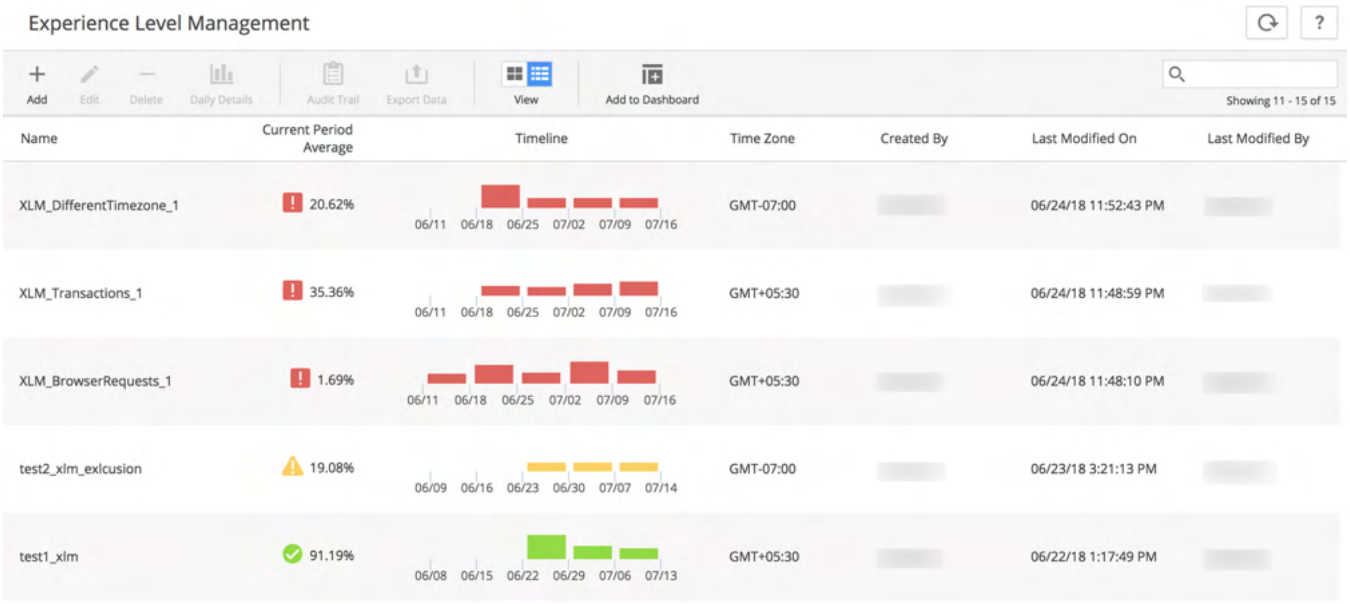
This brings you to an **Analytics > Searches** page where you can inspect the entire day's events.



List View

As in the card view, you can:

- Click any reporting period for a daily breakdown
- Right-click any day to view associated events

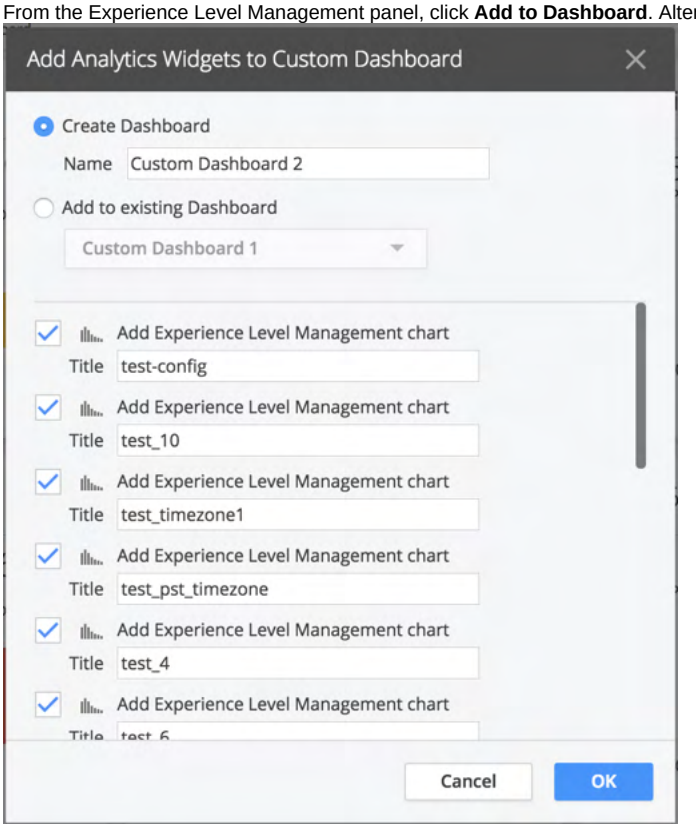


Add XLM Reports to Custom Dashboards

You can add XLM reports as read-only widgets to your custom dashboards:

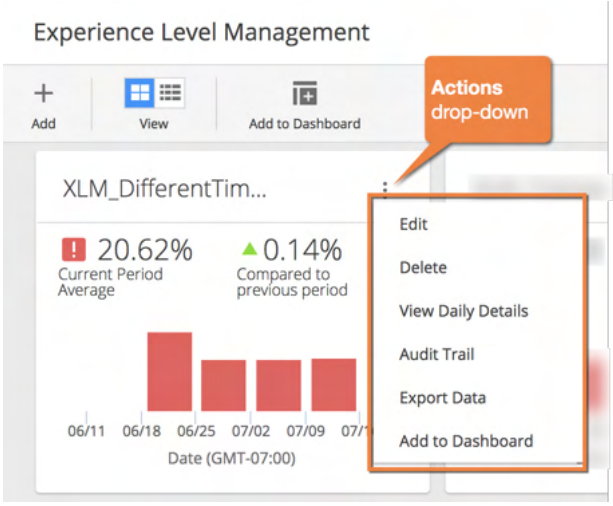
- One at a time, from the Actions drop-down, or
- One or more at once, from **Add to Dashboard** in the Experience Level Management panel

From the Experience Level Management panel, click **Add to Dashboard**. Alternatively, click the **Actions** dropdown on an individual XLM report.



Export XLM Data

You can export XLM data using the **Actions** dropdown, which also offers additional views and actions.



You can export as many reporting periods as you wish. The export format is CSV.

> The XLM export file shows the latest configuration along with compliance data.

Migrate XLM Configurations Between Environments

Related pages:

- Configure Experience Level Management
- View and Export XLM Compliance Data

You can migrate XLM configurations from one environment to another. For example, you might first deploy a configuration to a staging or test environment and later migrate it to production.

To migrate an XLM configuration, make two API calls:

1. Export XLM configuration(s) from one environment.

   `GET /controller/analytics/xlm/configuration`

   Optionally, supply a configuration ID, or a comma-delimited list of configuration IDs, as a query parameter.

   If this parameter is omitted, the API returns a bulk export of all XLM configurations.

2. Import a single XLM configuration into another environment.

   `POST /controller/analytics/xlm/configuration`

Troubleshoot Analytics Issues

Related pages:

- Agent Log Files
- Analytics Agent Logging
- .NET Agent on Windows Logging

This page provides general troubleshooting techniques for deployment issues and workarounds for certain scenarios in Application Analytics.

General Troubleshooting Issues

If you encounter problems with your Analytics deployment, check the logs for errors and warnings.

These logs are useful to:

- Determine if the Analytics Dynamic Service is enabled or disabled.
- Determine if the Analytics Dynamic Service encounters errors sending messages to the Analytics Agent. For example, when the Analytics Dynamic Service is not able to communicate with the Analytics Agent due to invalid connection configuration.
- Determine if messages are being dropped by the Analytics Dynamic Service because its internal buffers are full.
- View the configurations used on startup.

The following components write log information.

- Analytics Agent: The Analytics Agent writes log messages to files in the `<analytics_agent_home>/logs` directory.
- Events Service: The Events Service writes log messages to files in the `<events_service_home>/logs` directory. The `analytics-api-store.log` file can help with troubleshooting.
- Java Agent: The Analytics Dynamic Service is built into the Java Agent and writes logs to the same file as the app agent. The logs are in the `<application_home>/<app_agent_home>/logs` directory.
    - The primary log file to use for troubleshooting is the file named `agent.<timestamp>.log`. Search the file for messages written by the Analytics Dynamic Service.
- .NET Agent: The .NET Agent writes log messages to files in the `%ProgramData%\AppDynamics\DotNetAgent\Logs` directory. By default, the .NET Agent writes Analytics logs from the `com.appdynamics.ee.service.analytics.Analytics` logger to determine if the Analytics Dynamic Service is enabled/disabled and to view the configuration on startup. Errors are logged in the `warnfile`.
    - The `com.appdynamics.CONFIG.AnalyticsDynamicServiceConfigListener` logger gathers Analytics Dynamic Service configuration details from the Controller. Ensure that the `eventServiceURL` comes from the Controller.
    - The `com.appdynamics.analytics.EventsServiceSink` logger provides information about the communication between the .NET Agent and the Events Service.

  For additional logger information, see .NET Agent on Windows Logging.

Configuration Issues

Verify that your configuration settings are properly configured with the required account name and key. Slashes in account names and key values must be escaped.

Windows Commands

On Windows, you cannot delete a log file with the `del` command when the Analytics Agent is collecting log data from the file.

Do not use robocopy commands to move files on Windows. AppDynamics recommends that you use the move command instead.

Starting the Analytics Agent Issues

If you are running the Analytics Agent as an extension to the Machine Agent with JRE 1.8.0_171 or later, starting the Analytics Agent with encrypted credentials will fail. Workarounds include:

- Disable encryption in the Machine Agent
- Use a standalone Analytics Agent with encryption
- Use JRE 1.8.0_162 to run the Machine Agent with encryption

PID File

If an instance of the Analytics Agent terminates and leaves behind a process ID file (PID file), the next agent startup will fail with the following error:

```
java.lang.RuntimeException: Unable to create file [D:\AppDynamics\analytics-agent\analytics-agent.id] to store the process id because it already exists. Please stop any currently running process and delete the process id file
```

In agent versions prior to 4.3, you needed to delete the old process id file and restart the agent to work around this issue.

For versions 4.3 and later, you can use the `-f` flag option while starting the agent. This option causes a pre-existing process id file to be deleted. The flag is not required when you are starting the agent as a Windows service.

Use the `-f` flag as follows:

- UNIX type OS: `start -f`
- Windows CLI: `start -f`

> The `-f` flag option only works for the standalone Analytics Agent. This option does not work for the Analytics Agent embedded in the Machine Agent. In versions 20.6 and later of the Machine Agent, the bundled Analytics Agent no longer drops a PID file.
>
> For the Analytics Agent running inside the Machine Agent (embedded mode), upgrade the Machine Agent to 20.6. You no longer need to delete PID files from previous runs where the agent crashed.

High CPU Usage in Transaction Analytics without Analytics Agent (Agentless Analytics)

Reaching the Transaction Analytics license limit may cause high CPU usage in older Java Agents. AppDynamics recommends that you upgrade to the Java Agent version 20.3 and later. Look out for high CPU and memory usage in the .NET Agent.

Analytics Data Issues

Clock Management

AppDynamics recommends maintaining clock-time consistency throughout your monitored environment. If Analytics metrics are consistently reporting zero, confirm that clocks are synchronized across the application, Controller, and Events Service nodes.

Timestamps

There are potentially four time zones involved with Log Analytics:

1. The timestamp and time zone from the log file.
2. The event timestamp and pickup timestamp time zones can be different from those in the log for a number of reasons:
    a. When the time zone is overridden.
    b. The time zone is not provided correctly in the log.
    c. The timestamp and time zone parsing goes awry.
    d. When no time zone is specified in the log timestamp, then local time is assumed.
3. The Events Service time zone and the Events Service stores all timestamps in UTC time.
4. The browser used to view the Analytics data in the Controller UI (such as the **Event Timestamp** column in the UI search results or the **Time Picker** widget) converts all timestamps to the browser's local time.

Environment Variables

If you are not seeing or only seeing partial Analytics data in the Controller, configure the following environment variables.

Environment Variable	Description
`appdynamics.analytics.agent.send.attempt.max`	The maximum number of times that the Dynamic Service running in the APM Agent will try sending the Analytics data after a communication failure.
`appdynamics.analytics.agent.send.attempt.pauseMillis`	The wait time between attempts to send Analytics data to the Analytics Agent after a communication failure.
`appdynamics.analytics.numOfSinkWriterTasks`	The number of sinks used by the Dynamic Service running in APM Agent to send data to Analytics Agent. The default number is 2 and can be increased as needed.

Log Analytics Missing Field Extractions Issues

If you are missing fields in your Log Analytics data that you expect to see based on your source rule configuration or using regex (including grok patterns) in your field extraction, you may encounter a performance safeguard.

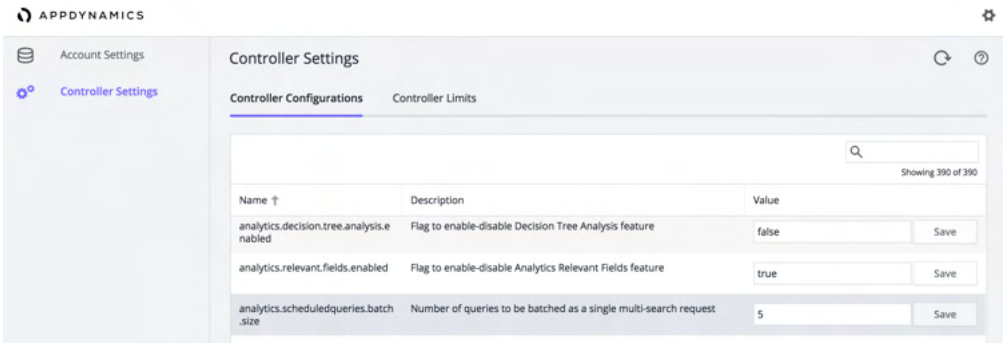
If a regex pattern takes more than five seconds to match against a log line, the attempt to extract the fields is terminated. No further processing that requires the extracted fields will occur. As a result, some fields may be missing when viewed on the Controller for that log line. In this case, the following error message appears in the Analytics Agent log:

```
[ERROR ] java.util.concurrent.TimeoutException: The current regex has spent 5 seconds attempting to match the
log line, further processing has been stopped for this log line.
```

Another reason for missing fields is if the log line doesn't contain the field to be extracted as defined in the pattern.

Custom Analytics Metrics Issues

If you are having issues with metrics created from saved searches or with the alerts performance for those metrics, try increasing the query batch size. You can increase the size using the `analytics.scheduledqueries.batch.size` Controller setting in **Controller Settings** > **Controller Configurations**. The default value for this setting is 5.



See Access the Administration Console for information about accessing the setting.

Business Transaction Events Limits

The Analytics Dynamic Service sends messages to the Events Service where the request body is an array of event segments. A business transaction event consists of one or more segments that are related to each other by the business transaction `requestGUID`. There are ingestion limits related to messages.

- Event (segment) size: The maximum size of an individual business transaction segment collected by the Analytics Dynamic Service is .1 MB. This limit is defined by the `appdynamics.analytics.message.maxSizeBytes` Java system property. To change this value, pass it on the command line as a system property when the Java Agent starts. For example:

  ```
  '-Dappdynamics.analytics.message.maxSizeBytes=1024000'
  ```

- Events per request: The maximum number of segments per request is defined by the `appdynamics.analytics.agent.send.batch.items.max` Java system property. The default value of this property is 16. To change this value, pass it on the command line as a system property when the Java Agent is started.
- Message Size: This limit refers to the size of a single request body sent to the Events Service, which is usually an array of event segments. Publish requests for all event types are limited to 1 MB. If the limit is exceeded, you will see exceptions in agent log file and messages in Events Service logs.

Agent-Side Metrics

Analytics components in the Java Agent reports performance metrics. These metrics can be used to monitor the health of Transaction Analytics reporting in the agent/node. These metrics are under the `Agent|Analytics|<Metric Name>` path in the Metric Browser.

Metric Name	Metric Time Rollup Type	Metric Cluster Rollup Type	Metric Description
Collector Objects Count	Current	Individual	The current count of Analytics data collectors in use.
Messages Sent	Sum	Collective	The number of messages sent.
Messages Dropped Since Input Queue Full	Sum	Collective	The number of dropped messages due to the input queue being full.
Messages Dropped Since Size Exceeded	Sum	Collective	The number of dropped messages due to message size exceeding limit.
Message Send Errors	Sum	Collective	The number of transient and permanent errors in event registration or publish.
Message Latency (ms)	Average	Individual	Latency calculated in milliseconds.
Pending Messages In Queue	Current	Individual	Messages pending in the buffer.
Sink Writers Count	Current	Individual	Number of active sink writer threads.

Firewall Considerations

Transaction Analytics without an Analytics Agent requires a connection between the Java/.NET Agent and the Events Service. If you have a firewall that blocks requests from the Java/.NET Agent to the SaaS or on-premises Events Service, you need to open the firewall to avoid gaps in Analytics data. See Connection Ports and Agent-to-Controller Connections.

For SaaS-based installations, configure the Analytics endpoint by modifying the `http.event.endpoint` setting in the `\conf\analytics-agent.properties` file (as described in Install Agent-Side Components). For example, `http.event.endpoint=http://analytics.api.appdynamics.com:443`.

If your firewall rules require you to use specific IP addresses, rather than hostnames, note the following information. If you are unable to see Transaction Analytics data collected as expected (even after configuring your firewall rules) and you see repetitive Connection Reset messages in the logs similar to the following, your firewall rules may not include the correct IP addresses.

```
[2015-12-18T16:08:39,907-06:00] [INFO ] [AD Thread Pool-Global0] [o.a.http.impl.execchain.RetryExec] I/O
exception (java.net.SocketException) caught when processing request to {s}->https://analytics.api.appdynamics.
com:443: Connection reset
```

Your firewall rules may not include the correct IP addresses.

In SaaS environments, `analytics.api.appdynamics.com`, `fra-ana-api.saas.appdynamics.com`, and `syd-ana-api.saas.appdynamics.com` are round-robin DNS aliases and may resolve to multiple DNS (54. vs 52.) in the following examples.

```
Name: analytics.api.appdynamics.
com
Address: 54.213.173.141
Name: analytics.api.appdynamics.
com
Address: 52.88.111.157
```

```
Name: fra-p-con-2.saas.
appdynamics.com
Address: 52.28.42.67
Name: fra-p-con-2.saas.
appdynamics.com
Address: 52.57.96.225
```

```
Name: syd-p-con-1.saas.
appdynamics.com
Address: 54.153.248.179
Name: syd-p-con-1.saas.
appdynamics.com
Address: 54.153.246.219
```

Amazon Web Services (AWS) controls the IPs used, so they may change from time to time. AWS publishes its current IP address ranges in JSON format, so if you are unable to open firewalls to hostnames, you can download the AWS IP address ranges. If you want to be notified whenever there is a change to the AWS IP address ranges, you can subscribe to receive notifications using Amazon SNS.

See Analytics IP Ranges for the AWS regions for the SaaS Analytics environments. You can view the AWS IP Ranges for each listed region.

Monitoring Analytics Agent Health

This describes monitoring the health of the Analytics Agent.

Check-Health Command Usage

Argument	Description
`-hp (--host-and-port) STRING[]`	Optional. Application host name or IP:HTTP admin port (Multiple values separated by space).
`-p (--properties) TYPE_PATH`	Optional. Path to the properties file (Multiple values separated by space).
`-v (--verbose) TYPE_BOOLEAN`	Optional. Verbose output.

The check-health command returns the status of the Analytics Agent. You can specify the agent to check using a properties file or IP address and port.

For example, using a properties file.

```
./bin/analytics-agent.sh check-health --properties conf/analytics-agent.properties
```

You can also use the -hp argument and pass the host IP address and the port number for the Analytics Agent.

For example, the default Analytics Agent port for the health check is 9091.

```
bin/analytics-agent.sh check-health -hp <ip-address>:9091
```

AppDynamics Application Performance Monitoring Platform

This page provides information on installing, configuring, and administering an on-premises AppDynamics Application Performance Monitoring (APM) Platform deployment.

Installation Overview

Before you install the platform, review the requirements for the components you plan to install and prepare the host machines. The requirements vary based on the components you deploy and the size of your deployment.

For the Controller and Events Service, you first need to install the AppDynamics Enterprise Console. You then use the application to deploy the Controller and Events Service. Note that the Events Service can be deployed as a single node or a cluster. The Enterprise Console is not only the installer for the Controller and Events Service; it can manage the entire lifecycle of new or existing AppDynamics Platforms and components.

You cannot use the Enterprise Console to perform the End User Monitoring (EUM) Server installation. Instead, you must use a package installer that supports interactive GUI or console modes, or a silent response file installation.

Follow these tasks before you start the installation process for the AppDynamics APM Platform:

- Review the Platform Requirements.
- Verify the Enterprise Console host meets the requirements to host the application and the Controller since they share the same host by default for Express Install. You do have the option to install the Enterprise Console on a different host than the Controller's using Custom Install.
- On Linux, verify that you have assigned execute permissions to the installation script with the following command:

```
chmod 775 platform-setup-64bit-linux.sh
```

You can get the software for installing the platform components from the AppDynamics download site. See Download AppDynamics Software

Platform Components and Tools

An on-premises AppDynamics Platform installation consists of several, separately installed and configured components. These include the Controller, MySQL database, Events Service, and optionally the EUM Server.

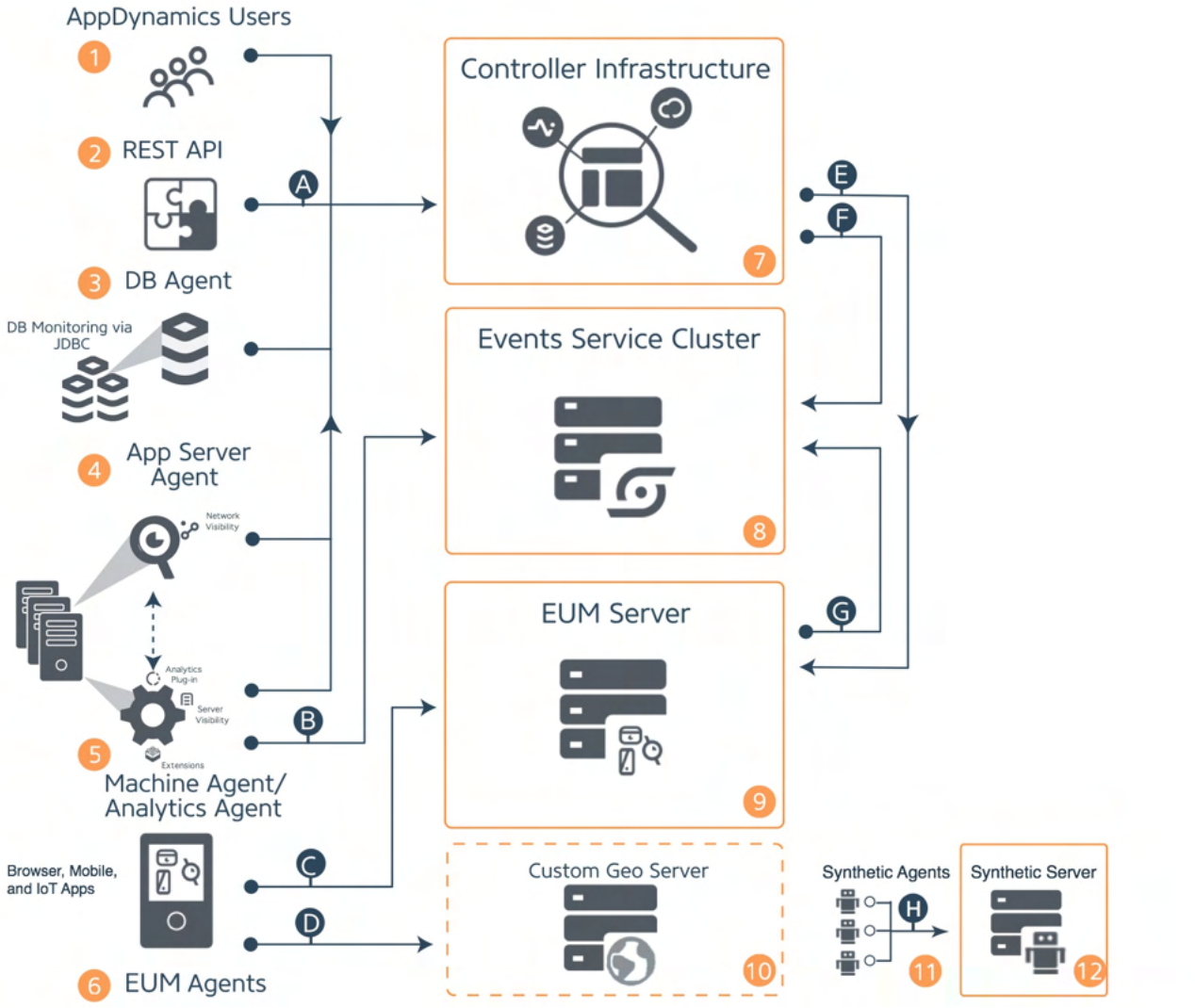
The AppDynamics Enterprise Console is a GUI and command-line based application that can manage the installation, configuration, and administration of the Controller and Events Service.

For the EUM Server, you must continue to use the package installer to deploy the EUM Cloud. See EUM Server Deployment

After you install the platform, you can configure and manage different components with component-specific scripts. Based on how you deploy the platform, you might use a combination of the Enterprise Console and package installers to install and manage the various components of the platform.

On-Premises Deployment Architecture

The following diagram depicts the components of a complete on-premises AppDynamics APM Platform deployment. It shows how the components interact to fulfill application, database, infrastructure, end-user monitoring, and more.



Depending on the scale of your deployment, your requirements, and the products you are using, your own deployment is likely to consist of a subset of the components shown in the diagram.

You can find a more detailed diagram, as well as a SaaS architecture diagram on PDFs. For a diagram of the Enterprise Console, see the Enterprise Console Platforms Architecture. For a diagram of the Synthetic Server Deployment, see the Synthetic Server Deployment Architecture.

Platform Components

The following table describes how the components work together in the AppDynamics platform.

Product Feature	Components Involved
Application Performance Management	4 App Server Agents attach to monitored applications and send data to the 7 Controller via connection A.
Server Visibility	5 Machine Agents reside on monitored servers and report data to the 7 Controller via connection A.
Application Analytics	The Analytics Dynamic Service (formerly called the Analytics plugin) on the 4 App Server agent communicates with a local 5 Analytics Agent instance. One or more Analytics Agents in a deployment send data to the 8 Events Service via connection B. The Analytics Agent is bundled with the Machine Agent but can be installed and run individually as well.

Database Visibility	The 3 Database Agent connects by JDBC to monitored databases. The agent sends data to the 7 Controller (via connection A), which uses the 8 Events Service to store certain types of data.
End-User Monitoring	For an on-premises EUM installation, you configure a connection to the web and mobile real user monitoring agents to the on-premises 9 EUM Server via connection C. The EUM Server sends data to the 8 Events Service cluster via connection G. The optional 10 Custom EUM Geo Server stores EUM Geo Resolution data taken via connection D. The optional 12 Synthetic Server receives synthetic job requests from the Controller, which are then fetched from the Synthetic Services via connection H.

Platform Connections

The following table lists and describes the traffic flow between AppDynamics platform components.

Connection	Source	Destination	Traffic	Protocol	Default Port(s)
A	1 AppDynamics users through the web GUI, 2 REST API, 3 Database Agent, 4 Application Server Agent, and 5 Machine and Analytics Agents	7 Controller	APM/Database Metrics	HTTP	8090
				HTTPS	8181
B	5 Analytics Agent	8 Events Service Cluster	Log and Transaction Analytics Event Data	HTTP	9080
C	6 Real User Monitoring (RUM) Agents	9 End-User Monitoring (EUM) Server	EUM Beacon Data	HTTP	7001
				HTTPS	7002
D	6 Real User Monitoring (RUM) Agents	10 Custom EUM Geo Server	EUM Geo Resolution Mapping Data	HTTP	80
				HTTPS	443
E	7 Controller	9 EUM Server	EUM Metric Data	HTTP	7001
				HTTPS	7002 (demo mode only)
F	7 Controller	8 Events Service Cluster	Events Service API Store	HTTP(S)	9080
			Events Service API Store Admin	HTTP(S)	9081
G	9 EUM Server	8 Events Service Cluster	Events Service API Store (EUM Event Data)	HTTP(S)	9080
			Events Service API Store Admin (EUM Event Data)	HTTP(S)	9081
H	11 Synthetic Agents	12 Synthetic Server	Synthetic Measurement Data	HTTP	10101
				HTTPS	10102

The default port 9081 is the Admin port (HTTP).

Data Storage Location

Data is stored in the following locations:

- APM configuration and metric data in the on-premises Controller MySQL database
- EUM event data in the Events Service
- Transaction and log analytics data in the Events Service
- EUM Geo Resolution data in the on-premises GeoServer
- EUM Synthetic data in the on-premises Synthetic Server

Installation and Upgrade Overview

The installation and upgrade process for the AppDynamics platform consists of pre-installation steps to prepare your network and host machines for installation, installation tasks, and post-install steps to complete the required configuration. See Planning Your Deployment

After this process, you can perform optional configurations and administrative tasks described in Secure the Platform.

To start the installation or upgrade process, see Platform Requirements for information about requirements and pre-installation tasks.

Planning Your Deployment

This section provides an overview of how to plan your on-premises deployment.

Related pages:

- Platform Requirements
- Controller System Requirements
- EUM Server Requirements
- Events Service Requirements

Before You Begin

Before you upgrade or install the platform, perform the following tasks:

- Choose your appropriate Enterprise Console install or upgrade path.
- Review the requirements for the components you plan to install and prepare the host machines. The requirements vary based on the components you deploy and the size of your deployment.
- Download the Enterprise Console and start your platform installation.
- Verify that a user account with write permissions for the installation directory you want to use exists. Install all components with the same user or a user with equivalent permissions.

You can refer to the child pages for the platform requirements and deployment guides.

Irrespective of the server's computing power, installing more than one controller on the same server is not supported.

Choose How You Want to Install

The deployment option you choose to follow should be based on the components of the AppDynamics platform that you want to deploy. Deployment options that include the Controller and Events Service installations require the Enterprise Console, while the EUM Server installation requires the use of package installers.

Based on your AppDynamics deployment, you may use the Enterprise Console for the following tasks:

- Install a Controller and an embedded Events Service.
- Install an HA pair and scaled-out Events Service.
- Install or upgrade the Events Service on Linux.
- Install or upgrade an Events Service on Windows that runs on a single host.
- Discover and manage a Controller and Events Service.
- Upgrade a Controller or HA pair and preserve customizations after the upgrade.
- Upgrade MySQL to the latest supported version.

The following tasks cannot be performed using the Enterprise Console, and therefore, must be performed manually:

- Install or upgrade an HA pair on Windows.
- Install or upgrade an Events Service on Windows that runs on multiple hosts as a cluster.
- Install the EUM Server.
- Install the Synthetic Server.

Installation and Upgrade Quick Starts

Based on how you deploy and what components of the AppDynamics you deploy, there are different installation and upgrade steps to follow.

New Installations Quick Start

This installation path describes the use of the Enterprise Console to install the Controller and Events Service. The EUM Server must be installed separately as it cannot be installed using the Enterprise Console.

1. Install the Enterprise Console.
2. Use the Enterprise Console to create the platform and add hosts.
3. Install the Controller and Events Service on the same host by using Express Install. Use Custom Install to install a scaled-out Events Service that runs on a host that is separate from the Controller. Custom installations give more flexibility on where and how to install Controller and Events Service.
4. Install the EUM Server.
5. Complete the post-install tasks for the Controller, Events Service, and EUM Server.

For a more detailed description of this path, see Platform Installation Quick Start.

Upgrade Quick Start

This is the upgrade path to follow using the Enterprise Console and EUM package installer:

1. Install the Enterprise Console.
2. Discover and upgrade an existing Events Service with the Enterprise Console.
3. Upgrade the Production EUM Server with the package installer.

For a more detailed description of this path, see Discovery and Upgrade Quick Start.

Platform Component Compatibility

For versions 4.5.x+, to ensure that the platform components work properly, be sure you install compatible component versions based on the following guideline:

Platform Version	Platform Component Version Compatibility	Example of Compatible Versions
4.5.0–4.5.1	Events Service version >= EUM Server version >= Controller version	• Events Service 4.5.1 • EUM Server 4.5.0 • Controller 4.5.0
4.5.2+	EUM Server version >= Controller version (Events Service 4.5.2+ is backward compatible with the other platform components.)	• EUM Server 4.5.3 • Controller 4.5.2

The Synthetic Server 4.5.x+ versions are compatible with any of the other component versions 4.5.0+ and the Synthetic Private Agent 4.5.4+.

To get the versions of the Events Service, Controller, and Events Service, see Getting Platform Versions. For the EUM Server and Synthetic Server, use the following endpoints:

- `http(s)://<on-prem-eum-server_domain-name>:7001/eumcollector/get-version`
- `http(s)://<on-prem-synthetic-server_domain-name>:10101/version`

Download the Software

To use the Enterprise Console to install the AppDynamics Platform, you need to download the Enterprise Console installer and, if needed, the EUM Server package installer.

For more information on downloading the software, see Download AppDynamics Software.

Platform Requirements

The requirements for different components in the AppDynamics platform are based on the performance profile you select. This page describes the different performance profiles and how to determine the profile size you need.

CPU and Memory Space Requirements

When the Enterprise Console host is shared with the Controller host, it should have enough space to match the Controller host requirements, since there is no need for additional memory for the Enterprise Console.

However, when the Enterprise Console host is not shared with the Controller host, then it requires additional memory and disk space.

See Enterprise Console Requirements and Prepare the Controller Host for additional space requirements.

Network Considerations

Your network or the host machine may have built-in firewall rules that you will need to adjust to accommodate the AppDynamics on-premises platform. Specifically, you may need to permit network traffic on the ports used in the system. For more information, see Port Settings.

For expected bandwidth consumption for the agents, see the requirements documentation for app agents, see Install App Server Agents.

System User Account

You need to install all platform components with a single user account or accounts that have equivalent permissions on the operating system. The user needs to have write permissions for the installation directory.

Operating System Support

These operating systems support the AppDynamics platform:

Linux (64 bit)	Microsoft Windows (64 bit)
• RHEL 7.x and 8.x • CentOS 7 and 8.x • Ubuntu 14, 16, 18.x, and 20.x • openSUSE Leap 12 and 15 • Amazon Linux 1 and 2	• Windows Server 2012 and 2012 R2 • Windows Server 2016 • Windows Server 2019

You can use the following file systems for machines that run Linux:

- ZFS
- EXT4
- XFS

On-premises controller deployments on Linux are only supported on x86-64.

Internationalization Support

The Controller and App Agents provide full internationalization support, with support for double- and triple-byte characters. This support provides the following abilities:

- Controller UI users can enter double- or triple-byte characters into text fields in the UI.
- The Controller can accept data that contains double- or triple-byte characters from instrumented applications.

Network Bandwidth Requirements

See Administer App Server Agents for information on bandwidth usage in an AppDynamics deployment.

More Information

For requirements that are specific to product components, see the following pages:

- Controller System Requirements
- EUM Server Requirements

Port Settings

When deploying AppDynamics, you may need to open ports in a network firewall or configure a load balancer to enable communication between the Controller and the rest of the AppDynamics platform.

For SaaS, you only need to adjust your infrastructure to accommodate the HTTPS port provided to you by AppDynamics. For an on-premises deployment, however, you may need to make additional adjustments based on the information here.

Platform Component Ports

The following ports are open in a platform deployment. The "external" column indicates whether connections to the port occur entirely within the Controller host or from outside the host, and therefore may require firewall or load balancer configuration changes.

Port Name	Default	External?
Enterprise Console port	9191	Yes. The application uses port 9191 for all traffic.
SSH port	22	The port needs to be open between the Enterprise Console and the remote hosts it manages. This is for Unix only and is not configurable. If you have a requirement to configure the port, contact AppDynamics support.
Database server port	3388	No
Default database port	3377	No
Application server admin port	4848	No
Application server JMS port	7676	No
Application server IIOP port	3700	No
Application server primary port (HTTP)	8090	Yes
Application server SSL port (HTTPS)	8181	Yes
Events Service REST API port	9080	If the Events Service and Controller are on different hosts, you need to configure the port in the firewall or load balancer.
Events Service REST API admin port	9081	If the Events Service and Controller are on different hosts, you need to configure the port in the firewall or load balancer.
Reporting service HTTP port	8020	No
Reporting service HTTPs port	8021	No
EUM server port (HTTP)	7001	If EUM and the Controller are on different hosts, you need to configure the port in the firewall or load balancer.
EUM server SSL port (HTTPS)	7002	If EUM and the Controller are on different hosts, you need to configure the port in the firewall or load balancer.

At installation time, you can enter different ports manually. After installation, you can change the port settings by either reinstalling the Controller or by editing the port configuration as defined on the **Enterprise Console Configurations** page or in the underlying GlassFish application server, as described in the following sections.

Editing Controller Port Configurations

You can modify connection settings through the Enterprise Console UI. The Enterprise Console will automatically update all occurrences in the controller for you. You do not need to manually update all the files and manage the sequence to restart services. See Update Platform Configurations.

You can also edit the ports manually by editing configuration files used by the application server for the Controller domain. Updating the ports manually, however, will cause the Enterprise Console to have no visibility into the updates and cause health-check errors.

The following sections list the settings you need to modify to change a port.

Change the Primary Server Listening Port

1. In `domain.xml`, change the port number as it appears in these locations:
    - The value of the `network-listener` element with the attribute id="`http-listener-1`" for the primary listening port, or `http-listener-2` for the secure listening port to the new port setting.
    - The JVM argument values for the Controller HTTP port and Controller services port under the config element named `server-config`.
2. For each deployed agent, navigate to proxy/conf in the agent home directory and change the `controller-port` value in `controller-info.xml`.

Change the Database Port

1. In `domain.xml`, change the database listening port where it appears under the `jdbc-connection-pool` element named `controller_mysql_pool`. It appears as the value of the property named `portNumbe`.
2. Edit the file `appserver/glassfish/domains/domain1/imq/instances/imqbroker/props/config.properties` to change the "`imq.persist.jdbc.mysql.property.url`" variable so that it includes the new port number. This variable is the JDBC connection string.
3. In `db/db.cnf`, set the "`port=`" variable to your new port setting.
4. In `bin/controller.bat (.sh)`, change the "`DB_PORT`" variable to your new port setting.

Change the Glassfish Admin Listening Port

1. In `domain.xml`, change the port attribute value of the `http-listener` element to the new port. This is the element with an id attribute value of "`admin-listener`".
2. Also in the Controller home directory, change the `adminPort` value in `.install4j/response.varfile`. This ensures that the new port number is not overwritten in a future Controller upgrade.

Change the JMS Port

1. In `domain.xml`, change the port attribute value for the `jms-host` element with the name attribute of `default_JMS_host`.
2. Change the `jmsPort` value in `.install4j/response.varfile`. This ensures that the new port number is not overwritten in a future Controller upgrade.

Change the IIOP Listening Port

1. In `domain.xml`, edit the port attribute value of the `iiop-listener` element with an id attribute of `orb-listener-1`.
2. Change the `iiopPort` value in `<controller_home>/.install4j/response.varfile`. This ensures that the new port number is not overwritten in a future Controller upgrade.

Enable Appserver Health Checks for HTTPS

If you disable or lock the Controller's HTTP port, you will need to configure the Appserver health check to contact the Controller's HTTPS port instead. You can do so by completing the following steps:

1. In `domain.xml`, set the HTTP listener, `http-listener-1`, to `enabled=false`.
2. Restart the Controller.
3. Use the Enterprise Console to discover and upgrade the Controller.
   The Enterprise Console will default to the HTTPS port.

Physical Machine Controller Deployment Guide

The following pages describe considerations and instructions for deploying the Controller on physical machines.

Prepare the Controller Host

Related Pages:

- Controller System Requirements

This page describes the common configuration, tuning, and environment requirements for the machine that hosts the Controller.

These considerations apply whether the machine runs Linux or Windows or is a virtual machine. For specific considerations for your operating system type, see the related pages links.

Time Synchronization Service

A time synchronization service, such as the Network Time Protocol daemon (ntpd), should be enabled on the Controller host machine.

MySQL Conflict

Certain Linux installation types include MySQL as a bundled package. No MySQL instances other than the one included in the Controller host should run on the Controller host. Verify that no such MySQL processes are running.

Virtual Memory Space

The virtual memory size (swap space on Linux or Pagefile space on Windows) should be at least 10 GB on the target system, and ideally 20 GB.

Verify the size of virtual memory on your system and modify it if it is less than 10 GB. Refer to the documentation for your operating system for instructions on modifying the swap space or Pagefile size.

Disk Space

In addition to the minimum disk space required to install the Controller for your profile size, the Enterprise Console writes temporary files to the system temporary directory, typically /tmp on Linux or c:\tmp on Windows. The Enterprise Console requires 1024 MB of free temp space on the controller host.

On Windows, in case of an error due to not meeting the above requirement, you can set the temporary directory environment variable to a directory with sufficient space for the duration of the installation process. You can restore the setting to the original temp directory when the installation is complete.

Network Ports

Review the ports that the Controller uses to communicate with agents and the rest of the AppDynamics platform. For more information, see Port Settings.

Note that on Linux systems, port numbers below 1024 may be considered privileged ports that require root access to open. The default Controller listen ports are not configured for numbers under 1024, but if you intend to set them to a number below 1024 (such as 80 for the primary HTTP port), you need to run the Enterprise Console as the root user.

Prepare Linux for the Controller

This page describes the configuration requirements and considerations for using a Linux system as a Controller host machine.

User Account Requirements

The user account you use to perform the installation must have the following permissions:

- read, write and execute permissions on the directory where you install the Controller
- write permission on the `/etc/.java/.systemprefs` directory

If you are installing other AppDynamics Platform server components, such as the EUM Server or Application Analytics Processor, on the same machine, it is recommended that you perform the installation as the same user or a user with the same permissions on the target machine.

Virus Scanners

Configure virus scanners on the target machine to ignore the AppDynamics Enterprise Console directory and database directory (or simply the entire Controller directory). Code is never executed from the data directory, so it is generally safe to exclude this directory from virus scanning. The default location of the data directory is `<controller_home>/db/data`.

Also configure virus scanners to trust the Controller launcher, database executable, reporting service launcher, and events service (analytics processor) launchers. The launcher names are:

- Controller launcher: `AppDynamicsDomain1Service.sh`
- MySQL executable: `mysqld.sh`
- Events service launcher: `analytics-processor.sh`
- Reporting service launcher: `appdynamicsreportingservice.sh`

Anti-virus Exclusions

If you are running an antivirus program on your Linux system, it must meet one of the following conditions:

- The anti-virus program is read-only; it only detects and reports issues but never modifies files
- The anti-virus program excludes the MySQL data directory (`datadir`), which is often set to the path `db/data`.

If the program does not meet either of those conditions, it can randomly corrupt the MySQL database and hence the controller.

netstat Network Utility

Verify that your distribution of Linux includes the `netstat` network utility. If it does not, install the utility. The Controller installation uses `netstat` to determine whether MySQL processes are running.

For example, you can install the package that includes `netstat` with the following command on CentOS:

```
yum install net-tools
```

libaio Requirement

The Controller requires the `libaio` library to be on the system. This library facilitates asynchronous I/O operations on the system. Note if you have a NUMA based architecture, then you are required to install the `numactl` package.

Install `libaio` on the host machine if it does not already have it installed. The following table provides instructions on how to install `libaio` for some common flavors of Linux operating system.

Linux Flavor	Command

<table>
<tr>
<td>

- Red Hat
- CentOS
- Amazon

</td>
<td>

Use yum to install the library, such as:

- yum install libaio
- yum install numactl
- yum install tzdata
- yum install ncurses-libs-5.x

For RHEL8, CentOS8, and Amazon2, you install the ncurses-libs-5.x library using a rpm file downloaded from a trusted source:

```
sudo rpm -ivh --force ncurses-base-5.x.rpm
sudo rpm -ivh --force ncurses-libs-5.x.rpm
```

Note: The ncurses-libs depends on the ncurses-base so you should install the ncurses-base first.

Example of a trusted source for rpm download:

http://mirror.centos.org/centos/7/os/x86_64/Packages/ncurses-base-5.9-14.20130511.el7_4.noarch.rpm

http://mirror.centos.org/centos/7/os/x86_64/Packages/ncurses-libs-5.9-14.20130511.el7_4.x86_64.rpm

- To install version 6, follow these steps:

You must either create symlinks for ncurses-libs-5 which points to ncurses-libs-6, or install the ncurses-compat-libs pac kage, to provide ABI version 5 compatibility.

RHEL8 symlink:

```
sudo ln /usr/lib64/libtinfo.so.6.1 /usr/lib64/libtinfo.so.5
sudo ln /usr/lib64/libncurses.so.6.1 /usr/lib64/libncurses.so.5
```

CentOS8 symlink:

```
sudo ln /usr/lib64/libtinfo.so.6.1 /usr/lib64/libtinfo.so.5
sudo ln /usr/lib64/libncurses.so.6.1 /usr/lib64/libncurses.so.5
```

Amazon2 symlink:

```
sudo ln -s /usr/lib64/libncurses.so.6.0 /usr/lib64/libncurses.so.5
sudo ln -s /usr/lib64/libtinfo.so.6.0 /usr/lib64/libtinfo.so.5
```

RHEL8 compat-libs:

```
sudo yum install -y ncurses-compat-libs
```

CentOS8 compat-libs:

```
sudo yum install -y ncurses-compat-libs
```

Amazon2 compat-libs:

```
sudo yum install -y ncurses-compat-libs
```

</td>
</tr>
<tr>
<td>Fedora</td>
<td>

Install the library RPM from the Fedora website:

- yum install libaio
- yum install numactl
- yum install tzdata

</td>
</tr>
</table>

<table>
<tr><td>Ubuntu</td><td>Use <code>apt-get</code>, such as:
<code>sudo apt-get install libaio1</code>
<code>sudo apt-get install numactl</code>
<code>sudo apt-get install tzdata</code>
<code>sudo apt-get install libncurses5</code> For Ubuntu20 you can install <code>libncurses5</code> or <code>libncurses6</code>. If you choose <code>libncurses5</code>:<pre>sudo apt-get install libncurses5</pre>
If you choose <code>libncurses6</code>:<pre>sudo apt-get install libncurses6</pre>Note: For <code>libncurses6</code> you need to create symlink for <code>libncurses5</code> pointing to <code>libncurses6</code>.<pre>sudo ln -s /usr/lib/x86_64-linux-gnu/libncurses.so.6.2 /usr/lib/x86_64-linux-gnu/libncurses.so.5
sudo ln -s /usr/lib/x86_64-linux-gnu/libtinfo.so.6.2 /usr/lib/x86_64-linux-gnu/libtinfo.so.5</pre></td></tr>
<tr><td>Debian</td><td>Use a package manager such as APT to install the library (as described for the Ubuntu instructions above).</td></tr>
<tr><td>SLES12 and SLES15</td><td>Use <code>zypper</code> to install the library, such as:<pre>sudo zypper install libxml2-2

sudo zypper install libxml2-tools

sudo zypper install libaio1

sudo zypper install numactl

sudo zypper install libcurses5

sudo zypper install tzdata</pre></td></tr>
</table>

tzdata Requirement

Ubuntu version 16 and higher requires the `tzdata` package in order to install the Enterprise Console and Controller.

The `tzdata` package is also required by the MySQL connector.

To install `tzdata`, use `apt-get`, such as:

- `sudo apt-get install tzdata`

Configure User Limits in Linux

AppDynamics requires the following hard and soft per-user limits in Linux:

- Open file descriptor limit (`nofile`): 65535
- Process limit (`nproc`): 8192

The following log warnings may indicate insufficient limits:

- Warning in database log: "Could not increase number of max_open_files to more than xxxx".
- Warning in server log: "Cannot allocate more connections".

To check your existing settings, as the root user, enter the following commands:

```
ulimit -S -n
ulimit -S -u
```

The output indicates the soft limits for the open file descriptor and soft limits for processes, respectively. If the values are lower than recommended, you need to modify them.

Where you configure the settings depends upon your Linux distribution:

- If your system has a `/etc/security/limits.d` directory, add the settings as the content of a new, appropriately named file under the directory.
- If it does not have a `/etc/security/limits.d` directory, add the settings to `/etc/security/limits.conf`.
- If your system does not have a `/etc/security/limits.conf` file, it is possible to put the `ulimit` command in `/etc/profile`. However, check the documentation for your Linux distribution for the recommendations specific for your system.

To configure the limits:

1. Determine whether you have a `/etc/security/limits.d` directory on your system, and take one of the following steps depending on the result:
    - If you *do not* have a `/etc/security/limits.d` directory:
        a. As the root user, open the `limits.conf` file for editing:

        ```
        /etc/security/limits.conf
        ```

        b. Set the open file descriptor limit by adding the following lines, replacing `<login_user>` with the operating system username under which the Controller runs:

        ```
        <login_user> hard nofile 65535
        <login_user> soft nofile 65535
        <login_user> hard nproc 8192
        <login_user> soft nproc 8192
        ```

    - If you *do* have a `/etc/security/limits.d` directory:
        a. As the root user, create a new file in the `limits.d` directory. Give the file a descriptive name, such as the following:

        ```
        /etc/security/limits.d/appdynamics.conf
        ```

        b. In the file, add the configuration setting for the limits, replacing `<login_user>` with the operating system username under which the Controller runs:

        ```
        <login_user> hard nofile 65535
        <login_user> soft nofile 65535
        <login_user> hard nproc 8192
        <login_user> soft nproc 8192
        ```

2. Enable the file descriptor and process limits as follows:

    > This step is not required for RHEL/CentOS version 5 and later. The below file has been combined into `/etc/pam.d/system-auth`, and already contains the required line.

    a. Open the following file for editing:

    ```
    /etc/pam.d/common-session
    ```

    b. Add the line:

    ```
    session required pam_limits.so
    ```

3. Save your changes to the file.

When you log in again as the user identified by `login_user`, the limits will take effect.

Fonts Needed for the Reporting Service

The Reporting Service depends upon certain system libraries and resources that are usually included in standard Linux distributions. However, certain lightweight flavors of Linux may be lacking the requirements, primarily font libraries. The Reporting Service requires Fontconfig and FreeType installed as well as at least one sans-serif font. Errors in the reporting server log will indicate missing components, such as a missing `libfontconfig.so` file.

The following table lists one operating systems and the commands to install the required libraries:

Operating System	Command

CentOS 6.1, 6.2; CentOS 6.3, 6.4, and 6.5, Fedora 14	`$ yum install fontconfig freetype urw-base35-fonts`
	
	`$ yum groupinstall hebrew-support`
	
	`$ yum langinstall he_IL`

CentOS 7.x, Redhat 7.x	`$ yum install fontconfig`
	
	`$ yum groupinstall Fonts # Only needed for Chinese/Japanese`

Ubuntu 8, 12, 14	`$ sudo apt-get update`
	
	`$ sudo apt-get install libfreetype6 libfreetype6-dev libfontconfig`
	
	`$ sudo apt-get install language-support-he language-pack-he`
	
	`$ sudo apt-get install culmus culmus-fancy xfonts-efont-unicode xfonts-efont-unicode-ib xfonts-intl-european msttcorefonts`

Ubuntu 13	`$ sudo apt-get install libfontconfig`
	
	`$ sudo apt-get install language-support-he language-pack-he`
	
	`$ sudo apt-get install culmus culmus-fancy xfonts-efont-unicode xfonts-efont-unicode-ib xfonts-intl-european msttcorefonts`

See Administer the Reporting Service for information on configuring the service.

GNU C Libraries

The Reporting Service requires GLIBCXX_3.4.9 or later and GLIBC_2.7 or later to run.

For more information and download instructions, see https://www.gnu.org/software/libc/.

Prepare Windows for the Controller

This page provides operational and setup guidelines for running the Controller on Windows.

User Account Requirements

The user account you use to install the Controller must have administrative privileges on the host Windows machine.

> If the host Windows machine is the Enterprise Console host, the Windows user account is configured to run with administrative privileges by default.

Virus Scanners

Configure virus scanners on the target machine to ignore the AppDynamics Enterprise Console directory and database directory (or simply the entire Controller directory). Code is never executed from the data directory, so it is generally safe to exclude this directory from virus scanning. The default location of the data directory is `<controller_home>\db\data`.

Also configure virus scanners to trust the Controller launcher, database executable, reporting service launcher, and events service (analytics processor) launchers. The launcher names are:

- Controller launcher: `AppDynamicsDomain1Service.exe`
- MySQL executable: `mysqld.exe`
- Events service launcher: `analytics-processor.exe`
- Reporting service launcher: `appdynamicsreportingservice.exe`

Windows Defender Scanning

Exclude the Controller data directory (`<controller_home>\db\data`), or simply the entire Controller directory, from scanning by Windows Defender. If you are not sure whether Windows Defender is running on the system, check for it in your local Services list. You can either configure the Controller data directory to be excluded in the Windows Defender Control Panel, or disable the service altogether if it is not needed.

For details on how to view services and exclude directories in Windows Defender, refer to the documentation for your version of Windows.

Windows Indexing Service

Ensure that the Windows indexing service is configured to ignore the Controller data directory (`<controller_home>\db\data`), or simply the entire Controller directory.

The data directory does not contain any files that are meaningful to the indexer, so it can be excluded from indexing. To exclude the directory from indexing, you can add the directory to the excluded directories list in the **Indexer Control Panel**, disable indexing in directory preferences, or stop the indexing service entirely.

To add the directory to the excluded directory list, follow these steps:

1. From the **Control Panel Indexing Options** dialog, click the **Modify** button.
2. In the **Indexed Locations** dialog, navigate to and select the **Controller** data directory.
3. Clear the checkbox for the data directory and click **OK**.

Windows Update

Configure the Windows Update preferences so that the server is not automatically restarted after an update. To configure the restart policy:

1. Open the **Local Group Policy Editor** dialog (search for and run the `gpedit` executable).
2. Navigate to the Windows Update component. In the tree, you can find it under **Local Computer Policy > Computer Configuration > Administrative Templates > Windows Components**.
3. Double-click on the **No auto-restart...** setting.
4. Select the **Enabled** option and click **Apply**.

.NET Framework

Components of the .NET Framework 3.5 are required to allow the Controller to be installed as a Windows service on the target machine. The installer checks your system and indicates if .NET 3.5 is not found. Follow the instructions on the Enterprise Console to get the required components.

Even if you have the latest version of .NET installed, you still have to install .NET 3.5. This is due to a Glassfish requirement where the Glassfish launcher explicitly requires .NET 3.5.

Windows 7

The Controller is automatically installed as a Windows service. Windows 7 operating system must have the hotfix described in http://support.microsoft.com/kb/2549760. This hotfix ensures that the Windows registry modifications made by the installer to extend the default service timeouts work as expected. The installer checks for the presence of the hotfix and warns you if it is not found.

Controller Data and Backups

This page provides an overview of how to configure and administer Controller data storage.

About Controller Data Storage

The Controller requires persistent data storage to store the following type of information:

- Design of your applications (all metadata about business transactions, tiers, policies, and so on)
- History of the performance of your applications (metric data)
- Transaction snapshot data and events
- History of incidents that occurred (both resolved and unresolved incidents)

Controller Data Directory Location

By default, the AppDynamics Controller uses MySQL as its storage mechanism. The Controller bundles a MySQL instance with the Controller. At installation time, the Enterprise Console creates the necessary tables and artifacts in the database.

By default, the database files and data are stored in: `<controller_home>/db`.

Manage the Database User Password

The Enterprise Console creates the user account that the Controller uses to log into the database to perform database-related operations. The username of the account is "root", and the password is the one you supply to the Enterprise Console during the Controller installation process.

When attempting to access the data, the Controller reads the database user password from these sources and in the priority shown:

- From the `MYSQL_ROOT_PASSWD` environment variable
- From user input to a command line prompt

If you do not keep the password in the environment variable, you will need to supply it in response to a command-line prompt whenever performing an operation that involves accessing the database, starting the database, stopping the database, or logging into the database.

Moving the Controller Data Directory

After installation, you can move the data directory to a new location. This may be necessary, for example, if there is not enough disk space available during Controller installation.

If you are using symlinks, you must create the symlink outside of the root Controller install directory and move the data directory to the new volume after you install the Controller.

Warning: Do not mount a file system on `<controller_home>/db/data`. During Controller upgrade, the Enterprise Console moves the data directory to `data_orig`. Upgrade will fail if the Enterprise Console cannot complete this move.

You can also update the `datadir` path on the Controller Database Configurations page of the Enterprise Console GUI.

To relocate the Controller data directory

1. Stop the Controller and its database. See Start or Stop the Controller.
2. Modify following properties in the `<controller_home>/db/db.cnf` file to point to the new location of the data directory.

   ```
   datadir
   tmpdir
   log
   slow_query_log_file
   ```

3. Copy (or move) the existing data directory `<controller_home>/db` to the new location.
   For example, to copy the data on Linux:

   ```
   cd <controller_home>/db/
   cp data <new-location>
   ```

4. Start the Controller. See Start or Stop the Controller.
5. Check the `database.log` and `server.log` for any errors related to the database connection.

Controller Data Backup and Restore

Related pages:

- Enterprise Console Back Up and Restore

AppDynamics strongly recommends that you perform routine data backups of the Controller.

One method of maintaining backups of the Controller is to implement high availability. With high availability, the database on the secondary Controller keeps a replicated copy of the data on the primary Controller. A secondary Controller also makes it practical to take cold copies of the Controller data, since you can shut down the secondary to copy its data without affecting Controller availability. For information on HA, see Controller High Availability (HA).

Other approaches include using a disk snapshot mechanism or using database backup tools. The BackupTools section describes tools that support each approach. In addition to regular backups, back up the Controller and Enterprise Console before upgrading or migrating them from one server to another.

This page provides an overview of the tasks and considerations related to backing up the Controller. Note that your Controller should be shut down before performing any import functions.

It is to be noted that controller versions 4.3 and later will work only on restoring and backing up the `<Controller Home>/.appd.scskeystore` file.

Best Practices for Backups

Backing up the entire system each night may not be feasible when dealing with the large amount of data typically generated by a Controller deployment. To balance the risk of data loss against the costs of performing backups, a typical backup strategy calls for backing up the system at different scopes at different times. That is, you may choose to perform partial backups more frequently and full backups less frequently.

The scope of a Controller backup can be categorized into these levels:

- Level 1: A light backup of the installation environment only
- Level 2: A metadata backup involving all metadata associated with the installation except for big data tables.
- Level 3: Backs up all data, either by performing a cold backup of the /data directory or a hot backup using a third-party tool.

A possible backup strategy may be to perform a level 1 and level 2 backup very frequently, say nightly, and a level 1 and level 3 backup about once a week. In addition to performing a level 1 or 2 backup, you should also back up the data for the Enterprise Console with `mysqldump` on a regular basis. A level 3 backup also backs up the Enterprise Console data. See Enterprise Console Back Up and Restore for more information.

Light Backup (Level 1)

A light backup targets Controller configuration files like `db.cnf` and `domain.xml`. This type of backup lets you avoid having to reconfigure the Controller in case of machine failure.

To perform this type of backup, simply copy everything in the Controller installation directory EXCEPT the data directory.

While it is recommended that you copy the entire Controller home except for the data directory when performing a light backup, particularly before performing a Controller upgrade, there are scenarios in which you may wish to copy only site-specific configuration files. This may be the case if you are migrating an existing Controller configuration to a new Controller installation, for example. For a list of those files, see Migrate the Controller.

Metadata Backup (Level 2)

A metadata backup exports the data that encapsulates the environment monitored by the Controller. Metadata defines the applications monitored by the Controller, business transactions, policies, and so on. It does not include what can be thought of as "runtime data", the big data tables that contain the metrics, snapshots, events, and top summary stats (top SQL, top URLs, and so on) generated in the monitored environment. By backing up metadata, you can avoid having to reconfigure monitored applications in the Controller in the event of a failure.

To perform this type of backup:

1. Run the script described in Using mysqldump to back up the Controller.
2. Then augment it with a copy of:
    a. The Controller Java keystore (600 byte file): `<controller install>/.appd.scskeystore`
    b. And the Enterprise Console keystore: `platform-admin/.appd.scs`

Complete Backup (Level 3)

A complete backup saves all runtime data associated with the Controller installation. It captures the actual metrics data, snapshots, and so on.

Some third-party backup tools, such as Percona XtraBackup, do not rely on transactions so you can perform a hot backup of your system (that is, back up the Controller database while it is running).

You can perform a complete backup as either:

- A cold backup of the `/data` directory. A cold backup means that, with the Controller app server and database shut down, you create an extra copy of the Controller database using, for example, the `cp -r` command, the `tar` utility, `rsync`, or others.

- A hot backup, which means the Controller is running. For a hot backup, use a third-party tool such as `Xtrabackup`.

This type of backup allows you to skip the Level 2 backup and the Enterprise Console backup. You still need to perform the Level 1 backup to back up the Controller home directory.

Backup Tools

This section lists a few third-party tools that you can use to back up Controller data. The list is not exhaustive; you can use any tool capable of backing up MySQL data with the Controller. It is up to you to test your backup and restore process. However, the tool you decide on should back up the data as binary data.

For Linux systems:

- Percona XtraBackup

For Windows systems:

- Zmanda Recovery Manager for MySQL

An alternative to using a database backup tool is to use a disk snapshot tool to replicate the disk or partition on which the Controller data resides. Options include:

- ZFS volume manager. For more information, see Using ZFS methods for data backup.

Details for performing this type of backup are beyond the scope of this documentation. For more information, refer to administration documentation applicable to your specific operating system.

Back Up the Controller with mysqldump

The `mysqldump` utility is a MySQL backup tool that is included with the Controller instance of MySQL.

While `mysqldump` is not recommended for use on large data tables, such as the Controller metric data tables, it is useful for backing up Controller metadata. Metadata defines the monitored domain for the Controller, including applications, business transactions, alert configurations, and so on.

The following instructions assume that the binary path for the Controller's MySQL instance is in the PATH variable. The path to the Controller's instance of MySQL must precede any other MySQL path on your system. This prevents conflicts with other database management systems on your machine, such as a MySQL instance included by default with Linux.

The database binary files for the Controller database are in `<controller_home>/db/bin`.

> ⚠ Before using `mysqldump`, first ensure that the Controller app server is stopped. If you attempt to run `mysqldump` while the app server is running, it will severely degrade the performance and stability of the Controller.

To use `mysqldump`, run the `mysqldump` executable, passing the root username, password, and output file. The executable is located in the following directory:

`<controller_home>/db/bin`

The command should be in the form:

```
mysqldump -u root -p<password> <ignore-table_statements> > /tmp/metadata_dump.sql
```

For a full example that shows which tables to exclude for a metadata backup, see the contents of the metadata backup script described in the next section.

Sample mysqldump Script

The following script illustrates how to use `mysqldump` to export Controller metadata while excluding runtime data tables by script.

- Linux: ControllerMetadataBackup.sh.txt

> ⚠ Backing up the Controller with a custom MySQL data directory location using mysqldump will result in an incomplete and unusable metadata dump. The default location for the Controller's MySQL data directory is `appd_install_dir/db/data`. If you do not see the data directory here then that means an alternate directory or mount point was selected and configured for your MySQL data directory.
>
> To fix this issue:
>
> 1. Examine `<Controller Home>/db/db.cnf`
>   a. Locate the `datadir` parameter. It contains the path to the MySQL data directory on your Controller host.
> 2. Edit the metadata backup script
>   a. Replace `$appd_install_dir/db/data` with the `datadir` path you located in `db.cnf`

To use the script:

1. Download the version appropriate for your operating system.
2. Rename the file to remove the .txt extension.
3. Modify the contents of the file as described in the script comments.

Import Controller Data with mysql

When you restore or migrate the Controller, you can import the data you exported with `mysqldump`.

Shut down your Controller before using the following command to import the data into a database:

```
$install/db/bin/mysql -u controller -p<ControllerDBpassword> < metadata_dump.sql
```

It will overwrite the tables. You can clone your installation to another host and test your restoration there.

A Controller that is installed with a custom MySQL data directory location requires additional flags.

Sample Data Backup Script

The following script uses Percona XtraBackup to back up Controller data. To use it, you need the `percona-xtrabackup` or `xtrabackup` and qpress pac kages. For information on installing XtraBackup, see the Percona installation documentation.

To use the script, download the following file:

- appdynamics-backup.sh.txt

Rename the script (by removing the .txt extension). In the script:

- Verify or edit the values of the `CONTROLLER_HOME` and `DESTINATION` variables at the beginning of the script for your environment.
- Edit the if/then/else clause at the end of the script if you want to implement backup file rotation, call your enterprise backup system to pick up the compressed Controller database image, or send an alert if the backup fails for any reason.

The following commands demonstrate how to restore a compressed backup image:

```
mkdir /path/to/big/staging/folder

# unpack the compressed backup archive
cd /path/to/big/staging/folder && xbstream -xv < /path/to/backups/dir/controller-yyyymmdd.xbstream

# decompress the backup image and apply the log taken during backup
CONTROLLER_HOME=/path/to/AppDynamics/Controller && cd /path/to/big/staging/folder \
&& innobackupex --decompress --parallel=16 . && innobackupex \
--defaults-file=$CONTROLLER_HOME/db/db.cnf --use-memory=1GB --apply-log --parallel=16 .

# Move a prepared backup into an empty controller data directory
CONTROLLER_HOME=/path/to/AppDynamics/Controller && cd /path/to/big/staging/folder \
&& innobackupex --defaults-file=$CONTROLLER_HOME/db/db.cnf --move-back .
```

For more information on these options, see the Percona innobackupex option reference.

Using a Backup to Migrate to a New Physical Server

You can use either a hot or cold backup procedure to migrate Controller data to a new server. However, we recommend performing cold backups. While a hot backup does not bring down the Controller for an extended amount of time, it does introduce the possibility of data loss, since hot backups capture the state of the data only when the hot backup starts.

To perform a cold backup, simply shut down the Controller and back up the data directory located in `<controller_home>/db`.

Controller Disk Space and the Database

This page discusses best practices for managing disk space for the MySQL database used by the Controller.

Disk Space Considerations

To ensure database integrity, the Controller automatically shuts itself down when available disk space falls below 1 GB.

Before it reaches that point, the Controller displays a low disk space alert in the UI and writes an error level event to server.log. The point at which the Controller generates the alert depends on its profile, as follows:

- For large and extra large profiles: 10 GB or less
- For all other profiles: 2 GB or less

The Controller shuts itself down when there is less than 1 GB on the disk regardless of the Controller profile type.

> ⚠ It's important to note that the Controller monitors the disk or partition that it is installed on. If the Controller data resides on a different disk or partition from the Controller home directory, you will need to monitor available space on that disk or partition separately.

Managing Disk Space

If the disk space is low, you need to reduce the size of the Controller database.

To manage how much disk space the Controller database uses, you can change the amount of data retained in the Controller database. See Database Size and Data Retention.

Database Size and Data Retention

This page provides both the on-premises and SaaS default data retention periods for data stored by the Controller and instructions for modifying on-premises values. AppDynamics manages SaaS Controller deployments, which eliminates the need for manual modifications.

Migrate the Controller

Related pages:

- Controller High Availability

This page describes how to migrate a Controller from a physical or virtual machine (VM) to a new physical machine.

Before Starting

Migrating the Controller often results from the need to move the Controller to new hardware due to increased load. Before starting, make sure that the new hardware meets the AppDynamics requirements as described in Controller System Requirements. Specifically, you should review the Controller hardware performance profiles and the hardware requirements per profile information to verify that the target Controller hardware meets the RAM size and Disk I/O requirements.

You will need to update the MAC address associated with your license since licenses are tied to the machine MAC addresses. You can also acquire new license files for the new Controller hardware. Send the MAC addresses to salesops@appdynamics.com and request a new license file or two new licenses, if upgrading to an HA pair. See Apply or Update a License File for more information.

If you are performing a migration and upgrade for a 4.3 version Controller, you should first migrate the Controller. Then, you can upgrade the Controller to 4.5 or higher by installing the Enterprise Console and using the Discover & Upgrade feature. This also applies to migrations involving different OS environments.

VMotioning, or migrating a VMware guest with a running Controller inside it from one host to another, is not supported. Doing so will lead to dropped metrics and UI performance problems.

Migrating a Linux Controller

You can use the high availability features provided by the Enterprise Console to migrate a Controller from one machine to another. It is assumed that the Controller you need to migrate is already managed by the Enterprise Console (it has been installed or discovered by the Enterprise Console). See Enterprise Console for more information.

You add the new host as an HA pair to the old host, set the new host as active, and then remove (decommission) the old host. When finished, the Controller will run on the new host.

Before starting, you should review the requirements and concepts related to Controller High Availability.

To migrate a Linux Controller:

1. Log in to the **Enterprise Console** UI interface.
2. Select the Platform containing the host you want to migrate.
3. In the **Hosts** page, add the new host (the one to serve as the new target host) and provide the credentials for connecting to that host.
4. In the **Controller** page, select **Add Secondary** and select the new target host. Provide the DB root password and Controller root password, and select **Submit**.
5. In the **Controller** page, select **HA failover**.
   The Primary Controller is now running on the new host.
6. Update the license MAC address or apply a new license for the new machine. See Before Starting for more information.
7. Decommission the old Controller from the **Controller** page:
   a. Select **Remove Controller**, or run the following command on the Enterprise Console host:

```
platform-admin.sh submit-job --job remove --service controller --args removeBinaries=false
```

   b. Select the **remove binaries** option. (Do not select **Remove entire cluster**.)

The Controller is now running on the newly provisioned host.

You can keep the same access key from the old Controller. To migrate or update your access key, see Controller Secure Credential Store.

You must update the license rule access keys.

Migrating a Windows Controller

Since high availability features are not available on Windows, you must use an alternative procedure to migrate a Controller from one machine to another. You use the Enterprise Console to manually install and move the `.appd.scskeystore` and the `datadir` from the old host to the new host. Once completed, the Controller will run on the new host.

Before starting, you should review the requirements and concepts related to Controller High Availability.

To migrate a Windows Controller:

1. Install the Enterprise Console on the new Controller host from where you are running the existing Controller host.
2. Use the Enterprise Console to install the same version of the Controller on your new Controller host using the same passwords as on the existing Controller host.
3. Shut down the Controller Appserver and database on both Controller hosts.
4. Move `<controller_home>/.appd.scskeystore` and the Controller's MySQL `datadir` on the new host.

   > When migrating the data, ensure that the destination MySQL version is the same as the source version.

5. Copy `<controller_home>/.appd.scskeystore` and the Controller's MySQL `datadir` from the old host to the correct locations on the new host.
6. Start the Controller Appserver and database on the new host.
   The Controller is now running on the newly provisioned host.

You can keep the same access key from the old Controller. To migrate or update your access key, see Controller Secure Credential Store.

> You must update the license rule access keys.

Prepare Virtual Machines for the Controller

The following are considerations and requirements for the machine hosting the Controller.

Controller Sizing Restrictions

Demo, small, and medium profile Controllers can run on virtual machines that meet the performance requirements otherwise specified. Large deployments are not supported except as indicated in Controller System Requirements.

> ⚠ The Controller is not supported on virtual machines with oversubscribed physical CPUs like T2 AWS instances. These Burstable Performance Instances are CPU-throttled and do not have dedicated storage bandwidth. We recommend you use a Fixed Performance Instance type instead.

Fully Reserved RAM

The memory allocation for the Controller's virtual machine must be fully reserved RAM. Reserve as much as possible of the total memory allocation.

On VMWare VMs

Reserved Memory Configuration

For information on how to configure reserved memory on VMWare, see Set Memory Reservation on a Virtual Machine.

Trend Micro Configuration

On vSphere platforms shipped with Trend Micro, the Trend Micro process needs to be disabled in order for replication jobs to work. You can also add the entire Enterprise Console and Controller directories to the exclusion list to avoid this conflict.

See Trend Micro and VMware Virtualization for more information.

On Hyper-V VMs

On Microsoft Hyper-V, "Dynamic Memory" needs to be disabled and "Static Memory" needs to be enabled.

To disable a Hyper-V VM from using Dynamic Memory:

1. Open the Hyper-V Manager.
2. Select the VM you want to configure in the **Virtual Machines** pane, making sure the VM is powered off.

   > ⓘ You cannot enable or disable Dynamic Memory if the VM is in either the Running or Saved state.

3. Right-click the VM to bring up the context menu.
   a. Select **Settings**.
   b. Click the **Memory** page.
4. Uncheck the **Enable Dynamic Memory** box.

See Virtualization: Optimizing Hyper-V Memory Usage for more information.

Licenses on VMs

The Controller license is bound to the MAC address of the host machine. To run the Controller on a virtual machine, you must ensure that the host virtual machine uses a fixed MAC address.

I/O Performance Requirements

A factor that often limits the performance of the Controller is the underlying disk I/O performance of the host machine.

Virtual machines, in particular, are less apt to provide sufficient I/O performance requirements for the Controller, compared to similarly specified physical machines. In any case, whether you are deploying the Controller to a physical or virtual machine, you need to ensure that the machine meets the I/O performance requirements set forth in Controller System Requirements.

Host Name Entry

The fully qualified hostname for the application server is the address at which Controller UI users and application agents will use to access the Controller. You specify this hostname when you install the Controller. The hostname needs to be in the /etc/hosts file on the machine.

The following example shows an entry in `/etc/hosts` with the IP 21.43.65.987, the fully qualified hostname `application1.mycompany.com` and the alias `app1`:

```
21.43.65.987 application1.mycompany.com app1
```

Elastic Network Interface (ENI)

For AWS, provision an ENI for each Controller host and link the license to the ENI. For more information about ENI, see the AWS documentation at the following link:

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-eni.html.

AWS Controller Deployment Guide

The AppDynamics Controller is certified to run on an AWS environment with the Aurora Database. This page provides information on installing, configuring, and administering a Controller deployment on AWS with Aurora Database.

Installation Overview

You can deploy a medium- or large-scale Controller in AWS using Aurora. Aurora provides higher performance than MySQL, which allows you to scale your Controller to handle more metrics.

Before you install the Controller, review the requirements for the components you plan to install and prepare the host machines. The requirements vary based on the components you deploy and the size of your deployment.

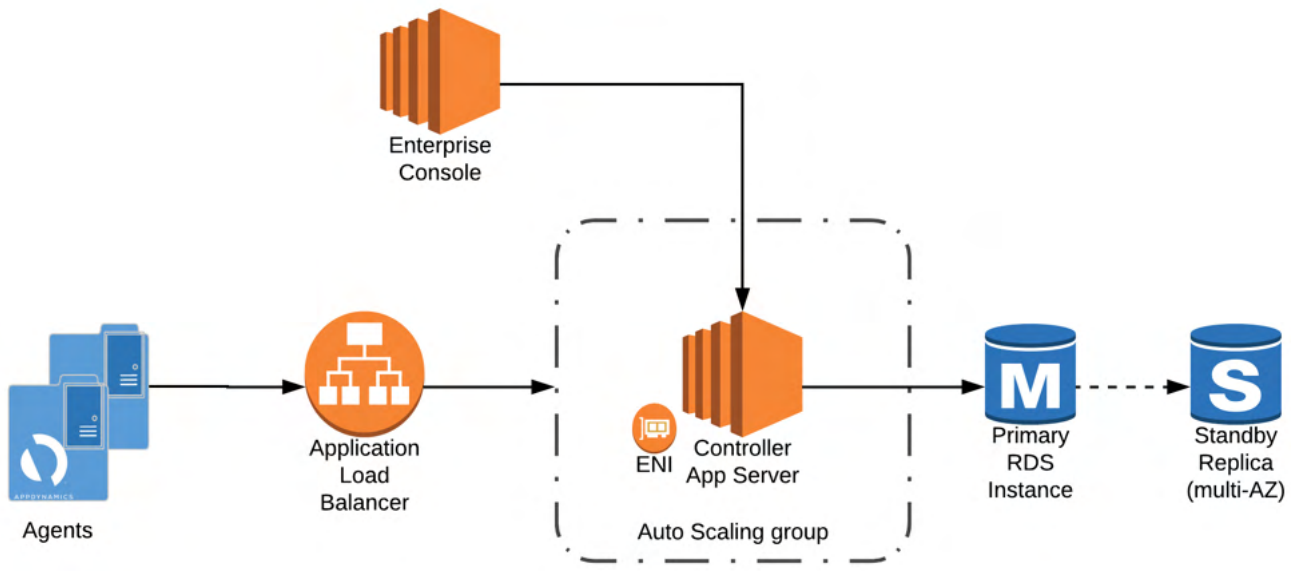
You can manually deploy your Controller on AWS. See Deploy the Controller on AWS
This deployment requires attention and time because you have to set up all of the configurations. However, this gives you freedom to customize your deployment.

You use the Enterprise Console to deploy the Controller by specifying Aurora as the database type. The Enterprise Console is the installer for the Controller, and you can use it to manage the entire lifecycle of new or existing AppDynamics Platforms and components.

You can get the software for installing the platform components from the AppDynamics download site. See Download AppDynamics Software

AppDynamics on AWS Architecture

The following diagram depicts the components of an AppDynamics Controller deployment on AWS.



Configure an Application Load Balancer in front of the Controller and ensure SSL terminates at the Elastic Load Balancer.

With Amazon Relational Database Service (RDS), you no longer need to worry about database backups, as it takes care of this for you. Also, you no longer need to implement high availability (HA) on your own, since you can instead leverage the Standby Replica that Aurora/RDS offers and the Aurora database is horizontally scalable. With the multi-AZ deployment option, Aurora offers 99.95% availability.

Controller High Availability and AWS

- If your data is migrated to the Aurora DB, you can create a new Controller from the Amazon Machine Image (AMI) in case of failure. With Aurora as the database, HA scenarios are not required because the Aurora database is horizontally scalable.
- It is a good practice to cut the new root AMI every time you make a configuration change to the Controller.
- You can configure a read replica instance while creating Aurora DB, which should satisfy most database replication requirements.
- You can configure the backup policy while creating the Aurora DB and modify it later.

Prepare the AWS Machine for the Controller

Related pages:

- Prepare Linux for the Controller
- Prepare Windows for the Controller

This page provides considerations and requirements for AWS instances that host the Controller.

Instance Sizing

For AWS instance sizing by metric ingestion rate, see the Controller Sizing table.

The actual metrics generated can vary greatly depending on the nature of the application and the AppDynamics configuration. Be sure to validate your sizing against the metric ingestion rate before deploying to production.

Elastic Network Interface (ENI)

For AWS, provision an ENI for each Controller host and link the license to the MAC address of the ENI. For more information about ENI, see the AWS documentation at the following link:

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-eni.html.

Deploy the Controller on AWS

This page describes the procedure used to deploy the AppDynamics Controller to an AWS environment.

You can set custom configurations when you manually deploy the Controller to AWS. From these steps, you manually set up security groups, database parameter groups, Amazon Relational Database Service (RDS), Aurora DB instance, EC2 instance, Elastic Network Interface (ENI) for the Controller, DNS CNAMEs, and listeners for your load balancer.

Afterward, you install the Enterprise Console, and then use the Enterprise Console to install the Controller and configure it for the AWS environment.

Based on the Amazon Machine Image (AMI), you can provision a replacement EC2 instance for the Controller as needed.

For help with your deployment, contact your AppDynamics account, sales, or professional services representative.

Before You Begin

- Review if Amazon Aurora DB is available in your region – Check the AWS Region Table in the AWS documentation to see if the Amazon Aurora - MySQL-compatible service is available in your particular region.
- Amazon RDS Password Requirements – There are some naming constraints in the Amazon Relational Database Service (RDS). The master password for Aurora DB can be any printable ASCII character except "/", """, or "@", and must contain 8 to 41 characters. Master password constraints differ for each database engine.
  For details on naming constraints in Amazon RDS, see the AWS documentation.

Deploy the Controller on AWS

To manually deploy the Controller on AWS:

1. Create Security Groups
2. Create Custom DB Parameter Groups
3. Launch an Amazon RDS Aurora DB Instance
4. Create Database User for Controller
5. Launch an EC2 Instance for the Controller
6. Create the ENI for the Controller
7. Create DNS CNAMEs
8. Install the Enterprise Console in an AWS Environment
9. Install the Controller in AWS Using Aurora
10. Apply Controller Optimizations
11. Configure a Load Balancer
12. Configure Listener Rules

Troubleshooting the Installation

Issue: Controller EC2 Instance Stops

If the Controller EC2 instance stops almost immediately after you install the Controller, there may be an issue with your EBS devices. AWS may report that it is not able to boot from the volumes. If the EC2 machine stops, check and update your EC2 volumes so that they are mounted correctly.

Create Security Groups

A security group acts as a virtual firewall for your instance to control inbound and outbound traffic. For each security group, you add rules that control the inbound traffic to instances, and a separate set of rules that control the outbound traffic.

At a minimum, we recommend creating the following security groups when deploying AppDynamics in AWS using Aurora DB.

> You can create additional security groups to align with your organization's standards.

Required Security Groups

Use the instructions provided in the AWS security groups documentation to create these required security groups:

Security Group for the AppDynamics Enterprise Console

Security group name: `appd-ec-security-group`

Inbound rule: Allow all inbound TCP traffic on ports 22 and 9191

Outbound rules:

- Allow outbound TCP traffic to `appd-appserver-security-group` on port 22
- Allow outbound TCP traffic to `appd-db-security-group` on port 3388

Security Group for the AppDynamics Controller Appserver

Security group name: `appd-appserver-security-group`

Inbound rules:

- Allow all inbound TCP traffic on port 22
- Allow inbound TCP traffic on ports 8090-8097 from `appd-elb-security-group`

Outbound rule: Allow outbound TCP traffic to `appd-db-security-group` on port 3388

Security Group for AppDynamics Database Instances

Security group name: `appd-db-security-group`

Inbound rule: Allow inbound traffic on port 3388 from `appd-appserver-security-group` and `appd-ec-security-group`

Outbound rule: No outbound access allowed

Security Group for Load Balancer in Front of the AppDynamics Controller

Security group name: `appd-elb-security-group`

Inbound rule: Allow all inbound HTTPS traffic on port 443

Outbound rule: Allow outbound TCP traffic to `appd-appserver-security-group` on ports 8090-8097

Create Custom DB Parameter Groups

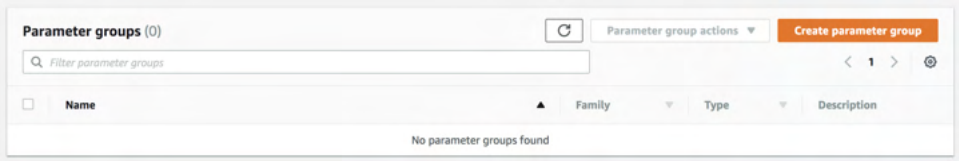
You must modify some of the parameters for your database instance.

> Any parameters not modified retain their default values.

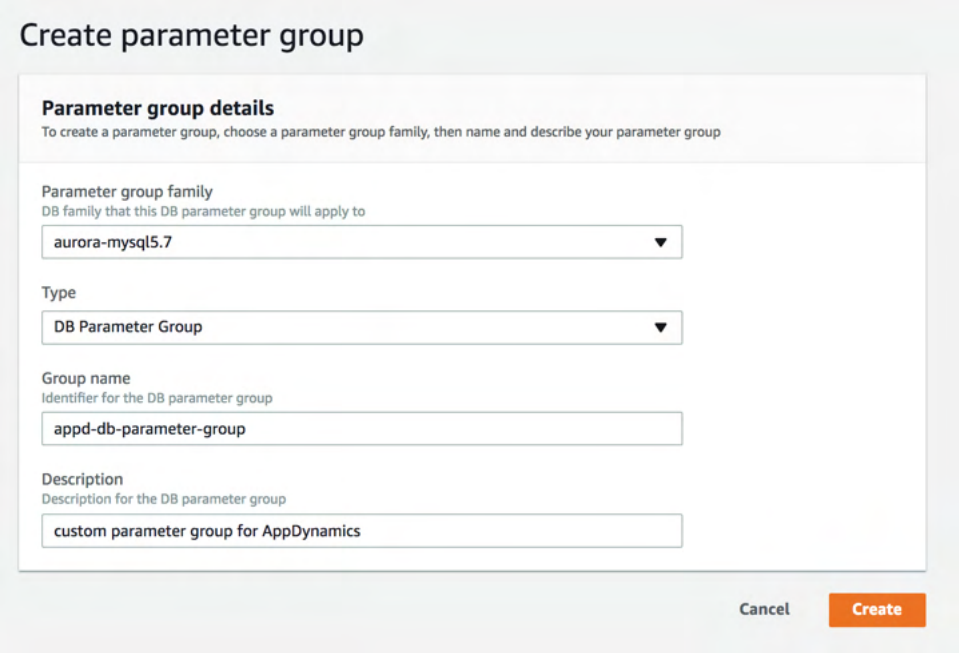
Parameter	Recommended Value
innodb_file_format	Barracuda
innodb_large_prefix	1 (0 for Aurora 5.6)
innodb_lock_wait_timeout	180
innodb_max_dirty_pages_pct	20
lock_wait_timeout	180
log_bin_trust_function_creators	1
max_allowed_packet	104857600
max_heap_table_size	1610612736
query_cache_type	0
sql_mode	0
tmp_table_size	67108864
wait_timeout	31536000

To create custom DB parameter groups:

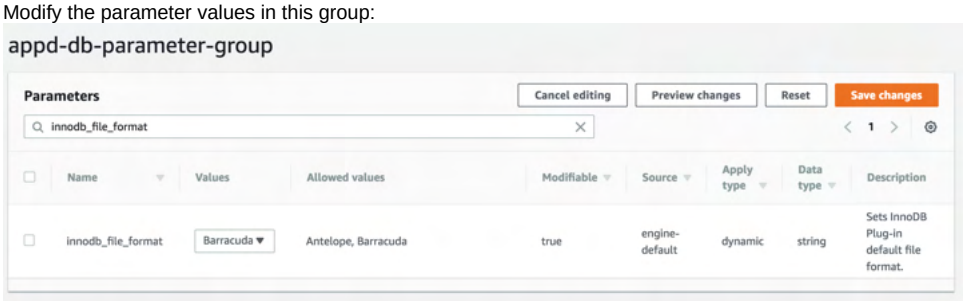
1. Navigate to the Parameter groups page on the Amazon RDS in the AWS console.
2. Click **Create parameter group** on the top right of the page.



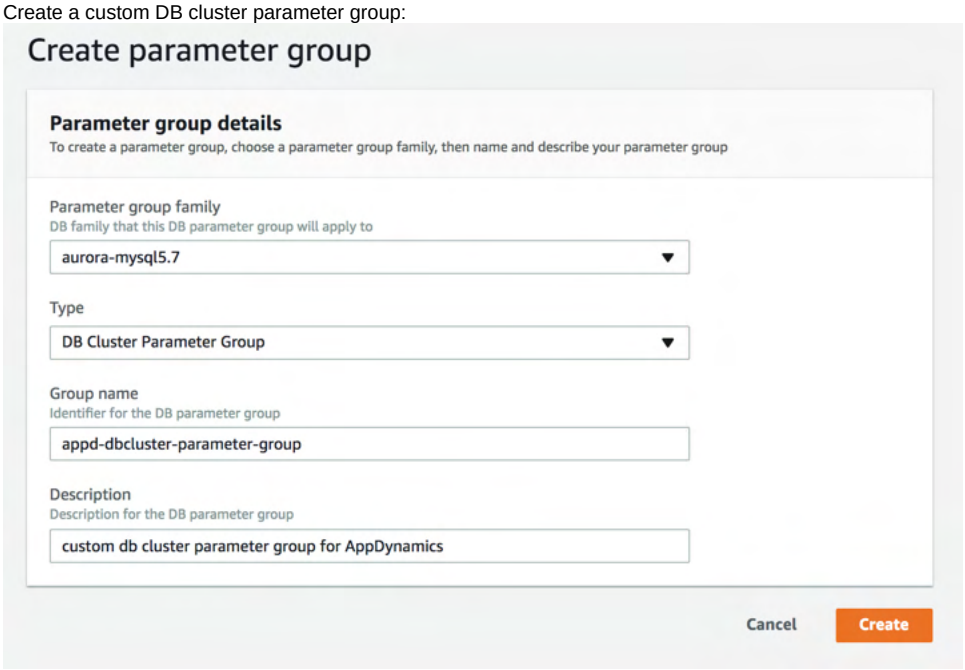
3. Enter the group details:



4. Modify the parameter values in this group:



5. Create a custom DB cluster parameter group:



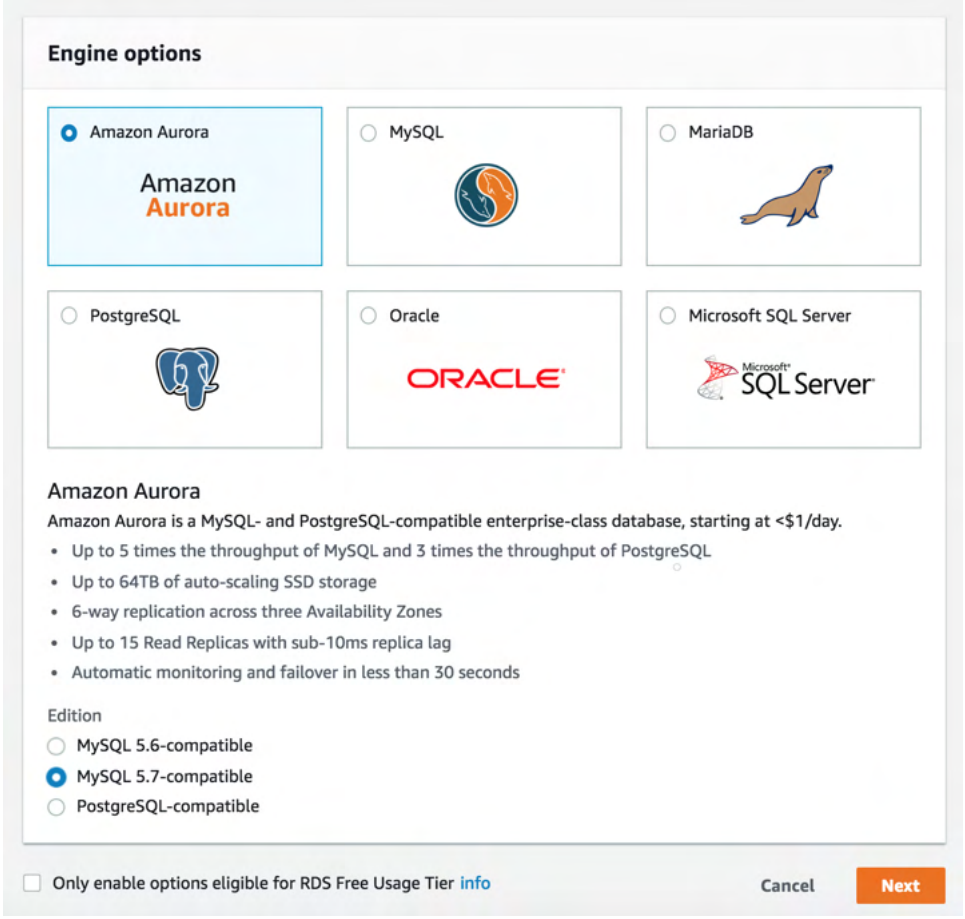
6. Modify the parameter values in this group as follows:

Parameter	Recommended Value
character_set_client	utf8
character_set_connection	utf8
character_set_database	utf8
character_set_filesystem	binary
character_set_results	utf8
character_set_server	utf8
collation_connection	utf8_general_ci
collation_server	utf8_unicode_ci
innodb_default_row_format	DYNAMIC
innodb_file_per_table	1
lower_case_table_names	1

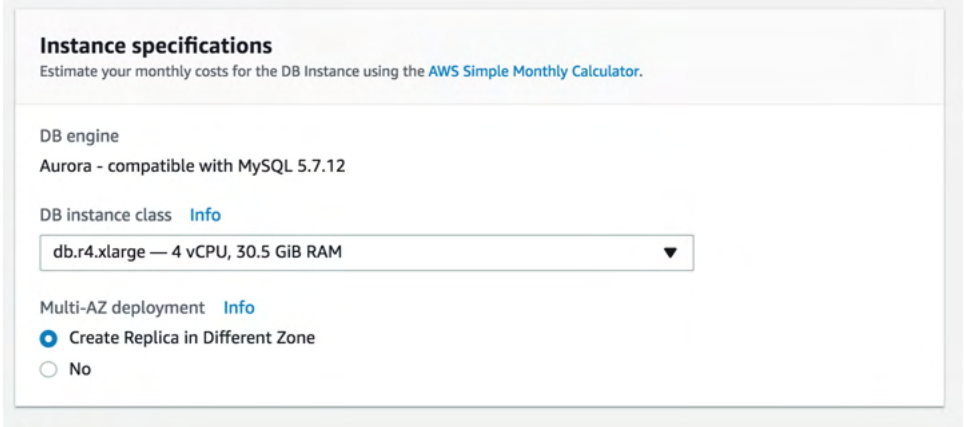
Launch an Amazon RDS Aurora DB Instance

You launch an Amazon RDS Aurora DB Instance from the AWS console.

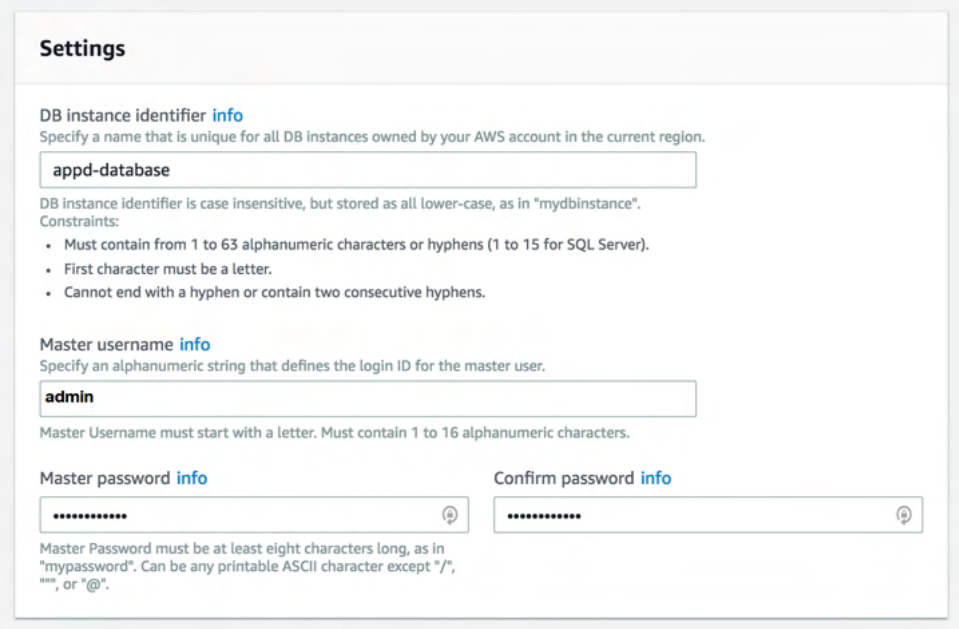
1. Select **Aurora** with the **MySQL 5.7-compatible** option.



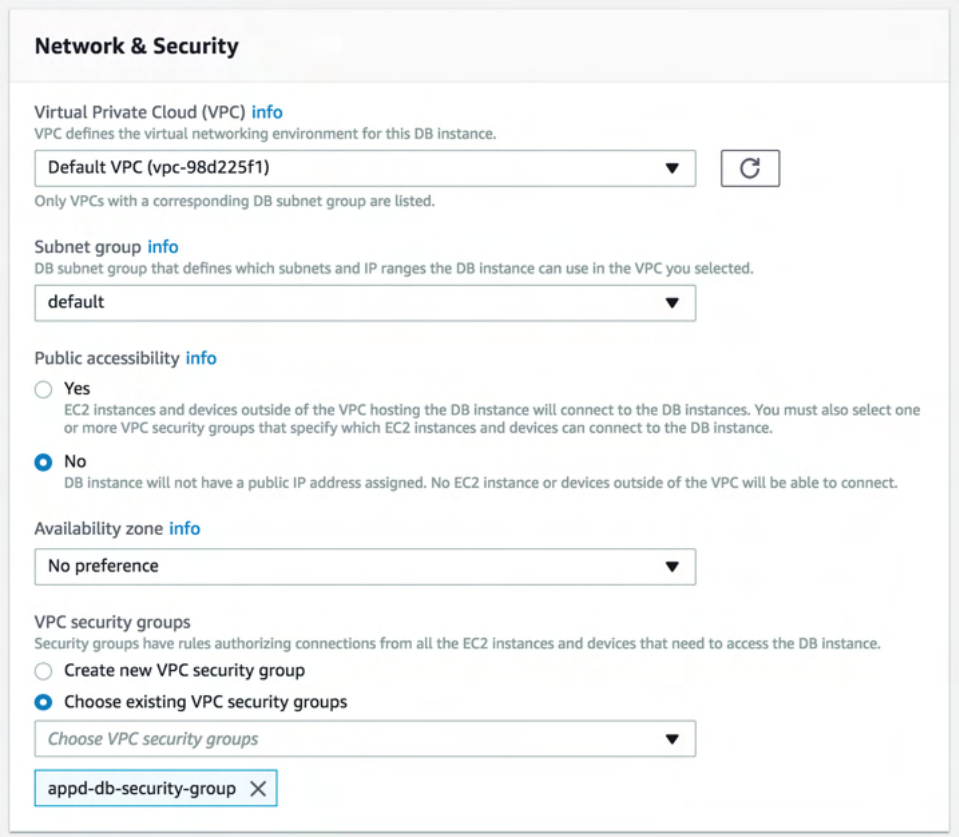
2. Select the desired DB Instance class. See Prepare the AWS Machine for the Controller for instance sizing information
3. Select **Create Replica in Different Zone** to have Amazon RDS maintain a synchronous standby replica in a different Availability Zone than the DB instance. If a planned or unplanned outage of the primary occurs, Amazon RDS will automatically fail over to the standby replica.



4. Enter an identifier for the DB instance. The master username must be `admin`.

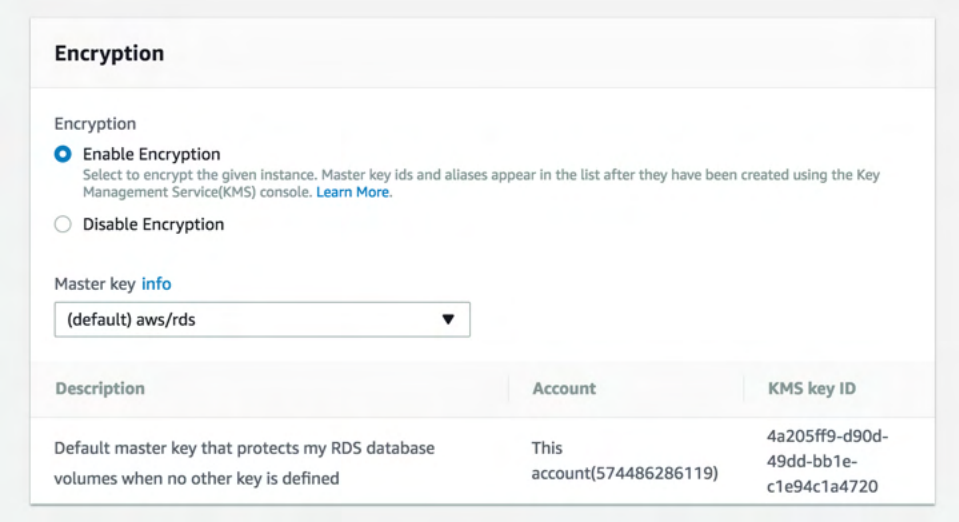


5. Select the Default VPC and subnet group. The DB should not be accessible publicly, so select **No** for this option.
6. For the security group, select the **appd-db-security-group** that you created previously.



7. For the database options, you do not need to specify the DB cluster identifier, nor enter a database name because the installer creates the necessary databases for you.
8. Use the default database port of 3388.
9. Specify the custom parameter groups that you created previously.

10. We recommend that you select **Enable Encryption** for data at rest.



11. Use the default options for the remaining settings.
12. Click **Launch DB Instance**, and your new database will be available in a few minutes.

Create Database User for Controller

During installation, AppDynamics must create additional databases and users in the Aurora database for the AppDynamics Controller application to interact with the Aurora database server.

To create the Aurora database:

1. Create the Aurora database using `admin` as the primary username.
2. After the Aurora database instance is created successfully, log in to the ec2 instance as `admin`:

```
mysql -u admin -h <rds-aurora-endpoint> -P 3388 -p
```

3. To create a new `'root'` user, enter:

```
CREATE USER 'root'@'%' IDENTIFIED BY 'controller';
```

4. To check for the grants of the primary username (`admin`), enter:

```
mysql> SHOW GRANTS FOR admin;
```

   Resulting output:

```
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-------------------------------------------------
Grants for admin@%
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-------------------------------------------------
GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, PROCESS, REFERENCES, INDEX, ALTER, SHOW
DATABASES, CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, REPLICATION SLAVE, REPLICATION CLIENT, CREATE
VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER, LOAD FROM S3, SELECT INTO
S3, INVOKE LAMBDA ON *.* TO 'admin'@'%' WITH GRANT OPTION
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-------------------------------------------------
 1 row in set (0.00 sec)
```

5. Apply the grants (listed in the output) for the new `root` user that you created in Step 1. The `root` user will have the same grants as the `admin` user.

```
mysql> GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, PROCESS, REFERENCES, INDEX, ALTER,
SHOW DATABASES, CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, REPLICATION SLAVE, REPLICATION CLIENT,
CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER, LOAD FROM S3, SELECT
INTO S3, INVOKE LAMBDA ON *.* TO 'root'@'%' WITH GRANT OPTION
```

   Resulting output:

```
Query OK, 0 rows affected (0.01 sec)
```

6. Once the `root` user has the same privileges as the primary username `admin`, verify that you can log in to the database as `root`, and then continue with the installation.

   - If you do not have users "root@*x.x.x.x*" and "root@ip-*x-x-x-x*.ec2.internal", ignore these users and continue to work with the `root@%`.
   - If you have users "root@*x.x.x.x*" and "root@ip-*x-x-x-x*.ec2.internal", then instead of using the previous `GRANT` command, use this `GRANT` command:

```
mysql> GRANT ALL ON `%`.* TO 'root'@'ip-x-x-x-x.ec2.internal' identified by 'controller' WITH
GRANT OPTION;
mysql> GRANT ALL ON `%`.* TO 'root'@'x.x.x.x' identified by 'controller' WITH GRANT OPTION;
mysql> GRANT RELOAD ON *.* TO 'root'@'ip-x-x-x-x.ec2.internal' identified by 'controller' WITH
GRANT OPTION;
mysql> GRANT RELOAD ON *.* TO 'root'@'x.x.x.x' identified by 'controller' WITH GRANT OPTION;
```

After installation, you can revoke the primary-level privileges from the Aurora *root* user without interfering with the Controller. However, primary-level privileges for Aurora *root* user are required prior to upgrading the Controller.

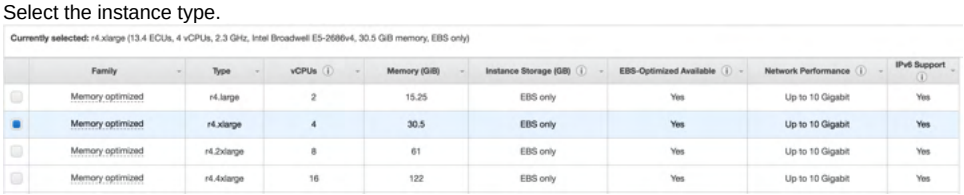
Launch an EC2 Instance for the Controller

This example uses an Amazon Linux AMI (which is provided by AWS: `Amazon Linux AMI 2017.09.1 (HVM), SSD Volume Type - ami-f63b1193`). The Amazon Linux AMI is an EBS-backed, AWS-supported image.

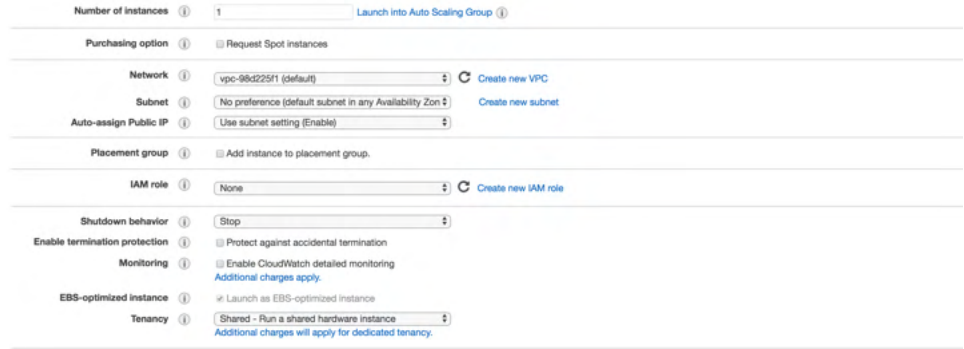
The default image includes AWS command-line tools: Python, Ruby, Perl, and Java.

The repositories include: Docker, PHP, MySQL, PostgreSQL, and other packages.

1. Select the instance type.



2. Use the default instance settings.



3. Increase the storage amount from the default 8 GB to 32 GB.

> ⚠ You will need considerably more space on a larger server.

4. Assign the instance to the `appd-appserver-security-group` that you previously created.

5. To launch the instance, you must specify a key pair. If you are using an existing key pair, ensure that you have access to the private key file; otherwise, generate a new key pair and download it from the AWS console. The private key file is required to connect to the instance using SSH. The new instance should be available after a few minutes. Once the instance is available, you can verify the status using the AWS console. Connect to it through SSH, enter:

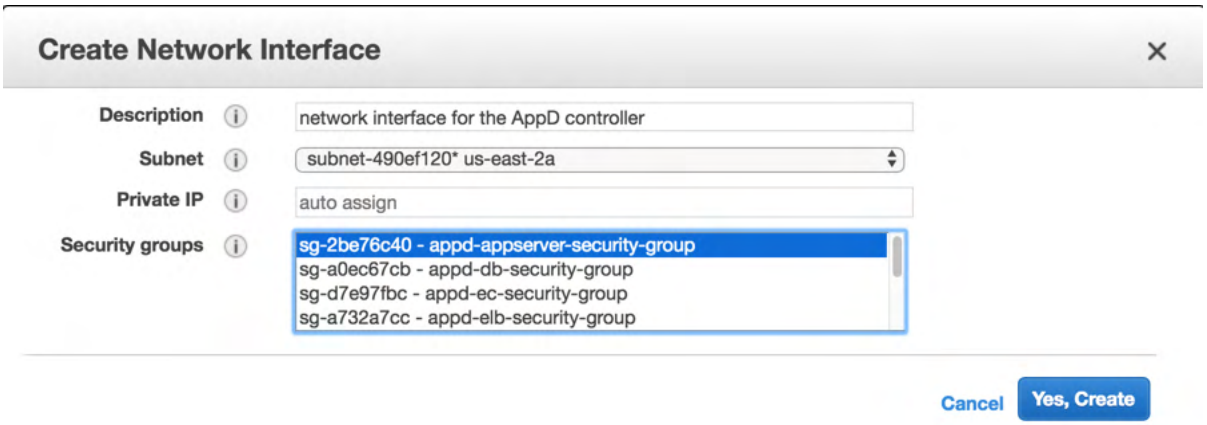
```
ssh -i "<private key file>.pem" ec2-user@ec2-18-222-75-189.us-east-2.compute.amazonaws.com
```

6. Substitute the appropriate path and filename for your private key file.

Create the ENI for the Controller

You need to introduce an ENI which acts as a secondary network interface that you can attach and detach from the Controller EC2 instance.

You then have to associate the AppDynamics license file to the MAC address of this network interface instead of the MAC address of the underlying EC2 instance.



Attach the new network interface to the Controller EC2 instance.

Create DNS CNAMEs

AppDynamics recommends that you create aliases used to connect the Aurora database instance to the Controller EC2 instance.

For example:

```
appdcontroller.mydomain.com  appdcontrollerdb.cxylwiexaqo2.us-east-2.rds.amazonaws.com (DNS name for the Aurora DB instance)

appd-database.mydomain.com  ip-172-31-22-161.us-east-2.compute.internal (private DNS name for the ENI attached to the Controller)
```

You should use these aliases when you install the Controller through the Enterprise Console. Using aliases prevents you from tightly coupling the Enterprise Console with the specific Aurora DB instance or EC2 instance hosting the Controller.

For example, if the database were to fail completely, and you needed to restore the database from a snapshot, then you would have a new DNS name for the Aurora DB instance. Pointing the Enterprise Console at an alias, instead of the DNS name for the Aurora DB instance itself, allows you to only update the DNS alias, and leave the Enterprise Console configuration unchanged.

However, if you need to move the Controller to a different EC2 instance, which resides in another Availability Zone (AZ), you can just update the DNS alias to point to the ENI in the new AZ.

For purposes of testing an AWS configuration, it may not be possible to increase DNS aliases. As a result, you can just add entries to the `/etc/hosts` file on both the Enterprise Console and Controller EC2 instances. For example:

```
172.31.17.84 appdcontroller
172.31.25.80 appd-database
```

Install the Enterprise Console in an AWS Environment

This page describes how to install the the Enterprise Console in an AWS environment using an EC2 instance. You then use the Enterprise Console to install the Controller on a separate EC2 instance (using Aurora DB as the backend).

Launch the EC2 Instance

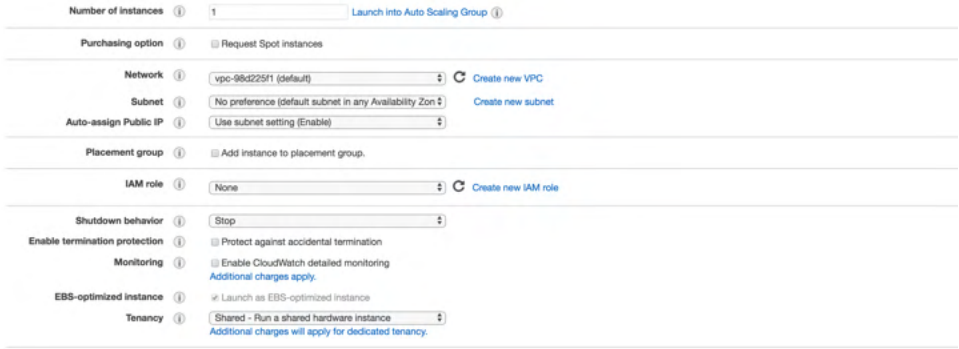
This example uses this AMI (which is provided by AWS): `Amazon Linux AMI 2017.09.1 (HVM), SSD Volume Type - ami-f63b1193`. The Amazon Linux AMI is an EBS-backed, AWS-supported image.

The default image includes AWS command-line tools: Python, Ruby, Perl, and Java.

The repositories include: Docker, PHP, MySQL, PostgreSQL, and other packages.

1. Select the Instance type. The Enterprise Console has only modest requirements, therefore you can select the **t2.medium** instance type.

2. Use the default instance settings.



3. Increase the storage amount from the default 8 GB to 32 GB.

4. Assign the instance to the `appd-ec-security-group` that you created previously.

Prepare the Instance

To launch the instance, you must specify a key pair. If you are using an existing key pair, ensure you have access to the private key file; otherwise, generate a new key pair and download it from the AWS console. The private key file is required to connect to the instance using SSH.

The new instance should be available after a few minutes. Once the instance is available, you can verify the status using the AWS console. Connect to it through SSH, enter:

```
ssh -i "<private key file>.pem" ec2-user@ec2-18-222-75-189.us-east-2.compute.amazonaws.com
```

Then, substitute the appropriate path and filename for your private key file.

Install the Enterprise Console

Use `scp` to transfer the Enterprise Console installer binary to your EC2 instance, enter:

```
scp -i "<private key file>.pem" platform_setup.sh ec2-user@ec2-18-222-75-189.us-east-2.compute.amazonaws.com:
/data
```

Then, SSH to your EC2 instance and run the `installer` to install Enterprise Console:

```
cd /data
chmod 700 platform_setup.sh
./platform_setup.sh -c
```

While installing the Enterprise Console, you are prompted to either select a database port, or accept the default port of 3377.

> Do not use port 3388 because it conflicts with the Controller database port which is used later in the installation process.

You must have write access to the Enterprise Console installation directory you select.

When installing one Controller in the AWS environment, it is easier to install both the Controller and Enterprise Console on the same host.

However if you plan to install multiple Controllers and want to manage them through a single Enterprise Console instance, then you should install the Enterprise Console and the Controller on separate hosts.

Complete the installation of Enterprise Console, and make a note of any passwords you specify during the installation process.

Install the Controller in AWS Using Aurora

This page describes how to install the Controller in an AWS environment using an EC2 instance for the Controller Appserver, and an AWS RDS Aurora instance for the database. This page uses the Enterprise Console that you previously installed.

> Before you install the Controller using Aurora as the database, you must adjust the time zone of the Aurora database to match the time zone of the Controller server. By default, AWS sets the time zone equal to the UTC time zone. See updating the Aurora RDS time zone

To install the Controller in an AWS environment using an EC2 instance for the Controller Appserver, and an AWS RDS Aurora instance for the database:

1. From the Enterprise Console instance, create a new platform:

   ```
   cd ./appdynamics/platform/platform-admin/bin
   ./platform-admin.sh create-platform --name <platform_name> --installation-dir /data/appdynamics/platform
   /product
   ```

2. Add a new host to the platform and install the Controller on the same host as the Enterprise Console:

   ```
   ./platform-admin.sh add-hosts --platform-name testplatform --hosts localhost
   ```

3. Install the Controller using Aurora as the database. Substitute the appropriate values for the `admin user name`, `passwords`, and `Controller host` and `port`. Ensure that `databaseType` is set to `Aurora`, and use the `private DNS name` of the network interface attached to the Controller EC2 instance instead of the `DNS name` for the EC2 instance itself.

   ```
   ./platform-admin.sh submit-job --platform-name testplatform --service controller --job install --args
   controllerProfile=<profile_size> controllerPrimaryHost=<network_interface_private_DNS_name>
   controllerTenancyMode=single controllerRootUserPassword="<password>" mysqlRootPassword="<password>"
   controllerAdminUsername="admin" controllerAdminPassword="<password>" databaseType=Aurora
   controllerDBPort=3388 controllerDBHost="<auroraHost>"
   ```

   The installer connects to the Aurora DB instance, and creates the necessary databases, tables, and other objects. After a few minutes, the Controller should be installed and ready to use.

Apply Controller Optimizations

This page describes how to apply optimizations to the Controller.

Glassfish Configuration

You can configure domain protocols, network listeners, transports, and thread pools from the Enterprise Console UI. You can edit them from the AppServer Configurations page by selecting the platform, and navigating to **Configurations > Controller Settings > Appserver Configurations**.

These files contain sample content for each of the profiles:

- `domain protocols.txt`
- `domain network listeners`
- `domain transports.txt`
- `domain thread pools.txt`

The Enterprise Console restarts the Controller after you submit your configurations.

Configure a Load Balancer

You can configure a load balancer after you have installed the Controller, and it is running in EC2. The load balancer distributes traffic across multiple ports on the Controller.

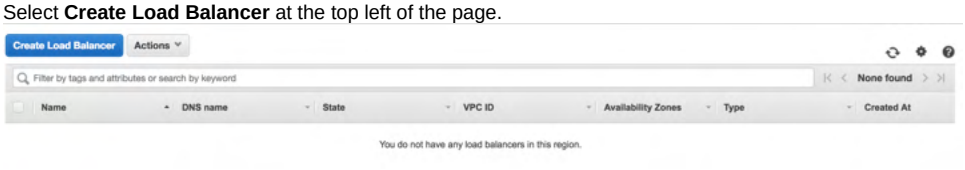
If using HTTPS, an SSL certificate should be available. For testing, you can generate a self-signed certificate using Open SSL (described here). You can then import that certificate using AWS Certificate Manager (ACM).

> ⚠ You can create a load balancer to align with your organization's standards.

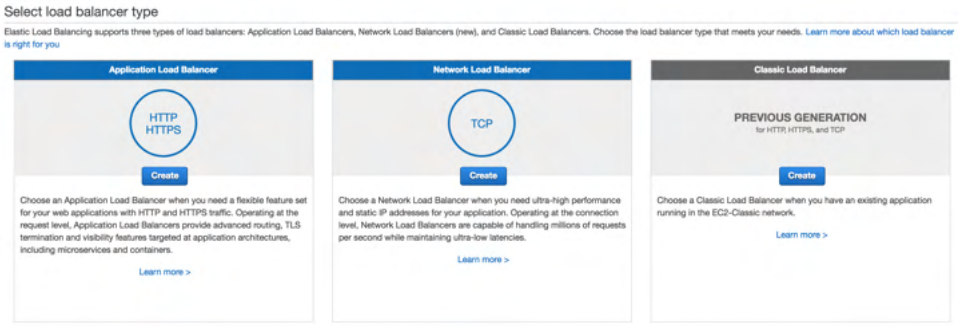
Create a Load Balancer

To create a load balancer:

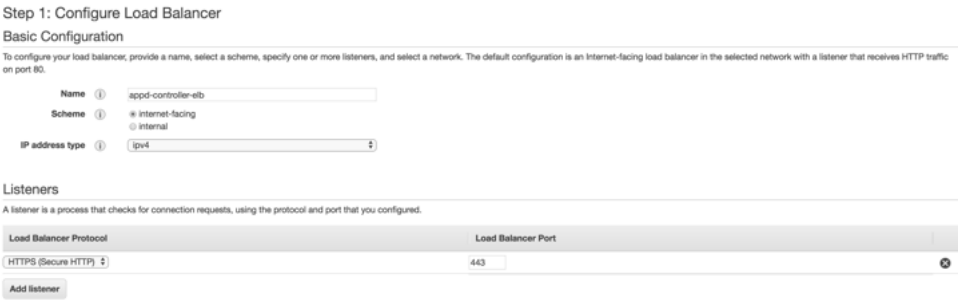
1. Navigate to the Load Balancers page on the EC2 Dashboard in the AWS console.
2. Select **Create Load Balancer** at the top left of the page.



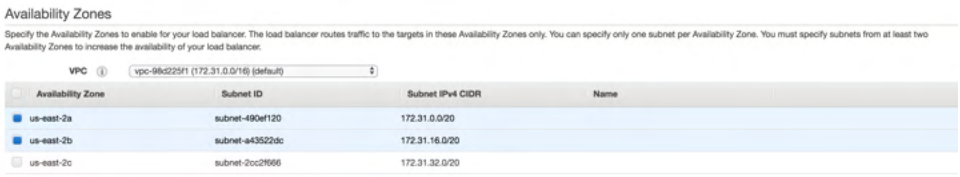
3. Select **Application Load Balancer** as the load balancer type to terminate SSL at the ELB.



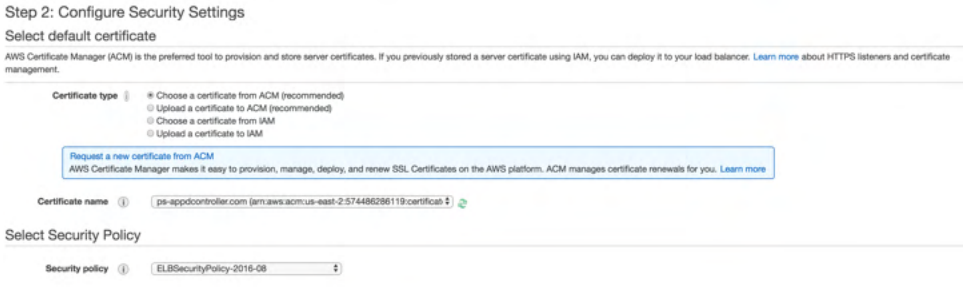
4. Enter a name for the new load balancer, and select **HTTPS (Secure HTTP)** as the load balancer protocol to accept HTTPS traffic only.



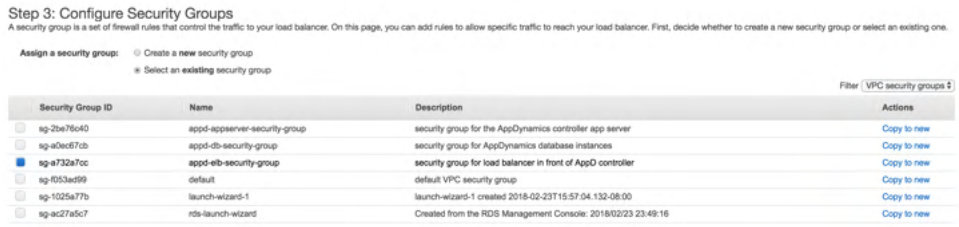
5. Select the availability zones to enable for the new load balancer. It should include the availability zone in which the Controller Appserver EC2 instance resides.



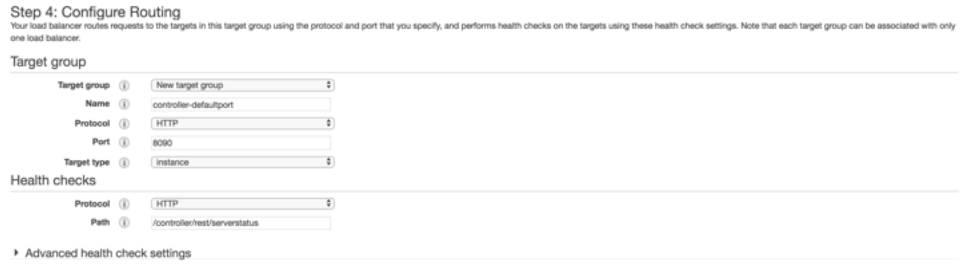
6. Select the certificate that was imported to the ACM.



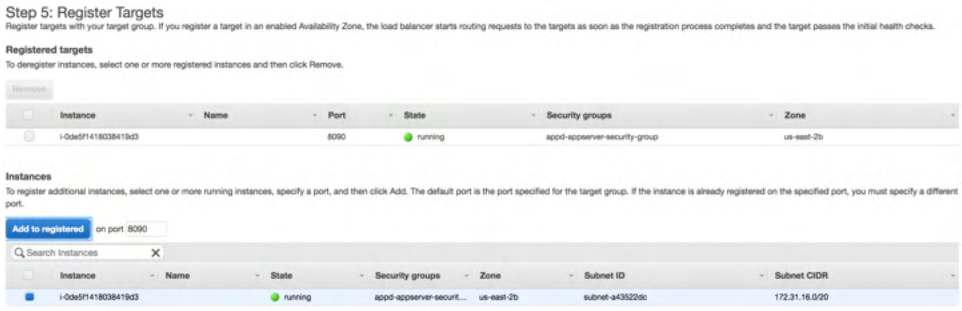
7. Specify the security group.



8. For the initial configuration, set the load balancer to route all traffic to port 8090 using HTTP, and define the standard health check for the Controller.



9. Add the Controller EC2 instance as a registered target.



10. Launch the new load balancer. It may take a few minutes before the load balance occurs.
11. Verify that you can access the Controller UI through the load balancer.

> If your SSL cert is self-signed, a browser warning displays. You can ignore this warning if you are testing the UI. However for Agent traffic, you need a valid certificate that is trusted by the Agent.

Specify the External Load Balancer URL

Use the Enterprise Console UI to update the Controller configuration and specify the external load balancer URL.

1. Open a browser and navigate to the UI:

```
http://<hostname>:<port>
```

   9191 is the default port.
2. Navigate to AppServer Configurations by selecting the platform: **Configurations > Controller Settings > Appserver Configurations**.
3. Enter the external load balancer URL in the appropriate field and select **Save**.

⚠ After applying the optimizations, you should create an AMI.

Configure Listener Rules

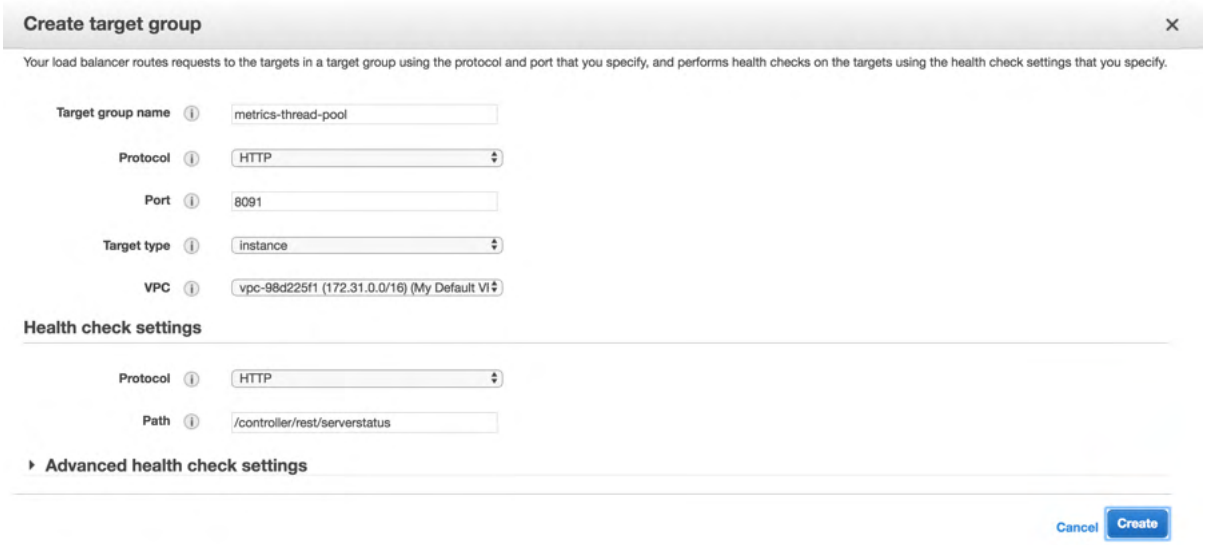
Depending on the type of request, you can define additional rules to route traffic to various ports on the Controller. This table defines rules that users typically include for their load balancer:

Pool Name	URL Pattern	Port Number	Description
`metrics-thread-pool`	`/controller/instance/*/metrics` `/controller/instance/*/metrics*`	8091	Agent metric data upload
`config-thread-pool`	`/controller/instance/*/applicationConfiguration*`	8092	Agent configuration requests
`agent-thread-pool`	`/controller/instance/*`	8093	Other Agent requests
`status-thread-pool`	`/controller/rest/serverstatus`	8094	Server status ping by load balancer
`http-thread-pool`	Default / User Traffic	8090	Default thread pool for all other traffic
`restui-default-thread-pool`	`/controller/restui/*`	8095	Default thread pool for all `restui` traffic
`restui-analytics-thread-pool`	`/controller/restui/analytics/*`	8096	Thread pool traffic for analytics traffic

Create Target Groups

You must create a target group for each of the rules listed in the table, with the exception of the default `http-thread-pool` rule (for which you can use the default target group).

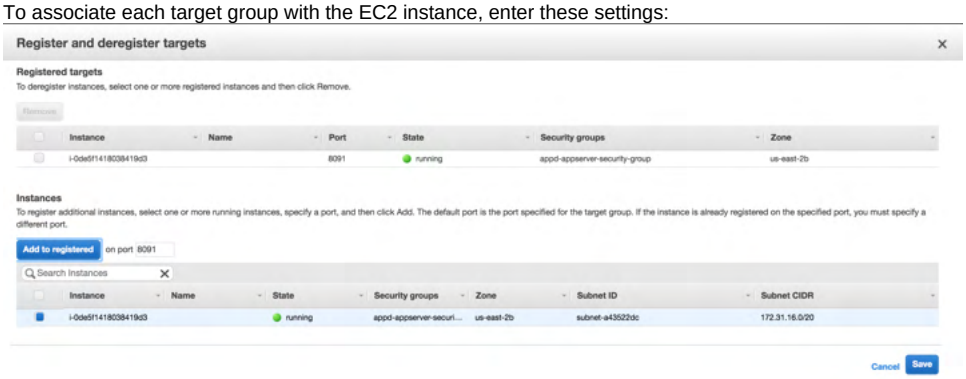
For example, you create a target group for the `metrics-thread-pool` with these settings:



> You can use the health check path for all of the target groups, however you may want to decrease the frequency because performing the same check on all ports every 30 seconds is not required.

Register Targets

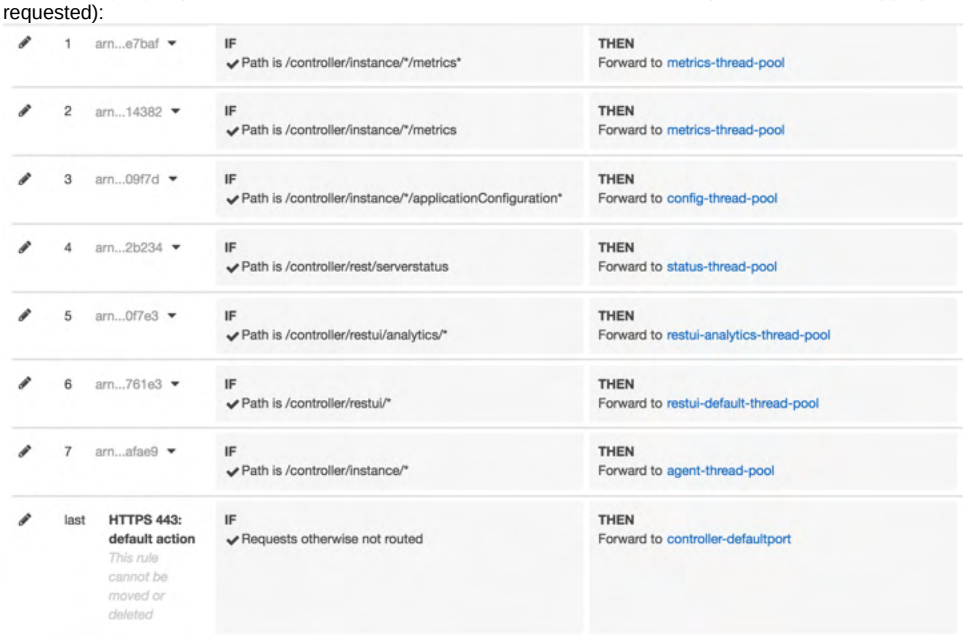
1. To associate each target group with the EC2 instance, enter these settings:



The full list of target groups should display:

Name	Port	Protocol	Target type	VPC ID
agent-thread-pool	8093	HTTP	instance	vpc-98d225f1
config-thread-pool	8092	HTTP	instance	vpc-98d225f1
controller-defaultport	8090	HTTP	instance	vpc-98d225f1
metrics-thread-pool	8091	HTTP	instance	vpc-98d225f1
restui-analytics-thread-pool	8096	HTTP	instance	vpc-98d225f1
restui-default-thread-pool	8095	HTTP	instance	vpc-98d225f1
status-thread-pool	8094	HTTP	instance	vpc-98d225f1

2. After the target groups have been defined, you can add new listener rules to map the traffic to the appropriate target group (based on the path requested):



The order of the rules is important because some paths may match multiple rules.

Create and Manage the AMI

You can create an Amazon Machine Image (AMI) to recover from an unexpected event or create a new Controller from the same image.

The AMI image should be created for the Controller, including required optimizations, and an auto-scaling group should be created based on the AMI. You can configure the Elastic Load Balancer (ELB) to send traffic to instances in the auto-scaling group, which will consist of one EC2 instance at a time.

Create an AMI and Auto Scaling Group

To ensure that you can recover from any interruptions, you need to create an AMI that includes the Controller. Then, you need to create an auto scaling group based on the AMI.

Creating an AMI and auto scaling group involves the following steps:

Step 1: Shut Down the Enterprise Console

You must stop the Enterprise Console before creating an AMI:

```
bin/platform-admin.sh stop-platform-admin
```

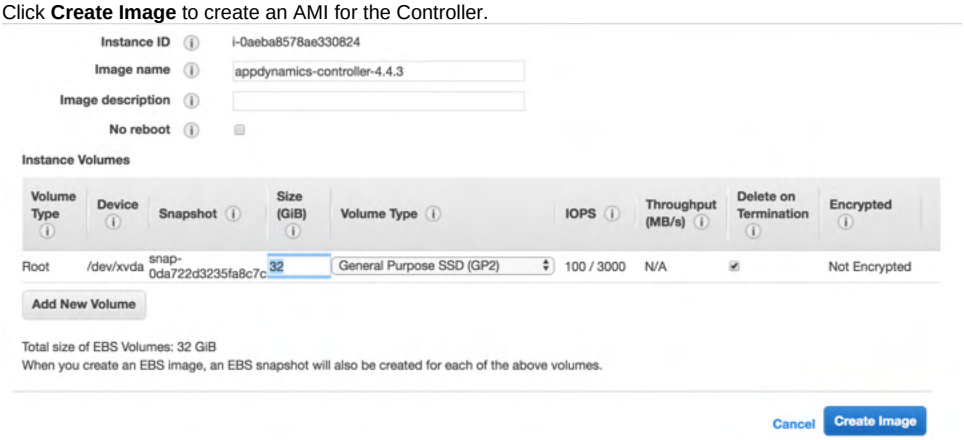
This ensures that all jobs and services are stopped in order to create a clean image.

Step 2: Create an AMI

You can create an AMI to use to deploy a new Controller without having to go through the installation steps again.

> A new AMI should be created whenever the Controller version is upgraded, or configuration changes are made. This ensures that the changes are propagated to the new EC2 instance, in the event that you need to move to a different availability zone, for example. At the same time, previous AMI instances should be retained as well, in case a rollback is required.

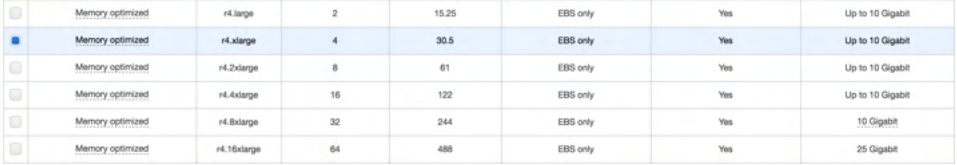
1. Stop the Controller Appserver once optimizations have been applied to it.
2. Click **Create Image** to create an AMI for the Controller.



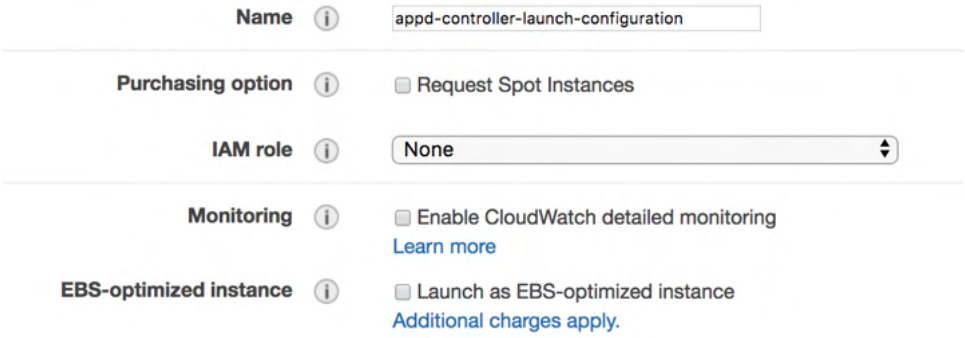
Step 3: Create the Launch Configuration

Once the AMI is created, you can create an auto scaling group that uses it. But first, you need to create the launch configuration:

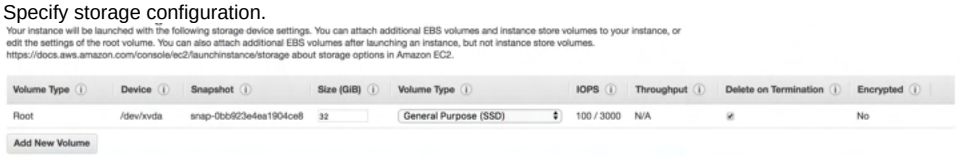
1. Select the appropriate instance type for the app server of your AMI.



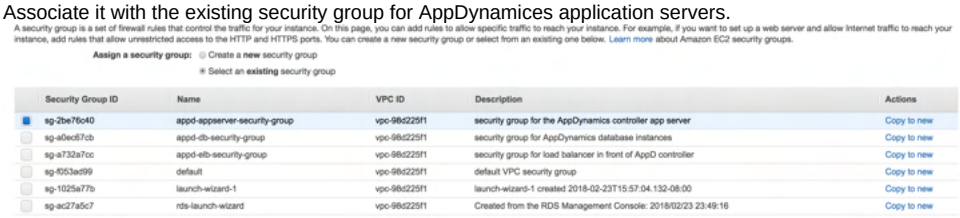
2. Specify a name for the launch configuration.



3. Specify storage configuration.



4. Associate it with the existing security group for AppDynamices application servers.



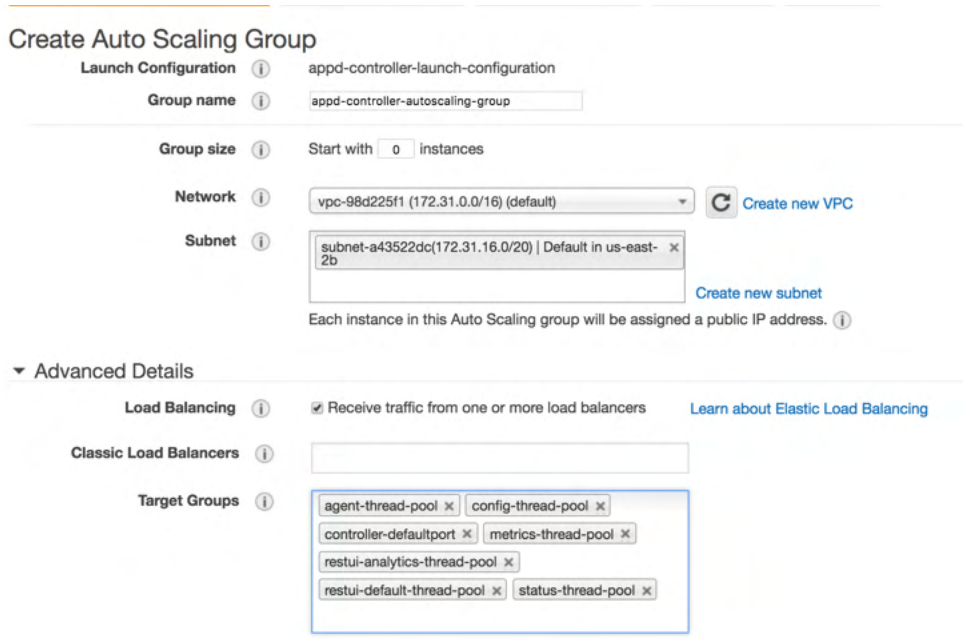
Step 4: Create an Auto Scaling Group

You should create an auto scaling group based on this AMI to ensure that one node is running at any given time.

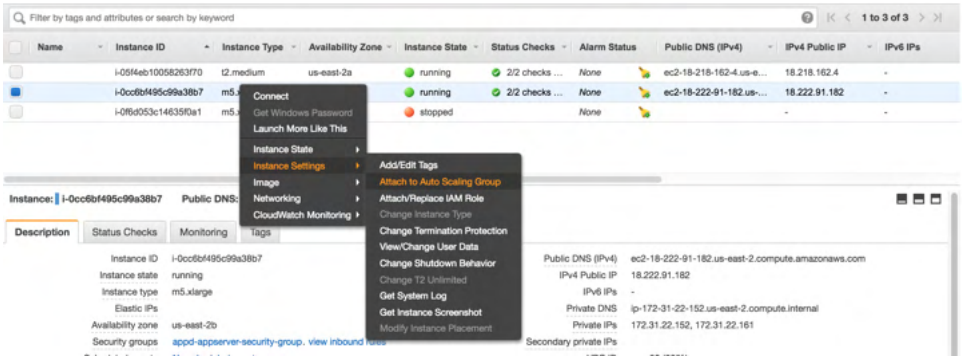
To create an auto scaling group:

1. Specify a name for the auto scaling group, enable traffic from the load balancer, and map the target groups created earlier.

> ⚠ Start with zero instances, as you will need to add your existing instance to the group later.
> Keep the group at its initial size of zero, as you do not want to add more than one Controller at any point in time.



2. Accept the default values for the rest of the settings, and create the auto scaling group.
3. Once the auto scaling group is successfully created, edit the properties to specify the max allowable instances as 1 (up from 0).
4. Go back to the list of EC2 instances, and add the EC2 instance for the Controller Appserver to the new auto scaling group.



Updating the AMI

The AMI should be updated whenever changes are made to the controller EC2 instance. For example:

- When a new license file is dropped on the Controller.
- When O/S configuration changes are made.
- When Controller configuration changes are made, either via the Enterprise Console or directly to domain.xml or other configuration files.

Migrate the Controller Database to Amazon Aurora

You have the option to migrate an existing 4.4.3 or latest on-premises Controller database to Aurora DB. The Controller might already reside in AWS, or in your data center.

Although the Controller already contains a MySQL database, you are recommended to migrate the MySQL database to Aurora because it offers replication, high availability, and elasticity. The Amazon Relational Database Service (RDS) tool handles provisioning, patching, backup, recovery, failure detection, and repair of the database. Also, Aurora DB offers encryption at rest, with encryption of all automated backups, snapshots, and replicas in the same cluster.

If your Controller is not already in AWS, then follow Migrate the Controller to migrate it. Once this is complete, you should have a Controller running on one or two EC2 instances in AWS, depending on whether or not your existing Controller deployment is high availability (HA), with the MySQL database hosted on those instances.

> ⚠ Commands to start and stop the database do not work with Aurora DB.

Migrate MySQL to an Aurora Database for 4.4.3 or Latest

You can create and configure Amazon Aurora to serve as the Controller database for 4.4.3 or the latest version. This process, which uses mysqldump to migrate the database, is required for Controllers running MySQL version 5.7. See Back Up the Controller with mysqldump for more information.

> ⓘ Since Controller upgrades to 4.4.3 from 4.3.x or earlier would use MySQL 5.5, it is important that you know what your Controller MySQL version is. Please refer to Bundled MySQL Database Version to learn how to check your MySQL version and upgrade it if necessary.

Note that running a Controller on AWS requires that some of the `cluster` parameter group and `db` parameter group settings be adjusted. See Deploy the Controller on AWS for more information.

Migrating MySQL to an Aurora Database involves the following steps:

> ⓘ If you attempt to upgrade or move a Controller migrated without its `liquibase-stored` procedures, the upgrade will fail. You must recreate these stored procedures manually in AWS.

1. Step 1: Provision an Empty Aurora Database
2. Step 2: Use mysqldump to Export from MySQL
3. Step 3: Use mysqldump to export stored procedures from the AppDynamics database
4. Step 4: Use mysql to Import to Aurora
5. Step 5: Configure the Controller to Use the Aurora Database

Step 1: Provision an Empty Aurora Database

You first need to start up a new instance of Aurora, using the desired instance type and other custom settings as explained in Deploy the Controller on AWS. Ensure that the database instance is created using port 3388.

Step 2: Use mysqldump to Export from MySQL

> ⚠ Before using `mysqldump`, first, ensure that the Controller app server is stopped. If you attempt to run `mysqldump` while the app server is running, it will severely degrade the performance and stability of the Controller.

To use `mysqldump`, run the `mysqldump` executable, passing the root username, password, and output file:

1. Run the following command to navigate to the executable directory:

   ```
   cd <controller_home>/db/bin
   ```

2. Use the following command to export the database from MySQL:

   ```
   ./mysqldump -u root --databases controller licensing mds_auth mds_configuration mds_entitysearch
   mds_infra_core mds_infra_server mds_license mds_metadata mds_metering mds_rbac mds_topology --single-
   transaction --compress --order-by-primary -p "<password>" > backup.sql
   ```

3. In order to import the resulting file into Aurora, you need to replace the following line:

```
/*!50013 DEFINER=`controller`@`localhost` SQL SECURITY DEFINER */
```

With:

```
/*!50013 DEFINER=`controller`@`%` SQL SECURITY DEFINER */
```

Step 3: Use mysqldump to export stored procedures from the AppDynamics database

Run the following command to export the stored procedures from the AppDynamics database.

```
./mysqldump -u root -p --protocol=TCP -h 127.0.0.1 -P <controller_mysql_port> --no-create-db --skip-add-drop-table --no-create-info --skip-disable-keys mysql proc --result-file=/staging/path/for/mysql.proc.sql
```

This command, through the `--result-file` option, dumps the stored procedures to `/staging/path/for/mysql.proc.sql`.

Step 4: Use mysql to Import to Aurora

1. Run the following command to navigate to the executable directory:

   ```
   cd <controller_home>/db/bin
   ```

2. Connect to the new Aurora instance:

   ```
   ./mysql -u root -p"<password>" -h <hostname>.<aws-region>.rds.amazonaws.com -P 3388 --protocol=TCP
   ```

3. Then create the Controller user, and grant it permissions:

   ```
   CREATE USER 'controller'@'%' IDENTIFIED BY 'controller';
   GRANT USAGE ON *.* TO 'controller'@'%';
   GRANT ALL PRIVILEGES ON `controller`.* TO 'controller'@'%';
   GRANT ALL PRIVILEGES ON `licensing`.* TO 'controller'@'%';
   GRANT ALL PRIVILEGES ON `mds_auth`.* TO 'controller'@'%';
   GRANT ALL PRIVILEGES ON `mds_configuration`.* TO 'controller'@'%';
   GRANT ALL PRIVILEGES ON `mds_entitysearch`.* TO 'controller'@'%';
   GRANT ALL PRIVILEGES ON `mds_infra_core`.* TO 'controller'@'%';
   GRANT ALL PRIVILEGES ON `mds_infra_server`.* TO 'controller'@'%';
   GRANT ALL PRIVILEGES ON `mds_license`.* TO 'controller'@'%';
   GRANT ALL PRIVILEGES ON `mds_metadata`.* TO 'controller'@'%';
   GRANT ALL PRIVILEGES ON `mds_metering`.* TO 'controller'@'%';
   GRANT ALL PRIVILEGES ON `mds_rbac`.* TO 'controller'@'%';
   GRANT ALL PRIVILEGES ON `mds_topology`.* TO 'controller'@'%';
   FLUSH PRIVILEGES;
   ```

   Note

   The Aurora database is protected by security groups to prevent access from unauthorized sources.

4. Import the database backup:

   ```
   ./mysql -u controller -P 3388 -H <hostname>.<aws-region>.rds.amazonaws.com -p "controller" --protocol=TCP < backup.sql
   ```

5. Import the stored procedures:

   The import must be made by an admin or root user.

```
./mysql -u admin -P 3388 -H <hostname>.<aws-region>.rds.amazonaws.com -p "controller" --protocol=TCP <
/staging/path/for/mysql.proc.sql
```

Step 5: Configure the Controller to Use the Aurora Database

1. Change the configuration in the Controller to use the Aurora DB:
    a. In the file `<controller_home>/appserver/mq/lib/props/broker/default.properties`, set the property `imq.persist.jdbc.mysql.property.url` to your Aurora database:

    ```
    imq.persist.jdbc.mysql.property.url=jdbc\:mysql\://<aurora-db>.<aws-region>.rds.amazonaws.com\:3388
    /controller
    ```

    b. In the file `<controller_home>/appserver/glassfish/domains/domain1/config/domain.xml`, set the property `serverName` to the value of your Aurora database:

    ```
    <property name="serverName" value="<aurora-db>.<aws-region>.rds.amazonaws.com"></property>
    <property name="portNumber" value="3388"></property>
    <property name="url" value="jdbc\:mysql\://<aurora-db>.<aws-region>.rds.amazonaws.com\:3388
    /controller?nullNamePatternMatchesAll=true&amp;allowLoadLocalInfile=true&amp;
    cachePrepStmts=true&amp;prepStmtCacheSize=25&amp;dumpQueriesOnException=true&amp;
    rewriteBatchedStatements=true&amp;useSSL=false&amp;maxAllowedPacket=104857600"/>
    ```

    Add the following line to the same file, right before the `javaagent` option:

    ```
    <jvm-options>-Dappdynamics.controller.use.global.datadir.query.for.disk.space.check=false</jvm-
    options>
    ```

    c. In the file `<controller_home>/bin/controller_maintenance.xml`, set the property `db-host` to the value of your Aurora database:

    ```
    <property name="db-host" value="<aurora-db>.<aws-region>.rds.amazonaws.com"/>
    <property name="db-port" value="3388"/>
    ```

    d. In the file `<controller_home>/bin/setup.xml`, set the property `db-host` to the value of your Aurora database:

    ```
    <property name="db-host"value="<aurora-db>.<aws-region>.rds.amazonaws.com"/>
    <property name="db-port" value="3388"/>
    ```

2. Remove the following cache folders from `<controller_home>/appserver/glassfish/domains/domain1` as follows:
    a. `rm -rf osgi-cache/`
    b. `rm -rf generated/`
3. With the Controller service installed, start the Controller with root:

    ```
    service controller start
    ```

4. Verify that the Controller is running successfully. The local MySQL database should be shut down, and you should see the migrated data in Aurora, which can be verified via the Controller UI.

Reset the Controller Database Root User Password

To reset the Controller Database root user password:

1. Log in to the RDS instance as admin:
    a. `./mysql -u admin -h <rds-aurora-endpoint> -P 3388 -p`
2. Use MySQL to run the commands:

a. To specify the Controller database, enter: `use mysql;`
b. To reload the MySQL grant tables, enter: `FLUSH PRIVILEGES;`
c. To determine your MySQL version, enter: `select version();`
d. If you are using MySQL version 5.5, to configure the new password for the root user, enter: `update mysql.user set password=password('<new-password-here>') where user like 'root%';`
e. If you are using MySQL version 5.7. to configure the new password for the root user, enter: `update mysql.user set authentication_string=password('<new-password-here>') where user like 'root%';`
f. To reload the MySQL grant tables, enter: `FLUSH PRIVILEGES;`
g. To exit MySQL, enter: `quit`

Change the Controller Database Root User Password

By default, the password for the controller user of the Aurora database used in your AppDynamics deployment is `controller`.

To change the Controller Database root user password:

1. Log in to the RDS instance as root:
   a. `./mysql -u root -h <rds-aurora-endpoint> -P 3388 -p`
2. Use MySQL to run the commands:
   a. To specify the Controller Database, enter: `use mysql;`
   b. To reload the MySQL grant tables, enter: `FLUSH PRIVILEGES;`
   c. To determine your MySQL version, enter: `select version();`
   d. If you are using MySQL version 5.5, to configure the new password for the root user, enter: `update mysql.user set password=password('<new-password-here>') where user like 'controller%';`
   e. If you are using MySQL version 5.7, to configure the new password for the root user, enter: `update mysql.user set authentication_string=password('<new-password-here>') where user like 'controller%';`
   f. To reload the MySQL grant tables, enter: `FLUSH PRIVILEGES;`
   g. To exit MySQL, enter: `quit`
3. Update the `controller-db-password` alias with the new Controller DB password in Glassfish:

```
cd <controller_home>/appserver/glassfish/bin
./asadmin update-password-alias controller-db-password
```

4. Restart the Controller AppServer for the changes to take effect:

```
cd <controller_home>/bin
./controller.sh stop-appserver
./controller.sh start-appserver
```

5. Verify the change in asadmin:

```
cd <controller_home>/appserver/glassfish/bin
./asadmin ping-connection-pool controller_mysql_pool
```

Upgrade or Move the Controller on AWS

If your Controller instance is deployed on AWS and uses Aurora for its database, you can use the Enterprise Console CLI commands to either *discover and update* or to just *upgrade* your Controller to the latest version. You *cannot* use the Enterprise Console UI, however, to perform the upgrade.

After backing up the Aurora database, use the upgrade method below that best meets your needs:

- Discover and Upgrade - Use this method if you are not sure if you need to upgrade.
- Upgrade - Use this method if you know you are using an older version and want to upgrade. For example, if you want to upgrade from 4.4.3 to the latest.

You can also move the Controller to a new EC2 instance to meet updated performance requirements. For platform-agnostic upgrade instructions, see Upgrade the Controller Using the Enterprise Console.

Back Up the Aurora Database

Before upgrading the Controller, you should back up the Aurora database instance. You should also ensure that you have an Amazon Machine Image (AMI) that accurately reflects the current Controller instance.

> The Enterprise Console upgrades the schema to the latest version. However, upgrading the Controller does not upgrade the Aurora DB server.

Discover and Upgrade the Controller on AWS

The Enterprise Console provides the `discover_upgrade` command that will discover your Controller, determine its version, and then upgrade it if needed.

1. Log in to the Enterprise Console:

   ```
   bin/platform-admin.sh login --user-name <admin_username> --password <admin_password>
   ```

2. Run the following command:

   ```
   bin/platform-admin.sh submit-job --service controller --job discover-upgrade --args
   controllerPrimaryHost="<hostname>" controllerRootUserPassword="<password>" mysqlRootPassword="
   <password>"  databaseType=aurora destinationDirectory="<controllerInstallationDirector>"
   ```

   If your upgrade fails, you can resume by passing the flag `useCheckpoint=true` as an argument after `--args.`
3. Update the AMI after the job finishes.

Upgrade the Controller on AWS

The Enterprise Console provides the `upgrade` command to upgrade the Controller to the latest version. For example, if you are using Controller 4.4.3 and want to upgrade to the latest, you can simply use the `upgrade` command.

1. Log in to the Enterprise Console:

   ```
   bin/platform-admin.sh login --user-name <admin_username> --password <admin_password>
   ```

2. Run the following command on the Enterprise Console host:

   ```
   bin/platform-admin.sh submit-job --service controller --job upgrade --args controllerRootUserPassword="
   <password>" mysqlRootPassword="<password>"
   ```

   If your upgrade fails, you can resume by passing the flag `useCheckpoint=true` as an argument after `--args.`
3. Update the AMI after the job finishes.

Rollback to a Previous AMI

If a rollback is required, complete the following steps:

1. Create a new Aurora instance, using the database snapshot you took earlier as the source.
2. Stop the upgraded Controller if it is still running:

```
bin/platform-admin.sh stop-controller-appserver
```

3. Repoint the database DNS alias to the new Aurora instance.
4. Terminate the EC2 instance hosting the current Controller. This should cause a new EC2 instance to be provisioned using the existing AMI, with the older Controller version.
5. Attach the ENI to the new EC2 instance.
6. Verify that the Controller is working as it was before the upgrade.

Move the Controller on AWS

You can move the Controller to a new EC2 instance by completing the following steps:

1. Terminate the EC2 instance hosting the current Controller. This should cause a new EC2 instance to be automatically provisioned using the AMI.

   > The auto-scaling group and launch configuration are defined with the AMI. Therefore, if the existing EC2 instance in the auto-scaling group dies, it is automatically replaced with a new EC2 instance based on the same AMI.

2. Attach the ENI to the new EC2 instance.
3. Verify that the Controller is working as expected.

Platform Installation Quick Start

This page provides a high-level view of using the Enterprise Console to install the AppDynamics platform, including the Controller and Events Service.

See Discovery and Upgrade Quick Start for the upgrade quick start guide.

About these Steps

The process described in this page uses the Enterprise Console to perform the following tasks:

- Install a Demo Controller and embedded Events Service.
- Install an Events Service that runs on hosts that are separate from the Controller.
- Switch the Controller to a high availability pair.

This quick start is intended to introduce you to the Enterprise Console and AppDynamics platform. Before you start, review the requirements pages for the platform components.

Install the Enterprise Console

1. Install the Enterprise Console with the following command:

   Linux

   ```
   ./platform-setup-64bit-linux.sh
   ```

   Windows

   ```
   platform-setup-64bit-windows.exe
   ```

   When the installation wizard launches, complete it to install the Enterprise Console. For more information about how to install the Enterprise Console, see Install the Enterprise Console.
2. Verify that the Enterprise Console successfully installed by opening the GUI using the host and port you specified during installation:

   ```
   http(s)://<hostname>:<port>
   ```

   9191 is the default port.
3. Complete the installation process with the GUI or command line.

GUI Installation Quick Start

After you install the Enterprise Console, you can complete the platform installation process with the GUI. The following steps describe how to install a Controller with an embedded Events Service. It then describes how to scale up the Events Service to run on a separate host. To install a Controller with a scaled-up (unembedded) Events Service, see Custom Install.

You can install the Events Service on a separate host directly by selecting a Custom Installation at step 2.

1. Open a browser and navigate to the GUI:

   ```
   http(s)://<hostname>:<port>
   ```

   9191 is the default port.
2. Select **Express Install** to install a Controller and Events Service on a shared host.
3. In the **Add a Host** section, choose **Use Enterprise Console Host**. This option installs a Controller with an embedded Events Service on the same host as the Enterprise Console.
4. Choose **Demo** as the Controller profile if you are just getting to know the system.
5. Fill out the Controller configuration fields as indicated. See Express Install for details.
   You now have a running, testable system and can deploy agents. Next, we'll walk you through the steps for scaling up the Events Service. A scaled-up deployment is strongly recommended for installations meant for something other than for demonstration or exploratory purposes.

Scale Up the Events Service

In this step, you expand your platform to use a scaled-up Events Service deployment. It is important to note that data is not migrated from an embedded instance to a scaled-up instance.

1. In the Enterprise Console UI home page, click on the Platform you created previously.

2. Click **Hosts** in the left navigation menu.
3. Add a least three hosts for a scaled-up Events Service, providing a host name and an SSH credential that enables the Enterprise Console to access the host.
4. When finished, go to the **Events Service** page.
5. From the more menu (...), choose **Scale Up Events Service**.
6. In the **Scale Up Events Service** dialog, choose the **Prod** profile and select the hosts you created previously from the **Host** drop-down menu.
7. Click **Submit**.
8. Now set up a load balancer for the Events Service hosts, as described in Load Balance Events Service Traffic.
9. When done configuring the load balancer, you need to indicate to the Controller the addressable URL for the Events Service. In the Controller Administration Console, set these Controller properties:
    - Set `appdynamics.non.eum.events.use.on.premise.events.service` to true.
    - Set `appdynamics.analytics.server.store.url` to the URL of the Events Service as exposed at the load balancer. For example, `appdynamics.analytics.server.store.url=http://es-master-host:9080/`.

Set Up Controller High Availability

Now that you have a running Controller and Events Service, try setting up high availability for the Controller.

Before proceeding, see Set Up a High Availability Deployment. Specifically, follow the steps indicated in the section on configuring Controller High Availability pair environment in for prerequisites.

Once you have fulfilled the prerequisites, follow these steps:

1. In the Enterprise Console UI home page, click on the Platform you created previously.
2. Click **Hosts** in the left navigation menu.
3. Add a single new host. Provide a hostname and an SSH credential that enables the Enterprise Console to access the host.
4. When finished, go to the **Controller** page.
5. Click **Add Secondary Controller**.
6. Provide the same passwords as the primary database and Controller root password, and click **Submit**.

Now you have two Controllers running in passive-active high availability mode. To complete the configuration, you need to set up a load balancer, so that you can easily switch traffic between the Controllers. See Use a Reverse Proxy. After configuring a load balancer, you would need to specify the URL address for reaching the Controller pair in the Enterprise Console UI by clicking **Configurations** > **Controller Settings** > **Appserver Configurations**, and specifying the URL in the **External URL** field.

You now have a functioning platform, consisting of Controller pair and a scaled up Events Service instance.

Install the EUM Server

If using EUM, you can continue by installing an EUM Server. The EUM Server is installed separately, not with the Enterprise Console.

Follow the steps below to get started:

1. Read an overview of the EUM Server in EUM Server Deployment.
2. Learn to install and configure the EUM Server by following the instructions in Install a Demo EUM Server.
3. Install the EUM Server for production.

Discovery and Upgrade Quick Start

Related pages:

- Upgrade Platform Components
- Discover Existing Components

This quick start guide describes how to use the Enterprise Console to discover existing AppDynamics components, such as a Controller, and use it to upgrade them.

Install the Enterprise Console

You can install the Enterprise Console on the same host as the Controller, or provide a separate host for installation.

1. Install the Enterprise Console with the following command:

   Linux

   ```
   ./platform-setup-64bit-linux.sh
   ```

   Windows

   ```
   platform-setup-64bit-windows.exe
   ```

   The installation wizard launches. Complete the wizard to install the Enterprise Console. For more information about how to install the application, see Enterprise Console.
2. Verify that the Enterprise Console successfully installed by opening the GUI using the host and port you specified during installation:

   ```
   http(s)://<hostname>:<port>
   ```

   9191 is the default port.

Discover and Upgrade with the Enterprise Console GUI

The options presented are based on the state of your platform. If you have not created a platform, choose Discovery from the options presented. Complete the wizard to create a platform, add hosts, and discover the Controller and Events Service.

If you created a platform already, such as with the CLI, complete the following steps:

1. Identify the host(s) where the Controller and Events Service are installed.
2. On the **Credentials** page, add credentials to the platform if you are using remote hosts.

   > Remember to provide the private key file for the Enterprise Console machine when adding a credential.

3. Navigate to the Hosts page and add hosts.
   a. If the Controller and Events Service are installed on the same host as the Enterprise Console, select **Add Enterprise Console Host**.
   b. If the Controller and Events Service are installed on different hosts than the Enterprise Console, input the host(s). Select a credential for the host(s) and add credentials to the platform.

Discover and Upgrade with the CLI

1. Navigate to the directory where you installed the Enterprise Console.
2. Create a platform with the following command:

   ```
   bin/platform-admin.sh create-platform --name <platform_name> --installation-dir
   <platform_installation_directory>
   ```

3. Add credentials that the platform can use to access hosts with the following command:

   ```
   bin/platform-admin.sh add-credential --credential-name <name> --type ssh --user-name <username> --ssh-
   key-file <file_path_to_the_key_file>
   ```

`<file path to the key file>` is the private key for the Enterprise Console machine. The installation process deploys the keys to the hosts.

4. Add hosts to the platform:

```
bin/platform-admin.sh add-hosts --hosts localhost host_2 host_3 --credential <credential_name>
```

At a minimum, make sure to bootstrap all of the hosts where the AppDynamics server-side components, namely the Controller and Events Service, are installed.

You may also use the loopback address '127.0.0.1' or the machine's actual hostname in place of 'localhost'.

Enterprise Console

Related pages:

- Install the Enterprise Console
- Administer the Enterprise Console

The Enterprise Console is the installer for the Controller and Events Service. You can use it to install and manage the entire lifecycle of new or existing on-premises AppDynamics Platforms and components. The application provides a GUI and command-line interface.

There is no customer-facing application for SaaS Controllers since they are managed by the AppDynamics Operations team.

If your Enterprise Console host goes down, it does not impact Controllers, Events Service, or High Availability (HA) pairs. Those services will continue to run independently of the application. You can then discover all platforms on a new Enterprise Console host without any impact on the components.

> ⚠ If the Enterprise Console is not available, the auto-failover option for HA pairs will also not be available when there is an issue with the primary Controller. Therefore, it is important to keep the Enterprise Console host in a healthy state.

Enterprise Console Details

The Enterprise Console encompasses management features, installation modes, and lifecycle monitoring.

Enterprise Console Features

The Enterprise Console allows you to perform the following tasks:

Multi-Platform Management

- Manage multiple platforms at the same time using the application

  > ⓘ The Enterprise Console does not require all services within a given platform to have the same major version number.

- Discover, install, and upgrade Controllers, Event Services, and MySQL nodes

  > ⚠ Note that all services on Windows machines must be installed on the Enterprise Console host when using the Enterprise Console since the application does not support remote operations on Windows.

HA-Lifecycle Management (Available on Linux only)

- Manage HA pair lifecycle without the use of the CLI based HA-toolkit or sudo privileges
- Perform failover management

Other Features

- Manage Controller and Events Service lifecycle
- Utilize GUI & CLI support
- Manage AppServer and database configurations

Platform Install Modes

There are two install types that you can use to deploy your platform:

Express Install

- The quickest way to get started for fresh Controller installations
- Install the Controller and an embedded Events Service on a single host

Custom Install

- Configure and customize user inputs and installation/data directories for the Controller and Events Service
- Install or upgrade single or HA Controllers and scaled-up Events Service in a distributed setup

> ⚠ Controller HA pairs and Events Service clusters are not available on Windows machines through the Enterprise Console since the application does not support remote operations on Windows.
>
> You can manually install or upgrade multi-node Events Service clusters. See Install the Events Service on Windows or Upgrade the Events Service Manually.

- Add one or more Events Service nodes
- Discover & Upgrade older platform services

Lifecycle Monitoring

On the Platforms page, you can see all of your platforms, their statuses, and the statuses of their services. Once you have selected a platform to view, the screen is separated into different tabs:

Hosts

Hosts are the actual hardware devices that are connected to the platform. You can add, remove, or change the credentials of your hosts in this tab.

> ⚠ You cannot add a host in a Windows Enterprise Console machine.

Controller

The Controllers page shows the primary and secondary roles of the Controllers and their MySQL nodes. The entire lifecycle operations of Controllers and MySQL nodes can be performed here. You can also see the External URL, which is the IP of the primary machine. Health statuses for the Controllers are also available. You can Add a Secondary Controller if you would like to create an HA pair, then initiate an HA failover if you want to trigger a failover. You can also start or stop a Controller, Upgrade a Controller and MySQL, and more.

Events Service

The Events Service page displays your Events Service cluster, which can be made using one to three machines. Again there is an entire lifecycle of operations you can do.

Credentials

Credentials are your host's usernames and private keys. They are required to SSH or connect to the hosts via system user name and private keys.

Jobs

All of the jobs that you perform on your platform can be seen on the Jobs page. It is a nice way to keep track of your jobs and also see which jobs have failed.

Configurations

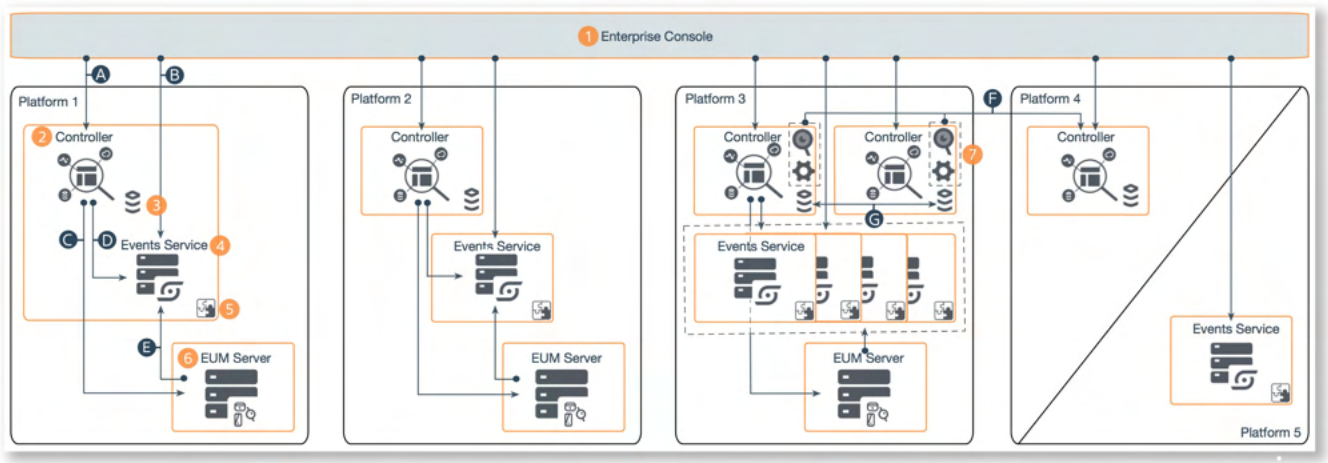
Configurations are important since they let you customize your installations. Configuration settings on the Enterprise Console are separated into three categories: Platform, Controller, and Events Service Settings.

The Controller Settings contains the most configurable settings. The AppServer Configurations under Controller Settings allows you to see all of the Domain configurations which you can initiate from this point or configure your ports. The Database Configurations lets you edit your MySQL settings. So you do not have to tweak the machine, you can do everything from the Console itself.

Enterprise Console Platforms Architecture

The following diagram depicts five platform examples that can be deployed and managed by the Enterprise Console.

> ⚠ You cannot use the Enterprise Console to install the End User Monitoring (EUM) Server. Instead, you must use a package installer that supports interactive GUI or console modes, or a silent response file installation.



Depending on the scale of your deployment, your requirements, and the products you are using, your own application environment is likely to consist of a subset of the components shown in the diagram.

You can find the full On-Premises Deployment Architecture diagram on Application Performance Monitoring Platform, as well as a more detailed On-Premises and SaaS architecture diagram on PDFs.

Enterprise Console Platforms

The following table describes how the components work together in the above platforms.

Platform Number	Components Involved
Platform 1	Platform 1 depicts a single 2 Controller with a local 4 Events Service and 6 EUM Server. The local Events Service contains an 5 API Store.
Platform 2	Platform 2 depicts a single 2 Controller with a remote, single host 4 Events Service and 6 EUM Server. The remote Events Service contains an 5 API Store and can be expanded to a cluster by adding two or more machines.
Platform 3	Platform 3 depicts an HA 2 Controller pair with a remote 4 Events Service cluster and 6 EUM Server. The Events Service cluster contains an 5 API Store on all nodes. The cluster must have three or more nodes.
Platform 4	Platform 4 depicts a single monitoring 2 Controller. This Controller monitors the HA pair in platform 3 by receiving metrics via connection F from the 7 App and Machine Agents. See Manage a High Availability Deployment for more information.
Platform 5	Platform 5 depicts a single shared 4 Events Service. A shared Events Service can connect to multiple 2 Controllers from other platforms, minimizing required maintenance and cost. See Events Service Deployment for more information.

Enterprise Console Platform Connections

The following table lists and describes the traffic flow between the above components in the platforms.

Connection	Source	Destination	Traffic		Protocol	Default Port(s)
A	1 Enterprise Console	2 Controller	Controller Health Checks / Controller Management		HTTP	8090
					HTTPS	8181
B	1 Enterprise Console	4 Events Service	Events Service API Store	Events Service Health Checks / Events Service Management	HTTP(S)	9080
			Events Service API Store Admin		HTTP(S)	9081
C	2 Controller	6 End User Monitoring (EUM) Server	EUM Metric Data		HTTP	7001
					HTTPS	7002 (demo mode only)
D	2 Controller	4 Events Service	Events Service API Store	Analytics Event Data	HTTP(S)	9080
			Events Service API Store Admin		HTTP(S)	9081
E	6 EUM Server	4 Events Service	Events Service API Store	EUM Event Data	HTTP(S)	9080
			Events Service API Store Admin		HTTP(S)	9081
F		2 Controller	Monitoring Metric Data		HTTP	8090

	7 App and Machine Agents			HTTPS	8181
	3 MySQL Database	3 MySQL Database	MySQL Database Replication	TCP	3388

There is no communication from the Controller to the Enterprise Console.

Enterprise Console Requirements

Related pages:

- Controller System Requirements
- Events Service Requirements

The Enterprise Console can run on the same host as the Controller and the embedded Events Service. If this is the case, the machine you choose to run the Enterprise Console must meet the requirements for all the components that run on that machine.

However, we recommend that you place the Enterprise Console on its own separate dedicated host, particularly if you deploy Controllers as High Availability pairs.

Supported Web Browsers

The AppDynamics Enterprise Console UI is an HTML 5-based browser application that works best with the latest version of any browser.

The Enterprise Console UI has been tested with and supports the last two versions of these browsers:

- Safari
- Chrome
- Firefox
- Microsoft Edge
- Internet Explorer

Certain types of ad blockers can interfere with features in the Enterprise Console UI. We recommend disabling adblockers while using the Enterprise Console UI.

CPU Requirements

The Enterprise Console is not CPU intensive and therefore can manage multiple platforms with two Cores.

Memory Space Requirements

The Enterprise Console requires an additional memory of one GB of free RAM for Java and MySQL processes.

Disk Space Requirements

The Enterprise Console requires ten GB of free space to install. After the Enterprise Console installation, there must be at least one GB of additional space on the Enterprise Console host in order to perform any operations, such as installing a remote Controller.

Network Protocol Requirements

The Enterprise Console requires SSH or Secure File Transfer Protocol (SFTP) to be properly configured and enabled for it to use remote hosts.

To access remote hosts, the Enterprise Console uses Java Secure Channel (JSch) API with the provided key file. The Enterprise Console does not support SSH Jump Server. If you use an SSH Jump Server, or have jump host configuration, please contact your AppDynamics representative for deployment options.

cURL

You must install cURL on systems that run Linux.

Required Libraries

Linux systems must include these libraries for Enterprise Console operation:

- `libaio`
- `numactl` package, which includes `libnuma.so.1` for RHEL, CentOS, and Fedora, and `libnuma1` for Ubuntu and Debian
- `glibc2.12`

  This `glibc` version is included into a given operating system release, and therefore cannot be updated.

- `tzdata` for RHEL, CentOS, Fedora, , openSUSE Leap 12 and Leap 15, and Ubuntu version 16 and higher

> The `tzdata` package is also required by the MySQL connector.

- `libncurses5` (and above) for Ubuntu, CentOS, Debian, openSUSE Leap 12 and Leap 15, and Amazon Linux 2

  > As of MySQL 5.5.57 and 5.7.19, `libtinfo.so.5` is a required prerequisite library.

- `ncurses-libs-5.x` for RHEL and CentOS

  > As of MySQL 5.5.57 and 5.7.19, `libtinfo.so.5` is a required prerequisite library.

- SLES12 and SLES15 using `libxml2-2` and `libxml2-tools`

This table provides instructions on how to install the libraries on some common flavors of the Linux operating system.

> If you cannot install the library, check that you have a supported version of your Linux flavor.

Linux Flavor	Command

<table>
<tr>
<td>

- Red Hat
- CentOS
- CentOS Stream
- Amazon

</td>
<td>

Use yum to install the library, such as:

- yum install libaio
- yum install numactl
- yum install tzdata
- yum install ncurses-libs-5.x

Ensure that only one package mgr for rpm and is installed before running the Enterprise Console installer.

For RHEL8, CentOS8, and Amazon2 you can either manually install version 5 of ncurses or use version 6.

- To install version 5, follow these steps:

1. sudo rpm -ivh --force ncurses-base-5.x.rpm
2. sudo rpm -ivh --force ncurses-libs-5.x.rpm

The ncurses-libs depends on the ncurses-base so you must install the ncurses-base first. These are examples of a trusted source for rpm download:

- http://mirror.centos.org/centos/7/os/x86_64/Packages/ncurses-base-5.9-14.20130511.el7_4.noarch.rpm
- http://mirror.centos.org/centos/7/os/x86_64/Packages/ncurses-libs-5.9-14.20130511.el7_4.x86_64.rpm

- To install version 6, follow these steps:

You must either create symlinks for ncurses-libs-5 which points to ncurses-libs-6, or install the ncurses-compat-libs package, to provide ABI version 5 compatibility.

RHEL8 symlink:

```
sudo ln /usr/lib64/libtinfo.so.6.1 /usr/lib64/libtinfo.so.5
sudo ln /usr/lib64/libncurses.so.6.1 /usr/lib64/libncurses.so.5
```

CentOS8 symlink:

```
sudo ln /usr/lib64/libtinfo.so.6.1 /usr/lib64/libtinfo.so.5
sudo ln /usr/lib64/libncurses.so.6.1 /usr/lib64/libncurses.so.5
```

Amazon2 symlink:

```
sudo ln -s /usr/lib64/libncurses.so.6.0 /usr/lib64/libncurses.so.5
sudo ln -s /usr/lib64/libtinfo.so.6.0 /usr/lib64/libtinfo.so.5
```

RHEL8 compat-libs:

```
sudo yum install -y ncurses-compat-libs
```

CentOS8 compat-libs:

```
sudo yum install -y ncurses-compat-libs
```

Amazon2 compat-libs:

```
sudo yum install -y ncurses-compat-libs
```

Use the following prerequisites to install on CentOS Stream:

- sudo yum install -y libaio.x86_64
- sudo yum install -y numactl.x86_64
- sudo yum install -y tzdata
- sudo yum install -y ncurses-compat-libs.x86_64

</td>
</tr>
<tr>
<td>Fedora</td>
<td>

Install the library RPM from the Fedora website:

- yum install libaio
- yum install numactl
- yum install tzdata

</td>
</tr>
</table>

<table>
<tr>
<td>Ubuntu</td>
<td>
Use <code>apt-get</code>, such as:

<code>sudo apt-get install libaio1</code>
<code>sudo apt-get install numactl</code>
<code>sudo apt-get install tzdata</code>
<code>sudo apt-get install libncurses5</code>

⚠ Ensure that only one package <code>mgr</code> between <code>dpkg</code> and <code>rpm</code> is installed before running the Enterprise Console installer. This <code>pkg</code> manager utility will be used to verify mandatory <code>pkgs</code> before the Enterprise Console installation.
ⓘ For Ubuntu20 you can install <code>libncurses5</code> or <code>libncurses6</code>.

If you choose <code>libncurses5</code>:

<code>sudo apt-get install libncurses5</code>
If you choose <code>libncurses6</code>:

<code>sudo apt-get install libncurses6</code>

Note: For <code>libncurses6</code> you need to create symlink for <code>libncurses5</code> pointing to <code>libncurses6</code>.

<code>sudo ln -s /usr/lib/x86_64-linux-gnu/libncurses.so.6.2 /usr/lib/x86_64-linux-gnu/libncurses.so.5</code>
<code>sudo ln -s /usr/lib/x86_64-linux-gnu/libtinfo.so.6.2 /usr/lib/x86_64-linux-gnu/libtinfo.so.5</code>

</td>
</tr>
<tr>
<td>Debian</td>
<td>Use a package manager (such as APT) to install the library (as previously described in the Ubuntu instructions).</td>
</tr>
</table>

<table>
<tr>
<td>openSUSE Leap 12 and Leap 15</td>
<td>
Use zypper to install the library, such as:

sudo zypper install libaio
sudo zypper install libnuma1
sudo zypper install tzdata
sudo zypper install libncurses5

For openSUE Leap 15, you can install libncurses5 or libncurses6.

If you choose libncurses5:
sudo zypper install libncurses5
If you choose libncurses6:
sudo zypper install libncurses6

Note: For libncurses6 you need to create symlink for libncurses5 pointing to libncurses6.
<pre>sudo ln /lib64/libncurses.so.6.1 /lib64/libncurses.so.5
sudo ln -s /lib64/libtinfo.so.6.1 /lib64/libtinfo.so.5ncurses-compat-libs</pre>
Ensure that only one package mgr for rpm and is installed before running the Enterprise Console installer. Also, you need to add the openSUSE machine repository before installing the tzdata package.
<pre>For openSUSE Tumbleweed run the following as root:
zypper addrepo https://download.opensuse.org/repositories/home:amshinde
/openSUSE_Tumbleweed/home:amshinde.repo
zypper refresh
zypper install tzdata

For openSUSE Leap 42.1 run the following as root:
zypper addrepo https://download.opensuse.org/repositories/home:amshinde
/openSUSE_Leap_42.1/home:amshinde.repo
zypper refresh
zypper install tzdata

For openSUSE 13.2 run the following as root:
zypper addrepo https://download.opensuse.org/repositories/home:amshinde/openSUSE_13.2
/home:amshinde.repo
zypper refresh
zypper install tzdata

For openSUSE 13.1 run the following as root:
zypper addrepo https://download.opensuse.org/repositories/home:amshinde/openSUSE_13.1
/home:amshinde.repo
zypper refresh
zypper install tzdata

You may run into file conflicts when two packages attempt to install files with the
same name but different contents. If you choose to continue, the old files and their
contents will be replaced.

See the openSUSE website (https://software.opensuse.org/download.html?project=home%
3Aamshinde&package=tzdata) to manually download and install the tzdata package.</pre>
</td>
</tr>
<tr>
<td>SLES12 and SLES15</td>
<td>
Use zypper to install the library, such as:

sudo zypper install libxml2-2
sudo zypper install libxml2-tools
sudo zypper install libaio1
sudo zypper install numactl
sudo zypper install libcurses5
sudo zypper install tzdata

</td>
</tr>
</table>

See Platform Requirements for operating system support information.

High Availability Requirements

You must install rsync if you plan on deploying a Controller (HA) pair. In addition, when using SSH or an SSH client, note that OpenSSH 5.3p1 is the minimum version supported by the Enterprise Console for HA.

Supported SSH Key Exchanges and Cipher Algorithms

You can use these `ssh` key exchanges and cipher algorithms to customize the `ssh` configuration on your host(s):

Supported ssh	Details
Key Exchanges	• `diffie-hellman-group-exchange-sha1` • `diffie-hellman-group1-sha1` • `diffie-hellman-group14-sha1` • `diffie-hellman-group-exchange-sha256` • `ecdh-sha2-nistp256` • `ecdh-sha2-nistp384` • `ecdh-sha2-nistp521`
Cipher Algorithms	• `blowfish-cbc` • `3des-cbc` • `aes128-cbc` • `aes192-cbc` • `aes256-cbc` • `aes128-ctr` • `aes192-ctr` • `aes256-ctr` • `3des-ctr` • `arcfour` • `arcfour128` • `arcfour256`

Install the Enterprise Console

This page provides information and instructions for installing the AppDynamics Enterprise Console to automate the task of installing and administering the Controller and Events Service. You must install the Enterprise Console to install these components.

About the Enterprise Console Installation

Though the Enterprise Console can run on the same host as the Controller and, if installed, the embedded Events Service, it is recommended that you install it on a separate host. Regardless, the machine that runs the Enterprise Console must meet the requirements for all the components that run on that machine, as outlined below.

The Enterprise Console and Controller must run on separate MySQL instances allowing the Enterprise Console to manage the Controller's instance independent of the Controller host, creating a lightweight setup that consumes less memory.

> ⚠ If you install the Enterprise Console on the Controller machine, it must be in a different directory in order to keep data separate. For instance, if the Controller is installed in `/opt/appdynamics/controller`, the Enterprise Console might be in `/opt/appdynamics/enterpriseconsole`.

You must also avoid port conflicts with the Controller database, which is 3388 by default, whereas the Enterprise Console database is 3377 by default.

The Enterprise Console installation path you choose must be writeable, i.e. the user who installed the Enterprise Console should have write permissions to that directory.

The Enterprise Console prevents multiple users from running commands at the same time. If a second user attempts to run a command while another command is in progress, the second command is not completed and an error message appears indicating that another command is in progress. To avoid such conflicts, the Enterprise Console should generally be used by a single user at a time.

You can enable HTTPS for the Enterprise Console during installation. See HTTPS Support for the Enterprise Console.

> ⚠ Cross-platform (OS) installation, e.g., installing the Enterprise Console on Linux and the Controller on Mac or Windows is not supported.

Installing on AWS Host

When installing the Enterprise Console on an AWS host, you must add the values for the following hostnames and IP addresses to the SAN:

- Public DNS (IPv4)
- IPv4 Public IP
- Private DNS
- Private IPs

Disk and Memory Space Requirements

The host must have enough disk space for the Enterprise Console and a platform, which includes a Controller. There is no need for additional memory for the Enterprise Console when it shares the same host as the Controller. However, when the Enterprise Console host is not shared with the Controller host, then it requires additional disk and memory space. See Enterprise Console Requirements and Prepare the Controller Host to make sure you meet the minimum space requirements.

Software Requirements

On systems that run Linux, you must have cURL and `netstat` installed. Linux systems must also have the `libaio` library installed. This library provides for asynchronous I/O operations on the system.

See Required Libraries for how to install `libaio` and other libraries on some common flavors of the Linux operating system.

Password Requirements

Due to browser incompatibilities, AppDynamics recommends using only ASCII characters for usernames, account names, and passwords

Configure your installation using these password settings:

GUI Mode Screen or Option Label	Response File variable	Description
Database Root User's Password	`mysqlRootUserPassword`	The password of the user account that the Controller uses to access its MySQL database. Do not use the single quotation mark ('), double quotation mark ("), or at sign (@) characters in this password.

Controller root User's Password	`rootUserPassword`	The Controller root user password. The root user is a Controller user account with privileges for accessing the system Administration Console. This password is used for the admin user of the built-in Glassfish application server as well. The Glassfish admin user lets you access the Glassfish console and the `asadmin` utility. See Access the Administration Console. Allowed characters in the password are: a-z, A-Z, 0-9, ., +, =, @, _, -, $, :, #, ,, (,), !, {, }
User Name (Admin User Setup)	`userName`	The username of the administrator account in the Controller UI. This is the administrator for the built-in account if single-tenant systems, or for the initial account for multi-tenant. See Update the Root User and Glassfish Admin Passwords. Usernames and passwords cannot include the @ or ! character. Also note that if this account will be used to access the REST API, additional limitations on the use of special characters in usernames apply. See Create and Manage Tenant Users.

In the password field, you may include the space character at the beginning as well as in the middle of the password string. However, you cannot start passwords with the space character when using the `response.varfile`.

GUI Installation

Before starting, get the Enterprise Console installer version appropriate for your target system. You can get the installer from the AppDynamics Downloads. When ready, follow these steps to install the Enterprise Console:

1. Navigate to the directory where you downloaded the install file.
2. Run the following commands:

   Linux

   ```
   ./platform-setup-64bit-linux.sh
   ```

   You can run the installer as non-root or root.

   Windows

   ```
   platform-setup-64bit-windows.exe
   ```

   It is recommended that you right-click the .exe file and select **Run as Administrator**.

3. After the GUI launches, use it to complete the installation. In Linux, you may also follow the steps in the installation wizard to complete the console installation.

   If you install the Enterprise Console on AWS, use the public DNS for the Enterprise Console hostname when prompted.

Silent Installation

To use the silent installation method, add the -q option, the response file, and the destination directory to the command to run the installer. For example, in Linux, run the following command:

```
./platform-setup-64bit-linux.sh -q -varfile ~/response.varfile
```

It is recommended that, if possible, you provide an absolute path as the installation path specified as the `dir` argument value and not a relative path as shown in the example.

For a Windows system:

```
platform-setup-64bit-windows.exe -q -varfile c:/response.varfile
```

Sample response files for silent installation

Linux

```
serverHostName=HOST_NAME
sys.languageId=en
disableEULA=true

platformAdmin.port=9191
platformAdmin.databasePort=3377
platformAdmin.dataDir=/opt/appdynamics/platform/mysql/data
platformAdmin.databasePassword=ENTER_PASSWORD
platformAdmin.databaseRootPassword=ENTER_PASSWORD
platformAdmin.adminPassword=ENTER_PASSWORD
platformAdmin.useHttps$Boolean=false
sys.installationDir=/opt/appdynamics/platform
```

The `sys.languageID` and `platformAdmin.dataDir` properties are optional. If not specified, the data directory will be in the `/mysql` directory under the platform directory.

Windows

```
serverHostName=HOST_NAME
sys.languageId=en
disableEULA=true
sys.adminRights$Boolean=true

platformAdmin.port=9191
platformAdmin.databasePort=3377
platformAdmin.dataDir=C\:\\AppDynamics\\Platform\\platform-admin\\mysql\\data
platformAdmin.databasePassword=ENTER_PASSWORD
platformAdmin.databaseRootPassword=ENTER_PASSWORD
platformAdmin.adminPassword=ENTER_PASSWORD
platformAdmin.useHttps$Boolean=false
sys.installationDir=C\:\\AppDynamics\\Platform
```

The `sys.languageID` and `platformAdmin.dataDir` properties are optional. If not specified, the data directory will be in the `\mysql` directory under the platform directory.

If you install the Enterprise Console on AWS, use the public DNS for the `serverHostName` value.

After the Installation

After you install the Enterprise Console, you can use the following methods to install the AppDynamics Platform:

- GUI: A graphical interface within a web browser to install the Controller and Events Service. You can select from Express Install or Custom Install of the platform, which includes the option to install a Controller and Events Service.
- Command-line: A CLI to install the Controller and Events Service.

After installing the Enterprise Console, you can select from the Express Install or Custom Install of the platform, which includes the option to install a Controller. For more information about those options, see Enterprise Console.

For information on installing the Controller or Events Service in unattended mode or via the command line, see Enterprise Console Command Line.

Accessing the Enterprise Console

Access the GUI for the Enterprise Console with the following URL:

```
http(s)://<hostname>:<port>
```

Specify the port and hostname you used when you installed the Enterprise Console. The default port is 9191. This port needs to be exposed from your firewall rules so you can access the port from any place. See Port Settings.

For example:

```
http(s)://aHost.aDomain:9191
```

With the GUI, you can install and manage the components of the AppDynamics platform, including tasks such as adding hosts or credentials, installing a Controller, and monitoring jobs.

If you cannot access the GUI, verify that the hostname and port number are correct. Additionally, ensure that the Enterprise Console is running.

The first time you access the GUI, the Enterprise Console shows the following options for installing the AppDynamics Platform:

- **Express:** Select this option for new installations of the Controller and Events Service. The services are installed on the same host.
- **Custom:** Select this option to customize your installation, including installing or upgrading the Controller and Events Service on separate hosts. By installing the Events Service on a separate host, you can create a 1 or 3+ node Events Service based on your needs. Installing an Events Service on a separate host with the Enterprise Console is only supported on Linux. If you want to install the Events Service on a separate host on Windows, see Install the Events Service on Windows.

  > The Events Service is installed by default with a Custom Installation unless you choose to unselect the Install Events Service option.

  - **Discover and Upgrade:** When performing a custom installation, you have the option to discover and upgrade an existing AppDynamics deployment, such as a Controller or Events Service. For example, if you use the package installer to install the Controller in a previous version of AppDynamics, you can use the discover and upgrade option to add the Controller to the AppDynamics platform that the Enterprise Console manages. The application will then upgrade the Controller to the same version of the Enterprise Console. Verify that the Controller and MySQL are running before you attempt to discover and upgrade them.

Troubleshooting the Installation

This section provides troubleshooting information for issues that may arise during the Enterprise Console installation.

Installation Stuck at License Agreement

If your installation becomes stuck at displaying the license agreement on the console, then the EULA may be having issues with special characters. To fix this issue, add the `-VdisableEULA=true` flag to your installation command or `response.var` file. For example:

```
./platform-setup-64bit-linux.sh -c -VdisableEULA=true
```

Enterprise Console Application Is in Use Error

If you get an error that states that the "Enterprise Console Application (9191/3377) is in use," you should check that you have specified the correct hostname during the installation.

Default Font Change on Linux Machines

If your Enterprise Console installation fails on a Red Hat system, it may be due to an install4j issue. If the default font has been changed, the JRE cannot interpret it, leading to a "could not display the GUI" error. You can fix this error by running the installation with `-VdisableEULA=true` and creating the file `/etc/fonts/local.conf` with the following contents:

```
<?xml version='1.0'?>
<!DOCTYPE fontconfig SYSTEM
'fonts.dtd'>
<fontconfig>
  <alias>
    <family>serif</family>

<prefer><family>Utopia</family></prefer>
  </alias>
  <alias>

<family>sans-serif</family>
    <prefer><family>Utopia</family></prefer>

</alias>
  <alias>
    <family>monospace</family>

<prefer><family>Utopia</family></prefer>
  </alias>
  <alias>

<family>dialog</family>
    <prefer><family>Utopia</family></prefer>

</alias>
  <alias>
    <family>dialoginput</family>

<prefer><family>Utopia</family></prefer>
  </alias>
</fontconfig>
```

Rename the Directory Error on Windows Machines

If the Enterprise Console installation fails with an error on "rename of the directory," it may be due to an antivirus scan. Stopping the antivirus scan on the machine fixes the issue. You should also exclude the Enterprise Console directory from the scan if it sits outside of the Controller directory. See Prepare Windows for the Controller.

Express Install

Related pages:

- Controller System Requirements
- Events Service Requirements

You can choose Express Install on the Install page when you first use the Enterprise Console or when you want to create a new platform. This install option provides you with a quick and simple way to install a fresh single node Controller and embedded Events Service.

This page guides you through the steps to perform an Express Install. Before you install the Controller with the Enterprise Console, verify that the Enterprise Console is running and the host machine meets the requirements for the Controller.

If you use the GUI to install the Controller, you can create the platform at the same time. If you use the command line, you must create the platform prior to installing the Controller. For more information, see Administer the Enterprise Console. Express Install does not give you the option to choose the version you would like to install, and instead automatically installs the latest version of each service. If you would like more control over your installation, see Custom Install.

Install the Platform Using GUI

Express Install is the quickest way to get started with setting up your own AppDynamics Platform. You can use it to install the Controller and an Events Service on a single host.

After you install the Enterprise Console, you can complete the platform installation process with the GUI:

1. Check that you have fulfilled the Enterprise Console prerequisites before starting.
2. Open a browser and navigate to the GUI:

   ```
   http(s)://<hostname>:<port>
   ```

   9191 is the default port.
3. Select **Express Install** to install a Controller and Events Service on a shared host.
4. Enter a Name and the Installation Path for your platform.

   > The Installation Path is an absolute path under which all of the platform components are installed. The same path is used for all hosts added to the platform. Use a path which does not have any existing AppDynamics components (Controller, Events Service, etc.) installed under it.
   > The path you choose must be writeable, i.e. the user who installed the Enterprise Console should have write permissions to that folder. Also, the same path should be writeable on all of the hosts that the Enterprise Console manages.
   >
   > Example path: `/home/appduser/appdynamics/product`

5. Add a host by entering host machine-related information: Host Name, Username, and Private Key. To add the Enterprise Console host, click **Add Enterprise Console Host,** which will automatically populate the text field with the hostname of the Enterprise Console machine. You do not need to provide a credential. The Controller and Events Service will be installed on the same host as the Enterprise Console.

   > If the Controller is to be installed on a Windows machine, the Enterprise Console should be on the same machine. This is because Windows hosts are not supported on the Enterprise Console.

   > If you use the Enterprise Console host to install the Controller, then no Username or Private Key is required. You can also install the Controller and Events Service on a remote host. In that case, the Username and Private Key of the remote host would be required.
   >
   > As of OpenSSH version 7.8, the `ssh` keys generated are in OpenSSH format using Ed25519. However, the Enterprise Console expects the `ssh` keys to be formatted using the older `pem` format.

6. Install the Controller:
   a. Select a Profile size for your Controller. See Controller System Requirements for more information on the sizing requirements.
   b. Enter the required Username and Passwords. The default Controller Admin Username is admin.
7. Click **Install**.

After clicking Install you can monitor the status of your platform creation jobs on the Jobs page, which include Add Hosts, Controller Install, and Events Service Install Jobs. Note that the Controller Install Job takes a considerably longer time to complete than the other two jobs. When the jobs successfully complete, you can check the status of your platform, obtain the URL of the Controller, update platform configurations, and manage the lifecycle of your services.

See Install the Events Service on Linux for additional setting requirements and to learn how to scale up an embedded Events Service.

Custom Install

Related pages:

- Controller System Requirements
- Events Service Requirements

The Controller is the central component of the AppDynamics platform. Other components such as the Events Service connect to the Controller and stream metrics to be displayed. The Enterprise Console Custom Install option provides you with a configurable way to install a fresh Controller and Events Service. You can also Discover & Upgrade older platform services using Custom Install.

This page describes how to use Custom Install to install the Controller. Before you install the Controller with the Enterprise Console, verify that the Enterprise Console is running and the host machine meets the requirements for the Controller.

If you use the GUI to install the Controller, you can create the platform at the same time. If you use the command line, you must create the platform prior to installing the Controller. For more information, see Administer the Enterprise Console. If instead, you would like to get started with your platform as soon as possible, see Express Install.

Install the Controller Using GUI

Installing a fresh Controller, HA-pair, or Events Service cluster is simple with Custom Install. You can use it to configure your platform deployment freely.

After you install the Enterprise Console, you can complete the platform installation process with the GUI:

1. Check that you have fulfilled the Enterprise Console prerequisites before starting.
2. Open a browser and navigate to the GUI:

   ```
   http(s)://<hostname>:<port>
   ```

   9191 is the default port.
3. Navigate to the **Install** homepage and click **Custom Install**.
4. Enter a name and installation path for your platform.

   > The Installation Path is an absolute path under which all of the platform components are installed. The same path is used for all hosts added to the platform. Use a path which does not have any existing AppDynamics components installed under it.
   > The path you choose must be writeable, i.e. the user who installed the Enterprise Console should have write permissions to that folder. Also, the same path should be writable on all of the hosts that the Enterprise Console manages.
   >
   > Example path: `/home/appduser/appdynamics/product`

5. Add a host by entering host machine-related information: Host Name, Username, and Private Key. This is where the Controller and Events Service will be installed onto.

   > As of OpenSSH version 7.8, the `ssh` keys generated are in OpenSSH format using Ed25519. However, the Enterprise Console expects the `ssh` keys to be formatted using the older `pem` format.

   For more information about how to add credentials and hosts, see Administer the Enterprise Console.

   > If the Controller is to be installed on a Windows machine, the Enterprise Console should be on the same machine. This is because Windows hosts are not supported on the Enterprise Console.

6. Install Controller:
   a. Select **Install**.
   b. Select an available Target Version from the dropdown list.

      > The list is populated by versions that the Enterprise Console is aware of. This means that you can install the Controller to any intermediate version or to the latest version as long as the Enterprise Console installer has been run for those versions.

   c. Select a Profile size for your Controller. See Controller System Requirements for more information on the sizing requirements.
   d. Enter the Controller Primary Host.
   e. Enter the Controller Secondary Host if you would like to install an HA pair.

> It is highly recommended that you have the Enterprise Console on its own separate dedicated host, especially in the case of HA installations. This means you would need three hosts for the recommended HA setup. If the Enterprise Console and Controller are on the same host and that host becomes unavailable, the Enterprise Console will not be able to failover to the other Controller.
> You can set up an HA pair at a later time. See Set Up a High Availability Deployment for additional setting requirements and to learn how to add a secondary controller after your initial installation.

f. Optional: Enter the Tenancy Mode.
g. Under Advanced, enter additional information for your HA pair.
h. Enter the required Username and Passwords. The default Controller Admin Username is admin.

> If you do not install a Controller at this time, you can always do so later by navigating to the **Controller** page in the GUI and clicking Install Controller.

7. Install Events Service:
   a. Select **Install**.
   b. Select the Profile size for your Events Service. See Controller System Requirements for more information on the sizing requirements.
   c. Optional: Enter the Installation and Data Directory.

> You do not need to specify the installation or data directory for the Events Service installation. If you do, use a different one from the platform or mysql data directory.

   d. Optional: Enter the Elastic Search, REST API Admin, REST API, and Unicast Ports.
   e. Enter the Events Service Host. You can deploy a single Events Service node or an Events Service cluster made up of three or more nodes. The minimum size of an Events Service cluster is three. You can always add more at a later time on the Events Service page.

> You can set up a scaled up Events Service at a later time. See Install the Events Service on Linux for additional setting requirements and to learn how to scale up an embedded Events Service.

> If you do not install an Events Service at that time, you can always do so later by navigating to the **Events Service** page in the GUI and clicking Install Events Service.

8. Click **Install**.

After clicking Install you can monitor the status of your platform creation jobs on the Jobs page, which include Add Hosts, Controller Install, and Events Service Install Jobs. Note that the Controller Install Job takes a considerably longer time to complete than the other two jobs. When the jobs successfully complete, you can check the status of your platform, obtain the URL of the Controller, update platform configurations, and manage the lifecycle of your services.

Once any high availability controllers are installed, they can be found under the Controller page. All high availability lifecycle operations such as start, stop, upgrade, and failover can be performed from this page.

Administer the Enterprise Console

You can use the GUI or command line to perform the following platform administration tasks with the Enterprise Console:

- Create new platforms
- Remove existing platforms
- Manage credentials
- Manage hosts

> Some tasks may not be available through both the GUI and command line. Most of the commands described here are based on the Linux command line and cover common tasks for managing the platform. You can run similar commands with the Windows command prompt by replacing the `platform-admin.sh` script with `platform-admin.exe cli`.

Run the commands from the `<Enterprise Console installation directory>/platform-admin` directory. This page contains the minimum options and parameters required to run a command.

Some commands may have more options and parameters. To see these additional options, run the command with `-h` specified. For example, run the following command to see all the options and parameters for the create a platform command:

Start or Stop the Enterprise Console

The Enterprise Console must be running before you can perform other tasks or use the GUI.

> The same user who installed the Enterprise Console should be the same user who starts or stops the Enterprise Console.

Use the following commands to start or stop the Enterprise Console:

To stop the Enterprise Console, replace `start` with `stop`.

Manage Platforms

The platform is the collection of AppDynamics components and their hosts. The Enterprise Console supports up to 20 platforms at a time by default.

Create a Platform

To use the Enterprise Console for end-to-end installation and management, you must first create a platform. The Enterprise Console creates the platform when you complete the Express or Custom installations or discover existing components in the GUI.

To use the command line to create a platform, run the following command:

The platform installation directory is the absolute directory where the Enterprise Console installs all AppDynamics components on all of its hosts. Once you add a host to the platform, you can no longer change the directory. Additionally, the directory cannot contain a space.

Delete a Platform

You can use the Enterprise Console to delete a platform that is no longer in use. You may also want to consider editing the Platform's configuration instead. You can perform either action on the Platform view page of the GUI.

To use the command line to delete a platform, run the following command:

> If you just deleted your current platform, you must clear the value of the `APPD_CURRENT_PLATFORM` variable to prevent unexpected errors when running future commands.

Manage Credentials

Manage the credentials that the Enterprise Console uses to access and perform tasks on hosts, such as adding a node to an Events Service. You can use the Credentials page in the GUI or the command line to manage credentials. You should add credentials to the platform before you add hosts.

Generate an RSA Key Pair

An RSA private key file is required to add credentials to a platform. The following steps will generate an RSA key pair that consists of the public key file `~/.ssh/id_rsa.pub` and the private key file `~/.ssh/id_rsa`.

1. Log in to the Enterprise Console host machine via SSH.

2. Switch to the user that is the owner of the Enterprise Console:

```
sudo -i -u <user-owner of the EC>
```

3. Create the RSA key pair:

```
ssh-keygen -t rsa -b 2048 -N '' -m pem
```

4. Accept the default location for the key pair at `~/.ssh`.
5. Confirm that the RSA public and private key files have been created:

```
ls ~/.ssh/
id_rsa
id_rsa.pub
```

Add Credential

When you add a credential, you need the following information:

- Credential name
- Username
- Private key file

The credential name is the unique identifier for a credential and is used to specify the credential when you perform tasks such as adding a host. AppDynamics recommends that you follow the naming convention for all of your credential names. The `id_rsa`, RSA private key, should be created using the OpenSSL PEM encoding format over the Open SSH standard encoding.

The Unix system user specified in the username field must have writeable access to the platform directory.

You can add a credential in the GUI by clicking **Add**.

```
bin/platform-admin.sh add-credential --credential-name <name> --type <ssh> --user-name <username> --ssh-key-
file <file path to the key file>
```

Where `<file path to the key file>` is the private key for the Enterprise Console machine. The installation process deploys the keys to the hosts.

Remove Credential

Remove a credential that is no longer used. You cannot remove a credential that is still used by a host. You can remove a credential in the GUI by selecting the credential and clicking **Delete**.

```
bin/platform-admin.sh remove-credential --credential-name <name>
```

List Current Credentials

```
bin/platform-admin.sh list-credentials
```

Manage Hosts

The hosts are the machines used to run AppDynamics components such as the Controller and Events Service. For example, the Events Service can run on the same host as the Controller, a single host, or a cluster of three or more hosts.

You must properly configure the credential you use to add a new host on a remote host. This means that for a private key that you have specified, you must add the corresponding public key to the remote host `~/.ssh/authorized_hosts` file.

The Controller and Events Service must reside on the same local network and communicate by internal network. Do not deploy the cluster to nodes on different networks, whether relative to each other or to the Controller where the Enterprise Console runs. When identifying cluster hosts in the configuration, you will need to use the internal DNS name or IP address of the host, not the externally routable DNS name.

For example, in terms of an AWS deployment, use the private IP address such as 172.31.2.19 rather than public DNS hostname such as `ec2-34-201-129-89.us-west-2.compute.amazonaws.com`. You must then go to the Appserver Configurations under Controller Settings in the Enterprise Console GUI, and edit the external URL so you can access the page.

The host that runs the Enterprise Console is automatically created and added to the platform as the hostname of the Enterprise Console machine if you use the GUI to install or discover components. If you do not use the GUI, you must manually add this host.

Hosts can be managed through the Enterprise Console GUI on the Hosts page or the command line.

The `id_rsa`, RSA private key, should be created using the OpenSSL PEM encoding format over the Open SSH standard encoding.

Set Up Remote Hosts

To set up seamless Enterprise Console communications with remote hosts, perform the following steps:

From the command line:

1. Set up the following passwordless SSH:
    - From the Enterprise Console to the Controller.
    - If HA then:
        - from the Enterprise Console to the primary Controller.
        - from the Enterprise Console to the secondary Controller.
        - From the primary Controller to the secondary Controller.
        - from the secondary Controller to the primary Controller.

From the Enterprise Console:

2. Identify the Enterprise Console Linux AppDynamics user's public/private key pair (usually in `~/.ssh`).

3. Add the Enterprise Console Linux AppDynamics user's public key (`~/.ssh/id_rsa.pub`) into the remote Controller server Linux user's `~/.ssh/authorized_keys`.

4. Then `chmod 600 /.ssh/authorized_keys`.

5. Test the SSH connection from the Enterprise Console server with the following:

```
ssh <remote user>@<remote-server> hostname
```

6. Verify that the remote hostname is printed. You may have to first answer yes to trust the server fingerprint.

7. Access the Enterprise Console UI Credential page, and add or edit the existing credentials.

8. Create a single credential, which will likely be the same for all remote hosts, with a name like: `EC-<ec linux user name>-<remote appd user name>` For example, `EC-ecappduser-appduser` , or `EC-appdyn` if the username is the same on the Enterprise Console and remote server.

9. Enter the remote server Linux username. It might be the same as the local Enterprise Console AppDynamics user.

10. Supply the Enterprise Console Linux user's `~/.ssh/id_rsa` contents as the private key as chosen in step 2.

Add Hosts

Before you add hosts to the platform, ensure that the required credentials are added to the platform.

You should also check that the user ID created matches those in the credentials. Additionally, the path you specify for the platform base directory must exist.

You can add a host in the GUI by clicking **Add**.

```
bin/platform-admin.sh add-hosts --hosts host_1 host_2 host_3 --credential <credential name>
```

Instead of listing the hosts with --hosts, you can specify a text file with a line-separated list using the following command:

```
bin/platform-admin.sh add-hosts --host-file <file path to host file> --credential <credential name>
```

If you do not use the GUI, you must add the host for the Enterprise Console. This is also the host used by the Controller and embedded Events Service. The host is named "localhost" and does not require credentials. For example, run the following command:

> You may also use the loopback address '127.0.0.1' or the machine's actual hostname.

Remove Hosts

Before you remove a host, ensure that you remove all AppDynamics components from the host. You can remove a host in the GUI by selecting the host and clicking **Remove**.

```
bin/platform-admin.sh remove-hosts --hosts host_1 host_2 host_3
```

Instead of listing the hosts with `--hosts`, you can specify a text file with a line-separated list using the following command:

```
bin/platform-admin.sh remove-hosts --host-file <file path to host file>
```

If a host becomes unreachable, you can use the following command to remove it:

```
bin/platform-admin.sh remove-dead-hosts --hosts <host name>
```

This removes the host and all of its associated metadata from the Enterprise Console database.

> Running `remove-dead-hosts` could leave various services in an inconsistent state.

List Current Hosts

Update Host Credentials

Change the credential that the Enterprise Console uses to access hosts. You can change the host credential in the GUI by selecting the host and clicking **Change Credentials**.

```
bin/platform-admin.sh update-host-credential --hosts host_1 host_2 host_3 --credential <credential name>
```

Similar to the add and remove host commands, you can specify a text file instead of providing a list of hosts within the command.

Manage the Enterprise Console Admin User Password

Change the Admin User Password

You can change the password for the admin user with the following command:

Reset the Admin User Password

You can reset the password for the Enterprise Console's root user with the following command:

Resetting the password sets it to "admin".

Manage the Enterprise Console Database Root User Password

Change the Database Root User Password

To change the `platformAdmin.databaseRootPassword`, follow these steps:

1. Stop the Enterprise Console by running the following command in <EC_home>/Platform/platform-admin/bin:

```
bin/platform-admin.sh stop-platform-admin
```

2. Run the following command in <EC_home>/Platform/mysql/bin:

```
bin/mysqld --defaults-file="/<EC_home>/Platform/mysql/db.cnf" --skip-grant-tables
```

   Replace <EC_home> before running the command.
3. Open a new command prompt window.
4. Connect to the database without using a password by running the following command in <EC_home>/Platform/mysql/bin:

```
bin/mysql -u root -h 127.0.0.1 -P 3377 --protocol=TCP
```

5. Execute the following queries:

```
update mysql.user set authentication_string=password('<new_password_here>') where user like 'root%';
flush privileges;
quit;
```

   Replace <new_password_here> before executing the queries.
6. Quit the command prompt.
7. Stop the Enterprise Console Database by running the following command in <EC_home>/Platform/platform-admin/bin:

```
bin/platform-admim.sh stop-platform-admin
```

   ⚠ This is to stop the MySQL DB and start it without using the --skip-grant-tables option that is in the next step.

8. Start the Enterprise Console by running the following command in <EC_home>/Platform/platform-admin/bin:

```
bin/platform-admin.sh start-platform-admin
```

9. Verify the login by running the following command in <EC_home>/Platform/mysql/bin:

```
bin/mysql -u root -p -P 3377 -h 127.0.0.1
```

Manage the Enterprise Console Database User Password

Change the platformAdmin Database User Password

To change the platformAdmin.databasePassword, follow these steps:

ⓘ Use the application ID that owns the AppDynamics platform installation when running the commands.

1. Start the Enterprise Console, if it is not already running.
   Run the following command, replacing <EC_home> before running the command:

```
<EC_home>/platform/platform-admin/bin: ./platform-admin.sh start-platform-admin
```

2. Navigate to the <EC_home>/platform/mysql/bin directory.
3. Log in to the database as the root user by running the following command:

```
./mysql -u root -p -h 127.0.0.1 -P 3377 --protocol=TCP
```

4. Execute the following queries, replacing <new_password_here> before executing the query.

```
update mysql.user set authentication_string=password('<new_password_here>') where user like
'platformadmin%'; flush privileges; quit;
```

5. Verify the login by running the following command in the <EC_home>/Platform/mysql/bin directory:

```
./mysql -u platformadmin -p -P 3377 -h 127.0.0.1
```

6. Navigate to the <EC_home>/platform/platform-admin/bin directory.
7. Run the command below to encrypt the new password that was set for platformadmin in step 4.

```
./platform-admin.sh encrypt --text '<new_password_here>'
```

   Take note of the encrypted password.
8. Navigate to the <EC_home>/Platform/platform-admin/conf directory.
9. Backup the PlatformAdminApplication.yml file.
10. Using a text editor, such as vi, open the PlatformAdminApplication.yml file and find the encrypted password: line in the database: section.
11. Update the encrypted password, replacing the text after the password: entry with the password noted after step 7 and save the changes.
12. Stop the Enterprise Console by running the following command in the <EC_home>/Platform/platform-admin/bin directory:

```
./platform-admin.sh stop-platform-admin
```

13. Start the Enterprise Console by running the following command in the <EC_home>/Platform/platform-admin/bin directory:

```
./platform-admin.sh start-platform-admin
```

14. Validate that the Enterprise Console started successfully.

Change the Installation Directory

The installation directory you specify when creating the platform is used to install all AppDynamics components. This directory can be changed but requires that you uninstall all AppDynamics components.

1. Uninstall all AppDynamics components that you installed.
2. Remove all hosts from the platform, including the localhost. For example, run the following command to remove the localhost:

```
bin/platform-admin.sh remove-hosts --hosts localhost
```

3. Change the installation directory:

```
bin/platform-admin.sh update-platform --installation-dir <directory>
```

> The installation directory cannot contain a space

After you change the installation directory, you must add hosts and reinstall AppDynamics components.

Troubleshooting Administration Tasks

Check Availability of the Enterprise Console

You can use the following API to check the availability of the Enterprise Console:

```
http://econsole-host:9191/service/version
```

Error While Adding Hosts

While adding hosts to your platform, you may run into an "Enterprise Console host expansion failed" error. If you do, you should ensure that SFTP is enabled in your `sshd_config`, and restart the SSH service.

Remove Unused Artifacts

The purge clean up script (`purge.sh`) is available from Enterprise Console version 4.5.17 or later.

To remove unused artifacts from a prior Enterprise Console installation, you can run a purge clean up script (`purge.sh`). The `purge.sh` script preserves any artifacts required to run and upgrade the components in your existing platform.

- The purge clean up script must be run with the same user who installed the Enterprise Console.
- Verify that there are no jobs currently running from the Enterprise Console.

From the CLI:

1. Use the `plan` command to show the purge script execution plan.

   ```
   <platform-admin>/pa-purger/purge.sh plan
   ```

   The execution plan shows you the exact commands to run when you use the `apply` command.
2. Use the apply command to run and apply the commands of the purge script execution plan.

   ```
   <platform-admin>/pa-purger/purge.sh apply
   ```

3. Restart the Enterprise Console.

Old JREs that are not being used by any product services (Events Service or Controller) are cleared during the implementation of job `upgrade-orcha` which is ran when the Enterprise Console is upgraded. You can use the `upgrade-orcha` job on the `adhoc` basis to clean up old JREs.

```
#linux
./platform-admin.sh upgrade-orcha
#windows
platform-admin cli upgrade-orcha
```

Enterprise Console Command Line

The Enterprise Console command line utility allows you to perform orchestration tasks in an automated way. It is designed with the following limitations in mind:

- There is not a complete match of all the functionalities provided by the web UI.
- The command line utility is only supported to run on the host where the Enterprise Console is installed.

Command Line Directory

The platform-admin.sh|bat script in the `<Enterprise Console home>/platform-admin/bin` directory on the host machine provides a set of commands to install and manage the AppDynamics platform.

To see the operations available for the Enterprise Console, from the command line, navigate to directory and run the script with -h specified:

And you can view the format for each command in the command line by specifying the -h argument for a specific command:

You can also use `bin/platform-admin.sh list-jobs --service <controller or events-service>` to see a list of jobs available for the provided service. You can then see what parameters are required for the provided job using `bin/platform-admin.sh list-job-parameters --job <job_name> --service <controller or events-service>.`

Not all commands available on Linux are available on Windows. Refer to the list of the Enterprise Console commands displayed with the `-h` parameter.

The Enterprise Console prevents multiple users from running commands at the same time. If a second user attempts to run a command while another command is in progress, the second command is not completed and an error message appears indicating that another command is in progress. To avoid such conflicts, the Enterprise Console should generally be used by a single user at a time.

Logging into and out of the Enterprise Console

Commands for logging in and out of the Enterprise Console are:

- `login --user-name <admin_username> --password <admin_password>`
  If it has been more than one day since your last session, you will have to log in before you are able to use the command line utility. You will also have to log in again if you see the following error message:

  ```
  error: Command failed due to an error: Unauthorized
  API code 401
  Session expired. Please login and run the command again.
  ```

- `logout`

Managing the Password of the Enterprise Console

Commands for resetting or changing your Enterprise Console password are:

- `reset-password`
  If you forget your admin password or run into a 401 error, run this command to reset your password to its default value, `admin`. You will need to log out then log back in for this change to take effect.
- `change-password --user-name <username> --password <old_password> --new-password <new_password>`

Starting and Stopping the Enterprise Console

Just as you can start and stop the Controller and Events Service, you can start and stop the Enterprise Console process. Commands for starting and stopping the Enterprise Console are:

- `start-platform-admin`
- `stop-platform-admin`

The Enterprise Console must be running to install or administer Events Service nodes.

Getting Platform Versions

You can get the version of the Enterprise Console and any other components that you installed with the Enterprise Console.

To get the version of the Enterprise Console:

To get the version of one of the platform components:

Setting the Current Platform

The optional parameter, `--platform-name`, can be passed in each of your commands to set the current working platform. However, you can set the environment variable, `APPD_CURRENT_PLATFORM`, so that you do not have to pass the current working parameter with each of your commands. You can set this variable using `setenv` or `export`. The Enterprise Console will pick up the value if it is present.

If you happen to provide the `--platform-name` parameter while `APPD_CURRENT_PLATFORM` is set, the value passed through the flag will override the environment variable.

All Platforms Flag

You can use the `--all` flag to denote that you want to modify all platforms with your command. For example, you can upgrade all platform binaries at once by running the following command:

Additional Enterprise Console Commands

The following is a list of frequently used Enterprise Console commands:

- `create-platform --name <platform_name> --description <description> --installation-dir <install_dir>`
- `delete-platform --name <platform_name>`
- `upgrade-orcha --platform-name <platform_name>`
  This command triggers an upgrade of the Orcha module binaries on all remote Orcha machines in the provided platform. The platform should not be running any jobs on its services when you run this command.
- `list-supported-services`
- `show-platform-admin-version`

Update Platform Configurations

Configurations are important since they let you customize your installations. The Enterprise Console enables you to configure these settings via GUI and CLI. However, note that there is limited support for updating service configurations through the CLI. Therefore, it is recommended that you use the GUI for updating configurations, especially for multi-line values.

Configuration settings on the Enterprise Console are separated into three categories: Platform, Controller, and Events Service Settings.

Platform Properties

The Platform configuration allows you to update platform description in the UI. Updating the Platform path is only allowed in the CLI.

Controller Settings

The Controller Settings pages allow you to tune your controller. You can configure settings such as database configurations, JVM options, listeners, and thread pools, for both single or high availability controllers.

Appserver Configurations

The AppServer Configurations page under Controller Settings allows you to edit most of the domain.xml configurations. You can also change the ports and update the controller from a smaller to a higher profile. The configurations are categorized under Basic, JVM Options, and SSL Certificate Management:

Basic

- Profile: Demo, small, medium, or large

  You can change the Controller profile from a smaller profile to a larger one. Before doing so, ensure that the host machine meets the requirements for the profile size you want to use.

  > The Enterprise Console checks the disk size for the transaction log dir and db data dir for medium and large profiles only. If the transaction log is in a separate mount, then it will check for half of the minimum recommended disk size.

  This process is not reversible, and you cannot move from a larger to a smaller profile size. If you tune the Controller heap settings or database configuration settings, even to be greater than the recommended settings for the new profile, those settings will be preserved. Otherwise, the AppDynamics recommended settings are applied.
  To increase the Controller profile size, navigate to AppServer Configurations by choosing the platform, **Configurations**, **Controller Settings**, and **Appserver Configurations**. At the top of the page, select a new profile, then click **Save**.
  Alternatively, you can also use the CLI to increase the Controller profile size to meet increased demand:

  ```
  bin/platform-admin.sh update-controller-profile --profile <profile size>
  ```

  For more information, see Controller System Requirements.
- Tenancy Mode
- External Load Balancer URL (HA only): This is the deep link URL.
- Internal Virtual IP Address (HA only)
- Ports: Server Port: 8090; Admin Port: 4848; SSL Port: 8181; IIOP Port: 3700; JMS Port: 7676
  To change the Controller ports, navigate to AppServer Configurations by choosing the platform, **Configurations**, **Controller Settings**, and **Appserver Configurations**. Near the top of the page, specify new ports and scroll down to click **Save**. This will restart the Controller. Note that the new ports should be available.
- Glassfish Admin Password
- Database User Password
  If you do not specify this password then it will use the default. Click How to Change the Controller Database Root User Password for detailed steps.

  > After a fresh installation or upgrade, the database user password will be hidden in `domain.xml` in the Appserver directory as an alias.

  Alternatively, you can also use the CLI to change your database user password:

  For MySQL DB:

  ```
  bin/platform-admin.sh submit-job --platform-name <platform_name> --service controller --job update-
  passwords --args newDatabaseUserPassword=<password>
  ```

- Advanced configurations
  - NUMA Controller Configuration: This setting is preserved upon upgrades.
  - NUMA Database Configuration: This setting is preserved upon upgrades.

JVM Options

- JVM Options: You can update the JVM options via this page without having to use the `modifyJvmOptions` utility or any other external scripts.
- Domain Http Services
- Domain Protocols
- Domain Network Listeners

> ⚠ Disabling the HTTPS listener is not allowed.

- Domain Transports
- Domain Thread Pools

You can also update the domain config settings using the CLI by following the steps below:

1. Download all four configurations to individual files. See Deploy the Controller on AWS for more information on the config file.
2. Create and load four variables:
    - `new_network=`cat domain-network-listeners.txt``
    - `new_protocol=`cat domain-protocols.txt``
    - `new_thread=`cat domain-thread-pools.txt``
    - `new_transports=`cat domain-transports.txt``
3. Go to `platform-admin/bin` and log in.

```
cd platform-admin/bin
```

```
./platform-admin.sh login --user-name=admin --password=password
```

4. Run the following command on the Enterprise Console host:

```
./platform-admin.sh update-service-configurations --service controller --job update-configs --args
domainProtocols="$new_protocol" domainTransports="$new_transports" domainNetworkListeners="$new_network"
domainThreadpools="$new_thread"
```

SSL Certificate Management

- You can view and edit the SSL Certificate here.

Controller Database Configurations

You can make database configuration changes using:

- Database Configuration UI Page
- Enterprise Console CLI

Database Configuration UI Page

Use the Database Configuration UI page to edit your MySQL settings. This is helpful since you do not have to tweak the configuration file on the database host.

If your RAM memory is greater than 200 GB and you are using a NUMA based architecture, you can specify the Linux nodes (typically CPU socket numbers) from which both processes and memory will be allocated for each AppDynamics component. For example on a two-socket motherboard, AppDynamics recommends the following node configuration settings:

- Glassfish should should allocate its threads/processes and memory on the first node: 0
- MySQL should allocate its threads/processes and memory on the second node: 1

> ⓘ For the node configuration settings, you can enter an integer or comma separated list of integers. For example, for Glassfish, you can enter **0** or **0,1**; for MySQL, you can enter **1** or **2,3**.

The configurations include:

- DB Configuration Settings
  Data Directory: You can change the `datadir` path and database port via this page.

> ⓘ You cannot change certain configurations, such as the MySQL root directory, through the Enterprise Console.

- DB Root Password
  The Enterprise Console does not allow you to change the MySQL root password. However, if you change the MySQL root password for the Controller, you should update the database root password in the Database Configurations page so that the Enterprise Console is aware of the new password.

Enterprise Console CLI

You can update the Controller database configuration programmatically using the Enterprise Console CLI. This enables you to preserve configuration settings during an upgrade.

- If you are using High Availability Toolkit (HATK), you must manually apply these settings on the secondary Controller and restart the secondary server.
- These instructions are specific to the UNIX operating system.

To update the database configuration using CLI:

1. Copy the `db.cnf` file from your primary Controller host onto the Enterprise Console host, for example `db.cnf.new` file.
2. Edit the `db.cnf.new` file to add new settings or update existing values.
3. Load the `db.cnf.new` file into an environment variable:

   ```
   new_db_cnf=`cat db.cnf.new`
   ```

4. Go to the `platform-admin/bin` directory and log in:

   ```
   ./platform-admin.sh login --user-name=admin --password=password
   ```

5. Run the following command on the Enterprise Console host:

   ```
   bin/platform-admin.sh submit-job --service controller --job db-update-config --args
   mysqlCnfContent="$new_db_cnf"
   ```

Reports Service Configurations

The Reports Service Configurations page allows you to update the reports service ports:

- Reporting Service HTTP Port: 8020
- Reporting Service HTTPS Port: 8021

You can also view and edit the SSL Certificate here.

Events Service Settings

The Events Service configurations are read-only. See Administer the Events Service to learn how you can manage your Events Service. You can see the following configurations:

- Profile: This value is either Dev or Prod.
- Installation Directory
- Data Directory
- Ports
  - Elastic Search Port: 9200
  - REST API Port: 9080
  - REST API Admin Port: 9081
  - Unicast Port: 9300

Update Controller Configurations

You can update Controller configurations such as the deep link URL, JVM options, and network listeners.

Update the Deep Link URL

To update the deep link URL:

1. Go to `platform-admin/bin` and log in.

```
cd platform-admin/bin
```

```
./platform-admin.sh login --user-name=admin --password=password
```

2. Run the following command on the Enterprise Console host:

```
./platform-admin.sh update-service-configurations --service controller --job update-configs --args
controllerExternalUrl=<server-protocol>://<controller-host>:<controller-port>
```

where `server-protocol` is http or https.

Update JVM Options

To update JVM options and network listeners:

1. Go to `platform-admin/bin` and log in.

```
cd platform-admin/bin
```

```
./platform-admin.sh login --user-name=admin --password=password
```

2. List all of the Controller configurations from the Enterprise Console:

```
./platform-admin.sh list-service-configurations --service controller > controller-configs.conf
```

3. Open `controller-configs.conf`, and copy all JVM options. Then paste them into a separate file, and edit the desired parameters.
4. Run the following command on the Enterprise Console host:

```
./platform-admin.sh update-service-configurations --service controller --job update-configs --args
controllerNonHostJvmOptions='All JVM options from the previous step including the updated fields'
```

For example:

```
./platform-admin.sh update-service-configurations --service controller --job update_configs --args
controllerNonHostJvmOptions='-Djava.awt.headless=true, -Djdk.corba.allowOutputStreamSubclass=true, ...'
```

Retaining Configuration Changes

The Enterprise Console recognizes and retains many common customizations to the `domain.xml`, `db.cnf`, and other configuration files, but is not guaranteed to retain them all. If you have made manual configuration changes to the files, verify the configuration after updating.

You can also remove the Controller from the Enterprise Console and rediscover it to preserve the configuration changes:

1. Go to `platform-admin/bin` and log in.

```
cd platform-admin/bin
```

```
./platform-admin.sh login --user-name=admin --password=password
```

2. On the Controller page, click on **Remove Controller**, or run the following commands on the Enterprise Console host:

If `removeBinaries=false` then the Enterprise Console forgets the Controller without impacting or uninstalling the Controller.

3. Discover the Controller by using the **Discover & Upgrade** feature as if you were upgrading the Controller using the Enterprise Console, or run the following command on the Enterprise Console host:

> You must specify and provide the full path to the existing Controller directory.

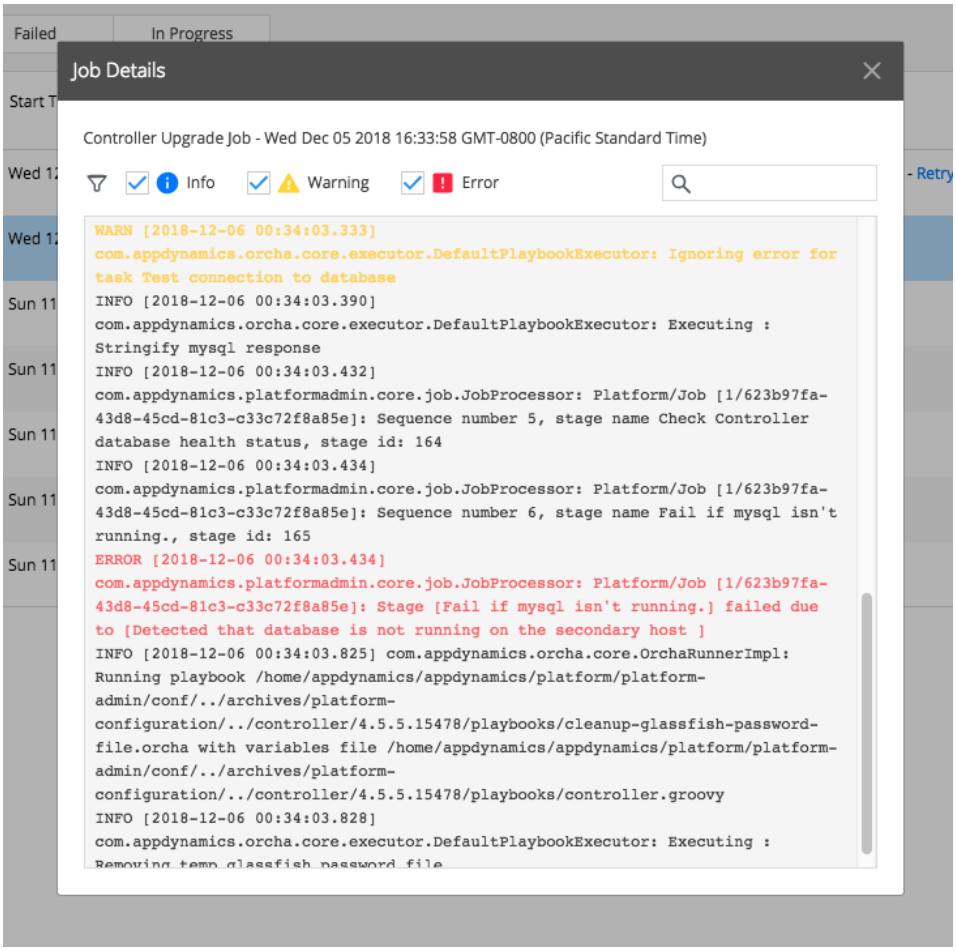
Enterprise Console Jobs

The Enterprise Console saves all jobs that you perform on the application on the Jobs page.

View Job Details

The Jobs page displays all of your jobs by their names, start times, last updated times, and statuses. You can classify the jobs based on their status: Successful, Failed, and In Progress. You can also search through all of your jobs.

You can view additional job details by clicking **View Details** in the Status column of the job.

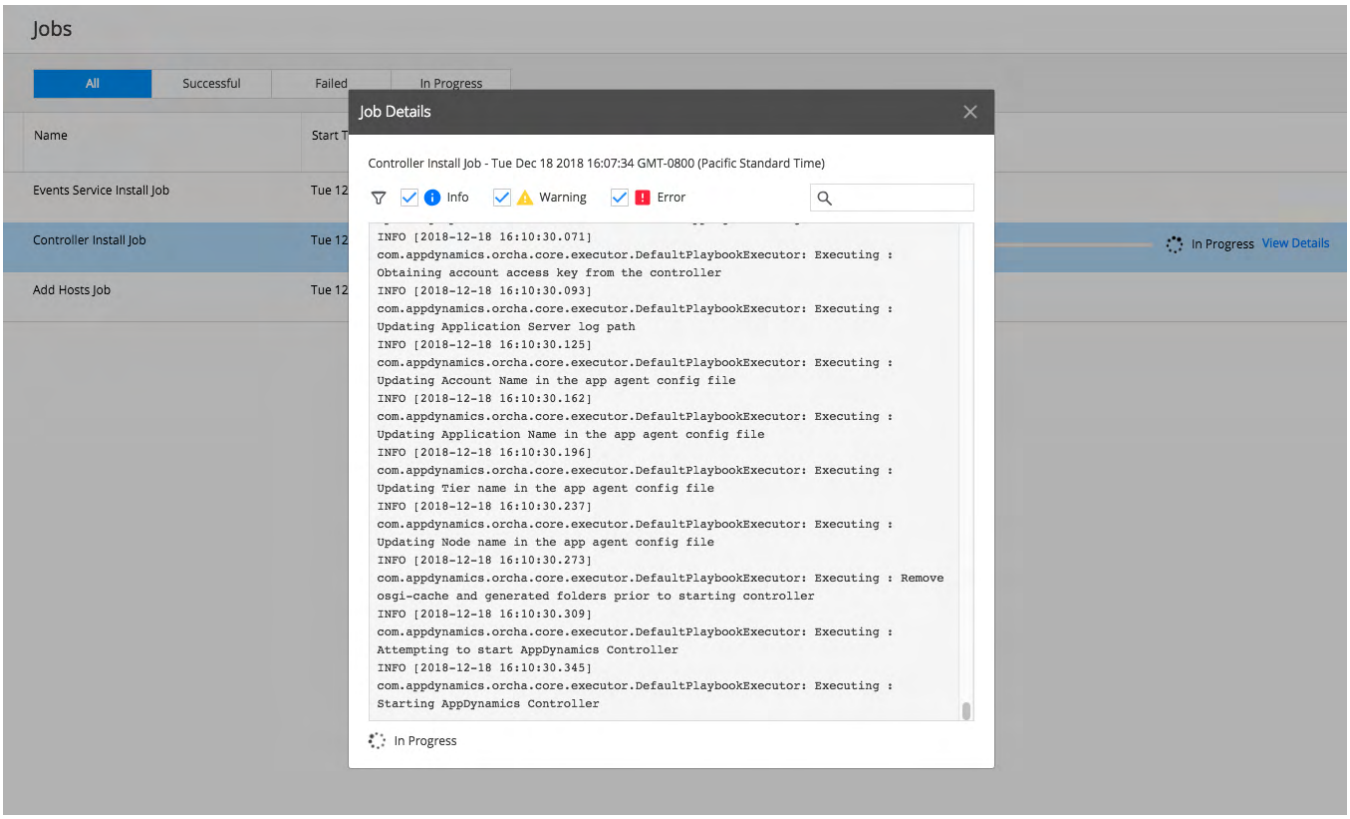


If the job you are viewing failed, then you can see what the error was that caused it to fail, and retry it at its last checkpoint.

> Warnings and errors are color-coded in yellow and red, respectively.

View Job Progress

You can view job details on the UI while the job is in progress by selecting a job and clicking **View Details**. This should help with troubleshooting.


Jobs
All
Successful
Failed
In Progress
Name
Start T
Events Service Install Job
Tue 12
Controller Install Job
Tue 12
In Progress
View Details
Add Hosts Job
Tue 12
Job Details
Controller Install Job - Tue Dec 18 2018 16:07:34 GMT-0800 (Pacific Standard Time)
Info
Warning
Error
INFO [2018-12-18 16:10:30.071]
com.appdynamics.orcha.core.executor.DefaultPlaybookExecutor: Executing :
Obtaining account access key from the controller
INFO [2018-12-18 16:10:30.093]
com.appdynamics.orcha.core.executor.DefaultPlaybookExecutor: Executing :
Updating Application Server log path
INFO [2018-12-18 16:10:30.125]
com.appdynamics.orcha.core.executor.DefaultPlaybookExecutor: Executing :
Updating Account Name in the app agent config file
INFO [2018-12-18 16:10:30.162]
com.appdynamics.orcha.core.executor.DefaultPlaybookExecutor: Executing :
Updating Application Name in the app agent config file
INFO [2018-12-18 16:10:30.196]
com.appdynamics.orcha.core.executor.DefaultPlaybookExecutor: Executing :
Updating Tier name in the app agent config file
INFO [2018-12-18 16:10:30.237]
com.appdynamics.orcha.core.executor.DefaultPlaybookExecutor: Executing :
Updating Node name in the app agent config file
INFO [2018-12-18 16:10:30.273]
com.appdynamics.orcha.core.executor.DefaultPlaybookExecutor: Executing : Remove
osgi-cache and generated folders prior to starting controller
INFO [2018-12-18 16:10:30.309]
com.appdynamics.orcha.core.executor.DefaultPlaybookExecutor: Executing :
Attempting to start AppDynamics Controller
INFO [2018-12-18 16:10:30.345]
com.appdynamics.orcha.core.executor.DefaultPlaybookExecutor: Executing :
Starting AppDynamics Controller
In Progress

Platform Log Files

The platform log files include the following:

- **platform-admin-server.log:** Information about events of the install process such as extraction, preparation, and other post-processing tasks. It is located at `<platform_admin_home>/logs`.
- **server.log:** Information for the embedded Glassfish application server used by the Controller. It is located at `<controller_home>/logs`.
- **audit.log:** Information about Account/User/Group/Role CRUD/User login/logout/SSH connections, and all other operations. This can be used to forward auditable events from the AppDynamics controller into a central log management system or SIEM. It is located at `<controller_home>/logs`, and replicates the Controller Audit Report.
- **database.log:** Information for the MySQL database that is used by the Controller. It is located at `<controller_home>/db/logs`.
- **startAS.log**: Output generated by the underlying Glassfish domain for the Controller.
- **orcha-modules.log**: Information about all the tasks and commands that are run during installation. platform-admin-server.log has high-level information about the task executed. However, `orcha-modules.log` has more information on the specific command and output. It is located at `<platform-dir>/orcha/<version>/orcha-modules/logs/orcha-modules.log`.

Retrieve Log Files

You can use the Enterprise Console to retrieve log files for the Controller and Events Service. On the Controller or Events Service page, expand **More** options and click **Retrieve Log** to start a Retrieve Log job. When the job has successfully completed, it will retrieve and save the Controller or Events Service log files, which include AppServer, Reports Service, and DB logs, as a zip file to `/home/appdynamics/appdynamics/platform/platform-admin` on the Enterprise Console host.

You can also run the following command on the Enterprise Console host:

```
bin/platform-admin.sh submit-job --service <controller or events-service> --job retrieve-log --platform-name
<name_of_the_platform>
```

Manage Log Files

You can manage your log files by setting up log rotation and adjusting the retention periods.

Log Rotation

The application server is preconfigured to rotate the server.log file regularly, based on settings in the domain configuration file. On Windows, there is an additional log file for Glassfish service launcher that needs to be rotated.

For the other log files, such as database.log or audit.log, you may need to set up log rotation to prevent them from consuming excessive disk space. The audit.log file, in particular, may grow quickly because it contains SSH connections to Controller and Event Services hosts that occur every minute. You also need to set up log rotation for an additional log file, `<controller_home>/db/data/slow.log`. This log contains information about slow MySQL queries.

The tool you use to perform the rotation depends on your operating system. On Linux, you can use the `mysql-log-rotate` script. The script is included with the Controller database installation at `<controller_home>/db/support-files`. You need to modify the script for your environment since it is not set up to rotate the database.log file by default. On other systems, you need to create or install a script that performs log rotation and make sure that it get run regularly, for example, by cron or an equivalent task scheduler.

Retention Period

Enterprise Console logs in `platfom-admin/logs` are automatically archived in a .gz format with a date-time stamp after they grow too large. The size of each archived log is around 300 KB. Only the latest seven files are archived in the logs directory.

If you want to retain the archived logs for longer periods of time, you can configure the settings in the Dropwizard configuration file, `PlatformAdminApplication.yml`. Under the logging/appenders/type: file section, you need to specify a larger `archivedFileCount`, as well as `maxFileSize`. See the Drop wizard Configuration Reference for more information.

Change the Default Location or Names of the Controller Logs

1. In a web browser, log in to the Controller's Glassfish administration console, as described in Access the Administration Console.
2. From the left-side navigation tree, expand **Configurations -> server-config** and click **Logger Settings**.
3. Set the new location for server.log by modifying the **Log File** value.
   The default value points to the logs directory located at the root of the Controller home directory.

   ```
   ${com.sun.aas.instanceRoot}/../../../../logs/server.log
   ```

   If you specify a directory that does not exist, it is created when you restart the application server.

4. Change the `database.log` location by opening the `<controller_home>/db/db.cnf` file. You can also change the `db.cnf` file from the Enterprise Console GUI Database Configurations page. Doing so also restarts the database server.

   > The Enterprise Console takes a backup when you modify these configurations.

5. Set the value of the `log-error` property to the new location of the `database.log` file. This directory location must exist before you restart the Controller or you will get start-up errors.
6. From either Linux or Windows, change the log location to the new location:

   Linux File

   ```
   <controller_home>/bin/controller.sh
   nohup ./asadmin start-domain domain1 > $INSTALL_DIR/logs/startAS.log
   ```

   Windows File

   ```
   <controller_home>/bin/controller.bat
   call asadmin.bat start-domain domain1 > %INSTALL_DIR%\logs\startAS.log
   ```

7. Open the `<controller_home>/appserver/glassfish/domains/domain1/config/domain.xml` file, and change the log location specified in the following:

   - log-root attribute of the domain element.
   - file attribute of the log-service element.
   - tx-log-dir attribute of the transaction-service element.

8. Copy if desired any existing logs from the default directory (`<controller_home>/logs`) to the new location.
9. Restart the Controller. See Start or Stop the Controller.
10. Verify that the `database.log` and `server.log` files are being written to the new location and remove the old log files. The `database.log` location is `controller/db/logs/database.log`.

Log Level Granularity

The Controller logs provide information about possible errors in Controller operations. By default, the Controller writes to the log at the `INFO` level. When debugging your Controller deployment, you may need to increase the logging level to generate additional information.

You can set the logging level by Controller component, which include:

- Agents
- Business Transactions (BTS)
- Events
- Incidents
- Information Points (IPS)
- Metrics
- Orchestration
- Rules
- Snapshots

Change Default Logging Level by Component

By default, the Controller generates logs at the INFO level. You can change the level for one or all of the components. This may be needed, for example, when you are debugging your system, and want the Controller to generate more information in the form of logs. On the other hand, you may wish to reduce logging verbosity to minimize the reduce the rate of growth of log files. The following steps describe how to change the default log levels.

1. In a web browser, log in to the Controller's Glassfish administration console, as described in Access the Administration Console.
2. From the left navigation tree, expand **Configurations > server-config** and click **Logger Settings**.
3. Click the **Log Levels** tab.
4. Modify those components that start with "com.appdynamics". By sorting the list by name, you can quickly access the com.appdynamics components. For each component, modify the log level by choosing a new level from the **Log Level** menu. For example, to debug the system, we suggest setting the log level to `FINE`.
5. Click **Save**.

Enterprise Console Back Up and Restore

The Enterprise Console keeps all data pertaining to its managed AppDynamics platform deployment in a MySQL database. To back up an Enterprise Console installation, you use MySQL commands to export and restore data. You will also need to back up the Enterprise Console's secure credential store file.

> Backing up Enterprise Console data is a separate consideration from backing up a Controller. For more information on Controller backups, see Controller Data and Backups.

In the event that your Enterprise Console host fails, follow these steps to ensure that you can recover.

To back up the Enterprise Console:

1. Export the Enterprise Console data:

   ```
   mysqldump -u root -p <password> -h <mysql-hostname> -P <mysql-port> platform_admin > /tmp
   /platform_admin_dump.sql
   ```

   This puts the export file into a named file in the `/tmp` directory. Choose another location, if appropriate.
2. Change to the `mysql/bin` directory of Enterprise Console:

   ```
   cd <platform>/mysql/bin
   ```

3. Import the data into the backup Enterprise Console instance:

   ```
   mysql -u root -p <db_root_user_password> -h <mysql-hostname> -P <mysql-port> platform_admin < /tmp
   /platform_admin_dump.sql
   ```

4. Copy the Enterprise Console's secure credential store (SCS) file `<platform>/platform_admin/.appd.scs` to the same directory on your backup Enterprise Console instance. Make sure you retain the same file permissions for the SCS file (644) on the backup instance of the Enterprise Console.

Upgrade the Enterprise Console

Related pages:

- Upgrade Platform Components
- Install the Enterprise Console

To upgrade the Enterprise Console, you run the installer for the version of the application to which you want to upgrade on the Enterprise Console machine. The installer detects the Enterprise Console installation and upgrades that instance.

About the Upgrade

- You can upgrade across multiple versions at a time; that is, you do not need to run the installer individually for each intermediate version.
- If you upgrade the Enterprise Console to 4.5.x version, it will copy the Controller and Events Service artifacts into the `platform-admin/archives/<service>/<version>` folder. When you upgrade to the next version of the Enterprise Console, the same will be repeated for newer binary versions, e.g. 4.5.1, 4.5.2, 4.6. Note that it is not possible to downgrade the Enterprise Console to any intermediate version.

  If you are upgrading from Platform Admin 4.3.x to the Enterprise Console, make sure you specify a different installation path from the Platform Admin 4.3.x. This is will ensure that any existing Events Service managed by the Platform Admin 4.3.x can be discovered and upgraded using the Enterprise Console.

- You can enable HTTPS for the Enterprise Console during upgrades. See HTTPS Support for the Enterprise Console for more information.

Before Upgrading

- Before you start upgrading the Enterprise Console and your platform, make sure that you are using the correct upgrade order.
- The user upgrading the Enterprise Console should be the same user who installed the Enterprise Console.
- Make sure your Enterprise Console is running before you run the upgrade. This is to validate the Database Root User Password and Platform Admin Database Password. You will need to input the passwords when upgrading through the GUI.
- For first-time upgrades, you need to use the same response file you used for your silent install. See Silent Installation.
- Irrespective of first time or subsequent upgrades, you need to provide the relevant passwords in the response file.

Upgrade the Enterprise Console

The following steps describe how to upgrade the Enterprise Console on Linux and Windows. You use the Enterprise Console installer in GUI mode, console mode, or silent mode to perform the upgrade.

GUI Installation

If there is a newer version of Enterprise Console available, you can begin the upgrade process by downloading and installing the latest version from the AppDynamics download site on top of the existing application.

Before starting, get the Enterprise Console installer version appropriate for your target system. You can get the installer from the AppDynamics download site. When ready, follow these steps to install the Enterprise Console:

1. Navigate to the directory where you downloaded the install file.
2. Run the following commands:

   Linux

   ```
   ./platform-setup-64bit-linux.sh
   ```

   You can run the installer as non-root or root.

   Windows

   ```
   platform-setup-64bit-windows.exe
   ```

   It is recommended that you right-click the exe and select Run as Administrator.

3. After the GUI launches, use it to complete the installation. In Linux, you may also follow the steps in the installation wizard to complete the console installation.

If you install the Enterprise Console on AWS, use the public DNS for the Enterprise Console host name when prompted.

Silent Installation

To use silent mode, pass the response file that the installer generated at first installation to the installer. This response file is at the following location

- `<Enterprise_Console_home_directory>/.install4j/response.varfile`

If you have made any changes to the settings as originally configured by the installer—such as to the connection port numbers, tenancy mode, or data directory—make the same change in the response file before starting the upgrade.

You must also add the existing passwords to the file:

```
platformAdmin.databasePassword= ENTER_PASSWORD
platformAdmin.databaseRootPassword= ENTER_PASSWORD
platformAdmin.adminPassword= ENTER_PASSWORD
```

If you do not, the installer prompts you to add the password.

After Upgrading

After upgrading the Enterprise Console, you must first clear your browser cache before you can successfully log in to the Enterprise Console through the browser.

Uninstall the Enterprise Console

This page describes how to remove the Enterprise Console software and associated files using the uninstaller utility located in the Enterprise Console directory.

Before Starting

Uninstalling the Enterprise Console will not uninstall any of the components it has deployed or managed since the Enterprise Console installer is agnostic of the Controller and other services. Therefore, you do not need to first uninstall any components before uninstalling the Enterprise Console. However, if you accidentally uninstall the Enterprise Console without first uninstalling any components, then you will have to manually manage the remaining platforms and components. Leftover environments can be also be rediscovered using another Enterprise Console application.

Uninstall the Enterprise Console Manually

1. Open a console on the machine where the Enterprise Console is installed:
    - On Linux, open a terminal window and switch to the user who installed the Enterprise Console or to a user with equivalent directory permissions.
    - On Windows, open an elevated command prompt by right-clicking on the Command Prompt icon in the Windows Start Menu and choosing **Run as Administrator**.
2. From the command line, navigate to the Enterprise Console home directory.
3. Execute the uninstaller script to uninstall the Enterprise Console, as follows:
    - On Linux:

      ```
      ./installer/uninstallPlatform
      ```

    - On Windows:

      ```
      run installer/uninstallPlatfom.exe
      ```

    - To uninstall in quiet mode, add the `-q` option. For example:

      ```
      ./installer/uninstallPlatform -q
      ```

      With this option, you do not need to interact with the installer to complete the removal.

      > On the Enterprise console host and any of the hosts that the Enterprise Console manages, `<platform home dir>/jre` and `<platform home dir>/orcha` are not removed.

FAQs for the Enterprise Console

What is the difference between the Enterprise Console installer and the Controller installer?

The Enterprise Console installer only installs the Enterprise Console application. You need to use the Enterprise Console later on to install the Controller as well as other AppDynamics platform components.

The Controller Installer is used to install only the Controller application. From 4.4, AppDynamics does not support the Controller installer anymore. You should use the Enterprise Console instead to install, monitor, upgrade, and configure the Controller.

Where should I install the Enterprise Console application?

The Enterprise Console can be installed on the same host as the Controller. However, you should not install the Enterprise Console on the same host if the Controller is part of an HA pair. If the Enterprise Console and Controller are on the same host and that host becomes unavailable, the Enterprise Console will not be able to failover to the other Controller. For large deployments, it is also recommended that you install the Enterprise Console on a separate host.

Which hosts does the Enterprise Console need SSH access to?

The Controller and Events Service hosts.

How many SSH connections does the Enterprise Console make per minute?

For each single Controller node (HA Controller deployments will have two Controllers), the Enterprise Console will open approximately 10 SSH connections per minute. For each Events Services node (an Events Service cluster may have 3–5 nodes), the Enterprise Console will open 1 SSH connection per minute.

What protocols does the Enterprise Console use to connect to remote hosts?

The Enterprise Console uses Java Secure Channel (JSch) API with the provided key file to access remote hosts. In scenarios where you have an SSH jump server or jump host configuration, you will have to invest in additional provisions for your application to work. Consult your AppDynamics representative in such cases.

Can platforms share the same installation path?

No, platforms cannot share the same installation path or hosts. You must also choose an installation path that does not overlap with where the Enterprise Console is installed.

Can I use Express installation to discover and upgrade existing controllers and events services?

Express installation is a convenient way to create a complete platform on a single host in one step. It does not support discovering and upgrading an existing controller managed outside the Enterprise Console.

Can I make configuration changes outside of Enterprise Console?

It is not recommended that you make changes directly to AppDynamics configuration files, such as in domain.xml or db.conf. If you do make changes directly in configuration files, it is recommended that you make the equivalent change in the Enterprise Console. Note that such changes result in a restart the controller.

Do I need to back up the Enterprise Console database?

Yes, the Enterprise Console database is not automatically backed up. See Controller Data Backup and Restore for information on backing up the Controller and Enterprise Console databases.

Is Enterprise Console supported on Windows?

Yes, it is, but there are several differences between the Linux and Windows deployment. First, the platform components (Controller, MySQL database, Events Service, and EUM Server) all have to be installed on the same machine for Windows. The second difference is that the Enterprise Console on Windows does not support Controller High Availability. Finally, on Windows, you do not have the replication of the MySQL database and cannot install a second node through the Enterprise Console. On Linux, it is recommended to have at least three node clusters for the Events Service.

If you need to scale up the Events Service on Windows, see Install the Events Service on Windows for instructions.

How can I install the Controller without the Enterprise Console?

Starting from AppDynamics version 4.4, you can only install the Controller through the Enterprise Console. For earlier versions, you can still use the Controller installer.

Controller Deployment

Related pages:

- Install the Controller Using the CLI

This page introduces you to the tasks involved with deploying AppDynamics to its operating environment, including host preparation and Controller installation.

The system resources of the machine that hosts the Controller in a live environment must be able to support the expected workload.

Deployment Overview

Installing AppDynamics to a test or evaluation setting typically involves verifying system requirements, preparing the host, and then performing the Controller installation. These topics are described in Prepare the Controller Host and Install the Controller Using the Enterprise Console.

Deploying the Controller to its production operating environment normally introduces additional requirements and considerations. Security, availability, scalability, and performance all play an important role in production deployment planning. The following section lists the tasks related to deploying the Controller.

Deployment Tasks

Depending on your specific requirements and environment, deployment tasks may include:

- Ensure that target systems meet the Controller System Requirements for the Controller's expected workload.
- Implement Controller High Availability to ensure service continuity in the event of a failure of the Controller server.
- Configure the network environment. If deploying the Controller with a reverse proxy, configure passthrough of Controller traffic. Also, note other Network Requirements for the deployment environment.
- Implement security requirements for your environment. If clients will connect to the Controller by HTTPS, install your custom SSL server certificate on the Controller.
- Generate a password management strategy for the built-in system accounts in the Controller and platform.
- Make sure the mail server is properly configured for the Controller in the target environment and define your alerting strategy.
- Devise your backup strategy. A typical backup strategy consists of frequent partial backups with intermittent full backups.
- Plan your configuration maintenance and enhancement strategy. Changes to the configuration should be staged in a non-critical environment, and rolled into the live environment only after thorough testing. The AppDynamics UI and REST API offer the ability to export and import configuration settings from various contexts.
- Deploying App Agents is likely to be an ongoing task, especially in dynamic environments where monitored systems are regularly taken down and new ones brought up. There are two basic strategies for deploying large numbers of App Agents across a managed environment:
    1. Deploy the agents independently of the application inside the application server. This method ensures that re-deployments of the application do not overwrite the agent deployment.
    2. Integrate deployment of AppDynamics agents into the deployment of applications. This more sophisticated approach requires modifying the existing application deployment automation scripts.

  For details, see:
    - Automate Java Agent Deployment
    - Unattended Installation for .NET

Network Requirements

Deploying the Controller often calls for configuration changes to existing network components, such as firewalls or load balancers in the network. If the Controller will reside behind a load balancer or reverse proxy, you need to set up traffic forwarding for the Controller. You may also need to open ports used by AppDynamics on firewalls or any other device through which traffic must traverse.

The following are general considerations for the environment in which you deploy AppDynamics. See AppDynamics Quick Start for other network configuration requirements.

Correlation HTTP Header

AppDynamics adds a custom header to traffic in the monitored environment named `singularityheader`. This header enables AppDynamics to correlate traffic across tiers. It's important to ensure that any load balancer, proxy, or firewall in the network between monitored tiers or between the tiers and the Controller preserves the header added by AppDynamics.

Clock Management

To ensure consistent event time reporting across the AppDynamics deployment, App Agents attempt to synchronize their time with the Controller time.

They do so by retrieving the time from the Controller every five minutes. App Agents then compare the Controller's time to its own local machine's clock time. If the times are different, whether ahead or behind, it applies a time skew based on the difference to the timestamps for the metrics it reports to the Controller.

If, despite the agent's attempt to report metrics based on the Controller time, the Controller receives metrics that are time-stamped ahead of its own time, the Controller rejects the metrics. To avoid this possibility, AppDynamics recommends maintaining clock-time consistency throughout your monitored environment.

Admin Accounts Created at Installation

During the installation process, you need to configure several accounts for the Controller. These include the embedded MySQL database account, a root user account in the Controller, and an administrator in the Controller.

Usernames and passwords should not include the & or ! characters. If a user account needs to access the Controller REST API, additional limitations on the use of special characters in usernames apply. See Create and Manage Tenant Users.

About Controller Tenancy Mode

In most installations, the Controller operates in single-tenant mode. In multi-tenant mode, the Controller UI context is divided into separate accounts. Each account has its own set of users, agents reporting to it, and application monitoring configuration.

You choose the tenancy mode at installation time. You can switch the tenancy mode from single-tenant to multi-tenant mode later. It is not possible to switch from multi-tenant to single-tenant mode.

Having a single tenancy Controller is suitable for most installations. Only very large installations or installations that have very distinct sets of users may require multi-tenancy.

A summary of the differences between the modes follows:

- In multi-tenant mode:
  - You can create multiple accounts (tenants) in the Controller.
  - Each account will have its own set of users and applications.
  - The Controller login page includes an additional field where users need to choose an account to log in to.
  - Essentially, multi-tenant mode allows you to partition users and access application data in a logical, secure way.
- In single-tenant mode:
  - There is only one account (tenant) in the Controller system.
  - All users and applications are part of this single built-in account, so all users have access to all monitored Applications in this mode.
  - The account is not exposed to users in the Controller UI. The account field in the login page is omitted for single-tenant mode.
  - AppDynamics recommends a single-tenant mode for most installations.

For more information, see Multi-Tenant Controller Accounts.

Self-Monitoring the Controller

You can use the system account to self-monitor the Controller. When your system experiences noticeable performance issues with the Controller, you can log in to the system account to access and review the memory utilization trend for the last few hours.

Only the internal Java Agent bundled in the path: `<Controller_home>//appserver/glassfish/domains/domain1/appagent(-javaagent:${com.sun.aas.instanceRoot}/appagent/javaagent.jar)` with the controller, the application is supported for self-monitoring. Using a custom Java Agent from a different path is not supported for self-monitoring.

To self-monitor the Controller using the system account:

1. If you are already logged in to the Controller from your Browser, you must first log out as a non-administrative user of the Controller.
2. Log back into the Controller using the following credentials:
   - Account: `system`
   - User: `root`
   - password: `<root password>`
3. Select **Appdynamics Controller** application.
4. Select **Tiers and Nodes** > **Node1** > **Memory** to review the memory trend for the past 24 hours and locate the time when the performance issues occurred.

Controller System Requirements

Related pages:

- Prepare the Controller Host

This page describes hardware and software requirements for the Controller hosted on private or public cloud to help you prepare for your AppDynamics deployment.

> **Note**
>
> The Controller requirements do not include Enterprise Console and Event Service. You need to prepare memory for each of those components.

About Controller Sizing Information

Every deployment is unique. Factors such as the nature of the application, workload, and the AppDynamics configuration can all affect the resources required for your specific scenario. Be sure to test the performance of your system in a staging environment, so that you can fully understand your requirements before deploying AppDynamics to its live operating environment.

Before installation, it's usually easiest to estimate your deployment size based on the number of nodes. For Java, for example, a node corresponds to a JVM. However, the best indicator of the actual workload on your Controller is provided by the metric ingestion rate.

After initial installation, you should verify your Controller sizing using the metric upload rate. You then need to continue to monitor the Controller for changing workload brought about by changes in the monitored application, its usage patterns, or in the AppDynamics configuration.

General Hardware Requirements

The following general requirements that apply to the machine on which you install the Controller:

- The Controller should run on a dedicated machine. A production Controller *must* run on a dedicated machine. The requirements here assume that no other major processes are running on the machine where the Controller is installed, including no other Controllers.
- The Controller is not supported on machines that use Power Architecture processors, including PowerPC processors. The Controller is supported on amd64 / x86-64 architectures.
- Ensure that the Controller host has approximately 200 MB of free space available in the system temporary directory.
- Disk I/O is a key element to Controller performance, particularly low latency. See Disk I/O Requirements for more information.

Controller Sizing

The following table shows Controller installation profiles by metric ingestion rate and node count. As previously noted, the actual metrics generated by a node can vary greatly depending on the nature of the application on the node and the AppDynamics configuration. Be sure to validate your sizing against the metric ingestion rate before deploying to production.

General Sizing for On-Premises

Profile	Max Metrics /Minute	Max Agents (Approx)	OS	Compute	Storage
Demo	10,000	5	Linux or Windows	2 Cores, 8 GB RAM	50 GB Note: This profile is not supported when installing with Aurora DB.
Small	50,000	50	Linux or Windows	4 Cores, 16 GB RAM	400 GB Note: This profile is not supported when installing with Aurora DB.
Medium	1,000,000	1,500	Linux or Windows	Bare-metal: 8 Cores, 128GB RAM	5 TB SAS SSDs. Hardware-based RAID 5 configuration
			Linux or Windows	VM: 16 vCPUs, 128GB RAM	5 TB SSD-based SAN with 10 Gb/s FCoE
Large	5,000,000	10,000	Linux	Bare-metal: 28 cores, 512 GB RAM	• 2 x 800 GB write-intensive NVMe cards for MySQL redo logs. Software-based (mdadm) RAID 1 configuration. • 20 TB SAS SSDs for main data volume. Hardware-based RAID 5 configuration
Extra Large	**For deployments that exceed the Large profile configuration defined here, contact AppDynamics Professional Services for a thorough viability evaluation of your Controller.**				

Amazon Web Services (AWS) Sizing for On-Premises

AWS Profile with Aurora	Max Metrics /Minute	Max Agents (Approx)	OS	Compute	Instance Size for AWS Aurora Storage	Block Storage (for Controller application files only)*
Medium	1,000,000	1500	Linux	EC2: r4. 2xlarge	db.r4.4xlarge	10 GB GP2 EBS Volume. We recommend using a different volume than the instance's root volume.
Large	5,000,000	10000	Linux	EC2: r4. 8xlarge	db.r4.16xlarge	10 GB GP2 EBS Volume. We recommend using a different volume than the instance's root volume.

* The specified disk space must be available for use by the Controller. Specifications do not include overhead from the operating system, file system, and so on.

Elastic Network Interface (ENI)

The ENI numbers were last updated on Feb 28, 2018.

For AWS, provision an ENI for each Controller host and link the license to the MAC address of the ENI. For more information about ENI, see the AWS documentation at the following link:

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-eni.html.

Additional Sizing Considerations

Note the following additional requirements:

- Large installations are not supported on virtual machines or systems that use network-attached storage.
- The RAM recommendations leave room for operating system processes. However, the recommendations assume that no other memory intensive applications are running on the same machine. While the Enterprise Console can run on the same host as the Controller in small or demo profile Controllers, it is not recommended for medium and larger profiles or for high availability deployments. Refer to the Enterprise Console Requirements if the Enterprise Console is on the same host as the Controller.
- Disk sizing shown in the sizing table represents the approximate space consumption for metrics, about 7 MB for each metric per minute.
- The motherboard should not have more than 2 sockets.
- See Calculating Node Count in .NET Environments for information related to sizing a .NET environment.
- The agent counts do not reflect additional requirements for EUM or Database Visibility. See the following sections for more information.

Disk I/O Requirements

A critical factor in a machine's ability to support the performance requirements of a Controller in a production environment is the machine's disk I/O performance.

There are two requirements related to I/O latency:

- This disk I/O must perform such that the maximum write latency for the Controller's primary storage must not exceed 3 milliseconds while the Controller is under sustained load. AppDynamics cannot provide support for Controller problems resulting from excessive disk latency.
- Self-monitoring must be set up for the Controller. Self-monitoring consists of a SIM agent that measures the latency of data partitions on the Controller host, and the configuration needs to include dashboard and health rule alerts that trigger when the maximum latency exceeds 3 ms. For details on Controller self-monitoring, contact your AppDynamics account representative.

Disk I/O Operations

The AppDynamics Controller performs two types of I/O operations important to Controller performance:

- The MySQL intent log is very sensitive to latency, and MySQL performs writes using varying block sizes.
- MySQL's InnoDB storage engine uses random, asynchronous, 16Kb reads and writes to move database pages between storage and cache. In a properly sized Controller, most reads are satisfied from one of the software caches.

It's important for best performance that the stripe size of the RAID configuration matches the write size. The two write sizes are 16Kb (for the database) and 128Kb (for the logs). You should use the smallest stripe size supported, but no smaller than 16Kb. If using a hardware-based RAID controller, be sure that it supports these stripe sizes. The stripe size can be determined by the number of data disks multiplied by the strip/segment/chunk (the portion of data stored on a single disk).

SAN-based Storage Limitations

While onboard disks typically satisfy I/O requirements, SAN-based storage could be hampered by poor I/O latency performance. In addition, AppDynamics discourages the use of an NFS-mounted filesystem. NFS adds latency and throughput constraints that can negatively affect Controller performance and even lead to data corruption. Similarly, you should avoid iSCSI or other SAN technologies that are subject to quality of service issues from the underlying network.

If you choose to deploy one of these latency-challenged storage technologies on a system that is expected to process 1M metrics/min or greater, a mirrored NVMe configured as a write-back cache for all storage accesses is recommended. Configuring such a device will hide some of the longer latencies that have been seen in these environments.

In all cases, be sure to thoroughly test the deployment with real-world traffic load before putting an AppDynamics Controller into a live environment.

End User Monitoring (EUM) Considerations

End User Monitoring (EUM) typically increases the number of metrics collected. Accordingly, the Small Controller profile is not supported for installations that use EUM. A Medium profile running 20+ high-traffic BRUM/MRUM agents should be sized at a specification closer to a Large profile for EUM.

Specifically, EUM impact metrics as follows:

- Web RUM can increase the number of individual metric data points per minute by up to 22000
- Mobile RUM can increase the number of individual metric data points per minute by as much as 15 to 25K per instrumented application if your applications are heavily accessed. The actual number depends on how many network requests your applications receive.
- Monitoring EUM is memory intensive and may require more space allocated to the metrics cache.

The number of separate EUM metric *names* saved in the Controller database can be larger than the kinds of individual data points saved. For example, a metric name for a metric for iOS 5 might still be in the database even if all your users have migrated away from iOS 5. So the metric name would no longer have an impact on resource utilization, but it would count against the default limit in the Controller for metric names per application. The default limit for *names* is 200,000 for Browser RUM and 100,000 for Mobile RUM.

Database Monitoring Considerations

The following guidelines can help you determine additional disk and RAM required for the machine hosting the Controller that is monitoring the Database Agent. For very large installations, you should work with your AppDynamics representative for additional guidelines.

For on-premises installations, the machine running the Controller and Event Service will require the following additional considerations, for a data retention period of 10 days:

- 1 – 10 collectors: 2 GB RAM, Single CPU
- 10 – 20 collectors: 4 GB RAM, 2 CPUs
- More than 20 collectors: 8 GB RAM, 4 CPUs

Sizing the Controller for the Events Service

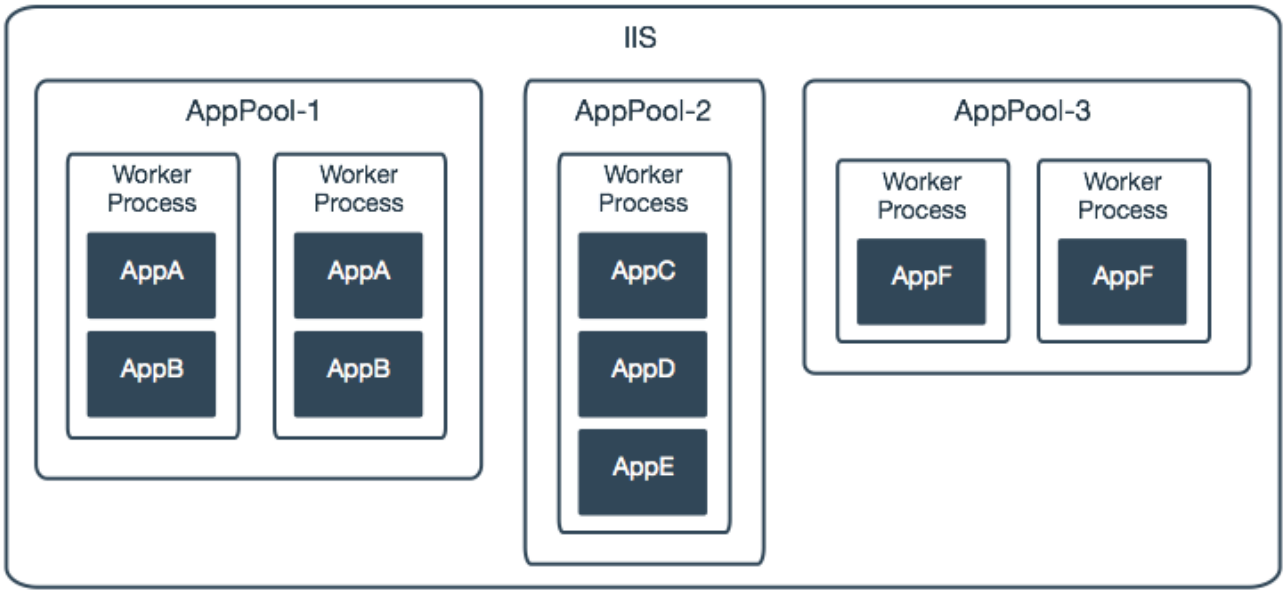
The Events Service is a file-based storage facility used by EUM, Database Monitoring, and Analytics. Database Monitoring uses the Events Service instance embedded in the Controller by default. The disk space required will vary depending upon how active the databases are and how many are being monitored.

For redundancy and optimum performance, the Events Service should run on a separate machine. For details on sizing considerations, see Events Service Requirements.

Calculating Node Count in .NET Environments

The .NET Agent dynamically creates nodes depending on the monitored application's configuration in the IIS server. An IIS server can create multiple instances of each monitored IIS application. For every instance, the .NET Agent creates a node. For example, if an IIS application has five instances, the .NET Agent will create five nodes, one for each instance.

The maximum number of instances of a particular IIS application is determined by the number of worker processes configured for its application pool, as illustrated in the following diagram:



The diagram shows three application pools — AppPool-1, AppPool-2, and AppPool-3 — with the following characteristics:

- AppPool-1 and AppPool-3 can have a maximum of two worker processes (known as a web garden), containing two applications (AppA, AppB) and one application (AppF), respectively.
- AppPool-2 can have one worker process. It has three applications.

To determine the number of nodes, for each AppPool, multiply the number of applications by the maximum number of worker processes. Add those together, as well as a node for the Windows service or standalone application processes.

The example would result in nine AppPool nodes. Adding one for a Windows service would result in a total of ten nodes, calculated as follows:

```
AppPool-1: 2 (applications) * 2 (max number of worker processes)  = 4
AppPool-2: 3 (applications) * 1 (max number of worker processes)  = 3
AppPool-3: 1 (application) * 2 (max number of worker processes)   = 2
Windows Service or standalone application process                 = 1
------
Total:                                                            = 10
```

To find the number of CLRs that will be launched for a particular .NET Application/App Pool:

1. Open the IIS manager and see the number of applications assigned to that AppPool.
2. Check if any AppPools are configured to run as a Web Garden. This would be a multiplier for the number of .NET nodes coming from this AppPool as described above.

Also see: http://technet.microsoft.com/en-us/library/cc725601(v=ws.10).aspx.

Asynchronous Call Monitoring Considerations

The Small profile is not supported for installations with extensive async monitoring. A Medium profile running 40+ agents may need to upgrade to a configuration closer to a Large profile if extensive async monitoring is added.

Specifically, monitoring asynchronous calls increases the number of metrics per minute to a maximum number of 23000 per minute.

Install the Controller Using the CLI

Related pages:

- Controller System Requirements

This page describes how to install the AppDynamics Controller using the CLI. You can also find information on settings you should have after installation. Al ternatively, you can use the Enterprise Console GUI to install the Controller. For information about how to use the Enterprise Console to install the Controller, see Enterprise Console.

The Controller can be installed on the same host as the one on which the Enterprise Console is running or a remote host. Installing on the same host is not recommended, however, particularly for medium and large scale profiles or for high availability deployments.

Install the Controller Using the Command Line

Determine which host you plan to install your Controller on before starting. The host that runs the Enterprise Console is "localhost".

> You may also use the loopback address '127.0.0.1' or the machine's actual hostname.

Complete the following steps carefully if you choose to install the Controller on this shared host rather than on a remote host. Note that all services on Windows machines must be installed on the Enterprise Console host since the Enterprise Console does not support remote operations on Windows.

In the `<Installation directory>/platform-admin` directory, run the following commands to install the Controller:

1. Create a platform:
2. Add the credential.

   For a remote host on Linux machines only:

   ```
   platform-admin.sh add-credential --credential-name <name> --type <ssh> --user-name <username> --ssh-key-
   file <file path to the key file>
   ```

   `<file path to the key file>` is the private key file. The installation process deploys the key to the Controller host.
   For the localhost:
   The localhost does not require credentials. You can, therefore, skip this step, especially for Windows deployments. For more information, see Man age Hosts.
3. Add the host.
   For a remote host on Linux machines only:

   ```
   platform-admin.sh add-hosts --hosts remotehost --credential <credential name>
   ```

   For the localhost:
4. Install the Controller on the host.
   On a remote host for Linux machines only:

   ```
   platform-admin.sh submit-job --service controller --job install --args
   controllerPrimaryHost=<remotehost> controllerAdminUsername=<user1> controllerAdminPassword=<password>
   controllerRootUserPassword=<rootpassword> mysqlRootPassword=<dbrootpassword>
   ```

   On the localhost:

Note that these are the required parameters for installing a Controller with a demo profile size. For information about optional configuration options, run the following command:

Installation Settings

The installation does some basic system checking of your environment as it performs the installation. It notifies you if it encounters conditions that need to be addressed.

Listening ports are configured for the Controller during installation. In GUI and CLI mode, the installation checks to make sure that each port it suggests is available on the system before suggesting it. You only need to edit a default port number if you know it will cause a future conflict or if you have some other specific reason for choosing another port.

Due to browser incompatibilities, AppDynamics recommends using only ASCII characters for usernames, passwords, and account names. The characters " %" and "|" are allowed in the Controller root password.

Verifying Controller Installation

Verify by navigating in a browser to the URL of the Controller UI:

```
http://<application_server_host_name>:<http-listener-port>/controller/rest/serverstatus
```

Log in using the credentials of the initial Controller administrator.

Licensing the Controller

Upon the first login, the Controller UI may prompt you that you need a valid license. You may have acquired the license file from AppDynamics.

To apply a license file manually, copy the license file to the Controller home directory. After moving the license file, allow up to 5 minutes for the license change to take effect.

Troubleshooting the Installation

A log for the installation process is automatically created in the platform-admin/logs/platform-admin-server.log. This file contains information for troubleshooting installation issues.

While installation is in progress, you can find the log file in the `platform-admin/logs/platform-admin-server.log`.

During installation and setup, the Enterprise Console tries to start the Controller. This procedure can take some time. If Controller installation fails, you can troubleshoot and identify the fix and retry from a checkpoint.

To diagnose the Controller, run the following command:

Refer to the Controller diagnostic data in `platform-admin-server.log`.

Controller High Availability

A High Availability (HA) Controller deployment helps you minimize the disruption caused by a server or network failure, administrative downtime, or other interruptions. An HA deployment is made up of two Controllers, one in the role of the primary and the other as the secondary.

The Enterprise Console automates the configuration and administration tasks associated with a highly available deployment on Linux systems. Controller HA pairs are not available on Windows Enterprise Console machines.

Essentially, to set up high availability for Controllers, you are configuring master-master replication between the MySQL instances on the primary and secondary Controllers.

An important operational point to note is that while the databases for both Controllers should be running, both Controller application servers should never be active (i.e., running and accessible by the network) at the same time. Similarly, the traffic distribution policy you configure at the load balancer for the Controller pair should only send traffic to one of the Controllers at a time (i.e., do not use round-robin or similar routing distribution policy at the load balancer).

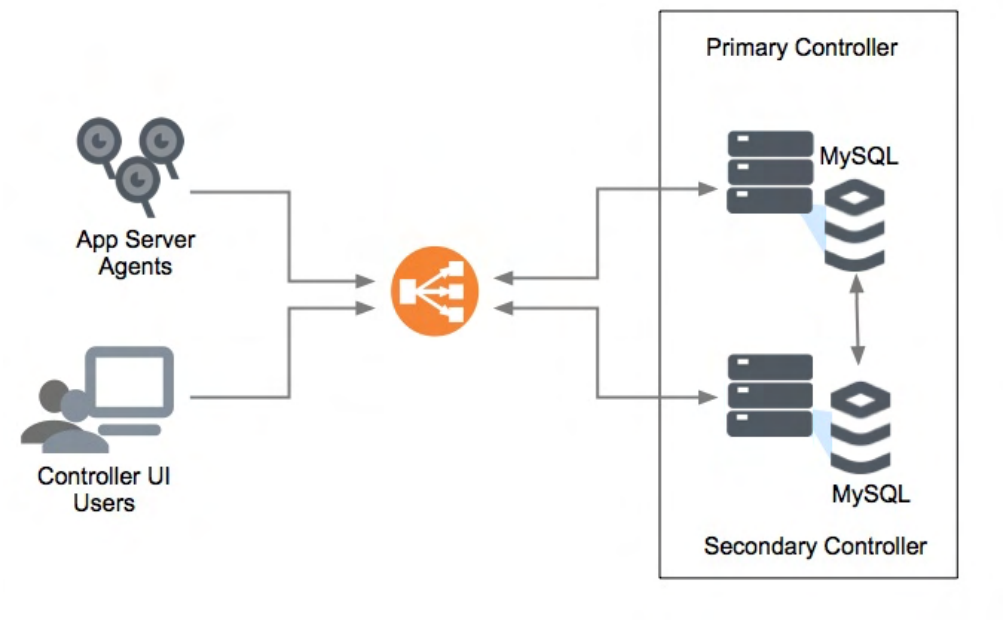
> The Controller supports encrypted database replication.

Overview of High Availability

Deploying Controllers in an HA arrangement provides significant benefits. It allows you to minimize the downtime in the event of a server failure and take the primary Controller down for maintenance with minimal disruption. It fulfills requirements for backing up the Controller data since the secondary maintains an updated copy of the Controller data. The secondary can also be used to perform certain resource-intensive operations that are not advised to be performed on a live Controller, such as performing a cold backup of the data or accessing the database to perform long-running queries, say for troubleshooting or custom reporting purposes.

In HA mode, each Controller has its own MySQL database with a full set of the data generated by the Controller. The primary Controller has the master MySQL database, which replicates data to the secondary Controller's replica MySQL database. HA mode uses a MySQL Master-Master replication type of configuration. The individual machines in the Controller HA pair need to have an equivalent amount of disk space.

The following figure shows the deployment of an HA pair at a high level. In this scenario, the agents connect to the primary Controller through a proxy load balancer. The Controllers in an HA pair must be equivalent versions, and be in the same data center.



In the diagram, the MySQL instances are connected via a dedicated link for purposes of data replication. This is an optional but recommended measure for high volume environments. It should be a high capacity link and ideally a direct connection, without an intervening reverse proxy or firewall. See Load Balancer Requirements and Considerations on Set Up a High Availability Deployment for more information on the deployment environment.

Operating Considerations

In a high availability deployment, it is important that only one Controller is the active Controller at one time. Only the database processes should be running on the secondary so that it can maintain a replicated copy of the primary database.

The Controller app server process on the HA secondary can remain off until needed. Having two active primary Controllers is likely to lead to data inconsistency between the HA pair.

When a failover occurs, the secondary app server must be started or restarted (if it is already running, which clears the cache).

To benefit from increased replication setup speeds, your server will need access to network resources capable of some hundreds of MB per second. By specifying replication setup parallelism, you can radically reduce setup times.

For example, if a single `rsync` is using only one-fifth of the available network capacity, you can achieve maximum throughput for setup by appending `-P r5` to end of the `replicate.sh` command. If this level of network traffic interferes with the ongoing Controller operation, you should monitor and adjust this setting.

- If you are using HA Toolkit version 3.54 and later, append `-P r5` to end of the `replicate.sh` command
- If you are using the HA module with Enterprise Console (version 4.5.17 and later), you must add the `--args numberThreadForRsync=5` to the CLI
- From the Enterprise Console UI, select **Number of parallel rsync threads** for incremental or finalize (depending on what stage you are performing)

Connecting Agents to Controllers in an HA Scenario

Under normal conditions, the App Agents and Machine Agents communicate with the primary Controller. If the primary Controller becomes unavailable, the agents need to communicate with the secondary Controller instead.

AppDynamics recommends that traffic routing be handled by a reverse proxy between the agents and Controllers, as shown in the figure above. This removes the necessity of changing agent configurations in the event of a failover or the delay imposed by using DNS mechanisms to switch the traffic at the agent.

If using a proxy, set the value of the Controller host connection in the agent configuration to the virtual IP or virtual hostname for the Controller at the proxy, as in the following example of the setting for the Java Agent in the `controller-info.xml` file:

```
<controller-host>controller.company.com</controller-host>
```

For the .NET Agent, set the Controller high availability attribute to true in `config.xml`. See .NET Agent Configuration Properties.

If you set up automation for the routing rules at the proxy, the proxy can monitor the Controller at the following address:

```
http://<controller>:<port>/controller/rest/serverstatus
```

An active node returns an HTTP 200 response to GET requests to this URL, with `<available>true</available>` in the response body. A passive node returns 503, Service Unavailable, with a body of `<available>false</available>`.

For more information, see Use a Reverse Proxy.

Prerequisites for High Availability

Before You Begin

Ensure that these requirements are met:

- Controller installation pre-requisites for both servers are met. See Platform Requirements
- Two dedicated machines running Linux. The Linux operating systems can be Fedora-based Linux distributions (such as Red Hat or CentOS) or Debian-based Linux distributions (such as Ubuntu).
- In a Controller HA pair, a load balancer should route traffic from the Controller clients (Controller UI users and App Agents) to the active Controller. Before starting, make sure that a load balancer is available in your environment and that the virtual IP address for the Controller pair is known as presented by the load balancer.
- Open port number 3388 between the machines in an HA pair.
- The login shell must be bash (`/bin/bash`).
- A network link connecting the HA hosts can support a high volume of data. The primary and replica must be in the same data center, and there must be a dedicated network link between the hosts.

> ⚠ IO Latency must be under 3 ms.

- Passwordless ssh has been set up between two Controller hosts. See Set Up the SSH Key.
- SSH keys on each host allow `ssh` and `rsync` operations by the AppDynamics user.
- The `hosts` file (`/etc/hosts`) on both Controller machines should contain entries to support reverse lookups for the other node in the HA pair.
- Because Controller licenses are bound to the network MAC address of the host machine, the HA replica Controller requires an additional HA license. You should request a secondary license for HA purposes in advance.
- While adding high availability hosts as part of the add host operation, you determine and provide the remote user, of which the Controller needs to be installed as. The platform path you specify (while creating the platform) must be writable on the two HA hosts for the remote user specified during add host operation.
- The following packages are installed on both Controller hosts, and the relevant installation commands are provided:

Command	Yum based installer (RH, Centos, Amazon Linux)	Apt based installer (Ubuntu)
`lsof`	`yum install lsof`	`apt-get install lsof`
`ssh`	`yum install openssh-server`	`apt-get install openssh-server`
`awk`	`yum install gawk`	`apt-get install gawk`
`scp`	`yum install openssh-clients`	`apt-get install openssh-client`
`rsync`	`yum install rsync`	`apt-get install rsync`
`curl`	`yum install curl`	`apt-get install curl`
`sed (GNU)`	`yum install sed`	`apt-get install sed`
`openssl`	`yum install openssl`	`apt-get install openssl`
`ps`	`yum install procps`	`apt-get install procps`
`xmllint`	`yum install libxml2-utils`	`apt-get install libxml2-utils`
`timeout/base64/tr`	`yum install coreutils`	`apt-get install coreutils`

Set Up a High Availability Deployment

This page describes how to set up and deploy Controllers as a high availability pair. For installation and upgrade details, see Custom Install and Upgrade an HA Pair.

> ⚠ The Enterprise Console HA deployment works on Linux systems only. Controller HA pairs are not available on Windows machines using Enterprise Console.

About the HA Deployment Using the Enterprise Console

The Enterprise Console automates HA-related setup and administration tasks for the Linux operating system. It does not require `sudo` privileges and can be deployed as a non-root user on Unix operating systems. It works with most flavors of Linux, including Ubuntu and Red Hat/CentOS.

> ⚠ The servers of Controllers in an HA pair must be identical in terms of OS, CPU, RAM, and Disk. See Controller System Requirements.

You can:

- Configure Controllers in a high availability pair arrangement.
- Use the Enterprise Console to monitor the health of the primary Controller, App Server, and database, and failover to the secondary when needed.
- Use scripts that allows you to install the Controllers as a Linux service, and gracefully stop and start service in the event of a machine reboot.
- Failover to a secondary Controller manually (for example, when you need to perform maintenance on the primary).
- Revive a Controller (restore a Controller as an HA secondary after its database is more than seven days behind the primary as a replica).
- Set up a Controller HA pair.

Deploying Controllers as an HA pair ensures that service downtime in the event of a Controller machine failure is minimized. It also facilitates other administrative tasks, such as backing up data. For more background information, including the benefits of HA, see Controller High Availability (HA).

Before Starting

For general guidelines and requirements on how to deploy HA in your environment, see Prerequisites for High Availability. Your environment must meet the prerequisites.

User Privilege Escalation Requirements

After installing a Controller high availability via the Enterprise Console, it is recommended to install MySQL as a Linux service. This is to prevent against MySQL data integrity issues. Installing the Controller and MySQL as a Unix service will ensure that whenever the machine reboots, the service will be shut down and started gracefully.

- `/etc/sudoers.d/appdynamics` contains entries to allow the AppDynamics user to access the `/sbin/service` utility using `sudo` without a password. This mechanism is not available if the AppDynamics user is authenticated by LDAP.
- `/sbin/appdservice` is a `setuid` root program distributed in source form in `<controller_home>/controller-ha/init/appdservice.c`. It is written explicitly to support auditing by security audit systems. The `install-init.sh` script compiles and installs the program. It is executable only by the AppDynamics user and the root user. The script requires a C compiler to be available on the system. You can install a C compiler using the package manager for your operating system. For example, on Yum-based Linux distributions, you can use the following command to install the GNU Compiler, which includes a C compiler:

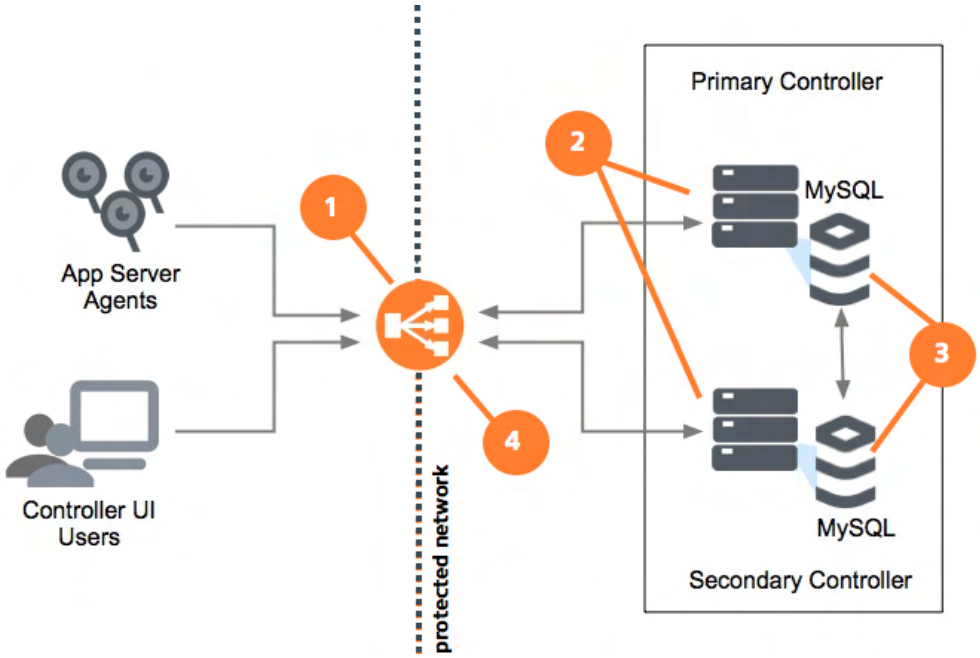
  ```
  sudo yum install gcc
  ```

Load Balancer Requirements and Considerations

Before setting up HA, a reverse proxy or load balancer needs to be available and configured to route traffic to the active Controller in the HA pair. Using a load balancer to route traffic between Controllers (rather than other approaches, such as DNS manipulation) ensures that a failover can occur quickly, without, for example, delays due to DNS caching on the agent machines.

An HA deployment requires the following IP addresses:

- One for the virtual IP of the Controller pair as presented by the load balancer used by clients, such as App Agents, to access the Controller. (Callout 1 in the following diagram.)
- Two IP addresses for each Controller machine, one for the HTTP primary port interface (2) and one for the dedicated link interface between the Controllers (3) on each machine. The dedicated link is recommended but not mandatory.
- If the Controllers will reside within a protected, internal network behind the load balancer, you also need an additional internal virtual IP for the Controller within the internal network (4).



When configuring replication, you specify the external address at which Controller clients, such as app agents and UI users, will address the Controller at the load balancer. The Controllers themselves need to be able to reach this address as well. If the Controller will reside within a protected network relative to the load balancer, preventing them from reaching this address, there needs to be an internal VIP on the protected side that proxies the active Controller from within the network. This is specified using the `-i` parameter.

The load balancer can check the availability of the Controller at the following address:

`http://<controller_host>:<port>/controller/rest/serverstatus`

If the Controller is active, it responds to a GET request at this URL with an HTTP 200 response. The body of the response indicates the status of the Controller in the following manner:

```
<serverstatus vendorid="" version="1">
   ...
   <available>true</available>
   ...
```

Ensure that the load balancer policy you configure for the Controller pair can send traffic to only a single Controller in the pair at a time (i.e., do not use round-robin or similar routing distribution policy at the load balancer). For more information about setting up a load balancer for the Controller, see Use a Reverse Proxy.

Set Up the Controller High Availability Pair

To set up high availability:

Step 1: Configure the Controller High Availability Pair Environment

The following sections provide more information on how to configure a few of the system requirements. They describe how to configure the settings on Red Hat Linux as a sample deployment. Note that the specific steps for configuring these requirements may differ on different systems. Consult your system documentation for specific details.

Host Reverse Lookups

You need to set up a reliable symmetrical reverse host lookup on each machine. To do this, enter the hostnames of the pair into the hosts files (`/etc/hosts`) on each machine. This is preferable over other approaches, such as using reverse DNS, which adds a point of failure.

To enable reverse host lookups, on each host:

1. In `/etc/nsswitch.conf`, enter `files` before `dns` to have the `hosts` file entries take precedence over DNS. For example:
   `hosts: files dns`
2. In `/etc/hosts` file, add an entry for each host in the HA pair. For example:
   `192.168.144.128 host1.domain.com host1`
   `192.168.144.137 host2.domain.com host2`

> To reduce errors, use the correct format of `/etc/hosts` files. If you have both dotted hostnames and short versions, you need to list the dotted hostnames with the most dots first and the other versions subsequently. This should be done consistently for both HA server entries in each of the two `/etc/hosts` files. Note that in the examples provided, the aliases are listed last.

Set Up the SSH Key

SSH must be installed on both hosts in a way that gives the user who runs the Controller passwordless SSH access to the other Controller system in the HA pair. You can accomplish this by generating a key pair on each node, and placing the public key of the other Controller into the authorized keys (authorized_keys) file on each Controller.

The following steps describe how to perform this configuration. The instructions assume an AppDynamics user named `appduser,` and the Controller hostnames are node1, the active primary, and node2, the secondary. Adjust the instructions for your particular environment. Also note that you may not need to perform every step (for example, you may already have the `.ssh` directory and don't need to create a new one).

Although not shown here, some of the steps may prompt you for a password.

On the primary (node1) host:

1. Change to the AppDynamics user, `appduser` in our example:

```
su - appduser
```

2. Create a directory for SSH artifacts (if it doesn't already exist) and set permissions on the directory, as follows:

```
mkdir -p .ssh
chmod 700 .ssh
```

3. Generate the RSA-formatted key:

```
ssh-keygen -t rsa -N "" -f .ssh/id_rsa -m pem
```

4. Secure copy the key to the other Controller:

```
scp .ssh/id_rsa.pub node2:/tmp
```

On the secondary (node2) host:

1. As you did for node1, run these commands:

```
su - appduser
mkdir -p .ssh
chmod 700 .ssh
ssh-keygen -t rsa -N "" -f .ssh/id_rsa -m pem
scp .ssh/id_rsa.pub node1:/tmp
```

2. Add the public key of node1 that you previously copied to the secondary Controller host's authorized keys and set permissions on the authorized keys file:

```
cat /tmp/id_rsa.pub >> .ssh/authorized_keys
chmod 700 ~/.ssh/authorized_keys
```

On the primary (node1) again:

1. Move the secondary's public key to the authorized keys

```
cat /tmp/id_rsa.pub >> ~/.ssh/authorized_keys
chmod 700 ~/.ssh/authorized_keys
```

To test the configuration, enter:

```
ssh -oNumberOfPasswordPrompts=0 <other_node> "echo success"
```

Verify that the `echo` command is successful.

Step 2: Install a Controller High Availability Pair

Once you have set up the environment, you can install a Controller HA pair on the primary machine. To add an HA secondary to an existing standalone Controller deployment, you only need to verify that the user who runs the Controller has write access to the Controller home.

To install a Controller HA pair:

1. Check that you have fulfilled the Enterprise Console prerequisites before starting.
2. Open a browser and navigate to the GUI:

   ```
   http(s)://<hostname>:<port>
   ```

   9191 is the default port.
3. Verify that the credentials and hosts you want to use are added to the AppDynamics platform. For more information, see Administer the Enterprise Console.

   a. On the **Credential** page, add the SSH credentials for the host you want to install the primary Controller on. You can also run the following command on the Enterprise Console host:

   ```
   bin/platform-admin.sh add-credential --credential-name <name> --type <ssh> --user-name <username>
   --ssh-key-file <file path to the key file>
   ```

   > Remember to provide the private key file for the Enterprise Console machine when adding a credential.

   b. On the **Hosts** page, add the host using the credentials from above. You can also run the following command on the Enterprise Console host:

   ```
   bin/platform-admin.sh add-hosts --hosts secondaryhost --credential <credential name>
   ```

4. Navigate to the **Install** homepage and click **Custom Install**.
5. Name the platform:

   a. Enter a Name and the Installation Path for your platform.

   > The Installation Path is an absolute path under which all of the platform components are installed. The same path is used for all hosts added to the platform. Use a path which does not have any existing AppDynamics components installed under it. The path you choose must be writeable, i.e. the user who installed the Enterprise Console should have write permissions to that folder. Also, the same path should be writable on all of the hosts that the Enterprise Console manages.
   >
   > Example path: `<path_to_AppD>/appdynamics/platform`

   Or run the following command on the Enterprise Console host:

   ```
   bin/platform-admin.sh create-platform --name <platform_name> --installation-dir
   <platform_installation_directory>
   ```

6. Add two hosts:

   a. For each of the two hosts, enter their host machine information: Host Name, Username, and Private Key.
   This is the location onto which the Controllers will be installed. For more information about how to add credentials and hosts, see Administer the Enterprise Console.
7. Install Controller:

a. Select **Install**.
b. Select a Profile size for your Controller. See Controller System Requirements for more information on the sizing requirements.
c. Enter the Controller Primary and Secondary hosts.
d. Enter the required Username and Passwords. The default Controller Admin Username is `admin`.

> If you do not install a Controller at this time, you can always do so later by navigating to the **Controller** page in the GUI and clicking Install Controller.

Or run the following command on the Enterprise Console host:

```
bin/platform-admin.sh submit-job --service controller --job install --args
controllerPrimaryHost=<primaryhost> controllerSecondaryHost=<secondaryhost>
controllerAdminUsername=<user1> controllerAdminPassword=<password>
controllerRootUserPassword=<rootpassword> mysqlRootPassword=<dbrootpassword>
```

8. Click **Install**.

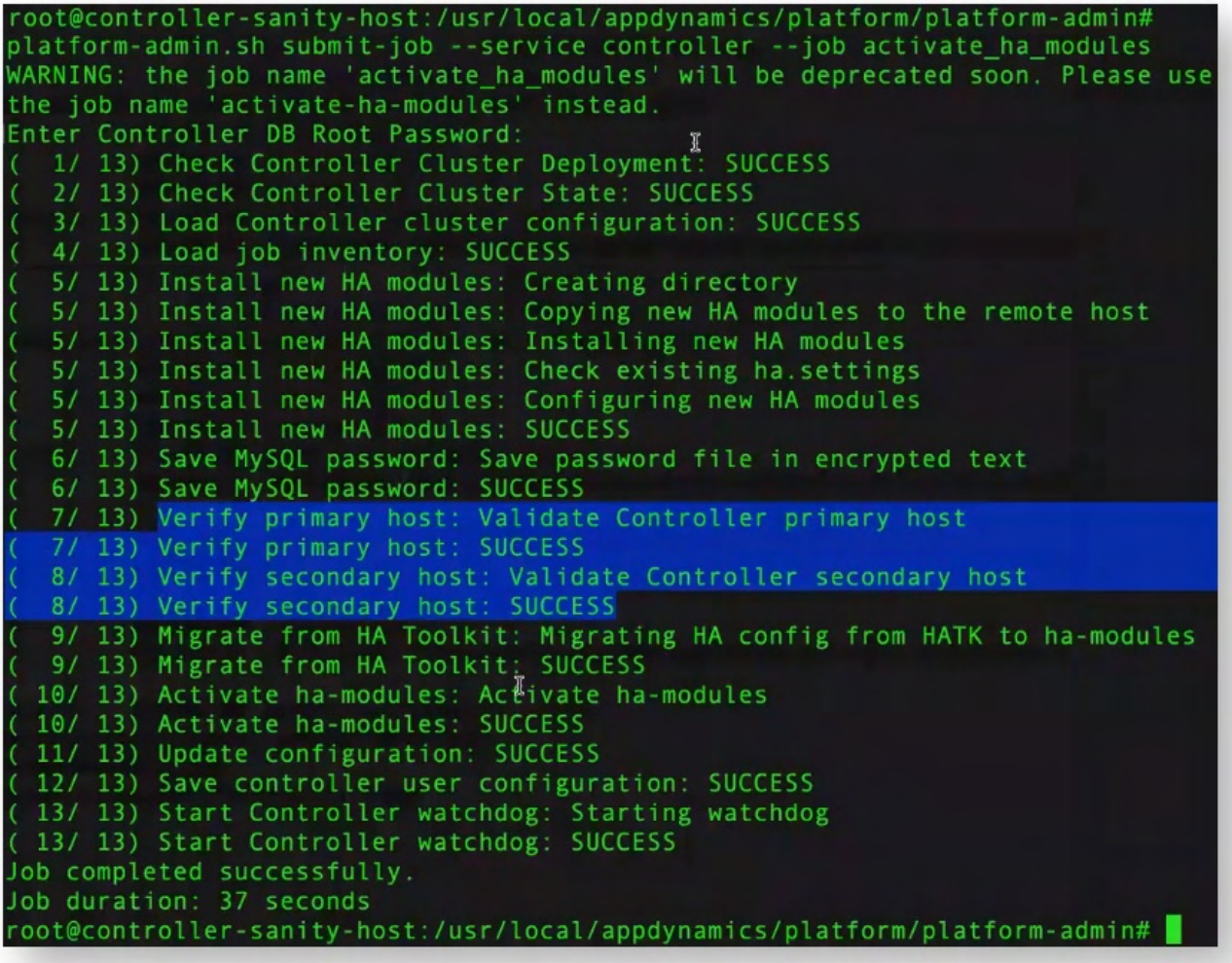
Step 3: Activate the Controller High Availability Pair

This step ensures that the Enterprise Console is no longer in the critical path of the HA Controller failover process.

Open a command shell on the Enterprise Console host and enter:

```
platform-admin.sh submit-job --service controller  --job activate-ha-modules
```

Your output should be similar to the following:



Step 4: Install as a Service

The Enterprise Console does not install the Controller as a service on Linux because it requires root user or `sudo` privileges. However, the Enterprise Console copies the `init` scripts on the Controller hosts in `<controller_home>/controller-ha`. To complete the installation, manually run the following:

1. Change directories to `<controller_home>/controller-ha`.
2. As the user who owns the Controller folder, run:

   ```
   set_mysql_password_file.sh -p <db root password> -s <secondary host>
   ```

   `-s` copies the file to the secondary host. Enter the MySQL database root user password in the command.
3. Change directories to `<controller_home>/controller-ha/init`.
4. Run `install-init.sh` as root user with one of the following options to select how to elevate the user privilege:

   - `-c` #use setuid c wrapper
   - `-s` #use sudo
   - `-p` #use prune wrapper
   - `-x` #use user privilege wrapper

You must run this script on both Controller HA pair servers. If you need to uninstall the service later, run the `uninstall-init.sh` script.

The status and progress of the deployment's various components are written to the logs.

Convert a Standalone Controller to a Controller High Availability Pair

You can convert a standalone Controller to a Controller HA pair through the Enterprise Console GUI. Ensure that you have completed the prerequisites in the above Configure the Controller High Availability Pair Environment section that requires both the primary and secondary hosts to talk to each other via passwordless SSH.

Additionally, you can use incremental replication to add a secondary Controller. See Initiate Controller Database Incremental Replication for more information.

If you are starting from a fresh installation, you will need to first create a platform, then add two credentials and hosts for your HA pair.

To convert a standalone Controller to a Controller HA pair:

1. Open the Enterprise Console GUI.
2. Verify that the credentials and hosts you want to use are added to the AppDynamics platform. For more information, see Administer the Enterprise Console.

    a. On the **Credential** page, add the SSH credentials for the host you want to install the secondary Controller on. You can also run the following command on the Enterprise Console host:

    ```
    bin/platform-admin.sh add-credential --credential-name <name> --type <ssh> --user-name <username>
    --ssh-key-file <file path to the key file>
    ```

    > Remember to provide the private key file for the Enterprise Console machine when adding a credential.

    b. On the **Hosts** page, add the host. You can also run the following command on the Enterprise Console host:

    ```
    bin/platform-admin.sh add-hosts --hosts secondaryhost --credential <credential name>
    ```

    The Enterprise Console uses this host for the HA pair.
3. On the **Controller** page, click **Add Secondary Controller**, and complete the wizard:

    a. Select the Controller Secondary Host that you added for the secondary Controller.
    b. Optional: Enter the External URL. This is the external load balancer URL, which should reflect this format: `http(s)://<external.vip>:<port>`
    c. Enter the DB Root Password, and re-enter it for confirmation.

    > Ensure to provide the same passwords during the secondary server installation as those that you provided for the primary server.

4. Select **Submit**.

Your HA pair will automatically set up, each with their own MySQL node.

Manage a High Availability Deployment

This page describes how to manage and troubleshoot Controllers as a high availability pair.

Set Up Monitoring for the HA Pair

You can set up monitoring for your HA pair by installing another Controller to act as the monitoring Controller.

1. If you do not already have an HA pair, set one up.
2. Install the monitoring Controller on the Enterprise Console host in a new platform by selecting Custom Install:
    a. Create a platform (e.g.: Controller Monitor Platform).

    > ⚠ This platform should not be used for installing any other services.

    b. Install a Controller.
    c. Make sure to unselect the Install Events Service option before clicking **Install**.
3. Complete the monitoring setup by installing and configuring the App Agents and Machine Agents on your HA pair:
    - Set Up App Agents for Monitoring
    - Install and Set Up Machine Agents for Monitoring

Set Up App Agents for Monitoring

You can set up App Agents, which are automatically installed on the Controller hosts by the Enterprise Console, on both Controllers of an HA pair to report to the monitoring Controller. This can be done by updating the JVM options of your HA pair platform. To set up your App Agents using the Enterprise Console, perform the following steps:

1. SSH into the primary Controller box and update the primary Controller App Agent's `controller-info.xml` by running the following commands:

    ```
    cd <controller-install-dir>/appserver/glassfish/domains/domain1/appagent
    cp conf/controller-info.xml ver<version#>/conf/
    ```

2. Repeat step 1 for the secondary Controller.
3. In the Enterprise Console UI, select your HA pair platform, and navigate to the JVM Options section by clicking **Configurations > Controller Settings** > **Appserver Configurations**.
4. Make the following updates to JVM Options:

    a. Update the `appdynamics.controller.hostName` to the monitoring Controller's IP.
    b. Add the following required `jvm-options` for monitoring:

    ```
    -Dappdynamics.agent.applicationName=<app_name>, -Dappdynamics.agent.tierName=<tier_name>,
    -Dappdynamics.agent.nodeName=<node_name>, -Dappdynamics.agent.accountName=<account_name>,
    -Dappdynamics.agent.accountAccessKey=<access_key>
    ```

    > ⓘ You can get your access key from the Controller UI: navigate to **Settings**> **License** > **Account**. Then click to show your access key. Note, when you log in to the Controller, use the account specified in `appdynamics.agent.accountName`.

5. Scroll down the page and click **Save**. The job will apply these properties and restart both the primary and secondary Controllers.
6. In the Enterprise Console UI, select your Controller Monitor Platform, and navigate to the Controller page.
7. Click on **External URL** on the widget to open the monitoring Controller's UI.
8. Log in to the Controller. You should be able to see the monitoring application for both the primary and secondary Controllers.

Install and Set Up Machine Agents for Monitoring

You must install Machine Agents on both Controllers of an HA pair to report to the monitoring Controller. These agents are Java programs that collect hardware metrics. To install and set up your machine agents, perform the following steps:

1. Install the Machine Agent on the primary Controller box. Do not start the agent.
2. Repeat step 1 for the secondary Controller.
3. Configure the Machine Agent properties for both Machine Agents by editing the `controller-info-xml` file located in the `<machine_agent_home>/conf` directory.
    a. Update the `<controller-host>` to the monitoring Controller's IP.
    b. Model the rest of your `controller-info-xml` file after the Example Configuration.
4. Start both Machine Agents.
5. In the Enterprise Console UI, select your Controller Monitor Platform, and navigate to the Controller page.
6. Click on **External URL** on the widget to open the monitoring Controller's UI.
7. Log in to the Controller. You should be able to see the monitoring application for both the primary and secondary Controllers.

Bouncing the Primary Controller Without Triggering Failover

The Enterprise Console does not allow you to stop and start the primary Controller without initiating failover. Therefore, to work around this, you will need to perform the following steps:

1. Log in to the Enterprise Console and navigate to the Appserver Configurations page by clicking through Configurations, followed by Controller Settings.
2. Deselect **Enable Auto Failover** and click **Save**.
3. SSH to the Controller machine where the Controller is installed.
4. Run the following commands on the Enterprise Console host:

   ```
   bin/platform-admin.sh stop-controller-appserver
   bin/platform-admin.sh start-controller-appserver
   ```

   This will bounce the primary Controller in HA mode.
5. Re-enable auto failover on the Enterprise Console Appserver Configurations page.

Starting and Stopping the Controller

The Enterprise Console does not allow you to shut down the primary Controller. However, you can restart the secondary Controller via the start and stop Controller commands.

To start or stop the Controller manually, use the following commands:

- To start:

  ```
  bin/platform-admin.sh start-controller-appserver --with-db
  ```

- To stop:

  ```
  bin/platform-admin.sh stop-controller-appserver --with-db
  ```

Automatic Failover

The Enterprise Console monitors the health of the primary Appserver and database. If the Appserver or database is unresponsive, the Enterprise Console will by default wait for five minutes before initiating a failover. This interval can be configured by updating the default value in the Domain Protocol text field on the Appserver Configurations page under Controller settings.

You can also disable or enable automatic failover through the CLI.

> Version 4.5.14 and above of the Enterprise Console comes with the High Availability (HA) modulewhich utilizes the Controller Watchdog for auto-failover. If you want to enable or disable the auto-failover, then the `watchdog` script needs to be running or stopped.

To disable and enable the Controller Watchdog with CLI using the following commands:

- To stop the Controller Watchdog:

```
./platform-admin.sh submit-job --job stop-controller-watchdog --service controller
```

- To start the Controller Watchdog:

```
./platform-admin.sh submit-job --job start-controller-watchdog --service controller
```

Performing a Manual Failover and Failback

To failover from the primary to the secondary manually, click the **HA Failover** option on the **Controller** page of the Enterprise Console or run the following command on the Enterprise Console host:

```
bin/platform-admin.sh submit-job --service controller --job ha-failover --platform-name <name_of_the_platform>
```

This changes the Appserver on the secondary as primary and database on the secondary as the replication master. It also changes the old primary to secondary.

The process for performing a failback to the old primary is the same as failing over to the secondary. Simply run the following command on the Enterprise Console host:

```
bin/platform-admin.sh submit-job --service controller --job ha-failover --platform-name <name_of_the_platform>
```

Note that if it has been down for more than seven days, you need to revive the database, as described in the following section.

Initiate Controller Database Incremental Replication

Re-enable Broken Replication

Incremental replication, replication via rsync when the primary database is up, is required in cases where the database replication on the secondary Controller is lagging behind the primary Controller by more than three days. This type of replication allows the primary Controller to keep operating while the disk contents are copied to the secondary node.

To initiate incremental replication:

1. Run the following command on the Enterprise Console host:

   ```
   bin/platform-admin.sh submit-job --service controller --job incremental-replication
   ```

   This launches a continuously running background job.
2. Make sure replication occurs four or more times, by checking `mysqlDir/incremental_sync.status` on the primary database host.

   Sample rsync status file output:

   ```
   rsync started at Mon Mar  5 11:49:56 PST 2018
   rsync completed at Mon Mar  5 11:50:56 PST 2018
   rsync started at Mon Mar  5 11:51:01 PST 2018
   rsync completed at Mon Mar  5 11:51:11 PST 2018
   ```

   If replication fails, go to the secondary host and stop all rsync and ha-replicate.sh processes. Then try running the incremental-replication job again.
3. Finalize the job by running the following command on the Enterprise Console host:

   ```
   bin/platform-admin.sh submit-job --service controller --job finalize-replication
   ```

   This stops the incremental replication loop. The command will restart the primary Controller, resulting in downtime.
4. Make sure replication is working by checking that there is no significant gap between the primary and secondary Controllers. You can run the following command on the Enterprise Console host to check the replication status:

   ```
   bin/platform-admin.sh show-service-status --platform-name <platform_name> --service controller
   ```

   It may take a few minutes for the secondary status to catch up.

Add a Secondary Controller Using Incremental Replication

You can convert a single Controller with a large amount of data to an HA pair by using incremental replication. This way, you can rsync most of the Controller's data while the Controller is still running, limiting the downtime of adding a secondary Controller.

To add a secondary Controller using incremental replication:

1. Start the incremental replication, giving host and rsync parameters:

   ```
   bin/platform-admin.sh submit-job --service controller --job incremental-replication --args
   controllerSecondaryHost=1.1.1.1 rsyncThrottle=40000 rsyncCompress=true
   ```

This launches a continuously running background job.

2. Make sure replication occurs four or more times, by checking `<controller_home>/controller-ha/tmp/replication.status` on the primary database host.
   Sample rsync status file output:

```
rsync started at Mon Mar  5 11:49:56 PST 2018
rsync completed at Mon Mar  5 11:50:56 PST 2018
rsync started at Mon Mar  5 11:51:01 PST 2018
rsync completed at Mon Mar  5 11:51:11 PST 2018
```

   > If replication fails, go to the secondary host and stop all rsync and ha-replicate.sh processes. Then try running the incremental-replication job again.

3. Run the add secondary job. The Enterprise Console will perform a final rsync and add the secondary.

```
bin/platform-admin.sh submit-job --service controller --job add-secondary --args
controllerSecondaryHost=secondary mysqlRootPassword=‘password'
```

   The command will restart the primary Controller, resulting in downtime.

   > Until you trigger the add-secondary command, the secondary Controller is not added to the Enterprise Console platform. Therefore, the Enterprise Console will not be able to perform any other operations on the secondary Controller.

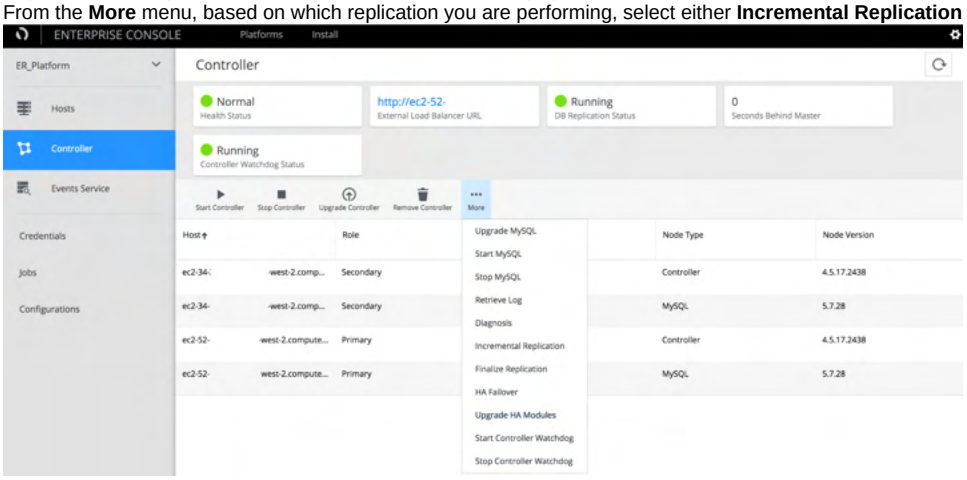
If you need to stop replication, you can run the following command:

```
bin/platform-admin.sh submit-job --service controller --job stop-incremental-replication
```

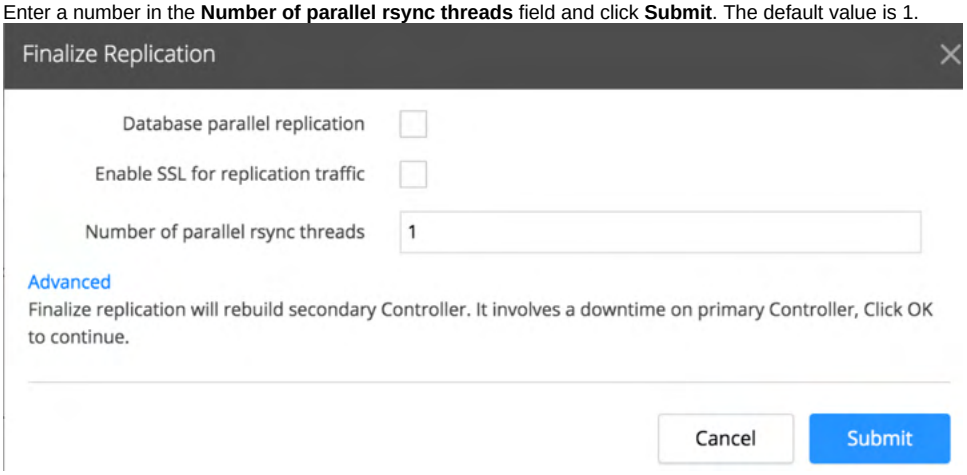
Set Replication Factors for Rsync Threads

Using the Enterprise Console UI or the CLI, you can set the number of parallel rsync threads as a job parameter when you perform incremental or finalize replication.

- From the Enterprise Console UI:
  1. Log in to the Enterprise Console and access the Controller page.
  2. From the **More** menu, based on which replication you are performing, select either **Incremental Replication** or **Finalize Replication**.



3. Enter a number in the **Number of parallel rsync threads** field and click **Submit**. The default value is 1.



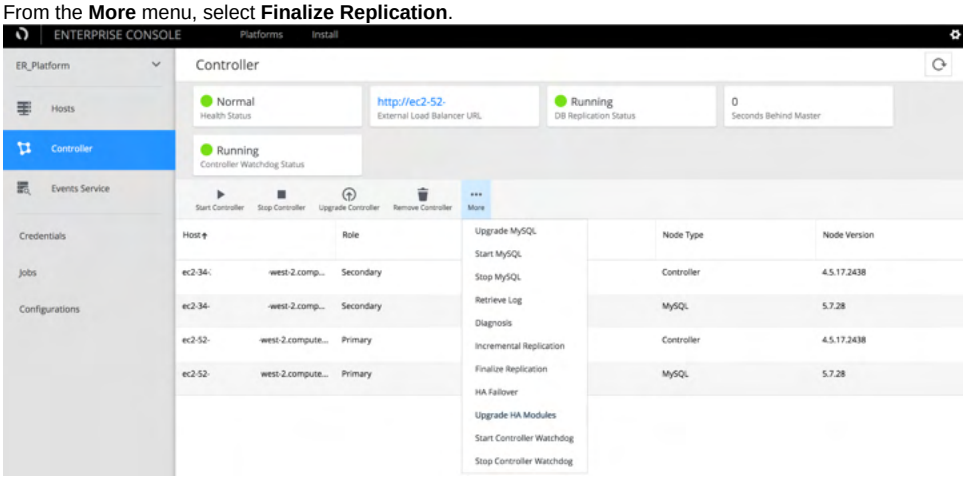
- From the CLI, based on which replication you are performing, run either of the following commands from the Enterprise Console host and set the n umberThreadForRsync argument.

```
bin/platform-admin.sh submit-job --job incremental-replication --args numberThreadForRsync=<number> bin
/platform-admin.sh submit-job --job finalize-replication --args numberThreadForRsync=<number>
```

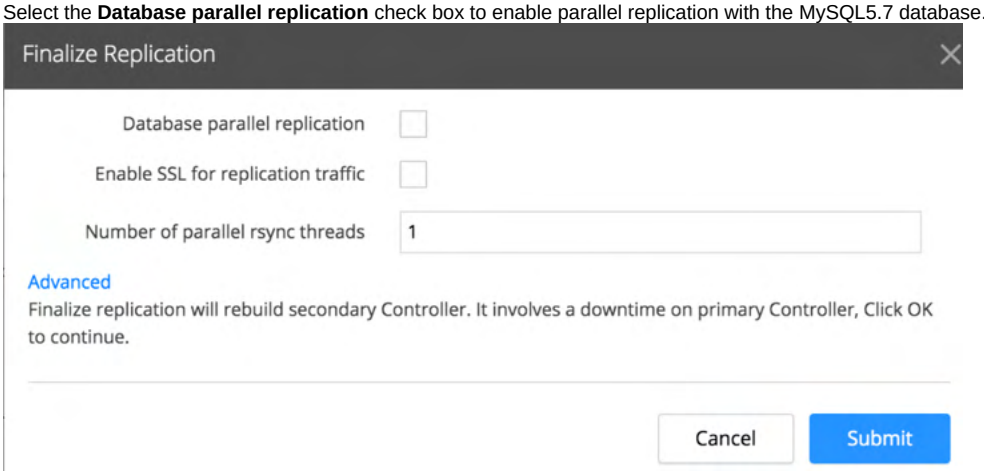
Enable MySQL5.7 Parallel Replication

Using the Enterprise Console UI or the CLI, you can enable MySQL5.7 parallel replication when you perform finalize replication.

- From the Enterprise Console UI:
  1. Log in to the Enterprise Console and access the Controller page.
  2. From the **More** menu, select **Finalize Replication**.



  3. Select the **Database parallel replication** check box to enable parallel replication with the MySQL5.7 database.



  4. Click **Submit**.
- From the CLI, run the following command from the Enterprise Console host to enable MySQL5.7 parallel replication. The default value is true.

```
bin/platform-admin.sh submit-job --job finalize-replication --args dbParallelReplication=true
```

Troubleshooting the Incremental Replication Status

If your first incremental replication run is taking longer than usual, you can refer to the status file, `<controller_home>/incremental_sync.status`, to review a detailed list of files that are being rsynced. You can find the file in the primary Controller host under the Platform folder: `mysqlDir /<controller_home>/incremental_sync.status`.

Re-enable Controller Database Replication

The Controller databases can be synchronized using the replicate script if they have been out of sync for more than seven days. Synchronizing a database that is more than seven days behind a master is considered reviving a Controller database. Reviving a database involves essentially the same procedure as adding a new secondary Controller to an existing production Controller, as described in Set Up the Secondary Controller and Initiate Replication. You can also follow these steps in the case of an HA failover that failed at replication.

To re-enable replication or revive a Controller database:

1. On the Controller page, click **Remove Controller**, or run the following command on the Enterprise Console host:

   ```
   bin/platform-admin.sh submit-job --job remove --service controller
   ```

2. Enter the database root credentials.
3. Check **Remove Binaries**, or run the following command on the Enterprise Console host:

   ```
   bin/platform-admin.sh submit-job --job remove --service controller --args removeBinaries=true
   ```

4. Uncheck **Remove Controller Cluster**. If it is already unchecked, remove the secondary server.
5. Click **Submit**.
6. Add a secondary controller from the Controller page, or run the following command on the Enterprise Console host:

   ```
   bin/platform-admin.sh submit-job --service controller --job add-secondary --args
   controllerSecondaryHost=secondary mysqlRootPassword='password'
   ```

   The command will restart the primary Controller, resulting in downtime.

The Enterprise Console will onboard the secondary Controller and re-enable replication.

Backing Up and Restoring Controller Data in an HA Pair

An HA deployment makes backing up Controller data relatively straightforward since the secondary Controller offers a complete set of production data on which you can perform a cold backup without disrupting the primary Controller service.

After setting up HA, perform a back up by stopping the Controller on the Enterprise Console and performing a file-level copy of the AppDynamics home directory (i.e., a cold backup). When finished, simply restart the Controller from the Enterprise Console. The secondary will then catch up its data to the primary.

When restoring the database from a back up in an HA or standalone environment, you should check that the primary and secondary servers ha.type and ha.mode are set properly to active and passive, respectively.

Updating the Configuration in an HA Pair

The Enterprise Console will copy any file-level configuration customizations made on the primary controller to the secondary controller, such as changes in domain.xml and `db.cnf`.

Over time, if you need to make modifications to the Controller configuration, always do those changes in the Enterprise Console on the Controller Settings page under Configurations. These changes will be preserved during upgrades. Any changes made outside the Enterprise Console will not be preserved after upgrade.

Troubleshooting HA

Controller Diagnostic Data

The Enterprise Console writes log messages pertaining to HA to the `platform-admin-server.log` on the Enterprise Console host.

To diagnose the Controller, run the following command:

```
bin/platform-admin.sh submit-job --platform-name <name_of_the_platform> --job diagnosis --service controller
```

Refer to the Controller diagnostic data in the `platform-admin-server.log`.

Sample Controller diagnostic data

Linux

```
Controller diagnostic data:
123.45.0.1:
controller_database: running
controller_appserver: running
reports_service: running
operating_system: Linux
controller_version: 004-004-001-000
controller_performance_profile: small
controller_ha_type: primary
controller_appserver_mode: active
controller_metric_data_per_min: N/A
slave_io_state: Waiting for master to send event
seconds_behind_master: 0
master_server_id: 567.
master_host: controller-secondary
master_ssl_allowed: No

123.45.0.2:
controller_database: running
controller_appserver: not running
reports_service: running
operating_system: Linux
controller_version: 004-004-001-000
controller_performance_profile: small
controller_ha_type: secondary
controller_appserver_mode: passive
```

Invalid HA Controller Roles

If your HA Controller roles in the Controller databases are incorrect, the Enterprise Console will prevent discover and upgrade jobs. An invalid HA Controller state is when both of your Controller role types are identical, such as in a primary/primary or secondary/secondary case.

To fix this issue:

1. Identify which server is the primary.
    a. Log in to one of the Controller databases by running the following command in the Controller installation directory:

    ```
    bin/controller.sh login-db
    ```

    b. Run the following command:

    ```
    select * from global_configuration_local where name='ha.controller.type';
    ```

2. Ensure that `ha.controller.type` is set correctly in the database.
    a. Log in to the Controller database you would like to change by running the following command in the Controller installation directory:

    ```
    bin/controller.sh login-db
    ```

    b. Run the following commands to set the database to the primary or secondary:

3. Restart the database for the change to take effect on the Appserver:

```
bin/platform-admin.sh stop-controller-appserver --with-db
bin/platform-admin.sh start-controller-appserver --with-db
```

If the secondary Appserver is already in a shutdown state, then there is no need to restart the database.

4. Verify the replication is healthy:

```
show slave status\G
```

`Slave_IO_Running` and `Slave_SQL_Running` should show `Yes`.

You may now retry the discover and upgrade job.

Failover Prevention

If failover is prevented on your Controller HA configuration, it may be due to one of two scenarios:

- The secondary database is down. Failover cannot occur when the secondary database is not running.
  To fix this issue:
  1. Restart the secondary database by running the following command on the secondary host:

     ```
     bin/controller.sh start-db
     ```

  If this does not enable failover, then it may be due to the second scenario.

- Database replication is not healthy. Failover is not allowed when the database replication is not healthy.
  There are various reasons why this may be the case. Please work closely with your AppDynamics account representative to correct the issue.

Migrate to the New HA Module Using Enterprise Console

As part of the new Controller HA Module in the Enterprise Console, Enterprise Console no longer manages the Controller failover. Instead, the new HA Module installs the Controller watchdog on the Controller hosts and the Controller hosts are now responsible for performing a failover. The new HA Module is packaged in Enterprise Console and allows you to migrate seamlessly from older HA implementations, such as the HA Toolkit (HATK), to this new HA Module. The new HA Module is included with the latest version of Enterprise Console, and is installed when you install or upgrade your Controller. However, it is not activated until you migrate an HA pair.

Who Should Use the HA Module?

- If you are a new user just starting to implement AppDynamics HA, then you should use the HA Module (this is the default option).
- If you are an existing user and you prefer to use the Enterprise Console UI integration instead of the command line (CLI) and HATK, then you should use the HA Module.
- If you are an existing user and you do not want to change existing DevOps, conduct additional training, or would like to continue using HATK features, then use HATK.

If your Controller version is earlier than version 4.5.13, then you must use the HA Toolkit (HATK) instead of the HA Module to install and configure High Availability.

Before You Begin

In addition the Prerequisites for High Availability, ensure the following requirements are met:

- Both servers have the same Linux username and groupname.
- Both servers have the same AppDynamics directory structure, same AppDynamics pathname on both servers without using symbolic links.

Migrate a Controller High Availability Pair

This section describes the Enterprise Console and Controller HA Pair upgrade processes required for two different scenarios:

- Scenario One: Deployment currently using both Enterprise Console and HA Toolkit
- Scenario Two: Deployment where Enterprise Console alone manages the HA Pair

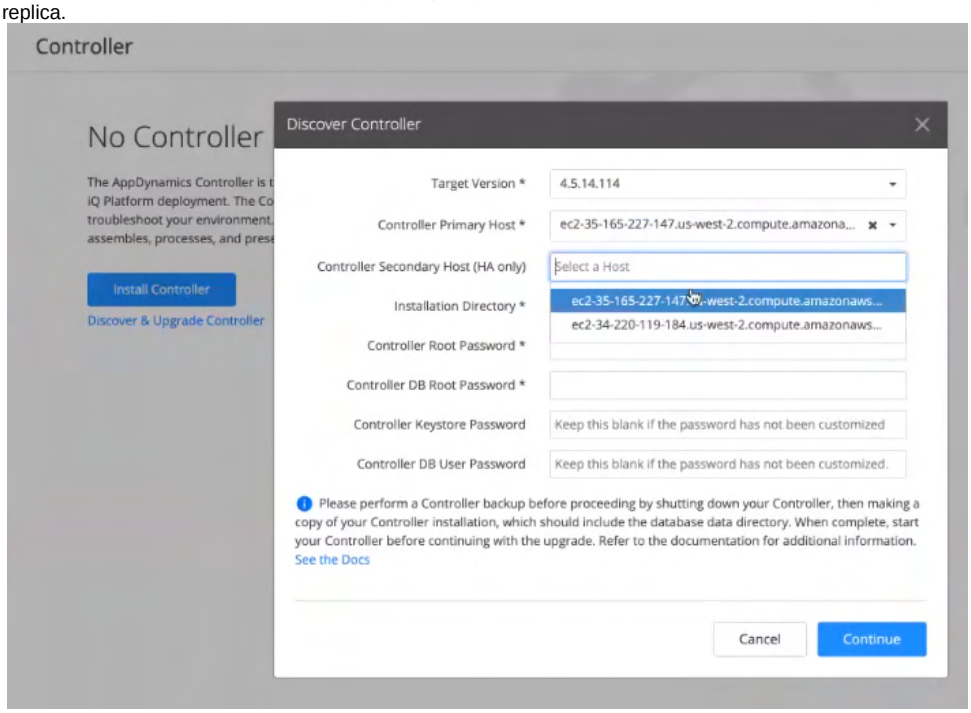
Scenario One: Upgrade Deployment that uses Enterprise Console and HA Tookit

1. Download and install the latest version of Enterprise Console from the Downloads page.
2. Open Enterprise Console and select **Controller**. In the Controller list, it displays only the primary Controller of the HA Pair. If you have an existing pair of HA Controllers which are not managed by Enterprise Console, you need to forget the Controller from within Enterprise Console. It will only show the primary Controller.
3. Select the Controller and select **Remove**.

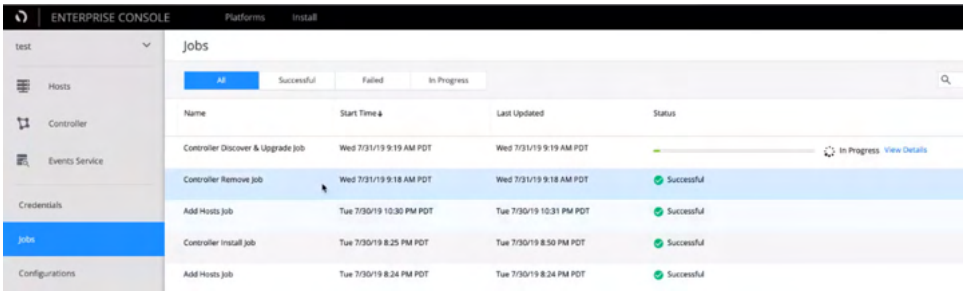
   In the Remove Controller dialog, deselect **Remove Binaries**. At this point, you are just removing the Controller from Enterprise Console but not the AppDynamics software that is running on the HA Controllers.

4. Before we can activate the new HA Module, we need to discover both Controllers from within Enterprise Console. Once the remove Controller job completes, select **Controller > Discover & Upgrade Controller**.

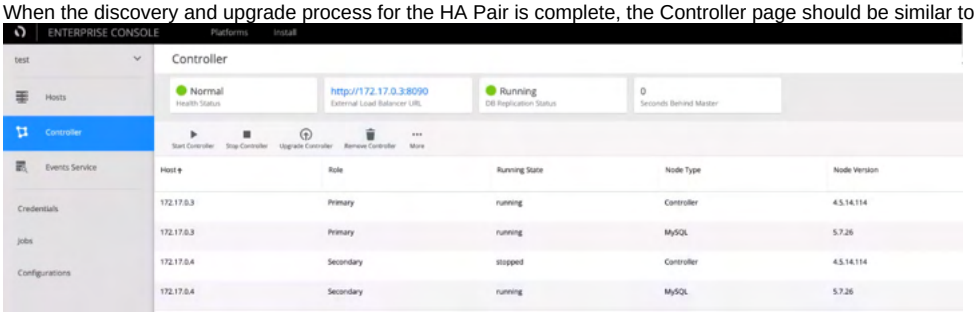
5. Note that in the Discover Controller dialog, you specify the hostnames of both the Controller Primary Host, and the Controller Secondary Host, or replica.



6. Complete the fields of the Discover Controller dialog and select **Continue**. This adds the HA Pair to Enterprise Console where you can then manage and upgrade.
   Access the Jobs page to see several jobs that have completed successfully. The Controller Discover & Upgrade Job will take a while to complete. Select **View Details** to track the progress of the tasks involved to discover and upgrade the Controller.



When the discovery and upgrade process for the HA Pair is complete, the Controller page should be similar to the following:

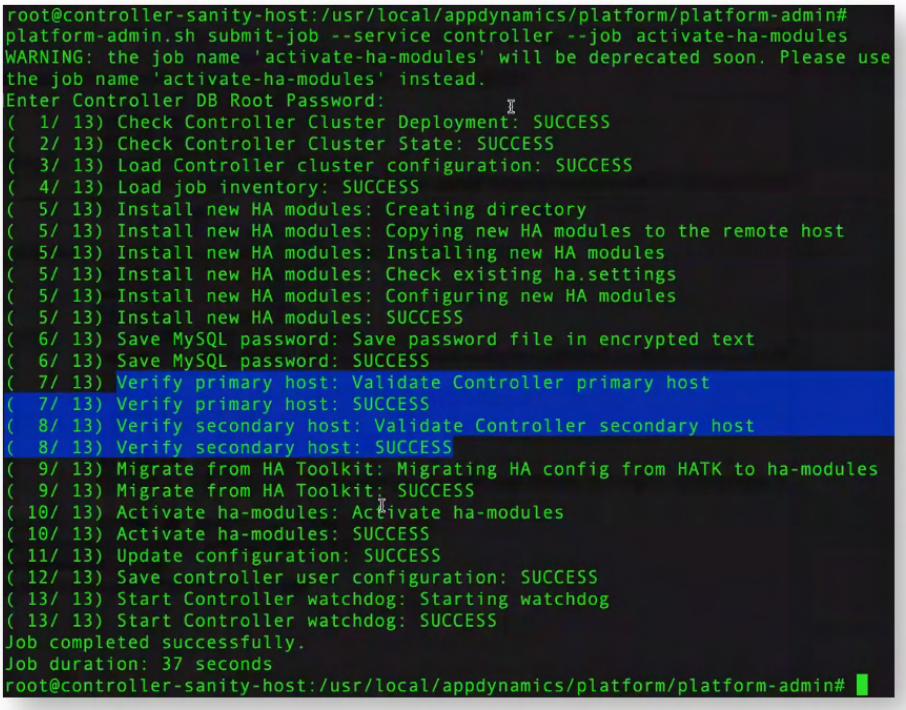


Now, Enterprise Console knows about the HA Pair and has copied the new HA Module to the primary and secondary Controllers in the HA Pair.

7. To activate the HA Pair, open a command shell on the Enterprise Console host and enter the following:

```
platform-admin.sh submit-job --service controller  --job activate-ha-modules
```

You should see output similar to the following:



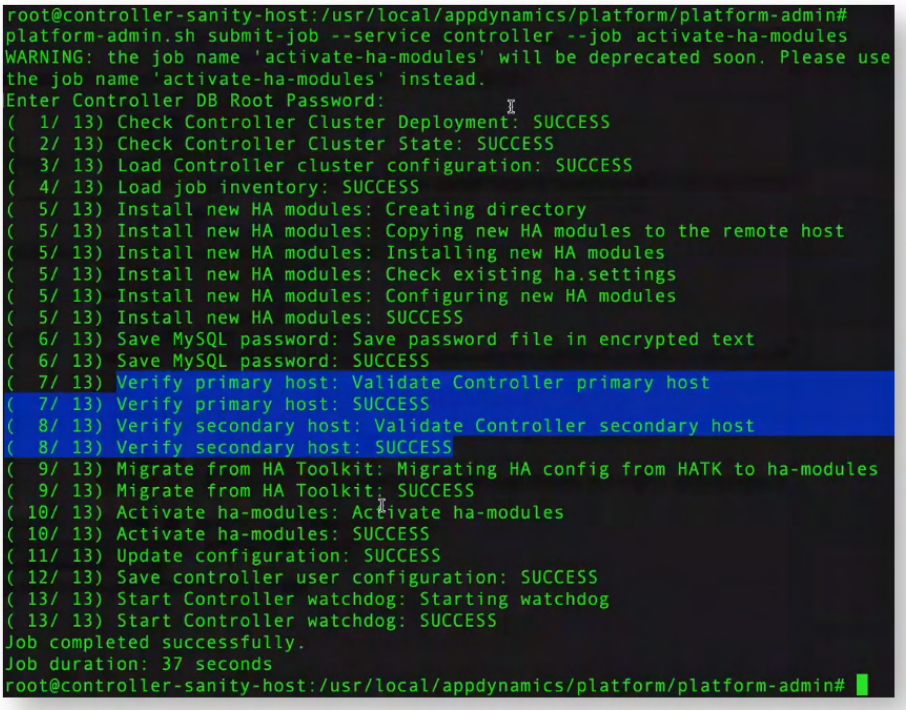
8. Verify that the Controller HA failover is working by terminating the Primary Controller and allow Controller Watchdog to trigger a failover. Then, wait a few minutes to ensure AppDynamics agents are reporting metrics to the Controller.

Scenario Two: Upgrade Deployment Managed by Enterprise Console Alone

1. Download and install the latest version of Enterprise Console from the Downloads page.
2. Upgrade the installed version of Enterprise Console by running the installer on the Enterprise Console host.
3. Upgrade both Controllers by selecting **Upgrade Controller**.
4. To activate the HA Pair, open a command shell on the Enterprise Console host and enter:

```
platform-admin.sh submit-job --service controller  --job activate-ha-modules
```

You should see output similar to the following:



5. Verify that the Controller HA failover is working by terminating the Primary Controller and allows Controller Watchdog to trigger a failover. Then wait a few minutes to ensure AppDynamics agents are reporting metrics to the Controller.
6. When you enable the HA module, the HATK folder is renamed which results in broken services.
   To re-install the services correctly on a pair of HA servers, you must enter the following on the primary and secondary servers:
   a. Log in with `root` privileges.
   b. Change directories to: `<controller_home>/controller-ha/init`
   c. Using the HA module, you must first remove the services installed by HATK by running this script:

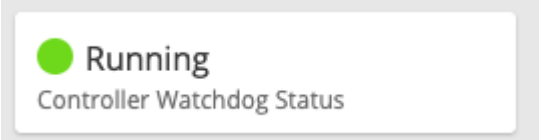
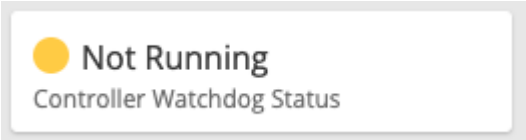
      ```
      ./uninstall-init.sh
      ```

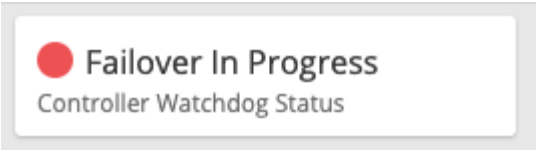
   d. Run `install-init.sh` and install the same services with one of the following options to elevate the user privilege:

      - `-c` #use setuid c wrapper
      - `-s` #use sudo
      - `-p` #use prune wrapper
      - `-x` #use user privilege wrapper

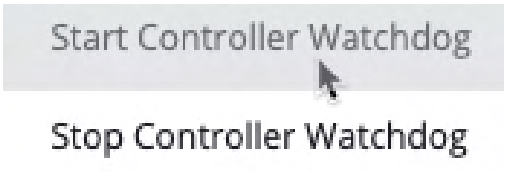
Watchdog Widget

After activating the HA Module, a new widget displays in the Enterprise Console UI indicating the Controller Watchdog status. It has three states:

Controller Watchdog Status	Definition
	The Controller watchdog process is constantly checking the health of your primary Controller. No user action is required.
	In the event that your primary Controller goes down, the failover is not automatic. AppDynamics recommends that you start the Controller watchdog using the *Start Controller Watchdog* option in Enterprise Console.
	Failover is currently in progress where your primary Controller and secondary Controller switch roles



Prior to performing any maintenance operations on your Primary Controller host (which could result in stopping the Controller), AppDynamics recommends that you select **Stop Controller Watchdog** to prevent a failover from initiating. Use the Enterprise Console UI to start and stop watchdog by clicking the watchdog widget, and then selecting **Stop** or **Start Controller Watchdog**.



Start and Stop the Controller Watchdog with CLI

You can start and stop the Controller Watchdog using the following commands:

Stop the Controller Watchdog

```
./platform-admin.sh submit-job --job stop-controller-watchdog --service controller
```

Start the Controller Watchdog

```
./platform-admin.sh submit-job --job start-controller-watchdog --service controller
```

Administer the Controller

This section contains pages on administering the Controller. Depending on how you installed and deployed the platform, you may have more than one method to administer the platform.

If you use the Enterprise Console, many tasks can be done through the GUI or command line. For information about how to use the Enterprise Console, see the pages in the Enterprise Console section.

Additionally, AppDynamics provides command-line tools for common operations, such as starting and stopping the platform components and their services. Some tasks involve using the administration interface for the underlying Glassfish application server, either the web interface or the command-line tool.

- Administer Controller configurations via the Enterprise Console.
- Other configurations:
    - `admin.jsp`: Access basic operational settings for the AppDynamics installation, including data retention settings, tenancy mode, and more.
    - Glassfish administration tool: Use the Glassfish admin tool, as admin, to configure settings in the underlying application server.

The pages in this section describe how to use the tools along with other administration tasks.

Start or Stop the Controller

You can start or stop the Controller, and also check the Controller health from the Enterprise Console GUI or CLI.

Scripts Used to Start or Stop the Controller

The scripts to start and stop the Controller and other AppDynamics platform processes are located in the `platform-admin/bin` directory for standalone or secondary HA Controllers. Using the `platform-admin` script, you can start the platform processes individually or all at once.

> To avoid the possibility of data corruption errors, be sure to stop the application server and database gracefully by using the stop scripts before shutting off or rebooting the machine on which the Controller is running.

You can start and stop the Controller and Controller services on the Controller page in the GUI or from the command line. When you start and stop the Controller, services and processes related to the Controller also start and stop, including the Reporting Service. If you use the GUI to start and stop the Controller, specify that you want to stop MySQL if you want to also stop the Controller Database.

> If your Enterprise Console manages multiple platforms, to distinguish the Controller platform, you must use the command line for standalone or secondary HA Controllers and specify the `--platform-name <name_of_the_platform>`.
>
> For example:
>
> ```
> platform-admin.sh start-controller-db --platform-name <name_of_the_platform>
> platform-admin.sh start-controller-appserver --platform-name <name_of_the_platform>
> ```
>
> To see all options, run `platform-admin.sh list-jobs --service controller --platform-name <platform-name>` from the command line. For more information, see Enterprise Console Command Line or Administer the Enterprise Console.

The Enterprise Console has a max wait time of 45 minutes when starting or stopping the Controller. You can set a timeout which exits the command and returns a failure by appending `--args controllerProcessTimeoutInMin=<minutes>` to the end of your start or stop command.

How to Start or Stop a Standalone or Secondary High Availability Controller

The commands below only apply to standalone and secondary HA Controllers. See Starting or Stopping a Primary Controller for information on how to start or stop primary Controllers.

Start and Stop the Controller App Server

> - When using the Enterprise Console, starting or stopping the Controller will also start or stop the Reporting Service.
> - You can only start or stop the Secondary Controller.

Start the Controller App Server

The `start-controller-appserver` command starts the database automatically.

Stop the Controller App Server

The `stop-controller-appserver` command does not stop the database.

Start and Stop the Controller Database

Start the Controller Database

Stop the Controller Database

How to Start or Stop a Primary Controller in High Availability Pairs

Use the following commands to start or stop primary Controllers in HA pairs. You must log in to the primary Controller host before running the scripts to start the `controller.sh` processes.

- You should disable auto-failover before restarting a primary Controller. Do not forget to reenable auto-failover afterward.
- If you enabled auto-failover in the Enterprise Console, and you stopped the app server to update certifications, the Enterprise Console will trigger a failover if it takes longer than five minutes to update.
- If you are using a combination of the Enterprise Console with the High Availability Toolkit (HATK), then you can start or stop the Controller using services.

Start and Stop the Controller

To start the Controller, run this script from the Controller home:

To stop the Controller:

Start and Stop the Controller App Server

To start the app server, run this script from the Controller home:

To stop the app server, run this command:

Start and Stop the Controller Database

On Linux

To start the database, run this script from the Controller home:

```
bin/controller.sh start-db
```

To stop the database:

```
bin/controller.sh stop-db
```

How to Check Controller Health

On Linux

To check on the health of the Controller, run:

```
bin/platform-admin.sh check-controller-health
```

The following output shows the status of the Controller and its uptime:

```
Controller status : HEALTHY; Started 6 seconds ago.
```

Use a Reverse Proxy

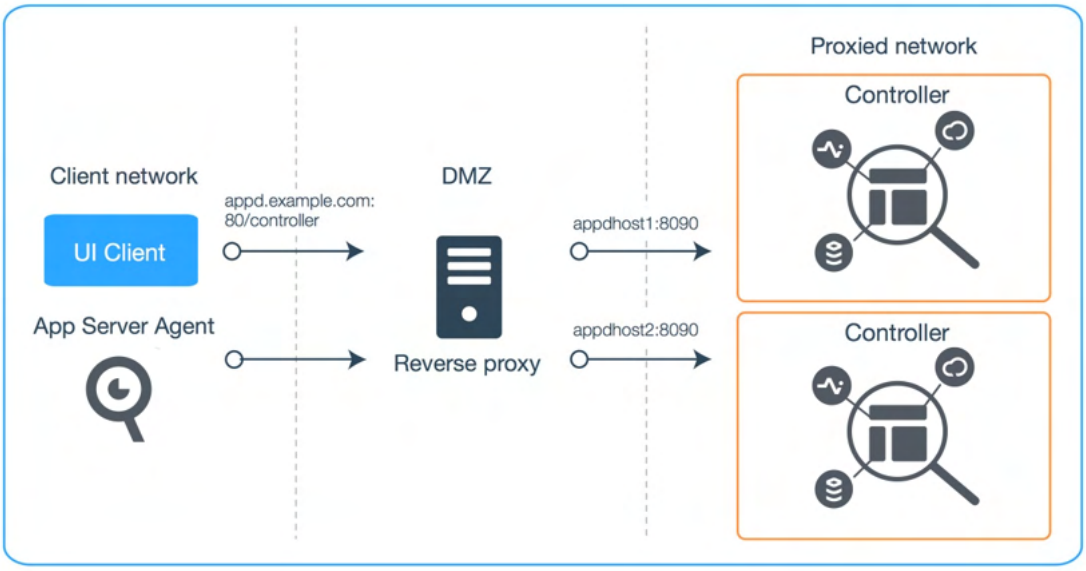
This page describes how to set up the Controller behind a reverse proxy.

Reverse Proxy Architectures

The AppDynamics Controller is often deployed behind a reverse proxy. The proxy presents a virtual IP address to external connections, such as agents and browser clients. The proxy often resides in the DMZ for the network, where it terminates SSL connections from external clients.

The proxy provides a security layer for the Controller, but it also enables you to move a Controller to another machine or switch between high availability pairs without having to reconfigure agents.

The following diagram illustrates the scenario:



As shown, the reverse proxy listens for incoming requests on a given path, /controller in this case, on port 80. It forwards matching requests to the HTTP listening port of the primary Controller at `appdhost1:8090`.

In terms of network impact in this scenario, switching active Controllers from the primary to the secondary in this scenario only requires the administrator to update the routing policy at the proxy so that traffic directed to the secondary instead of the primary.

These instructions describe how to set up the Controller with a reverse proxy. They also provide sample configurations for a few specific types of proxies, including NGINX and Apache Web Server.

This information is intended for illustration purposes only. The configuration requirements for your own deployment are likely to vary depending on your existing environment, the applications being monitored, and the policies of your organization.

While AppDynamics supports Controllers that are deployed with a reverse proxy, AppDynamics Support cannot guarantee help with specific set up questions and issues particular for your environment or the type of proxy you are using. For this type of information, please consult the documentation provided with your proxy technology. Alternatively, ask the AppDynamics community.

General Guidelines

The following describes general requirements, considerations, and features for deploying the AppDynamics Controller and App Agents with a reverse proxy.

- Set the deep link JVM option, `-Dappdynamics.controller.ui.deeplink.url`, to the value of the Controller URL. Use either the hostname or IP address of the Controller host (if directly accessible to clients) or to the VIP for the Controller as exposed at the proxy in the following format:

```
modifyJvmOptions add Dappdynamics.controller.ui.deeplink.url=http[s]://controller.corp.example.com[:port]
/controller
```

Use the URI scheme (http or https), hostname and port number appropriate for your Controller. The Controller uses the deep link value to compose URLs it exposes in the UI.

- If terminating SSL at the proxy, also set the following JVM options:

```
-Dappdynamics.controller.services.hostName=<external_DNS_hostname>
-Dappdynamics.controller.services.port=<external_port_usually_443>
```

- You should use the Enterprise Console's Controller Settings page to edit the JVM options to retain your settings. See Update Platform Configurations for more information.
- If the proxy sits between monitored tiers in the application, make sure that the proxy passes through the custom header that AppDynamics adds for traffic correlation, singularity header. Most proxies pass through custom headers by default.
- For App Agents, the Controller Host and Controller Port connection settings should point to the VIP or hostname and port exposed for the Controller at the reverse proxy. For details see Agent-to-Controller Connections.
- If using SSL from the agent to the proxy, ensure that the security protocols used between the App Agent and proxy are compatible. See the compatibility table for the SSL protocol used by each version of the agent.
- If the proxy (or another network device) needs to check for the availability of the Controller, it can use Controller REST resource at: `http://<host>:<port>/controller/rest/serverstatus`. If the Controller is active and if in high availability mode, is the primary, it returns an XML response similar to this one:

```
<serverstatus vendorid="" version="1">
   <available>true</available>
   <serverid/>
   <serverinfo>
       <vendorname>AppDynamics</vendorname>
       <productname>AppDynamics Application Performance Management</productname>
       <serverversion>003-008-000-000</serverversion>
   </serverinfo>
</serverstatus>
```

If the Controller is in standby mode, this resource returns a 503 response.

The following sections provide notes and sample configurations for a few specific types of proxies, including Nginx and Apache Web Server.

Using Nginx as a Simple HTTP Reverse Proxy

Nginx is a commonly used web server and reverse proxy available at http://nginx.org/.

To use Nginx as a reverse proxy for the Controller, simply include the Controller as the upstream server in the Nginx configuration. If deploying two Controllers in a high availability pair arrangement, include the addresses of both the primary and secondary Controllers in the upstream server definition.

The following steps walk you through the set up at a high level. It assumes you have already installed the Controller and have an Nginx instance, and you only need to modify the existing configuration to have Nginx route traffic to the Controller.

To route Controller traffic through an Nginx reverse proxy

1. Add a JVM option named `-Dappdynamics.controller.ui.deeplink.url`. Set its value to the URL for the Controller, as described in the guidelines above.
2. Shut down the Controller.
3. If terminating SSL at the proxy, also set the `-Dappdynamics.controller.services.hostName` and `-Dappdynamics.controller.services.port` JVM options to the external DNS hostname for the Controller and the external port number, typically 443.
4. In the Nginx home directory on the reverse proxy machine, open the `conf/nginx.conf` file for editing.
5. In the configuration file, add a cluster definition the specifies each Controller as an upstream server. For example:

```
 upstream appdcontroller {
  server 127.0.15.11:8090 fail_timeout=0;
}

server {
    listen 80;
    server_name appdcontroller.example.com;

    expires 0;
    add_header Cache-Control private;

    location / {
        proxy_set_header    Host $host;
        proxy_set_header    X-Real-IP $remote_addr;
        proxy_set_header    X-Forwarded-Proto https;
        proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;

        proxy_pass          http://appdcontroller;
    }
}
```

In the sample, the Controller resides on 127.0.15.11 and has the fully qualified domain name `appdcontroller.example.com`.

6. Restart the Nginx server to have the change take effect.
7. Restart the Controller.

After the Controller starts, it should be able to receive traffic through Nginx. As an initial test of the connection, try opening the Controller UI via the proxy, that is, in a browser, go to `http://<virtualip>:80/controller`. For the App Agents, you'll need to configure their proxy host and port settings as described in the general guidelines above.

Using Apache as a Reverse Proxy

To use Apache as a reverse proxy, you need to make sure the appropriate Apache module is installed and enabled in your Apache instance. For HTTP proxying, this is typically `mod_proxy_http`. The `mod_proxy_http` module supports proxied connections that use HTTP or HTTPS.

To configure Apache with mod_proxy_http

1. Add a JVM option named `-Dappdynamics.controller.ui.deeplink.url`. Set its value to the URL for the Controller, as described in the guidelines above.
2. Shut down the Controller.
3. If terminating SSL at the proxy, also set the `-Dappdynamics.controller.services.hostName` and `-Dappdynamics.controller.services.port` JVM options to the external DNS hostname for the Controller and the external port number, typically 443.
4. On the machine that runs Apache, check whether the required modules are already loaded by your Apache instance by running this command:

```
apache2ctl -M
```

In the output, look for proxy modules as follows:

```
proxy_module (shared)
proxy_http_module (shared)
```

The `proxy_module` is a dependency for `proxy_module_http`.

5. If they are not loaded, enable the Apache module as appropriate for your distribution of Apache. For example, on Debian/Ubuntu:
    a. Enter the following:

    ```
    sudo a2enmod proxy_http
    ```

    b. Restart Apache:

    ```
    sudo service apache2 restart
    ```

6. Add the proxy configuration to Apache. For example, a configuration that directs clients requests to the standard web port 80 at the proxy host to the Controller could look similar to this:

```
<Proxy *>
    Order deny,allow
    Allow from all
</Proxy>

ProxyRequests       Off
ProxyPreserveHost   On

ProxyPass /controller http://controller.example.com:8090/controller
ProxyPassReverse /controller  http://controller.example.com:8090/controller
```

7. Apply your configuration changes by reloading Apache modules. For example, enter:

```
sudo service apache2 reload
```

8. Start the Controller.

After the Controller starts, test the connection by opening a browser to the Controller UI as exposed by the proxy. To enable AppDynamics App Agents to connect through the proxy, be sure to set the proxy host and port settings in the proxy, as described in the general guidelines above. Also, be sure to apply any of the other general guidelines described in the general guidelines above.

Configure SSL Termination at the Reverse Proxy

This section describes how to set up security when the client-side connection to the proxy uses SSL that's terminated at the proxy. This assumes that the proxy and Controller are in a secured data center and the App Agents or UI browser client connections are from a potentially insecure network.

Terminating SSL at a proxy offloads the burden of SSL processing from the Controller to the proxy. This configuration is strongly recommended when deploying the Controller to large scale, high workload environments. Terminating SSL at a proxy also provides the benefit of having a central point in the data center for the security certificate and for key management.

This section provides a sample configuration for Nginx, but the concepts translate to other types of reverse proxies as well.

Configure the Proxy for SSL Termination

To perform SSL termination at the reverse proxy, you need to:

- Ensure that the App Agents can establish a secure connection with the proxy. See Agent and Controller Compatibility for SSL settings for various versions of the agent. Ensure that the proxy includes a server certificate signed by an authority that is trusted by the agent. Otherwise, you will need to install the proxy machine's server key.
- If using .NET App Agents in your environment, verify that the reverse proxy server uses a server certificate signed by a certificate authority (CA). The .NET App Agent does not permit SSL connections based on a self-signed server certificate.
- Configure the proxy to forward traffic between it and the Controller to a secure port between it and the client.
- The client App Agents and browser clients under this configuration *must use* the secure port to communicate with the Controller (i.e., the proxy). Configuring a mixed channel on the Controller as described here causes the agents to perform as if they were using a secure port. Therefore, you need to ensure that they use a secure port only.

A complete example configuration with Nginx performing SSL termination for the Controller would look something like this:

```
 upstream appdcontroller {
  server 127.0.15.11:8191 fail_timeout=0;
}

server {
    listen 80;
    server_name appdcontroller.example.com;
    return 301 https://$host$request_uri;
}
server {
    listen 443;
    server_name appdcontroller.example.com;

    ssl on;
    ssl_certificate /etc/nginx/server.crt;
    ssl_certificate_key /etc/nginx/server.key);

    ssl_session_timeout  5m;
    ssl_protocols  TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers  ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+EXP;
    ssl_prefer_server_ciphers   on;

    expires 0;
    add_header Cache-Control private;

    location / {
        proxy_set_header    Host $host;
        proxy_set_header    X-Real-IP $remote_addr;
        proxy_set_header    X-Forwarded-Proto https;
        proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_redirect      http:// https://;
        proxy_pass          http://appdcontroller;
    }
}
```


TLSv1 should only be enabled if absolutely necessary.

This example builds on the configuration shown in the simple passthrough example. In this one, any request received on the non-SSL port 80 is routed to port 443. The server for port 443 contains the settings for SSL termination. The `ssl_certificate_key` and `ssl_certificate` directives should identify the location of the server security certificate and key for the proxy.

The configuration also indicates the SSL protocols and ciphers accepted for connections. The security settings need to be compatible with the AppDynamics App Agent security capabilities, as described on the Agent and Controller Compatibility page.

Cookie Security

The Controller sets the secure flag on the `X-CSRF-TOKEN` cookie sent over HTTPS. The secure flag ensures that clients only transmit the cookies on secure connections.

If you terminate HTTPS at a reverse proxy in front of the Controller, the Controller does not set cookie security by default because connections to the Controller would occur over HTTP.

To ensure cookie security, configure the reverse proxy to include the `secure` statement in the `set-cookie` statement. How to add the secure flag varies on the type of reverse proxy you use.

The following example shows how to set cookie security using HAProxy:

- If using HAProxy version 2.0 or earlier, enter:

  ```
  rspirep ^(set-cookie:.*) \1;\ Secure
  ```

- If using HAProxy version 2.1 or later, enter:

  ```
  acl secured_cookie res.hdr(Set-Cookie),lower -m sub secure
  http-response replace-header Set-Cookie (.*) \1;\ Secure if !secured_cookie
  ```

Using SSL from the Reverse Proxy to the Controller

Have the proxy connect to the Controller with SSL requires a minor modification to the proxy configuration. Simply specify the use of HTTPS as the protocol to connect to the backend or upstream server. In other words, for the Nginx configuration, this simply requires you to modify the `proxy_pass` value as follows:

```
proxy_pass          https://appdcontroller;
```

To complete the configuration, make sure you have configured SSL on the Controller as described in Controller SSL and Certificates.

Enable an Email Server

The SMTP email server must be configured to enable email and SMS notifications and digests to be sent by the Controller.

Permissions

For this activity, users need the predefined Account Owner role or another role with the **Configure Email / SMS** permission.

Configure an SMTP server

1. In the Controller UI, from the top navigation menu bar, click **Alert & Respond > Email/ SMS Configuration**.
2. Provide the connection information for the SMTP host and port.
    - A SaaS Controller should be preconfigured with the appropriate settings, but verify the settings as the following:
        a. **SMTP Host:** localhost
        b. **SMTP Port:** 25

      No authentication is needed.
    - For an on-premises controller, use a host and port settings for an SMTP server available in the controller deployment environment.
3. Customize sender address in notifications emails in the **From Address** field. By default, emails are sent by the root Controller user.
4. If the SMTP host requires authentication, configure the credentials in the **Authentication** settings.
5. If you want to add any text to the beginning of the notification, enter it in the **Notification Header Text** field.
6. If you are using SMS do one of the following:
    - Select **Default** and choose one of the available carriers from the pulldown menu.
    - Select **Custom** and enter the phone number receiving the message as <phone number>@<sms gateway>.

      For example, a mobile phone in the United States serviced by AT&T might be:

      ```
      4151234567@txt.att.net
      ```

      A mobile phone in the United Kingdom serviced by Textlocal might be:

      ```
      7412345678@txtlocal.co.uk
      ```

      See SMS gateway by country for information on most common SMS gateways.
7. Test the configuration by sending an email.
8. Save the settings.

Troubleshoot Notifications

If you do not receive notifications for health rule violations, it could be because the default SMTP server timeout period is too short. To troubleshoot, increase the value of the mail.smtp.socketiotimeout Controller Setting in the Administration Console. The default value is 30 seconds.

Update the Root User and Glassfish Admin Passwords

This page describes how to change the passwords for the root user and Glassfish admin for the Controller.

Root User and Account Owners

The root user is a built-in Controller user with global administrator privileges in the Controller environment. Only the root user can access the System Administration Console, where you can create and manage accounts in multi-tenant Controllers and configure global Controller settings in both single- or multi-tenant Controllers.

The root user is a superuser for the Controller. Unlike other types of users, you cannot remove the root user account or create other superuser accounts in the Controller. The password for the root user is set during installation but you can change the root password in the Administration Console.

While the root user has global administrative privileges, account administrators act as administrators only within individual accounts in a multi-tenant Controller. It's typically the role of the root user to create accounts and an initial administrator for the account, and the role of each account administrator to create additional users within the account. See Roles and Permissions and Manage Users and Groups

Change the Controller Root User Password

You can change the root user password from the AppDynamics Administration Console page.

To change the root user password

1. Log in to the administration console as described in Access the Administration Console.
2. Click **Settings** and choose **My Settings**.
3. Click **Edit > Change Password.**
4. Type the new password for the root user in the **New Password** and **Repeat New Password** fields.
5. Click **Save**.

> Logging in to the Administration Console requires you to have the root user password. If you do not have the root user password and need to reset it.

Reset Root User Password

If you have lost the AppDynamics root user password for your installation and need to reset it, follow these steps:

1. From the command line, change to the Controller's `bin` directory. For example, on Linux:

   ```
   cd <controller_home>/bin
   ```

2. Use the following script to log in to the Controller database of the Controller;
   - For Windows: `controller.bat login-db`
   - For Linux: `sh controller.sh login-db`

   You will see a MySQL prompt.
3. After running the script, you will be prompted to enter a password. Enter the root password for the Controller database.
4. From the MySQL prompt, enter the following SQL command to get root user details:

   ```
   select * from user where name='root' \G;
   ```

5. Use the following SQL command to change the password:

   ```
   update user set encrypted_password = sha1('<NewPassword>') where name = 'root';
   ```

   The hash for the password will be upgraded to PBKDF2 when you log in.
6. Restart the Appserver.

For information on setting the database root user password, see Controller Data and Backups.

Change the Glassfish Admin User Password

The Controller uses the built-in administrator account in the underlying GlassFish application server. Changes to the GlassFish admin user password must be made in GlassFish and the Enterprise Console.

Change the Password in GlassFish

1. Run the `change-admin-password` subcommand in remote mode, for example:

```
asadmin> change-admin-password --user <username>
```

2. Enter the existing password and new password when prompted.
3. Restart the Controller using the Enterprise Console.

Update the Password in the Enterprise Console

The Enterprise Console connects to your underlying GlassFish administration server via the GlassFish admin password during Controller upgrades. If you change the GlassFish admin user password, you need to update the password in the Enterprise Console as well to ensure successful Controller upgrades.

To update the GlassFish admin user password in the Enterprise Console, see the following steps:

1. Log in to the Enterprise Console and select the desired platform.
2. Select **Configurations** > **Controller Settings** > **Appserver Configurations**.
3. In the **Basic** tab, find **Glassfish Admin Password**. Enter the new password
4. Reenter the new password in **Confirm Glassfish Admin Password**.
5. Click **Save**.

The password change takes immediate effect.

Change the Controller Database Root User Password

Downtime is required to change the Controller database root user password. If you have installed a Controller HA pair, you must disable auto-failover to avoid an accidental failover while changing the password. For more details about disabling auto-failover, click Automatic Failover.

To change the Controller database root user password:

1. Log in to the Controller host. From a command line, enter:

```
cd <controller_home_dir>
```

2. To stop the App Server and the database, enter:

```
bin/controller.sh stop
```

If you are using MS Windows, you must use the Windows services to stop the Controller.

3. To start the database in insecure mode, enter:

```
bin/controller.{sh|bat} start-db insecure
```

The insecure option starts the database without password requirements. Use this option only to reset the password for the database. The option is similar to starting MySQL with the `--skip-grant-tables` option.

4. To log in to the database, enter:

```
bin/controller.{sh|bat} login-db insecure
```

5. Use MySQL to run the following commands:
    a. To specify the Controller database, enter:

```
use mysql;
```

    b. To reload the MySQL grant tables, enter:

```
FLUSH PRIVILEGES;
```

c. To determine your MySQL version, enter:

```
select version();
```

d. Based on which MySQL version you are using:
MySQL 5.5 version: Configure the new password for the root user by entering:

```
update mysql.user set password=password('<new-password-here>') where user like 'root%';
```

MySQL 5.7 version: Configure the new password for the root user by entering:

```
update mysql.user set authentication_string=password('<new-password-here>') where user like
'root%';
```

e. To reload the MySQL grant tables, enter:

```
FLUSH PRIVILEGES;
```

f. To exit MySQL, enter:

```
quit
```

6. To stop the database, enter:

```
bin/controller.{sh|bat} stop-db
```

7. To start the App Server, enter:

```
bin/controller.sh start
```

If you are using MS Windows, you must use the Windows services to start the Controller.

You need to change the password on both of the Controller HA pairs.

8. To update the Controller database root password and ensure that the Enterprise Console is aware of the new password:

a. Log in to the Enterprise Console and select the desired platform.
b. Select **Configurations** > **Controller Settings** > **Database Configurations**.
c. Enter the new password in **Controller DB Root Password**.
d. Reenter the password in **Confirm Controller DB Root Password**.
e. Select **Save**.

The password change takes immediate effect.

If you previously disabled auto-failover, you should now enable it.

Change the Controller DB User Password

Downtime is required to change the Controller database root user password. If you have installed a Controller HA pair, you must disable auto-failover to avoid an accidental failover while changing the password. For more details about disabling auto-failover, click Automatic Failover.

To change the Controller DB user password:

1. Log in to the Enterprise Console and select the desired platform.
2. Select **Configurations > Controller Settings > AppServer Configurations**.
3. In the **Basic** tab, enter the new password in **New password for Controller DB user**.
4. Re-enter the password in **Confirm New password for Controller DB user**.
5. Click **Save**.

The password change takes immediate effect.

Change the Controller Owner

This page provides instructions for changing the Controller owner.

You may need to change the user who is running the Controller services during a system migration or other event.

The procedure varies based on whether you are using the Enterprise Console.

In order to change the owner of the Controller, complete the following steps.

1. Stop all running Controller services using the following command in the machine terminal.
    - `./controller.sh stop`
2. Change the username and user group of the Controller directory.
    - `chown -R <New User>:<User Group> <Controller Folder>`
3. Update the new user in the `db.cnf` file located at `Controller/db/db.cnf`.
    - `user=<New User>`
4. Start the Controller.
    - `./controller.sh start`

If you are not using the Enterprise Console

1. As the current user running the Controller services, shut down the Controller process:

```
CONTROLLER_HOME_DIR/bin/controller.sh stop
```

2. Change the ownership (recursively) of the entire Controller directory to the new user. In this example, `appdynamics:admin` is the `user:group`, respectively:

```
chown -R appdynamics:admin CONTROLLER_HOME_DIR/
```

3. If the Controller's data directory is outside of the `root` Controller's folder, then you must also change the owner of the database data files:

```
chown -R appdynamics:admin .../data/
```

4. Change the user to the new username:

```
CONTROLLER_HOME_DIR/db/db.cnf
```

5. Log in as the new user and start the Controller services:

```
CONTROLLER_HOME_DIR/bin/controller.sh start
```

If you are using the Enterprise Console

1. Remove the Controller from the Enterprise Console by de-selecting the **Remove Binaries** option; otherwise, the binaries will be removed from the disk. To remove the Controller without uninstalling the Controller:

```
PLATFORM_HOME_DIR/bin/platform-admin.sh submit-job --service controller --job remove --args
removeBinaries=false --skip-confirm
```

2. As the current user running the Controller services, shut down the Controller process.

```
CONTROLLER_HOME_DIR/bin/controller.sh stop
```

3. Change the ownership (recursively) of the entire Controller directory to the new user. In this example, `appdynamics:admin` is the `user:group`, respectively:

```
chown -R appdynamics:admin CONTROLLER_HOME_DIR/
```

4. If the Controller's data directory is outside of the `root` Controller's folder, then you must also change the owner of the database data files:

```
chown -R appdynamics:admin .../data/
```

5. Change the user to the new username:

```
CONTROLLER_HOME_DIR/db/db.cnf
```

6. Log in as the new user and start the Controller services:

```
CONTROLLER_HOME_DIR/bin/controller.sh start
```

7. From the Enterprise Console, remove the hosts that were added:

```
PLATFORM_HOME_DIR/bin/platform-admin.sh remove-dead-hosts --hosts $CONTROLLER_HOST --skip-confirm
```

8. Remove the credentials because the credentials are connected to the previous user.
9. Add the credentials using the new user, and then add the host.
10. Perform a Discover and Upgrade for the Controller.
11. (Optional) If you have installed the Linux services, then:
    a. Logged in as root, uninstall the services:

```
HA/uninstall-init.sh
```

    b. Logged in as root, install the services using either the `-c` or `-s` options:

```
HA/install-init.sh
```

Change the User Running the Controller Services

This page provides instructions for changing the user running the Controller services.

You may need to change the user who is running the Controller services during a system migration or other event.

The procedure varies based on whether you are using the Enterprise Console.

If you are not using the Enterprise Console

1. As the current user running the Controller services, shut down the Controller process:

   ```
   CONTROLLER_HOME_DIR/bin/controller.sh stop
   ```

2. Change the ownership (recursively) of the entire Controller directory to the new user. In this example, `appdynamics:admin` is the `user:group`, respectively:

   ```
   chown -R appdynamics:admin CONTROLLER_HOME_DIR/
   ```

3. If the Controller's data directory is outside of the `root` Controller's folder, then you must also change the owner of the database data files:

   ```
   chown -R appdynamics:admin .../data/
   ```

4. Change the user to the new username:

   ```
   CONTROLLER_HOME_DIR/db/db.cnf
   ```

5. Log in as the new user and start the Controller services:

   ```
   CONTROLLER_HOME_DIR/bin/controller.sh start
   ```

If you are using the Enterprise Console

1. Remove the Controller from the Enterprise Console by de-selecting the **Remove Binaries** option; otherwise, the binaries will be removed from the disk. To remove the Controller without uninstalling the Controller:

   ```
   PLATFORM_HOME_DIR/bin/platform-admin.sh submit-job --service controller --job remove --args
   removeBinaries=false --skip-confirm
   ```

2. As the current user running the Controller services, shut down the Controller process.

   ```
   CONTROLLER_HOME_DIR/bin/controller.sh stop
   ```

3. Change the ownership (recursively) of the entire Controller directory to the new user. In this example, `appdynamics:admin` is the `user:group`, respectively:

   ```
   chown -R appdynamics:admin CONTROLLER_HOME_DIR/
   ```

4. If the Controller's data directory is outside of the `root` Controller's folder, then you must also change the owner of the database data files:

   ```
   chown -R appdynamics:admin .../data/
   ```

5. Change the user to the new username:

   ```
   CONTROLLER_HOME_DIR/db/db.cnf
   ```

6. Log in as the new user and start the Controller services:

```
CONTROLLER_HOME_DIR/bin/controller.sh start
```

7. From the Enterprise Console, remove the hosts that were added:

```
PLATFORM_HOME_DIR/bin/platform-admin.sh remove-dead-hosts --hosts $CONTROLLER_HOST --skip-confirm
```

8. Remove the credentials because the credentials are connected to the previous user.
9. Add the credentials using the new user, and then add the host.
10. Perform a Discover and Upgrade for the Controller.
11. (Optional) If you have installed the Linux services, then:
    a. Logged in as root, uninstall the services:

    ```
    HA/uninstall-init.sh
    ```

    b. Logged in as root, install the services using either the `-c` or `-s` options:

    ```
    HA/install-init.sh
    ```

Access the Administration Console

This page provides information and access instructions for the AppDynamics Administration Console.

Deployment Support

Related pages:

- Controller Settings for Machine Agents
- Controller Settings for Server Visibility
- Configure Controller Settings for Monitoring Database

Do not confuse the AppDynamics Administration Console with the GlassFish application server administration console or the general application administration page in the Controller UI.

The AppDynamics Administration Console lets you configure certain global settings such as metric retention periods, UI notification triggers, tenancy mode, and accounts in multi-tenancy mode.

AppDynamics recommends that you do not change Controller settings in the console unless under the guidance of an AppDynamics representative or as specifically directed by documentation.

Access the AppDynamics Administration Console

1. If you are logged into a Controller UI session with an account other than the root user, log out or open a new browser window in private (incognito) mode. If you do not, you will get an "Access Denied" error when you attempt to open the console page.
2. In the browser enter the URL of the Administration Console:

   ```
   http://<hostname>:<port>/controller/admin.jsp
   ```

   The console is served on the same port as the Controller UI, port 8090 by default.
3. Log into the system account with the root user password. The root user is a built-in global administrator for the Controller. Use the password you set for this user when installing. See Update the Root User and Glassfish Admin Passwords

   The root user password is different from a normal AppDynamics account password. It is not the same as the account owner or account administrator password. If you are logged into the Controller using your current account, you need to log out of that account and then back in as the root user to access the Administration Console. You can change the Controller root user password if you wish. See Update the Root User and Glassfish Admin Passwords

Access the Glassfish Administration Console

In rare cases, you may need to log in to the AppDynamics Administration Console for the application server underlying the GlassFish server. The GlassFish administration console provides a browser interface for performing many of the same tasks you can perform using the admin command-line utility. You can access the console for the GlassFish server using the built-in user account named admin.

For security reasons, access to the GlassFish browser interface is limited to local machine access by default, so the following steps should be performed from a browser on the Controller machine. Attempts to access the console remotely trigger the error message "Secure Admin must be enabled to access the DAS remotely."

To access the GlassFish administration console:

1. From a web browser on the Controller machine, open the following URL:

```
http://localhost:4848
```

Note that port 4848 is the default port number for the GlassFish administration console, but it may have been set to another value at installation time. If the default port doesn't work and you are unsure of what port number to use, you can check the port configured for the network-listener element named admin-listener in the domain.xml file.

2. Log in as user admin.
   By default, the GlassFish user admin password is the same as the root user password for the Administration Console. You can change the GlassFish user admin password if you wish. See Update the Root User and Glassfish Admin Passwords

Modify the User Session Timeout

The Controller logs users out of Controller UI sessions after 60 minutes of inactivity by default. For an on-premises Controller, it's possible to modify the default timeout value, as follows:

1. Log in to the Administration Console as the AppDynamics root user.
2. Find and set the values for these properties:

    - `http.session.inactive.timeout`: The amount of time without a client request to the Controller after which the user session times out and the user will need to log in again to continue. The default is 3600 seconds (60 minutes).
    - `ui.inactivity.timeout`: The amount of time without user activity in the Controller UI after which the user session times out and the user will need to log in again to continue. The default is -1 (disabled).

Customize System Notifications

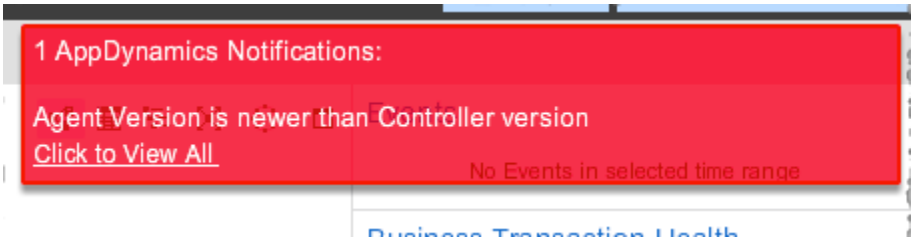
Related pages:

- Access the Administration Console

You can customize system events and system use notification messages from the Administration Console.

System Events Notification

Certain system events trigger event notification popups in the Controller UI.



You can configure which type of events appear as notifications in the UI, as described here.

Configure Events that Trigger UI Notifications

1. Log in to the Administration Console.
2. Go to the **Controller Settings**.
3. Search for the `system.notification.event.types` property. The value of this property determines which type of events result in UI notifications.
4. Set the types of events you want to see by adding them to the comma-separated string in the dialog box. To disable notifications of a particular type of event, remove it from the list. Do not use spaces between commas.

Notification Event Types

Event Value	What This Event Notification Means
`LICENSE`	There is an issue with the status of your license.
`DISK_SPACE`	There is an issue with the amount of disk space left on your system.
`CONTROLLER_AGENT_VERSION_INCOMPATIBILITY`	A mismatch between the version of the agent and the version of the controller has been detected.
`CONTROLLER_EVENT_UPLOAD_LIMIT_REACHED`	The limit on the number of events per minute that can be uploaded to the controller from this account has been reached. Once the limit is reached no more events — other than certain key ones — are uploaded for that minute.
`CONTROLLER_RSD_UPLOAD_LIMIT_REACHED`	The limit on the number of request segment data (RSDs) per minute that can be uploaded to the controller from this account has been reached. RSDs are related to snapshots. Once the limit is reached no more RSDs — other than certain key ones — are uploaded for that minute.
`CONTROLLER_METRIC_REG_LIMIT_REACHED`	The limit for registering metrics for this account has been reached. No further metric registrations are accepted.
`CONTROLLER_ERROR_ADD_REG_LIMIT_REACHED`	The limit for registering error Application Diagnostic Data (ADDs) for this account has been reached. No further error ADD registration is accepted.
`CONTROLLER_ASYNC_ADD_REG_LIMIT_REACHED`	The limit for registering async ADDs for this account has been reached. No further async ADD registration is accepted.
`CONTROLLER_STACKTRACE_ADD_REG_LIMIT_REACHED`	The limit for registering StackTrace ADDs for this account has been reached. No further StackTrace ADD registration is accepted.
`AGENT_ADD_BLACKLIST_REG_LIMIT_REACHED`	If the Agent attempts to register an ADD above the limit, the Controller rejects the attempt and adds the ADD to a blacklist. There is a limit to the size of the blacklist. This event indicates that that limit has been reached.

AGENT_METRIC_BLACKLIST_REG_LIMIT_REACHED	If the Agent attempts to register a metric above the limit, the Controller rejects the attempt and adds the metric to a blacklist. There is a limit to the size of the blacklist. This event indicates that that limit has been reached.

System Use Notification

The system use notification is an optional and configurable message that includes information on privacy and security notices. If enabled, the displayed message must be acknowledged before granting the user further access.

Configure System Use Notification

1. Log in to the Administration Console.
2. Go to the **Controller Settings**.
3. Search for the `system.use.notification.message` property. The value of this property is the message of the system use.
4. Enter your post-login message, which will be displayed every time a user logs in, informing the user of the system usage requirements. There is a 1000 character limit.
   Here is an example message:

```
This is a U.S. Government computer system, which may be accessed and used only for authorized Government
business by authorized personnel. Unauthorized access or use of this computer system may subject
violators to criminal, civil, and/or administrative action. All information on this computer system may
be intercepted, recorded, read, copied, and disclosed by and to authorized personnel for official
purposes, including criminal investigations. Such information includes sensitive data encrypted to
comply with confidentiality and privacy requirements. Access or use of this computer system by any
person, whether authorized or unauthorized, constitutes consent to these terms. There is no right of
privacy in this system.
```

Multi-Tenant Controller Accounts

This page describes how to create and manage accounts in a multi-tenant Controller. The tenant mode determines whether the Controller UI offers single or multiple environments. See Controller Deployment.

Switch from Single-Tenant to Multi-Tenant Mode

> Switching from single-tenancy to multi-tenancy mode is supported. However, switching from multi-tenancy to single-tenancy is not. Take precautions to ensure multi-tenancy is the correct mode for your environment.

If multi-tenancy is enabled for an on-premises Controller, users must enter the account name in the **Account** field when logging in to the Controller UI.

1. Navigate to the Administration Console.
2. Locate the `multitenant.controller` setting.
3. Set the value to `true`.

Create Accounts in Multi-Tenant Mode

In multi-tenant mode, you can add accounts as follows:

1. Log in to the AppDynamics Administration Console as the AppDynamics root user.
2. Click **Account Settings** and then **Add**.
3. Define the licensing entitlements that apply to the account.
   Account-level license unit limits let you prevent a particular account from using more licensing units than it should. You can view the total license units available through **Settings** > **Admin** > **License**. See License Management.

   > The overall license limits applicable at the Controller level are independent of any specific limits you apply at the account level.

   Agent-based Licensing: For example, if an account is set up with a Java Agent limit of 100, you can ensure that the new account never interferes with the license availability of another account by setting the **Java Units Provisioned** value for the account to a much smaller limit. However, if you set it to 100 and other accounts are also set to that amount, the first 100 agents that connect to the Controller would occupy those units, regardless of the accounts they report in to. Similarly, you can limit the lifespan of the account by setting an expiration date for the license.

   Infrastructure-based Licensing: For example, if an account is set up with an Infrastructure Monitoring limit of 100, you can ensure that the new account never interferes with the license availability of another account by setting the **Infrastructure Monitoring** value for the account to a much smaller limit. However, if you set it to 100 and other accounts are also set to that amount, the first servers with CPU cores totalling up to 100 would occupy those units, regardless of the accounts they report in to. Similarly, you can limit the lifespan of the account by setting an expiration date for the license.
4. When finished defining entitlements, click **Save**.

After enabling multi-tenant mode, users must specify the account they want to log into in the **Account** field in the Controller UI login screen. See:

- Java Agent Configuration Properties
- .NET Agent Configuration Properties
- Database Agent Configuration Properties
- Machine Agent Configuration Properties

Administer the Reporting Service

Related pages:

- Reports
- Port Settings

The Reporting Service is a standalone Controller process responsible for generating and transmitting reports. The Controller uses the Reporting Service to send both one-time reports and scheduled reports. For more information about the Reporting Service, see Fonts Needed for the Reporting Service and Inst allation Settings.

Configure the Service

You can configure the Reporting service with the files in the following directory: `<Controller home>/reporting_service/reports/config`. Configure Reporting Service behavior in the `user-config.json` file. Any configuration changes made in user-config.json override default behavior specified in default-config.json. You can configure properties such as the load timeout.

You can configure the Reporting Service with files in the following directory:

`<Controller home>/reporting_service/reports/config`

You can configure Reporting Service behavior in the *user-config.json* file. Any configuration changes made in `user-config.json` override default behavior specified in `default-config.json`.

Disabling HTTP or HTTPS Port

Some on-premise installations can disable the http or https connection. This is done using the same `reportServer:port` config value used to set the listening port. The `default-config.json` installation has the following values for port(s):

```
"reportServer": {
        "port": "8020",
        "portSecure": "8021",
    }
```

To disable https for a localhost system change the "portSecure" to "0":

```
"reportServer": {
        "port": "8020",
        "portSecure": "0",
    }
```

Alternatively, for security reasons if you want to force https, and http you can change the "port" to "0":

```
 "reportServer": {
        "port": "0",
        "portSecure": "8021",
    }
```

After making the change, stop and start the Reports Server as follows:

Windows:

```
cd <installroot>\controller\reporting_service\reports\bin
  reports-service.sh stop
  reports-service.sh start
```

Linux/Mac:

```
cd <installroot>/controller/reporting_service/reports/bin
  ./reports-service.sh stop
  ./reports-service.sh start
  ./reports-service.sh list
```

The reporting-server.log contains info on the port settings during startup.

Limit Reports Service Port Listening to Localhost

Many on-premise installations may want to limit port listening for the Reports Service node server and just listen for localhost connects. These requests do not go onto the network. This is done using the `reportServer:portHostname` and `reportServer:portSecureHostname` value used to set the `listen` hostname parameter. The `default-config.json` installation has these values for the port hostname values, shown below with their port(s):

Default values are:

```
"reportServer": {
      "port": "8020",
      "portHostname" : "",
      "portSecure": "8021",
      "portSecureHostname" : ""
    },
```

The `reporting-server.log` shows info on the hostname settings during startup.
Adding the port*Hostname fields to "localhost" as shown below in `user-config.json` limits the Report Service from connecting to a controller installed on the same host.

```
"reportServer": {
        "portHostname" : "localhost",
        "portSecureHostname" : "localhost"
    },
```

After making the change, stop and start the Reports Server as follows:

Linux/Mac:

```
cd <installroot>/controller/reporting_service/reports/bin
  ./reports-service.sh stop
  ./reports-service.sh start
  ./reports-service.sh list
```

Windows:

```
cd <installroot>\controller\reporting_service\reports\bin
  reports-service.bat stop
  reports-service.bat start
```

See `default-config.json` for more information about configurable properties.

Start and Stop the Service

You can start or stop the Reporting Service independent of the Controller. Run the following commands from the Controller home directory.

When using the Enterprise Console, starting or stopping the Controller will also start or stop the Reporting Service.

Check to see if the Reporting Service is running with the following command:

```
./reporting_service/reports/bin/reports-service.sh|bat list
```

Start the Reporting Service with the following command:

```
./reporting_service/reports/bin/reports-service.sh|bat start
```

Stop the Reporting Service with the following command:

```
./reporting_service/reports/bin/reports-service.sh|bat stop
```

View Logs

The Reporting Service uses the following logs in the Controller home directory:

- `/logs/reporting-server.log`. Prints if the report email was sent and details of the report object that was requested by the user.
- `/reporting_service/reports/logs/reporting-process.log`. Confirms the reporting service process started and whether or not exceptions occurred. Note that this log file is only used on Linux Controllers.

Troubleshooting the Reporting Service

To begin troubleshooting why a report failed to send, open the `server.log` file and find the runUUID for the report you tried to send. Then search for the log entry for the report.

Resolutions for common Reporting Service issues include the following:

- Verify that the Reporting Service is running.
- Verify the default ports for the service: 8020 for HTTP and 8021 for HTTPS
- Verify that the user account used to start the Reporting Service is the same as the account used to start the Controller.

Controller Audit Log

This audit capability creates an `audit.log` file and is used to monitor user activities and configuration changes in the Controller. Be aware that SaaS customers do not have access to the `audit.log` file as it is held on the AppD Controller server. The information is retrieved through the following actions.

Schedule a Controller Audit Report

You must have account-level permissions to view and configure scheduled reports. Use this report to view changes made to the user information, controller configuration, and application properties.

1. Click **Dashboards & Reports** > **Reports** > **Add Report**.
2. Enter **Report Title** and **Report Subtitle**.
    a. You can label a report CONFIDENTIAL using **Report Subtitle**.
    b. Optionally, select **Show Title Page** to include a title page at the beginning of your report file.
3. Select **Report Type** > **Controller Audit** to define the fields in the **Reports Data** tab.
4. Set the time ranges. You can create and manage custom time range if required.
    a. Note: Custom time range options are available for all the **Report Types**.
5. Select your report file format as PDF, JSON, or CSV.
    a. Optionally, uncheck the **Show Diff** box to remove the **Object Changes** column from your report file.
6. Choose the data to include or exclude from the drop-down list.
    a. Repeat as necessary with the following options:
7. Enter the attribute value.
8. Click **+ Add**.

You can create new, duplicate existing, or modify current reports as well as set an email delivery schedule to a defined list of recipients. You can also choose the **Send Report Now** right-click option for an immediate look at the audit details. Review the Reports documentation for more details on other types of reporting.

The Controller Audit reports on the following attributes:

`Date` and `time` range	`ObjectType`
`UserName`	`ObjectName`
`AccountName`	`ApiKeyId` (if applicable)
`Action`	`ApiKeyName` (if applicable)
`ApplicationName`	

Retrieve Controller Audit Log Report

The Controller Audit Log Report is sent by email according to the addresses added to the configurations page. This report captures the following information:

- User logins and information changes
- Controller configuration changes
- Application properties and object changes such as policies, health rules, and entities listed in the above table.
- Environment properties changes

AppDynamics supports PDF, JSON, and CSV output formats.

Retrieve Controller Audit History via API

You can retrieve Controller audit history through the ControllerAuditHistory API method, which returns the configuration and user activities record in a JSON or CSV file for the time range specified. This information is the same as that found in the file.

Format

```
GET /controller/ ControllerAuditHistory?startTime=<start-time>&endTime=<end-time>&include=<field>:
<value>&exclude=<field>:<value>
```

For example:

```
http://localhost:8080/controller/ControllerAuditHistory?startTime=yyyy-MM-dd&&endTime=yyyy-MM-
dd&include=filterName1:filterValue1&include=filterName1:filterValue1&exclude=filterName1:
filterValue1&exclude=filterName1:filterValue1
```

```
curl --user user1@customer1:welcome "http://demo.appdynamics.com:8090/controller/ControllerAuditHistory?
startTime=2015-12-19T10:50:03.607-0700&endTime=2015-12-19T17:50:03.607-
0700&timeZoneId=America&Francisco&include=userName:user1&include=action:LOGIN&exclude=accountName:
system&exclude=action:OBJECT_UPDATE"

[{"timeStamp":1450569821811,"auditDateTime":"2015-12-20T00:03:41.811+0000","accountName":"customer1","
securityProviderType":"INTERNAL","userName":"user1","action":"LOGIN"},{"timeStamp":1450570234518,"
auditDateTime":"2015-12-20T00:10:34.518+0000","accountName":"customer1","securityProviderType":"INTERNAL","
userName":"user1","action":"LOGIN"},{"timeStamp":1450570273841,"auditDateTime":"2015-12-20T00:11:13.841+0000","
accountName":"customer1","securityProviderType":"INTERNAL","userName":"user1","action":"OBJECT_CREATED","
objectType":"AGENT_CONFIGURATION"},
...
{"timeStamp":1450570675345,"auditDateTime":"2015-12-20T00:17:55.345+0000","accountName":"customer1","
securityProviderType":"INTERNAL","userName":"user1","action":"OBJECT_DELETED","objectType":"
BUSINESS_TRANSACTION"},{"timeStamp":1450570719240,"auditDateTime":"2015-12-20T00:18:39.240+0000","accountName":"
customer1","securityProviderType":"INTERNAL","userName":"user1","action":"APP_CONFIGURATION","objectType":"
APPLICATION","objectName":"ACME Book Store Application"},{"timeStamp":1450571834835,"auditDateTime":"2015-12-
20T00:37:14.835+0000","accountName":"customer1","securityProviderType":"INTERNAL","userName":"user1","action

curl --user user1@customer1:welcome "http://127.0.0.1:8080/controller/ControllerAuditHistory?startTime=2019-05-
28T08:00:03.607-0700&endTime=2019-05-28T11:32:03.607-0700&timeZoneId=America%2FSan%
20Francisco&include=applicationName:ACME"
[{"timeStamp":1559066415823,"auditDateTime":"2019-05-28T18:00:15.823+0000","accountName":"customer1","
securityProviderType":"INTERNAL","userName":"user1","action":"LOGIN","objectId":0,"applicationName":"ACME"}]
```

Input parameters

Parameter Name	Parameter Type	Value	Mandatory
`start-time`	Query	Start time in the format: "yyyy-MM-dd'T'HH:mm:ss.SSSZ"	Yes
`end-time`	Query	End time in the format: "yyyy-MM-dd'T'HH:mm:ss.SSSZ"	Yes
`time-zone-id`	Query	Time zone	No
`include`	Query	Restricted information in the Controller audit history	No
`exclude`	Query	Restricted information in the Controller audit history	No

To control the size of the output, the range between the start-time and end-time cannot exceed twenty-four hours. For periods longer than 24 hours, use multiple queries with consecutive time parameters.

- Multiple filters of the same type are allowed.
- The backend API treats include filters with the same <field> and relationship as "OR", and filters with different <field> and relationship as "AND".
- There is no direct interaction between include and exclude filters.
- Each filter needs to be a parameter, e.g., `include=filterName1:filterValue1&include=filterName2:filterValue2`. See the below examples.

Log File Information by Platform

What is Audited

The following entries are audited:

ACCOUNT	HTTP_REQUEST_ACTION
ACCOUNT_ROLE	HTTP_REQUEST_ACTION_MEDIA_TYPE_CONFIG
ACTION_SUPPRESSION_WINDOW	HTTP_REQUEST_ACTION_PLAN_CONFIG
AGENT_CONFIGURATION	HTTP_REQUEST_DATA_GATHERER_CONFIG
ANALYTICS_DYNAMIC_SERVICE_HIERARCHICAL_CONFIGURATION	INFO_POINT
APPLICATION	JIRA_ACTION
APPLICATION_COMPONENT	JMX_CONFIG
APPLICATION_COMPONENT_NODE	MEMORY_CONFIGURATION
APPLICATION_CONFIGURATION	METRIC_BASELINE
APPLICATION_DIAGNOSTIC_DATA	MOBILE_APPLICATION
ASYNC_TRANSACTION_CONFIG	NODEJS_ERROR_CONFIGURATION
BACKEND_DISCOVERY_CONFIG	NOTIFICATION_CONFIG
BUSINESS_TRANSACTION	OBJECT_INSTANCE_TRACKING
BUSINESS_TRANSACTION_CONFIG	PHP_ERROR_CONFIGURATION
BUSINESS_TRANSACTION_GROUP	POJO_DATA_GATHERER_CONFIG
CALL_GRAPH_CONFIGURATION	POLICY
CUSTOM_ACTION	PYTHON_ERROR_CONFIGURATION
CUSTOM_CACHE_CONFIGURATION	RULE
CUSTOM_EMAIL_ACTION_PLAN_CONFIG	RUN_LOCAL_SCRIPT_ACTION
CUSTOM_EXIT_POINT_DEFINITION	SCHEDULED_REPORT
CUSTOM_MATCH_POINT_DEFINITION	SERVICE_ENDPOINT_DEFINITION
DASHBOARD	SERVICE_ENDPOINT_MATCH_CONFIG
DBMON_ACCOUNT_CONFIGURATION	SMS_ACTION
DBMON_CUSTOM_METRIC	SQL_DATA_GATHERER_CONFIG
DB_MONITOR_CONFIG	THREAD_DUMP_ACTION
DIAGNOSTIC_SESSION_ACTION	TRANSACTION_MATCH_POINT_CONFIG
DOT_NET_ERROR_CONFIGURATION	USER
EMAIL_ACTION	WORKFLOW
ERROR_CONFIGURATION	WORKFLOW_ACTION
EUM_CONFIGURATION	
EVENT_REACTOR	
GLOBAL_CONFIGURATION	
GROUP	

- The Audit report supports the Application Name for the above entities when applicable.
- To fetch the audit log for the `Remove literal` flag in the report, ensure to specify the object name as **Remove Literals** with the object type as **DBMON_ACCOUNT_CONFIGURATION**.

Supported Audit Actions

Below is the list of actions supported in auditing.

> Not all of these actions are supported for all of the Audit Entries listed in the preceding table.

ACCOUNT_REENABLED	OBJECT_CREATED
ACCOUNT_ROLE_ADD_PERMISSION	OBJECT_DELETED
ACCOUNT_ROLE_REMOVE_PERMISSION	OBJECT_UPDATED
ACKNOWLEDGE_GDPR_DATA_PRIVACY	SAML_AUTHENTICATION_CONFIG_CREATED
ANOMALY_DETECTION_CONFIG_CHANGED	SAML_AUTHENTICATION_CONFIG_DELETED
FLOW_ICON_MOVED	SAML_AUTHENTICATION_CONFIG_UPDATED
GROUP_ADD_ACCOUNT_ROLE	USER_ADD_ACCOUNT_ROLE
GROUP_REMOVE_ACCOUNT_ROLE	USER_ADD_TO_GROUP
LDAP_CONFIG_CREATED	USER_EMAIL_CHANGED
LDAP_CONFIG_DELETED	USER_PASSWORD_CHANGED
LDAP_CONFIG_UPDATED	USER_PASSWORD_RESET
LOG_LEVEL_CHANGED	USER_PASSWORD_RESET_COMPLETED
LOGIN	USER_REMOVE_ACCOUNT_ROLE
LOGIN_FAILED	USER_REMOVE_FROM_GROUP
LOGOUT	
LOGOUT_FAILED	

Troubleshoot Controller Issues

This page provides troubleshooting information for issues that may arise during Controller installation and operation.

Controller Server Log

The primary log file for the Controller at the following location:

```
<controller_home>/logs/server.log
```

The first step in troubleshooting Controller issues typically involves checking the log file. Search the log for errors that may correspond to the issue you are encountering. If found, an error log may help you identify and resolve the issue.

Also, see installation troubleshooting information in Custom Install.

Identify Controller Performance Issues

The following are indications of Controller performance issues:

1. The Controller UI performs slowly. For short time ranges, such as 15 or 30 minutes, responses that take longer than 10 to 20 seconds can indicate that your Controller is under stress.
2. When the Controller's metric reporting lags 7 to 10 minutes behind the current time, it can be an indication that your Controller is under stress. A lag of about 3 to 5 minutes is normal.
3. When monitoring the Controller environment, you see that CPU, memory, and disk metrics are at about 75% capacity.

If you observe degradation in Controller performance, it may be due to one of the following:

- The hardware resources for the Controller might not match the correct Controller profile.
- The Controller performance profile may be incorrectly configured.

To troubleshoot Controller performance issues:

1. Confirm that the hardware matches the Controller profile you use. For details see Controller System Requirements.
2. Confirm that your disk performance matches the recommended thresholds for minimum disk performance. For details see Controller System Requirements.
3. Confirm that the Java SDK version is exactly the same as the Java version on the Controller. To display the version of Java used by the Controller:
    - Open the command-line utility.
    - Go to `<Controller_Installation_Directory>/jre/bin`
    - Run `java -version`.

Monitor heap usage

- On Windows, use the Task Manager to measure the memory usage for the Controller.
- On Linux, use the **top** command to get statistics for the memory data.

```
ps -elf (expect to see a "java" process and a "mysql" process)

top (expect to see java and mysql with cpu greater then 0)
```

Timeout errors during Controller installation

While installing the Controller, the Enterprise Console attempts to start up the Controller application server and database. At first database startup, the application attempts to create the database schema, tables, and other artifacts needed by the Controller.

By default, the Enterprise Console waits 45 minutes for the Controller app server or database to start. When installing a medium or large profile Controller or into certain types of environments such as virtual machines, the time it takes to start up the system can exceed the default startup timeout period.

Controller does not start properly on Windows

Your Controller may not be starting due to file extensions of transaction logs created by Glassfish. Excluding the Controller data directory from being scanned by virus scanners as specified on Prepare Windows for the Controller does not account for these extent files found in the `<AppDynamicsInstall>\Controller\appserver\glassfish\domains\domain1\logs\server\tx` directory. When your antivirus detects these extensions, such as WRY, it may mistakenly stop the process of using these files so the Controller ultimately does not start.

These transaction logs are used to recover any failed Glassfish transactions, so deleting these logs on startup is not advised. Instead, configure your virus scanners to ignore the entire Controller directory.

No data in the Metrics Browser

This may indicate that the agents are not correctly configured. Begin troubleshooting by looking at the `server.log` file.

All log files for Controller are located in the `<Controller_Installation_Directory>/logs` folder.

Error Message	Solution
Error receiving metrics (node not properly modeled yet: Could not find component for node.	This error means the app agent tried to upload metric data for a specific node, but the node does not belong to any tier. Nodes must belong to tiers and these tiers must belong to a business application in order to receive metric data for that node. See Overview of Application Monitoring.
Received Metric Registration request for a machine that is NOT registered to any nodes. Sending back null!	This error indicates that the Controller received a registration request for metrics for a Machine Agent that listed a machine ID not yet associated with any node. Configure the Machine Agent to associate with the correct application, tier, and node. See Install the Machine Agent.
Agent upload blocked, as its reporting a time well into the future.	The App Agents attempt to report metric data using Controller time. The agents retrieve the time from the Controller every five minutes and report times using a skew of the local machine time, if different. If for some reason the App Agent reports metrics that are time-stamped ahead of the Controller time, the Controller rejects the metrics. To avoid this event, ensure that the system times for the machine on which the Controller is running and the machines for the app agents are in synchronization.

Controller shutdown does not increase free memory on Linux

You do not generally need to be concerned about the "free memory" value, as it will always trend towards zero. The Linux kernel tries to keep its cache as large as possible. As a result, the Linux kernel does not release the memory even after process termination. The memory is freed only if it is required by another process.

Controller process unexpectedly shut down

On Linux, memory allocation failures may cause the Controller process to be shut down unexpectedly by the Linux Out-of-Memory (OOM) Killer. The Controller log, `server.log`, does not provide information about the shutdown. Instead, to diagnose this event, check the system log (usually `/var/log/messages`) for "out of memory" entries written by the OOM killer, for example, as follows:

```
grep -i "Out of memory" /var/log/messages
```

If you encounter this log entry, make sure that you have allocated sufficient swap space on the Controller machine. AppDynamics recommends allocating a minimum of 10 GB of swap space.

Controller server swapping too often

If you encounter unexpected swapping on the Controller machine, you can configure how aggressive the operating system swaps by configuring the `swappiness` parameter. The `swappiness` parameter controls how often the Linux kernel moves processes out of physical memory and onto the swap disk. The default value for the parameter is usually 60. When you decrease the value, you lower the tendency of the operating system to swap. This results in less default file caching.

See the documentation for your Linux distribution for recommendations on the value for the `swappiness` parameter. For example, RedHat recommends setting swappiness to 10 for CentOS and RedHat kernels version 2.6.32-303 or later if you encounter OOM issues even though swap space is still available.

Before you configure the `swappiness` parameter though, ensure that the machine has sufficient RAM and that the buffer pool size for MySQL is properly configured.

To configure swappiness

1. Check the current value for `swappiness`.

   ```
   /sbin/sysctl -a | grep swappiness
   ```

2. Set the `swappiness` parameter.

   For example, add the following line to set the `swappiness` parameter to 10.

```
echo 10 > /proc/sys/vm/swappiness
```

3. Set the `swappiness` parameter in the `/etc/sysctl.conf` file to the same value you used in step 2.

   For example, add the following line to the `/etc/sysct1.conf` file:

```
vm.swappiness = 10
```

Could not determine the IP address of this host error during installation

During the installation process, the Enterprise Console attempts to ping the Controller by the hostname or IP address you enter. If the ping is unsuccessful during the user input validation, the following error message appears: "Could not determine the IP address of this host. Please ensure that the IP address of the Controller host resolves to its hostname or to localhost. You may need to add an entry in the hosts file on the Controller host and retry the operation."

To make the hostname resolvable, add an entry for it to the hosts file on the machine on which you are installing the Controller. On Linux, the hosts file is typically at `/etc/hosts`. On Windows, look for the file at the following location, `C:\Windows\System32\Drivers\etc\hosts`, or the location appropriate for your version of Windows.

Add the entry in the form of the following example:

```
127.0.0.1 localhost myhostname
```

Use the IP address and hostnames appropriate for your system.

For example, the following shows the entry added as the third line of the default RedHat hosts file:

```
127.0.0.1    localhost.localdomain localhost
::1          localhost6.localdomain6 localhost6
198.51.100.2 myhost myhost.example.org
```

Controller Cannot Connect to the MySQL Database

The following exception message in server.log file indicates that the Controller cannot connect to its embedded database.

```
*Server log exception:* "Caused by: java.net.ConnectException: Connection refused"
```

If you encounter this error, verify that the Controller database is running properly. On Linux, you can do so using one of the following commands:

Linux	Windows	Description
`lsof -i:3388`	SysInternals Process Explorer, will provide a list of files opened by process with pid 3388.	List open files opened by process with pid 3388.
`netstat -anp | grep 3388`	`netstat -ano | find "3388"`	List all networking ports opened by process with pid 3388.
`ps -aef | grep mysql`	`tasklist /v | find "mysql"`	Lists all processes and then checks if the process with name "mysql" is active and alive.

If no processes are found, it indicates that the Controller database was incorrectly terminated. Start the Controller database again and verify the Controller `server.log` file for any error messages.

Stack overflow exception when installing the Controller installation on Windows

This exception is usually caused when you set the `-Xss` option to a lower value. We recommend changing this value to 96000.

Triggering automatic collection of Controller logs

Use the following console commands to trigger automatic capture of Controller log files:

- On Linux, run:

```
bin/platform-admin.sh submit-job --platform-name test --service controller --job retrieve-log
```

- On Windows, open an elevated command prompt (in the Windows start menu, right-click the **Command Prompt** icon and choose **Run as Administrator**) and run:

```
bin/platform-admin.exe cli submit-job --platform-name test --service controller --job retrieve-log
```

  The logs will be copied in the Enterprise Console host under `platform-admin/logs-controller-<platform-name>-<date-time-stamp>.zip`.

See Platform Log Files to learn how to manage your Controller logs.

Collecting Troubleshooting Information for the Controller

If opening a support case for Controller troubleshooting, you can facilitate the diagnosis of the problem by providing the following information:

- Submit all `platform-admin/logs/*` and `platform-admin/logs-controller-*.zip`, in particular the `server.log` files. You can also use the log file utility described in Triggering automatic collection of Controller logs to collect logs.
- If the Controller runs out of memory, it generates a heap dump. Submit all files in `<controller_home>/appserver/glassfish/domains/domain1/config/hprof`.
- Submit all `<controller_home>/appserver/glassfish/domains/domain1/config/gc.log` files.
- Submit information about the hardware and operating system configuration of the machine that is currently hosting the Controller, including operating system, bit version, CPU cores, clock speed, disk configuration, and RAM.
- Indicate the Performance profile of Controller. Run the controller diagnosis command which captures the information in `platform-admin-server.log`:

  Refer to the Controller diagnostic data in the `platform-admin-server.log`. See a sample Controller diagnostic data on Manage a High Availability Deployment page.

Issues Generating Audit Reports Immediately after Upgrading the Controller to 4.5

When the Controller upgrade is complete, audit reports may not work immediately. The audit database table is getting migrated only after the upgrade process and the migration takes at least an hour to complete. If audit reports are run before completing the migration process, audit table migration messages are logged in the `server.log` file.

No actions are required, try running the audit reports again after an hour.

Controller Dump Files

The following steps describe how to collect troubleshooting information for your Controller. You may be requested for the information when troubleshooting with the AppDynamics support team.

Get Heap and Histogram Dump Files

It is recommended that you install JDK on your system before using the following commands.

- Get the **process id of the Controller** to use in subsequent commands.

```
ps -ef | grep java
```

- Get the **heap dump before garbage collection** using the following command:

```
<java-jdk-install-dir>/bin/jmap -dump:format=b,file=heap_before_live.bin <Controller_pid>
```

- Get the **histogram before garbage collection** using the following command:

```
<java-jdk-install-dir>/bin/jmap -histo <Controller_pid> | head -200 > histo_before_live.txt
```

- Get the **histogram after garbage collection** using the following command:

```
<java-jdk-install-dir>/bin/jmap -histo:live <Controller_pid> | head -200 > histo_after_live.txt
```

Take Four Thread Dumps at Three Second Intervals

- Using the Controller process ID, execute the following command:

```
kill -3 <Controller_pid>
```

- Save the `<Controller_Installation_Directory>/appserver/glassfish/domains/domain1/logs/jvm.log` file.

Send the Files to the AppDynamics Support Team

If asked to provide the information to the AppDynamics support team, send the following files generated by these steps:

- `heap_before_live.bin`
- `histo_before_live.txt`
- `histo_after_live.txt`
- `jvm.log`

Controller Component Versions

This page describes how to check version information and the version of bundled components. This information is useful when troubleshooting the system or performing other administrative tasks.

> ⚠ AppDynamics maintains and updates the bundled components as part of the AppDynamics platform. Do not attempt to upgrade a bundled component independently of the platform upgrade procedure.

Controller Version

You can retrieve the Controller version in two ways:

1. From the AppDynamics UI:
    a. Click the **Settings** ⚙.
    b. Select **About AppDynamics.**
    c. Note the **Controller build** number.
2. From the command line of the Controller machine:
    a. Access the `README.txt` file located in the Controller home directory.

Bundled Glassfish Server Version

The Glassfish server is installed in `<controller_home>/appserver`.

The Glassfish server version is Glassfish 4.1.1.

Bundled MySQL Database Version

The AppDynamics Controller uses MySQL as its default database, where it stores configuration data, metrics data, transaction snapshot data and events, and the history of incidents that occurred (both resolved and unresolved incidents are stored). The MySQL database files are installed in `<controller_home>/db` by default.

The latest AppDynamics release bundles MySQL version 5.7.33.

Check MySQL Version

To check what your MySQL version is in your Controller, you can run the following command:

```
<controller_home>/bin/controller.sh login-db
select version();
```

Upgrade MySQL Version

Optionally, after you install or upgrade the Controller, you can upgrade the MySQL version with the Enterprise Console. Note that you cannot reverse this process.

> ⓘ A newly installed 4.5 Controller packages and uses MySQL 5.7. However, a Controller that is upgraded to 4.5 from a previous version where MySQL 5.5 is used, will also use version 5.5.

You can upgrade the MySQL version on the Controller page in the GUI or with the following command:

Bundled Java Version

The Controller bundles and uses Java Runtime Environment 8.0.282.

Enterprise Console Version

Run the command in the `<Enterprise Console installation directory>/platform-admin` to check the version:

Other Component Versions

See Legal Notices for the latest components included in the AppDynamics products and modules.

Upgrade the Controller Using the Enterprise Console

Related pages:

- Controller Data Backup and Restore
- Upgrade a Single Controller
- Upgrade an HA Pair

You can upgrade a Controller instance using the Enterprise Console. The Enterprise Console simplifies the upgrade process by allowing you to discover and upgrade single Controllers and HA-pairs.

About the Upgrade

- The Enterprise Console supports Controller upgrades (standalone and HA-pair) starting from 4.4.x and higher, to the latest version. Use the following table to determine your course of action based on your circumstances:

If your current Controller version is...	Controller version you want to upgrade to...	Actions to take...
Older than version 4.4.x (such as: 4.1.*x*, 4.2.*x, 4.3.x*)	Controller version 4.5.*x* or the latest version	1. Open a support help desk ticket. 2. Support will provide you a 4.3.8 installer for you to use in the upgrade process. 3. Use the Controller installer and upgrade the Controller to version 4.3.8. 4. Then, use Enterprise Console version 4.4.3 to upgrade the Controller from version 4.3.8 to version 4.4.3. 5. Finally, use Enterprise Console version 4.5.*x* to upgrade the Controller from version 4.4.3 to version 4.5.*x*.
Equal to version 4.4.x or later	Controller version 4.5.x or the latest version	1. Access the downloads portal to download the Controller version. 2. Use the Enterprise Console to upgrade from your existing Controller version to the Controller version you want.

- Discovering and upgrading a Controller from 4.1 and higher to the latest version will not upgrade the Events Service automatically. This must be done by a separate upgrade job.
- You can choose which version you would like to upgrade the Controller to as long as the Enterprise Console is aware of that version. This means that you can upgrade the Controller to any intermediate version or to the latest version as long as the Enterprise Console installer has been run for those versions. However, you cannot upgrade the Controller to an older version.
- If you have a license for the older version, the license should work when upgrading the Controller to a new version. However, if you have a temporary license for the old version and now have a new license, the new license will not work on the old Controller. In this case, you should upgrade the Controller to the latest version before applying the license.
- An upgrade results in Controller downtime, but it is not necessary to stop agents during the Controller upgrade.
- Starting in 4.5, during a Controller discover and upgrade scenario, the Enterprise Console expects a `.passwordfile` file to be present in the Controller home directory. The Enterprise Console reads this password and validates it against the Controller. Once the upgrade is complete, the Enterprise Console removes the file, and stores the password in its encrypted database.

> ⚠ If you change the Glassfish Admin Password manually, you also need to update it in the Enterprise Console Controller Settings.

Before Upgrading

- Before you start upgrading the Controller, make sure that you are using the correct update order. Remember, if you are upgrading from a pre-4.1 version to 4.4.x or the latest version, you must first manually upgrade any embedded Events Service, then upgrade the Controller.
- Review the latest Release Notes and the release notes for any intermediate versions between the current version of your instance and the version you are targeting to learn about issues and enhancements in those releases.
- Check the most recent Controller System Requirements and Troubleshoot Controller Issues to review your Controller's current workload and determine whether you need to change your performance profile and increase your hardware resources, if necessary.

> ⓘ You may change your Controller profile on the Platform Configurations pages of the Enterprise Console GUI, either before or after you upgrade your Controller. This process is not reversible, and you cannot move from a larger to a smaller profile size.

- Check the Controller's database.log for any errors. You can find the log at `<controller_home>/db/logs/database.log`. There should not be any `InnoDB: Error` lines in the log. If any errors are found, please reach out to AppDynamics Support before attempting the upgrade. Upgrading the Controller with a corrupt database may put the Controller in a bad state, with high recovery time.
- If you changed any Glassfish settings that are not JVM options, note your changes. You may need to configure them after an upgrade. The Enterprise Console recognizes and retains many common customizations to the domain.xml, db.cnf, and other configuration files, but is not

guaranteed to retain them all. If you have made manual configuration changes to the files, verify the configuration after updating. See Retaining Configuration Changes to learn how to preserve changes.
- If you uninstall the Enterprise Console that was previously managing the Controller and use a new Enterprise Console instance to discover and upgrade the Controller, you need to first manually create the `.passwordfile` file in order for the Enterprise Console to continue with the discover and upgrade process. You can create the file in the Controller home directory, and add the `AS_ADMIN_PASSWORD=<controllerRootUserPassword>` value in it.
- When performing an upgrade while enabling HTTPS, check that the hostname does not start with a digit. The upgrade will fail if the Enterprise Console hostname starts with a digit due to a JDK limitation with the DNS name in CN/SAN.
- The database user password storage mechanism changed in Release 3.8. If upgrading from 3.7.x or earlier, see the Upgrade the Controller page in the 3.8 documentation for considerations related to the database user password.
- Back up the existing Controller following the steps in the next section.

Back up the Existing Controller

The Enterprise Console retains agent data, reports, configuration, and other types of data through an upgrade, including a copy of commonly modified configuration files under `<controller_home>/backup`. Nevertheless, to mitigate the risk of data loss in the event of an unexpected failure, be sure to back up the existing installation directory before applying an upgrade to the Controller.

The Enterprise Console does not back up MySQL data. You need to back up the data before upgrading standalone installations.

If the upgrade does not finish successfully for any reason, see Troubleshooting the Upgrade for more information.

To back up the Controller instance

1. Stop the Controller application server and database.
   - For Linux, run:

     ```
     platform-admin.sh stop-controller-appserver
     ```

   - For Windows, run:

     ```
     platform-admin.exe cli stop-controller-appserver --with-db
     ```

2. Back up the Controller home by copying the entire Controller home directory to a backup location. Note the following points:
   - If the data home for the Controller is not under the Controller directory, be sure to back up the database directory as well.
   - If it's not possible to back up the entire data set, you can selectively back up the most important tables. Use the Metadata Backup SQL script described and attached to the Controller Data Backup and Restore page.
3. Restart your Controller after completing the backup and before upgrading.

After Upgrading

- If you have configured settings in the `domain.xml` file, `db.cnf` or other configuration files manually or by using the Glassfish `asadmin` utility, verify those changes in the configuration files or Controller Configurations page of the GUI after the upgrade and re-apply any customizations that were not preserved. The Enterprise Console preserves several recommended customizations. After the upgrade, you can find backup copies of common configuration files in the `<controller_home>/backup` directory.
- As an optional step, your MySQL version can be upgraded after you upgrade the Controller, through the Enterprise Console GUI or by using the `mysql-upgrade` job. See Bundled MySQL Database Version for more information.

Troubleshooting the Upgrade

If the upgrade does not succeed for any reason, the Enterprise Console does not roll back changes on disk. This gives you an opportunity to diagnose and troubleshoot the issue before reattempting the installation or upgrade.

To troubleshoot the issue, check the installation log at `platform-admin/logs/platform-admin-server.log`. The Controller `server.log` file may contain additional information.

Resuming from Checkpoint

The Enterprise Console provides a feature to resume the upgrade from the last point of failure (checkpoint). The application creates a checkpoint at several stages during installation. If the installation fails, resuming from checkpoint would skip all the prior successfully completed stages and restart the installation from the beginning of the specific stage where the last point of failure occurred.

You can also resume a failed Controller job from the CLI by passing the flag `useCheckpoint=true` as an argument after `--args` in your command.

If for some reason, the upgrade from the last checkpoint is not successful, you may retry the upgrade from the beginning. Simply click **Retry** instead of Resume from the checkpoint. However, please note that retrying from the beginning after a failed upgrade may overwrite your customizations to `db.cnf` and `domain.xml`.

Timing Out

A common upgrade issue involves the upgrade process timing out. The Enterprise Console attempts to restart the Controller and database after applying an update or installation. For large databases and depending on the system resources, this can take a considerable amount of time. If the Enterprise Console cannot finish starting up the Controller within a set time-out period (30 minutes by default), the operation will fail.

You can increase the default time out period for system startup. The timeouts are defined in `platform-admin/archives/controller/<version>/playbooks/controller.groovy`. You can update the `controllerStartRetryTimeout = 10 * 60 seconds = 10 minutes`, and then retry the upgrade from the checkpoint.

When the Controller upgrade is complete, audit reports may not work immediately. The audit database table is getting migrated only after the upgrade process and the migration takes at least an hour to complete. If audit reports are run before completing the migration process, audit table migration messages are logged in the server.log file. No actions are required, try running the audit reports again after an hour.

Upgrade a Single Controller

Related pages:

- Controller Data Backup and Restore
- Upgrade the Controller Using the Enterprise Console

You can use the Enterprise Console to onboard and upgrade a single node Controller instance. The **Custom Install Discover & Upgrade** option in the GUI allows you to create a platform and discover a Controller.

Alternatively, if you have already created a platform, you must add credentials and hosts to the platform before you can perform discovery.
Then, discover the Controller on the page. Discovering a Controller means that the Enterprise Console learns about your existing Controller deployment, such as profile, tenancy mode, existing domain configuration, and database configuration. This information is used to perform an upgrade.

About the Upgrade

The Enterprise Console supports Controller upgrades (standalone and HA-pair) starting from 4.4.x and higher, to the latest version. Use the following table to determine your course of action based on your circumstances:

If your current Controller version is...	Controller version you want to upgrade to...	Actions to take...
Older than version 4.4.x (such as: 4.1.*x*, 4.2.*x, 4.3.x*)	Controller version 4.5.*x* or the latest version	1. Open a support help desk ticket. 2. Support will provide you a 4.3.8 installer for you to use in the upgrade process. 3. Use the Controller installer and upgrade the Controller to version 4.3.8. 4. Then, use Enterprise Console version 4.4.3 to upgrade the Controller from version 4.3.8 to version 4.4.3. 5. Finally, use Enterprise Console version 4.5.*x* to upgrade the Controller from version 4.4.3 to the latest version.
Equal to version 4.4.x or later	Controller version 4.5.x or the latest version	1. Access the downloads portal to download the Controller version. 2. Use the Enterprise Console to upgrade from your existing Controller version to the Controller version you want.

Upgrade the Controller Using GUI

If there is a Controller upgrade available, you can begin the upgrade process either on the Custom Install or Controller page in the GUI.

> Ensure that the Controller and database are running prior to the upgrade. The Enterprise Console validates the database root password and Controller root passwords provided during the upgrade.

Upgrade the Controller from 4.1.x, 4.2.x, and 4.3.x to 4.5.x or Latest

To upgrade the Controller from 4.1.*x*, 4.2.*x*, and 4.3.*x* to 4.5.*x* or the latest version, you can use the Discover and Upgrade feature:

1. Open a support help desk ticket. Support will provide you a 4.3.8 installer for you to use in the upgrade process.
2. Use the Controller installer and upgrade the Controller to version 4.3.8.
3. Check that you have fulfilled the Enterprise Console prerequisites before starting.
4. Open a browser and navigate to the GUI:

   ```
   http(s)://<hostname>:<port>
   ```

   9191 is the default port.
5. Navigate to the **Install** homepage and select **Custom Install**.
6. Name the Platform:

   a. Enter a Name and the Installation Path for your platform.

   > The Installation Path is an absolute path under which all of the platform components are installed. The same path is used for all hosts added to the platform. Use a path which does not have any existing AppDynamics components installed under it. The path you choose must be writeable (the user who installed the Enterprise Console should have write permissions to that folder). Also, the same path should be writable on all of the hosts that the Enterprise Console manages.
   >
   > Example path: `/home/appduser/appdynamics/product`

7. Add a Host:

   > ⚠ If the Controller is running on a Windows machine, the Enterprise Console should be on the same machine. Windows hosts are not supported on the Enterprise Console.

   a. Enter the host machine-related information: Host Name, Username, and Private Key. This is where the Controller will be upgraded. Therefore, this needs to point to the host machine where the Controller is currently up and running. For more information about how to add credentials and hosts, see Administer the Enterprise Console.
8. Discover Controller:
   a. Select **Discover & Upgrade**.
   b. Select an available Target Version from the dropdown.

      > ⓘ The list is populated by versions that the Enterprise Console is aware of. Of those versions, the list will only show versions that are the same or greater than the current Controller version.

   c. Enter the Controller Primary Host.
   d. Enter the Controller root password and database root password.

      > ⓘ Your Controller and MySQL needs to be up and running before performing an upgrade. The Enterprise Console validates whether the input passwords are correct.
9. Make sure to unselect the Install Events Service option before selecting **Submit**.
10. Select **Submit**.

The Enterprise Console will onboard the controller on the selected host machine to the application build. When the Enterprise Console discovers a component, it also checks to see if an upgrade is available and performs the upgrade. Plan for a downtime of the Controller availability during this time. You can view the status of the upgrade job on the **Jobs** page.

Once the upgrade is complete, the URL of the controller and other controller related information can be accessed from the Controller page.

Then, repeat Steps 3 through 10 and use Enterprise Console version 4.4.3 to upgrade the Controller from version 4.3.8 to version 4.4.3.

Finally, repeat Steps 3 through 10 and use Enterprise Console version 4.5.*x* to upgrade the Controller from version 4.4.3 to version 4.5.*x*.

Upgrade the Controller from 4.4.x to Latest

To upgrade the Controller from 4.4.x to the latest version, you can use the Upgrade Controller feature:

1. Check that you have fulfilled the Enterprise Console prerequisites before starting.
2. Upgrade the Enterprise Console to the latest version.
3. Open a browser and navigate to the GUI:

   ```
   http(s)://<hostname>:<port>
   ```

   9191 is the default port.
4. Navigate to the **Controller** page of the platform.
5. Select the Controller host you would like to upgrade.
6. Select **Upgrade Controller**.
7. Select an available Target Version from the dropdown.

   > ⓘ The list is populated by versions that the Enterprise Console is aware of. Of those versions, the list will only show versions that are the same or greater than the current Controller version.

8. Enter the required passwords and select **Submit**.

Upgrade the Controller Using CLI

If there is a Controller upgrade available, you can begin the upgrade process using the application CLI.

> ⓘ Ensure that the Controller and database are running prior to the upgrade. The Enterprise Console validates the database root password and Controller root passwords provided during the upgrade.

Upgrade the Controller from 4.1.x, 4.2.x, and 4.3.x to 4.5.x or Latest

To upgrade the Controller from 4.1.x, 4.2.x, and 4.3.x to 4.5.x or the latest version:

1. Open a support help desk ticket. Support will provide you a 4.3.8 installer for you to use in the upgrade process.
2. Use the Controller installer and upgrade the Controller to version 4.3.8.
3. Use the Enterprise Console CLI commands to:
    a. Create a platform:

    ```
    bin/platform-admin.sh create-platform --name <platform_name> --installation-dir
    <platform_installation_directory>
    ```

    b. Create a credential:

    ```
    bin/platform-admin.sh add-credential --credential-name <name> --type <ssh> --user-name <username>
    --ssh-key-file <file path to the key file>
    ```

    > Remember to provide the private key file for the Enterprise Console machine when adding a credential.

    c. Add hosts:

    ```
    bin/platform-admin.sh add-hosts --hosts host_1 host_2 host_3 --credential <credential name>
    ```

    d. Discover and upgrade the Controller. For the option `--target-version`, either specify the desired upgrade version of the Controller or `latest` to install the latest Controller version available to the Enterprise Console.

    If you updated the Database user password from the default, you can provide it after `--args` by including the parameter `newDatabaseUserPassword=<password>`. The job uses the default password if this parameter is excluded. If you have forgotten your Database user password and need to reset it, click Reset the Controller DB User Password.
    If your upgrade fails, you can resume by passing the flag `useCheckpoint=true` as an argument after `--args.`

    > You must specify and provide the full path to the existing Controller directory.

4. Confirm that the installation directory is the same as the previous Controller installation directory on this machine. The Controller will automatically migrate the data only when the `discover_upgrade` option is specified.
   The Enterprise Console completes the upgrade and restarts the Controller.
5. If you made any manual changes to the configuration files listed, verify those changes to the equivalent files in the upgraded instance. The Enterprise Console preserves `domain.xml` customizations and database `cnf` customizations.
6. Open a browser and access the AppDynamics user interface:

    ```
    http://<Controller_Host>:<Controller_Port>/controller
    ```

    If the UI does not display the new Controller, refresh your browser cache to view the new UI.

Then, repeat Steps 3 through 6 and use Enterprise Console version 4.4.3 to upgrade the Controller from version 4.3.8 to version 4.4.3.

Finally, repeat Steps 3 through 6 and use Enterprise Console version 4.5.*x* to upgrade the Controller from version 4.4.3 to version 4.5.*x*.

Upgrade the Controller from 4.4.x to Latest

Upgrades from 4.4.x to the latest version can be performed on the Controller page of the Enterprise Console or with the following commands:

1. Upgrade the Enterprise Console to the latest version.
2. Navigate to the `<Enterprise Console home directory>/platform-admin directory`.
3. If it has been more than one day since your last session, you will have to log in with the following command:

    ```
    bin/platform-admin.sh login --user-name <admin_username> --password <admin_password>
    ```

4. Apply the upgrade to the Controller with the following command:

    If your upgrade fails, you can resume by passing the flag `useCheckpoint=true` as an argument after `--args.`

How to Reset the Database User Password

You can customize the database user password. However, if you are upgrading your system, you may have forgotten the password. How you reset the database user password depends on whether the Enterprise Console has discovered the Controller.

If the Enterprise Console has discovered the Controller

Use the Enterprise Console CLI commands to:

1. Log in to the Enterprise Console.

   ```
   platform-admin/bin/platform-admin.sh login --user-name admin --password EC_GUI_PASSWORD
   ```

   Replace `EC_GUI_PASSWORD` with your actual value.
2. Reset the database user password.

   ```
   platform-admin/bin/platform-admin.sh submit-job --platform-name <platform_name> --service controller --
   job update-passwords --args newDatabaseUserPassword=<password>
   ```

   Replace `<platform_name>` and `<password>` with your actual values.

As a result, an Enterprise Console job is generated where you can verify the success of the password reset.

If the Enterprise Console has NOT discovered the Controller

1. Log in to the database as the root user by running the following command:

   ```
   ./mysql --user=root -p --host=127.0.0.1 --port=3388 --protocol=TCP
   ```

2. Execute the following queries, replacing `<new_password_here>` before executing the query.

   ```
   update mysql.user set authentication_string=password('<new_password_here>') where user like
   'controller%'; flush privileges; quit;
   ```

3. Verify the login by running the following command in the `<controller_home>/db/bin` directory:

   ```
   ./mysql -u controller -p -P 3388 -h 127.0.0.1
   ```

Upgrade an HA Pair

Related pages:

- Upgrade the Controller Using the Enterprise Console
- Controller High Availability
- Controller Data Backup and Restore

You can use the Enterprise Console to onboard and upgrade an HA Controller pair. For HA pairs that are not managed by the Enterprise Console, use the discover and upgrade option; for HA pairs that are managed by the Enterprise Console, you must use the upgrade option.

This topic describes:

- Upgrade method benefits
- Available upgrade methods
- How to shut down the HA Toolkit (HATK) implementation

Upgrade Method Benefits

The upgrade method enables you to:

- Perform a set of pre-upgrade validations. A summary of validation errors is provided to you before you modify the system's state.
- Quickly restore the older Controller version from the preserved secondary version in case of any upgrade issues. As you upgrade the primary server, the secondary server is isolated, thereby providing you a backup from which to quickly restore the service.

Available Upgrade Methods

Use the CLI, or follow a step-by-step Upgrade wizard that guides you through the UI, to perform the upgrade.

Use the CLI Method to Upgrade

You can use the CLI to upgrade the HA Controller pair. For more details, select upgrade using the CLI.

Use the Upgrade Wizard Method to Upgrade

You can use the Upgrade Wizard to follow a step-by-step procedure that guides you through the UI to perform the upgrade. For more details, select upgrade using the Upgrade Wizard.

Shut Down the HA Toolkit (HATK) Implementation

Before you start the Controller HA upgrade from a pre-Enterprise Console deployment, you must first shut down the watchdog and assassin processes. This is only required if you have installed the HA Toolkit previously.

To shut down the HA Toolkit implementation:

- If the Controller services are installed with privilege escalation using setups option (-c option in install-init.sh) then running the following command on the secondary will stop the secondary appserver, watchdog, and assassin.

  ```
  /sbin/appdservice appdcontroller stop
  ```

  or
- If the Controller services are installed with privilege escalation using sudoers option (`-s` option in `install-init.sh`) then running the following command on the secondary will stop the secondary appserver, watchdog, and assassin.

  ```
  sudo /sbin/service appdcontroller stop
  ```

To check if the appserver, watchdog, and assassin stopped on the secondary, you can use the following commands based on the privilege escalation method used to install the Controller services:

```
/sbin/appdservice appdcontroller status
```

or

```
sudo /sbin/service appdcontroller status
```

For both stopping and checking the statuses, if you do not remember the privilege escalation method used to install services, then you can use both variants, one after the other, in any order.

Upgrade the HA Controller Pair Using the CLI

If are using the HA ToolKit (HATK) and made any customizations to it, AppDynamics recommends that you review your particular situation and determine if you should proceed with the migration. For more details, see HA module in the Enterprise Console.

Steps 1 through 3 consist of different procedures depending on your HA Controller pair deployment:

- Follow **Option 1 - Discover and Upgrade** for deployments that are not managed by the Enterprise Console.
  Use the discover and upgrade job to onboard your HA Controller pair to the Enterprise Console before upgrading the primary Controller.
- Follow **Option 2 - Upgrade** for HA pair deployments that are managed by the Enterprise Console.
  Use the upgrade option to upgrade your primary Controller managed by the current Enterprise Console instance.

To upgrade using the CLI:

Step 1: Prepare the Upgrade

Step 2: Perform a Failover and Check the HA Pair State

To begin the upgrade, perform a failover to the secondary Controller. Since the primary Controller is known to be working, this provides a stable configuration to fail back to in case of upgrade issues.

To perform a failover and check the HA pair state:

Step 3: Upgrade the Primary Controller

You can upgrade the primary Controller using one of the following commands:

See Troubleshooting the Upgrade if the primary upgrade fails.

Step 4: Verify the Primary Upgrade

A primary upgrade success message displays if the upgrade is successful. However, AppDynamics recommends that you perform your own in-house verification on the Controller before proceeding to the secondary upgrade. For example, you can check that your agents are continuing to report to your Controller.

The following describes the expected state in the Enterprise Console.

Run the following commands on the Enterprise Console host:

```
bin/platform-admin.sh list-platform-service --platform-name <name_of_the_platform>
```

The desired output is `Controller: primary_upgraded`.

```
bin/platform-admin.sh list-node --service controller --platform-name <name_of_the_platform>
```

The output will display the host versions.

Sample Output

```
Available nodes in the controller service:
        Controller: role Primary, host 172.12.0.1, version 4.5.5.0, state running
         MySQL: role Primary, host 172.12.0.1, version 5.7.24 state running
        Controller: role Secondary, host 172.12.0.2, version 4.5.0.2, state stopped
         MySQL: role Secondary, host 172.12.0.2, version 5.7.21, state running
```

Check that all hosts show the new versions, that the Controller is running on the primary and stopped on the secondary, and that MySQL is running on both hosts. If you are not satisfied with the upgrade, see Verify the Primary Upgrade is Unsatisfactory.

Step 5: Upgrade the Secondary Controller

To upgrade the secondary Controller, run the the `upgrade-secondary` command on the Enterprise Console host:

```
bin/platform-admin.sh submit-job --service controller --job upgrade-secondary --platform-name
<name_of_the_platform> --args controllerRootUserPassword=<controller_root_password>
mysqlRootPassword=<mysql_root_password>
```

If the secondary Controller upgrade fails, see Upgrade the Secondary Controller Fails for possible recovery options.

Step 6: Verify the Secondary Upgrade

A secondary upgrade success message displays if the upgrade is successful. However, AppDynamics recommends that you perform your own in-house verification on the Controller before completing the upgrade.

The following describes the expected state in the Enterprise Console.

Run the following commands on the Enterprise Console host:

```
bin/platform-admin.sh list-platform-service --platform-name <name_of_the_platform>
```

The desired output is "Controller: provisioned".

```
bin/platform-admin.sh list-node --service controller --platform-name <name_of_the_platform>
```

The output will display the host versions.

Sample Output

```
Available nodes in the controller service:
        Controller: role Primary, host 172.12.0.1, version 4.5.5.0, state running
         MySQL: role Primary, host 172.12.0.1, version 5.7.24 state running
        Controller: role Secondary, host 172.12.0.2, version 4.5.5.0, state stopped
         MySQL: role Secondary, host 172.12.0.2, version 5.7.24, state running
```

Ensure that the secondary host version has been upgraded. Also, the primary should be in a running state, while the secondary Controller appserver should remain stopped. However, its MySQL process should be running.

Troubleshooting the Upgrade

Failover Issues

If you experience failover issues before upgrading, determine the condition of the secondary Controller. You may need to fix a broken secondary Controller before you attempt an upgrade.

The Primary Controller Upgrade Fails

Verify the Primary Upgrade is Unsatisfactory

If the primary upgrade succeeded, but you discovered problems during manual verification, you can roll back the upgrade by performing a failover and rebuilding the secondary.

If the downtime maintenance window is closing, you need to restore the older deployment version and service. Due to the recently performed failover, the current secondary host is a known-good host because it had been functioning as the primary host. You can quickly restore service by failing over to it, and then repairing the host (which experienced the failed upgrade) by rebuilding it as a secondary host.

From the Enterprise Console host:

1. Enter the `ha-failover` command to revert the primary host to the older version:

```
bin/platform-admin.sh submit-job --service controller --job ha-failover --platform-name
<name_of_the_platform> --args forceFailover=true
```

2. Enter the `incremental-replication` command to reduce downtime. When dealing with large data, this is recommended because replication time and downtime depend on data size.

```
bin/platform-admin.sh submit-job --service controller --job incremental-replication --platform-name
<name_of_the_platform>
```

3. Enter the `finalize-replication` command to rebuild the secondary host:

```
bin/platform-admin.sh submit-job --service controller --job finalize-replication --platform-name
<name_of_the_platform>
```

Upgrade the Secondary Controller Fails

If you receive the following error confirming a failed secondary upgrade:

```
Controller: secondary_upgrade_error
```

You can try the following recovery options:

- Option 1: If the failure is recoverable, you can retry the upgrade by entering the following command on the Enterprise Console host:

```
bin/platform-admin.sh submit-job --service controller --job upgrade-secondary --platform-name
<name_of_the_platform> --args controllerRootUserPassword=<controller_root_password>
mysqlRootPassword=<mysql_root_password> useCheckpoint=true
```

- Option 2: If retrying the upgrade does not work, you can run the incremental-replication and finalize-replication jobs. This involves downtime on the primary. Enter the following commands on the Enterprise Console host:

```
bin/platform-admin.sh submit-job --service controller --job incremental-replication --platform-name
<name_of_the_platform>
bin/platform-admin.sh submit-job --service controller --job finalize-replication --platform-name
<name_of_the_platform>
```

  For more details, see Initiate Controller Database Incremental Replication.
- Option 3: If the machine dies or cannot be fixed by running the incremental-replication and finalize-replication jobs, you can remove the secondary Controller. This is a unique fix for an uncommon situation. Enter the following command on the Enterprise Console host:

```
bin/platform-admin.sh submit-job --service controller --job remove --platform-name
<name_of_the_platform> --args entireCluster=false removeBinaries=true
```

  > The flag, removeBinaries=true, is optional. The use of this flag depends on the situation of your deployment.
  > If `removeBinaries=false` then the Enterprise Console forgets the Controller without impacting or uninstalling the Controller.

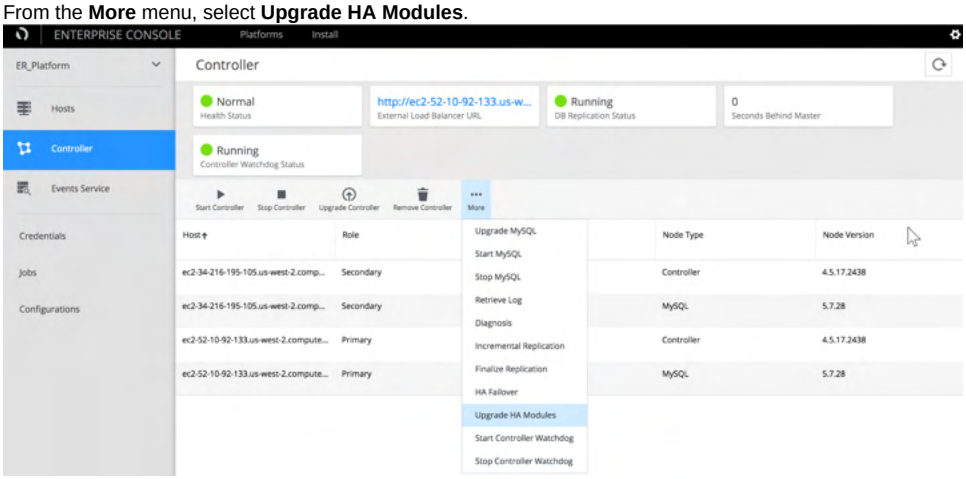
How to Upgrade the HA Modules Without Upgrading the Controller

Using the Enterprise Console UI or the CLI, you can upgrade the HA modules without having to upgrade the Controller. When both Controllers are managed by the Enterprise Console, the HA module is automatically upgraded when you upgrade your HA Controllers.

The following procedure describes an example scenario:

1. You are currently running version 4.5.13 of both Enterprise Console and Controller.
2. You activate the HA modules.
3. You then perform an upgrade only for the Enterprise Console to the latest version (4.5.15 or later).
4. You can upgrade the HA modules from either the Enterprise Console UI or the CLI:

    - From the Enterprise Console UI:
        a. Log in to the Enterprise Console and access the Controller page.
        b. From the **More** menu, select **Upgrade HA Modules**.



    - From the CLI, run the following command on the Enterprise Console host to upgrade the HA modules:

```
bin/platform-admin.sh submit-job --job upgrade-ha-modules --service controller
```

The HA modules are upgraded to the latest version without upgrading the Controller. When you upgrade the HA modules, no downtime is required on the Controller, and all HA settings are preserved.

Upgrade the HA Controller Pair Using the Upgrade Wizard

You can use the HA Controller Upgrade Wizard once both of your Controllers (primary and secondary) have been onboarded into the Enterprise Console.

> If are using the HA ToolKit (HATK) and made any customizations to it, AppDynamics recommends that you review your particular situation and determine if you should proceed with the migration. For more details, see HA module in the Enterprise Console.

> Upgrading the HA Controller pair consists of different procedures based on your deployment:
>
> - Follow **Option 1 - Discover and Upgrade** for deployments that are not managed by the Enterprise Console.
>   Use the discover and upgrade job to onboard your HA Controller pair to the Enterprise Console before upgrading the primary Controller.
> - Follow **Option 2 - Upgrade** for HA pair deployments that are managed by the Enterprise Console.
>   Use the upgrade option to upgrade your primary and secondary Controllers managed by the current Enterprise Console instance.

To upgrade using the Upgrade Wizard:

Step 1: Prepare the Upgrade

Step 2: Enter Controller Credentials

Step 3: Perform a Failover

Step 4: Upgrade the Primary Controller

See Troubleshooting the Upgrade if the primary upgrade fails.

Step 5: Verify the Primary Controller Upgrade

This step pauses the upgrade process and allows you to manually verify the new Controller version.

> Before you proceed with upgrading the secondary Controller, AppDynamics recommends that you verify that the Controller is working by logging in to the Controller and checking that metrics have been received within the last five minutes. If you are not satisfied with the upgrade, you can roll back to the older version from which you started.

Step 6: Upgrade the Secondary Controller

Troubleshooting the Upgrade

Fail Over and Rebuild Secondary

This completes rebuilding the current secondary server so both Controllers in the HA pair are now replicating data.

The Secondary Controller Upgrade Fails

When the secondary Controller upgrade fails:

How to Improve and Optimize Controller Database Performance

> The following procedure only applies to those Controllers that have been upgraded from version 4.2.8 or earlier, to a later version. In such cases, after you upgrade to later version, you may experience performance issues with the Controller database. However, for all new Controller installations using version 4.2.9 or later, the metric data tables are optimized.

To improve database performance when querying metrics, the primary key used by the metric data tables is read *optimized*. As a result, the primary key changes as follows:

From	To
ts_min, node/tier/app, metric_id	metric_id, node/tier/app, ts_min

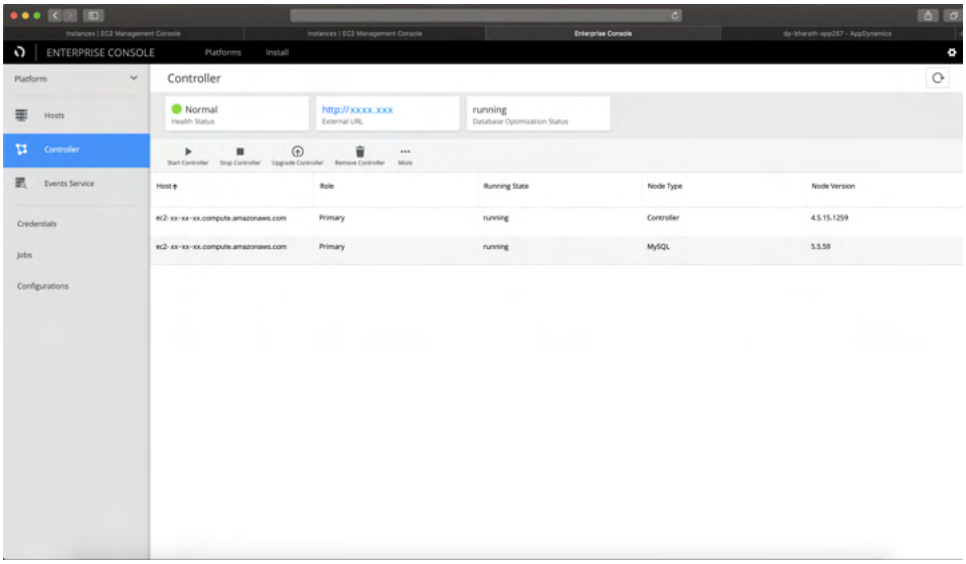
How to Run Database Optimization

You can use the Enterprise Console to run a database optimization job to optimize your database performance. To use this feature:

1. Upgrade your Enterprise Console to the latest version.
2. Upgrade your Controller to the latest version.
3. After the upgrade has completed, for any database table that can be optimized, you can run the database optimization job from the Enterprise Console.

   > No downtime is required and the database optimization job runs automatically.

   Select **Start Database Optimization** from the Controller page to start a process that runs in the background on your primary Controller host.

   The process performs several pre-checks to determine if there is enough disk space, and if any other database optimization process is running. The amount of disk space required is determined by the size of the tables to optimize. Based on the amount of Controller data, the database optimization job may take several hours to several days to complete.
4. Once all of the tables have been optimized successfully, the database optimization process completes and no longer displays on the page. To verify that all tables have been optimized, enter and run the following query:

```
cd <controller_home>/bin directory
./controller.sh login-db
        mysql>SELECT table_name
        FROM   information_schema.key_column_usage
        WHERE  table_name LIKE 'metricdata%'
     AND table_name != 'metricdata_min'
     AND table_name != 'metricdata_min_agg'
     AND column_name = 'ts_min'
     AND ordinal_position = 1;
```

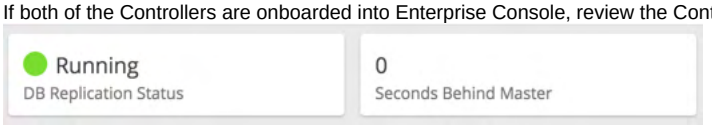
If the query returns any results, then those tables have not been optimized.
If the query returns zero records, then all of the tables were optimized successfully.

The database optimization job is supported on Linux OS only.

How to Run Database Optimization on a High Availability (HA) Controller Pair

Before you run database optimization on a HA Controller pair, you must ensure that the Controller database replication is in a healthy state.

- If both of the Controllers are onboarded into Enterprise Console, review the Controller page and note the following fields:



- If one of the Controllers is managed by HA toolkit (HATK):

a. Log in to the primary Controller host and enter:

```
cd <controller_home>/bin directory
```

b. Log in to the secondary Controller database and enter:

```
./controller.sh login-db
```

c. Enter:

```
SHOW SLAVE STATUS\G;
```

This results in the following output:

```
Seconds_Behind_Master: $Number_Of_Seconds_Behind_Master
```

If a non-zero number displays the output for this test, wait until the number changes to zero.

d. After you ensure that replication is working as expected, you can run the database optimization job from the Enterprise Console. Select **S tart Database Optimization** from the Controller page to start a process that runs in the background on your primary Controller host. The process performs several pre-checks to determine if there is enough disk space, and if any other database optimization process is running. The amount of disk space required is determined by the size of the tables to optimize. Based on the amount of Controller data, the database optimization job may take several hours to several days to complete.

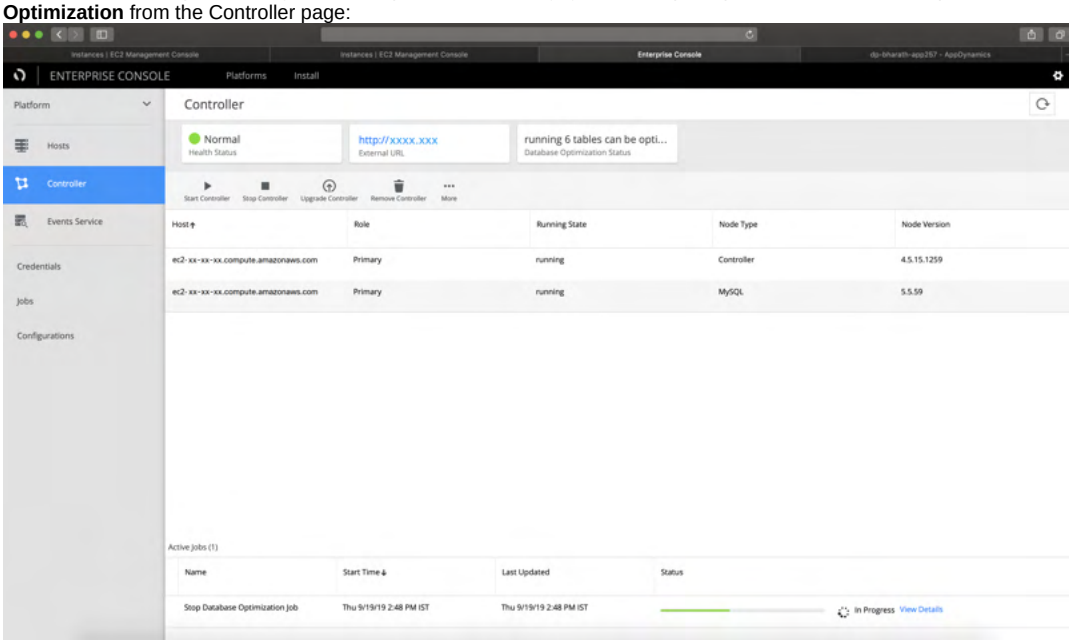
e. Once all of the tables have been optimized successfully, the database optimization process completes and no longer displays on the page. To verify that all tables have been optimized, enter and run the following query:

```
cd <controller_home>/bin directory
./controller.sh login-db
        mysql>SELECT table_name
        FROM   information_schema.key_column_usage
        WHERE  table_name LIKE 'metricdata%'
     AND table_name != 'metricdata_min'
     AND table_name != 'metricdata_min_agg'
     AND column_name = 'ts_min'
     AND ordinal_position = 1;
```

If the query returns any results, then those tables have not been optimized.
If the query returns zero records, then all of the tables were optimized successfully.

How to Stop Database Optimization

After the database optimization job has completed successfully, you can stop the process. From the Enterprise Console, select **Stop Database Optimization** from the Controller page:



> You may need to stop the database optimization process if it is using too many resources and you notice a performance impact on the Controller, or if you decide to reschedule the process to run at a later date.

Troubleshooting Database Optimization

The following table describes possible conditions that may cause errors to occur and actions to take to mitigate them:

Errors or Conditions	User Action
Job failed; `Database replication is broken` message displays.	Re-establish database replication incrementally, then finalize replication.
Ran out of disk space while the database optimization job was running, and job stops processing.	Free up disk space and restart database optimization job.

Uninstall the Controller

This page describes how to uninstall the Controller software and associated files from a platform using the Enterprise Console.

Before Starting

If you have installed the Events Service with the Enterprise Console, it is recommended that you uninstall the Events Service before you uninstall the Controller. See Uninstall the Events Service for more information.

In addition, if you have the EUM Server, Application Analytics, or other product modules installed, keep in mind that if you reinstall the Controller later, you will need to configure integration settings for the modules manually.

Optionally, stop the Controller before uninstalling as described in Start or Stop the Controller. If you do not stop the Controller, the uninstaller will do so for you. However, if your database or Controller generally take a long time to shut down, you can avoid the possibility of time-out errors during uninstallation by stopping the services manually.

Uninstall the Controller Using the Enterprise Console

You can uninstall the Controller on the Controller page in the GUI or by completing the following steps:

1. Open a console:
    - On Linux, open a terminal window and switch to the user who installed the Controller or to a user with equivalent directory permissions.
    - On Windows, open an elevated command prompt by right-clicking on the Command Prompt icon in the Windows Start Menu and choosing **Run as Administrator**.
2. From the command line, navigate to the Enterprise Console bin directory, `platform-admin/bin`.
3. Run the following command:

Note that you cannot use the other AppDynamics platform components without a Controller, so you must install a new Controller before you can resume using the platform.

EUM Server Deployment

The default End User Monitoring deployment assumes that EUM agents (Mobile and Browser) send their data to the EUM Cloud, a cloud-based processor. To deploy EUM completely on-premises, you need to install the EUM Server, the on-premises version of the EUM Cloud, as described here.

Installation Overview

The EUM Server receives data from EUM agents, processes and stores that data, and makes it available to the AppDynamics Controller. Certain EUM features—specifically, Browser Request Analytics and Mobile Request Analytics, features of Application Analytics that extend the functionality of Browser and Mobile Analyze—require access to the AppDynamics Events Service.

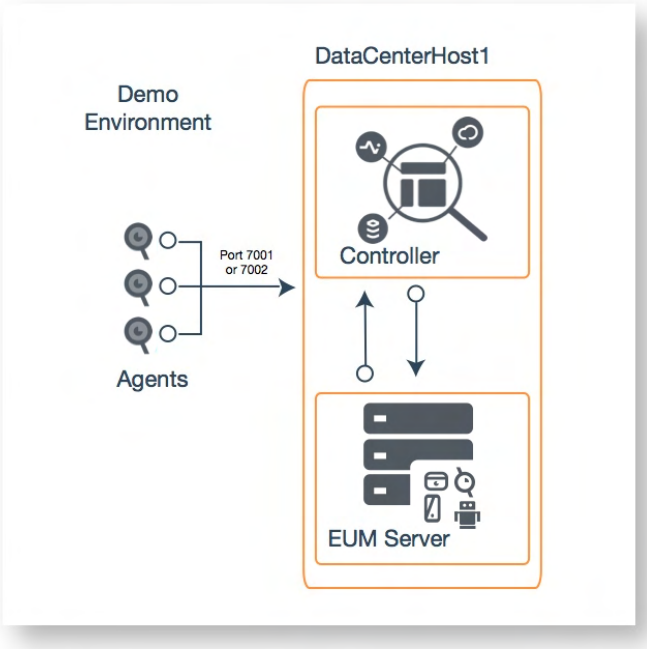
To set up a complete on-premises EUM Server deployment, therefore, you need to:

1. Determine which version of the EUM Server is compatible with your other platform components.
2. Install the on-premises Controller or prepare an in-service Controller to work with the EUM Server
3. Install the on-premises Events Service Deployment and configure it to work with your on-premises Controller
4. Install the on-premises EUM Server and configure it to work with your Events Service and Controller.

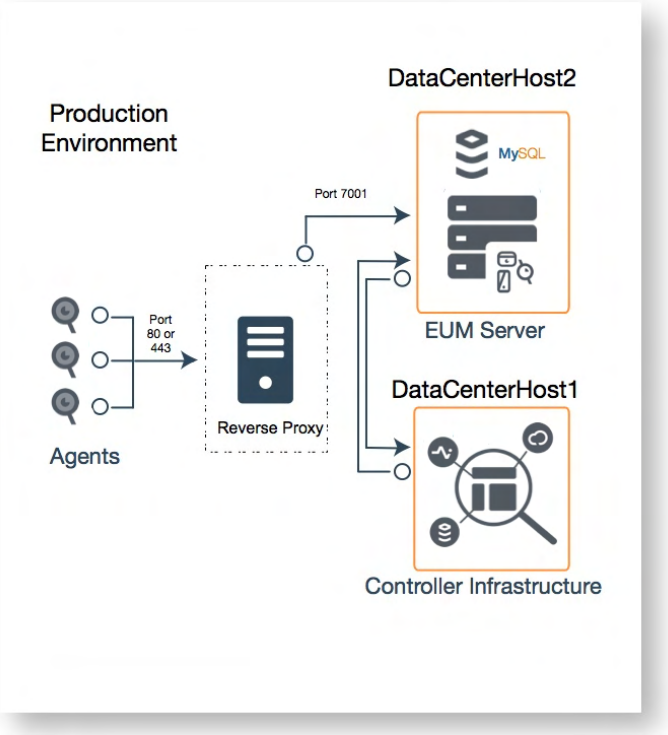
Deployment Modes for the EUM Server

For demonstration and light testing purposes, choose the Demo installation option, where the EUM Server and Controller are installed on the same host, and the EUM Server shares the Controller's MySQL instance. For production installation, choose the Product installation option, where the EUM Server and Controller sit on different hosts, and the EUM Server hosts its own MySQL instance.

In Demo mode, the EUM Server listens for connections on port 7001 or 7002. The secure port, 7002, uses a built-in, self-signed certificate, which is only used in demo mode.



In a production environment, the EUM Server is likely to operate behind a reverse proxy. A reverse proxy relieves the performance burden of SSL termination from the EUM Server. It also helps ease certificate management and security administration in general. Further, as the connection point for agent beacons, the Server needs to have the security layer of a proxy between itself and the external Internet.



Using a reverse proxy is the recommended method of setting up HTTPS connections for an on-premises EUM Server. If this is not possible in your installation, however, it is possible to set HTTPS support manually. See information on setting up a custom keystore in Secure the EUM Server.

Embedded Geo Server

The EUM Server includes an embedded Geo Server that provides the geo information. The Geo Server either obtains geo information from your custom Geo Server or by resolving incoming IPv4 address (IPv6 addresses are *not* supported) with Neustar data or custom geo data.

Add New Geo Data

To add new geo data, you can either add a new Neustar data file or create the custom geo data file geo-ip-mappings.xml and place it in the directory `eum-processor/bin/`. The EUM Server automatically detects and loads new geo data files.

Update the Geo Server

The Geo Server is updated when you update the EUM Server.

Host Your Own Geo Server

Follow the instructions given in Install and Host a Custom Geo Server for Browser RUM.

Check Controller Version

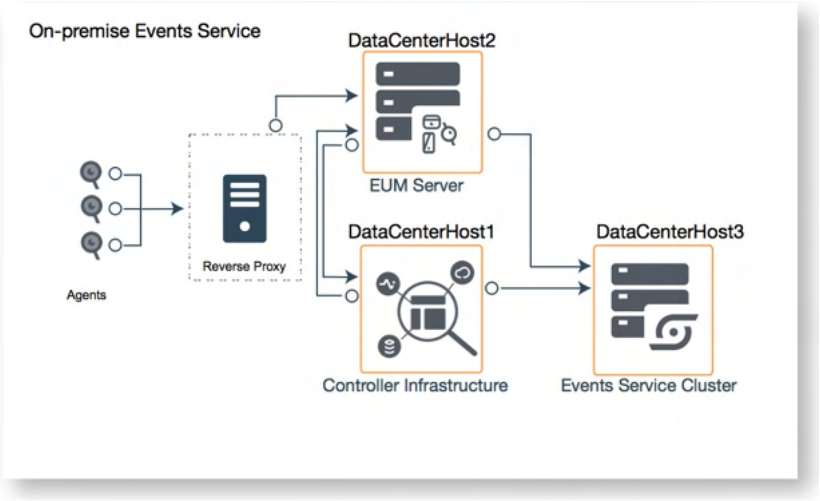
Before you run the EUM Server installer:

1. Check your Controller version. The 4.5 EUM Server works with the AppDynamics Controller version 4.5 or earlier. A Controller works with a EUM Server that is the same or a later version, which includes the major, minor, and patch version. Thus, a version 4.5.2 Controller works with a 4.5.2 or later version of the EUM Server, but that version 4.5.2 Controller does *not* work with a 4.5.1 or 4.5.0 version of the EUM Server. See Upgrade the Production EUM Server on information about upgrading the platform.
2. Back up the current version of your Controller.
3. Choose a time window that has minimum impact on service availability.

Install the On-Premises Events Service

The Analyze function in Browser RUM and the Crash Report and Analyze components in Mobile RUM rely on the AppDynamics Events Service, the Platform's unstructured document store. The Events Service that is configured by default for EUM is a cloud-based service.

If you are running on-premises and wish to keep all your processing on-premises, after installing and configuring the Controller, you must install an on-premises version of the Events Service as described in this section. Note that relying on the Events Service purely for use with the EUM UI does not require a separate Application Analytics license. Other uses *may* require a separate license.



There are multiple modes of deploying the Events Service. For detailed information on installing and configuring the Events Service, see Events Service Deployment.

Run the EUM Server Installer

Before starting, get the installer version appropriate for your target system. You can get the installer from the AppDynamics download site.

Run the EUM installer under the same user account on the target machine like the one used to install the Controller, or using an account that has read, write, and execute permissions to the Controller home directory. Installing with incompatible permission levels—for example, attempting to install the EUM Server as a regular user while the Controller was installed by root user—may result in installation or operation errors.

The EUM Server is automatically installed as a Windows service. All upgrades are automatically converted to a Window service.

The installer can be run in three modes:

- GUI
- Console
- Silent mode with `varfile`

See the following page for details on installing as appropriate for your deployment mode:

- For demo mode, see Install a Demo EUM Server.
- For production mode, see Install a Production EUM Server.

Update the Agents

You must update the address that agents use to send their beacons to the EUM Server based on your configuration. For Browser Real User Monitoring, the Controller updates the JavaScript agent. Simply re-download and deploy it, as described in Set Up and Configure Browser RUM. For Mobile RUM, the mobile applications themselves need to be updated, using the mobile SDKs. For more information, see Customize the Android Instrumentation and Customize the iOS Instrumentation.

Start and Stop the EUM Server and Database

The EUM Server is installed as a Windows service automatically. You can manage how you want this service to run using the Local Services dialog.

Start/Stop the EUM Server

On Linux, start the EUM server from the `eum-processor` directory in the EUM home as follows:

```
bin/eum.sh start
```

For a demonstration environment, run the command as sudo.

On Windows, if you ever need to start the EUM Server manually, you can do so by running:

```
bin\eum-processor.bat start
```

You can check if the server is running and accessible by going to `http://<hostname>:7001/eumaggregator/ping` with your browser. Your browser should display `ping`.

To stop the EUM Server, pass the stop command to the `eum` script. For example, on Linux, from the `eum-processor` directory, run:

```
bin/eum.sh stop
```

You can also start and stop the EUM database. On Windows, you can do so from the Windows Services.

Start/Stop the EUM MySQL Database

On Linux, you can start MySQL by navigating to the directory, `<EUM>/orcha/orcha-master/bin`, and running:

```
./orcha-master -d mysql.groovy -p ../../playbooks/mysql-orcha/start-mysql.orcha -o ../conf/orcha.properties -c
local
```

To stop MySQL on Linux, run:

```
./orcha-master -d mysql.groovy -p ../../playbooks/mysql-orcha/stop-mysql.orcha -o ../conf/orcha.properties -c
local
```

EUM Server Requirements

Related pages:

- On-premises EUM Server Sizing Guide
- Sizing XL profiles for on premises EUM Server and the Events Service: What's recommended?

This page lists the EUM Server requirements, offers sizing guidance, and shows you how to use configuration to modify the default settings. For additional EUM Processor sizing information, see Analytics' Recipe Book for on-prem configuration in the AppDynamics Community.

Hardware Requirements

The requirements and guidelines for the EUM Server machine (basic usage) are as follows:

- Minimum 50 GB extra disk space. See Disk Requirements Based on Resource Timing Snapshots to learn when more disk space is needed.
- 64-bit Windows or Linux operating system
- Processing: 4 cores
- 10 Mbps network bandwidth
- Minimum 8 GB memory total (4 GB is defined as max heap in JVM). See RAM Requirements Based on the Beacon Load to learn when more RAM is required.
- NTP enabled on both the EUM Server host and the Controller machine. The machine clocks need to be able to synchronize.

A machine with these specs can be expected to handle around 10K page requests a minute or 10K simultaneous mobile users. Adding on-premises Analytics capability requires increasing these requirements—particularly disk space—considerably, depending on the use case.

RAM Requirements Based on the Beacon Load

Beacons are sent to the EUM Server every 10 seconds, and each beacon can contain data for multiple events. You can configure the JavaScript Agent to limit the number of Ajax requests.

The table below specifies the required RAM based on your beacon load per minute and lists the content of a typical beacon.

Peak Beacons Per Minute	Typical Beacon Composition	RAM
~3K	• 600 sessions • 1K base pages • 2K virtual pages • 7K Ajax requests	8 GB
~16K	• 1.8K sessions • 5K base pages • 10K virtual pages • 40K Ajax requests	16 GB
~26K	• 3.6K sessions • 8K base pages • 17K virtual pages • 62 Ajax requests	16 GB
~33K	• 3.9K sessions • 10K base pages • 20K virtual pages • 74K Ajax requests	32 GB
>40K	12K base pages	32 GB

Disk Requirements Based on Resource Timing Snapshots

By default, the EUM Server accepts a maximum of 1K resource timing snapshots per minute and retains those snapshots for 15 days. On average, each snapshot takes 3 KB of disk space.

Because of the number of resource timing snapshots impact disk usage, you should follow the guidelines in the table below.

Number of Resource Timing Snapshots	Recommended Disk Space
~500	40 GB
~1000	64 GB
~1500	96 GB
~2000	128 GB

If needed, you can reduce the number of resource timing snapshots or reduce the disk space allotted for storing resource snapshots by doing one or more of the following:

- Configure the JavaScript Agent to modify and limit the number of resources to monitor.
- Use the EUM Server configuration `onprem.resourceSnapshotAllowance` to specify the maximum disk space allotted for storing resource snapshots. See EUM Server Configuration File for a complete list of configurations.
- Limit the number of snapshots retained by the EUM server by setting a global maximum, reducing the time that they are retained, or by filtering snapshots based on the network response time. See Limit the Number of EUM Snapshots for instructions.

Filesystem Requirements

The filesystem of the machine on which you install EUM should be tuned to handle a large number of small files. In practical terms, this means that either the filesystem should be allocated with a large number of inodes or the filesystem should support dynamic inode allocation.

Controller Version

The AppDynamics Platform you use with the EUM server must have a supported Controller version installed. Controllers only work with the same or later versions of the EUM Server. For example, the 4.5 EUM Server works with the AppDynamics Controller version 4.5 or earlier.

Open File Descriptor and User Process Limits

On Linux, also ensure that open file descriptor and user process limits on the EUM Server machine are set to a sufficient value. For the EUM Server, the hard and soft limits should be as follows:

- Open file descriptor limit (`nofile`): 65535
- Process limit (`nproc`): 8192

See "Configure User Limits in Linux" below for information on how to check and set user limits.

Configure User Limits in Linux

The following log warnings may indicate insufficient limits:

- Warning in database log: "Could not increase number of max_open_files to more than xxxx".
- Warning in server log: "Cannot allocate more connections".

To check your existing settings, as the root user, enter the following commands:

```
ulimit -S -n
ulimit -S -u
```

The output indicates the soft limits for the open file descriptor and soft limits for processes, respectively. If the values are lower than recommended, you need to modify them.

Where you configure the settings depends upon your Linux distribution:

- If your system has a `/etc/security/limits.d directory`, add the settings as the content of a new, appropriately named file under the directory.
- If it does not have a `/etc/security/limits.d` directory, add the settings to `/etc/security/limits.conf`.
- If your system does not have a `/etc/security/limits.conf` file, it is possible to put the `ulimit` command in `/etc/profile`. However, check the documentation for your Linux distribution for the recommendations specific for your system.

To configure the limits:

1. Determine whether you have a `/etc/security/limits.d` directory on your system, and take one of the following steps depending on the result:
    - If you *do not* have a `/etc/security/limits.d` directory:
        a. As the root user, open the `limits.conf` file for editing: `/etc/security/limits.conf`

b. Set the open file descriptor limit by adding the following lines, replacing `<login_user>` with the operating system username under which the EUM Server runs:

```
<login_user> hard nofile 65535
<login_user> soft nofile 65535
<login_user> hard nproc 8192
<login_user> soft nproc 8192
```

- If you *do* have a `/etc/security/limits.d` directory:
  a. As the root user, create a new file in the `limits.d` directory. Give the file a descriptive name, such as the following: `/etc/security/limits.d/appdynamics.conf`
  b. In the file, add the configuration setting for the limits, replacing `<login_user>` with the operating system username under which the EUM Server runs:

```
<login_user> hard nofile 65535
<login_user> soft nofile 65535
<login_user> hard nproc 8192
<login_user> soft nproc 8192
```

2. Enable the file descriptor and process limits as follows:
   a. Open the following file for editing: `/etc/pam.d/common-session`
   b. Add the line: `session required pam_limits.so`
3. Save your changes to the file.

When you log in again as the user identified by `login_user`, the limits will take effect.

Network Settings

The network settings on the operating system need to be tuned for high-performance data transfers. Incorrectly tuned network settings can manifest themselves as stability issues on the EUM Server.

The following command listing demonstrates tuning suggestions for Linux operating systems. As shown, AppDynamics recommends a TCP/FIN timeout setting of 10 seconds (the default is typically 60), the TCP connection keepalive time to 1800 seconds (reduced from 7200, typically), and disabling TCP window scale, TCP SACK, and TCP timestamps.

```
echo 5 > /proc/sys/net/ipv4/tcp_fin_timeout
echo 1800 >/proc/sys/net/ipv4/tcp_keepalive_time
echo 0 >/proc/sys/net/ipv4/tcp_window_scaling
echo 0 >/proc/sys/net/ipv4/tcp_sack
echo 0 >/proc/sys/net/ipv4/tcp_timestamps
```

The commands demonstrate how to configure the network settings in the `/proc` system. To ensure the settings persist across system reboots, be sure to configure the equivalent settings in the `etc/sysctl.conf`, or the network stack configuration file appropriate for your operating system.

libaio Requirement

The EUM processor requires the `libaio` library to be on the system. This library facilitates asynchronous I/O operations on the system. Note if you have a NUMA based architecture, then you are required to install the `numactl` package.

Install `libaio` on the host machine if it does not already have it installed. The following table provides instructions on how to install `libaio` for some common flavors of the Linux operating system.

Linux Flavor	Command
Red Hat and CentOS	Use `yum` to install the library, such as: • `yum install libaio` • `yum install numactl`
Fedora	Install the library RPM from the Fedora website: • `yum install libaio` • `yum install numactl`

Ubuntu	Use apt-get, such as: • `sudo apt-get install libaio1` • `sudo apt-get install numactl`
Debian	Use a package manager such as APT to install the library (as described for the Ubuntu instructions above).

Install a Production EUM Server

You can run the installer using one of three methods:

- Interactive console mode
- GUI Installer
- Silent Installer

The GUI and silent installation methods are described below. To start the installer using interactive console mode, start the installer with the `-c` switch. The console mode prompts you for the equivalent information that appears in the GUI installer screens.

Additionally, you can run the installer using a Response file (for unattended installations). See Installing with the Silent Installer.

Requirements

- Before starting, download the installer distribution and extract it on the target machine. You obtain the EUM installer from the AppDynamics Download Center.
- To secure connections from agents to the EUM Server, AppDynamics strongly recommends that SSL traffic is terminated at a reverse proxy that sits in front of the EUM Server in the network path, and forwards connections to the EUM Server. However if this is not possible in your installation, it is possible to connect with HTTPS directly to the EUM Server. For information on setting up a custom keystore for production, see Secure the EUM Server.

  > ⚠ If you install and configure the Events Service with HTTPS support, you must perform a workaround for your EUM Server installation to complete properly. After the Events Service certificate configuration, install the EUM Server without Analytics enabled. Then, install the certificate into the EUM Server keystore following the steps described on the Secure the EUM Server page. Configure Analytics in the Events Services Properties, and restart the EUM Server.

- Before you install the EUM Server, Linux systems must have the `libaio` library installed. See the EUM Server Requirements.

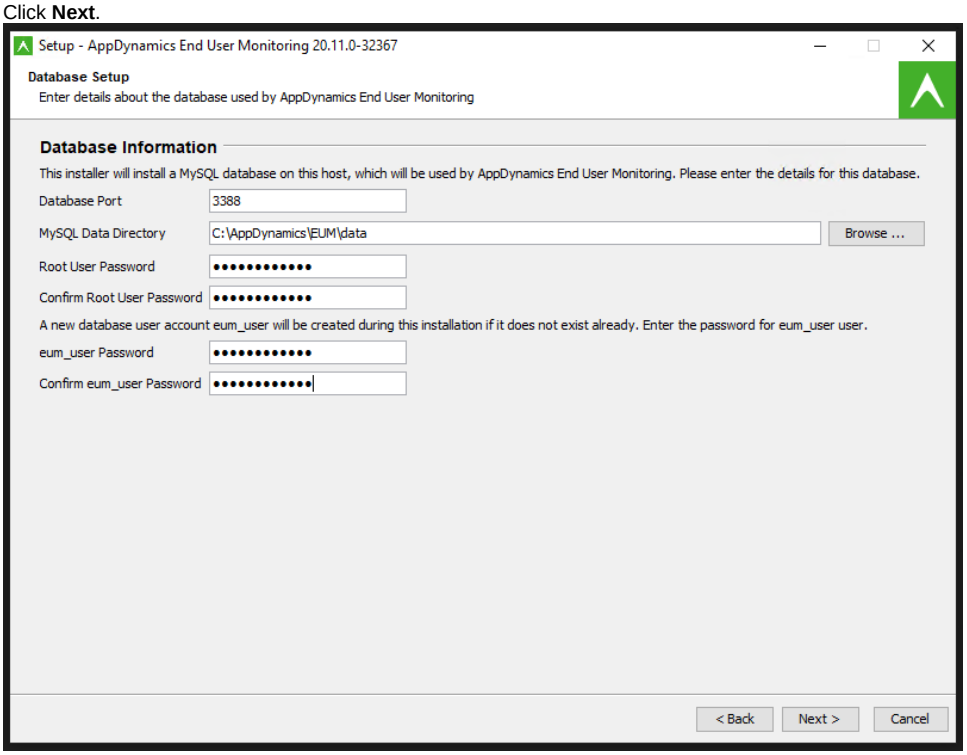
Install the EUM Server for a Production Deployment with the GUI Installer

Run the on-premises EUM installer on the machine on which you want to install the EUM Server.

1. Start the installer:

2. In the Welcome screen, click **Next** to continue.
3. Scroll to the end of the license agreement and accept the license agreement, then click **Next** to continue.
4. Select the destination directory, and click **Next** to continue.
5. Choose **Product** for the installation mode. This mode installs the EUM Server on this machine. Use this type if AppDynamics End User Monitoring and the AppDynamics Controller are installed on different hosts. Selecting this type will install a separate MySQL instance on this machine. Click **Next**.
6. In the Database Setup screen:
   a. Enter a new **Root User Password** and confirm it.
   b. Enter a new **eum_user Password** and confirm it.

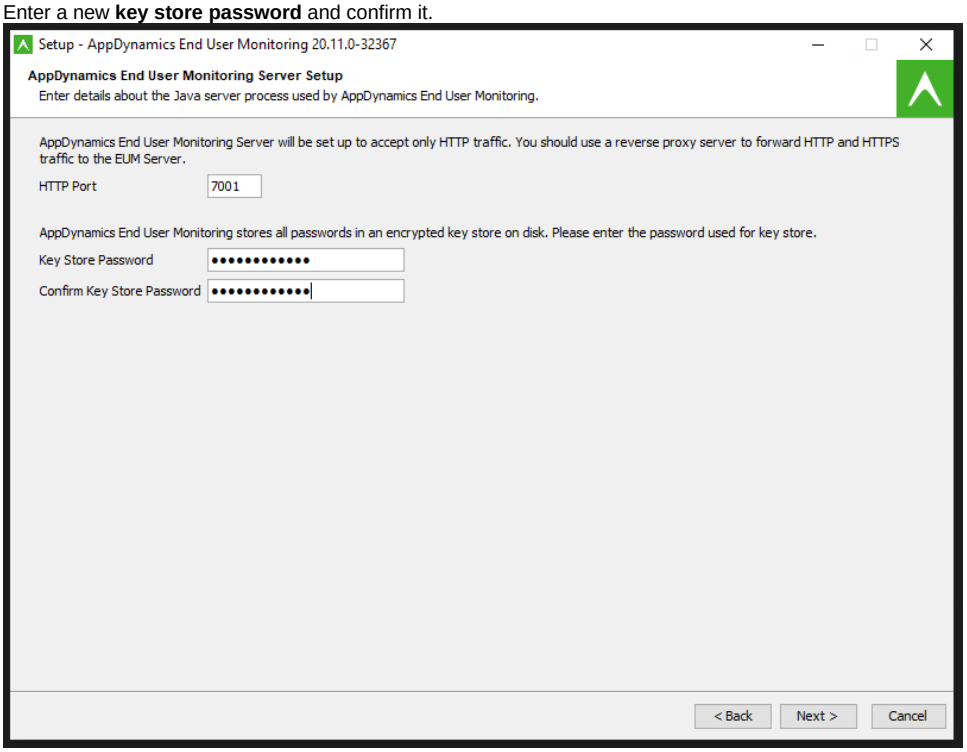
c. Click **Next**.



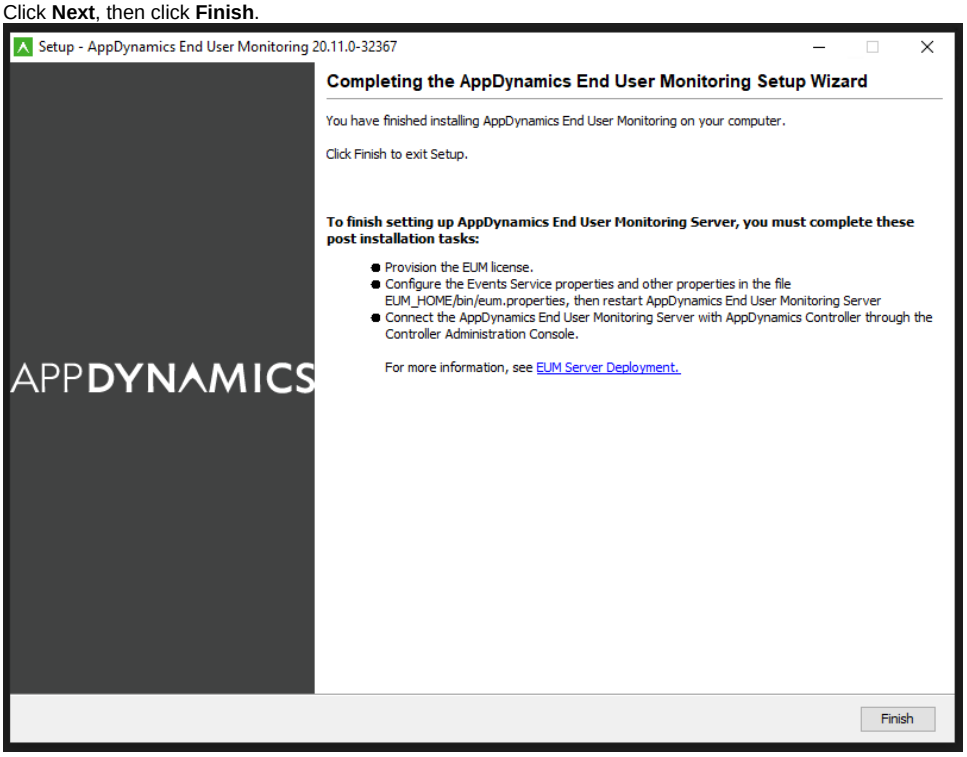
> Usernames and passwords can only consist of ASCII characters. In addition, passwords cannot include the characters '^', '/', or '$'.

7. In the AppDynamics End User Monitoring Setup screen:

a. Enter a new **key store password** and confirm it.



a. Click **Next**, then click **Finish**.



This completes the initial configuration and setup of the EUM Server. When finished, the EUM Server is running.

Post-Installation Tasks

To complete the AppDynamics EUM Server installation, you must perform these additional post-installation tasks (as shown in the last AppDynamics End User Monitoring Setup Wizard screen):

- Configure JVM options
- Provision the EUM license
- Configure the Events Services properties in the `eum.properties` file
- Connect the EUM Server with the AppDynamics Controller
- Secure the EUM Server by setting up a custom keystore

> The EUM Server Installer only configures the HTTP port. If you are installing the 4.5 EUM Server or upgrading to the 4.5 EUM Server, you must configure the EUM Server and update the AppDynamics Controller settings to connect to the HTTPS port. See Configure the HTTPS Port for the 4.5 EUM Server for instructions.

Configure JVM Options

Provision the EUM License

Follow the provision instructions based on your deployment type:

- Provision the EUM License for a Single-Tenant Controller
- Provision the EUM License for Multi-Tenant Controllers

Configure the Events Services Properties

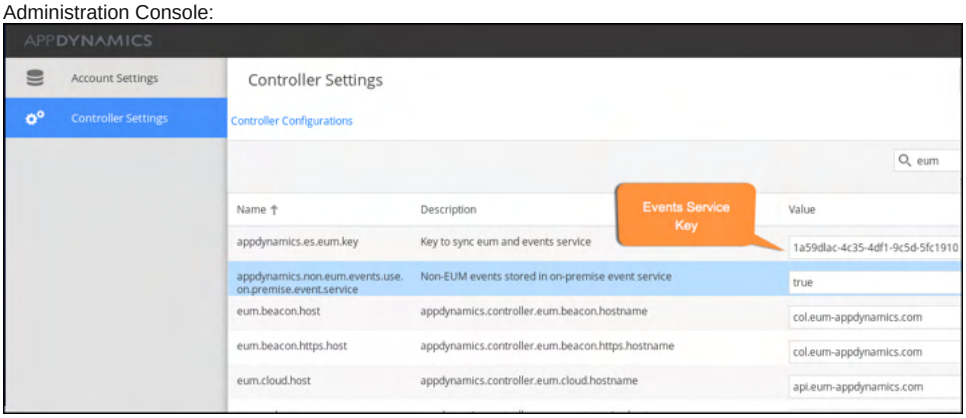
Configure the Events Services properties in the `eum.properties` file:

1. Ensure that Events Services is running.
2. Navigate to the `\EUM\eum-processor\bin` directory.
3. Open the `eum.properties` file to edit.
4. In the `eum.properties` file, enter these values:

Sample

```
analytics.enabled=true
analytics.serverScheme=http
analytics.serverHost=hostname-events-service (needs to be the hostname of your Events Service)
analytics.port=9080
analytics.accountAccessKey=1a59d1ac-4c35-4df1-9c5d-5fc191003441
```

The <analytics.accountAccessKey> is the Events Service key that appears as the appdynamics.es.eum.key value in the Administration Console:



The configuration should display similar to this example:

```
# Credential Key Store Information
onprem.useEncryptedCredentials=true
onprem.credentialKey=s_-001-12-F+ddxp+nzHI=dwDynZje09g=


# Web server properties
processorServer.httpPort=7001
processorServer.httpsPort=7002
processorServer.httpsProduction=true
processorServer.keyStorePassword=1wnl1u

# Analytics server properties
analytics.enabled=true
analytics.serverScheme=http
analytics.serverHost=events.service.hostname
analytics.port=9080
analytics.accountAccessKey=1a59d1ac-4c35-4df1-9c5d-5fc191003441

# Session properties
collection.sessionEnabled=true
crashProcessing.sessionEnabled=true
```

5. After updating the eum.properties file, restart the EUM Server.

Connect the EUM Server with the AppDynamics Controller

Connect the EUM Server with the AppDynamics Controller:

1. Log in to the Administration Console.
2. Set these Controller properties:
   - eum.cloud.host: http://eum-host-name:7001 – Location where the Controller will poll for EUM metrics.
   - eum.beacon.host: http://eum-host-name:7001 – Location where the JavaScript Agent will be configured to send out beacons over the HTTP protocol.
   - eum.beacon.https.host: https://eum-host-name:7002 – Location where JavaScript Agent will be configured to send out beacons over the HTTPS protocol.
   - eum.mobile.screenshot.host: http://host-name:7001 – Location where the Controller will search for mobile screenshots.

Installing with the Silent Installer

Instead of using the GUI installer, you can use the silent installer to perform an unattended installation. The silent installer uses a response file as a source for the initial configuration settings. It's useful for scripting installation or performing large scale deployments.

To use a response file for installation:

1. Create a file named `response.varfile` on the machine on which you will run EUM installer and include the following:

```
sys.adminRights$Boolean=false
sys.languageId=en
sys.installationDir=/AppDynamics/EUM
euem.InstallationMode=split
euem.Host=eumhostname
euem.initialHeapXms=1024
euem.maximumHeapXmx=4096
euem.httpPort=7001
euem.httpsPort=7002
mysql.databasePort=3388
mysql.databaseRootUser=root
mysql.dbHostName=localhost
mysql.dataDir=/usr/local/AppDynamics/EUM/data
mysql.rootUserPassword=singcontroller
mysql.rootUserPasswordReEnter=singcontroller
eumDatabasePassword=secret
eumDatabaseReEnterPassword=secret
keyStorePassword=secret
keyStorePasswordReEnter=secret
eventsService.isEnabled$Boolean=true
eventsService.serverScheme=http
eventsService.host=eventsservice_host
eventsService.port=9080
eventsService.APIKey=1a234567-1234-1234-4567-ab123456
```

2. Modify values of the installation parameters based on your own environment and requirements. Particularly ensure that the directory paths and passwords match your environment.
3. Run the installer with the following command:

Install a Demo EUM Server

You can run the installer in one of three modes. The GUI and silent installation methods are described below. To start the installer in interactive console mode, start the installer with the `-c` switch. The console mode prompts you for the equivalent information that appears in the GUI installer screens, as described below.

In addition, you can run the installer using a Response file (for unattended installations). See Installing with the Silent Installer.

About the Demo Installation

This mode is for demonstration and light testing only. If you are using the Events Service, it must be on a separate host.

If you do not already have an existing on-premises Controller, install it as described in Custom Install.

Installation Requirements

To install the Demo Installation, you are required to do the following:

- Install and run a Controller instance on the same host machine before starting the EUM Server installation.
- Install the EUM Server with the same user account used to install the Controller, or use an account that has read, write, and execute permissions to the Controller home directory.

Installing with the GUI Installer

1. Start the installer:
    - On Linux:
        a. From a command prompt, navigate to the directory to which you downloaded the EUM Server installer.
        b. Change permissions on the downloaded installer script to make it executable, as follows:

        ```
        chmod 775 euem-64bit-linux-4.5.x.x.sh
        ```

        c. Run the script as follows:

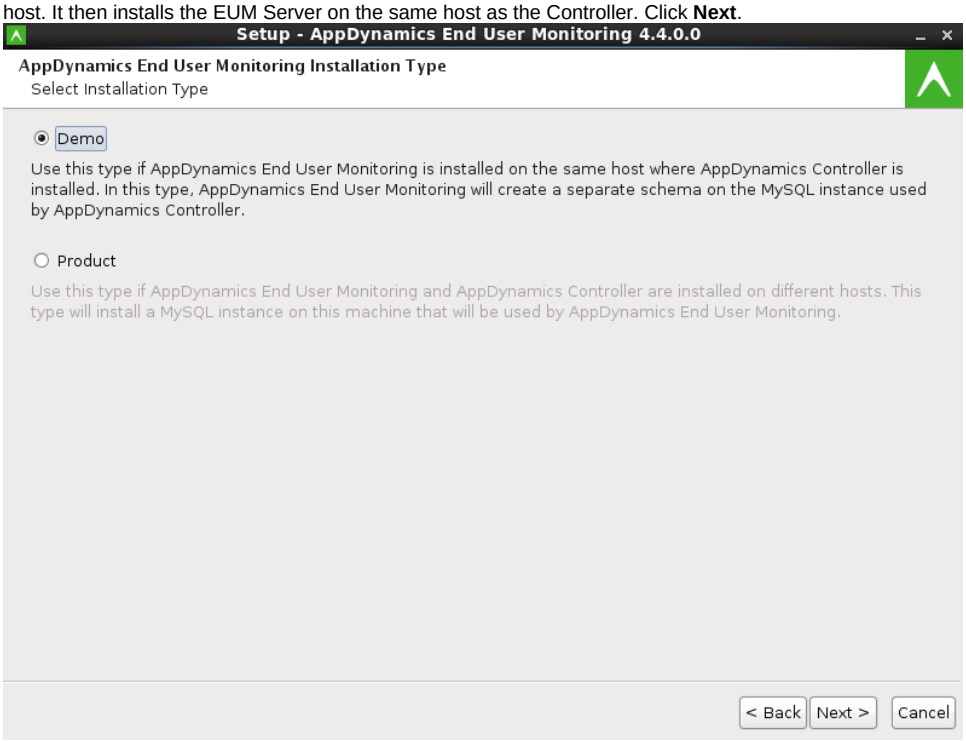
        ```
        ./euem-64bit-linux-4.5.x.x.sh
        ```

    - On Windows:
        a. Open an elevated command prompt (run as administrator) and navigate to the directory to which you downloaded the EUM Server installer.
        b. Run the installer:

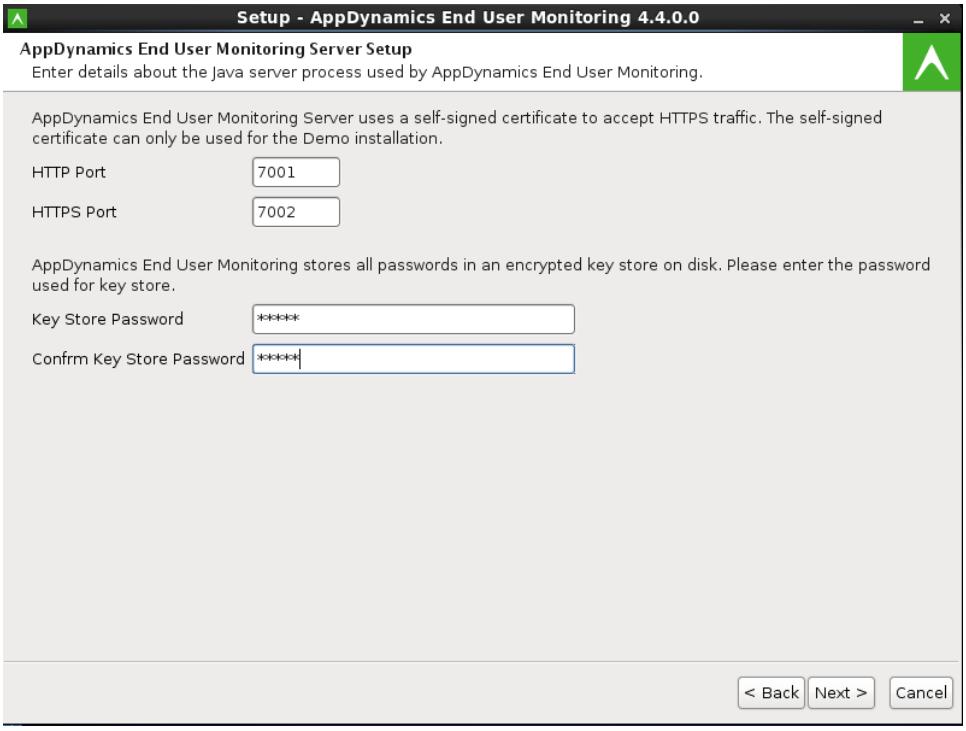
        ```
        euem-64bit-windows-4.5.x.x.exe
        ```

2. In the Welcome screen, click **Next**.
   The **License Agreement** page appears.
3. Scroll to the end of the license agreement, accept the license agreement and click **Next** to continue.
4. Select the directory in which you want to install the server and click **Next**.

5. Choose **Demo** for the installation mode. In this mode, the installer looks for a Controller on the current host and an Events Service on a separate host. It then installs the EUM Server on the same host as the Controller. Click **Next**.



6. In the **Database Setup** dialog box:
   a. Enter a password in the **Root User Password** field.
   b. Enter and confirm a password for the **eum_user** database user account.
7. In the End User Monitoring Server Setup screen:
   a. Enter the HTTP or HTTPs listening ports on the EUM Server at which the Controller will connect to the EUM Server (the default HTTP port is 7001 and HTTPS is 7002).
   b. Enter a new password in the **Key Store Password** and confirm it.



Usernames and passwords can only consist of ASCII characters. In addition, passwords cannot include the characters '^', '/', or '$'.

Note that ports shown here are the location both to which the EUM Agents send their beacons and from which the Controller fetches the processed beacon data. Click **Next** and click **Finish.**

Post-installation Tasks

After installing the EUM server, you must perform three additional post-installation tasks:

- Provision the EUM license
- Configure the Events Services properties in the `eum.properties` file (optional)
- Connect the EUM server with the AppDynamics Controller

Provision the EUM License

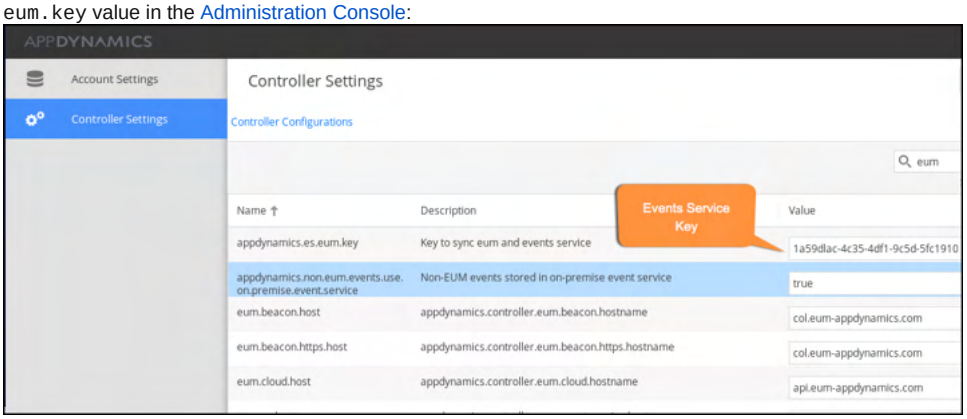
To provision the EUM license:

Configure the Events Service Properties

For the demo installation, Events Services configuration is not required for the EUM Server to start.

To configure the Events Service properties in the `eum.properties` file:

1. Navigate to and start your Controller.
2. Using the CLI, navigate to the `bin` directory.
3. Open the `eum.properties` file for editing.
4. In the `eum.properties` file, enter the following values:
    a. `analytics.enabled=true`
    b. `analytics.serverHost=<hostname>`, where `<hostname>` is the host where the event service is running
    c. `analytics.accountAccessKey=<eum_key>`, where `<eum_key>` is the Events Service key that appears as the `appdynamics.es.eum.key` value in the Administration Console:



The configuration should appear similar to the following example:

```
# Credential Key Store Information
onprem.useEncryptedCredentials=true
onprem.credentialKey=s_-001-12-F+ddxp+nzHI=dwDynZje09g=


# Web server properties
processorServer.httpPort=7001
processorServer.httpsPort=7002
processorServer.httpsProduction=true
processorServer.keyStorePassword=1wnl1u

# Analytics server properties
analytics.enabled=true
analytics.serverScheme=http
analytics.serverHost=events.service.hostname
analytics.port=9080
analytics.accountAccessKey=1a59d1ac-4c35-4df1-9c5d-5fc191003441

# Session properties
collection.sessionEnabled=true
crashProcessing.sessionEnabled=true
```

5. After updating the `eum.properties` file, restart the EUM Server.

Connect the EUM Server with the AppDynamics Controller

To connect the EUM server with the AppDynamics Controller:

1. Log in to the Administration Console.
2. Set the following Controller properties:
    a. `eum.cloud.host: http://localhost:7001` – This tells you where the Controller will poll for EUM metrics
    b. `eum.beacon.host: eum-host-name:7001` – This tells you where beacons are sent for page requests made with the HTTP protocol
    c. `eum.beacon.https.host: https://eum-host-name:7002` – This tells you where the beacons are sent for page requests made with the HTTPS protocol
    d. `eum.mobile.screenshot.host: eum-host-name:7001` – This tells you where the Controller will look for mobile screenshots

Installing with the Silent Installer

Instead of using the installer in GUI mode, you can use the silent installer to perform an unattended installation. The silent installer takes a response file as a source for the initial configuration settings. It's useful for scripting installation or performing large scale deployments.

To use a response file for a Demo installation:

1. Create a file named `response.varfile` on the machine on which you will run EUM installer with the following:

```
sys.adminRights$Boolean=false
sys.languageId=en
sys.installationDir=/AppDynamics/EUM
euem.InstallationMode=demo
euem.Host=controller
euem.initialHeapXms=1024
euem.maximumHeapXmx=4096
euem.httpPort=7001
euem.httpsPort=7002
mysql.databasePort=3388
mysql.databaseRootUser=root
mysql.dbHostName=localhost
mysql.dataDir=/usr/local/AppDynamics/EUM/data
mysql.rootUserPassword=singcontroller
mysql.rootUserPasswordReEnter=singcontroller
eumDatabasePassword=secret
eumDatabaseReEnterPassword=secret
keyStorePassword=secret
keyStorePasswordReEnter=secret
eventsService.isEnabled$Boolean=true
eventsService.serverScheme=http
eventsService.host=eventsservice_host
```

```
eventsService.port=9080
eventsService.APIKey=1a234567-1234-1234-4567-ab123456
```

2. Modify values of the installation parameters based on your own environment and requirements. Particularly ensure that the directory paths and passwords match your environment.
3. Run the installer with the following command:

```
./euem-64bit-linux-4.5.x.x.sh -q -varfile response.varfile
```

On Windows, use:

```
euem-64bit-windows-4.5.x.x.exe -q -varfile response.varfile
```

Provision EUM Licenses

Related pages:

- Install a Production EUM Server
- Troubleshoot EUM Server Installation
- Controller Deployment
- Multi-Tenant Controller Accounts

This page describes how to provision EUM licenses for single-tenant and multi-tenant Controllers.

Provision the EUM License for a Single-Tenant Controller

To provision the EUM license for a single-tenant Controller:

Provision the EUM License for Multi-Tenant Controllers

For each on-prem multi-tenant Controller account wanting EUM access, you are required to provision an EUM license. Provisioning EUM license units for different Controller accounts enable you to better track, manage, and limit license usage.

To enable the EUM Server to work with on-prem multi-tenant Controllers:

1. Complete the setup requirements.
2. Request EUM licenses for the Controller accounts requiring EUM access.
3. Provision EUM licenses for the Controller accounts.

Setup Requirements

- Deploy an on-premises AppDynamics Controller
- Set up multi-tenancy on the Controller

Request EUM Licenses

For each additional Controller account requiring EUM access, you will need to request EUM licenses from the AppDynamics Sales team. The AppDynamics Sales team can help you determine how many licenses and units per license will meet your needs.

Provision Licenses for Controller Accounts

To provision EUM licenses for each Controller account:

1. Log in to your EUM Server.
2. Change to the directory with the script for provisioning licenses:

3. Provision each license, one at a time, on the EUM Server by running the following command:

4. Log in as the administrator to your **Controller Admin Console** through `http://<hostname>:<port>/controller/admin.jsp`.
5. From **Account Settings**, select the Controller account that have EUM licenses and click **Edit**.
6. From the Controller account page:
    a. Enter the EUM license key and the EUM account name in the **EUM License Key** and the **EUM Account Name** fields.
    b. Click **Save**.
7. Repeat steps 5 and 6 for the other Controller accounts that have EUM licenses.
8. Verify that the EUM licenses are working and available in the Controller UI. There will be an error message if the new EUM Server is not connected properly.

Secure the EUM Server

If you use HTTPS connections in a production (split host) EUM Server installation, use a custom RSA security certificate for the EUM server. This page describes how to create an RSA security certificate, change the password for the credential keystore, and how to obfuscate a password for the security certificate keystore.

Set Up a Custom Keystore for Production

In demo mode, the EUM Server uses a default self-signed certificate named `ssugg.keystore`. This certificate is intended for demonstration and light testing only. Do not use self-signed certificates for production systems since they are less secure than Certificate Authority (CA) signed certificates. EUM requires that certificates use RSA as the key algorithm whether they are self-signed or CA-signed.

For Mobile Real User Monitoring, if you use the default or another self-signed certificate on your EUM Server for testing, you may receive the following error: "The certificate for this server is invalid". Ensure that your self-signed certificate is trusted by the simulator or device you use for testing. In real-world scenarios, a CA signed certificate should be used since a self-signed certificate needs to be explicitly trusted by every device that reports to your EUM processor.

To secure the EUM server with a custom certificate and keystore, generate a new JKS keystore and configure the EUM Server to use it.

The following instructions describe how to create a JKS keystore for the EUM Server with a new key-pair or an existing key-pair. Alternatively, you can also configure the EUM server to use an existing JKS keystore.

The instructions demonstrate the steps with the Linux command line, but the commands are similar to the commands used for Windows. Make sure to adjust the paths for your operating system.

Overview of the Steps

The procedure is made up of three parts:

1. Create a new certificate and keystore (1a) or import an existing certificate into a keystore (1b).
2. Configure the EUM Server to use the keystore.
3. Restart and test the new keystore.

Step 1a: Create a New Certificate and Keystore

1. At a command prompt, navigate to the `eum-processor` directory:

   ```
   cd <appdynamics_home>/EUM/eum-processor
   ```

2. Create a new keystore with a new unique key pair that uses RSA encryption:

   ```
   ../jre/bin/keytool -genkey -keyalg RSA -validity <validity_in_days> -alias 'eum-processor' -keystore bin
   /mycustom.keystore
   ```

   This creates a new public-private key pair with an alias of `'eum-processor'`. You can use any value you like for the alias.

   The "first and last name" required during the installation process becomes the common name (CN) of the certificate. Use the name of the server.

3. Configure the keystore.
4. Specify a password for the keystore. You need to configure this password in the EUM configuration file later.
5. Generate a certificate signing request (CSR):

   ```
   ../jre/bin/keytool -certreq -keystore bin/mycustom.keystore -file /tmp/eum.csr -alias 'eum-processor'
   ```

   This generates a certificate signing request based on the contents of the alias, in the example `'eum-processor'`. You should send the output file (`/tmp/eum.csr`, in the example) to a Certificate Authority for signing. After you receive the signed certificate, proceed as follows.
6. Install the certificate for the Certificate Authority used to sign the `.csr` file:

   ```
   ../jre/bin/keytool -import -trustcacerts -alias myorg-rootca -keystore bin/mycustom.keystore -file /path
   /to/CA-cert.txt
   ```

   This command imports your CA's root certificate into the keystore and stores it in an alias called `myorg-rootca`.
7. Install the signed server certificate as follows:

```
../jre/bin/keytool -import -keystore bin/mycustom.keystore -file /path/to/signed-cert.txt  -alias 'eum-processor'
```

This command imports your signed certificate over the top of the self-signed certificate in the existing alias, in the example, `'eum-processor'`.

8. Import the root certificate from step 6 to the Controller truststore:

```
keytool -import -trustcacerts -alias <alias_name> -file mycert.cer -keystore <complete_path_to_cacerts.jks>
```

Step 1b: Import an Existing Certificate into a JKS Keystore

If you have an existing public-private key pair that uses RSA, you must import them into a JKS keystore to use it for EUM.

1. At a command prompt, navigate to the `eum-processor` directory:

```
cd <appdynamics_home>/EUM/eum-processor
```

2. Stop the EUM process.

   Run the following command:

```
bin/eum.sh stop
```

3. If there is an existing custom JKS keystore, back it up:

```
mv <keystore>.jks <keystore>.jks.old
```

4. Import the private and public key for your certificate into a PKCS12 keystore:

```
openssl pkcs12 -inkey <private_key_file> -in <certificate_file> -export -out keystore.p12
```

5. Convert the PKCS12 keystore to JKS format:

```
keytool -importkeystore -srckeystore keystore.p12 -srcstoretype pkcs12 -destkeystore <JKS_keystore> -deststoretype JKS
```

   This command creates a JKS keystore with the name specified in the `-destkeystore` property.

6. Specify a password for the keystore. Use this password when you configure EUM to use the new keystore.

Step 2: Configure the EUM Server to Use the New Keystore

1. Place the new keystore file in the following directory: `<appdynamics_home>/EUM/eum-processor/bin`.
2. Edit the `eum.properties` file in the bin directory.
3. If the property `processorServer.keyStorePassword` is set, remove or uncomment it.
4. Add the keystore filename as the following property:

```
processorServer.keyStoreFileName=mycustom.keystore
```

5. Configure the password for the keystore. You can add the password to the file either in plain text or in the obfuscated form:
   - For a plain text password, add the password as the value for this property:

```
processorServer.keyStorePassword=mypassword
```

   - For an obfuscated password:
     a. Get the obfuscated password by running the following command in the `eum-processor` directory in a new command terminal:

```
bin/eum-credential-key.<bat|sh> obfuscate -plaintext <newpassword>
```

b. Copy the output of the command to your clipboard.
c. In `eum.properties`, paste the obfuscated password as the value of the `keyStorePassword` property:

```
processorServer.keyStorePassword=<obfuscated_key>
```

d. Add the `useObfuscatedKeyStorePassword` with the value set to true, as shown:

```
processorServer.useObfuscatedKeyStorePassword=true
```

6. Save and close the file.

Step 3: Restart and Test

1. Restart the EUM Server. From the `eum-processor` directory, run the following commands:

```
bin/eum.sh stop
bin/eum.sh start
```

2. Verify the new security certificate works by opening the following page in a browser:

```
https://<hostname>:7002/eumcollector/get-version
```

If you get a successful response, the configuration succeeded.

Change the Certificate Keystore Password

The previous steps describe how to create a new keystore which is likely to have a new password. To change the keystore password without creating a new keystore, perform the following steps:

1. At a command prompt, navigate to the `eum-processor` directory:

```
cd <appdynamics_home>/EUM/eum-processor
```

2. Run the `keytool` command for creating a new password:

```
../jre/bin/keytool -storepasswd -keystore bin/ssugg.keystore
```

The sample command creates the password for the default demo keystore, `ssugg.keystore`. In your command, use the name of your own keystore as the value for `-keystore`.
3. Enter the existing password and new password when prompted.
4. Get the obfuscated key by running the following command in the `eum-proccessor` directory:

```
bin/eum-credential-key.<bat|sh> obfuscate -plaintext <newpassword>
```

5. Copy the output of the previous command to your clipboard.
6. In the `eum.properties` file in the `eum-processor/bin` directory, paste the obfuscated password as the value for the `keyStorePassword` property:

```
processorServer.keyStorePassword=<obfuscated_key>
```

7. If you did not previously use an obfuscated password, add the following property:

```
processorServer.useObfuscatedKeyStorePassword=true
```

8. Save and close the file.
9. Restart the EUM Server.

Change the Credential Keystore Password for the EUM Database

When you install the EUM Server, you need to specify a password to use to secure the credential keystore for the EUM Server. After installation, you can change the password for the credential keystore. You may need to do this, for example, to comply with your organization's password rotation policy.

Note that completing these procedures requires a restart of the EUM Server.

To change the existing EUM server credential keystore password:

1. At a command prompt, navigate to the `eum-processor` directory:

   ```
   cd <appdynamics_home>/EUM/eum-processor
   ```

2. Generate a credential store with the new key using the following command:
   - On Linux:

     ```
     bin/eum-credential-key.sh generate_ks -storepass <new_password>
     ```

   - On Windows:

     ```
     bin\eum-credential-key.bat generate_ks -storepass <new_password>
     ```

   This creates and initializes a new credential file, `bin/credential.scs`.
3. Reencrypt the database password using the new credential store.
   - On Linux:

     ```
     bin/eum-credential-key.sh encrypt -storepass <new_password> -plaintext <DB_password>
     ```

   - On Windows:

     ```
     bin\eum-credential-key.bat encrypt -storepass <new_password> -plaintext <DB_password>
     ```

   The command prints out the encrypted form of the `DB_password` value you entered.
4. Copy the output from the previous command to your clipboard.
5. Open `bin/eum.properties` for editing, and replace the value of the `onprem.dbPassword` setting with the new encrypted password you copied to your clipboard.
6. Obfuscate the new credential key as follows:
   - On Linux:

     ```
     bin/eum-credential-key.sh obfuscate -plaintext <new_password>
     ```

   - On Window:

     ```
     bin\eum-credential-key.bat obfuscate -plaintext <new_password>
     ```

7. Copy the output of the previous command to your clipboard and in `eum.properties` replace the value of `onprem.credentialKey` with the value from your clipboard.
8. Save and close the properties file.
9. Restart the EUM server.

Change the EUM Database Password

At EUM Server installation time, you set a password for the EUM database. You can change it later as follows:

1. At a command prompt, navigate to the `eum-processor` directory:

   ```
   cd <appdynamics_home>/EUM/eum-processor
   ```

2. Encrypt the new database password using the credential key which you entered during installation:
   - On Linux:

```
bin/eum-credential-key.sh encrypt -storepass <plain_credential_key> -plaintext <New_DB_password>
```

- On Windows:

```
bin\eum-credential-key.bat encrypt -storepass <plain_credential_key> -plaintext <New_DB_password>
```

The command prints out the encrypted form of the `DB_password` value you entered.

3. Copy the output from the previous command to your clipboard.
4. Edit `bin/eum.properties` and replace the value of the `onprem.dbPassword` setting with the new encrypted password you copied to your clipboard.
5. Save and close the properties file.
6. Restart the EUM server.

Configure the EUM Server

This page describes administration and advanced configuration options for the EUM Server.

Configure Data Store Expiration

As part of the Analytics functionality used by EUM, the Server stores some data, like crash reports and resource snapshots, in a local blob store. The default setting of 30 days, but you can change the storage period to be longer or shorter by following these steps:

1. Open `$APPDYNAMICS_HOME/EUM/eum-processor/bin/eum.properties` with a text editor.
2. Open `$APPDYNAMICS_HOME/EUM/eum-processor/bin/eum.sample.properties` with a text editor.
3. Copy the `onprem.crashReportExpirationDays` property and the `onprem.resourceSnapshotExpirationDays` property from the sample file into `eum.properties` and set it to whatever value you wish. The unit is *days*.
4. Restart the Server.

Set the Maximum Length of Page URLs Read From Beacons

By default, after the EUM Collector receives beacons, the EUM Processor will only read 512 characters of page URLs contained in the beacon. If the page URL exceeds 512 characters, the EUM Processor will truncate the page URL. You can configure the EUM Processor to read a longer page URL with the configuration `beaconReader.maxUrlLength`. The maximum length that can be set is 2048, which is imposed by the JavaScript Agent creating the beacon.

To change the maximum length of the page URL read by the EUM Processor:

1. Open `$APPDYNAMICS_HOME/EUM/eum-processor/bin/eum.properties` with a text editor.
2. Add the property `beaconReader.maxUrlLength` to the desired length (maximum is 2048):

   ```
   beaconReader.maxUrlLength=<max_length>
   ```

3. Restart the Server.

Update the EUM Server's Geo Server

The on-prem EUM Server ships with Neustar's IP GeoPoint database for managing the geolocation of IP addresses. You can get daily updates of the Neustar IP GeoPoint from the AppDynamics download site.

To keep your version of the database current, you need to update your copy of the database manually:

Configure the Port for the EUM Agent

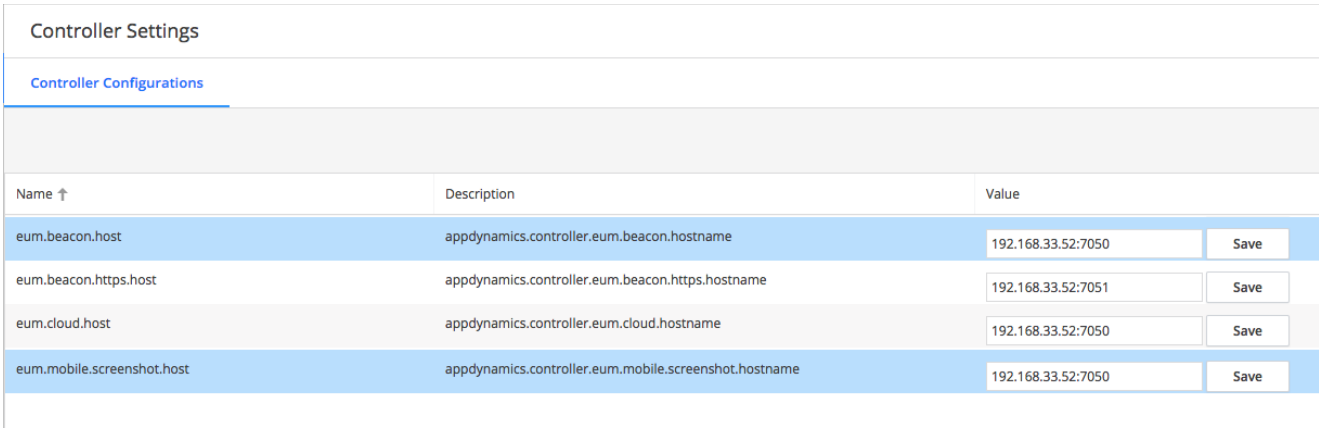
The on-prem EUM Server by default uses the same port to collect data from the EUM agent and to send data through the API server to the Controller. You can configure the EUM Server to use a different port to collect data from the EUM agent by following the instructions below.

1. Open `$APPDYNAMICS_HOME/EUM/eum-processor/bin/eum.properties` with a text editor.
2. Add the following lines to `$APPDYNAMICS_HOME/EUM/eum-processor/bin/eum.properties`, replacing `<PORT>` with the port you want the EUM server to listen to.

   ```
   processorServer.collectorHttpPort=<PORT>
   processorServer.collectorHttpsPort=<PORT>
   ```

3. Restart the EUM Server.
4. From the **Controller Admin UI**, change the ports for the properties `eum.beacon.host`, `eum.beacon.https.host`, `eum.cloud.host`, and `eum.mobile.screenshot.host` so that they are the same as those assigned to `processorServer.collectorHttpPort` and `processorServer.collectorHttpsPort`. This allows the beacon to communicate with the collector.

   For example, if you set `processorServer.collectorHttpPort=7050` and `processServer.collectorHttpsPort=7051`, you would then set the ports for the properties `eum.beacon.host` and `eum.mobile.screenshot.host` to `7050` for HTTP and `eum.beacon.https.host` to `7051` for HTTPS as shown below:



Limit the Number of EUM Snapshots

When an application has a high number of Ajax requests per page, the EUM Server retains a large number of snapshots that can include base, virtual, and Ajax pages as well as iFrames. You can limit the number of snapshots retained by the EUM server by setting a global maximum, reducing the time that they are retained, or by filtering snapshots based on the network response time.

Setting the Global Limit for Snapshots

You set the global limit on the number of snapshots to be retained per minute with the configuration `browserBeaconSampling.maxSamples`. The default value is `1000`. Once the limit is reached, all snapshots will be dropped indiscriminately. The limit can be globally configured through the `eum.properties` file.

1. Open `$APPDYNAMICS_HOME/EUM/eum-processor/bin/eum.properties` with a text editor.
2. Add the following line to `eum.properties`, replacing `<global_limit>` with the global maximum number of snapshots to retain.

```
browserBeaconSampling.maxSamples = <global_limit>
```

3. Restart the EUM Server.

Reduce the Lifespan of Event Snapshots

Another way to limit the number of EUM snapshots is to reduce the number of days that the event snapshots are retained. Event snapshots only apply to the crash reports, code issues, and IoT errors and are stored in the local blob store: `$APPDYNAMICS_HOME/EUM/eum-processor/store`

By default, the EUM Server retains the event snapshots for 90 days. If your Events Service retains events for fewer days (e.g., 14 days), you can safely change the EUM Server's retention period to be the same as the Events Service's retention period. If the EUM Server retains the event snapshots for fewer days than the Events Service, however, you may run into errors when viewing older events in the Controller UI.

> When reducing the lifespan of event snapshots, you are not modifying the retention period of the Controller or the Events Service.

Setting the Lifespan for the Event Snapshots

1. Open `$APPDYNAMICS_HOME/EUM/eum-processor/bin/eum.properties` with a text editor.
2. Add the following line to `eum.properties`, replacing `<no_of_days>` with the number of days that you'd like to retain the event snapshots. The default is 90.

```
eventSnapshotStore.lifespanInDays = <no_of_days>
```

3. Restart the EUM Server.

Filtering Snapshots Based on the Network Response Time

You set a threshold that filters the snapshots based on the network response time. If the network response time is at or below the configured threshold, the snapshot is then retained. You set the threshold with the configuration `browserBeaconSampling.hierarchyAwareSamplerPageUXThreshold`.

Below are the supported threshold values and the snapshots that would be retained. The default value is `Slow`.

- `Normal` - Using this threshold value will retain all snapshots.

- `Slow` - Using this threshold value will retain snapshots having a network response time of slow, very slow, and stalled.

Setting the Threshold for the Network Response Time

1. Open `$APPDYNAMICS_HOME/EUM/eum-processor/bin/eum.properties` with a text editor.
2. Add the following line to `eum.properties`, replacing `<threshold>` with one of the supported thresholds (`Normal` or `Slow`) for retaining snapshots.

```
browserBeaconSampling.hierarchyAwareSamplerPageUXThreshold = "<threshold>"
```

3. Restart the EUM Server.

Turn On Access Logs

By default, server access logging for the EUM Server's underlying application server is turned off. To turn it on, open `$APPDYNAMICS_HOME/EUM/eum-processor/conf/local-eum-processor.yml` with a text editor and find the following section under the `server` entry:

```
 requestLog:
        appenders: []
```

Add the following information:

```
requestLog:
        timeZone: UTC
        appenders:
          - type: file
            archive: true
            currentLogFilename: ../logs/access.log
            archivedLogFilenamePattern: ../log/accedd-%d.log.gz
```

Save the file and restart the EUM Server.

EUM Server Configuration File

You can configure the EUM Server by setting properties in the file `$APPDYNAMICS_HOME/EUM/eum-processor/bin/eum.properties`. You are recommended to copy the reference sample file `$APPDYNAMICS_HOME/EUM/eum-processor/bin/eum.sample.properties` to `$APPDYNAMICS_HOME/EUM/eum-processor/bin/eum.properties`, modify the settings to fit your needs, and then restart the EUM Server so that the new settings are applied.

The table below lists and describes the supported EUM properties, lists defaults, and specifies whether the property is required. The values for the database properties must conform with the MySQL syntax rules given in Schema Object Names.

EUM Property	Default	Required?	Description
`onprem.dbHost`	`dbHost`	Yes	The name of the database host.
`onprem.dbPort`	`3388`	Yes	The port to the database host.
`onprem.dbSchema`	`eum_db`	Yes	The name of the EUM database.
`onprem.dbUser`	`eum_user`	Yes	The user name for the EUM database.
`onprem.dbPassword`	N/A	Yes	The user password to the EUM database. The password can consist of any ASCII character except the characters '^', '/', or '$'.
`onprem.eventSnapshotDiskAllowance`	-1	No	The maximum disk space allotted for storing event snapshots. The default value of -1 allots unlimited disk space to store event snapshots. You can specify a positive integer representing the maximum number of bytes for storing event snapshots.
`onprem.fileStoreRoot`	`../store`	No	The path to the directory storing EUM data such as snapshots.
`onprem.crashReportExpirationDays`	365	No	The number of days that crash reports are retained.
`onprem.resourceSnapshotExpirationDays`	15	No	The number of days that resource snapshots are retained.

`onprem.resourceSnapshotDiskAllowance`	21474836480 (20 GB)	No	The maximum disk space allotted for storing resource snapshots. The default maximum disk space is 20 GB or 21474836480 bytes. You can specify a positive integer representing the maximum number of bytes for storing resource snapshots.
`processorServer.httpPort`	`7001`	No	The HTTP port to the EUM Processor. The EUM Processor runs in one process containing the collector, aggregator, crash-processor, and monitor services.
`processorServer.httpsPort`	`7002`	No	The HTTPS port to the EUM Processor.
`processorServer.httpsProduction`	`true`	No	The flag for turning enabling (true) or disabling HTTPS to the EUM Processor.
`processorServer.keyStorePassword`	N/A	No	The password to the Key Store for the EUM Processor.
`processorServer.keyStoreFileName`	`bin/ssugg.keystore`	No	The path to the file that stores the password to the Key Store for the EUM Processor.
`processorServer.collectorHttpPort`	`7001`	No	The HTTP port of the EUM Collector. By default, the EUM Collector shares the same port as the EUM Processor, but you can configure the port to be different. The EUM Collector receives the metrics sent from the JavaScript agent.
`processorServer.collectorHttpsPort`	`7002`	No	The HTTPS port of the EUM Collector.
`analytics.enabled`	`true`	Yes	The flag for enabling or disabling the Analytics Server.
`analytics.serverScheme`	`http`	No	The network protocol for connecting to the Analytics Server. It is only required with `analytics.enabled=true`.
`analytics.serverHost`	`events.service.hostname`	No	The hostname of the Analytics Server. It is only required with `analytics.enabled=true`.
`analytics.port`	`9080`	No	The port to the Analytics Server. It is only required when `analytics.enabled=true`.
`analytics.accountAccessKey`	`access-key`	No	The access key for connecting to the Analytics Server. It is only required with `analytics.enabled=true`.
`analytics.eventTypeLifeSpan.0.eventType`	`BrowserRecord`	No	The type of event to be saved. The following values are supported: • `BrowserRecord` • `MobileSnapshot` • `SessionRecord` • `MobileSessionRecord` If this property is set, you must also set `analytics.eventTypeLifeSpan.0.lifeSpan`.
`analytics.eventTypeLifeSpan.0.lifeSpan`	`8`	No	The number of days to retain the event records specified by `analytics.eventTypeLifeSpan.0.eventType`. If this property is set, you must also set `analytics.eventTypeLifeSpan.0.eventType`.
`analytics.eventTypeLifeSpan.1.eventType`	`MobileSnapshot`	No	The type of event to be saved. The following values are supported: • `BrowserRecord` • `MobileSnapshot` • `SessionRecord` • `MobileSessionRecord` If this property is set, you must also set `analytics.eventTypeLifeSpan.1.lifeSpan`.
`analytics.eventTypeLifeSpan.1.lifeSpan`	`8`	No	The number of days to retain the event records specified by `analytics.eventTypeLifeSpan.1.eventType`. If this property is set, you must also set `analytics.eventTypeLifeSpan.1.eventType`.
`analytics.eventTypeLifeSpan.2.eventType`	`SessionRecord`	No	The type of event to be saved. The following values are supported: • `BrowserRecord` • `MobileSnapshot` • `SessionRecord` • `MobileSessionRecord` If this property is set, you also must set `analytics.eventTypeLifeSpan.2.lifeSpan`.

analytics.eventTypeLifeSpan.2.lifeSpan	8	No	The number of days to retain the event records specified by analytics.eventTypeLifeSpan.2.eventType. If this property is set, you must also set analytics.eventTypeLifeSpan.2.eventType.
analytics.eventTypeLifeSpan.3.eventType	MobileSessionRecord	No	The type of event to be saved. The following values are supported: • BrowserRecord • MobileSnapshot • SessionRecord • MobileSessionRecord If this property is set, you also must set analytics.eventTypeLifeSpan.3.lifeSpan.
analytics.eventTypeLifeSpan.3.lifeSpan	8		The number of days to retain the event records specified by analytics.eventTypeLifeSpan.3.eventType. If this property is set, you must also set analytics.eventTypeLifeSpan.3.eventType.
onprem.mobileAppBuildTimeSeriesRequestCountRollupDays	7	No	The EUM Collector searches for the dSYM file in the beacon traffic for the configured number of days. If the dSYM file is not present during the configured time frame, a warning message is displayed in the Controller UI.
onprem.maxNumberOfMobileBuildsWithoutDsym	10	No	The maximum number of visible missing dSYM files in the Controller UI.
collection.sessionEnabled	true	No	The flag for enabling or disabling browser/mobile session collection. If you are upgrading the EUM Server from versions 4.2 and lower to 4.3 or higher, the default is false.
collection.accessControlAllowOrigins.{n}	*	No	By default, the EUM Collector responds with Access-Control-Allow-Origin: *. You can limit CORS to certain domains by using an integer property assigned to a URL as in the following: • collection.accessControlAllowOrigins.0=http://example1.com • collection.accessControlAllowOrigins.1=http://example2.com • collection.accessControlAllowOrigins.2=http://example3.com
eventSnapshotStore.lifespanInDays	90	No	The number of days that the event snapshots stored in the EUM Server's local blob store ($APPDYNAMICS_HOME/EUM/eum-processor/store) are retained. The event snapshots only apply to crash reports, code issues, and IoT errors.
sessionization.webSessionRetentionMins	5	No	The number of minutes that browser sessions are retained after they are closed. This allows browser sessions that begin and end at different times to be retained. The longer the configured retention time, the larger the number of closed sessions held in memory, resulting in higher memory usage.
sessionization.mobileSessionRetentionMins	5	No	The number of minutes that mobile sessions are retained after they are closed. This enables mobile sessions that begin and end at different times to be retained. The longer the configured retention time, the larger the number of closed sessions held in memory, resulting in higher memory usage.
throttling.resourceSnapshot.maxTotalPerMinPerAccount	1000	No	The maximum number of total resource snapshots retained each minute for an account.
throttling.resourceSnapshot.maxNormalPerMinPerAccount	800	No	The maximum number of resource snapshots of pages with a "Normal" user experience that are retained each minute for an account. In general, you want the number for this property to be smaller than that for throttling.resourceSnapshot.maxTotalPerMinPerAccount, so you can also retain resource snapshots of pages with a "Slow", "Very Slow", or "Stall" user experience.
throttling.session.maxTrackedSessionsPerAccount	50000	No	The maximum number of active sessions and unexpired closed sessions that are stored in memory for an account. When the maximum is reached, events that create new sessions will be dropped. This setting helps to control the memory used for sessions at the account level.

From EUM Server version 4.5.1 and later, the property crashProcessing.sessionEnabled is no longer supported. Instead, the association of crashes with sessions is enabled by default. If you are using an earlier version (<4.5.1) of the EUM Server and want to upgrade to 4.5.1 or higher, you will need to remove the property crashProcessing.sessionEnabled from the eum.properties file to prevent the EUM Server from throwing errors.

EUM Server Endpoints

Related pages:

- Port Settings
- Synthetic Server Endpoints

The EUM Server has different endpoints serving distinct functions. This page provides a reference for testing the health and getting information about on-prem EUM Servers.

The endpoints include the following:

- **EUM API** - acts as the interface between the EUM Server and the Controller. The Controller retrieves EUM data from the EUM Server through the EUM API endpoint.
- **EUM Collector** - collects metrics from the EUM agents. The JavaScript Agent and Mobile Agents transmit data to the EUM Server through the EUM Collector endpoint.
- **EUM Aggregator** - collects and rolls up all the metrics per application and provide an interface for Controllers to download the metrics by application and timestamp.
- **Screenshot Service** - collects and serves image tiles that form mobile screenshots. The Mobile Agents transmit the tiles to the Screenshot Service, and the Controller retrieves the tiles to display the screenshots in mobile sessions.

EUM Server Endpoint URLs

The table below lists the endpoints, the default URL, and the supported paths.

EUM Server Endpoint	Default URL	Paths / Description
EUM Collector	`http(s)://<domain-name>:7001/eumcollector`	`/adrum.gif` — Receives image beacons from the JavaScript Agent. `/beacons/browser` — Earliest endpoint for receiving CORS beacons from the JavaScript Agent. `/beacons/browser/v1/*` — The V1 endpoint for receiving CORS beacons from the JavaScript Agent. `/beacons/browser/v2/*` — The V2 and latest endpoint for receiving CORS beacons from the JavaScript Agent. `/get-version` — Returns the version, build, and commit, and timestamp of the EUM Processor. `/iot/v1/application/*` — The endpoint for IoT REST APIs. See the IoT REST API reference documentation for details. `/ping` — Returns whether the EUM Collector is accessible and running. `/mobileMetrics?version=2` — The endpoint used by the Mobile Agents to send mobile beacons via HTTP POST. `/whoami` — Returns the IP address, geo location, and information of the client making the request.
EUM Aggregator	`http(s)://<domain-name>:7001/eumaggregator`	`/currentTime` — Returns the current time of the EUM Aggregator as a Unix timestamp. `/get-version` — Returns the version, build, and commit, and timestamp of the EUM Processor. `/ping` — Returns whether the EUM Aggregator is accessible and running.
Screenshot Service	`http(s)://<domain-name>:7001/screenshots/v1`	`/version` — Returns the version, build, commit, and timestamp of the Screenshot Service.

Install and Host a Custom Geo Server for Browser RUM

Related pages:

- Host a Geo Server

By default, the locations of end-users are resolved using public geographic databases. You can host an alternate geo server for your countries, regions, and cities instead of using the default geo server hosted by AppDynamics.

You may prefer to host your own geo server because:

- You have intranet applications where the public IP address does not provide meaningful location information, but the user's private IP does.
- You have a hybrid application where some users access the application from a private location and some access it from a public one. If a user doesn't come from a specific private IP range mapped by the custom geo server, the system can be set to default to the public geo server.

To host a custom geo server:

1. Download the Geo Server File
2. Set the Location of the Geo Server
3. Create the IP Mapping File

Requirements for Geo Server Host

- 2 GB of memory
- Java 8

Download and Install the Geo Server File

Download the `GeoServer.zip` file from AppDynamics at https://download.appdynamics.com/download.

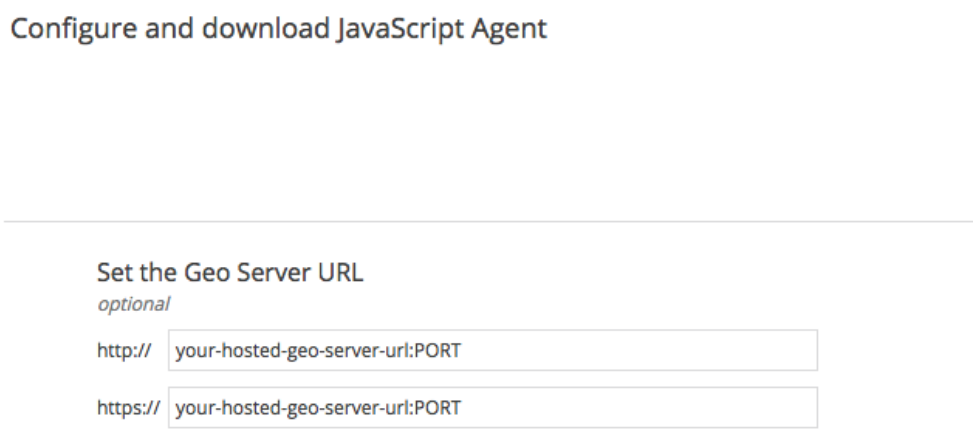
Uncompress the zip to a `GeoServer` folder with the following structure:

```
GeoServer
  schema.xsd                    <-- schema for geo-ip-mapping.xml configuration
  geo
     WEB-INF
        classes
           logback.xml       <-- configure logging in here
           ...
        web.xml               <-- other configurations here
        ...
  |    geo-ip-mappings.xml       <-- configure geo ip mapping here
   ...
```

To install the geo server, copy the `geo` folder to the `TOMCAT_HOME/webapps` of your Tomcat server. Do not deploy the server in the same container as the Controller.

Set the Location of the Geo Server

Enter the URL, including the context root, of your hosted geo server in the **Geo Server URL** field in the **Browser RUM** configuration screen in the Controller UI as shown below.



⚠️ If you are using manual injection for your JavaScript agent, you must make sure that the copy of the script that you use is one that you have downloaded *after* this URL is set.

Create the IP Mapping File

The `geo-ip-mappings.xml` IP mapping file specifies the locations for which Browser RUM provides geographic data. It maps IP addresses to geographic locations.

Use the sample file in the `geo` subdirectory as a template. Any modifications at runtime are reloaded without a restart.

This file contains a `<mapping>` element for every location to be monitored. The file has the following format.

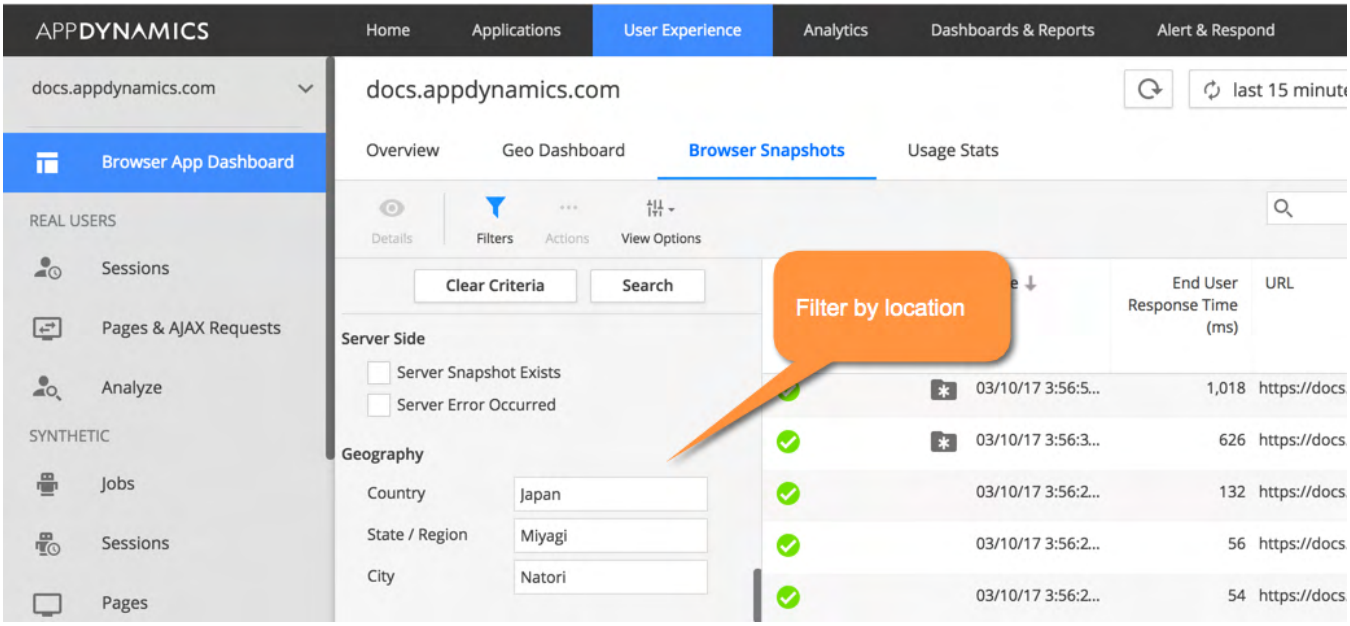
```
<mappings>
        <mapping>
                <subnet from="192.168.1.1" mask="255.255.255.0"/>
                <location country="United States" region="California" city="San Francisco"/>
        </mapping>

        <default country="United States" region="California" city="San Francisco"/>
</mappings>
```

You can also use IP-range-based mapping instead of subnet-based:

```
<mapping>
    <ip-range from="10.240.1.1" to="10.240.1.254" />
    <location country="France" region="Nord-Pas-de-Calais" city="ENGLOS" />
</mapping>
```

This data is visible in browser snapshots and can be used to filter browser snapshots for specific locations: The `<country>`, `<region>`, and `<city>` elem ents are required. If the values of <country> and <region> do not correspond to an actual geographic location already defined in the geographic database, map support is not available for the location in the map panel, but Browser RUM metrics are displayed for the location in the grid view of the geographic distribution, end user response time panel, trend graphs, browser distribution panel, and in the Metric Browser. The `<city>` element can be a string that represents the static location of the end-user. You will notice a `<default>` element. If there is an IP address that is not covered by your IP mapping file, this is the value that is used. To use a public geo server for non-covered IP addresses, see Using a Hybrid Custom-Public Geo Server Setup.



The valid names for country and region are those used in the map in the geo dashboard. You can hover over a region in the dashboard to see the exact name (including spelling and case) of the region. See The Browser Geo Dashboard View.

Using a Hybrid Custom-Public Geo Server Setup

If you want Browser RUM to evaluate any non-mapped IP address using the public geo server, remove the `<default>` element. In this case, locating any non-mapped IP address is done in the EUM cloud, not locally.

Customize File Locations

You can customize where certain files are stored in the GeoServer directory.

Change Log Location

By default, logs are written to `TOMCAT_HOME/logs`, but you can configure this using `TOMCAT_HOME/webapps/geo/WEB-INF/classes/logback.xml`. Open the file with a text editor and edit the `LOG_HOME` property.

```
<property name="LOG_HOME" value="{path-to-file}/logs"/>
```

Change Mapping File Location

By default, the geo server looks for `geo-ip-mappings.xml` in `TOMCAT_HOME/webapps/geo/`. To change the location, open `TOMCAT_HOME/webapps/geo/WEB-INF/web.xml` with a text editor and change value for `AD_GEO_CONFIG_FILE`.

```
<web-app ...>
    <!-- ... -->
    <servlet>
        <servlet-name>FrontControllerServlet</servlet-name>
        <servlet-class>com.appdynamics.eum.geo.web.FrontControllerServlet</servlet-class>
        <context-param>
            <param-name>AD_GEO_CONFIG_FILE</param-name>
            <param-value>{path-to-file}/geo-ip-mappings.xml</param-value>
        </context-param>
        <!-- ... -->
    </servlet>
    <!-- ... -->
</web-app>
```

In previous versions of the geo server, the enclosing tag was a `<context-param>`. This has now been changed to an `<init-param>`.

For On-Premises EUM Servers Only: Use geo-ip-mappings.xml

If your installation uses an on-premises EUM Server *and* you have internal browsers from the same network as the Server that you want to identify, instead of setting up a separate custom geo-server, you can choose to simply modify the EUM Server's `geo-ip-mappings.xml` file as described above. The sample is in the `bin` directory of the EUM Server. The EUM Server automatically reads the file and uses it first to try and resolve the location, before using the Neustar IP database.

Precedence in Resolving Locations

The custom geo server resolves locations based on the following precedence, from highest to lowest:

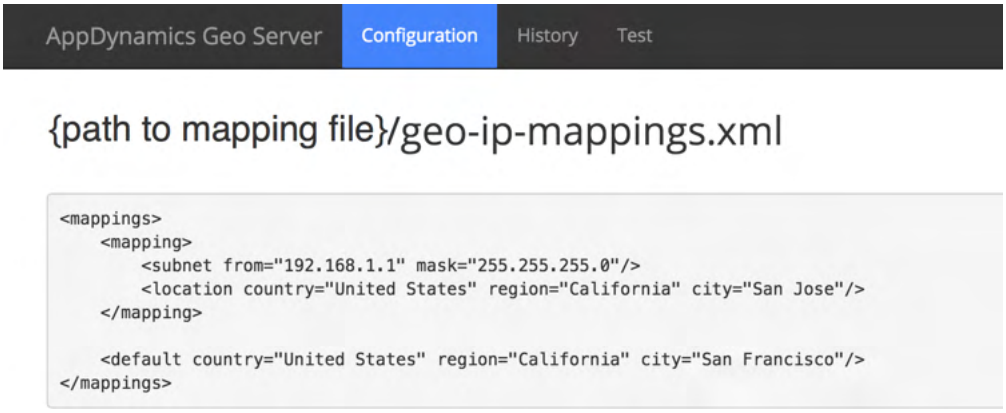
- An IP address set by customizing the JavaScript agent. For more information, see Set the Origin Location of the Request.
- An explicit query parameter: for example, `http://mycompany.com/geo/resolve.js?ip=196.166.2.1`.
- An IP provided using the `AD-X-Forwarded-For` header
- An IP provided using the `X-Real-IP` header
- An IP provided using the `X-Forwarded-For` header
- The remote address of the HTTP request

Debugging

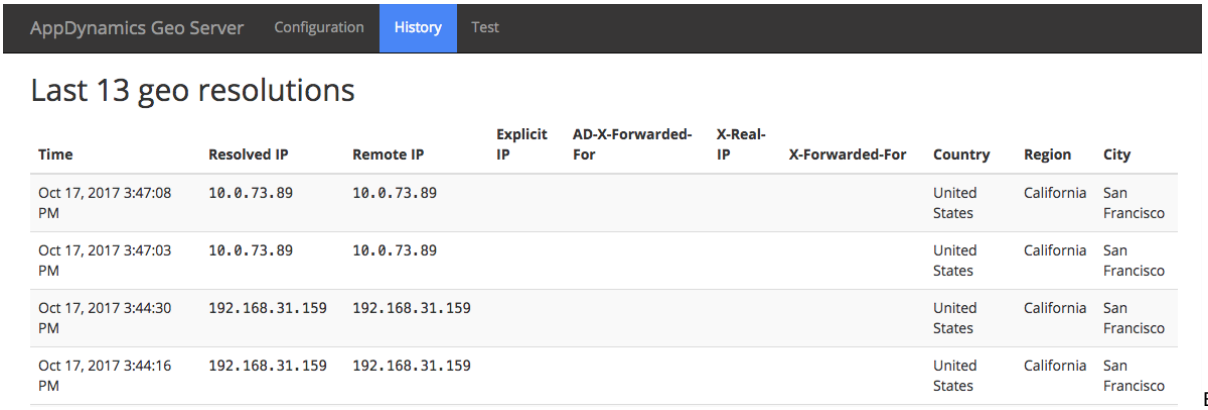
Because this debugging feature has a small performance impact, it should be turned off before putting the geo server into production.

To aid in debugging, the geo server ships with a debugging web interface enabled. You can reach this interface by navigating to `http://<host>:<port>/geo/debug` with a web browser.

The first tab, **Configuration**, displays the contents of the mapping file currently in use.



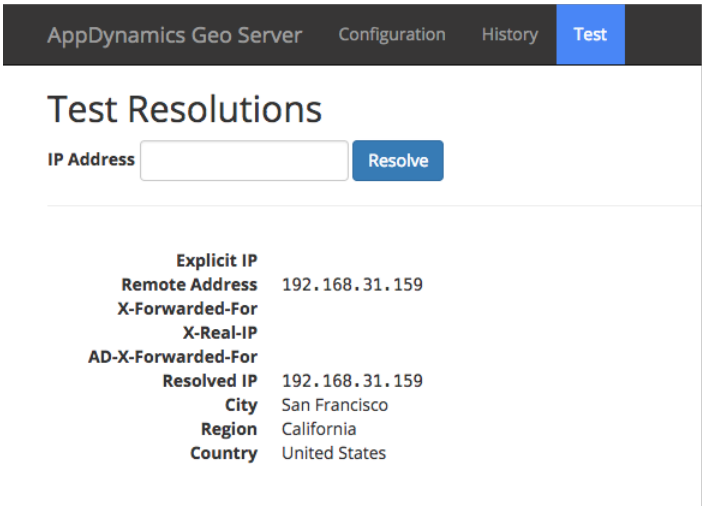
The second tab, **History**, shows the last few geo resolutions that have been performed.



By default, the last 20 resolutions are shown, but this can be configured in `TOMCAT_HOME/webapps/geo/WEB-INF/web.xml`.

```
<web-app ..>
    <!-- ... -->
    <servlet>
        <servlet-name>FrontControllerServlet</servlet-name>
        <servlet-class>com.appdynamics.eum.geo.web.FrontControllerServlet</servlet-class>
        <!-- ... -->
        <init-param>
            <param-name>HISTORY_MAX_COUNT</param-name>
            <param-value>20</param-value>
        </init-param>
    </servlet>
    <!-- ... -->
</web-app>
```

The third tab, **Test**, can be used to test the mapping file by trying to resolve an arbitrary IP address.



When you first navigate to this tab, it shows the geo resolution for your browser's IP address. The form in this tab can be used to try the resolution of another IP address.

Disabling debug

Open `TOMCAT_HOME/webapps/geo/WEB-INF/web.xml` and set `DEBUG_ENABLED` to `false`.

```
<web-app ..>
    <!-- ... -->
    <servlet>
        <servlet-name>FrontControllerServlet</servlet-name>
        <servlet-class>com.appdynamics.eum.geo.web.FrontControllerServlet</servlet-class>
        <!-- ... -->
        <init-param>
            <param-name>DEBUG_ENABLED</param-name>
            <param-value>false</param-value>
        </init-param>
    </servlet>
    <!-- ... -->
</web-app>
```

EUM Server Component Versions

This page describes how to check version information and the version of components bundled with the EUM Server. This information is useful when troubleshooting the system or performing other administrative tasks.

> ⚠ AppDynamics maintains and updates the bundled components as part of the AppDynamics Platform. Do not attempt to upgrade a bundled component independently of the platform upgrade procedure.

EUM Server Version

To view the version of your running EUM Server, run the following:

```
curl http(s)://<domain-name>:7001/v2/version
```

To get more information about the EUM Server, see EUM Server Endpoints.

Bundled MySQL Database Version

The AppDynamics EUM Server uses MySQL as its default database, where it stores license/account information, metadata, and applications names. The MySQL database files are installed in `<eum_home>/data` by default.

The latest AppDynamics release bundles MySQL version 5.7.33.

Bundled Java Version

The EUM Server bundles and uses Java 1.8.0_162.

Upgrade the Production EUM Server

This page describes how to upgrade an EUM Server to the latest production installation. This is usually done alongside an upgrade to the other platform components, such as the Controller and Events Service.

Who Should Use This Document

Anyone wanting to upgrade the EUM Server to the latest available version should use this document.

Before You Begin

Before you start upgrading the EUM Server, make sure that you are using the correct update order.

Upgrade Procedure

The instructions below show you how to upgrade your EUM Server to the latest version of the production EUM Server. Because in the EUM Server 4.4, the EUM MySQL database was moved from the Controller host machine to the EUM Server host machine, there are separate instructions below to help you migrate your data from versions earlier than 4.4 to the latest version. If you are upgrading from EUM Server 4.4 or higher to the latest version, you will not have to migrate your data, but you are advised to make a backup of your data.

Using SSL with the EUM Server

If you are upgrading to EUM Server 4.5.6 or higher, you are recommended to downgrade the version of the JRE bundled with the EUM Server to 1.8.0 _152 to avoid performance issues.

Troubleshoot EUM Server Installation

The following sections provide troubleshooting information for the EUM Server installation.

End User Data Does Not Appear in the Controller

If end user data does not appear in the Controller, follow these steps to troubleshoot the installation:

1. Check the Controller logs for errors in attempting to connect to the EUM Server. Also, see if the Controller UI allows you to enable EUM. If so, it's likely that the connection between the Controller and EUM Server is working.
2. Check the logs of the EUM Server, especially `<EUM_home>/logs/eum-processor.log`. In the log, verify that the server started successfully and is receiving beacons from agents.
3. Make sure that the EUM JavaScript Agent is actually injected into the monitored page and that the agent can load the remote JavaScript.
4. Use browser debugging tools to check for JavaScript errors in the monitored page.

License Not Installed

If the installer indicates that it was not able to install the license, or after installation, if the EUM Server fails to start with a license exception, try installing the license manually.

With the Controller running and accessible to the EUM Server machine, install the license manually. Before starting, make sure the `license.lic` file is at an accessible location on the EUM Server machine. Then install the license as follows:

1. Verify that the `JAVA_HOME/bin` is in the system PATH variable and points to a Java 1.7 instance.
2. In Windows, open an elevated command prompt (run as administrator).
3. From the command line, navigate to the `eum-processor` directory under your AppDynamics home.
4. From the `eum-processor` directory, run the following script:
    - On Linux:
      `./bin/provision-license <path_to_license_file>`
    - On Windows:
      `bin\provision-license.bat <path_to_license_file>`

EUM License Has Not Been Provisioned for an Account

Follow the instructions below to provision new EUM licenses for accounts on a multi-tenant Controller UI.

1. Navigate to the **Administration** page of your on-premises Controller: `http(s)://<hostname>:<port>/controller/admin.jsp`
2. Click **Accounts**.
3. Select the account name that you want to provision EUM for and click **Edit**.
4. Scroll down to the **End User Monitoring (EUM)** panel.
5. From the **Browser Real User Monitoring** section:
    a. Copy the EUM license key from your license file into the **EUM License Key** field.
    b. Select a license type (**EUM Lite** or **EUM Pro**) from the **License Type** dropdown.
    c. Enter your allotted Browser RUM units into the **Browser RUM Units Licensed** field.
    d. Set overages from the **Allow Overages** dropdown.
6. Complete the steps above for the **Mobile Real User Monitoring** section.
7. Click **Save**.

Exception When Updating Application Store

You may need to tune the database thread pool. The EUM Server ships with a blank c3p0 xml configuration file to help manage this. For information on using c3p0, see the docs here. You should make your changes to `<eum_server_home>/bin/c3p0.xml`.

"Too Many Open Files" Exception

Exception messages such as the following indicate insufficient open file descriptor limits on the EUM Server machine:

```
java.io.IOException: Too many open files
```

See EUM Server Deployment for operating system requirements, including recommended settings for `nofile` and `nproc` limits for the EUM Server operating system.

Controller Cannot Reach the Events Service

In the Administration Console, make sure that the Controller setting named `eum.es.host` is set to the correct connection settings for the Events Service instance, and that the Events Service is properly installed and running at that location. If the Events Service is a cluster with a load balancer in front of it, this should be the VIP of the Events Service as exposed at the load balancer.

For more information, see Connect to the Events Service.

Events Service Deployment

Related pages:

- Events Service Requirements
- Install the Events Service on Linux
- Install the Events Service on Windows

The AppDynamics Events Service is the on-premises data storage facility for unstructured data generated by Application Analytics, Database Visibility, and End User Monitoring deployments.

> The following information and instructions are intended for on-premises deployments only. SaaS deployments are managed by AppDynamics.

About the Events Service

The MySQL database embedded in the Controller stores application metric and configuration data generated by the Controller. While an embedded database is sufficient for storing this type of data, the high-volume, performance-intensive nature of analytics data requires dedicated, horizontally scalable storage. In an AppDynamics deployment, this role is served by the AppDynamics Events Service.

If you are installing the server components for End User Monitoring, Application Analytics, or Database Visibility, you need to use a scaled-out on-premises Events Service. Scaled-out means that the service is not installed on the Enterprise Console host. This type of Events Service can be deployed as a single node or a cluster of three or more nodes based on your needs. Additionally, you can add nodes to a scaled-out Events Service after you install it. It is not recommended that you add the Controller host as part of the cluster. For information, see Administer the Events Service.

However, for data redundancy and storage scalability or if you are using End User Monitoring, Application Analytics or Database Visibility in an on-premises Controller deployment, you need to deploy a dedicated Events Service installation.

Multiple AppDynamics components that depend on the Events Service should use the same Events Service instance or cluster.

Deployment Topology Overview

You can deploy the Events Service to a single node or to a cluster of three or more nodes. Clusters are horizontally scalable, so nodes can be added as your data storage requirements grow. A cluster also provides data replication and redundancy, helping to ensure data integrity in the event of a node failure.

The Controller includes an embedded Events Service instance used by the Database Visibility product by default. However, the embedded Events Service is not meant to be used with production Application Analytics or EUM installations, since it runs on the same machine as the Controller and does not offer data replication or scalability. It may be used for small-scale environments, especially those intended for demonstration or learning purposes. Note however that it's not possible to migrate data from the embedded Events Service to an external Events Service instance if upgrading later.

Your Events Service can be deployed to support multiple Controllers, becoming a shared Events Service.

Single Node Deployment

In a single node Events Service deployment, the Events Service runs on a dedicated machine. The Controller and other Events Service clients can connect directly to the Events Service node or through a load balancer. Deploying a single node Events Service behind a load balancer allows you to grow the deployment to a multi-node cluster easily, without having to modify the clients.

> Single node deployment is recommended for test environments only. Production environments should deploy a multi-node cluster (see below for details).

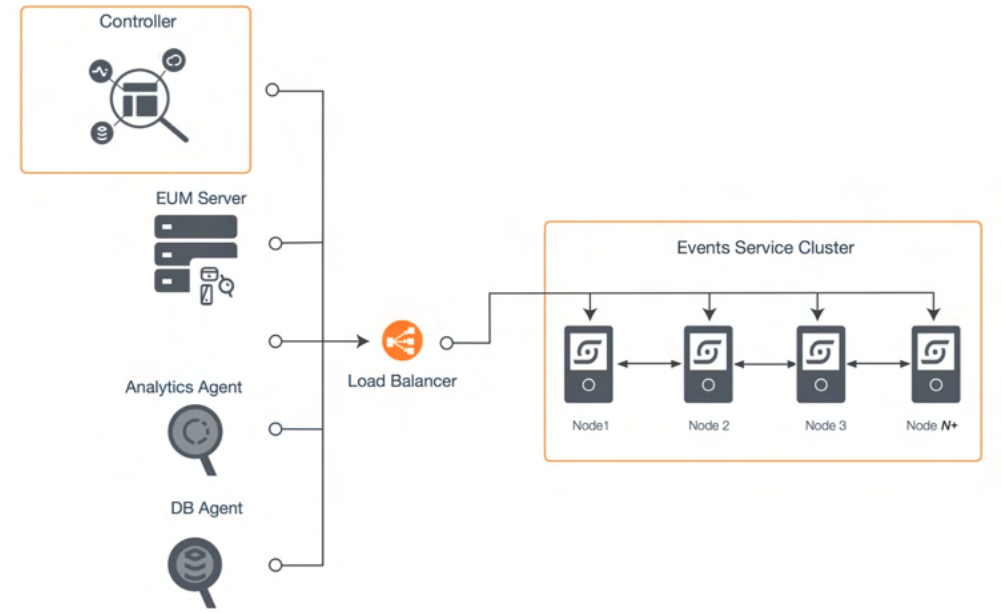
Multi-Node Cluster

A multi-node cluster is made up of three or more nodes. With a cluster, the Controller and other Events Service clients, the EUM Server and Analytics Agent connect to the Events Service through a load balancer, which distributes load to the Events Service cluster members.

AppDynamics recommends multi-node clusters for production environments. Multi-node clusters provide the following benefits:

- Safeguards against data loss: multi-node clusters replicate your data. If one node goes down in production, you would have at least two more nodes storing your data. Additionally, the Events Service continues to run as long as one node runs. Should a node go down in a single node deployment, the Events Service stops running and you would lose your data.
- Provides redundancy: in a multi-node cluster, you can swap nodes.

In a single-node deployment, connect through a load balancer or directly to the Events Service.



The nodes in a cluster swap a large amount of data. For this reason, when deploying a cluster, make sure to install all cluster nodes within the same local network, ideally, attached to the same network switch.

Supported Deployments

Use the following table to determine which deployment types and environments are supported for different versions of the Events Service.

Deployment Types	Development Environment	Production Environment
Controller Embedded Events Service (version 4.3.x)	Yes	No. Only shipped through the Controller installer.
Controller Embedded Events Service (version 4.4.x)		
Controller Embedded Events Service (version 4.5.x)		
Single Node Events Service (Dedicated Host) (version 4.3.x)		No
Single Node Events Service (Dedicated Host) (version 4.4.x)		
Single Node Events Service (Dedicated Host) (version 4.5.x)		
Multi-Node Clustered Events Service (3+ nodes) (version 4.3.x or later)		Yes
Multi-Node Clustered Events Service (3+ nodes) (version 4.4.x or later)		
Multi-Node Clustered Events Service (3+ nodes) (version 4.5.x or later)		

Shared Events Service

You can connect multiple Controllers to a single on-premises Events Service deployment using the same procedure that is required to connect a single Controller. You would need to configure the Controller URLs for the Events Service to point to the shared Events Service, and make sure the Controller keys are correct. The Controllers can then handle syncing to the shared Events Service. There are no additional requirements.

When compared to a multiple Events Service cluster configuration, a shared Events Service configuration requires less maintenance and lowers cost. Since the Events Service is horizontally scalable, having a single large instance provides the same functionality as multiple instances.

There is no limit to the number of Controllers that can share an Events Service. However, it is recommended that you use separate Events Services for dev and prod. Work with your AppDynamics account representative if you plan to expand your shared Events Service cluster.

Default Ports

The default ports used by the Events Service are:

- Events Service API Store Port: 9080
- Events Service API Store Admin Port: 9081

The Events Service cluster members use additional ports for internal communication among the cluster members. All the ports used within the cluster are listed in the Events Service configuration file, `conf/events-service-api-store.properties`.

Secure the Events Service

By default, the `ad.es.node.http.enabled` property in the Events Service configuration file is set to `false`. To debug or troubleshoot issues with the Events Service, you or the AppDynamics Support team can enable this property and allow HTTP requests (such as node stats) in ElasticSearch by changing the setting to `true`. Following the debugging session, immediately disable this property to prevent any (potential) security vulnerabilities and restart the Events Service. The downtime is dependent on the hardware and the service will be unavailable until it restarts.

Events Service Requirements

Related pages:

- Prepare the Events Service Host

This page describes general hardware and software requirements for the machines that host Events Service nodes.

General Requirements

- Determine which version of the Events Service that is compatible with your other platform components.
- Use a supported Windows 64-bit or Linux 64-bit based operating system supported by the platform. See Platform Requirements.
- Solid-state drives (SSD) can significantly outperform hard disk drives (HDD), and are therefore recommended for production deployments. Ideally, the disk size should be 1 TB or larger.
- The Events Service must run on dedicated machines with identical directory structures, user account profiles, and hardware profiles.
- For heap space allocation, AppDynamics recommends allocating half of the available RAM to the Events Service process, with a minimum of 7 GB up to 31 GB.
- When testing the events ingestion rate in your environment, it is important to understand that events are batched. Ingestion rates observed at the scale of a minute or two may not reflect the overall ingestion rate. For best results, observe ingestion rate over an extended period of time, several days at least.
- The Events Service requires Java 8u172.
- Keep the clocks on your application, Controller, and Events Service nodes in sync to ensure consistent event time reporting across the AppDynamics deployment.
- Your firewall should not block the Events Service REST API port 9080, otherwise, the Enterprise Console will not be able to reach the Events Service remotely.

Hardware Capacity and Resource Planning

When estimating your hardware requirements, the first step is to determine the event ingestion rate (for Transaction Analytics) or the amount of data being indexed (for Log Analytics). This helps you to determine the number of analytics license units you will need.

Once you determine your license units requirements, it is important to consider other factors that affect the hardware capacity, such as the processing load of queries run against the Events Service and the actual type of hardware used. A physical server is likely to perform better than a virtual machine. You should also take into account seasonal or daily spikes in activity in your monitored environment in your considerations.

An event is the basic unit of data in the events service. In terms of application performance management, a Transaction Analytics event corresponds to a call received at a tier. A business transaction instance that crosses three tiers, therefore, would result in three events being generated. In application performance management metrics, the number of business transaction instances is reflected by the number of calls metric for the overall application. In End User Monitoring, each page view equates to an event, as does each Ajax request, network request, or crash report.

Events Service Node Sizing Based on License Units

You can plan your hardware requirements with the data in the section. It describes recommended hardware configurations (in the context of Amazon EC2 instance types) corresponding to the number of license entitlement units for Log and Transaction Analytics. See License Entitlements and Restrictions for details about license units for Log and Transaction Analytics.

For additional Events Service sizing information, see the following AppDynamics Community articles:

- Understanding EUM and Events Service concepts (describes the concepts necessary to build the profile)
- Build the Analytics traffic profile
- Size the Events Service and EUM using the profile (contains the Analytics Recipe Book for on-premises configuration)
- Limit EUM and Analytics usage (describes how to configure rules to enforce Analytics trade-offs when deploying Analytics on-premises)

The hardware shown for each license amount represents the hardware capacity of a theoretical combined load of both Transaction Analytics and Log Analytics events. The numbers used were derived from actual tests that were performed with an uncombined load, from which the following numbers were extrapolated. Note that the test conditions did not include query load and so may not be representative of a true production analytics environment.

The following table shows sizing recommendations and describes the size of the cluster used for testing. This does not mean you are limited to a seven-node event service. If you need to go beyond seven nodes, contact your AppDynamics account representative to ensure proper sizing for your specific environment.

It is to be noted that the retention can be 8, 30, 60 or 90 days which will directly affect storage requirements.

Event Type	AWS Machine Instance Type										
	i2.2xlarge (61 GB RAM, 8 vCPU, 1600 GB SSD)				i2.4xlarge (122 GB RAM, 16 vCPU, 3200 GB SSD)				i2.8xlarge (244 GB RAM, 32 vCPU, 6400 GB SSD)		
	1 node	3 nodes	5 nodes	7 nodes	1 node	3 nodes	5 nodes	7 nodes	1 node	3 nodes	5 nodes

Transaction Analytics license units	20	37	44	63	22	41	84	113	53	94	120
Log Analytics license units	7	10	17	19	16	19	32	44	39	116	270

The following points describe the test conditions under which the license units-to-hardware profile mappings in the table were generated:

- Average Log event size in bytes: 350
- Average size of business transaction event: 1 KB
- Tiers in business transaction: 3

The tests were conducted on virtual hardware and programmatically generated workload. Real-world workloads may vary. To best estimate your hardware sizing requirements, carefully consider the traffic patterns in your application and test the Events Service in a test environment that closely resembles your production application and user activity.

Minimum Events Service Node Sizing

To configure the Events Service 3 nodes minimum are required.

Database Visibility Events Service Sizing

Database Visibility features use the Events Service for storage. The ingestion capacity and sizing profile for Database Visibility Analytics events are equivalent to that of Log Analytics, with the size of the raw event being about 2 kilobytes on average.

End User Monitoring Events Service Sizing

End User Monitoring includes Analytics-related features that send data to the Events Service.

In End User Monitoring, each page view equates to an event, as does each Ajax request, network request, or crash report. There can be a few dozen Ajax requests for every page load. In general, the ingestion capacity and sizing profile for EUM or Database Visibility Analytics events are equivalent to that for Log Analytics, with the size of the raw events being about 2 kilobytes on average.

To calculate the sizing for EUM, multiply the peak number of browser records in a day by 12 KB. If peak capacity is reached, the Events Service simply starts dropping traffic.

The table below provides details about the memory and storage of different types of browser records. The default retention period is configurable.

Browser Record Type	Memory Requirements Per Event	Optional	Default Retention
BasePage, iFrame, Virtual Page	1 KB / 1.5 KB (with sessions enabled)	No	8 days
Ajax requests	1 KB	Yes	

By default, Ajax requests are not stored in the Events Service.

Prepare the Events Service Host

Related pages:

- Events Service Requirements

This page describes how to prepare the machine that will host Events Service nodes, along with general requirements for the environment.

Network and Port Settings

The Controller and Events Service must reside on the same local network and communicate by the internal network. Do not deploy the cluster to nodes on different networks, whether relative to each other or to the Controller and the Enterprise Console. When identifying cluster hosts in the configuration, you will need to use the internal DNS name or IP address of the host, not the externally routable DNS name.

For example, in terms of an AWS deployment, use the private IP address such as `172.31.2.19` rather than public DNS hostname such as `ec2-34-201-129-89.us-west-2.compute.amazonaws.com`.

On each machine, the following ports need to be accessible to external (outside the cluster) traffic:

- Events Service API Store Port: 9080
- Events Service API Store Admin Port: 9081

For a cluster, ensure that the following ports are open for communication between machines within the cluster. Typically, this requires configuring iptables or OS-level firewall software on each machine to open the ports listed

- 9300 – 9400

The following shows an example of iptables commands to configure the operating system firewall:

```
-A INPUT -m state --state NEW -m tcp -p tcp --dport 9080 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 9081 -j ACCEPT
-A INPUT -m state --state NEW -m multiport -p tcp --dports 9300:9400 -j ACCEPT
```

If a port on the Events Service node is blocked, the Events Service installation command will fail for the node and the Enterprise Console command output and logs will include an error message similar to the following:

```
failed on host: <ip_address> with message: Uri [http://localhost:9080/_ping] is un-pingable.
```

If you see this error, make sure that the ports indicated in this section are available to other cluster nodes.

Configure Cluster Nodes that Run Linux

If deploying to Linux machines, on each node in the Events Service cluster, make these configuration changes:

1. Using a text editor, open `/etc/sysctl.conf` and add the following:

   ```
   vm.max_map_count=262144
   ```

2. Raise the open file descriptor limit in `/etc/security/limits.conf`, as follows:

   ```
   <username_running_eventsservice>     soft    nofile          96000
   <username_running_eventsservice>     hard    nofile          96000
   ```

   Replace `username_running_eventsservice` with the username under which the Events Service processes run. So if you are running Analytics as the user `appduser`, you would use that name as the first entry.

Configure SSH Passwordless Login

For Linux deployments, you will use the Enterprise Console to deploy and manage the Events Service cluster.

The Enterprise Console needs to be able to access each cluster machine using passwordless SSH for a non-embedded Events Service. Before starting, enable key-based SSH access as described here.

This setup involves generating a key pair on the Enterprise Console and adding the public key as an authorized key on the cluster nodes. The following steps take you through the configuration procedure for an example scenario. You will need to adjust the steps based on your environment.

If you are using EC2 instances on AWS, the following steps are taken care of for you when you provision the EC2 hosts. At that time, you are prompted for your PEM file, which causes the public key for the PEM file to be copied to the authorized_keys of the hosts. You can skip these steps in this case.

On the host machine, follow these steps:

1. Log in to the Enterprise Console host machine or switch to the user you will use to perform the deployment:

   ```
   su - $USER
   ```

2. Create a directory for SSH artifacts (if it doesn't already exist) and set permissions on the directory, as follows:

   ```
   mkdir -p ~/.ssh
   chmod 700 ~/.ssh
   ```

3. Change to the directory:

   ```
   cd .ssh
   ```

4. Generate PEM public and private keys in RSA format:

   ```
   ssh-keygen -t rsa -b 2048 -v -m pem
   ```

5. Enter a name for the file in which to save the key when prompted, such as `appd-analytics`.
6. Rename the key file by adding the `.pem` extension:

   ```
   mv appd-analytics appd-analytics.pem
   ```

   You will later configure the path to it as the `sshKeyFile` setting in the Enterprise Console configuration file, as described in Deploying an Events Service Cluster.
7. Transfer a copy of the public key to the cluster machines. For example, you can use scp to perform the transfer as follows:

   ```
   scp ~/.ssh/myserver.pub host1:/tmp
   scp ~/.ssh/myserver.pub host2:/tmp
   scp ~/.ssh/myserver.pub host3:/tmp
   ```

   Continuing with the example, `myserver` should be appd-analytics.
   The first time you connect you may need to confirm the connection to add the cluster machine to the list of known hosts and enter the user's password.
8. On each cluster node (host1, host2, and host3), create the .ssh directory in the user home directory, if not already there, and add the public key you just copied as an authorized key:

   ```
   cat /tmp/appd-analytics.pub >> .ssh/authorized_keys
   chmod 600 ~/.ssh/authorized_keys
   ```

9. Test the configuration from the host machine by trying to log in to a cluster node by `ssh`:

   ```
   ssh host1
   ```

   If unable to connect, make sure that the cluster machines have the `openssh-server` package installed and that you have modified the operating system firewall rules to accept SSH connections. If successful, you can use the Enterprise Console to deploy the platform.

If you encounter the following error, use the instructions in this section to double-check your passwordless SSH configuration:

```
Copying JRE to the remote host failed on host: 172.31.57.204 with message: Failed to upload file: java.net.
ConnectException: Connection timed out
```

Install the Events Service on Linux

Related Pages:

- Events Service Requirements

This page describes how to install and administer the Events Service on Linux systems through the CLI. Steps for scaling up an embedded Events Service using the Enterprise Console are also included.

The AppDynamics Enterprise Console automates the task of installing and administering an Events Service deployment through either the GUI or CLI. For information on installing Events Service using the Enterprise Console, see Custom Install.

You do not need to specify the installation or data directory for the Events Service installation. If you do, use a different one from the platform directory.

Events Service Host Requirements

Before starting, be sure to review the Release Notes for known issues and late-breaking information on using the Events Service and Enterprise Console. Also, observe the following requirements:

- The Events Service can be deployed as a single node or as a multi-node cluster of 3 or more nodes.
- The versions of Linux supported include the flavors and versions supported by the Controller, as indicated by Prepare Linux for the Controller.
- The Events Service must run on a dedicated machine. The machine should not run other applications or processes not related to Events Service.
- Use appropriately sized hardware for the Events Service machines. The Enterprise Console checks the target system for minimum hardware requirements. For more information on these requirements, see the description of the profile argument to the Events Service install command in Install the Events Service Cluster.
- The Controller and Events Service must reside on the same local network and communicate by the internal network. Do not deploy the cluster to nodes on different networks, whether relative to each other or to the Controller where the Enterprise Console runs. When identifying cluster hosts in the configuration, you will need to use the internal DNS name or IP address of the host, not the externally routable DNS name.
  For example, in terms of an AWS deployment, use the private IP address such as `172.31.2.19` rather than public DNS hostname such as `ec2-34-201-129-89.us-west-2.compute.amazonaws.com`.
- Make sure that the appropriate ports on each Events Service host are open. See Port Settings for more information.
- The Enterprise Console uses an SSH key to access the Events Services hosts. See the section below for information on generating the key.
- Events Service nodes normally operate behind a load balancer. When installing an Events Service node, the Enterprise Console automatically configures a direct connection from the Controller to the node. If you deploy a cluster, the first primary node is automatically configured as the connection point in the Controller. You will need to reconfigure the Controller to connect through the load balancer VIP after installation, as described below. For sample configurations, see Load Balance Events Service Traffic.

Port Settings

Each machine must have the following ports accessible to external (outside the cluster) traffic:

- Events Service API Store Port: 9080
- Events Service API Store Admin Port: 9081

For a cluster, ensure that the following ports are open for communication between machines within the cluster. Typically, this requires configuring `iptables` or OS-level firewall software on each machine to open the ports listed

- 9300 – 9400

The following shows an example of `iptables` commands to configure the operating system firewall:

```
-A INPUT -m state --state NEW -m tcp -p tcp --dport 9080 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 9081 -j ACCEPT
-A INPUT -m state --state NEW -m multiport -p tcp --dports 9300:9400 -j ACCEPT
```

If a port on the Events Service node is blocked, the Events Service installation command will fail for the node and the Enterprise Console command output and logs will include an error message similar to the following:

```
failed on host: <ip_address> with message: Uri [http://localhost:9080/_ping] is un-pingable.
```

If you see this error, make sure that the ports indicated in this section are available to other cluster nodes.

Create the SSH Key

When installing Events Service, you will need to provide the SSH key that the Enterprise Console can use to access Events Service hosts remotely. Before starting, create the PEM public and private keys in RSA format. The key file must not use password protection.

For example, using `ssh-keygen`, you can create the key using the following command:

```
ssh-keygen -t rsa -b 2048 -v -m pem
```

Configure SSH Passwordless Login

The Enterprise Console needs to be able to access each cluster machine using passwordless SSH. Before starting, enable key-based SSH access.

This setup involves generating a key pair on the Enterprise Console host and adding the Enterprise Console's public key as an authorized key on the cluster nodes. The following steps take you through the configuration procedure for an example scenario. You will need to adjust the steps based on your environment.

If you are using EC2 instances on AWS, the following steps are taken care of for you when you provision the EC2 hosts. At that time, you are prompted for your PEM file, which causes the public key for the PEM file to be copied to the `authorized_keys` of the hosts. You can skip these steps in this case.

On the Enterprise Console machine, follow these steps:

1. Log in to the Enterprise Console machine or switch to the user you will use to perform the deployment:

   ```
   su - $USER
   ```

2. Create a directory for SSH artifacts (if it doesn't already exist) and set permissions on the directory, as follows:

   ```
   mkdir -p ~/.ssh
   chmod 700 ~/.ssh
   ```

3. Change to the directory:

   ```
   cd .ssh
   ```

4. Generate PEM public and private keys in RSA format:

   ```
   ssh-keygen -t rsa -b 2048 -v -m pem
   ```

   The key file must not use password protection.
5. Enter a name for the file in which to save the key when prompted, such as `appd-analytics`.

6. Rename the key file by adding the `.pem` extension:

   ```
   mv appd-analytics appd-analytics.pem
   ```

   You will later configure the path to it as the `sshKeyFile` setting in the Enterprise Console configuration file, as described in Deploying an Events Service Cluster.
7. Transfer a copy of the public key to the cluster machines. For example, you can use `scp` to perform the transfer as follows:

   ```
   scp ~/.ssh/myserver.pub host1:/tmp
   scp ~/.ssh/myserver.pub host2:/tmp
   scp ~/.ssh/myserver.pub host3:/tmp
   ```

   Continuing with the example, `myserver` should be `appd-analytics`.
   The first time you connect you may need to confirm the connection to add the cluster machine to the list of known hosts and enter the user's password.
8. On each cluster node (host1, host2, and host3), create the `.ssh` directory in the user home directory, if not already there, and add the public key you just copied as an authorized key:

   ```
   cat /tmp/appd-analytics.pub >> .ssh/authorized_keys
   chmod 600 ~/.ssh/authorized_keys
   ```

9. Test the configuration from the Controller machine by trying to log in to a cluster node by `ssh`:

```
ssh host1
```

> If unable to connect, make sure that the cluster machines have the `openssh-server` package installed and that you have modified the operating system firewall rules to accept SSH connections. If successful, you can use the Enterprise Console on the Controller host to deploy the Events Service cluster, as described next.

If the Enterprise Console attempts to install the Events Service on a node for which passwordless SSH is not properly configured, you will see the following error message:

```
./bin/platform-admin.sh add-hosts --hosts <es_host_1> <es_host_2> <es_host_3> --credential <credential name> --
platform-name <name of platform>
...
Failed to connect to the remote host. Please verify that the host name and credentials you provided are correct.
For more information, consult the documentation: https://docs.appdynamics.com/display/PRO45
/Administer+the+Enterprise+Console#AdministertheEnterpriseConsole-manage-hostsManageHosts
```

If you encounter this error, use the instructions in this section to double-check your passwordless SSH configuration. Also, you need to and `add-hosts` firs t then install `event-service`. If SSH configuration is not setup correctly, `add-hosts` commands will fail.

> The `add-hosts` command is to add hostnames into platforms. During the `events-service` installation, the hosts come from the platform hosts, and then they are used on the `events-service`.

Tune the Operating System for Production Cluster Nodes

Before installing the Events Service cluster, you need to perform a few manual changes as described below. These are particularly relevant for production Events Service deployments. On each node in the cluster, make these configuration changes:

1. Using a text editor, open `/etc/sysctl.conf` and add the following:

```
vm.max_map_count=262144
```

2. Raise the open file descriptor limit in `/etc/security/limits.conf`, as follows:

```
<username_running_eventsservice>     soft    nofile          96000
<username_running_eventsservice>     hard    nofile          96000
<username_running_eventsservice>     soft    memlock         unlimited
<username_running_eventsservice>     hard    memlock         unlimited
```

3. Disable swap memory by running the following command. Remove swap mount points by removing or commenting the lines in `/etc/fstab` that contain the word swap.

```
swapoff -a
```

Installing the Events Service Using the GUI

In the GUI, the Express Install option automatically installs an Events Service on the same host as the Controller. The Custom Install option can install an embedded or scaled-up Events Service. If you install an embedded Events Service and want to switch to a scaled-up Events Service, complete the steps described in Scaling Up an Embedded Events Service.

Installing the Events Service Using the CLI

1. Set up load balancing. See Load Balance Events Service Traffic for information about configuring the load balancer.
2. At the command line, navigate to the `platform-admin` directory created at Enterprise Console installation. See Install the Enterprise Console.
3. If it has been more than one day since your last session, you will have to log in with the following command:

```
bin/platform-admin.sh login --user-name <admin_username> --password <admin_password>
```

4. Create a platform as follows:

```
bin/platform-admin.sh create-platform --name <platform_name> --installation-dir
<platform_installation_directory>
```

The installation directory is the directory where the Enterprise Console installs all platform components.

> The same installation directory should exist and is used on all remote nodes. This is done to maintain the homogeneity of the configuration across different nodes.

5. Add the SSH key that the Enterprise Console will use to access and manage the Events Service hosts remotely. (See Create the SSH Key for more information):

```
bin/platform-admin.sh add-credential --credential-name <name> --type ssh --user-name <username> --ssh-
key-file <file path to the key file> --platform-name <name of platform>
```

`<file path to the key file>` is the private key for the Enterprise Console machine. The installation process uses the keys to connect to the Events Service hosts. The keys are not deployed, but instead, encrypted and stored in the Enterprise Console database.

The `platform-name` parameter is optional.

6. Add hosts to the platform, passing the credential you added to the platform:

```
bin/platform-admin.sh add-hosts --hosts es_host_1 es_host_2 es_host_3 --credential <credential name> --
platform-name <name of platform>
```

The `platform-name` parameter is optional.

7. On each Events Service destination node in the cluster, create an installation directory for the Events Service. This is the directory you specified as the `installation-dir` argument while creating the platform in step (2).
8. Again at the command line for the Enterprise Console machine, run the following command from the `platform-admin` directory. Pass the cluster configuration settings as arguments to the command. The format for the command is the following:

```
bin/platform-admin.sh submit-job --platform-name <platform-name> --service events-service --job install
--target-version <latest> --args profile=<dev> serviceActionHost=<es_host_1>
serviceActionHost=<es_host_2> serviceActionHost=<es_host_3>
```

The `platform-name` and `jvmTempDir` parameters are optional.

Arguments are:

- `jvmTempDir`: Use this argument to override default JVM temporary `/tmp` directory in Linux installations.
- `hosts`: Use this argument or `host-file` to specify the internal DNS hostnames or IP addresses of the cluster hosts in your deployment. With this argument, pass the hostnames or addresses as parameters. For example:

  ```
  --hosts 192.168.32.105 192.168.32.106 192.168.32.107
  ```

- `host-file`: As an alternative to specifying hosts as `--hosts` arguments, pass them as a list in a text file you specify with this argument. Specify the internal DNS hostname or IP address for each cluster host as a separate line in the plain text file:

  ```
  192.168.32.105
  192.168.32.106
  192.168.32.107
  ```

- `profile`: By default (with profile not specified), the installation is considered a production installation. Specifying a developer profile (`profile dev`) directs the Enterprise Console to use a reduced hardware profile requirement, suitable for non-production environments only. The Enterprise Console checks for the following resources:
  - For a dev profile, 1 core CPU, 1 GB RAM, and 2 GB disk space.
  - Otherwise, 4 core CPU, 12 GB RAM, and 128 GB of disk space.

For example:

```
bin/platform-admin.sh add-hosts --hosts ip-172-31-20-21.us-west-2.compute.internal ip-172-31-20-22.us-
west-2.compute.internal ip-172-31-20-23.us-west-2.compute.internal
```

If using a hosts text file, use the following command:

```
bin/platform-admin.sh add-hosts --hosts --host-file=/home/appduser/hosts.txt
```

9. Log in to each Events Service node machine, and run the script for setting up the environment as follows:

   > The `tune-system.sh` script is used for optimizing your environment. It is optional.

   a. Add execute permission to the `tune-system.sh` script:

   ```
   chmod +x tune-system.sh
   ./tune-system.sh
   ```

   b. Run the script:

   ```
   sudo <installation_dir>/events-service/processor/bin/tool/tune-system.sh
   ```

10. If you are using a load balancer, use the virtual IP for the Events Service as presented at the load balancer. Configure the Controller connection to the Events Service as follows:

    a. Open the Administration Console.
    b. In the Controller settings pane, find `appdynamics.on.premise.event.service.url` and change its value to the URL of the virtual IP for the Events Service at the load balancer.

    It may take a few minutes for the Controller and Events Service to synchronize account information after you modify connection settings in the console.

When finished, use the Enterprise Console for any Events Service administrative functions. You should not need to access the cluster node machines directly once they are deployed. In particular, do not attempt to use scripts included in the Events Service node home directories.
The Enterprise Console automatically updates the Controller configurations after installation.

Scaling Up an Embedded Events Service

The following steps describe how to scale up an Events Service that is on a shared host with the Controller. This allows the embedded Events Service to run on a separate host. You can also install the Events Service on a separate host directly by using the Custom Install.

1. Set up load balancing. See Load Balance Events Service Traffic for information about configuring the load balancer.
2. Open the Enterprise Console GUI.
3. Verify that the credentials and hosts you want to use are added to the AppDynamics platform. For more information, see Administer the Enterprise Console.

   a. On the **Credential** page, add the SSH credentials for the hosts on which you want to install the Events Service.
   b. On the **Hosts** page, add the hosts. The Enterprise Console uses these hosts for the scaled-up Events Service, which requires at least one host or three or more hosts.

4. On the **Events Service** page, navigate to More link and select the Events Service and click **Scale Up Events Service** under More and complete the wizard. When you enter hosts to use for a scaled-up Events Service, do not include the Controller host.

   > You do not need to restart the Controller since that is automatically done for you by the scale-up job.

5. Log in to each node machine, and run the script for setting up the environment as follows:

   ```
   sudo <installation_dir>/events-service/latest/bin/tool/tune-system.sh
   ```

6. Navigate to the **Controller** page.
7. By default, the Enterprise Console configures the Events Service connection in the Controller to refer to the first primary node defined in the cluster. If you are using a load balancer, as recommended, you need to change this Controller setting to point to the Events Service VIP as presented at the load balancer instead, as follows:

   a. Open the Administration Console.
   b. In the **Controller settings** pane, find `appdynamics.on.premise.event.service.url.`
   c. Change the value of the setting for the URL to the VIP for the Events Service at the load balancer.

8. By default, Database Monitoring stores events data in the Events Service embedded in the Controller. To have it use the Events Service you just deployed, ensure that the `appdynamics.on.premise.event.service.key` value matches with `ad.accountmanager.key.controller` value inside the events-service-api-store.properties file.

> Note that only newly generated Database Monitoring data will be stored in the Events Service; previously collected data will remain in the embedded Events Service instance unless it is migrated to the new Events Service. See Connect to the Events Service.

9. It may take a few minutes for the Controller and Events Service to synchronize account information after you modify connection settings in the Enterprise Console.

Troubleshooting Installation

If the Enterprise Console crashes or shuts down while installing the Events Service, the GUI may display that the installation is in progress. To resolve this issue, uninstall the Events Service with the CLI. Then, install the Events Service with the CLI.

Install the Events Service on Windows

Related pages:

- Events Service Requirements

You can install and administer the Events Service on Microsoft Windows systems as a single node or a cluster. Common use cases include:

- **Single-node Events Service** — Good for demonstration purposes and other scenarios where data redundancy and high availability are not required.
- **Three-node cluster** — The minimum size for a production Events Service cluster.
- **Cluster of four nodes or more** — For deployments where increased load or expected sizing exceeds the capacity of a three-node cluster.

Contact AppDynamics customer support if you anticipate deploying a cluster of 10 or more nodes.

Creating more than one instance of the same service type is not supported on Windows.

You do not need to specify the installation or data directory for the Events Service installation. If you do, use a different one from the platform directory.

Decide Which Node(s) to Make Master Nodes

Every Events Service node is either a master node or a data node. In an Events Service cluster, the master node both acts as a storage node, and manages the state of the data across the cluster, including the state of the replica. In a single-node deployment, there's not much for the master to do.

The master node is the first node to start up. If the master node becomes unavailable, the worker nodes attempt to elect a new master.

As you install Events Service, you specify the configuration for each node, which consists of two pieces of information:

- Whether to enable the node to serve as a master, and
- How many nodes (minimum) must be available in the cluster for a new master to be elected

This table describes what values to specify as you install:

Order of Node in the Deployment	Master-enabled?	Minimum available nodes required to elect a new master
First (only) node in a single-node installation	true	1
First, second, and third nodes of a three-node cluster	true	2
Fourth (or higher) node of a cluster	false	2

Specifying the installation or data directory is optional for the Events Service. If you do this, the directory you specify must not be the platform directory.

Installation Quick Start

Before you can install the Events Service on Windows, you must use Enterprise Console to install the Controller. The EUM Server must be installed separately as it cannot be installed using the Enterprise Console.

1. Install the Enterprise Console.
2. Use the Enterprise Console to create the platform and add hosts.
3. To install the Controller and Events Service on the same host, use the Express Install option.
   Use the Custom Install to install a scaled-out Events Service that runs on a host that is separate from the Controller. Custom installations provide more flexibility on where and how to install Controller and Events Service.
4. (Optional) Install the EUM Server.
5. Complete the post-install tasks for the Controller, Events Service, and EUM Server.

Deploy a Single-Node Events Service

Ensure that you have met all of the current Events Service Requirements, including:

- Have Java Runtime Environment (JRE) version = 1.8
- Defined Java in the Windows environment variable path, or a JRE folder that is accessible by `events-service.exe` in the relative path `..\..\jre`

To manually install the Events Service on a single node:

1. Unzip the Events Service distribution archive to a directory on the target host. This creates the `events-service` directory with the Events Service artifacts.
2. Begin configuring the connection to the Events Service in the Controller.
    a. With the Controller running, open the Administration Console as the root user.
    b. In the **Controller Settings** page, search for `appdynamics.on.premise.event.service.url`.
    c. Replace the default value to the internal hostname for the Events Service machine, and the default value for the Events Service listen port, 9080.
    For example: http://*hostname:9080*. Select **Save**.

    If you are putting a load balancer in front of the Events Service (required for a cluster), this will be the VIP for the Events Service as exposed at the load balancer. In this case, it is likely you will need to return to this step after you finish deploying the node and configuring the load balancer.

3. Obtain the Controller key:
    a. While on the **Controller Settings** page, search for the `appdynamics.on.premise.event.service.key` setting.

    If you do not install the default Events Service, then the `appdynamics.on.premise.event.service.key` setting is blank. To create the `appdynamics.on.premise.event.service.key`, use a UUID generator.

    b. Copy the setting's value. You will need this to complete Step 4.
    c. Close the Administration Console.
4. Configure the connection from the Events Service to the Controller:
    a. From your Windows Explorer, open the `events-service\conf\events-service-api-store.properties` file to edit.
    b. Locate the `controller-key` property and add it:

    ```
    ad.accountmanager.key.controller=controller-key
    ```

5. Verify the heap settings for the Events Service processes:
    a. Verify that the minimum and maximum heap settings for the two Events Service processes (the Events Service JVM, and the Elasticsearch processes, respectively) are correct and sufficient for your deployment.
    The settings:
        - Are located in the `events-service-api-store.properties` file.
        - Use `g` for gigabyte (GB) and `m` for megabyte (MB).
    b. For the Events Service process, verify:

    ```
    ad.jvm.options.name=events-service.vmoptions
    ad.jvm.heap.min=1g
    ad.jvm.heap.max=1g
    ```

    c. For the ElasticSearch process, verify:

    ```
    ad.es.jvm.options.name=events-service.vmoptions
    ad.es.jvm.heap.min=8g
    ad.es.jvm.heap.max=8g
    ```

    A production Elasticsearch installation requires 8 GB. For a demonstration installation, you can retain the default of 1 GB.

6. Save and close the `events-service-api-store.properties` file.
7. Install the Events Service as a Windows service, and open the command prompt as an Administrator:

> Do not use PowerShell to do this.

Change the directory to `events-service`, and then enter:

```
bin\events-service.exe service-install -p conf\events-service-api-store.properties --auto-start
```

- This command also installs Elasticsearch (even though it contains no explicit reference to Elasticsearch).
- The optional `auto-start` flag causes the Events Service to be installed as an automatically started service; without this flag, the Events Service is installed as a manually started service.
- For verbose installation and operation logging (useful for troubleshooting), include the `log-verbose` flag.

8. Locate the service name for the Events Service:

```
bin\events-service.exe service-list
```

9. Open the Windows services and select the **AppDynamics Events Service AppStore *xxxxx***. Select **Start**.
10. Check the health of the new node and verify service status:
    - If "Healthy" appears as the service status, then it indicates that the process is operating normally:

      ```
      bin\events-service.exe check-health -hp localhost:9081
      ```

      For the port, pass the administration port for the Events Service, 9081 by default.

      ```
      [appduser@controller-one events-service]$ bin/events-service.exe check-health -hp 192.168.33.22:
      9081
      [2015-12-09T18:30:45,342-08:00] HV000001: Hibernate Validator 5.0.2.Final
      [2015-12-09T18:30:45,956-08:00] Individual statuses below:
      [2015-12-09T18:30:45,956-08:00] [192.168.33.22:9081] status is [200 OK]
      [2015-12-09T18:30:45,956-08:00] Overall status Healthy
      ...
      ```

    - If the service status does not display as `Overall status Healthy`, then the service is unhealthy. To correct it, you need to determine the correct key mappings between the following Events Service configuration and the Controller settings:
      - `appdynamics.es.eum.key` should map to: `ad.accountmanager.key.eum`
      - `appdynamics.saas.event.service.key should map to: ad.accountmanager.key.controller`
      - `appdynamics.saas.event.service.key should map to:  ad.accountmanager.key.mds`

      i. If the values of the key mappings are blank in the Admin Console, then use a UUID generator to create them.
      ii. Use a UUID generator to create a value for `ad.accountmanager.key.ops`.
      iii. Use a UUID generator to create a different value for the following keys (you can use the same UUID for all three keys)
        - `ad.accountmanager.key.slm`
        - `ad.accountmanager.key.jf`
        - `ad.accountmanager.key.service`
11. Configure the connections from the Analytics Agent, EUM Server, or Database Monitoring agents to the Events Service, as described in Connect to the Events Service.

Deploy an Events Service Cluster (Three Nodes)

The following steps describe how to deploy an Events Service cluster made up of three nodes, the minimum size of the Events Service cluster. These steps apply whether you are performing a new installation of a cluster or expanding a single node deployment into a three-node cluster. For information on expanding beyond a three-node cluster, see Adding Nodes to a Cluster.

> Note that all services on Windows machines must be installed on the Enterprise Console host since the Enterprise Console does not support remote operations on Windows. Therefore, you cannot use the Enterprise Console GUI to deploy an Events Service cluster.

Before starting, review the topology notes in Events Service Deployment and make sure that all machines in the cluster meet the system requirements.

When ready, perform these steps on each of the three nodes.

1. Follow the steps for configuring a single node cluster in the 1-node installation above. *Additionally*, configure the following settings in the `conf\events-service-api-store.properties` file:

   a. Change the value of the `ad.es.node.minimum_master_nodes` property to 2:

```
ad.es.node.minimum_master_nodes=2
```

The setting specifies the minimum number of master-eligible instances that must be available in order to elect a new master. Since an Events Service cluster has three master nodes, this value should be two for a cluster.

b. Set the value of `ad.es.event.index.shards` to the number of nodes, in this case, three:

```
ad.es.event.index.shards=3
```

You do not need to change this value if it is already higher than the number of nodes.

c. Set the replication factor to 1 by changing the `ad.es.event.index.replicas` and `ad.es.metadata.replicas` properties, as follows:

```
ad.es.event.index.replicas=1
ad.es.event.index.hotLifespanDays=10
ad.es.metadata.replicas=1
```

d. For the unicast `hosts` property, add the hostname or IP address, along with the port 9300, for each node in the cluster:

```
ad.es.node.unicast.hosts=node1.example.com:9300,node2.example.com:9300,node3.example.com:9300
```

e. Change the publish host to the IP address or hostname of this machine. For example:

```
ad.es.node.network.publish.host=node2.example.com
```

f. Configure heap space for the Events Service and ElasticSearch processes, as follows:

i. To set the Events Service process heap size to 1 GB, for example, use the following properties:

```
ad.jvm.options.name=events-service.vmoptions
ad.jvm.heap.min=1g
ad.jvm.heap.max=1g
```

For the setting value, g indicates gigabyte (GB), and m indicates megabyte (MB).

ii. For the ElasticSearch process, the heap size should be set to half the size of the available RAM on the system, up to a maximum of 31 GB. To set the ElasticSearch process heap size to 8 GB, for example, set the properties as follows:

```
ad.es.jvm.options.name=events-service.vmoptions
ad.es.jvm.heap.min=8g
ad.es.jvm.heap.max=8g
```

For the setting value, g indicates gigabyte (GB), and m indicates megabyte (MB).

g. Save and close the file.

2. Install the Events Service as a Windows service:

```
bin\events-service.exe service-install -p conf\events-service-api-store.properties  --auto-start
```

The optional auto-start flag causes the Events Service to be installed as an automatically started service. If you do not include the flag, the Events Service is installed as a manually started service. An additional option, `log-verbose`, increases the verbosity of installation and operation logging, which is useful for troubleshooting.

3. Enter the following command to find the service name for the Events Service:

```
bin\events-service.exe service-list
```

4. Pass the service name returned by the service-list command as the -s parameter argument in the following command:

```
bin\events-service.exe service-start -s "<Name from service-list>"
```

Be sure to enclose the name in double-quotes.

5. Check the health of the new node using the following command. At least two nodes must be running before you run the command.

   ```
   bin\events-service.exe check-health -hp localhost:9081
   ```

   For the port, pass the administration port for the Events Service, 9081 by default. Verify that "Healthy" appears as the service status, indicating that the process is operating normally:

   ```
   [appduser@controller-one events-service]$ bin/events-service.exe check-health -hp 192.168.33.22:9081
   [2015-12-09T18:30:45,342-08:00] HV000001: Hibernate Validator 5.0.2.Final
   [2015-12-09T18:30:45,956-08:00] Individual statuses below:
   [2015-12-09T18:30:45,956-08:00] [192.168.33.22:9081] status is [200 OK]
   [2015-12-09T18:30:45,956-08:00] Overall status Healthy
   ...
   ```

6. Configure a load balancer to distribute traffic to the Events Service cluster, as described in Load Balance Events Service Traffic.
7. Connect the Controller and other clients—Analytics Agent, EUM Server, or Database Monitoring agents—to the Events Service, as described in Connect to the Events Service.

Expand an Events Service Cluster

The Events Service cluster is horizontally scalable. You can add nodes to an existing cluster without affecting or having to restart the existing nodes.

> ⚠ Note that all services on Windows machines must be installed on the Enterprise Console host since the Enterprise Console does not support remote operations on Windows. Therefore, you cannot use the Enterprise Console GUI to expand an Events Service cluster.

Before starting, prepare the new cluster machine. Verify system requirements and prepare the environment as described above.

For each node beyond the original three master nodes, download and configure the nodes as previously described. The configuration steps for any nodes added to the cluster after the initial three master nodes are as follows:

1. For each cluster nodes beyond the initial three master nodes, open the `conf\events-service-api-store.properties` for editing and make these configuration changes:
   a. Set the `ad.es.node.master` value to false:

      ```
      ad.es.node.master=false
      ```

   b. Set the `ad.es.node.minimum_master_nodes` value to 2.

      ```
      ad.es.node.minimum_master_nodes=2
      ```

   c. Set the value of `ad.es.event.index.shards` to the number of nodes in the cluster. You do not need to change this value if it is already higher than the number of nodes.

      ```
      ad.es.event.index.shards=<number_of_nodes>
      ```

      You do not need to change this value if it is already higher than the number of nodes.
   d. For the unicast hosts property, add the hostnames or IP addresses of all nodes in the cluster, including the node you are adding. For each node specify the ports on which the nodes communicate, 9300-9400. For example:

      ```
      ad.es.node.unicast.hosts=node1.example.com[9300-9400],node2.example.com[9300-9400],node3.example.com[9300-9400],node4.example.com[9300-9400]
      ```

      You do not need to reconfigure the unicast hosts settings for existing cluster members, as the new node can join the cluster dynamically.
   e. Change the publish host to the IP address or hostname of this machine. For example:

      ```
      ad.es.node.network.publish.host=node4.example.com
      ```

   f. Configure heap space for the Events Service and ElasticSearch processes, as follows:
      i. To set the Events Service process heap size to 1 GB, for example, use the following properties:

```
ad.jvm.options.name=events-service.vmoptions
ad.jvm.heap.min=1g
ad.jvm.heap.max=1g
```

For the setting value, g indicates gigabyte (GB), and m indicates megabyte (MB).

ii. For the ElasticSearch process, the heap size should be set to half the size of the available RAM on the system, up to a maximum of 31 GB. To set the ElasticSearch process heap size to 8 GB, set the properties as follows:

```
ad.es.jvm.options.name=events-service.vmoptions
ad.es.jvm.heap.min=8g
ad.es.jvm.heap.max=8g
```

For the setting value, g indicates gigabyte (GB), and `m` indicates megabyte (MB).

g. Save and close the file.

2. Install the Events Service as a Windows service:

```
bin\events-service.exe service-install -p conf\events-service-api-store.properties --auto-start
```

The optional auto-start flag causes the Events Service to be installed as an automatically started service. If you do not include the flag, the Events Service is installed as a manually started service. An additional option, `log-verbose`, increases the verbosity of installation and operation logging, which is useful for troubleshooting.

3. Enter the following command to find the service name for the Events Service:

```
bin\events-service.exe service-list
```

4. Pass the service name returned by the service-list command as the -s parameter argument in the following command:

```
bin\events-service.exe service-start -s "<Name from service-list>"
```

5. Check the health of the new node:

```
bin\events-service.exe check-health -hp localhost:9081
```

Note: At least two nodes must be running before you run the command.

For the port, pass the administration port for the Events Service, 9081 by default. Verify that "Healthy" appears as the service status, indicating that the process is operating normally:

```
[appduser@controller-one events-service]$ bin/events-service.bin check-health -hp 192.168.33.22:9081
[2015-12-09T18:30:45,342-08:00] HV000001: Hibernate Validator 5.0.2.Final
[2015-12-09T18:30:45,956-08:00] Individual statuses below:
[2015-12-09T18:30:45,956-08:00] [192.168.33.22:9081] status is [200 OK]
[2015-12-09T18:30:45,956-08:00] Overall status Healthy
...
```

6. Modify your load balancer rules to include the new cluster node. For more information, see Load Balance Events Service Traffic.

Start and Stop the Events Service

At installation, the Events Service is installed as a service and is left running upon completion of the installation. You can stop and stop it as a service or as a foreground process, as described here or by using the GUI.

Start and Stop as a Foreground Process

To start the Events Service as a foreground process, however, use the following command:

```
bin\events-service.exe start -p conf\events-service-api-store.properties
```

To stop the Events Service as a foreground process, use this command:

```
bin\events-service.exe stop
```

Stop and Start as a Windows Service

You can stop and start the Events Service as a Windows service from the Services Manager. You can also stop and start it using the `events-service.exe` tool, as here:

1. Enter the following command to find the service name for the Events Service:

   ```
   bin\events-service.exe service-list
   ```

2. Pass the service name returned by the `service-list` command as the `-s` parameter argument to the following command. Enclose the service name in double-quotes.

   ```
   bin\events-service.exe service-start -s "<Name from service-list>"
   ```

To stop the service, run this command:

```
bin\events-service.exe service-stop -s "<Name from service-list>"
```

Remove a Node

To remove a node that is not enabled for operation as a master node from the cluster, simply stop the Events Services on the node or remove the machine it runs on from the network.

Note the following guidelines:

- You cannot remove nodes such that the resulting cluster size is two
- A cluster that consists of three or more nodes can't be reduced in size to a single node Events Service.

After you remove a node, be sure to adjust your load balancer rules to remove the old cluster member. See Load Balance Events Service Traffic for more information.

If you are not using a load balancer with a cluster deployment, keep in mind that the connection settings for the first master node that reports to the Controller at installation time are written to the Controller setting that identifies the Events Service to the Controller. If you remove a master node in that case, check whether the removed master node is node identified as the Events Service destination URL in the Controller connection settings; adjust the setting if so. See Connect to the Events Service for more information.

To reconfigure an existing node to enable operation as a master node, or add a new node with the master option enabled:

```
ad.es.node.master=true
```

If reconfiguring a node, restart the node after changing the configuration.

Administer the Events Service

This component is available for on-premises deployments only. SaaS deployments are managed by AppDynamics.

This page describes how to manage Events Service with the Enterprise Console. All of these tasks can be performed on the GUI or CLI.

Start and Stop the Events Service

The Events Service is an internal data storage engine used by the Database Visibility module. To use Database Monitoring, you need to start the Events Service.

The Events Service is automatically started after you install it.

On Linux

Start the Events Service on Linux by running this command:

```
bin/platform-admin.sh submit-job --platform-name <platform_name> --service events-service --job start
```

Stop the Events Service by running this command:

```
bin/platform-admin.sh submit-job --platform-name <platform_name> --service events-service --job stop
```

On Windows

Start the Events Service on Windows by running this command:

```
bin/platform-admin.exe cli submit-job --platform-name <platform_name> --service events-service --job start
```

Stop the Events Service by running this command:

```
bin/platform-admin.exe cli submit-job --platform-name <platform_name> --service events-service --job stop
```

Monitor Cluster Node Health

It is important to carefully monitor the health of the Events Service cluster, for a new deployment, especially to monitor disk consumption.

You can check the status of the cluster from the Controller page in the Enterprise Console GUI or the Controller machine using this command:

```
bin/platform-admin.sh show-events-service-health
```

The output shows possible issues and the steps you need to take to resolve them. For example, if the available disk is low, the resolution is to add nodes to the cluster.

The following are the potential errors and remediation steps:

Error	Explanation	Remediation
Cluster out of capacity	If the heap size of any Events Service Java process exceeds 80% utilization	Add Events Service nodes
Disk size remaining drops below 30%	The disk size of the identified node dropped below 30%	Add Events Service nodes
Events Service is not reachable but the host is reachable	The Events Service process on the identified node is not functioning properly	Restart the node
Machine is not reachable	The machine may be down, disconnected, or suffering failure	Try restarting the machine; if it continues, the node may need to be removed from the cluster
Cluster needs restart	A condition has been identified that requires a cluster restart	Restart the cluster

Cluster size is 2	Events Service cluster requires more than two nodes	Add a node

Expand the Cluster

You can use the Enterprise Console GUI or CLI to horizontally scale the Events Service cluster on Linux. To grow your existing deployment to contend with an increased workload, simply add nodes.

Before starting, prepare the new cluster machine. Verify system requirements and prepare the environment as described in Events Service Requirements.

It is important for any new machine in the cluster to have the same SSH-enabled user account as existing cluster members.

Once you have prepared the system, run the command for adding nodes:

```
bin/platform-admin.sh add-events-service-nodes --hosts host1  host2  host3
```

Alternatively, pass the hostnames of the new node as a file.

```
bin/platform-admin.sh add-events-service-nodes --host-file=/home/appduser/hosts.txt
```

The file you pass to the command (`hosts.txt` in the example) should contain the internal DNS hostnames or IP addresses of the nodes to add. It does not need to list existing nodes in the cluster. These hosts should be part of the platform. For more information about how to add a host to the platform, see Administer the Enterprise Console.

Be sure to modify your load balancer rules to include the new cluster member in its routing rules. See Load Balance Events Service Traffic for more information.

Restart the Node

You can use the Enterprise Console GUI or CLI to restart your node.

Once you have prepared the system, run the command for restarting your node:

```
bin/platform-admin.sh submit-job --platform-name <platform_name> --service events-service --job restart-node --
args nodeActionHost=<node_name>
```

where <node_name> is the hostname of the node from the command:

```
bin/platform-admin.sh list-nodes --platform-name <platform_name> --service events-service
```

Restart the Cluster

You can use the Enterprise Console GUI or CLI to restart your cluster.

Once you have prepared the system, run the command for restarting your cluster nodes:

```
bin/platform-admin.sh submit-job --platform-name <platform_name> --service events-service -job restart-cluster
```

Remove or Replace a Node

The `uninstall-events-service` command removes the Events Service software and data from all cluster nodes. The Controller and Events Service share a database, so if you are uninstalling the Controller instance under which you ran the Enterprise Console to install the Events Service, you need to uninstall the Events Service with this command *before* you uninstall the Controller.

The `remove-events-service-node` command removes the Events Service software and data from a single node that you specify by hostname. You should only use this command if you have at least four nodes in your cluster. Removing an Events Service node from a three-node cluster is not supported. Identify the node to remove using the `--node` command line parameter.

After you remove a node, be sure to adjust your load balancer rules to remove the old cluster member. See Load Balance Events Service Traffic for more information.

If you are not using a load balancer with a cluster deployment, keep in mind that the connection settings for the first primary node that reports to the Controller at installation time are written to the Controller setting that identifies the Events Service to the Controller. If you remove a primary node in that case, check whether the removed primary node is node identified as the Events Service destination URL in the Controller connection settings (e.g., `appdynamics.on.premise.event.service.url`) and adjust the setting if so. See Connect to the Events Service for more information.

Note the following guidelines:

- You cannot remove nodes such that the resulting cluster size is two
- A cluster that consists of three or more nodes can't be reduced in size to a single node Events Service.

The `remove-events-service-node` command removes the Events Service software and data from a single node that you specify by hostname. You should only use this command if you have at least four nodes in your cluster. Removing an Events Service node from a three-node cluster is not supported. Identify the node to remove using the `--node` command line parameter.

This command removes the node specified in the argument:

```
bin/platform-admin.sh remove-events-service-node  --node 10.0.100.51
```

If you attempt to remove a primary node using the command shown above, the Enterprise Console notifies you that you are attempting to remove a primary node and cancels the operation. As indicated in the output, you can proceed to remove the primary node by rerunning the command with the `-f` for ce flag. When you remove a primary node, the cluster elects a new primary node from the existing data nodes. The election process may take a few seconds, during which new events cannot be processed. Be sure to perform this operation at a time when the impact of a brief interruption of service will be minimal.

If you have an unreachable node you would like to remove, but cannot due to the above restrictions, you can choose to replace it instead.

This command replaces the old node specified in the argument with the new node:

```
bin/platform-admin.sh submit-job --service events-service --job replace-node --args
originalNode=<old_host_address> newNode=<new_host_address>
```


- After removing the Events Service nodes from the cluster, you may observe that the value `appdynamics.es.eum.key` changed in the Controller `admin.jsp` and in the Events Service properties file, but not in the EUM properties file, `analytics.accountAccessKey`.
- Check if the key value changed in the Controller and the Events Service, then replace the key value with the EUM properties file: `analytics.accountAccessKey`.

Enable the Events Service to Use SSL

You can use the Enterprise Console CLI to enable the Events Service to use SSL. You will need a KeyStore file in JKS format, the password and the alias for the KeyStore. See Create a Certificate and Generate a CSR for detailed instructions for creating the Keystore.

1. Log in to the Enterprise Console:

   ```
   bin/platform-admin.sh login --user-name <admin_username> --password <admin_password>
   ```

2. After successfully logging in, submit an `enable-ssl` job for the Events Service, providing the path to the KeyStore file, the KeyStore password, and KeyStore alias.

   ```
   bin/platform-admin.sh submit-job --platform-name <platform_name> --service events-service --job enable-
   ssl --args keystorePath=<path-to-keystore-jks-file> keystorePassword=<keystore_password>
   keystoreAlias=<keystore_alias>
   ```

3. Confirm that SSL has been enabled:

   ```
   curl -k -v -X GET https://<events-service-domain>:9080/_ping
   ```

4. The output of the cURL command should show the TLS handshakes and the HTTP status 200:

```
...
* TLSv1.2 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS handshake, Server hello (2):
...
< HTTP/1.1 200 OK
< Date: Fri, 10 May 2019 00:13:49 GMT
< X-Content-Security-Policy: default-src 'self'
...
```

Collect Events Service Logs

The Enterprise Console can collect logs from the nodes in the cluster. The following command retrieves node logs and bundles them, along with the Enterprise Console's own logs:

```
bin/platform-admin.sh retrieve-events-service-logs
```

When the command is finished, a ZIP file named `events-service.log.zip` is created in the location from which you ran the script. You can then extract the archive to troubleshoot or submit the archive for troubleshooting assistance to your AppDynamics representative. If the Enterprise Console failed to connect to one of the cluster nodes to retrieve logs for any reason, the connection error is written to a log file included in the archive.

Changing the SSH Key File

After initial installation, you may need up update the PEM file that gives the Enterprise Console access to node machines.

You can do so by creating the PEM file, as described in the discussion of configuring SSH passwordless login on Prepare the Events Service Host, and using the following command to install the new PEM file.

```
bin/platform-admin.sh set-user-credentials --ssh-key-file newkeyfile.pem
```

The change takes effect immediately.

Upgrade the Events Service

You can use the Enterprise Console to perform a rolling upgrade of the Events Service software on deployed nodes. For more information, see Upgrade the Events Service Using the Enterprise Console.

The general steps for upgrading an Events Service deployment are as follows:

1. Upgrade the Controller. (See Upgrade the Controller Using the Enterprise Console.)
2. Apply the upgrade to the Events Service nodes using the following command:

```
bin/platform-admin.sh upgrade-events-service
```

The Enterprise Console checks whether the Events Service is up to date relative to the current Controller version and, if not, performs the update.

Monitor Events Service Nodes with AppDynamics

You can use AppDynamics agents to monitor an Events Service node or cluster and generate diagnostic information for troubleshooting. The following steps outline the workflow.

1. Download the following agents from the AppDynamics Download Center:
    - Java Agent
    - Standalone Machine Agent
2. Install both agents on each node in the cluster: first the Java Agent, then the Standalone Machine Agent.
3. On each node in the cluster, update the VM options for the Java Agent:
    a. Open the following file in a text editor:
    `<controller_home>/platform_admin/events-service/conf/events-service.vmoptions`
    b. Add the following lines to the end of the file:

```
-javaagent:/opt/appdynamics/events-service/java_agent/ver4.5.0.0/javaagent.jar
-Dappdynamics.agent.accountName=<account_name>
-Dappdynamics.agent.applicationName=<events_service_app_name>
-Dappdynamics.controller.hostName=<controller_host>
-Dappdynamics.controller.port=443
-Dappdynamics.controller.ssl.enabled=true
-Dappdynamics.agent.nodeName=<events_service_node_name>
-Dappdynamics.agent.tierName=<events_service_tier_name>
```

Adjust the path to the Java Agent JAR, account name, and other values as appropriate.

> The business application name (`events_service_app_name`) and tier name (`events_service_tier_name`) should normally be the same for all nodes in an Events Service cluster, while each node must have a unique name (`events_service_node_name`).

For more information on modeling applications in AppDynamics, see Application Modeling.

4. On each node in the cluster, define the Node Name and Tier Name used by the Standalone Machine Agent, as described in Independent Machine Agent Installation.

   > The Node Name and Tier Name for each Standalone Machine Agent should be the same as the `events_service_node_name` and `events_service_tier_name` that you specify on each node.

5. Restart the Events Service on all nodes in the cluster: on the Controller host, navigate to `<controller_home>/platform_admin/events-service/` and enter the following command: `/bin/platform-admin.sh restart-events-service`
6. In the Controller UI, go to the Applications table and open the dashboard for the events_service_app_name application. (You might need to wait a few minutes for this application to appear as the Events Services on the nodes restart and begin sending data to the Controller.)
7. In the Application Dashboard, choose **Configure** > **Instrumentation**.
8. Select the `events_service_tier_name` tier and choose **Use Custom Configuration for this Tier**.
9. Under Custom Match Rules, create a new rule with the following attributes:

   - Entry Point = **Servlet**
   - Split Transactions Using Request Data = Use the first **4** segments in Transaction names

Update the Java Temporary Directory for Events Service Nodes

You can set up an optional override of the temporary Java directory for the Events Service by using the Platform Admin in the CLI. Complete the steps below to update the Java Temporary Directory for the Event Service.

> After the Events Service settings update, the Events Service will require a restart initiated from the Platform Admin CLI or Enterprise Console GUI.

1. Run the command after installation:

```
/root/install/pa/platform-admin/bin/platform-admin.sh submit-job --service events-service --job
update_jvm_temp_dir --args jvmTempDir=/var/tmp
```

2. Make sure that the configuration is effective by running the following command:

```
/root/install/pa/platform-admin/bin/platform-admin.sh list-service-configurations --service events-service
```

3. Add the configurations for job `update_jvm_temp_dir`:

```
jvmTempDir (STRING): [/var/tmp]
```

4. To undo, use the following command:

```
/root/install/pa/platform-admin/bin/platform-admin.sh submit-job --service events-service --job
update_jvm_temp_dir
```

5. To add configurations for job `update_jvm_temp_dir` use the following command:

```
jvmTempDir (STRING): [null]
```

Load Balance Events Service Traffic

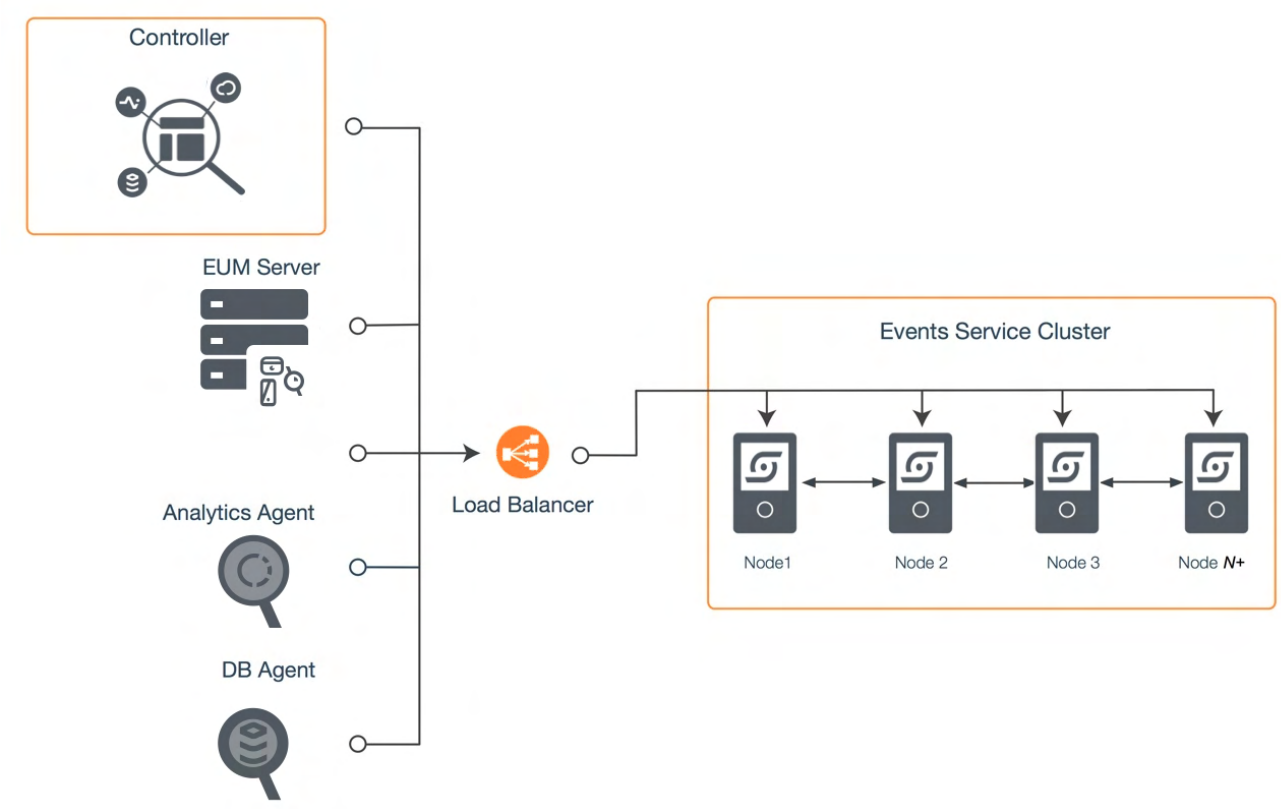
This page takes you through the sample configuration for a load balancer for the Events Service. It introduces you to the concepts and requirements around load balancing Events Service traffic.

Load Balancing Events Service Traffic Overview

To distribute load among the members of an Events Service cluster, you need to set up a load balancer. For a single node Events Service deployment, using a load balancer is optional but recommended, since it minimizes the work of scaling up to an Events Service cluster later.

To configure the load balancer, add the Events Service cluster members to a server pool to which the load balancer distributes traffic on a round-robin basis. Configure a routing rule for the primary port (9080 by default) of each Events Service node. Every member of the Events Service cluster, primary node or not, needs to be included in the routing rule. Keep in mind that increasing the size of the cluster will involve changes to the load balancer rules described here.

The following figure shows a sample deployment scenario. The load balancer forwards traffic for the Controller and any Events Service clients, Analytics Agents in this example.



About these Instructions

The following instructions describe how to install and configure a load balancer for the Events Service. The steps below provide two examples: load balancing with an Nginx and load balancing with HAProxy with SSL termination at the load balancer. The steps demonstrate commands in a CentOS 6.6 Linux operating system environment.

No two environments are exactly alike, so be sure to adapt the steps for your load balancer type, operating systems, and other site-specific requirements.

Nginx Sample Configuration

1. Install the Nginx software. You can install Nginx on most Linux distributions using the built-in package manager for your type of distribution, such as `apt-get` or `yum`. On a CentOS system, you can use `yum` as follows:

   ```
   sudo yum install epel-release
   sudo yum install nginx
   ```

2. Add the following configuration to a new file under the Nginx configuration directory, for example, to `/etc/nginx/conf.d/eventservice.conf`.

```
upstream events-service-api {
   server 192.3.12.12:9080;
   server 192.3.12.13:9080;
   server 192.3.12.14:9080;
   server 192.3.12.15:9080;
   keepalive 15;
}
server {
   listen 9080;
   location / {
      proxy_pass http://events-service-api;
      proxy_http_version 1.1;
      proxy_set_header Connection "Keep-Alive";
      proxy_set_header Proxy-Connection "Keep-Alive";
   }
}
```

In the example, there's a single upstream context for the API-Store ports on the cluster members. By default, Nginx distributes traffic to the hosts on a round-robin basis.

3. Check the following operating system settings on the machine:
   - Permit incoming connections in the firewall built into the operating system, or disable the firewall if it is safe to do so. On CentOS 6.6, use the following command to insert the required configuration in `iptables`:

     ```
     sudo iptables -I INPUT -p tcp --dport 9080 -j ACCEPT
     ```

     To turn off the firewall, you can run these commands

     ```
     sudo service iptables save
     sudo service iptables stop
     sudo chkconfig iptables off
     ```

   - Disable if necessary selinux security enforcement by editing `/etc/selinux/config` and setting `SELINUX=disabled`. Restart the computer for this setting to take effect.
4. Start Nginx:

   ```
   sudo nginx
   ```

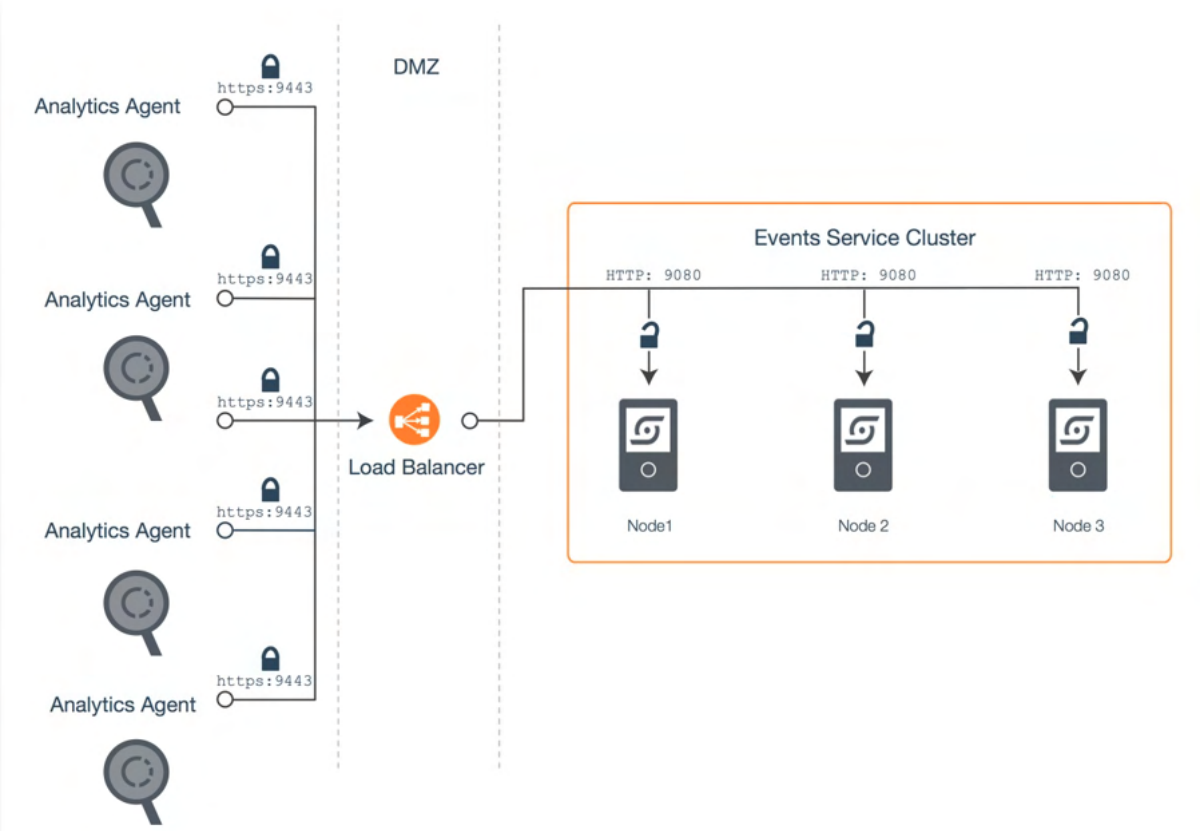
   Nginx starts and now direct traffic to the upstream servers. If you get errors regarding unknown directives, make sure you have the latest version of Nginx.

HA Proxy Sample Configuration: Terminating SSL at the Load Balancer

By terminating SSL at the load balancer in front of the Events Service cluster, you can relieve the Events Service machines from the processing burden of SSL. Since the connections between the load balancer and Events Service machines are not secured in this scenario, it is only suitable for deployments in which the load balancer and Events Service machines reside within an internal, secure network.

The following instructions describe how to set up SSL termination at the load balancer. These steps use HAProxy as the example load balancer. An overview of the steps are:

The following diagram shows a sample deployment reflected in the configuration steps:



Before Starting

To perform these steps, you need:

- Root access on the load balancer machine
- OpenSSL installed on the load balancer machine
- HAProxy software (minimum version HAProxy 1.5) on the load balancer machine

Step 1: Install the HAProxy Software

If not already installed, install HAProxy on the load balancer machine. The manner in which you install it depends on your operating system and the package manager it uses. If using `yum` package manager on Linux, for example, enter the following command:

```
sudo yum install haproxy
```

Step 2. Create the Security Certificate

The security certificate secures the connection between the load balancer and Events Service clients, including the Application Analytics Agent. You can use a self-signed certificate or a certificate signed by a certificate authority (CA) to secure the connection between the load balancer and clients. The following steps walk you through each scenario:

- Create a Self-Signed Certificate on the Load Balancer Machine
- Create a CA-Signed Certificate

For production use, AppDynamics strongly recommends the use of a certificate signed by a third-party CA or your own internal CA rather than a self-signed certificate.

Create a Self-Signed Certificate on the Load Balancer Machine

1. From the command line prompt on the Load Balancer machine, create a directory for the certificate resources and change to that directory:

   ```
   sudo mkdir -p /etc/ssl/private
   cd /etc/ssl/private/
   ```

2. Create the certificate by running the following command, replacing <number_of_days> with the number of days for which you want the certificate to be valid, such as 365 for a full year:

```
sudo openssl req -x509 -nodes -days <number_of_days> -newkey rsa:2048 -keyout ./events_service.key -out .
/events_service.crt
```

3. Respond to the prompts to create the certificate. For the Common Name, enter the hostname for the load balancer machine as identified by external DNS (that is, the hostname that agents will use to connect to the Events Service). This is the domain that will be associated with the certificate.
4. Put the certificate artifacts in a PEM file, as follows:

```
chmod 600 events_service.crt events_service.key
cat events_service.crt events_service.key > events_service.pem
chmod 600 events_service.pem
```

Create and Install a Certificate Signed by a Certificate Authority

1. From the command line prompt of the Load Balancer machine, create a directory for the certificate resources and change to that directory:

```
sudo mkdir -p /etc/ssl/private
cd /etc/ssl/private/
```

2. Generate a Certificate signing request (CSR) based on the private key. For example:

```
openssl req -new -sha256 -key /etc/ssl/private/events_service.key -out /etc/ssl/private/events_service.
csr
```

3. Submit the `events_service.csr` file to a third-party CA or your own internal CA for signing. When you receive the signed certificate, install it and the CA authority root certificate.
4. Depending on the format of the certificates returned to you by the Certificate Authority, you may need to put the certificate and key in PEM format, for example:

```
chmod 600 <ca_crt> events_service.key
cat <ca_crt> <intermediate_ca_crt_if_any> events_service.key > events_service.pem
chmod 600 events_service.pem
```

In the command, replace <ca_crt> with the certificate returned to you by the Certificate Authority. Include any intermediate CA certs, if present, when creating the PEM file.

Step 3. Configure the Load Balancer

1. Open the HAProxy configuration file for editing, `/etc/haproxy/haproxy.cfg`.
2. Insert the following configuration at the end of the file. Replace the placeholder addresses with the host names or IP addresses of the cluster machines. The port should be the primary listening ports of the Events Service nodes.

```
     frontend events_service_frontend
      bind *:9443 ssl crt /etc/ssl/private/events_service.pem
      mode tcp
      reqadd X-Forwarded-Proto:\ https
      default_backend events_service_backend

      backend events_service_backend
      mode tcp
      balance roundrobin
        server node1 192.3.12.12:9080 check
        server node2 192.3.12.13:9080 check
        server node3 192.3.12.14:9080 check
```

3. Start the HAProxy load balancer:

```
sudo service haproxy restart
```

Step 4: Configure the Agent

Perform these steps on each machine on which the Analytics Agent runs.

1. Transfer a copy of the signed certificate, `events_service.crt`, to the home directory (denoted as `$HOME` in the instructions below) of the machine running the agent using Secure Copy (`scp`) or the file transfer method you prefer.
2. Copy the certificate file to the directory location of the trust store used by the agent:

   ```
   cp $HOME/events_service.crt $JAVA_HOME/jre/lib/security/
   ```

3. Navigate to the directory and make a backup of the existing `cacerts.jks` file:

   ```
   cd $JAVA_HOME/jre/lib/security/
   cp cacerts.jks cacerts.jks.old
   ```

4. Import the certificate into the Java keystore:

   - If using a signed certificate, import the certificate as follows:

     ```
     keytool -import -trustcacerts -v -alias events_service -file /path/to/CA-cert.txt -keystore
     cacerts.jks
     ```

   - If using a self-signed cert, import the certificate as follows:

     ```
     keytool -import -v -alias events_service -file events_service.crt -keystore cacerts.jks
     ```

   When prompted, enter the password for the truststore (default is changeit) and enter yes when asked whether to trust this certificate.
5. Verify that the certificate is in the truststore:

   ```
   keytool -list -keystore cacerts.jks -alias events_service
   ```

6. Navigate to the installation folder of the Analytics Agent and edit `conf/analytics-agent.properties` to change the value of the HTTP endpoint property:

   ```
   http.event.endpoint=https://<External_DNS_hostname_Load_Balancer>:9080
   ```

7. Configure the following property names and pathways:

   ```
   https.event.trustStorePath=<absolute path to trust store>
   ```

   ```
   https.event.trustStorePassword=<base64 encoded password>
   ```

8. Start the Analytics Agent (or restart it, if it is already running).
9. Check the health of the agent. In a web browser, you can do so by going to the health check URL at `http://<analytics_agent_host>:9091/healthcheck?pretty=true`.

   If the agent is operating normally, the healthy field is set to true, as in the following example:

   ```
   "analytics-agent / Connection to https://<External_DNS_hostname_Load_Balancer>:9443/v1" :
   { "healthy" : true }
   ```

Step 5: Configure the Controller

If not already done, configure the connection from the Controller to the Events Service through the load balancer using a secure connection as well:

1. Transfer a copy of the signed certificate, `events_service.crt`, to the home directory (denoted as `$HOME` in the instructions below) of the machine running the Controller using Secure Copy (`scp`) or the file transfer method you prefer.
2. Navigate to the directory containing the Controller trust-store (as determined by the Controller startup parameter -Djavax.net.ssl.trustStore).
3. Make a backup of the existing `cacerts.jks` file:

```
cp cacerts.jks cacerts.jks.old
```

4. Import the certificate into the Java keystore:
    - If using a signed certificate, import the certificate as follows:

      ```
      keytool -import -trustcacerts -v -alias <ca_cert_name> -file /path/to/CA-cert.txt -keystore
      cacerts.jks
      ```

    - If using a self-signed cert, import the certificate as follows:

      ```
      keytool -import -v -alias events_service -file events_service.crt -keystore cacerts.jks
      ```

    When prompted, enter the password for the truststore (default is changeit) and enter yes when asked whether to trust this certificate.
5. Verify that the certificate is in the truststore:

   ```
   keytool -list -keystore cacerts.jks -alias events_service
   ```

6. Restart the Controller.
7. From the Administration Console, search for the following Controller Setting: `appdynamics.on.premise.event.service.url`.
8. Set its value to the Load Balancer URL value: `https://<External_DNS_hostname_Load_Balancer>:9443/v1`

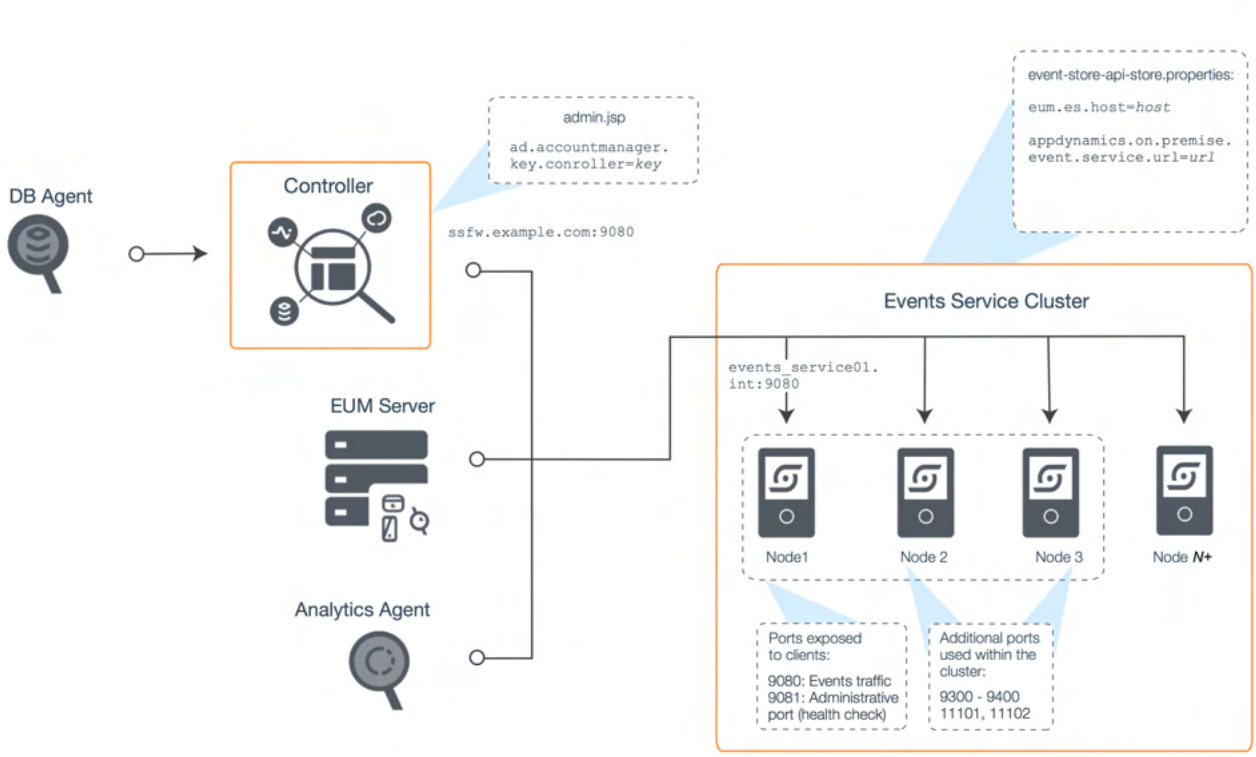
You can now verify that the Analytics UI is accessible and showing data.

Connect to the Events Service

This component is available for on-premises Controllers only. SaaS Controllers are managed by AppDynamics.

You must be connected to the AppDynamics Events Service to send data to it. AppDynamics uses API keys and connection URLs to establish connections between components.

You can configure these connection settings on the **Controller Settings** page of the **Admin Console** (see Access the Administration Console). The following sections describe the required connection settings for Database Visibility, Analytics, and End User Monitoring.



Database Visibility and Analytics Connection Settings

The Database Visibility module stores some of its data (such as wait state information and query information) in the Events Service. The Database Visibility module is the Database Monitoring product, which is already bundled with the controller. If you have both Analytics and Database Visibility installed, we recommend that they share an Events Service instance or cluster instance.

To connect these modules to the Events Service, open the **Admin Console** > **Controller Settings** and set values for the following properties:

- Set `appdynamics.on.premise.event.service.key` to the corresponding key in the properties file for Database Visibility.
- Set `appdynamics.on.premise.event.service.url` to the Events Service endpoint URL.
- Set `appdynamics.non.eum.events.use.on.premise.event.service` to `true`.

To set up the properties files required for Database Visibility:

1. Make note of the `appdynamics.on.premise.event.service.key` value from the **Admin Console**.
2. Open the file called `events-service-api.properties`, which you can find under the Events Service conf directory.
3. In the Events Service file, `events-service-api-store.properties`, ensure that the On-Premise Events key matches with the values of `ad.accountmanager.key.controller` and `ad.accountmanager.key.mds`
    a. See Configure the Events for details.
4. Stop and start the Analytics Service through the Enterprise Console or command line to pick up the new values.

End-User Monitoring Connection Settings

1. Open **Admin Console > Controller Settings**
2. Locate `eum.es.host` and verify it is set to the proper Events Service endpoint URL.
3. Navigate to the `<EUM_HOME>/eum-processor/bin/eum.properties` file
4. Set the following property values to connect the EUM to the Events Service:

Sample

```
analytics.enabled=true
analytics.serverScheme=http
analytics.serverHost=events.service.hostname
analytics.port=9080
analytics.accountAccessKey=1a59d1ac-4c35-4df1-9c5d-5fc191003441
```

5. Open the **Admin Console** > **Controller Settings**
6. Set the `eum.es.host` to the Events Service endpoint URL.

The value of `appdynamics.es.eum.key` will automatically be set to the property `analytics.accountAccessKey` of the file `<EUM_HOME>/eum-processor/bin/eum.properties`.

Proxy Connection Settings

If you are connecting Database Visibility or Analytics through a proxy, you must set values for the following properties:

- `appdynamics.on.premise.event.service.proxy.host`
- `appdynamics.on.premise.event.service.proxy.port`
- `appdynamics.on.premise.event.service.proxy.user`
- `appdynamics.on.premise.event.service.proxy.password.file`
- `appdynamics.on.premise.event.service.proxy.use.ssl`

If you are connecting EUM through a proxy, you must set values for the following properties:

- `appdynamics.controller.http.proxyHost`
- `appdynamics.controller.http.proxyPort`
- `appdynamics.controller.http.proxyUser`
- `appdynamics.controller.http.proxyPasswordFile`

Back Up Events Service Data

Backing up Events Service data helps you to recover from hardware or another type of failure of an Events Service machine. A snapshot represents the backed up data for the entire Events Service cluster. In addition to using it for failure recovery, you can use a snapshot to migrate Events Service to a new instance.

The Events Service tool—`events-service.sh` for Linux and `events-service.exe` for Windows—includes commands for preparing the system for backing up with snapshots, generating a snapshot, and restoring from a snapshot, as described below.

The following instructions show sample commands for Linux. If using Windows, be sure to use the `events-service.exe` form of the executable rather than the `.sh` form, and adjust the sample directory paths as needed.

Prepare the File System

When planning your backup strategy, it is important to consider the storage location and frequency of backups. The system that will serve at the repository for snapshots must be able to handle high I/O demands in a performant manner. SSD-based storage is recommended. Also, ensure you have enough disk space on the repository system.

Only the first snapshot results in a full copy of the data. Each subsequent snapshot is incremental, applying only the changes since the last snapshot. Backing up frequently, therefore, does not result in substantially more storage overhead than backing up infrequently.

The Events Service includes tools for setting up the snapshot repository. It supports the following snapshot repository location types:

- A file system location that is shared among the Events Service nodes.
- An Amazon S3 bucket

After choosing and preparing the system that will host the snapshot, set up each Events Service node as follows:

1. If using FS, mount the shared filesystem at the default location for the backup repository, `<appd_home>/events-service/appdynamics-events-service-backup`. To change this backup location, set the the `ad.es.backupmanager.path.repo` setting in `conf/events-service-api-store.properties`. Keep in mind, however, that changing the properties file requires a restart of the Events Service node to have the change take effect.
2. Set up the repository on each node. Use the appropriate command for your repository type:
    - For shared file system:

      ```
      bin/events-service.sh snapshot-configure-fs -p conf/events-service-api-store.properties
      ```

    - For Amazon S3:

      ```
      bin/events-service.sh snapshot-configure-s3 -p conf/events-service-api-store.properties -bucket "s3-bucket-name"
      ```

      The `snapshot-configure-s3` command accepts additional optional arguments, including arguments for passing the access key and secret key for S3. Run `"bin/events-service.sh -h"` to view all options.

    Look for a message similar to the following to verify that the configuration succeeded:

    ```
    [2015-12-17T15:43:04,092-08:00] Successfully configured snapshot repository!
    ```

Create a Snapshot

After setting up the repository, you can generate a snapshot of the Events Service data. If you are backing up a cluster, you only need to run the command from one of the primary nodes. If you have more than one cluster, generate a snapshot for each one. Snapshots are cluster-specific.

Generate a snapshot:

```
bin/events-service.sh snapshot-run -p conf/events-service-api-store.properties
```

You can use this command to script regular backups based on your backup policy. The following output indicates that backup was successful:

```
Take snapshot request executed successfully. Snapshot itself may still be in progress.
```

To check the progress of the snapshot, use `snapshot-status`.

```
bin/events-service.sh snapshot-status -p conf/events-service-api-store.properties
```

If you don't specify a snapshot ID, the command gets the status for the most recent snapshot. You can use the `snapshot-list` command to see a list of available snapshots.

Restore from a Snapshot

By restoring a snapshot, you replace the data store configured for the Events Service with one that was previously saved as a snapshot. When you restore from a snapshot, you restore from a snapshot for a specific cluster. Run the command for each cluster

To restore a snapshot, use the `snapshot-restore` command, passing the properties file for the Events Service instance you are backing up. The following shows an example with sample output:

```
bin/events-service.sh snapshot-restore -p conf/events-service-api-store.properties
[2015-12-17T17:02:52,264-08:00] HV000001: Hibernate Validator 5.0.2.Final
[2015-12-17T17:02:52,811-08:00] Restore snapshot request executed successfully. Restore is now in progress. Use
the snapshot-status command to view the current restore status.
```

Check the status of the snapshot restore using the `snapshot-restore-status` command. For example:

```
bin/events-service.sh snapshot-restore-status -p conf/analytics-api-store.properties
[2017-01-03T14:37:46,647-08:00] HV000001: Hibernate Validator 5.2.2.Final
[2017-01-03T14:37:47,027-08:00] Restore is complete, your cluster should be fully functional!
```

You can restore a specific snapshot by passing the snapshot ID with the command. Otherwise, the most recent snapshot is restored.

```
bin/events-service.sh snapshot-restore -p conf/events-service-api-store.properties -id <snapshot_id>
```

You can use the `snapshot-list` command to get a list of snapshot IDs.

Migrating Events Service Data

In addition to data backup and recovery, you can use the snapshot utility to migrate data from one Events Service instance to another.

The target Events Service needs to be a fresh installation; that is, data in two different Events Service instances cannot be merged. Be sure to avoid configuring the Events Service URL with the new instance location in the Controller configuration until you have completed these steps.

To migrate Events Service data, follow these general steps:

1. Prepare the new Events Service nodes, as described above.
2. On each new Events Service node, mount the shared directory where the repository is located.
3. From a primary node in the new cluster, restore the snapshot by ID, as described above, passing the property file that defines the new cluster as the `-p` argument.
4. When finished, change the connection from the Controller to the Events Service and any Events Service clients, as described in Connect to the Events Service, to use the new instance.

Upgrade the Events Service

On this page:

Related pages:

- Data Field Naming for Events Service 4.5.3 and Above

You can upgrade the Events Service either manually or by using the Enterprise Console.

You must upgrade manually if you:

- Do not use the Enterprise Console to deploy the Events Service
- Upgrade Events Service nodes hosted on remote Windows machines. The Enterprise Console does not support remote operations on Windows.

For on-premises deployments, 4.5.2 is the latest version of the Events Service. If you upgrade to a version of the Events Service other than the latest, run the Enterprise Console installer for the desired Events Service version.

For SaaS deployments upgrading to Events Service version 4.5.3 or later, modify your data field names to conform to the latest requirements, as described here.

Before the Upgrade

AppDynamics removed Search Guard from the on-premises Events Service version 4.5.2.20561. If your deployment requires Search Guard or a comparable feature, do not upgrade to this version of the Events Service.

AppDynamics will provide an alternative security feature with the next on-premises Events Service release.

See the Support Advisory for more details.

1. Download and install the new Enterprise Console.
2. Plan the order in which to upgrade platform components (not just Events Service).
3. Modify the `events-service-api-store.properties` file.
    - Replace the absolute path `APPLICATION_HOME` property in the file with an actual path.
    - This is required because while performing a discover and upgrade job, the Enterprise Console is unable to migrate the data directory for custom environment variables in the file. Failure to modify the file causes the upgraded Events Service to start with a new, blank data set.
4. Back up your `events-service.vmoptions` file.
    - This is required because the `events-service.vmoptions` file is not maintained when the Events Service is upgraded. After the upgrade completes, merge your backup copy of `events-service.vmoptions` into the new file.

Upgrading to a pre-4.1 Events Service

To upgrade the Events Service software to a version earlier than 4.1, you must first manually upgrade the service to 4.1, and then use the Enterprise Console to discover the Events Service nodes.

Upgrade the Events Service

1. Run the upgrade Events Service command.
2. Discover the Events Service nodes using the Enterprise Console.

Verify the Upgrade

Once the upgrade completes:

1. The Events Service process should have restarted—verify its health status in the Enterprise Console GUI.
2. Merge your backup copy of the `events-service.vmoptions` file into the new copy of the file created by the upgrade.

Startup Script Paths

After an upgrade, you will find the Events Service startup script paths below:

```
<installDir>/appdynamics/events-service/processor/bin/events-service.sh
```

This may be different from their paths before the upgrade.

Upgrade the Events Service Using the Enterprise Console

This page describes how to upgrade a scaled-out Events Service on primarily Linux machines using the Enterprise Console.

Upgrade the Events Service Using GUI

If there is a Events Service upgrade available, you can begin the upgrade process either on the Custom Install or Events Service page in the GUI.

> You do not need to stop the Events Service before upgrading because the Enterprise Console does this for you.

After you upgrade the Events Service, upgrade the EUM server if it is part of your deployment. Then, upgrade the Controller.

Upgrade the Events Service from 4.1.x, 4.2.x, and 4.3.x to 4.4.x or Latest

> The Enterprise Console supports the installation of the Events Service on a Windows environment for a single node install. If you have several remote nodes, you will need to do a manual upgrade of the Events Service cluster and set the keys manually. See Connect to the Events Service

> The Enterprise Console manages the Controller and Events Service together, so their keys found in `events-service-api-store.properties` are set and synced to each other upon upgrade. However, we recommend confirming that the keys were synced correctly: `appdynamics.on.premise.event.service.key == ad.accountmanager.key.controller`

To upgrade the Events Service from 4.1.x, 4.2.x, and 4.3.x to 4.4.x or the latest version, you can use the Discover and Upgrade feature:

1. Check that you have fulfilled the Enterprise Console prerequisites before starting.
2. Open a browser and navigate to the GUI:

   ```
   http(s)://<hostname>:<port>
   ```

   9191 is the default port.
3. Navigate to the **Install** homepage and click **Custom Install**.
4. Name the Platform:

   a. Enter a Name and the Installation Path for your platform.

   > The Installation Path is an absolute path under which all of the platform components are installed. The same path is used for all hosts added to the platform. Use a path which does not have any existing AppDynamics components installed under it. The path you choose must be writeable, i.e. the user who installed the Enterprise Console should have write permissions to that folder. Also, the same path should be writable on all of the hosts that the Enterprise Console manages.
   >
   > Example path: `/home/appduser/appdynamics/product`

5. Add a Host:

   > Note that all services on Windows machines must be installed on the Enterprise Console host since the Enterprise Console does not support remote operations on Windows. Therefore, you cannot add a host in a Windows Enterprise Console machine.

   a. Enter the host machine-related information: Host Name, Username, and Private Key. This is where the Events Service will be upgraded. Therefore, this needs to point to the host machine where the Events Service is currently up and running. For more information about how to add credentials and hosts, see Administer the Enterprise Console.
6. Click **Platforms**. Select the newly created platform and navigate to the **Events Service** page.
7. Discover Events Service:
   a. Select **Discover & Upgrade Events Service**.
   b. Select an available Target Version from the dropdown.

   > The list is populated by versions that the Enterprise Console is aware of. This means that you can upgrade the Events Service to any intermediate version or to the latest version as long as the Enterprise Console installer has been run for those versions.

   c. Enter the Installation Directory.
   d. Enter the Events Service Host.
8. Click **Submit**.

The Enterprise Console will onboard the Events Service on the selected host machine to the application build. When the Enterprise Console discovers a component, it also checks to see if an upgrade is available and performs the upgrade. Plan for a downtime of the Events Service availability during this time. You can view the status of the upgrade job on the Jobs page.

Once the upgrade is complete, the Events Service Health status and related information can be accessed from the Events Service page.

After upgrading to 4.4.x or the latest, the commands to start and stop the Events Service change. See Administer the Events Service for more information.

Upgrade the Events Service from 4.4.x to Latest

To upgrade the Events Service from 4.4.x to the latest version, you can use the Upgrade Events Service feature:

1. Check that you have fulfilled the Enterprise Console prerequisites before starting.
2. Upgrade the Enterprise Console to the latest version.
3. Open a browser and navigate to the GUI:

   ```
   http(s)://<hostname>:<port>
   ```

   The default port is 9191.
4. Navigate to the **Events Service** page of the platform.
5. Select the Events Service host you would like to upgrade.
6. Click **Upgrade Events Service**.
7. Select an available Target Version from the dropdown list.

   The list is populated by versions that the Enterprise Console is aware of. This means that you can upgrade the Events Service to any intermediate version or to the latest version as long as the Enterprise Console installer has been run for those versions.
8. Confirm the Upgrade.

Upgrade the Events Service Using CLI

If there is an Events Service upgrade available, you can begin the upgrade process using the application CLI.

You do not need to stop the Events Service before upgrading because the Enterprise Console does this for you.

After you upgrade the Events Service, upgrade the EUM server if it is part of your deployment. Then, upgrade the Controller.

Upgrade the Events Service from 4.1.x, 4.2.x, and 4.3.x to 4.4.x or Latest

To upgrade the Events Service software from 4.1.x, 4.2.x, and 4.3.x to 4.4.x or the latest version, you will need to first download and install the Enterprise Console installer before performing the following steps:

1. Create a platform as follows:

   ```
   bin/platform-admin.sh create-platform --name <platform_name> --installation-dir
   <platform_installation_directory>
   ```

   The installation directory is the directory where the Enterprise Console installs all platform components.

   To avoid any failures, do not use the 4.3 or earlier Platform Admin installation directory. Instead, provide a new/empty directory.
2. Add the SSH key that the Enterprise Console will use to access and manage the Events Service hosts remotely. (See Create the SSH Key for more information):

   ```
   bin/platform-admin.sh add-credential --credential-name <name> --type ssh --user-name <username> --ssh-
   key-file <file path to the key file>
   ```

   `<file path to the key file>` is the private key for the Enterprise Console machine. The installation process deploys the keys to the Events Service hosts.

3. Add hosts to the platform, passing the credential you added to the platform:

```
bin/platform-admin.sh add-hosts --hosts es_host_1 es_host_2 es_host_3 --credential <credential name>
```

4. Discover the Events Service nodes that are not yet integrated:

```
bin/platform-admin.sh submit-job --service events-service --job discover-upgrade --platform-name
<name_of_the_platform> --args destinationDirectory=<path_to_events_service> serviceActionHost=<es_host_1
es_host_2 es_host_3>
```

   This command integrates the nodes into the Enterprise Console and also upgrades them. If your upgrade fails, you can resume by passing the flag `useCheckpoint=true` as an argument after `--args.`

After upgrading to 4.4.x or the latest, the commands to start and stop the Events Service change. See Administer the Events Service for more information.

Upgrade the Events Service from 4.4.x to Latest

Upgrades from 4.4.x to the latest version can be performed on the Events Service page of the Enterprise Console or with the following commands:

1. Upgrade the Enterprise Console to the latest version.
2. Navigate to the `<Enterprise Console home directory>/platform-admin directory`.
3. If it has been more than one day since your last session, you will have to log in with the following command:

```
bin/platform-admin.sh login --user-name <admin_username> --password <admin_password>
```

4. Apply the upgrade to the Events Service nodes with the following command:

```
bin/platform-admin.sh submit-job --service events-service --job upgrade --platform-name
<name_of_the_platform>
```

   If your upgrade fails, you can resume by passing the flag `useCheckpoint=true` as an argument after `--args.`

Upgrade the Events Service Manually

This page describes how to manually upgrade an Events Service. This is useful for when you did not use the Platform Administration Application or the Enterprise Console to deploy the Events Service. The primary case for this would be for when you need to upgrade the Events Service nodes hosted on remote Windows machines.

Perform the Manual Upgrade

> **Prerequisite**
>
> Java 1.8 is required for `events-service.exe` to work.

To upgrade the Events Service manually:

1. Download the Events Service distribution, `events-service.zip`, from the AppDynamics download site to the Events Service machine.
2. Stop the Events Service processes:

   ```
   bin/events-service.sh stop
   ```

3. Rename the existing Events Service directory, for example, to `events-service-backup`.
4. Unzip the Events Service distribution archive you downloaded to the location where you want the Events Service to run.
5. Migrate configuration changes from the properties files in the backup Events Service directory to the `conf/events-store-api-store.properties` file in the new Events Service directory. Depending on which type of deployment you are using, this involves inspecting and migrating settings from:
   - `events-service-all.properties`, or
   - `events-service-api-store.properties`

   > If you are upgrading from 4.2 to 4.3.x or a later version you must edit the `events-service-api-store.properties` file by replacing the port ranges [9300-9400] with :9300.
   >
   > For example, in 4.2, the `events-service-api-store.properties` file looks like this:
   >
   > ```
   > ad.es.node.unicast.hosts=node1.example.com[9300-9400],node2.example.com[9300-9400],node3.example.com[9300-9400]
   > ```
   >
   > While in 4.3 or later, the file should look like this:
   >
   > ```
   > ad.es.node.unicast.hosts=node1.example.com:9300,node2.example.com:9300,node3.example.com:9300
   > ```

6. Configure the connection to the Events Service in the Controller, and get the Controller key for the Events Service configuration as follows:
   a. With the Controller running, open the Administration Console as the root user.
   b. In the Controller Settings page, search for `appdynamics.on.premise.event.service.url`.
   c. Replace the default value to the internal hostname for the Events Service machine and default Events Service listen port, 9080.
   d. Search for an additional setting, the one you will need to enable the connection from the Events Service to the Controller, `appdynamics.on.premise.event.service.key`.
   e. Copy the value of the property to your clipboard. You will need to configure this in the Events Service properties file next.
7. Configure the connection from the Events Service to the Controller:
   a. In a terminal, navigate to the `events-service` directory:

      ```
      cd events-service
      ```

   b. Open the `conf/events-service-api-store.properties` file for editing.
   c. Find the following property and replace `controller-key` with the copied key:

      ```
      ad.accountmanager.key.controller=controller-key
      ```

8. Ensure that the following critical properties are configured appropriately in the `events-service-api-store.properties` file:
   - `ad.accountmanager.keyNamesCSV=EUM,CONTROLLER,MDS,OPS,SLM,JF`
   - `ad.accountmanager.key.eum=`
   - `ad.accountmanager.key.controller=`

- ad.accountmanager.key.mds=
- ad.accountmanager.key.ops=
- ad.accountmanager.key.slm=
- ad.accountmanager.key.jf=
- ad.accountmanager.key.service=
- ad.jvm.heap.min=1g
- ad.jvm.heap.max=1g
- ad.es.jvm.heap.min=1g
- ad.es.jvm.heap.max=1g

9. Move ad.es.node.unicast.hosts property in events-services-api-store.properties while upgrading from 4.2 to 4.3.x or later.
10. Save and close the events-service-api-store.properties file.
11. Verify that the new Events Service home directory exists. The Event Service home directory is determined by the ad.es.path.home property in the property file used to start up the Events Service.
    If the directory does not exist, create it. For example, create the following directory: /opt/appdynamics/events-service/appdynamics-events-service
12. Move (do not copy) the old Events Service data directory to the new Events Service home directory. For example:

    ```
    mv /opt/appdynamics/events-service-backup/appdynamics-events-service/data /opt/appdynamics/events-service
    /appdynamics-events-service/
    ```

13. Restart the Events Service processes from the new directory:

    ```
    nohup bin/events-service.sh start -p conf/events-service-api-store.properties &
    ```

14. Check the health of the node.
    Windows
    `bin\events-service.exe check-health -hp localhost:9081`
    Linux
    `curl -XGET localhost:9081/healthcheck?pretty=true`

    Verify that "Healthy" appears as the service status, indicating that the process is operating normally:

    ```
    [appduser@controller-one events-service]$ bin/events-service.exe check-health -hp 192.168.33.22:9081
    [2015-12-09T18:30:45,342-08:00] HV000001: Hibernate Validator 5.0.2.Final
    [2015-12-09T18:30:45,956-08:00] Individual statuses below:
    [2015-12-09T18:30:45,956-08:00] [192.168.33.22:9081] status is [200 OK]
    [2015-12-09T18:30:45,956-08:00] Overall status Healthy
    ...
    ```

15. Configure the connections from the Analytics Agent, EUM Server, or Database Monitoring agents to the Events Service, as described in Connect to the Events Service.
    For information on performing these steps, see Install the Events Service on Windows.

If you upgrade to 4.4.x from 4.3.x or an earlier version, the commands to start and stop the Events Service change. See Administer the Events Service for more information.

Data Field Naming for Events Service 4.5.3 and Above

This page explains how to update data field names.

To use this procedure appropriately for your deployment, follow the guidelines below.

Deployment	Version	Required Action
SaaS	4.5.3 and later	Update data field names as described below.
On-premises	4.5.2 and older	No action is required, but AppDynamics strongly recommends that you update data field names to prepare for future Events Service releases.

Update Overview

Version 4.5.3 of the Events Service runs Elasticsearch 5.6, whereas previous Events Services run earlier versions of Elasticsearch. To upgrade to Events Service 4.5.3, you may need to rename some fields used in collecting transaction analytics, log analytics, or other types of data.

Review the requirements below and follow instructions where applicable. To understand the rationale for the changes, see More About Field Names.

Requirements

Do Not Use Empty Field Names

In pre-5.6 versions of Elasticsearch it was possible to create fields with empty names. This is no longer allowed. Empty field names now cause indexing errors.

Required action: Give alphanumeric names to any fields whose names are empty.

Do Not Use Dots in Field Names

In the past, some customers have used dots to separate name components in a semantically meaningful way. This is no longer recommended and may cause the upgrade to Events Service 4.5.3 to fail.

Strongly recommended action: Replace dots in field names with hyphens or underscores.

More About Field Names

This section discusses dotted field names, meaning field names with embedded periods ('.'), such as `a.b.c` or `transit.signals.yellow`.

Elasticsearch stores data in JSON documents whose structure is hierarchical. Dotted field names can be used to query into those JSON documents: the field name is treated as a path whose components are separated by dots. See https://www.elastic.co/guide/en/elasticsearch/reference/2.4/dots-in-names.html.

It can be impossible to know whether a dotted field name is intended as a path to a JSON element, or just a plain field name. To treat this problem in a consistent way, Elasticsearch, beginning with version 5.6, always automatically expands dotted field names into hierarchical JSON structures. Each dot creates another level nested lower in the hierarchy.

Events Service 4.5.3 attempts to gracefully handle dots in field names and allow the default behavior of Elasticsearch. However, in a few corner cases, Elasticsearch still fails to index events to which field names correspond. In these situations, the only recourse is to change the field names.

These corner cases can be avoided by following three rules:

1. Field names cannot contain multiple consecutive dots
2. Field names cannot start or end with dots
3. Field names cannot share prefixes

The rest of this section explains these rules.

Field Names Cannot Contain Multiple Consecutive Dots

Field names that contain multiple consecutive dots expand into structures where the name of some elements is the empty string. These are invalid as JSON objects. Note the empty names of the most deeply nested nodes in the following example:

```
"a.very.long.field.name.truncated.with.dots..." ->
"a": {
  "very": {
    "long": {
      "field": {
        "name": {
          "truncated": {
            "with": {
              "dots": {
                "": {
                 "": {
                  "": {
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

Field Names Cannot Start or End with Dots

Field names that start or end with dots expand into structures where the name of some elements is the empty string. These are invalid as JSON objects. Fo r example:

```
".a.b.c" ->
"": {
  "a": {
    "b": {
      "c": "value of field"
    }
  }
}


"a.b.c." ->
"a": {
  "b" : {
    "c": {
      "" : {
      }
    }
  }
}
```

Field Names Cannot Share Prefixes

When two or more field names have the same prefix, it becomes impossible to create valid JSON objects for them all.

Consider the following field names and values:

```
a.b = "alphabaker"

a.b.c = "alphabakercharlie"
```

Trying to share a prefix runs into trouble because:

- all the nodes that need to be created are text nodes, and
- some nodes need to be nested, but
- in JSON, text nodes are not allowed to contain other objects.

We'll demonstrate the problem by examining what happens when Elasticsearch tries to create the JSON objects for our example.

- The first field name results in "b" being mapped to a text node with the value "`alphabaker`."

```
"a": {
  "b": = "alphabaker"
}
```

- To expand the second name, Elasticsearch tries to map "c" to a text node with the value "`alphabakercharlie.`" This fails because "c" needs to be nested within "`b,`" which is a text node and cannot contain nested objects.

Use the Data Migration Tool

The Data Migration tool uses a collection of `Python3` files.

Install the Data Migration Tool

To install the Data Migration tool, download `migration_tool.zip`.

This example shows the unzipped structure:

```
tool/ main.py readme.txt requirements.txt src/ tool.json
```

Set Up the `python3` Environment

1. To verify if `python3` is installed on your system, enter:

   ```
   which python3
   ```

   ⚠ If `python3` is not found, then install `python3` by entering: `sudo yum install python36 -y`

2. To verify if the `pip3` package manager is installed, enter:

   ```
   which pip3
   ```

   ⚠ If the `pip3` package manager is not found, then see Installing Packages and enter: `sudopython3 -m pip install--upgrade pip setuptools wheel`

3. To install the libraries which run the migration python script, enter this command within the Data Migration tool directory:

   ```
   pip3 install -r requirements.txt --user
   ```

Configure the Data Migration Tool

Before you can run the migration script, you must configure it properly. All configuration is stored in the `tool/tool.json` file in JSON format. These sections provide guidance on how to configure each property.

Clusters

Clusters are defined as a collection of Events Service clusters. Each cluster has the following properties:

Properties	Description
`api_url`	URL pointing to one of the Events Service's API nodes or its load balancer.
`certificate_file`	Path to the PEM file.
`check_hostname`	(Optional) Indicates whether to check the hostname when verifying the certificate. Default is `true`.
`es_url`	URL pointing to one of the Elasticsearch's master nodes.
`es_url_internal`	Internal URL pointing to one of the Elasticsearch's master nodes. Used by other clusters for remote re-indexing.
`es_version`	Relates to the Elasticsearch version.
`keys`	Controller and OPS keys for Events Service. These keys are located in the `conf/events-service-api-store.properties` file.

This example defines Events Services: `es2`, `es6`, and `xpack_es6`.

⚠ `es6` has SSL enabled, while `xpack_es6` has Elasticsearch X-Pack enabled.

```
{
  "clusters": {
    "es2": {
      "keys": {
        "CONTROLLER": "27410b11-296a-49e1-b2d2-d2371ab94d64",
        "OPS": "45c25bad-636c-432f-b0bd-f8ec428c8db4"
      },
      "api_url": "35.162.126.253:9080",
      "es_url": "35.162.126.253:9200",
      "es_url_internal": "172.31.12.185:9200",
      "es_version": 2
    },
    "es6": {
      "keys": {
        "CONTROLLER": "7db43bff-97d3-4d5e-828a-a2eacb693e07",
        "OPS": "ac7424d1-ae96-4e10-ad82-a2eca50db133"
      },
      "api_url": "https://34.209.245.68:9080",
      "certificate_file": "/Users/jun.zhai/es6.pem",
      "check_hostname": false,
      "es_url": "34.209.245.68:9200",
      "es_version": 6
    },
    "xpack_es6": {
      "keys": {
        "CONTROLLER": "07b055f8-a97b-4ccb-a239-2267d452c4ea",
        "OPS": "c582cc8a-f3bc-419f-a42a-7bb8adad05b8"
      },
      "api_url": "52.89.86.93:9080",
      "es_url": "http://elastic:1234@52.89.86.93:9200",
      "es_version": 6
    }
  }
}
```

Migration

This section describes the migration properties:

Properties	Description
`accounts`	(Optional) Specify which accounts to migrate. Defaults to everything in source Events Service.
`search_hits`	Maximum documents fetched in the Elasticsearch query. Default is 5000.
`remote_reindex_concurrency`	Maximum number of remote re-index tasks launched concurrently. Default is 4.
`remote_reindex_scroll_batch_size`	Batch size for remote reindex. Default is 8000. See Re-index API.
`reindex_task_polling_interval`	Frequency in seconds of how often to check the status of ongoing remote reindex tasks. Default is 60 seconds.
`starting_max_fields_per_index`	Maximum fields allowed when creating a new index. Default is set to 1000. The value should be same as the `ad.es.event.index.startingMaxFieldsPerIndex` in `conf/events-service-api-store.properties` file.

Migration Properties Examples:

Example 1: Migrates everything from source Events Service:

```
{
  "migration": {
    "search_hits": 5000,
    "remote_reindex_concurrency": 6,
    "remote_reindex_scroll_batch_size": 8000,
    "reindex_task_polling_interval": 60,
    "starting_max_fields_per_index": 1000
  }
}
```

Example 2: Migrate all event types in accounts, `customer15_9611293a-c56f-4c9a-aa11-9f6bffcb42ce`, `log_v1`, and `custom_event` event types in account `customer1_229f6fbf-b42f-4d66-a56b-a2324d8b169d`. This example does not migrate any other accounts.

```
{
  "migration": {
    "accounts": {
      "customer15_9611293a-c56f-4c9a-aa11-9f6bffcb42ce": [],
      "customer1_229f6fbf-b42f-4d66-a56b-a2324d8b169d": [
        "log_v1",
        "custom_event"
      ],
    },
    "search_hits": 5000,
    "remote_reindex_concurrency": 4,
    "remote_reindex_scroll_batch_size": 8000,
    "reindex_task_polling_interval": 60,
    "starting_max_fields_per_index": 1000
  }
}
```

Upgrade the Events Service to >= 20.9.0

Upgrade Procedure

To upgrade the On-premises Events Service to >= 20.9.0, AppDynamics recommends that you follow this procedure.

The Events Service version 20.9.0 is packaged with the Enterprise Console version 21.2.4 or newer.

If the incoming data load is heavy, you may expect delays in Step 3d during the data migration process.

Data Migration

- If you choose not to migrate data, then continue with Steps 1 through 3c, and do not complete Step 3d.
- If you choose to migrate your data then data conflicts in may occur in Step 3d.

Upgrade Completion

- If the upgrade fails, then you should revert the Controller to the old cluster; data loss may occur.
- If the upgrade succeeds, then you can delete the old cluster.

1. Prepare the machine instances:
    a. Identify a utility machine running Linux, to run the Enterprise Console and data migration scripts. There are no strict hardware requirements

Make sure that the older (source) Events Service cluster, new (target) Events Service cluster, and utility machine all reside on the same network.

    b. Ensure the target Events Service cluster has similar or better hardware than that of the source Events Service cluster. Ensure that the operating system (OS) on the new machine has these settings:
        i. Increase the file descriptors limit in the `/etc/security/limits.conf` file, and then reboot the machine. Use `ulimit -n` to verify the value.

```
*               soft    nofile          66000
*               hard    nofile          66000
# End of file
```

        ii. Set `vm.max_map_count` in the `/etc/sysctl.conf` file, and then reboot the machine. Use `cat /proc/sys/vm/max_map_count` to verify the value.

```
vm.max_map_count=262144
```

2. Set up the new staging cluster:
    a. Download and install the latest Enterprise Console, which includes Events Service 21.2.4. See AppDynamics Downloads and Platform Installation Quick Start.
    b. In the Enterprise Console, install the On-premises Events Service. See Install Events Service on Linux.
    c. Enable the load balancer. See Load Balance Events Service Traffic.
    d. To enable the Secure Socket Layer (SSL) in the Events Service, see Enable the Events Service to Use SSL.
    e. To configure the Events Service for data migration, complete this procedure:
        i. Enable HTTP port in `events-service-api-store.properties`:

```
ad.es.node.http.enabled=true
```

        ii. Add the remote reindex `whitelist` in the `events-service-api-store.yml` file:

```
    - className: com.appdynamics.analytics.processor.elasticsearch.configuration.
ElasticsearchConfigManagerModule
      properties:
          nodeSettings:
              cluster.name: ${ad.es.cluster.name}
              ...
              indices.fielddata.cache.size: ${ad.es.fielddata.cache.size}
              reindex.remote.whitelist: "<IP address to one of the nodes in older cluster>:
9200"
```

iii. Restart the new Events Service cluster from the Enterprise Console.

3. Migrate the data:

a. See Use the Data Migration Tool to install and configure the data migration script on the utility instance.
b. Enter this command to migrate the metadata:

```
python main.py migrate metadata es2 es6
```

c. Navigate to the Controller **Admin** page and set your Controller to the new Events Service. See Connect to the Events Service.

If you choose to migrate your data, then proceed to Step 3d.

d. Enter this command to migrate data from the old cluster to the staging cluster:

```
python main.py migrate data es2 es6
```

Uninstall the Events Service

You can remove an on-premises Events Service from individual nodes or all at once. An embedded Events Service is uninstalled along with the Controller.

Uninstall the Events Service with the Enterprise Console

You can uninstall the Events Service through the GUI on the Events Service page.

To do so through the CLI, you can use the `uninstall-events-service command`, which removes the Events Service software and data from all cluster nodes:

After uninstalling Events Service, the only trace of the Events Service remaining on the host may be a file named `orcha-modules.log`. It appears in the logs directory at the former installation root directory. To remove all traces of the Events Service, manually remove the log file after removing the Events Service with the Enterprise Console.

To uninstall the Events Service from a single node with the Enterprise Console, see the Removing a Node section on Administer the Events Service.

Uninstall the Events Service as a Windows Service

You can remove the Events Service as a Windows service after installation through the GUI. You can also use the following command to retain the Events Service on the machine.

To remove the Events Service as a Windows service:

1. Use the list service command to find the service name for the Events Service: `bin\events-service.exe service-list`

   Starting the ZooKeeper alone only brings up the process that manages index rollover. The Events Service node is not fully started until you start the API-Store process as well, as described next.
2. Use the name returned for the service as the -s parameter argument to the following command: `bin\events-service.exe service-uninstall -s "<Name from service-list>"`

   Be sure to enclose the name in double-quotes.

Synthetic Server Deployment

The Synthetic Server dispatches and processes requests and depends on Synthetic Agents for executing and reporting measurements.

The Synthetic Server receives synthetic job requests from the Controller and then the jobs are fetched from the Synthetic Services by the Synthetic Agents. Once the measurement results are received from the Synthetic Agents, the Synthetic Server stores, processes, and transmits the results to the EUM Server.

Installation Overview

To set up a complete on-premises Synthetic Server deployment, therefore, you need to:

1. Install the on-premises Controller or prepare an in-service Controller to work with the EUM Server.
2. Install the on-premises Events Service and configure it to work with your on-premises Controller.
3. Install the on-premises EUM Server and configure it to work with your Events Service and Controller.
4. Install the on-premises Synthetic Server and configure it to work with the EUM Server and the Controller.
5. Install and configure one or both types of Synthetic Agents.
6. Secure the Synthetic Server (recommended).
7. Monitor the Synthetic Server (recommended).

Synthetic Server Components

The on-premises Synthetic Server consists of the following three services:

- Synthetic Scheduler
- Synthetic Shepherd
- Synthetic Feeder Client

This document also discusses the Synthetic Server Feeder, which communicates with the Synthetic Client Feeder, but is only a service of the SaaS Synthetic Server.

Synthetic Scheduler

The first service is the Synthetic Scheduler, which is a cron-like service that sends job requests at configured intervals. The Synthetic Scheduler handles the CRUD operations for jobs and manages the events generated for synthetic warnings and errors that occur in the measurement results. The Synthetic Scheduler also validates the beacons, triggers warning and error events if needed, and forwards the beacons to the EUM Server.

Synthetic Shepherd

The second service is Synthetic Shepherd. This service manages and dispatches jobs to the Synthetic Agents. In addition, the Synthetic Shepherd saves the measurement results to the filesystem and sends beacons containing the data to the Synthetic Scheduler.

Synthetic Feeder Client

The third service is the Synthetic Feeder Client that communicates with the SaaS Synthetic Feeder Server to access the Synthetic Hosted Agents. (If you are only deploying Synthetic Private Agents, you do not need to use the Synthetic Feeder Client.) These services use the WebSocket protocol to coordinate data transfer to your system without having to open any ports in the firewall.

Synthetic Agents

When deploying the on-premises Synthetic Server, you can deploy one or both Synthetic Agent types:

- Synthetic Hosted Agents - Synthetic Agents that are hosted and maintained by AppDynamics
- Synthetic Private Agents - Synthetic Agents that you install, configure, run, and maintain in your infrastructure

Comparison of the Synthetic Agent Types

The following table compares the two types of agents and provides the benefits and main use cases for both.

Synthetic Agent Type	Key Benefits / Use Cases

Synthetic Hosted Agent	• Access to a fleet of geographically distributed agents • Reduced ownership/resource costs: no hardware or cloud computing costs • Ease-of-use: no need to deploy/configure/manage agents • Scalability: Synthetic Hosted Agents are only deployed when needed, and more agents are readily available if the workload increases
Synthetic Private Agent	• Monitoring of internal sites and services that are not publicly accessible • Complete control over the agent configurations and environment

Overview of Installation and Configuration Steps

The following table provides an overview of the installation and configuration steps for each type of Synthetic Agent.

Synthetic Agent Type	Required Steps
Synthetic Hosted Agent	1. Acquire the license "Browser Synthetic User Monitoring - Hosted Agent - On-Premise". 2. Verify that the license has an HMAC key. 3. Configure SSL for the Synthetic Server (recommended). 4. Connect the on-premises Synthetic Server to the SaaS EUM API Server and SaaS Synthetic Server.

Synthetic Private Agent	1. Acquire one of the following licenses:
	  • Browser Synthetic Monitoring - Private Agent - Per Location (on-premises)
	  • Browser Synthetic Monitoring - Private Agent - Unlimited Locations (on-premises)
	2. Install Synthetic Private Agents.
	3. Connect the Synthetic Private Agent to the on-premises Synthetic Server .
	4. Configure SSL for the Synthetic Server (recommended).
	5. Start and maintain the Synthetic Private Agent.

Summary of Synthetic Server and Agents

The table below summarizes the function and ports used by each service and the Synthetic Agents:

Service/Agent	Functions	Protocol	Default Ports
Synthetic Scheduler	• Sends requests to execute jobs based on a configured frequency. • Validates the beacons containing the measurement results. • Handles the CRUD operations for jobs.	HTTP	12101
		HTTPS	12102
Synthetic Shepherd	• Registers the Synthetic Agents. • Creates and maintains the queue of all measurement requests. • Manages how the results arriving from the agents are processed, stored and forwarded. • Saves accompanying screenshots and generates thumbnails for each screenshot that go with the measurement.	HTTP	10101
		HTTPS	10102
Synthetic Feeder Server (Synthetic Hosted Agents)	• Deployed in SaaS. • Pushes screenshots and measurement results to the Synthetic Feeder-Client.	WebSocket (encrypted)	16001
Synthetic Feeder Client (Synthetic Hosted Agents)	• Deployed in on-premises. • Establishes a WebSocket connection with the SaaS Synthetic Server. • Sends the screenshots and measurement results it received from the SaaS Synthetic Server to the Synthetic Shepherd.	WebSocket (encrypted)	16101

Synthetic Agents	• Fetches jobs from the Synthetic Shepherd. • Uses WebDriver and Selenium to execute these jobs on browsers. • Registers with Synthetic Shepherd. • Uploads screenshots to Synthetic Shepherd. • Handles communication with Synthetic Shepherd.	N/A	The Synthetic Agents do not listen on any port. They only temporarily open internal random ports to fetch job requests from the Synthetic Shepherd and to send the measurement results of executed jobs to the Synthetic Shepherd.

Synthetic Service Data

The Synthetic Server stores data on the filesystem of the host machine and data in the EUM Server's MySQL database. The table below lists the types of data and the storage location.

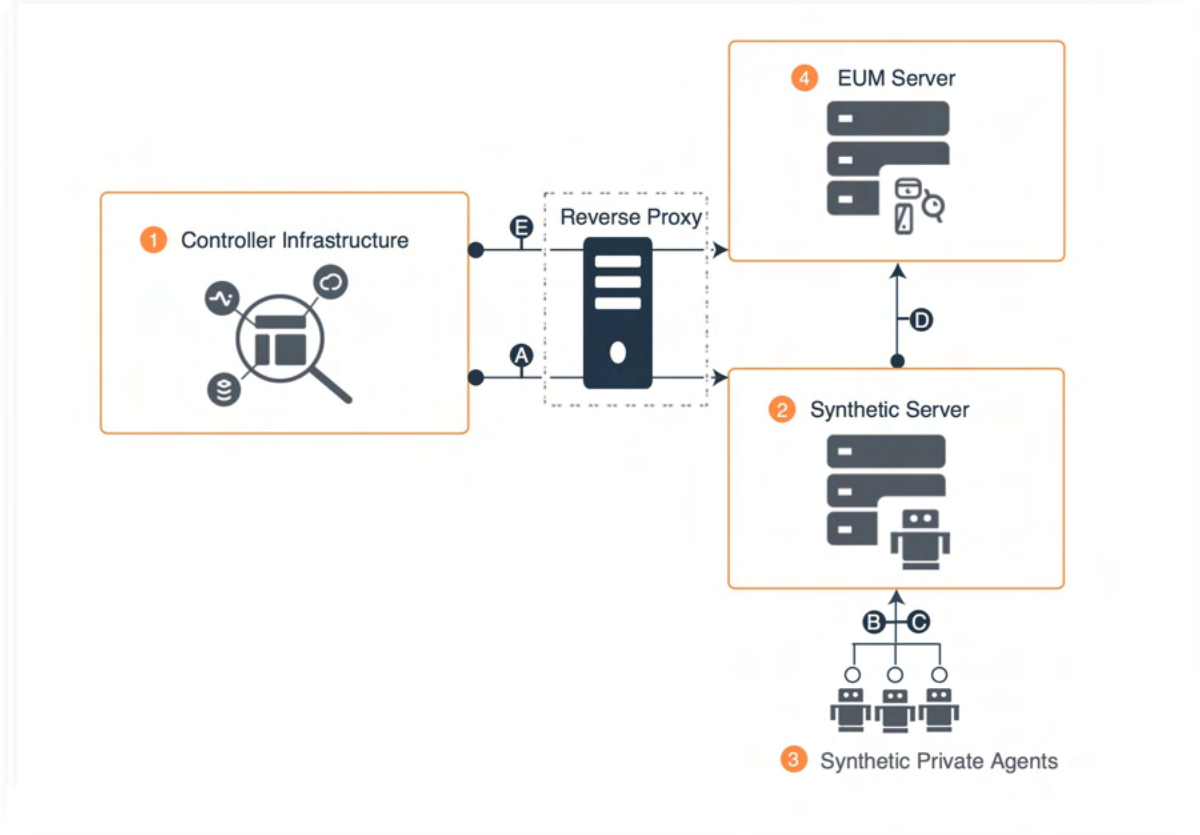
Data Storage Format	Data	Location of Data Storage
MySQL	• Information about the entire agent fleet. • Measurement request queues. • In-flight and archived measurements. • Schedules: a schedule is a configuration according to which the Synthetic Scheduler sends measurement requests to the Synthetic Shepherd. Those requests are queued until enough Synthetic Agents are available to process them, at which point they are dequeued and become measurements.	EUM Server's MySQL database
File System	• Resource snapshots • Script output • Measurement results • Screenshots	Host machine of the Synthetic Server

Synthetic Server Deployment Architecture

The following sections describe and provide diagrams of the different on-premises Synthetic Server deployments. The diagrams show the connections and data flow between the components of the deployments. For information about the other AppDynamics platform components, see Platform Components and Platform Connections.

Synthetic Private Agents Deployment

The following diagram shows the connections and data flow between the on-premises Synthetic Server and the EUM Server and Synthetic Private Agents.



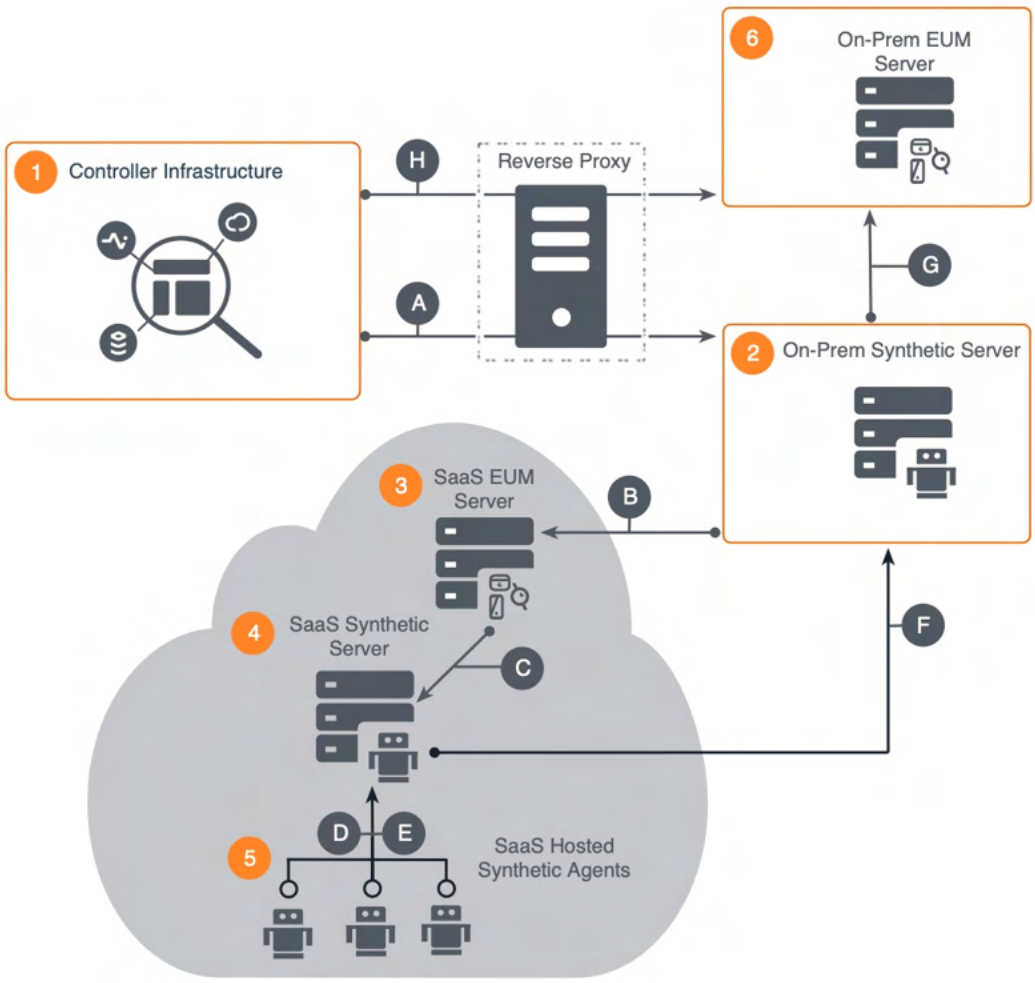
Synthetic Server Connections

The following table lists and describes the traffic flow between the Synthetic Server and the other components.

Connection	Source	Destination	Protocol	Default Port(s)
A	When a user creates a synthetic job, the 1 Controller sends a request for the job with its configured frequency to the 2 on-premises Synthetic Server. The synthetic jobs are then placed in a queue.	2 on-premises Synthetic Server	HTTP(S)	• 12101 /12102 • 10101 /10102
B	The 3 Synthetic Private Agent fetches the job requests from the Synthetic Server and then executes them on a browser using Selenium.	2 on-premises Synthetic Server	HTTP(S)	10101 /10102
C	The 3 Synthetic Private Agent then sends the measurement results to the Synthetic Server.	2 on-premises Synthetic Server	HTTP(S)	10101 /10102
D	The 2 on-premises Synthetic Server stores some data on file, and then processes and converts the data into a beacon, which is then transmitted to the 4 EUM Server through the EUM API. The Synthetic Server also writes data to the EUM Server's MySQL database.	4 on-premises EUM Server	HTTP(S) JDBC	7001/7002 3388
E	The 1 Controller polls the 4 EUM Server for the measurement results and displays them in the Synthetic Sessions.	4 on-premises EUM Server	HTTP(S)	7001/7002

Synthetic Hosted Agents Deployment

The following diagram shows the connections and data flow between the on-premises Synthetic Server, the SaaS EUM Server, the SaaS Synthetic Server, the Synthetic Hosted Agents, and the on-premises EUM Server.



Synthetic Server Connections

The following table lists and describes the traffic flow between the Synthetic Server and the other components.

Connection	Source	Destination	Protocol	Default Port(s)
A	When a user creates a synthetic job, the 1 Controller sends a request for the job with its configured frequency to the 2 on-premises Synthetic Server. The job requests are then placed in a queue.	2 on-premises Synthetic Server	HTTP(S)	12101 /12102
B	The 2 on-premises Synthetic Server sends the job requests to the 3 SaaS EUM Server.	3 SaaS EUM Server	HTTP(S)	7001/7002
C	The 3 SaaS EUM Server forwards the requests to the 4 SaaS Synthetic Server.	4 SaaS Synthetic Server	HTTP(S)	10001/10002

D	The 5 Synthetic Hosted Agents fetch the job requests from the 4 SaaS Synthetic Server and then executes them on a browser using Selenium.	4 SaaS Synthetic Server	WebSocket (encrypted)	16001
E	The 5 Synthetic Hosted Agents send the measurement results to the 4 SaaS Synthetic Server.	4 SaaS Synthetic Server	HTTP(S)	10001/10002
F	The 4 SaaS Synthetic Server Feeder sends the measurement results to the 2 on-premises Synthetic Client Feeder.	2 on-prem Synthetic Server	WebSocket (encrypted)	16101
G	The 2 on-premises Synthetic Server stores some data on file, and then processes and converts the data into a beacon, which is then transmitted to the 6 on-premises EUM Server through the EUM API. The on-prem Synthetic Server also writes data to the EUM Server's MySQL database.	6 on-premises EUM Server	HTTP(S) JDBC	7001/7002 3388
H	The Controller polls the 6 on-premises EUM Server for the measurement results and displays them in the Synthetic Sessions.	6 on-premises EUM Server	HTTP(S)	7001/7002

Synthetic Server Requirements

On this page:

This page lists the Synthetic Server requirements, offers sizing guidance, and shows you how to modify the default settings.

AppDynamics Platform Requirements

To deploy the Synthetic Server, you need to install the following AppDynamics platforms:

Component	Minimum Version
Controller	4.5.0 and higher
Events Service	4.5.0 and higher
Synthetic Agent	• Synthetic Private Agent 4.5.4 or higher • Synthetic Hosted Agent 4.5.13 or higher

Certain Synthetic Server features—specifically, Synthetic Sessions Analytics, features of Application Analytics that extend the functionality of Synthetic Sessions—require access to the AppDynamics Events Service.

Synthetic Agent Requirements

The following table lists the requirements for deploying Synthetic Private Agents and Synthetic Hosted Agents.

Synthetic Agent	Requirements
Synthetic Private Agents	See Requirements for the Synthetic Private Agent.
Synthetic Hosted Agents	• Synthetic Hosted Agent license • AppDynamics Access (HMAC) Key (part of the license file for Synthetic Hosted Agent)

Hardware Requirements

These requirements assume that the Synthetic Server is installed on a separate machine. If other AppDynamics platforms are installed on the same machine, the requirements (particularly for memory) could vary greatly and require many more resources.

- Storage: 50 GB free disk space
- Memory: 8 GB memory
- CPU: 64-bit CPU with at least 2 cores
- Network bandwidth: 50 Mbps

NTP should be enabled on both the EUM Server host and the Controller machine. The machine clocks need to be able to synchronize.

Scaling Requirements

You are required to have one EUM account for each on-premises deployment of the Synthetic Server. The machine hosting the Synthetic Server should be able to support 100 concurrent Synthetic Agents or 10 locations with 10 Synthetic Agents per location.

If you need the Synthetic Server to support more than 100 concurrent Synthetic Agents, see Increase the Synthetic Agent Support.

Operating System Support

The Synthetic Server is supported on the following operating systems:

Linux (64 bit)
• RHEL 6.x and 7.x • CentOS 6 and 7 • Ubuntu 14 and 16 • SUSE 12

You can use the following file systems for machines that run Linux:

- ZFS
- EXT4
- XFS

On-premises deployments on Linux are only supported on Intel architecture. Windows is not supported at this time.

Network Requirements

The network settings on the operating system need to be tuned for high-performance data transfers. Incorrectly tuned network settings can manifest themselves as stability issues.

The following command listing demonstrates tuning suggestions for Linux operating systems. As shown, AppDynamics recommends a TCP/FIN timeout setting of 10 seconds (the default is typically 60), the TCP connection `keepalive` time to 1800 seconds (reduced from 7200, typically), and disabling TCP window scale, TCP SACK, and TCP timestamps.

```
echo 5 > /proc/sys/net/ipv4/tcp_fin_timeout
echo 1800 >/proc/sys/net/ipv4/tcp_keepalive_time
echo 0 >/proc/sys/net/ipv4/tcp_window_scaling
echo 0 >/proc/sys/net/ipv4/tcp_sack
echo 0 >/proc/sys/net/ipv4/tcp_timestamps
```

The commands demonstrate how to configure the network settings in the `/proc` system. To ensure the settings persist across system reboots, be sure to configure the equivalent settings in the `etc/sysctl.conf` or the network stack configuration file appropriate for your operating system.

Software Requirements

The Synthetic Server requires the following software to run and function correctly. You are *required* to have outbound internet access to install Python, `pip`, and flake8.

Software	Required Version	Function
Java	8	The Synthetic Server requires JDK 8 to run services such as Synthetic Scheduler and Synthetic Shepherd. You need to set the environmental variable `JAVA_HOME` to the home directory of the JDK.
Python	2.7	The Synthetic Server relies on Python to validate scripts.

`pip`	9+	Python uses `pip` to install software. For example, `pip` could be used on some Linux distributions to install `flake8`, a Python utility used to lint scripts.
		
		If the machine where you're installing the Synthetic Server does *not* have Internet access, run the following steps to fetch and install `flake8`:
		
		1. From a machine with internet access and `pip` installed:
		a. Create a directory for the `flake8` library:
		`mkdir ~/flake8`
		b. Download the `flake8` package:
		c. Zip and tar the `flake8` package:
		`tar cvfz flake8.tgz ~/flake8`
		d. Copy `flake8.tgz` to the `$HOME` directory of the host machine of the Synthetic Server.
		2. From the host of the Synthetic Server that has no internet access, but does have `pip` installed:
		a. Unzip and extract the `flake8.tgz` file:
		`tar xvfz flake8.tgz ~/flake8`
		b. Change to the `flake8` directory.
		c. Install the `flake8` library with `pip` with the following command, replacing `<version>` with the correct version.

libaio	N/A	The Synthetic Server requires the `libaio` library to be on the system. This library facilitates asynchronous I/O operations on the system. See How to Install libaio for instructions.

How to Install libaio

Install `libaio` on the host machine if it does not already have it installed. You may require outbound internet access if you don't have a locally hosted repository.

The following table provides instructions on how to install `libaio` for some common flavors of the Linux operating system. Note, if you have a NUMA based architecture, then you are required to install the `numactl` package.

Linux Flavor	Command
Red Hat and CentOS	Use `yum` to install the library, such as: • `yum install libaio` • `yum install numactl`
Fedora	Install the library RPM from the Fedora website: • `yum install libaio` • `yum install numactl`
Ubuntu	Use apt-get, such as: • `sudo apt-get install libaio1` • `sudo apt-get install numactl`
Debian	Use a package manager such as APT to install the library (as described for the Ubuntu instructions above).

Install the Synthetic Server

You run the Synthetic Server installer from the command line. The installer relies on the`inputs.groovy` file to configure the network connections to the on-premises EUM Server, and if you are using Synthetic Hosted Agents, to the SaaS EUM Server and SaaS Synthetic Server as well.

To install the Synthetic Server, follow the steps below:

Prepare for the Installation

Before starting the installation, verify that you have:

- Successfully deployed the Controller, EUM Server, and the Events Service.
- Downloaded the Synthetic Server installer package from the AppDynamics Download Center. The installer package will be listed on the Downloads site as "Synthetic Server (zip)".

Grant Access to the EUM Server MySQL Database

The Synthetic Server installer modifies the EUM MySQL database schema and the Synthetic Server stores data in the EUM MySQL database. Thus, you will need to grant MySQL users from the machine hosting the Synthetic Server privilege to the EUM Server's MySQL database.

1. Log on to the machine where the EUM MySQL database is located.
2. Connect to the MySQL Server with the EUM Server database. For example, if you are using the default EUM MySQL database, do the following:
    a. Change to `<installDir>/AppDynamics/EUM`.
    b. Connect to the EUM MySQL database:

```
mysql/bin/mysql -u root -h <eum_server_hostname> -S <eum_server_mysql_sock> -P
<eum_server_mysql_port> -p
```

3. From the MySQL monitor, grant privileges to the MySQL user `root` of the Synthetic Server machine. The installer will use the MySQL user `root` t o update the EUM database schema. Be sure to replace `<on-prem-synthetic_server_hostname>` with the URL to your Synthetic Server.

```
mysql> GRANT ALL PRIVILEGES ON eum_db.* TO 'root'@'<on-prem_synthetic_server_hostname>' IDENTIFIED BY
<db-root-password>;;
```

> The MySQL `root` user from the Synthetic Server is not related to the Linux user account that is installing the Synthetic Server. For example, the Linux user account `ubuntu` can run the installer, but the installer will use the MySQL user `root` when connecting to the EUM Server MySQL database to update the database schema.

4. You will also need to grant access to the MySQL user `eum_user` to write data to the EUM database (`eum_db`). Be sure to replace `<on-prem-synthetic_server_hostname>` with the URL to your Synthetic Server.

```
mysql> GRANT ALL PRIVILEGES ON eum_db.* TO 'eum_user'@'<on-prem_synthetic_server_hostname>' IDENTIFIED
BY <db_eum_user_password>;
```

5. Set the password for the MySQL user `root.` The password should be the same as the one specified by the `db_root_pwd` in the `inputs. groovy` file.

```
mysql> SET PASSWORD FOR 'root'@'<on-prem-synthetic_server_hostname>' = PASSWORD('<root_password>');
```

6. Confirm that you have granted permission for `eum_user` and `root`:

```
show grants for eum_user@<on-prem_synthetic_server_hostname>;
show grants for root@<on-prem_synthetic_server_hostname>;
```

Unzip the Synthetic Server Installer Package

1. Copy the Synthetic Server installer package (`appdynamics-synthetic-server-<version>.zip`) to the machine that will be hosting the Synthetic Server.
2. Create a directory for storing the Synthetic Server installer, such as `synthetic-server`.
3. Move the Synthetic Server installer package to the directory you created.
4. Change to the directory you created for storing the Synthetic Server installer.
5. Unzip the Synthetic Server installer package.

Configure the Installation

1. From a command prompt, navigate to the directory where you unzipped the Synthetic Server installer package.
2. Copy the sample configuration file: `cp inputs.groovy.sample inputs.groovy`
3. Edit the file `inputs.groovy` and make changes to the properties listed below:

Property	Change to Make	Description
`db_host`	Assign the URL to the machine hosting your on-premises EUM Server to the `db_host` property.	The public DNS to the machine hosting the EUM Server.
`db_port`	Change the value to "3388". This is the default port for the EUM Server's MySQL database.	The port that the EUM Server's MySQL database is listening on.
`db_username`	Change the value to `eum_user`. This is the default user for the EUM Server.	The MySQL user that accesses the EUM Server's MySQL database.
`db_password`	Assign the password for the MySQL user `eum_user` to remotely access the EUM Server's MySQL database.	The password that you set for the user that is specified by `db_username`. The value of `db_username` should be `eum_user`.
`collector_host`	Assign the public DNS to the machine hosting your on-premises EUM Server to the `collector_host` property.	The public DNS to the machine hosting the EUM Server.
`collector_port`	Confirm that the value is "7001". This is the default port of the EUM Server.	The port that the EUM API Server and EUM Collector are listening on. The default is '7001'.
`key_store_password`	Assign the key store password you set when installing your on-premises EUM Server to the `key_store_password` property.	The key store password you set during the installation of the EUM Server.
`localFileStoreRootPath`	Assign a file path where you want the Synthetic Server to store data to the `localFileStoreRootPath` property. The Synthetic Server must be able to read and write to the path and the files in the path.	The path where the Synthetic Server stores data such as the measurement results and the screenshots.

Run the Installer

From the root directory of the installer, run the following command as the `root` user.

```
unix/deploy.sh install
```

In the output from the `install` command, you should see the log of completed tasks similar to the following:

```
Task [facts for localhost] completed executing in [274] ms.
Task [Create the encryption directory] completed executing in [78] ms.
Task [Create keystore for encryption] completed executing in [796] ms.
Task [Create the encrypted password] completed executing in [566] ms.
Task [Obfuscate the key store password] completed executing in [397] ms.
Task [Read created password] completed executing in [46] ms.
Task [Read the obfuscated key store password] completed executing in [43] ms.
Task [Change configurations for the shepherd and scheduler conf] completed executing in [81] ms.
Task [Read created password] completed executing in [24] ms.
Task [Read the obfuscated key store password] completed executing in [26] ms.
Task [Change configurations for the shepherd and scheduler conf] completed executing in [76] ms.
Task [Read created password] completed executing in [29] ms.
Task [Read the obfuscated key store password] completed executing in [20] ms.
Task [Change configurations for the shepherd and scheduler conf] completed executing in [31] ms.
Task [Delete the encryption directory] completed executing in [47] ms.
Task [Change configurations for the liquibase properties file] completed executing in [26] ms.
Task [Update schema of SQL DB to include synthetic schema] completed executing in [2412] ms.
Task [Install flake8 for script linting] completed executing in [1671] ms.
Task [Start the synthetic services] completed executing in [67] ms.
```

Verify the Installation Was Successful

1. Confirm that the Synthetic Server is running:

   ```
   ps aux | grep synthetic-processor
   ```

   > If you have `jps` installed, you can also just run it to verify the Synthetic Server are running.

2. Verify that the Synthetic Scheduler and Synthetic Shepherd are listening on the default ports:

   ```
   netstat -lan | grep "1[0,2,6]10[1,2]"
   ```

3. With `mysql` installed on the Synthetic Server machine, you can verify that the Synthetic Server machine can connect to the EUM Server MySQL e`um_db` database:

   ```
   mysql -h <eum_server_instance> -P 3338 -D eum_db -u eum_user -p
   ```

4. If you cannot connect to the EUM Server MySQL database, return to Grant Privileges to the EUM Server MySQL Database and complete the steps again.

Perform Post-installation Tasks

After installing the Synthetic Server, perform the following additional post-installation tasks:

1. Configure the Controller for the Synthetic Server
2. Install Synthetic Private Agents (Optional)
3. Make configurations to use one or both of the Synthetic Agents:
   a. Synthetic Private Agent
   b. Synthetic Hosted Agent
4. Secure the Synthetic Server (Recommended)
5. Monitor the Synthetic Server (Recommended)

Configure the Controller for the Synthetic Server

Related pages:

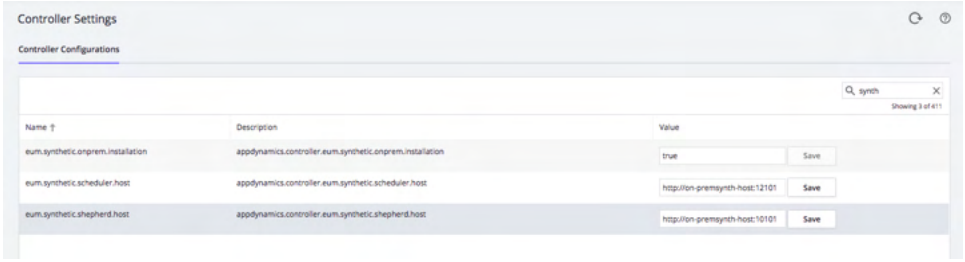
- Access the Administration Console
- Configure the EUM Server

For the Synthetic Server to function correctly, you need to set the URLs and ports of Synthetic Scheduler and Synthetic Shepherd in the Controller Administration Console.

1. Navigate to the Controller Administration Console: `http://<hostname>:<port>/controller/admin.jsp`
2. Click **Controller Settings**.
3. From the **Controller Configurations** pane, enter the correct values for the properties given in the table below. If no protocol is specified, the protocol defaults to `https://`.

Controller Configuration	Description	Example Value
`eum.synthetic.onprem.installation`	The flag for enabling on-premises Synthetic Server. This should be `true`.	`true`
`eum.synthetic.scheduler.host`	The URL and port to the Synthetic Scheduler on the machine hosting the Synthetic Server. The default port is 12101.	`http://<synthetic-server-domain>:12101`
`eum.synthetic.shepherd.host`	The URL and port to the Synthetic Shepherd on the machine hosting the Synthetic Server. The default port is 10101.	`http://<synthetic-server-domain>:10101`

4. Your configurations for the Synthetic Server should be similar to those in the screenshot of the **Controller Configurations** pane below.



Connect the Synthetic Private Agents to the Synthetic Server

The information and instructions below are intended for On-Premise deployments only. SaaS deployments are managed by AppDynamics.

Related pages:

- Install the Synthetic Private Agent

The Synthetic Private Agent fetches jobs from and reports measurement results to the Synthetic Shepherd service of the Synthetic Server. Thus, you must correctly configure the Synthetic Agent so it can connect to Synthetic Shepherd.

To connect the Synthetic Agent to on-prem Synthetic Server, follow the steps below:

Prepare for the Agent Configuration

Before you configure the connections:

- Confirm that you have installed the Synthetic Private Agent.
- Confirm that the EUM Server is running.
- Get the EUM account and the license key from the Controller Admin Console or the EUM Server properties file.

Configure the Synthetic Private Agent

1. Stop the Synthetic Private Agent if it's running by double-clicking the desktop icon 

.
2. Change to the directory `C:\appdynamics\synthetic-agent\synthetic-driver\conf`.
3. Edit the file `synthetic-driver.yml`.
4. Assign your EUM account and the license key to the properties `eumAccount` and `licenseKey`, and the URL to Synthetic Shepherd to `shepherdUrl` as shown below:

```
## Use the URL to your Synthetic Server and the port to the Synthetic Shepherd (10101)
shepherdUrl: http://<on-prem-synthetic-server-host>:10101
## You can get the values for this from the Controller Admin Console > Controller Settings
## or the properties 'property_eum-account-name' and 'property_eum-license-key' from your license file.
privateClient:
    eumAccount: "<eum_account>"
    licenseKey: "<license_key>"
```

5. Save the file.
6. Restart the Synthetic Private Agent by double-clicking the desktop icon 

.

Verify the Private Location in the Controller

Follow the instructions given in Create a Job and Choose Locations to create a synthetic job using the private location where your Synthetic Private Agent is running.

Connect to the SaaS Environment for the Synthetic Hosted Agent

Related pages:

- Synthetic Agent Locations

For your on-premises Synthetic Server to use Synthetic Hosted Agents, you need to configure the on-premises Synthetic Server to communicate with the SaaS EUM Server and SaaS Synthetic Server.

Follow these steps to use Synthetic Hosted Agents:

1 Verify the License Has an HMAC Key
2 Apply the License
3 Configure the Connection to the SaaS EUM and Synthetic Servers

For a complete list of Synthetic Hosted Agent browser locations, containers, and providers, go to Synthetic Agent Locations.

Verify the License Has an HMAC Key

After you have obtained a license to use the Synthetic Hosted Agent, be sure to check that the `license.lic` file has the `property_eum-hmac-key` field that is assigned a keyed-hash message authentication code (HMAC) similar to the following:

```
property_eum-hmac-key=1a88392cb0004b45b555a854b80f23f5
```

The HMAC key is used to authenticate your on-prem deployment to the SaaS EUM Server and the SaaS Synthetic Server. Without the HMAC key, your on-premises Synthetic Server will not be able to use Synthetic Hosted Agents.

Apply the License

To apply the license:

1. Copy the license file to the Controller home directory. After moving the license file, allow up to 5 minutes for the license change to take effect.
2. Follow the instructions given in Provision EUM Licenses based on your deployment.

Configure the Connection to the SaaS EUM and Synthetic Servers

Configure the connections from the on-prem Synthetic Server to the SaaS EUM and Synthetic Servers based on the region where you have an EUM account.

SaaS EUM Account in the Americas

If your SaaS EUM account is in the Americas, do not change the following default settings given in `inputs.groovy`:

```
feeder_server_url = "wss://synthetic-feeder.api.appdynamics.com"
saas_cloud_api_url = "https://api.eum-appdynamics.com"
```

SaaS EUM Account in Other Regions

If your SaaS EUM account is not in the Americas, follow these instructions to configure your on-premises Synthetic Server to use the SaaS EUM Server and SaaS Synthetic Server in your region.

1. From the on-premises Synthetic Server, edit the `inputs.groovy` file.
2. For the `feeder_server_url` and `saas_cloud_api_url` fields, enter the SaaS URLs for your region.

Administer the Synthetic Server

You can use the command line to perform platform administration tasks with the Synthetic Server, such as starting and stopping the services. This page describes the available commands, the log files, and the endpoints for testing the reachability and health of the Synthetic Server. Run the commands from the root directory of the Synthetic Server home.

Start and Stop the Synthetic Server

Start the Synthetic Server from the root directory of the Synthetic Server home as follows:

```
unix/deploy.sh start
```

To check if the Synthetic Server services are running and accessible, run the following command and confirm that there is output:

```
netstat -lan | grep "1[0,2,6]10[1,2]"
```

To stop the Synthetic Server:

```
unix/deploy.sh stop
```

Increase the Synthetic Agent Support

By default, the machine hosting the Synthetic Server should be able to support 100 concurrent Synthetic Agents. To support more than 100 concurrent Synthetic Agents, modify the throttle configuring for the Synthetic Shepherd and Synthetic Scheduler. The default maximum number of requests per second is 60. By increasing the maximum number of requests per second, the Synthetic Server can support more Synthetic Agents.

To increase the maximum number of requests per second that the Synthetic Server can receive:

1. Log on to the machine hosting the Synthetic Server.
2. Change to the root directory of the Synthetic Server home.
3. Edit the file `synthetic-processor/conf/synthetic-shepherd.yml` and increase the value for the property `maxRequestsPerSecondOverall`. In this example configuration, the value is increased to 80.

   ```
   throttleConfiguration:
       maxRequestsPerSecondOverall: 80
   ```

4. Edit the file `synthetic-processor/conf/synthetic-scheduler.yml` and increase the value for the property `maxRequestsPerSecondOverall`. Again, in this example configuration, the value is increased to 80.

   ```
   throttleConfiguration:
       maxRequestsPerSecondOverall: 80
   ```

5. Restart the Synthetic Server:

   ```
   unix/deploy.sh stop
   unix/deploy.sh start
   ```

6. You can verify that the settings have been updated by checking the logs for the Synthetic Server you changed:

   ```
   cat logs/scheduler/synthetic-scheduler.log | grep -oP -- "maxRequestsPerSecondOverall=\d+"
   cat logs/shepherd/synthetic-shepherd.log | grep -oP -- "maxRequestsPerSecondOverall=\d+"
   ```

7. You should also check the health of the Synthetic Server.

Create Preset Health Rules and Dashboards

Once you have monitored the Synthetic Server, run the following command to create the preset health rules and dashboards:

```
unix/post_deploy.sh
```

See Create Preset Dashboards and Health Rules to learn more about the preset health rules and dashboards.

Upgrade the Synthetic Server

You can update the Synthetic Server and the database schema without uninstalling and reinstalling the Synthetic Server using the `update` command. See Upgrade the Synthetic Server for instructions.

Check the Health of the Synthetic Server

To check if the Synthetic Server is running, you can run the following. You should receive the response `pong`.

```
curl <on-prem-synthetic_server_url>:10102/ping
curl <on-prem-synthetic_server_url>:12102/ping
curl <on-prem-synthetic_server_url>:16102/ping
```

To check the health of the Synthetic Server:

```
curl <on-prem-synthetic_server_url>:10102/healthcheck?pretty=true
curl <on-prem-synthetic_server_url>:12102/healthcheck?pretty=true
curl <on-prem-synthetic_server_url>:16102/healthcheck?pretty=true
```

If the Synthetic Server is healthy, the response should be similar to the following:

curl <on-prem-synthetic_server_url>:10102/healthcheck?pretty=true

```
{
  "authentication" : {
    "healthy" : true
  },
  "deadlocks" : {
    "healthy" : true
  },
  "httpClient" : {
    "healthy" : true
  },
  "quartzScheduler" : {
    "healthy" : true
  }
```

curl <on-prem-synthetic_server_url>:16102/healthcheck?pretty=true

```
{
  "deadlocks" : {
    "healthy" : true
  }
```

curl <on-prem-synthetic_server_url>:12102/healthcheck?pretty=true

```
{
  "authentication" : {
    "healthy" : true
  },
  "cluster" : {
    "healthy" : true
  },
  "deadlocks" : {
    "healthy" : true
  },
  "linter" : {
    "healthy" : true
  },
  "quartzSynthBackgroundScheduler" : {
    "healthy" : true
  },
  "quartzSynthJobScheduler" : {
    "healthy" : true
  }
```

Log Files for the Synthetic Server

The Synthetic Server creates the following error log files for each service:

- `<installDir>/logs/synthetic-scheduler.err`
- `<installDir>/logs/synthetic-shepherd.err`

- `<installDir>/logs/synthetic-feeder-client.err`

For general (non-error) log files, see the following directories:

- `<installDir>/logs/scheduler`
- `<installDir>/logs/shepherd`
- `<installDir>/logs/feeder-client`

The naming convention for the general log files is `<log>-YYYY-MM-DD.log`. You will need to set up a policy to archive or delete the general log files to prevent running out of disk space.

Secure the Synthetic Server

You are recommended to configure the Synthetic Server to use SSL to secure network connections. This page describes how to create a custom keystore and then configure the Synthetic Server to use it to implement SSL

Set Up a Custom Keystore for the Synthetic Server

The following sections describe and show an example of how to create a custom RSA security certificate, generate a new JKS keystore, and sign the certificate.

Install Prerequisite Libraries

Make sure the following libraries are installed on the Synthetic Server:

- `keytool`
- `openssl`

Create a Certificate and Keystore

Use the `keytool` command to create a keystore that uses RSA encryption then generate a certificate signing request (CSR).

The following steps show you an example of how to do both.

1. Log in to the Synthetic Server machine.
2. From a command-line shell, navigate to the root directory of the Synthetic Server:

   ```
   cd <synthetic_server_root>
   ```

3. Create a new keystore with a new unique key pair that uses RSA encryption:

   ```
   <path_to_jre>/jre/bin/keytool -genkey -keyalg RSA -validity <validity_in_days> -alias 'synthetic-server'
   -keystore ./mycustom.keystore
   ```

   This creates a new public-private key pair with an alias of "synthetic-server". You can use any value you like for the alias. The "first and last name" required during the installation process becomes the common name (CN) of the certificate. Use the name of the server.
4. Configure the keystore by entering the information requested at the command prompt.
5. Specify a password for the key store. You need to configure this password in the Synthetic Server configuration file later.
6. Generate a certificate signing request (CSR):

   ```
   <path_to_jre>/jre/bin/keytool -certreq -keystore ./mycustom.keystore -file /tmp/synthetic-server.csr -
   alias 'synthetic-server'
   ```

   This generates a certificate signing request based on the contents of the alias; in the example, it is "synthetic-server".

Sign and Install the Signed Certificate

Once you have a CSR, you request a Certificate Authority to sign it and then install the signed certificate.

The following steps are a continuation of the process from Create a Certificate and Keystore:

1. Send the output file from the last step (`/tmp/synthetic-server.csr` in this example) to a Certificate Authority for signing.
2. Install the certificate for the Certificate Authority used to sign the `.csr` file:

   ```
   <path_to_jre>/jre/bin/keytool -import -trustcacerts -alias myorg-rootca -keystore ./mycustom.keystore -
   file /path/to/<CA-root-cert>
   ```

   This command imports your CA's root certificate into the keystore and stores it in an alias called "myorg-rootca".
3. Install the signed server certificate as follows:

```
<path_to_jre>/jre/bin/keytool -import -keystore ./mycustom.keystore -file /path/to/<signed-cert>  -alias
'synthetic-server'
```

This command imports your signed certificate over the top of the self-signed certificate in the existing alias; in the example, it is "synthetic-server".

4. Import the root certificate to the other platform components connecting to the Synthetic Server through HTTPS:

```
keytool -import -trustcacerts -alias <alias_name> -file mycert.cer -keystore <complete_path_to_cacerts.
jks>
```

Configure the Synthetic Server to Use the Keystore

Follow the steps below to configure the Synthetic Server to use the signed certificate and its password.

1. Edit the Synthetic Scheduler configuration file at `<installation directory>/conf/synthetic-scheduler.yml` and add the `applicationConnectors` object shown below under `server`:

```
server:
    ...
    applicationConnectors:
        - type: https
          port: <port>
          keyStorePath: <path to JKS files>
          keyStorePassword: <jks file password>
          validateCerts: false
```

If you don't already have a signed certificate, see Create and Sign an RSA Security Certificate.

2. Edit the Synthetic Shepherd configuration file at `<installation directory>/conf/synthetic-shepherd.yml` and add the `applicationConnectors` object shown below under `server`:

```
server:
    ...
    applicationConnectors:
        - type: https
          port: <port>
          keyStorePath: <path to jks file>
          keyStorePassword: <jks file password>
          validateCerts: false
```

3. Restart the Synthetic Server.
4. Verify the connection to the HTTPS port.

Configure a Proxy for the Synthetic Server

Related pages:

- Use a Reverse Proxy

This page describes how to configure your on-prem Synthetic Server to use a proxy server to communicate with the SaaS EUM Server and SaaS Synthetic Server. You can set up a proxy to add a security layer for your on-prem Synthetic Server.

The configuration consists of the following steps:

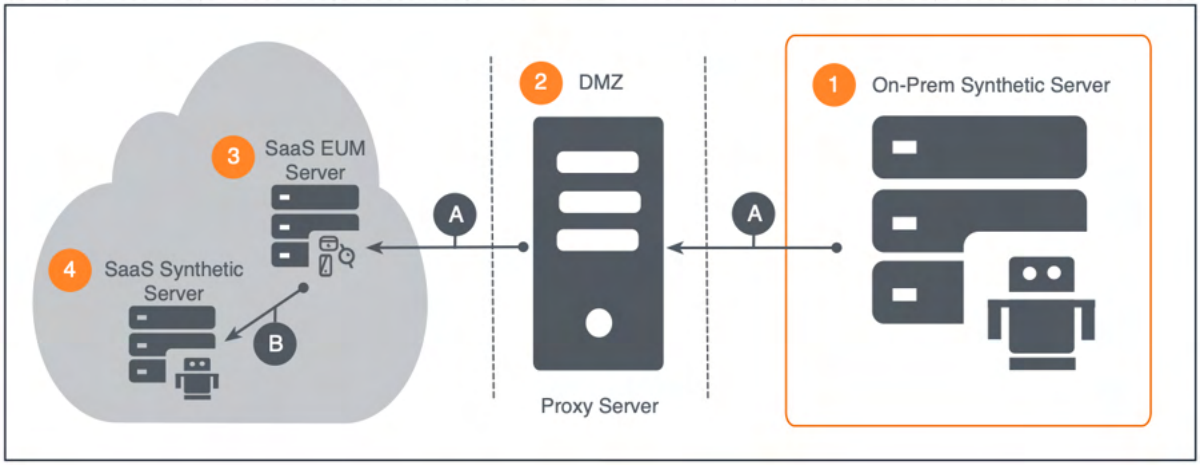
1 Configure the Synthetic Server to Use Proxy
2 Configure Your Proxy Server

Synthetic Server Proxy Architecture

You can also, optionally, set up a proxy to forward traffic from the SaaS Synthetic Server Feeder to the on-prem Synthetic Server Client Feeder, but your proxy server must support WebSockets.

Proxy for Downstream Traffic (Recommended)

The proxy often resides in the DMZ for the network and presents a virtual IP address to forward requests from the on-prem Synthetic Server to SaaS Synthetic Server through the SaaS EUM Server as shown below.

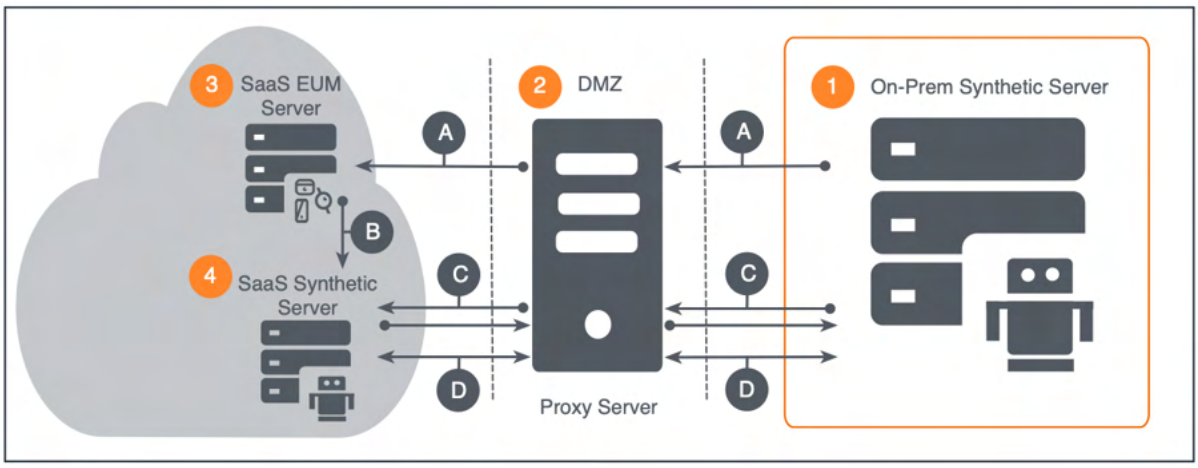


Connection	Source	Destination	Protocol	Default Port(s)
A	The 1 on-prem Synthetic Scheduler sends the job requests to the 2 proxy server, which then forwards the requests to the 3 SaaS EUM Server	3 SaaS EUM Server	HTTP(S)	7001/7002
B	The 3 SaaS EUM Server forwards the job requests to the 4 SaaS Synthetic Server.	4 SaaS Synthetic Server	HTTP(S)	N/A

The on-prem Synthetic Server Client Feeder communicates with the SaaS Synthetic Server Feeder through a bi-directional WebSocket connection that is not made through the proxy server.

Proxy for Downstream and Upstream (Optional)

When setting a proxy for downstream and upstream traffic, the proxy must support WebSocket. The on-prem Synthetic Server Client Feeder initializes the WebSocket connection with the SaaS Synthetic Server Feeder through the proxy server. Once the WebSocket connection is established, the on-prem Synthetic Server Client Feeder and the SaaS Synthetic Server Feeder can use the persistent connect to make bi-directional requests.



Connection	Source	Destination	Protocol	Default Port(s)
A	The 1 on-prem Synthetic Server (Scheduler) sends the job requests to the 2 proxy server, when then forwards the requests to the 3 SaaS EUM Server.	3 SaaS EUM Server	HTTP(S)	7001/7002
B	The 3 SaaS EUM Server forwards the requests to the 4 SaaS Synthetic Server.	4 SaaS Synthetic Server	HTTP(S)	N/A
C	The 1 on-prem Synthetic Server (Client Feeder) establishes a WebSocket connection with the 4 SaaS Synthetic Server (Synthetic Server Feeder) through the 2 proxy server.	4 SaaS Synthetic Server	HTTP(S)	80/443
		1 on-prem Synthetic Server	HTTP(S)	80/443
D	The 4 SaaS Synthetic Server (Synthetic Server Feeder) and 1 on-prem Synthetic Server (Client Feeder) communicate bi-directionally through the 2 proxy server. Most of the traffic is from the SaaS Synthetic Server Feeder to the on-prem Synthetic Client Feeder.	1 on-prem Synthetic Server	WebSocket	16101
		4 SaaS Synthetic Server	WebSocket	16001

Configure the Synthetic Server to Use Proxy

You need to configure the Synthetic Server to use the proxy to forward traffic to the SaaS EUM Server. In the examples configurations below, the proxy URL is `127.0.0.1` and the proxy port is `3128`. You will need to replace those values with the URL and port of your proxy server.

Synthetic Shepherd

From the host machine of the Synthetic Server, set the `proxyUrl` and `proxyPort` properties in the `<synthetic-server_installation_dir>/synthetic-processor/conf/synthetic-shepherd.yml` file to point to the URL and port of the proxy server, which the Synthetic Shepherd will send requests.

```
saasLink:
  proxyUrl: "127.0.0.1"
  proxyPort: 3128
```

Synthetic Scheduler

Set the `proxyUrl` and `proxyPort` properties in the `<synthetic-server_installation_dir>/synthetic-processor/conf/synthetic-scheduler.yml` file to point to the URL and port of the proxy server so that Synthetic Shepherd also sends requests to the proxy.

```
saasLink:
  proxyUrl: "127.0.0.1"
  proxyPort: 3128
```

Synthetic Feeder Client

Finally, set the `proxyUrl` and `proxyPort` properties in the `<synthetic-server_installation_dir>/synthetic-processor/conf/synthetic-feeder.yml` file to point to the URL and port of the proxy server so that Synthetic Server Client Feeder also sends requests to the proxy.

```
websocketConfiguration:
    proxyUrl: "127.0.0.1"
    proxyPort: 3128
```

> **Note**
>
> Feeder Client for Synthetic services does not support Proxy authentication.

Configure Your Proxy Server

You can use Squid, Apache, Nginx, or another proxy server, but these instructions only cover Squid. If you are using a proxy to forward traffic between the on-prem Synthetic Server Client Feeder and the Synthetic Server Feeder, your proxy will need to support the WebSocket protocol.

Squid

1. Add the following configurations to the Squid configuration file at `/etc/squid/squid.conf`.

   ```
   http_access allow localhost
   http_access allow all
   http_port 3128
   ```

2. Restart `squid`.

Monitor the Synthetic Server

AppDynamics recommends that you monitor the performance of the Synthetic Server with the Java Agent. Once you have instrumented the Synthetic Server, you can use preset capacity monitoring dashboards and health rules or create custom dashboards and health rules based on JMX and the Synthetic Server metrics.

Follow the steps below to monitor the Synthetic Server:

1 Install the Java Agent
2 Configure the Java Agent
3 Connect the Synthetic Server with the Controller
4 Attach the Java Agent to the Synthetic Server
5 Verify the Instrumentation of the Synthetic Server
6 Create Preset Dashboards and Health Rules
7 Create Custom Dashboards and Health Rules

Install the Java Agent

You should install the Java Agent in the same directory as the Synthetic Server installation directory. See Install the Java Agent for instructions.

Configure the Java Agent

Configure the Java Agent to report metrics to a specific Controller:

1. Change to `<agent_home>/conf/`.
2. Edit the `controller-info.xml` file so that the values for the following elements match your Controller information, application name, tier name, and node name:
    - `<controller-host>`
    - `<controller-port>`
    - `<application-name>`
    - `<tier-name>`
    - `<node-name>`
3. For example, your `controller-info.xml` file might look similar to the following:

```
<controller-info>
    <controller-host>192.168.1.20</controller-host>
    <controller-port>8090</controller-port>
    <application-name>SyntheticServer</application-name>
    <tier-name>SchedulerTier</tier-name>
    <node-name>SchedulerNode</node-name>
</controller-info>
```

> You must ensure that the application name, tier name, and node name are the same as the `javaagent.jar` parameters when you attach the Java Agent to the Synthetic Server.

Connect the Synthetic Server with the Controller

Before you installed the Synthetic Server, you needed to configure the Synthetic Server to connect to the EUM Server's MySQL database and the EUM Collector. In this section, you will set configurations in `inputs.groovy` to connect the Synthetic Server to the Controller, so that the predefined health rules and dashboards can be created.

In the `inputs.groovy` file, make sure you have set the following properties. Replace placeholders in brackets with information about your Controller as well as the application and tier that are being monitored.

```
controller_host = "http(s)://<url_to_machine_running_controller>"     // The URL to your on-prem Controller
controller_port = "<port_number>"                                     // The default is 8090.
controller_account = "<controller_account>"                           // Account used for running post-deploy
tasks
controller_username = "<controller_username>"                         // Username for making API calls to
controller
prompt_for_password = "false"                                         // When false, the password below will be
used without prompting.
controller_password = "<controller_password>"                         // Password used for username. It is not
stored in any config files.
controller_synth_app = "<app_name_set_in_controller-info.xml>"        // This is the application shown in the
Controller and is based on the value given in the <application-name> element in controller-info.xml.
controller_shepherd_entity = "<tier_name_set_in_controller-info.xml>" // This is the tier shown in the
Controller and is based on the value given in the <tier-name> element in controller-info.xml
```

Attach the Java Agent to the Synthetic Server

To attach the Java Agent to the Synthetic Server, set Java options through the variables `SCHEDULER_OPTS` and `SYNTHETIC_SHEPHERD_OPTS`. The node names and tier names given in the examples below can be modified for your use case. One JVM process requires one Java Agent to be attached.

1. Set the options for the Synthetic Scheduler so that the Java Agent is attached to the JVM process:

```
SCHEDULER_OPTS="-javaagent:./java_agent/javaagent.jar -Dappdynamics.agent.applicationName=synthonprem -
Dappdynamics.agent.nodeName=synthetic-scheduler -Dappdynamics.agent.tierName=scheduler-tier"
```

2. Set the options for the Synthetic Shepherd so that the Java Agent is attached to the JVM process:

```
SYNTHETIC_SHEPHERD_OPTS="-javaagent:./java_agent/javaagent.jar -Dappdynamics.agent.
applicationName=synthonprem -Dappdynamics.agent.nodeName=synthetic-shepherd -Dappdynamics.agent.
tierName=shepherd-tier"
```

3. Set the options for the Synthetic Feeder Client so that the Java Agent is attached to the JVM process:

```
FEEDER_CLIENT_OPTS="-javaagent:./java_agent/javaagent.jar -Dappdynamics.agent.
applicationName=synthonprem -Dappdynamics.agent.nodeName=synthetic-feeder-client -Dappdynamics.agent.
tierName=feeder-client-tier"
```

4. Export the variables for the Synthetic Server options:

```
export SYNTHETIC_SHEPHERD_OPTS
export SCHEDULER_OPTS
export FEEDER_CLIENT_OPTS
```

5. From the Synthetic Server installer directory, run the following to stop and start the Synthetic Server:

```
unix/deploy.sh stop
unix/deploy.sh start
```

Verify the Instrumentation of the Synthetic Server

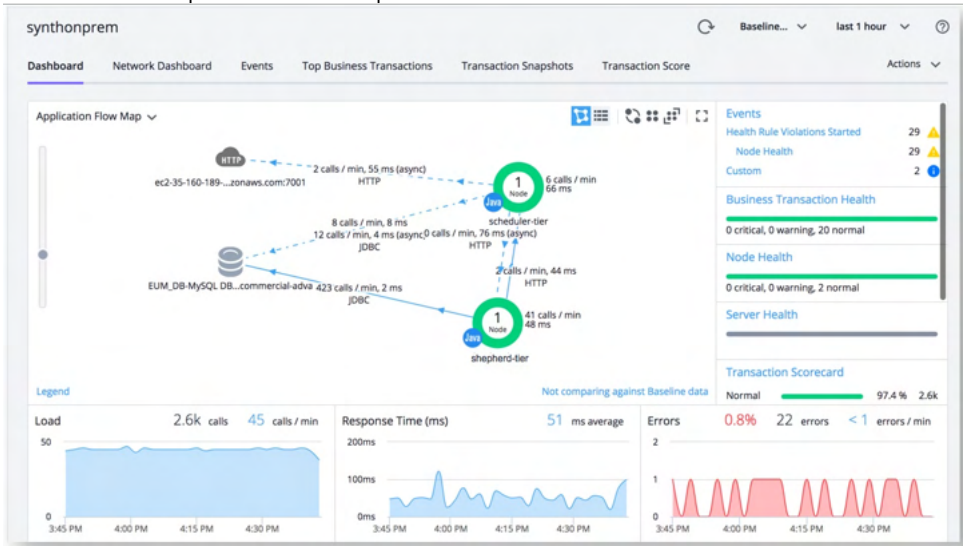
1. Confirm that the Java Agent is running:

```
ps -ef | grep javaagent
```

2. Check the Java Agent logs:

```
tail <java-agent>/ver<agent-version>/logs/<node-name>/agent-XXXX.log
```

3. Navigate to your Controller.
4. You should see a business application with the name assigned in the `controller-info.xml` file similar to the following. You should also see the tiers and nodes specified in the Java options.



Create Preset Dashboards and Health Rules

The Synthetic Server comes with the `post_deploy.sh` command to help create preset dashboards and health rules to monitor the Synthetic Server. The health rules are based on metrics generated by Synthetic Shepherd and not the host machine of the Synthetic Agent or the Java Agent, so they serve to complement the JMX health rules such as CPU usage, memory consumption, etc.

About the Preset Dashboards and Health Rules

The preset health rules issue warnings for when the Synthetic Server has a *busy* percentage of 80% and issue critical alerts when that busy percentage reaches 90%. The busy percentage is evaluated over the last 30 minutes.

Understanding the Busy Percentage

Each Synthetic Agent can only run one job at a time. When running a job, the Synthetic Agent is busy, and when it's not running, the Synthetic Agent is *not* busy. The busy percentage indicates the percentage of time that a Synthetic Agent is running jobs. For example, if a Synthetic Agent ran a job that took five minutes and no other jobs, then over the last 10 minutes, it was 50% busy. The busy percentage is based on a metric reported every minute and calculated using pages per minute (PPM). The busy percentage can be used to monitor one Synthetic Agent, a group of Synthetic Agents, or a location.

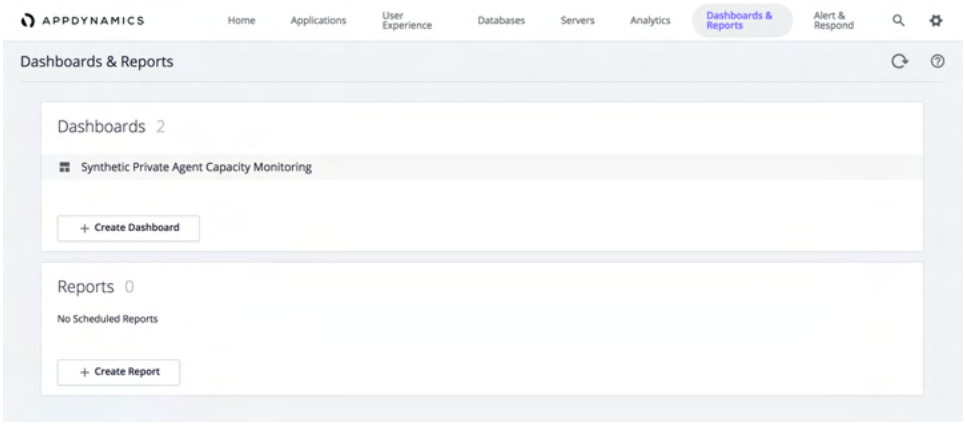
Create the Dashboards and Health Rules

Once you have verified that the instrumentation was successful and that business applications were created:

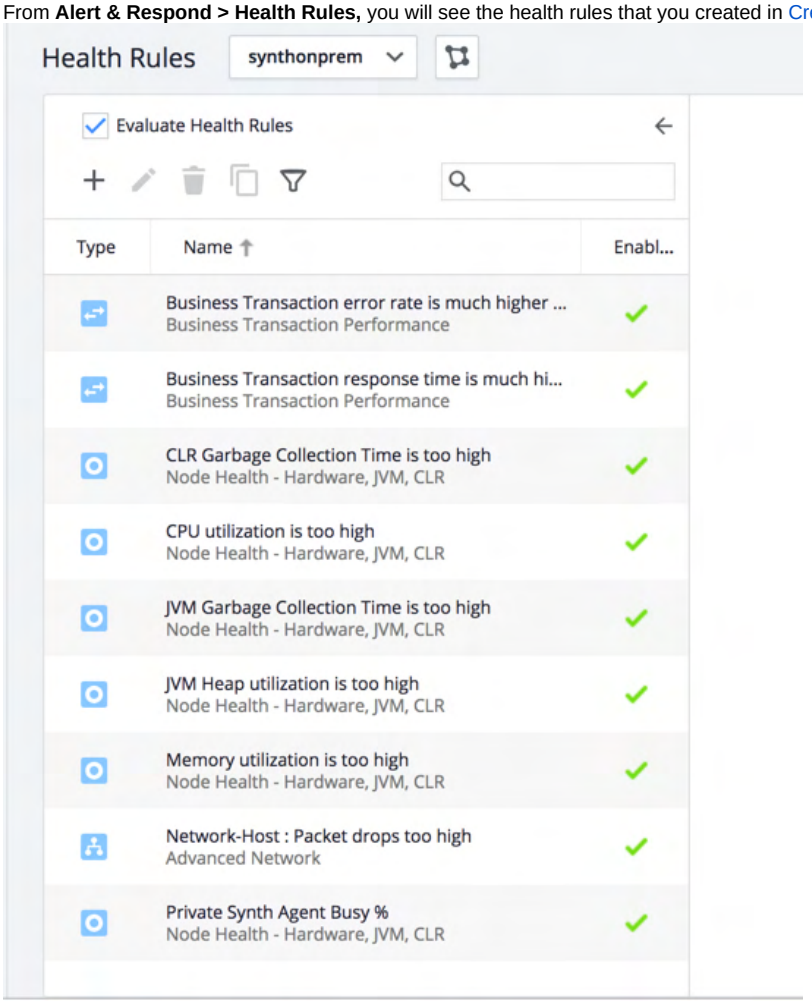
1. Log in to the machine hosting your Synthetic Server.
2. Change to the Synthetic Server home.
3. Run the following command to create the preset dashboards and health rules:

```
unix/post_deploy.sh
```

4. From the Controller UI, navigate to the **Dashboards & Reports** page.
5. You should see the dashboard **Synthetic Private Agent Capacity Monitoring** as shown here:



6. Open the business application for the on-premises Synthetic Server.
7. Navigate to **Alert & Respond > Health Rules**.
8. From **Alert & Respond > Health Rules,** you will see the health rules that you created in Create Preset Dashboards and Health Rules.



Create Custom Dashboards and Health Rules

The Java Agent will report metrics generated by Synthetic Server such as CPU usage, memory consumption, garbage collection, etc. You can use these metrics to create health rules and custom dashboards.

See the following pages for instructions:

- Custom Dashboards
- Configure Health Rules

Synthetic Server Endpoints

Related pages:

- Port Settings
- EUM Server Endpoints

The Synthetic Server has different endpoints serving distinct functions. This page provides a reference for testing the health and getting information about on-premises Synthetic Servers.

The endpoints include the following:

- **Synthetic Shepherd** - manages and dispatches jobs to the Synthetic Agents. In addition, the Synthetic Shepherd saves the measurement results to the filesystem and sends beacons containing the data to the Synthetic Scheduler.
- **Synthetic Scheduler** - handles the CRUD operations for jobs and manages the events generated for synthetic warnings and errors that occur in the measurement results.
- **Synthetic Feeder Client** - communicates with the SaaS Synthetic Feeder Server to access the Synthetic Hosted Agents.

Synthetic Server Endpoint URLs

The table below lists the endpoints, the default URL, and the supported paths.

Synthetic Server Endpoint	Default URL	Paths / Description
Synthetic Shepherd	`http(s)://<domain-name>:10101`	/version — Returns the version, build, commit, and timestamp of the Synthetic Shepherd.
Synthetic Scheduler	`http(s)://<domain-name>:12101`	/version — Returns the version, build, commit, and timestamp of the Synthetic Scheduler.
Synthetic Feeder Client	`http(s)://<domain-name>:16101`	/version — Returns the version, build, commit, and timestamp of the Synthetic Feeder Client.

Upgrade the Synthetic Server

This page describes how to upgrade the Synthetic Server to the latest version. This is often done alongside an upgrade to the other platform components, such as the EUM Server, the Controller, and the Events Service.

What the Upgrade Does

The upgrade procedure updates the schemas for the EUM Server database and the JAR files for the Synthetic Server if either is needed.

> The upgrade steps will not update 3rd-party software such as the Python library Flake8.

Who Should Use This Document

To upgrade the Synthetic Server to the latest available version, use this document.

Before You Begin

Before you upgrade the Synthetic Server:

1. Back up your current installation. There is no way to downgrade and changes are permanent.
2. Update platform components in the correct update order.

Upgrade Procedure

Follow the instructions below to upgrade your Synthetic Server.

Pre-requisite

The previous version of the Synthetic services database should be maintained.

Get the Latest Synthetic Server Installer

1. Go to the AppDynamics Downloads site and download the latest version of the Synthetic Server installer package.
2. Copy the Synthetic Server installer package (`synthetic-installer-linux-<version>.zip`) to the machine hosting your Synthetic Server.

Upgradation Steps

To upgrade Synthetic services first go to the installed Synthetic services path `cd <Synthetic_home>` and follow the steps given here:

1. Run the following command to stop Synthetic services:

```
"/unix/deploy.sh stop"
```

2. Take a back up of `inputs.groovy` file from `<Synthetic_home>` before deleting contents of `<Synthetic_Home>`. This is done to preserve previous configuration details.
3. Delete all the files and folders under `<Synthetic_home>.`
4. Copy and unzip the new Synthetic services installer under `<Synthetic_home>.`
5. Replace `inputs.groovy.sample` file with the backed up file (mentioned in step 2).
6. Run the following command to install new services:

```
"./unix/deploy.sh install"
```

Verify the New Version Has Been Installed

1. Once the installation is complete, confirm the Synthetic Server is running by entering:

```
ps aux | grep synthetic-processor
netstat -lan | grep "1[0,2,6]10[1,2]"
```

2. To view the updated version, curl to the `version` endpoint:

```
curl <on-prem-synthetic_server>:10101/version
curl <on-prem-synthetic_server>:12101/version
curl <on-prem-synthetic_server>:16101/version
```

Synthetic Private Agent Deployment

Related pages:

- Requirements for the Synthetic Private Agent
- Prepare the Host Machine for the Synthetic Private Agent
- Install the Synthetic Private Agent

Synthetic Private Agents enable you to monitor internal websites or services that aren't visible to the public Internet. You can also use Synthetic Private Agents to test from specific facilities on your network. For example, a bank might install a Synthetic Private Agent in each branch to monitor whether that branch can access the bank's applications.

The Synthetic Private Agent is a software package that you can install on Windows Server 2016, Windows Server 2012, or Windows Server 2012 R2. After installation, the Synthetic Private Agent connects to the Synthetic Server (on-premises or SaaS) and registers under your EUM account. Once this is done, you can configure synthetic jobs to run in your Synthetic Private Agent location.

Limitations

When using the Synthetic Private Agent, you will encounter the following limitations in the Controller UI:

- The **Synthetic Job Editor** displays options for all four browsers regardless of which are actually installed by your Synthetic Private Agent locations. If you select browsers that aren't available, you'll see an error when you save the job.
- When choosing a Synthetic Private Agent location from the **Synthetic Job Editor**, you will not be able to choose a connection speed because traffic shaping is not supported for Synthetic Private Agents.

Compatibility with 4.3 Controllers

Although you can use the Synthetic Private Agent with 4.3 Controllers, it is recommended that you upgrade to the 4.4 to avoid the following known issues seen in 4.3 Controllers:

- The **License** page will not display license usage for Synthetic Private Agents. The usage for Synthetic Hosted Agents will be correct.
- The **License** column on the **Synthetic Jobs** page will display incorrect information for jobs that use Synthetic Private Agents.
- The License information header at the top of the **Synthetic Jobs** page will be incorrect if you have any jobs that use Private Agents.
- In the **Synthetic Job Editor**, the locations for the Synthetic Private Agents and Synthetic Hosted Agents will be given in one flat list.

Requirements for the Synthetic Private Agent

To use the Synthetic Private Agent, make sure that you meet the requirements given in the table below:

Component	Requirement(s)	Notes
License	A license for Synthetic Private Agents.	The license for Browser Synthetic Monitoring does not include the usage of Synthetic Private Agents. You will need an additional license.
Hardware	• Processor: 2 GHz • Memory: 4 GB RAM • Disk Space: 40 GB	The machine should be a dedicated host.
Software	• Windows Server 2016 64-Bit (only supported for the Synthetic Private Agent 20.2 and higher) • Windows Server 2012 64-Bit or Windows Server 2012 R2 64-Bit (Only English version) • Microsoft .NET Framework 4.7 or greater	If you use Windows Server 2012, you'll be able to create jobs that test in IE10. If you use Windows Server 2012 R2, you'll be able to create jobs that test in IE11. If you need to test in both IE10 and IE11, then you can set up two Synthetic Private Agents (one on Windows Server 2012 and one on Windows Server 2012 R2) in the same location: They will automatically form a cluster that supports both browsers. Windows Server 2016 does not support synthetic tests of IE 10 browsers.
Network	Outbound network access to connect to the Synthetic Server: • SaaS: `https://synthetic.api.appdynamics.com:443` • On-Premises: `http://<on-prem-synthetic-server-host>:10101`	The Synthetic Server dispatches jobs from the Controller to your Synthetic Private Agent and collects the job results. For the on-premises Synthetic Server, you are recommended to use an HTTP proxy instead of directly connecting through SSL.

Prepare the Host Machine for the Synthetic Private Agent

Before you start installing the Synthetic Private Agent, you may need to make changes to the host machine. You should first try using the recommended settings for your host machine and then follow the steps for preparing the host machine based on your Windows environment.

Recommended Settings for the Host Machine

These are the recommendations for the host machine of the Synthetic Private Agent:

- Do not install other software on the Synthetic Agent machine. The Synthetic Private Agent measures performance, so running other software on the same server may adversely affect the measurement results. Also, the configuration changes for the installation might, conversely, adversely affect the other programs running on the machine.
- The host machine should be a Windows Workgroup and *not* part of a Windows Host Domain. See Windows Domains Versus Workgroups for an explanation.
- Do not run anti-virus software on the host machine.
- Use a password specific to the host machine.

Steps for Preparing the Host Machine

The table below lists the steps you will need to take to prepare your host machine based on the Windows environment.

Windows Environments	Running Anti-Virus Software?	Steps to Take
Workgroup (recommended)	Yes	1. Configure Local Policies for Workgroups 2. Remove Internet Browsers from Dedicated Machine 3. Prevent Anti-Virus Software from Affecting the Private Synthetic Agent
	No	1. Configure Local Policies for Workgroups 2. Remove Internet Browsers from Dedicated Machine
Windows Domain	Yes	1. Configure Group Policies for Windows Domains 2. Remove Internet Browsers from Dedicated Machine 3. Prevent Anti-Virus Software from Affecting the Private Synthetic Agent
	No	1. Configure Group Policies for Windows Domains 2. Remove Internet Browsers from Dedicated Machine

Windows Domains Versus Workgroups

The installer will create the user account `agent_user` that will be used to run the Synthetic Private Agent. You should *avoid* applying a Group Policy (GPO) on the host machine that will affect the `agent_user` account.

Machines that belong to Windows Domains generally have GPOs applied to them. Because of this, you are recommended to use a standalone Workgroup for the host machine when possible. In addition, if the host machine is a member of a Windows Domain, you will need to be a Domain Administrator to modify the GPOs. See Configure Group Policies for Windows Domains to learn which policy settings to change.

Ideally, in an enterprise deployment where Active Directory is set up, the host machine should be part of an Organizational Unit (OU) that has no interactive logon GPOs. This makes using Workgroups ideal. In addition, for Workgroups, you only need to be a Local Administrator to configure the Local Policies.

Configure Local Policies for Workgroups

The sections below show you how to change the following settings for Local Policies on Workgroups.

Remove Maximum Password Age

1. If the settings are not the same, from a PowerShell, open the **Local Security Policy** window:

```
> secpol.msc
```

2. From the **Local Security Policy** window, navigate to **Security Settings > Account Policies > Password Policy**.
3. Double-click **Maximum password age** to open the **Maximum password age Properties** dialog.
4. Set the value to **0** for the expiration date, so that the password never expires.
5. Click **OK**.

Add Users to the User Rights Assignment

1. From the **Local Security Policy** window, navigate to **Security Settings > Local Policies > User Rights Assignment**.
2. Double-click **Allow log on locally**.
3. If you don't see **Users** in the text area, click **Add User or Group**.
4. From the **Select Users or Groups** dialog, enter **Users** and click **Check Names**.
5. Click **OK**.

Configure Group Policies for Windows Domains

If the machine is a member of a Windows Domain, do the following:

Steps		Persona/User
1.	Check if the Group Policy settings are correct for installing the synthetic Private Agent.	Standard User Account, Local Administrator User Account, Domain Administrator Account
2.	Modify the Group Policy settings if needed.	Domain Administrator Account
3.	Verify that the Group Policies for interactive logon are not applied.	Domain Administrator Account

Check Group Policy Settings

1. From a PowerShell console, run the `gpresult` command to save the applied group policies to an HTML file:

   ```
   gpresult /H output.html
   ```

2. If one of the following is true, your Windows Domain is configured correctly:

   - The `output.html` file doesn't have any settings.
   - When you open `output.html` and navigate to **Settings > Policies > Windows Settings > Security Settings, you** see the policies and settings below.

Security Settings	Policy	Setting
Account Policies/Password Policy	Maximum password age	0 days
Local Policies/User Rights Assignments	Allow log on locally	Users (should be one of the settings)

3. If your Windows Domain is not configured correctly:

   a. Modify the relevant GPO settings
   b. Confirm that the interactive logon policies are not applied

Modify Group Policy Settings

Typically, the IT department will need to make GPO changes. The Domain Administrator in your IT department will need to change the following Group Policy settings that are applied to the host machine for the Synthetic Private Agent:

- No maximum password age is used for the Synthetic Private Agent account `agent_user`
- Permissions to log on have been granted to the `agent_user`

Confirm Interactive Logon Policies Are Not Applied

If you do apply Group Policies to the host machine, you will need to disable interactive logon Group Policies for autologon to work.

Verify that the **Interactive logon** policies below are not configured. If a policy has a setting, you will need to disable it.

Policy
Interactive logon: Message text for users attempting to log on
Interactive logon: Message title for users attempting to log on
Interactive logon: Prompt user to change the password before expiration

Remove Internet Browsers from Dedicated Machine

Before installing the Synthetic Private Agent, remove all internet browsers except Internet Explorer because the agent requires specific versions of the browsers.

Prevent Anti-Virus Software from Affecting the Private Synthetic Agent

Running anti-virus software may disrupt the operation or affect the performance of Private Synthetic Agents. If you are running anti-virus software on the same machine hosting the Synthetic Private Agent, you should do the following:

- exclude the agent's installation directory (the default is `C:\appdynamics\`) from virus scanning.
- do not allow your anti-virus software to quarantine the dynamic-link library `k9.dll`, which is injected into running processes for the Synthetic Private Agent to function properly.

If you have taken these precautions and are still having difficulty running anti-virus software, report your issue to your AppDynamics account representative.

Install the Synthetic Private Agent

The instructions below show you how to fetch and install a Synthetic Private Agent.

The installation of the Synthetic Private Agent will make the following changes to your system:

- Install the Synthetic Private Agent.
- Install browsers.
- Configure many aspects of Windows to make it amenable for collecting performance data from browsers, including changing several Windows registry settings.
- Turn on the Windows firewall and prevent any inbound network connections to the server.

Prerequisites

Before beginning the installation, confirm that you have completed the following:

- Met the requirements for the Synthetic Private Agent
- Prepared the Host Machine for the Synthetic Private Agent

Installation Steps

Follow the steps below to install the Synthetic Private Agent:

Fetch the Synthetic Private Agent

Navigate to the AppDynamics Downloads and download the Synthetic Private Agent.

Log in as the Administrator

Before you can copy and run the Synthetic Private Agent installer, you are required to log in as the *administrator* to the dedicated Windows machine for the Synthetic Private Agent.

Copy/Move the Synthetic Private Agent to Your Dedicated Machine

1. Move or copy the compressed archive file `SyntheticAgentInstaller.zip` containing the Synthetic Private Agent to this machine.
2. Unzip and decompress the archive file.
3. Confirm that the directory `SyntheticAgentInstaller` contains the installer script `install.ps1`.

Prepare Account and Location Information

Before you install the Synthetic Private Agent, you will need the following information:

- **EUM Account Name / EUM License Key**:
    1. Go to your SaaS Controller.
    2. Navigate to **License > Peak Usage**.
    3. Scroll down to the **User Experience** section.
    4. Copy the values for the **Account Name** and **License Key** fields.
- **Location name**: A user-friendly string for the location. You can use the same location information on multiple computers to add capacity to that location. For example, if the private location is for multiple machines running from the company office in SF, you might use "SF Office".
- **Location ID**: A unique alphanumeric string from 4-10 characters identifying a location where one or more agents may be running. You should create this ID based on the location name.
- **Latitude and longitude of the location**: Navigate to the **GPS Coordinates with Google Maps** tool at `https://www.gps-coordinates.net/` to get the latitude and longitude of your location.

Run the Installer

After you unzip and decompress the archive file:

1. Open a PowerShell console.

2. Change to the directory `SyntheticAgentInstaller`.
3. Set the PowerShell execution policy to `Unrestricted`:

```
> Set-ExecutionPolicy Unrestricted
```

4. Execute the PowerShell script `install.ps1` to install the Synthetic Private Agent, Internet browsers, and make changes to the Windows configuration:

```
> .\install.ps1
```

5. At the command prompts, enter the information that you prepared in the last step.

Connect the Agent to the Synthetic Server (Optional)

By default, the Synthetic Private Agent will report measurement results to the SaaS Synthetic Server located in the Americas business region. Thus, if your Synthetic Private Agent is located in the Americas business region and reporting to our SaaS Synthetic Server, continue to Start the Synthetic Private Agent.

If your Synthetic Private Agent is reporting measurement results to an on-prem Synthetic Server, see Connect Synthetic Agents to the Synthetic Server for configuration instructions.

To configure your Synthetic Private Agent to report to a SaaS Synthetic Server in the EMAC or APAC business regions, follow these instructions:

1. Select the **EMAC** or **APAC** tab in the following table to find the URL to the SaaS Synthetic Server in your region.

SaaS Domains and IP Ranges

2. Change to the directory `C:\appdynamics\synthetic-agent\synthetic-driver\conf`.
3. Edit the file `synthetic-driver.yml`.
4. Set the URL to SaaS Synthetic Shepherd in your region to the `shepherdUrl` as shown below:

```
## Use the SaaS Synthetic Server URL from the table above for EMAC or APAC.
shepherdUrl: http://<saas_synthetic-server_url>
```

5. Save your changes and close the `synthetic-driver.yml` file.

Start the Synthetic Private Agent

After installation is complete (the installer may restart your computer), start the Synthetic Private Agent by double-clicking the desktop shortcut **Start Agent**.

Confirm the Installation Was Successful

1. Go to your SaaS Controller and to the **Synthetic Job List.**
2. Click **Create a Job**.
3. From **Choose Locations**, select **Private** from the dropdown to view all the Synthetic Private Agents associated with your account.
4. Verify that your private location shows up as one of the available locations.

Configure Autologon

You should configure autologon so that the machine after booting up will automatically log on to the Administrator account. You are recommended to use the free Microsoft Sysinternals utility `Autologon.exe` for configuring autologon. `Autologon.exe` is easy to use, and more importantly, *encrypts* the account credentials stored in the Windows Registry.

To configure autologon:

1. Download `Autologon.exe` from https://docs.microsoft.com/en-us/sysinternals/downloads/autologon.
2. Double-click the executable **Autologon.exe**.
3. Enter the requested information (username, domain, password, etc.).
4. Click **Enable**.

You can also run `Autologon.exe` from the command line, but be sure to escape any special characters in your password.

To manually configure autologon by editing the registry, see How to turn on automatic logon in Windows. Note that the given instructions can change slightly depending on the version of Windows being used.

Schedule the Synthetic Private Agent to Start on Logon

After you have set up autologon, you still need to schedule the Synthetic Private Agent to automatically start after the machine logs on to the Administrator account.

1. Open a PowerShell console.
2. Right-click and save `startagent.xml` file to your `$HOME` directory.
3. Change to your `$HOME` directory.
4. If you are logged on as a different user than "Administrator", edit the `startagent.xml` file and replace the value "Administrator" for the following XML elements with your user name. (Remember, your user account still must have Administrator permissions.)
    - `<Author>`
    - `<UserId>`
5. Schedule the task of starting the Synthetic Private Agent:

    ```
    > schtasks /create /f /tn "Start the Synthetic Private Agent"/xml .\startagent.xml
    ```

6. Restart your machine.
7. After the machine boots up, you should see the Windows console open titled "Administrator: Windows PowerShell" that displays the log message "Synthetic-agent is running".

Advanced Options

Install Multiple Agents

After installing one Synthetic Private Agent, you should consider whether you need to install more Synthetic Private Agents.

Installing multiple agents is recommended for the following use cases:

- You want to **test multiple locations**. You need to install the Synthetic Private Agent on multiple machines and provide different location information during the installation. Each location will appear separately in the **Job Editor**.
- You need **more throughput to run more tests**. If you run a lot of synthetic jobs from a location, you may see errors like “Location is overloaded”. This means the machine is at capacity and you need to add another machine. You can simply install the agent on another machine and provide the same location information.
- You want **support for multiple versions of Internet Explore**r. See Software Requirements for more information.

Configure the Synthetic Private Agent

Related pages:

- Install the Synthetic Private Agent

This page describes the configuration options for the Synthetic Private Agent.

Configure Proxy for the Synthetic Private Agent

Although using a proxy for the Synthetic Private Agent is not mandatory, you can potentially improve security or provide load balancing.

Proxy Use Cases

The following are the various ways in which the Synthetic Private Agent could use a proxy to interact with an on-premises or SaaS Synthetic Server:

- Synthetic Private Agent interacting with a SaaS Synthetic Server
- On-premises Synthetic Server interacting with a SaaS Synthetic Server when Synthetic Hosted Agents are being used by the on-premises Synthetic Server
- Synthetic Private Agent interacting with an on-premises Synthetic Server

Proxy Configuration Steps

To configure the proxy:

1. Log on to the machine hosting the Synthetic Private Agent.
2. Change to the directory `C:\appdynamics\synthetic-agent\synthetic-driver\conf`.
3. Edit the file `synthetic-driver.yml`.
4. Find the setting `proxyConfig.`
5. Specify the proxy host and port with keys `proxyHost` and `proxyPort` as shown in the following example:

```
proxyConfig:
       proxyHost: <synthetic_server-hostname>
       proxyPort: 1501
       ...
```

6. (Optional) If you have enabled authentication with the proxy, specify the proxy username and password with the keys `proxyUsername` and `proxyPassword`:

```
proxyConfig:
       proxyHost: <synthetic_server-hostname>
       proxyPort: 1501
       proxyUsername: <proxy-username>
       proxyPassword: <proxy-password>
```

Set the Port for the Administration Console

The default port for the Administration connections is 8081. To avoid port conflicts, you can configure the Administration to listen to a different port.

In the file `C:\appdynamics\synthetic-agent\synthetic-driver\conf\synthetic-driver.yml`, navigate to the `server` section and modify the value of the `port` property for the `adminConnectors`.

```
server:
  applicationConnectors:
  - type: http
    port: 8888
  adminConnectors:
  - type: http
    port: 8091 # You can customize the port by changing the value here.
  requestLog:
    appenders: []
```

Filter Synthetic Traffic

You can configure the request headers in synthetic scripts to do the following:

Set Custom User Agent for Synthetic Scripts

You can set the user The EUM Server ignores requests for browser applications when the string "PTST/1.0" is part of the User Agent.

```
requestHeader:
   userAgent: "custom user agent"

driver.get('http://127.0.0.1:20000/green.html')
```

Include Synthetic Traffic in Browser RUM Traffic

The request header in synthetic scripts is tagged so that the requests are not included in Browser RUM pages. You can set the property `tagUserAgent` to `false` to include the synthetic traffic in the Browser RUM pages as shown below.

```
requestHeader:
   userAgent: "custom user agent"
   tagUserAgent: false

driver.get('http://127.0.0.1:20000/green.html')
```

Uninstall the Synthetic Private Agent

The following steps will delete the Synthetic Private Agent, but will not delete the third-party software packages installed with the Synthetic Private Agent. See Uninstall Third-Party Sofware Packages below for instructions.

1. From your desktop, double-click the **Stop Agent** shortcut icon.
2. From **Windows Explorer**, delete the directory `C:\appdynamics`.

Uninstall Third-Party Software Packages

When you install the Synthetic Private Agent, you also uninstall the following third-party software packages:

- chef-client
- Java runtime environment
- python

To delete these third-party software packages:

1. Navigate to **Control Panel > Programs > Uninstall**.
2. Select the software package(s) that you want to uninstall.
3. Click **Uninstall**.

Upgrade the Synthetic Private Agent

Follow these steps to upgrade the Synthetic Private Agent:

1. Sign on to the host machine of the Synthetic Private Agent.
2. From your desktop, double-click the **Stop Agent** shortcut icon.
3. Copy the file `C:\appdynamics\synthetic-agent\synthetic-driver\conf\synthetic-driver.yml` to `C:\Desktop\synthetic-driver.yml`.
4. Delete the directory `C:\appdynamics`.
5. From the AppDynamics Download site, download the latest version of the Synthetic Private Agent.
6. Move the zip file of the Synthetic Private Agent to the host machine.
7. Unzip the Synthetic Private Agent.
8. Prepare the following information for the installation:
    - EUM Account name
    - Location name
    - Location ID
    - Latitude/latitude of the location

    > If you don't recall this information, you can refer to the `location` and `privateClient` objects in the `C:\Desktop\synthetic-driver.yml` file.

9. Run the installer.
10. Once you've completed the installation, delete the YAML configuration file `C:\Desktop\synthetic-driver.yml.`

Troubleshoot the Synthetic Private Agent

This page provides troubleshooting instructions for some commonly experienced issues.

Authentication Errors

If you get authorization errors and do not see your private location in the **Synthetic Job Editor**, try the following:

- Check the log files for errors: `C:\appdynamics\synthetic-agent\log`
- Confirm that your account and location information is correct in the configuration file: `C:\appdynamics\synthetic-agent\synthetic-driver\conf\synthetic-driver.yml`

Update Account and Location Information

If you had to correct your account and location information, stop and restart the Synthetic Private Agent:

1. Stop the Synthetic Private Agent by double-clicking the desktop shortcut **`Stop Agent`**.
2. Restart the Synthetic Private Agent double-clicking the desktop shortcut **`Start Agent`**.

Administration Privilege Issues

If you have difficulty running the installer as a domain user with administrator privileges, create a local user account with administrative privileges and try again.

Anti-Virus Software Issues

See Using Anti-Virus Software for instructions on how to avoid anti-virus software scanning the Synthetic Private Agent files.

Add Users to Group Policy

If you encounter the following error message:

```
Users do not have allow logon permissions. Please add 'Users' to 'Allow log on locally' group policy. This is required because agent_user is a non administrator and is used to execute measurements.
```

1. Open a PowerShell console.
2. From the console, open the **Local Group Policy Editor**:

   ```
   > gedit.msc
   ```

3. Navigate to **Computer Configuration -> Windows Settings -> Security Settings -> Local Policies -> User Rights Assignment**.
4. Double-click the policy **Allow log on locally**.
5. Click **Add User or Group**.
6. From the **Select Users or Groups** dialog, enter **Users** as the object name.
7. Click **Object Types** to open the **Object Types** dialog.
8. Check the **Groups** checkbox.
9. Click **OK** to close the **Object Types** dialog.
10. Click **OK** to close the **Select Users or Groups** dialog.
11. From the **Allow log on locally Properties** dialog, click **Apply** to save your changes.
12. Click **OK** to close the **Allow log on locally Properties** dialog.
13. Close the **Local Group Policy Editor**.

Java Virtual Machine Issues

If you get errors from the Java Virtual Machine, verify the following details for your setup:

- Check if the Requirements for the Synthetic Private Agent are met, especially the amount of available memory. Also ensure that a 64-bit Operating System (OS) is installed.
- Check if there is any Java Runtime Environment (JRE) installed other than the one provided by the Synthetic Private Agent. Uninstall all other JREs if you find any.
- Make sure that you increase the maximum size of the memory allocation pool. For this:

    i. Open the following with an editor:

    ```
    C:\appdynamics\synthetic-agent\start_agent.bat
    ```

ii. Edit the following option set:

```
JVM_XMX=...
```

Synthetic Job Times Out for Internet Explorer 11

If you notice synthetic jobs running in Internet Explorer (IE) 11 in private agent locations are timing out, you may need to change the Protected Mode settings of IE 11 on the machine hosting your Synthetic Private Agent.

To diagnose and fix the potential issue with the Protected Mode settings:

1. From the **Synthetic Jobs** page, edit the job that is timing out to determine the private agent location.
2. Log on to the machine hosting the Synthetic Private Agent running in that private agent location.
3. Open the log file `C:\appdynamics\synthetic-agent\log\connector.log`.
4. If you see error messages similar to the one below, this verifies that you will have to update the Protected Mode settings for each of the four zones in IE 11.

```
CRITICAL 2018-06-25 15:13:26,019  ad.script.webdriver  measurementID=e325b030-c017-4155-bfd7-
19729b9ca3f1~8a0a5b3b-98a1-45e8-8746-9bxxxxx
      Failed to create WD Driver: unable to start remote browser: Message: Unable to create new remote
session. desired capabilities = Capabilities [\{ensureCleanSession=true, appdynamicsCapability=\
{flushSessions=true, outputDir=C:\appdynamics\synthetic-agent\measurements\1e0b7a1b-bffc-471d-8815-
1e40fxxxxx\results, commandWebhook=http://localhost:8888/v1/webdriver/action, testId=e325b030-c017-4155-
bfd7-19729b9ca3f1~8a0a5b3b-98a1-45e8-8746-9bf5xxxx}, acceptSslCerts=true, name=e325b030-c017-4155-bfd7-
19729b9ca3f1~8a0a5b3b-98a1-45e8-8746-9bxxxxxx, browserName=internet explorer, initialBrowserUrl=about:
blank, javascriptEnabled=true, version=, platform=WINDOWS}], required capabilities = Capabilities [\{}]
```

5. Launch IE 11.
6. From IE 11, navigate to  **> Internet options**.
7. Select the **Security** tab and then make sure that each zone (**Internet**, **Local intranet**, **Trusted Sites**, and **Restricted Sites**) has the same setting for the **Enabled Protected Mode** checkbox shown below. For example, you can either check the box and enable protected mode for each zone or uncheck the protected mode to disable protected mode for each zone.
8. Restart IE 11.
9. From your Controller, verify that the problematic synthetic job is no longer timing out for IE 11. You can trigger the job to run by editing and saving changes for the job.

Secure the Platform

AppDynamics includes security features that help to ensure the safety and integrity of your deployment.

About Controller Security

The Controller is installed with an HTTPS port enabled by default. SSL secures client connections and allows clients to authenticate the Controller. The Controller UI supports HTTP Basic Authentication, along with SAML and LDAP authentication. Role-based access controls in the UI allow you to manage user privileges.

While the security features of the Controller are enabled out of the box, there are some steps you should take to ensure the security of your deployment. These steps include but are not limited to:

- The SSL port uses a self-signed certificate. If you intend to terminate SSL connection at the Controller, you should replace the default certificate with your own, CA-signed certificate. If you replace the default SSL certificate on the Controller, you will also need to establish trust for the Controller's public key on the App Agent machine.

  As an alternative to terminating SSL at the Controller, you can put the Controller behind a reverse proxy that terminates SSL, relieving the Controller from having to process SSL.

- Along with a secure listening port, the Controller provides an unsecured, HTTP listening port as well. You should disable the port or block access to the point from any untrusted networks.
- Make sure that your App Agents connect to the Controller or to the reverse proxy if terminating SSL at a proxy, with SSL enabled.
- The Controller and underlying components, Glassfish and MySQL, include built-in user accounts. Be sure to change the passwords for the accounts regularly and in general, follow best practices for password management for the accounts. For information on changing the passwords for built-in users, see Update the Root User and Glassfish Admin Passwords.

Proxy Controller Connections

The AppDynamics Controller is often deployed to a protected network behind a proxy, which presents a virtual IP address to external connections, including to agents and browser clients. The proxy itself resides in the DMZ for the network and often terminates SSL connections from the client connections.

If clients use SSL, the reverse proxy can terminate SSL connections or maintain SSL through to the Controller. Terminating SSL at the proxy removes the processing burden from the Controller machine.

Using a secure proxy can simplify administration as a whole, by centralizing SSL key management to a single point. It allows you to use alternative PKI infrastructures, like OpenSSL.

See Use a Reverse Proxy for more information.

Additional Security Topics

- Secure the EUM Server
- Analytics and Data Security
- Agent-to-Controller Connections (for information on Securing Agent Connections)
- Roles and Permissions (for information regarding potentially sensitive user permissions)

HTTPS Support for the Enterprise Console

There is built-in HTTPS support for the Enterprise Console using a self-signed keystore file. The Enterprise Console supports either HTTP or HTTPS based on your choice during fresh installation or upgrade. You can reconfigure the Enterprise Console to revert from HTTPS back to HTTP, or vice versa, at any time.

About Enterprise Console SSL and Certificates

For the HTTPS client, the Enterprise Console packages the latest Mozilla truststore `cacerts.jks`, as it contains standard certificates. The Enterprise Console creates a `keystore.jks` file which contains a self-signed certificate. This certificate is imported into `cacerts.jks` during installation or upgrade.

For production use, AppDynamics strongly recommends that you replace the self-signed certificate with a certificate signed by a third-party Certificate Authority (CA) or your own internal CA.

This page describes how to:

- enable HTTPS for the Enterprise Console during installation or upgrade.
- update the certificate to a signed one.
- customize keystore credentials.

> ⚠ Replacing the entire keystore is not recommended unless you first export the existing artifacts from the default keystore and import them into your own keystore.
>
> It is also not recommended that you create your own self-signed certificate.

The exact steps to implement security typically vary depending on the security policies for the organization. For example, if your organization already has a signed certificate to use, such as a wildcard certificate used for your organization's domain, you can import it into the keystore using the Enterprise Console's update-certificate command. Otherwise, you can obtain a new one along with a certificate signing request.

Before Starting

On Linux machines, the Enterprise Console uses curl to check the responsiveness of the application URL. Therefore, SSL needs to have the latest NSS package to work.

For example, you can update the NSS package to the latest with the following command on CentOS:

```
yum update nss
```

> ⓘ Your update procedure or NSS package name may differ depending on your Linux operating system.

Enable Enterprise Console HTTPS for Fresh Installation

You can enable HTTPS during the Enterprise Console installation by selecting HTTPS as your preferred connection type:

1. Follow the steps to install the Enterprise Console:
   - For GUI Installation, click the HTTPS checkbox.

     > ⚠ If the checkbox is left unselected, then the installation will default to HTTP.

     The Enterprise Console will create a self-signed certificate and use it for your HTTPS connection.
   - For Silent Installation, edit the platform admin response `varfile` with the following parameter:

     ```
     platformAdmin.useHttps$Boolean=true
     ```

     This enables HTTPS connection and is the same as checking **Enable Https Connection** in the wizard installation option. Setting the parameter to false uses HTTP.
2. Optional: The Enterprise Console will create a self-signed certificate and use it for HTTPS connection. You can replace it with a certificate signed by a third-party CA. See Update to a Signed Certificate.

The Enterprise Console then configures the HTTPS protocol and disables HTTP in the Dropwizard configuration file, PlatformAdminApplication.yml. See the Dropwizard Configuration Reference for more information.

Example Enterprise Console HTTPS connector configuration:

```
server:
  type: simple
  connector:
    type: encrypted-https # encrypted-https is a customized HTTPS connector type in Enterprise Console, with
keystore password encrypted.
    port: 9191 # DO NOT REMOVE alternatives are 8080
    keyStorePath: /appdynamics/platform-admin/conf/keys/keystore.jks
    keyStorePassword: s_-001-12-v/yKyIweuGQ=iLpBEDTqfP7vj++WP+MKEg==
    trustStorePath: /appdynamics/platform-admin/conf/keys/cacerts.jks
    trustStorePassword: s_-001-12-hdLwJEOZbns=kDmS/pLvq2A43iCWLJEcTg==
    certAlias: ec-server # DO NOT change cert alias name in keystore files.
    validateCerts: false
    supportedProtocols: [TLSv1.2]
    bindHost: 0.0.0.0
  applicationContextPath: /
```

The Enterprise Console encrypts all plain text passwords in the configuration file.

Enable Enterprise Console HTTPS for Upgrades

You can enable HTTPS from HTTP during upgrades from 4.4.3 to 4.5 and post-4.5 upgrades. You can also enable HTTPS for an existing Enterprise Console instance by reinstalling the Enterprise Console application:

1. Follow the steps to upgrade the Enterprise Console.
    a. Change the platform admin response varfile with:

    ```
    platformAdmin.useHttps$Boolean=true
    ```

2. Complete the upgrade.
3. Optional: A self-signed certificate is created by the Enterprise Console and SSL is configured, just like it is with a fresh installation. You can replace it with a certificate signed by a third-party CA. See Update to a Signed Certificate.

When you upgrade your HTTPS supported Enterprise Console to future release versions, the Enterprise Console will follow the below protocols:

- **Upgrades with a self-signed certificate (the Enterprise Console installed certificate)**: The Enterprise Console will always recreate the new `keystore.jks`, with a new self-signed certificate in it, and update the `cacerts.jks` file with the new self-signed certificate under the `<EC_installationDir>/conf/keys` folder.
- **Upgrades with a signed certificate**: The Enterprise Console will not modify your signed certificate, leaving it as it was before the upgrade.
  Note: Do not change the `serverHostName` in the admin response varfile from a private IP/hostname (if you used a private IP /hostname as the `serverHostName` for a previous Enterprise Console fresh install or upgrade) to a public IP as the Enterprise Console will not support the signed certificate afterwards. This restriction only applies to upgrades with a signed certificate.
- **Upgrades with customized keystore/truststore path and passwords**: The Enterprise Console will back up the `.jks` files only if they are under the `<EC_installationDir>/conf/keys` folder. The Enterprise Console will restore your keystore/truststore paths and passwords in `PlatformAdminApplicationl.yml` before the upgrade, even if you move the `.jks` files or change the password.

  Changing the .jks files location is not recommended as they will not be backed up by the Enterprise Console if they are in another location.

Configure HTTPS for Enterprise Console in SAN Deployments

To configure HTTPS for the Enterprise Console deployed for a Subject Alternative Name (SAN) on AWS, you will need to generate keys from the `san.cnf` file. The instructions below show you how to enter multiple hostnames and aliases for the Enterprise Console in the `san.cnf` file and then generate the keys with it.

1. Create your `san.cnf` file for the SAN. In the following example `san.cnf` file, multiple domain names and aliases are defined in `[ alt_names ]`.

```
[ req ]
default_bits       = 2048
distinguished_name = req_distinguished_name
req_extensions     = req_ext
prompt             = no
[ req_distinguished_name ]
countryName            = IN
stateOrProvinceName    = Karnataka
localityName           = Bangalore
organizationName       = Appdynamics
commonName             = ECserver
[ req_ext ]
subjectAltName = @alt_names
[alt_names]
DNS.1   = ECserver.com
DNS.2   = ECserver.secondary.com
DNS.3   = ECserver.alias1.com
DNS.4   = ECserver.alias2.com
IP.1        = 10.10.10.10
IP.2        = 10.10.10.9
```

2. Using the san.cnf file, generate the private key and CSR with the following openssl command:

```
openssl req -new -newkey rsa:2048 -nodes -out sslcert.csr -keyout private.key -config san.cnf
```

3. Check the CSR to confirm the SANs are correct:

```
openssl req -noout -text -in sslcert.csr | grep DNS
openssl req -noout -text -in sslcert.csr | grep IP
```

4. Sign the CSR by a certified authority (CA).
5. Update the certificate for the Enterprise Console:

```
./platform-admin.sh update-certificate --private-key <privateKeyfile> --ssl-cert <sslCertFile> --ssl-
chain <sslChainfile1> <sslChainfile2> <...>
```

Customize Keystore Credentials

You can customize keystore credentials. The Enterprise Console preserves your customized keystore/truststore passwords.

> ⚠ To keep your customized files backed up, place them under the `<EC_installationDir>/conf/keys` directory. Placing your files anywhere besides `<EC_installationDir>/conf/keys` is not recommended because they may not be backed up.

If you change the keystore content for the Enterprise Console, you must re-run the `change-keystore-password` command and re-encrypt. Then, you need to restart the Enterprise Console. See Controller Secure Credential Store for more information.

Get Encrypted Password

You can update the password for the `.jks` files. If you do, you must also update the password in the `PlatformAdminApplication.yml` file.

> ⚠ Do not change the `supportedProtocols` in the `PlatformAdminApplication.yml` file.

To get an encrypted password:

> ⓘ - Make a note of the generated password to use in Step 4.
>
> - For the `keystore.jks` file to work for the Enterprise Console, the `storepass` and `keypass` must be the same.

1. Use the Enterprise Console CLI to encrypt the new password:

```
./platform-admin.sh encrypt -t '<plain_text_password>'
```

2. Change the storepass in keystore.jks by using the <plain_text_password> from Step 1:

```
keytool -storepasswd -keystore keystore.jks
```

3. Change the ec-server keypass in keystore.jks by using the <plain_text_password> from Step 1:

```
keytool -keypasswd -alias ec-server -keystore keystore.jks
```

4. Use the encrypted password to update the Enterprise Console Dropwizard confirmation yml file (PlatformAdminApplication.yml) for the key "keyStorePassword".
5. Restart the Enterprise Console.

Update to a Signed Certificate

You can update the built-in self-signed certificate created by the Enterprise Console with a CA signed certificate from an eligible CA authority.

> ⚠ If you want to reuse the existing public key from the Java keystore to generate a CSR request, you must import the signed certificates manually. See step 6 to 9 in Create a Certificate and Generate a CSR for more information.

Most Linux distributions include OpenSSL. If you are using Windows or your Linux distribution does not include OpenSSL, you may find more information on the OpenSSL website.

1. Obtain a signed certificate:
   a. Create a csr request.

```
//Some CAs will create everything for you, including the private key. You may use the following
keytool command to create a csr request from existing keystore.jks.
keytool -certreq -alias ec-server -keystore keystore.jks -file AppDynamics.csr
```

or

```
//You can also use the following openssl command to create your own private key and csr request.
openssl req -new -newkey rsa:2048 -nodes -out <name of csr request file>.csr -keyout <name of
private key>.key -subj "/C=<custom>/ST=<custom>/L=<custom>/O=<custom>/OU=<custom>/CN=<hostname>"
```

   b. Submit the certificate signing request file generated by the command (AppDynamics.csr in our example command) to your Certificate Authority of choice. When it's ready, the CA will return the signed certificate and any root and intermediate certificates required for the trust chain. The response from the Certificate Authority should include any special instructions for importing the certificate, if needed. If the CA supplies the certificate in text format, just copy and paste the text into a text file.
2. Run the Enterprise Console update certificate CLI command:

```
./platform-admin.sh update-certificate --private-key <privateKeyfile> --ssl-cert <sslCertFile> --ssl-
chain <sslChainfile1> <sslChainfile2> ...
```

> ⚠ Refer to the following help points when running this command:
>
> - The privateKeyfile, sslCertFile, and ssl-chain files do not have any file format restrictions. Any file format, such as .pkey and .txt, should work, as long as it is readable.
> - The privateKeyfile file content must follow the PKCS8 format.
> - sslCertFile is your SSL certificate file.
> - ssl-chain files are additional certificates, such as intermediate certificates. These are optional, and you may provide as many of them as you would like.

This command updates the certificate in the keystore and truststore in the configuration yml file.

3. Restart the Enterprise Console for the new SSL configurations to take effect.

Verify the Use of SSL

You can test that your HTTPS support works by logging into the Enterprise Console GUI.

To make sure the configuration works, use a browser to connect to the Enterprise Console over the default secure port, port 9191:

```
https://<hosthame>:9191
```

Specify the hostname you used when you installed the Enterprise Console. The default port is 9191. This port needs to be exposed from your firewall rules so you can access the port from any place. See Port Settings for more information.

Make sure the Enterprise Console entry page loads in the browser correctly.

Depending on your browser, you may have to perform additional steps to verify your connection. For instance, for self-signed certificates on Chrome, you have to click to proceed. On Firefox, you have to create a security exception to proceed.

You can also verify that your configuration works by running commands on the Enterprise Console CLI.

Expired Certificate

In case of an expired certificate, the Enterprise Console CLI will still continue to work, but the CLI will also print out a warning, notifying you that the certificate has expired.

Upgrades to the Enterprise Console remain unaffected by expired certificates; when you try to upgrade the Enterprise Console without knowing that the certificate has expired, the upgrade should still succeed.

You can update the Enterprise Console self-signed certificate by reinstalling the application. For your own signed certificate, you can obtain a new one from CA and run the CLI command from Update to a Signed Certificate.

Revert the Enterprise Console to HTTP

You can fall back to HTTP from HTTPS support by reinstalling the Enterprise Console application.

1. Follow the steps to upgrade the Enterprise Console.
    a. Change the platform admin response varfile with:

    ```
    platformAdmin.useHttps$Boolean=false
    ```

2. Complete the upgrade.

The Enterprise Console backs up any self-signed or signed certificate that is under `<EC_installationDir>/conf/keys`. However, if you manually move the `keystore.jks` and `truststore.jks` to your own location, then you will need to back up your customized certificates and SSL related files on your own before the upgrade.

Troubleshooting Common SSL Related Issues

This section covers a few of the most common Enterprise Console SSL related issues. It may help to enable `-Djavax.net.debug=ssl` in the platform-admin.sh execute CLI command section before troubleshooting.

Enterprise Console CLI Issue

If the installation or upgrade succeeds but the Enterprise Console CLI does not work, the CLI will remind you to check if the serverHostName you entered in the installer varfile is valid. The error will state that the `Hostname %s not verified.`

Certificates Without an Extension Field Issue

Certificates without an extension field, especially free signed certificates, will not work. They will lead to a KeyUsage error. Only signed certificates from an eligible CA authority should be used.

Enterprise Console Restart Issue

NSS package needs to be updated to the latest version on Linux machines. The Enterprise Console may not be able to restart if the NSS package is not updated to the latest version on Linux machines. See Before Starting for more information.

Controller SSL and Certificates

The Controller comes with a preconfigured HTTPS port (port 8181 by default) that is secured by a self-signed certificate. This page describes how to replace the default certificate with your own custom certificate.

About Controller SSL and Certificates

For production use, AppDynamics strongly recommends that you replace the self-signed certificate with a certificate signed by a third-party Certificate Authority (CA) or your own internal CA. If you are deploying .NET Agents, you must replace the self-signed certificate with one signed by a CA, since the .NET agents do not work with self-signed certificates.

Controller SSL Certificates

You can manage your Controller SSL certificate on the Enterprise Console UI under Configurations. The Appserver Configurations and Reports Service Configurations pages both contain sections that display the SSL certificate information and provide an Edit Certificate option.

Controller Keystore and Artifacts

This page describes how to replace the existing key in the default keystore. Replacing the entire keystore is not recommended unless you first export the existing artifacts from the default keystore and import them into your own keystore.

The default Controller keystore includes the following artifacts:

- **glassfish-instance**: A self-signed private key provided the Glassfish application server.
- **s1as**: A self-signed private key provided with the Glassfish application server used by the Controller for secure communication on port 8181.
- **reporting-instance**: A private key used by the reporting service, the service that enables scheduled reports.

Update Keystore Passwords

You can modify the password for the `keystore.jks` and `cacert.jks` files that are used to generate the keystore artifacts. The password for both files must be the same.

You cannot modify, however, the password for the `reporting-service.pfx` file that is generated by the keystore artifact `reporting-instance` and used by the Reporting Service.

How to View the Keystore

You can view the contents of the keystore yourself using the keytool utility. To do so, from the <controller_home>/jre/bin directory, run the following command. Enter the default keystore password changeit when prompted.

```
keytool -list -v -keystore /home/ec2-user/appDplatform/product/controller/appserver/glassfish/domains/domain1
/config/keystore.jks
```

The exact steps to implement security typically vary depending on the security policies for the organization. For example, if your organization already has a certificate to use, such as a wildcard certificate used for your organization's domain, you can import the existing certificate into the Controller keystore. Otherwise, you'll need to generate a new one along with a certificate signing request. The following sections take you through these scenarios.

Before Starting

The following instructions describe how to configure SSL using the Java `keytool` utility bundled with the Controller installation. You can find the `keytool` utility in the following location:

- `<controller_home>/jre/bin`

The steps assume that the keytool is in the operating system's path variable. To run the commands as shown, you first need to put the `keytool` utility in your system's path. Use the method appropriate for your operating system to add the `keytool` to your path.

While the directory paths in this topic use forward slashes, the instructions apply to both Linux and Windows Operating System environments. The steps note where there are differences in the use of commands between operating systems.

Create a Certificate and Generate a CSR

If you don't have a certificate to use for the Controller, create it as follows.

AppDynamics requires using a X.509 digital certificate, which works with any file type.

In these steps, you generate a new certificate within the Controller's active keystore, so it has immediate effect.

The steps are intended to be used in a staging environment, and require the Controller to be shut down and restarted. Alternatively, you can generate the key as described here but in a temporary keystore rather than the Controller's active keystore. After the certificate is signed, you can import the key from the temporary keystore to the Controller's keystore.

1. At a command prompt, change directories to the following location:

   ```
   <controller_home>/appserver/glassfish/domains/domain1/config
   ```

2. Create a backup of the keystore file. For example, on Linux, you can run:

   ```
   cp keystore.jks keystore.jks.backup
   ```

   On Windows, you can use the copy command in a similar manner.
3. If it's still running, stop the Controller.
4. Delete the existing certificate with the alias s1as from the keystore:

   ```
   keytool -delete -alias s1as -keystore keystore.jks
   ```

5. Create a new key pair in the keystore using the following command. This command creates a key pair with a validity of 1825 days (5 years). Replace 1825 with the validity period appropriate for your environment, if desired.

   ```
   keytool -genkeypair -alias s1as -keyalg RSA -keystore keystore.jks -keysize 2048 -validity 1825
   ```

   Follow the on-screen instructions to configure the certificate. Note that:

   - For the first and last name, enter the domain name where the Controller is running, for example, `controller.example.com`.
   - Enter the default password for the key, `changeit`.

   This generates a self-signed certificate in the keystore. We'll generate a signing request for the certificate next. You can now restart the Controller and continue to use it. Since it still has a temporary self-signed certificate, browsers attempting to connect to the Controller UI will get a warning to the effect that its certificate could not be verified.

   > See Change Keystore Password for information on changing the default password for the keystore and certificates.

6. Generate a certificate signing request for the certificate you created as follows:

   ```
   keytool -certreq -alias s1as -keystore keystore.jks -file AppDynamics.csr
   ```

7. Submit the certificate signing request file generated by the command (`AppDynamics.csr` in our example command) to your Certificate Authority of choice.
   When it's ready, the CA will return the signed certificate and any root and intermediary certificates required for the trust chain. The response from the Certificate Authority should include any special instructions for importing the certificate if needed. If the CA supplies the certificate in text format, just copy and paste the text into a text file.
8. Import the signed certificate:

   ```
   keytool -import -trustcacerts -alias s1as -file mycert.cer -keystore keystore.jks
   ```

   This command assumes the certificate is located in a file named `mycert.cer`.
9. If you get the error "Failed to establish chain from reply", install the issuing Certificate Authority's root and any intermediate certificates into the keystore. The root CA chain establishes the validity of the CA signature on your certificate. Although most common root CA chains are included in the bundled JVM's trust store, you may need to import additional root certificates, such as certificates belonging to a private CA. To do so:

   ```
   keytool -import -alias [Any_alias] -file <path_to_root_or_intermediate_cert> -keystore <controller_home>
   /appserver/glassfish/domains/domain1/config/cacerts.jks
   ```

When done importing the certificate chain, try importing the signed certificate again.

Import an Existing Keypair into the Keystore

These steps describe how to import an existing public and private key into the Controller keystore. We'll step through this scenario assuming that the existing public and private keys need to be converted to a format compatible with Java Keystore, say from DER format to PKCS#12. You'll need to use OpenSSL to combine the public and private keys, and then use `keytool` to import the combined keys into the Controller's keystore.

Most Linux distributions include OpenSSL. If you are using Windows or your Linux distribution does not include OpenSSL, you may find more information on the OpenSSL website.

This assumes that we have the following files:

- private key: `private.key`
- signed public key: `cert.crt`
- CA root chain: `ca.crt`

The private key you use for the following steps must be in plain text format. Also, when performing the following procedures, do not attempt to associate a password to the private key as you convert it to PKCS12 keystore form. If you do, the following steps can be completed as described, but you will encounter an exception when starting up the Controller, with the error message: `java.security.UnrecoverableKeyException: Cannot recover key`.

To import an existing keypair into the Controller keystore

1. Use OpenSSL to combine your existing private key and public key into a compatible Java keystore:

   ```
   openssl pkcs12 -inkey private.key -in cert.crt -export -out keystore.p12
   ```

2. If the Controller is still running, stop it.
3. Change to the `keystore` directory:

   ```
   cd <controller_home>/appserver/glassfish/domains/domain1/config/
   ```

4. Create a backup of the `keystore` file. For example, on Linux, you can run:

   ```
   cp keystore.jks keystore.jks.backup
   ```

   On Windows, you can use the copy command in a similar manner.
5. Delete the self-signed certificate with alias `s1as` from the default keystore:

   ```
   keytool -delete -alias s1as -keystore keystore.jks
   ```

6. Import the PKCS #12 key into the default keystore:

   ```
   keytool -importkeystore -srckeystore keystore.p12 -srcstoretype pkcs12 -destkeystore keystore.jks -
   deststoretype JKS
   ```

7. Update the alias name on the key pair you just imported:

   > ⚠ The alias name should be `s1as`. Do not change it from this name.

   ```
   keytool -changealias -alias "1" -destalias "s1as" -keystore keystore.jks
   ```

8. Change the password of the imported private key:

   ```
   keytool -keypasswd -keystore keystore.jks -alias s1as -keypass <.p12_file_password> -new <password>
   ```

   For the new private key password, use the default (`changeit`) or the master password set as described in Change Keystore Password, if changed.
9. If you get the error "Failed to establish chain from reply", install the issuing Certificate Authority's root and any intermediate certificates into the keystore. The root CA chain establishes the validity of the CA signature on your certificate. Although most common root CA chains are included in the `cacerts.jks` truststore, you may need to import additional root certificates. To do so:

```
keytool -import -alias <Any_alias> -file <path_to_root_or_intermediate_cert> -keystore <controller_home>
/appserver/glassfish/domains/domain1/config/cacerts.jks
```

When done, try importing the signed certificate again.

10. Start the Controller.

Verify the Use of SSL

To make sure the configuration works, use a browser to connect to the Controller over the default secure port, port 8181:

```
https://<controller_host>:8181/controller
```

Make sure the Controller entry page loads in the browser correctly. Also, verify that the browser indicates a secure connection. Most browsers display a lock icon next to the URL to indicate a secure connection.

After changing the certificate on the Controller, you will need to import the public key of the certificate to the agent truststore. For information on how to do this, see the topic specific for the agent type:

- EUM aggregator: Troubleshoot Your EUM Setup
- Java Agent: Enable SSL (Java)
- .NET: Enable SSL (.NET)

If there is no proxy configured and the agent is reporting to the Controller itself, then the following changes are also mandatory:

1. Run the following command:

   ```
   platform-admin.sh stop-controller-appserver
   ```

   On Windows, run this command from an elevated command prompt (which you can open by right-clicking on the Command Prompt icon in the Windows Start menu and choosing **Run as administrator**):

   ```
   platform-admin.exe cli stop-controller-appserver
   ```

2. Search for the following properties in `<controller_home>/appserver/glassfish/domains/domain1/config/domain.xml`, and replace the port with the SSL port, as the non-secure port is disabled:

   > ⚠ You should also edit the domain.xml configurations on the Controller Settings page of the Enterprise Console to retain your settings. See Update Platform Configurations for more information.

   ```
   -Dappdynamics.controller.port=
   -Dappdynamics.controller.services.port=
   ```

3. In the following property, change the protocol from HTTP to HTTPS, and change the port to the secure port.

   ```
   -Dappdynamics.controller.ui.deeplink.url=
   ```

   You can also use REST API to update the deeplink URL:

   ```
   curl -k --basic --user root@system --header "Content-Type: application/json" --data '
   { "controllerURL": "https://<controller>:<ssl_port>" }' https://<controller>:<ssl_port>/controller/rest
   /accounts/<ACCOUNT-NAME>/update-controller-url
   ```

4. Add the following JVM argument anywhere above or below the above JVM arguments to ensure the internal agent connects using SSL.

   ```
   -Dappdynamics.controller.ssl.enabled=true
   ```

5. Run the following command:

```
platform-admin.sh start-controller-appserver
```

On Windows, run the following in an elevated command prompt:

```
platform-admin.exe cli start-controller-appserver
```

You can also use the `modifyJVMOptions.sh` script to make the changes.

Change Keystore Password

The default password for the keystore used by the Controller is `changeit`. This is the default password for the Glassfish keystore, and is a well-known (and thus insecure) password. For a secure installation, you need to change it.

> Changing the password in this manner does not affect the administration password you use to access the Glassfish administration console. See Update the Root User and Glassfish Admin Passwords for information on changing this password.

To change the password you must use the Glassfish administration tool (rather than the `keytool` utility directly). Using the Glassfish administration tool allows the Glassfish instance to access the keys at runtime.

If you change the keystore password directly using the `keytool`, the Controller generates the following error message at start up:

```
Caused by: java.lang.IllegalStateException: Keystore was tampered with,
or password was incorrect
```

If you encounter this scenario, change the password using the `asadmin` utility.

To change Glassfish passwords

1. Stop the Controller.
2. Change the Glassfish master password:

```
<controller_home>/appserver/glassfish/bin/asadmin change-master-password --savemasterpassword=true
```

Changing the master password with `asadmin` changes the password for the domain-passwords, `cacerts.jks`, and `keystore.jks` stores (including the s1as, reporting-instance, and glassfish-instance private keys in `keystore.jks`).

However if you customized any additional keys or existing key passwords, and they do not match the master password, when you change the master password, the following error is generated:

```
./asadmin change-master-password --savemasterpassword=true
Enter the new master password>
Enter the new master password again>
Caught an Exception: {0}
Command change-master-password executed successfully.
```

This indicates that the store password for `keystore.jks` has been set to the master password, but one or more of the private keys still has a different key password and do not match the master password. This prevents the Controller application from starting and generates the following error:

```
java.lang.Error: java.security.UnrecoverableKeyException: Cannot recover key
```

1. To resolve this issue, update each of the private key passwords in `keystore.jks` (s1as, reporting-instance, and glassfish-instance) to ensure that they match the master password by entering the following `keytool` command:

> Replace the `<JRE_HOME>`, `<alias_name>`, and `<controller_home>` variables with your information before executing the `keytool` command.

```
<JRE_HOME>/bin/keytool -keypasswd -alias <alias_name> -keystore <controller_home>/appserver/glassfish
/domains/domain1/config/keystore.jks -storepass <master_password>
```

2. To confirm that the default values for `<alias_name>` are `s1as`, reporting-instance, and glassfish-instance, execute the following command to list the contents of `keystore.jks`:

> Replace the <JRE_HOME>, <alias_name>, and <controller_home> variables with your information before executing the keytool command.

```
<JRE_HOME>/bin/keytool -list -keystore <controller_home>/appserver/glassfish/domains/domain1/config
/keystore.jks -storepass <master_password>
```

If the key password matches the master password, the message "Passwords must differ*"* displays when entering the new key password. This validates that the key password was set correctly.

3. Restart the Controller and ensure it starts without errors.

Updating an Expired Certificate

The steps to renew an expired or soon-to-expire certificate are similar to those for replacing the default certificate, as documented in Create a Certificate and Generate a CSR. To update the expired certificate:

1. Back up the existing keystore.
2. At a command prompt, change directories to the following location:

```
<controller_home>/appserver/glassfish/domains/domain1/config
```

3. Create a backup of the keystore file.
    a. On Linux, you can run the following command:

    ```
    cp keystore.jks keystore.jks.backup
    ```

    b. On Windows, you can use the copy command in a similar manner.

    > If the controller is still running, stop the controller.

4. Since you already have a Java keystore, run the following command to issue a certificate signing request. You should use this keystore for the csr and **not create a new one**. You will be importing the new certificate into this keystore.

```
keytool -certreq -alias s1as -keystore keystore.jks -file AppDynamics.csr
```

5. Submit the certificate signing request file Appdynamics.csr generated by the above example command to your Certificate Authority of choice.
    a. When it's ready, the Certificate Authority will return the signed certificate and any root and intermediary certificates required for the trust chain.
    b. The response from the Certificate Authority should include instructions for importing the certificate if needed.

> If the Certificate Authority supplies the certificate in text format, copy and paste the text into a text file.

6. You can list out the obtained certificate as follows if it is not in text format.

```
keytool -printcert -v -file <your obtained certificate>
```

7. Import the signed certificate obtained into the keystore that you already have.

```
keytool -import -alias s1as -file <your obtained certificate> -keystore keystore.jks
```

    a. The imported certificate will replace the old one, provided you use the same alias as the previous one.
    b. Sometimes the root and intermediate certificates of the certification authority are also expired. If that's the case, you will see the message Failed to establish chain from reply.

8. If the root and intermediate certificates of the certification authority are expired, they also have to be imported in your cacerts.jks so that the chain of trust can be established. You can follow your certification authority's instructions to download the root and intermediate certificates.
    a. Keep the same alias as before for root and intermediate when you import these certificates into cacerts.jks.

```
keytool -import -alias <previous alias used for the certificate> -file
<path_to_root_or_intermediate_cert> -keystore <controller_home>/appserver/glassfish/domains/domain1
/config/cacerts.jks
```

Trust Stores and Keystores

- Java trust store, cacerts, contain root certificates of well-known certification authorities. The validity of a certificate presented during the TLS/SSL (Transport Layer Security/Secure Sockets Layer) session is checked from `cacerts.jks`. There are no private keys or passwords in cacerts. They will contain the intermediate and root certificates of certification authorities.
- Java Keystore is used to store private key and the identify certificate for the server, which means that the keystore is used to store your server's credentials.

Set the Security Protocol

Related Pages:

- Agent-to-Controller Connections

This page describes the security protocol used by an on-premises Controller, and how you can modify it.

Default Security Protocol

The Controller secures connections using TLSv1.2 by default. However, you can change the security protocols used by the Controller if needed. For instance, you need to change the protocol if you are using agents that don't support TLSv1.2. These agents include:

- Java Agent version 3.8.1 or earlier (see Agent and Controller Compatibility for complete SSL compatibility information)
- .NET Agent running on .NET Framework 4.5 or earlier

If upgrading the agents or .NET framework is not possible, you will need to enable TLSv1 and SSL3 on the Controller using the `asadmin` command-line utility. To use the utility, you will need to supply the password configured for the root user for the Controller.

These changes require a restart of the Controller application server, which results in a brief service downtime. You may wish to apply these change when the downtime will have the least impact.

To maintain a secure environment, APIs that are downstream of the Controller should also use TLS. If SSL3 is required, you can enable it. See the Oracle JDK 8 documentation.

Enable TLS for a Controller

1. Open a browser and navigate to the Enterprise Console GUI:

   ```
   http(s)://<hostname>:<port>
   ```

   9191 is the default port.
2. Navigate to AppServer Configurations by choosing the platform, **Configurations**, **Controller Settings**, and **Appserver Configurations**.
3. In the Domain Protocols box on the JVM Options tab, edit the `configs.config.server-config.network-config.protocols.protocol.http-listener-2.ssl.tls-enabled=false` parameter to `true`.
4. Click **Save**.

You do not need to restart the Controller application server since the configuration change job automatically does so for you.

Enabling Stronger Encryption Keys

By default, the Controller's embedded Java runtime only supports up to 128-bit encryption key lengths for secure connections. You can, however, enable up to 256-bit encryption keys so the Controller can establish connections using the stronger ciphers.

To enable stronger keys in encryption keys in the Controller, follow the instructions for the Controller version you are running.

After restarting the Controller app server, the following cipher suites become available:

- `TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384`
- `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384`
- `TLS_RSA_WITH_AES_256_CBC_SHA256`
- `TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384`
- `TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384`
- `TLS_DHE_RSA_WITH_AES_256_CBC_SHA256`
- `TLS_DHE_DSS_WITH_AES_256_CBC_SHA256`
- `TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA`
- `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA`
- `TLS_RSA_WITH_AES_256_CBC_SHA`
- `TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA`
- `TLS_ECDH_RSA_WITH_AES_256_CBC_SHA`
- `TLS_DHE_RSA_WITH_AES_256_CBC_SHA`
- `TLS_DHE_DSS_WITH_AES_256_CBC_SHA`

Configure a Controller SSH Key

You can set up SSH (Secure Shell) with public/private key pairs so that you do not have to type the password each time you access a Controller machine by SSH. Setting up keys allows scripts and automation processes to access the Controller easily. You can generate DSA or if you want stronger encryption, RSA keys.

Set up SSH Key Pairs Using DSA

1. Run the `ssh` command that sets up the key pair:

   ```
   % ssh-keygen -t dsa
   ```

2. At the following prompt, press **Enter** to accept the default key location, or type another:

   ```
   Generating public/private dsa key pair.

   Enter file in which to save the key (~/.ssh/id_dsa):
   ```

3. Press return at the password prompt:

   ```
   Enter passphrase (empty for no passphrase):
   ```

4. Press **Return** again to confirm the password:

   ```
   Enter same passphrase again:
   ```

   You should see the following information:

   ```
   Your identification has  been saved in ~/.ssh/id_dsa

   Your public key has been saved in  ~/.ssh/id_dsa.pub

   The key fingerprint is: <Some really long string>
   ```

   If SSH continues to prompt you for your password, verify your permissions in your remote `.ssh` directory. It should have only your own read/write /access permission (octal 700):

   ```
   % chmod 700 ~/.ssh
   ```

5. Open the local `~/.ssh/id_dsa.pub` file and paste its contents into the `~/.ssh/authorized_keys` file on the remote host.
6. Update the permissions on the `authorized_keys` file on the remote host as follows:

   ```
   % chmod 600 ~/.ssh/authorized_keys
   ```

Set up SSH Key Pairs Using RSA

Run the `ssh` command that sets up the key pair:

```
% ssh-keygen -t rsa
```

The generated files will be named `id_rsa` and `id_rsa.pub`, instead of `id_dsa` and `id_dsa.pub`.

Otherwise, the remaining steps are identical to those beginning with step 2 in the steps above.

Controller Secure Credential Store

Related pages:

- Controller Data and Backups

The Controller creates a secure credential keystore that holds a secret key used to encrypt credentials.

Stored Credentials

The secure credential store manages the following credentials:

- LDAP authentication user password. See LDAP Authentication.
- Database collector credentials, including database user password and the machine user password.
- SMTP server/Email passwords.
- AppDynamics account access keys.

Back up the credential store as part of your normal backup procedures for the Controller, as described in the following section.

Secure Credential Store Backup

Make sure your Controller backup plan includes the secure credential keystore file `.appd.scskeystore.` In the case that the secure credential keystore file should become corrupted, restore the `.appd.scskeystore` file from backup.

If you run the Controller in high availability mode, both the primary Controller and the secondary Controller must use the same secure credential keystore file. If you use an HA deployment strategy, verify that it propagates the secure credential keystore file from the primary to the secondary.

Replace a Compromised Secure Credential Store

The following steps describe how to replace a secure credential store. It assumes the following:

- You have a single-tenant Controller installation.
- You know the plain-text value of your Account Access Key. You can view the access key in the Controller under **Settings > License**.

As detailed in the sections that follow, the steps are broken into these parts:

1. Create a new secure credential store.
2. Update the Controller with the password of the new secure credential store.
3. Update the account access key.
4. Update the account access key for the system account.
5. Restart the Controller and update passwords.

Create a new Secure Credential Store

1. Rename the existing secure credential keystore file.
2. Initialize a new secure credential keystore using the secure credential store utility.

   By default the utility installs to: `<controller_home>/tools/lib/scs-tool.jar`

   For example:

   ```
   /controller/jre8/bin/java -jar ./scs-tool.jar generate_ks -filename '<controller_home>/.appd.
   scskeystore' -storepass 'MyCredentialStorePassword'
   ```

   The secure credential store utility confirms it created and initialized the keystore:

   ```
   Successfully created and initialized new KeyStore file: /opt/appdynamics/Controller/.appd.scskeystore
   Verification - New KeyStore file: /opt/appdynamics/Controller/.appd.scskeystore is properly initialized.
   ```

Update the Controller with the new Secure Credential Store Password

1. Shut down the Controller.
2. Obfuscate the password you used to initialize the secure credential keystore:

   ```
   /controller/jre8/bin/java -jar <controller_home>/tools/lib/scs-tool.jar obfuscate -plaintext
   '<Secure_Credential_Store_Password>'
   ```

For example:

```
/controller/jre8/bin/java -jar /opt/appdynamics/Platform/controller/tools/lib/scs-tool.jar obfuscate -
plaintext 'MyCredentialStorePassword'
```

The secure credential store utility writes out an obfuscated password for use in the Controller configuration. For example:

```
s_gsnwR6+LDch8JBf1RamiBoWfMvjjipkrtJMZXAYEkw8=
```

3. Log in as the root user:

```
<controller_home>/bin/controller.sh login-db
```

> On Windows, use controller.bat.

4. Update the secure credential keystore password to the newly obfuscated password:

```
UPDATE global_configuration_cluster
SET value = '<obfuscated_secure_credential_keystore_password>'
WHERE name = 'scs.keystore.password';
```

Update the Account Access Key

1. Log in as the root user:

```
<controller_home>/bin/controller.sh login-db
```

> On Windows, use controller.bat.

2. Update the account access key for the account to the plain text string. When the Controller starts, it will encrypt the account access key:

```
UPDATE account
SET access_key = '<plain_text_account_access_key>',
    encryption_scheme = NULL
WHERE id = <account_id>;
```

> You can get the account id by running the following query: `select id account_id,name account_name,access_key, encryption_scheme from account;`

3. Only if you changed the plain text value of the account access key. Update the account access key for the agent users:

```
UPDATE user
SET encrypted_password = SHA1('<plain_text_account_access_key>')
WHERE account_id = <account_id>
AND name = 'singularity-agent';
```

If you changed the plain text value of the account access key, you need to update the access key for all the agents.

> The access key belongs to the "customer1" account in a single-tenant Controller and the "default" account in a multi-tenant Controller. In addition, `account_id` is the account id of the "customer1" account in a single-tenant Controller and the "default" account in a multi-tenant Controller.

4. If you have default license rules, update the account access key using `v1_license_rules` API.

> ⚠ For earlier Controller versions, you must use browser tools to migrate license rules.

Update the Account Access Key for the System Account

1. Generate the new access key for the system account:

```
../jre/1.8.0_152/bin/java -jar ./tools/lib/scs-tool.jar encrypt -filename ./.appd.scskeystore -storepass
'REPLACE_TO_NOT_OBFUSCATED_STOREPASS_VALUE' -plaintext 'NEW_SYSTEM_ACCOUNT_ACCESS_KEY'
```

2. Once you have generated the system account access key:
    a. Edit the `controller-info.xml` file to add your specific information:

```
<controller-dir>/appserver/glassfish/domains/domain1/appagent/ver4.X.X.X/conf/controller-info.xml
```

    b. Edit the `credential-store-password` value with the obfuscated `storepass` value.
    c. Edit the `account-access-key` with new encrypted access key value.
    d. Run SQL:

```
update account set access_key='ENCRYPTED_SYSTEM_ACCOUNT_ACCESS_KEY' where id=1; update mds_account.
account set access_key='ENCRYPTED_SYSTEM_ACCOUNT_ACCESS_KEY' where id='00000000-0000-0000-0000-
000000000001'; update mds_account.account set access_key='ENCRYPTED_SYSTEM_ACCOUNT_ACCESS_KEY'
where id='00000000-0000-0000-0000-000000000002';
```

    e. Stop `appserver`.
    f. Start `appserver`.

> ⓘ If you use LDAP, DBmon, or HTTP Request Actions and Templates, then you must also reconfigure those components with the same passwords to ensure that they are encrypted with new SCS key.

Restart the Controller and Update Passwords

1. Restart the Controller.
2. Log in to the Controller as a user with the following permissions:
    - Administer users, groups, roles, authentication, etc.
    - Configure Email / SMS.
3. As necessary, re-enter the following passwords:
    - LDAP authentication user password. See Configure Authentication Using LDAP.
    - Database collector credentials:
        - database user password. See "Add a Database Collector" on Configure Database Collectors.
        - machine user password. See "Configure the Database Agent to Monitor Server Hardware" on Configure Database Collectors.
    - SMTP server / Email password. See Enable an Email Server.

Mutual Authentication

In addition to implementing Server Authentication, you can also implement mutual (client and server) authentication. Client authentication enables the Controller to ensure that only authorized and verified agents can establish connections. These procedures outline the workflow to implement mutual authentication.

Before Starting

- Theses agents support client authentication:
  - Java Agent
  - Database Agent
  - Standalone Machine Agent
  - .NET Agent for Windows
- It is good practice to set up and verify client authentication on one agent first. After you confirm that client authentication is working for that agent, proceed with configuring additional agents.
- If you have a "hybrid" environment, with Server Authentication only for some agents and Server and Client Authentication for others, you might want to set up and configure multiple HTTP Listeners in Glassfish: one for Server Authentication only, and another for both Server and Client Authentication.
- The procedures described on this page use the default key and keystore password (`changeit`) for the keystore. Before you proceed with this workflow, it is good practice to:
  1. Change this default password, as described in "Change Keystore Password" under Controller SSL and Certificates.
  2. Use the new password when you perform these procedures.
- Instead of using plain text passwords in the procedures, you can specify encrypted or obscured passwords as described in Encrypt Agent Credentials.

Set Up Client Authentication

These steps describe how to set up Client-based Authentication:

1. Set up server authentication on the Controller.
2. Set up server authentication on agents.
3. Set up a client keystore on the Controller.
4. Configure agents to access the client keystore on the Controller.
5. Enable client authentication on the Controller.

Set Up Server Authentication on the Controller

1. Open a CLI window on the Controller host and `cd` to the `<controller-keystore-home>` directory: `<controller-home>/appserver/glassfish/domains/domain1/config`
2. Create a trusted root certificate store to preserve the Controller public key. The following command exports the public key (`s1as`) and stores it in the new certificate store file `server.cer`.

```
<java-home>/bin/keytool -export -alias s1as -file server.cer \
                            -keystore keystore.jks
                            -storepass changeit
```

3. (*Optional*) To view information about the public and private keys in `keystore.jks`, enter the following command:

```
<java-home>/bin/keytool -list -v -alias s1as  \
                        -keystore ./keystore.jks \
                        -storepass  changeit
```

4. Import the Controller public key from `server.cer` into an agent keystore. The following command creates a new keystore (`agent-truststore.jks`); the Controller sends the public key from this keystore to agents when they set up an SSL connection.

```
<java-home>/bin/keytool -import -v -alias controller_alias -file server.cer \
                        -keystore agent_truststore.jks \
                        -storepass changeit
```

5. This command displays the certificate and asks if you trust it. Answer **yes**.

Set Up Server Authentication on Agents

For each authorized agent:

1. Copy the `agent_truststore.jks` file from the Controller to the root directory of the agent (`<machine-agent-home>` or `<java-agent-home>` or `<database-agent-home>`).
2. For each authorized agent, specify the following properties in the `<agent-home>/conf/controller-info.xml` file as follows:
    - `<controller-ssl-enabled>true</controller-ssl-enabled>`
    - `<controller-port>443</controller-port>`
    - `<controller-keystore-filename>agent_truststore.jks</controller-keystore-filename>`
    - `<controller-keystore-password>changeit</controller-keystore-password>`

Set Up a Client Keystore on the Controller

In this procedure, you create a signed certificate and import it into the client keystore. These steps use the Controller to sign the certificate, but you can also use a third-party Certificate Authority (CA).

1. (*Optional*) To view information about the public and private key in the Controller keystore, enter:

```
<java-home>/bin/keytool -list -v -alias s1as  \
                        -keystore ./keystore.jks \
                        -storepass changeit
```

2. Create a keystore (`clientkeystore.jks`) that includes the Controller public/private keypair. In `<controller-keystore-home>`, enter:

```
<java-home>/bin/keytool -genkey -alias client-alias -keyalg RSA \
                        -keystore clientkeystore.jks \
                        -storepass changeit \
                        -keypass changeit
```

   The keytool prompts you for your name, organization, and other information it needs to generate the key. AppDynamics App Agents use SunX509 as the default keystore factory algorithm. If keystores in your environment use something other than SunX509, you need to specify the algorithm to the App agent. You can do so using the system property `appdynamics.agent.ssl.keymanager.factory.algorithm`. For example, to set the algorithm to PKIX, add this to the startup command of the agent-monitored JVM:

```
-Dappdynamics.agent.ssl.keymanager.factory.algorithm=PKIX
```

3. Generate a certificate signing request (`client.csr`) that can be signed by a Certificate Authority (CA).

```
<java-home>/bin/keytool -certreq -v -alias client-alias -file client.csr \
                        -keystore clientkeystore.jks \
                        -storepass changeit \
                        -keypass changeit
```

4. Get the request (`client.csr`) signed by a trusted CA. This command uses the Controller as a CA, which creates a new file (`signedClient.cer`) with the Controller-signed certificate.

```
<java-home>/bin/keytool -gencert -infile ./client.csr -outfile signedClient.cer -alias s1as \
                        -keystore ./keystore.jks \
                        -storepass changeit \
                        -keypass changeit
```

5. (*Optional*) To view information about the signed certificate, enter:

```
<java-home>/bin/keytool -printcert -v -file ./signedClient.cer
```

6. Verify that the certificate is signed by the authentic Certificate Authority. You can:
    - Copy the public key of the signing authority into the trusted root set, or
    - Import the public key of the signing authority into the client keystore.

   This command does the latter by importing the Controller public key from `server.cer` into `clientkeystore.jks`.

```
<java-home>/bin/keytool -import -v -alias controller_alias -file server.cer \
                        -keystore clientkeystore.jks \
                        -storepass changeit
```

   This command asks if you trust the certificate; when you enter `yes`, this message should display:

```
        Certificate was added to keystore
        [Storing clientkeystore.jks]
```

7. (*Optional*) To view the contents of `clientkeystore.jks`, enter:

```
<java-home>/bin/keytool -list -v -keystore clientkeystore.jks -storepass changeit
```

   The keystore should show entries for `controller-alias` and `client-alias` (which is still unsigned).
8. Import the signed public key certificate into the client keystore. This command imports `signedClient.cer` into `clientkeystore.jks`.

```
<java-home>/bin/keytool -importcert -v -alias client-alias -file ./signedClient.cer \
                        -keystore clientkeystore.jks \
                        -storepass changeit \
                        -keypass changeit
```

   You now have a password-protected `clientkeystore.jks` file on the Controller with a signed certificate that verifies the Controller's authenticity.
9. Verify that the trusted root certificate on the Controller includes the public key of the signing authority. This procedure used the Controller as the Certificate Authority, so the public key is already included. To verify, enter:

```
<java-home>/bin/keytool -list -v -alias client-alias \
                        -keystore clientkeystore.jks -storepass changeit
```

   The public key of the signing authority should now be part of the agent's public key certificate.

Configure Agents to Access the Client Keystore on the Controller

For each authorized agent:

1. Copy the `clientkeystore.jks` file from the Controller to the following directory on the agent:
   - Database agent: `<database agent home>/conf`
   - Java agent: `<java-agent-home>/conf`
   - Machine Agent: `<machine-agent-home>`
2. Specify these properties in the `<agent-home>/conf/controller-info.xml` file as follows:

   - `<use-ssl-client-auth>`**`true`**`</use-ssl-client-auth>`
   - `<asymmetric-keystore-filename>`**`clientkeystore.jks`**`</asymmetric-keystore-filename>`
   - `<asymmetric-keystore-password>`**`changeit`**`</asymmetric-keystore-password>`
   - `<asymmetric-key-password>`**`changeit`**`</asymmetric-key-password>`
   - `<asymmetric-key-alias>`**`client-alias`**`</asymmetric-key-alias>`

Enable Client Authentication on the Controller

From the Controller:

1. If it is not currently running, start the Controller.
2. Open a CLI panel and `cd` to `<controller-home>/appserver/glassfish/bin`.
3. To start the AppServer Admin tool, enter: `./asadmin`
   You should now see the asadmin prompt: `asadmin>`
4. Enter: `list-http-listeners`
   This lists the available set of HTTP listeners. For example:
   ```
   http-listener-1
   http-listener-2
   admin-listener
   ```
5. Configure one of these listeners by running the following commands. In this example, we are configuring `http-listener-2`:

```
set configs.config.server-config.network-config.protocols.protocol.http-listener-2.ssl.key-
store=keystore.jks
set configs.config.server-config.network-config.protocols.protocol.http-listener-2.ssl.client-auth-
enabled=true
set configs.config.server-config.network-config.protocols.protocol.http-listener-2.ssl.trust-
store=cacerts.jks
```

6. To verify that the properties are set correctly, run the following command. Again, this example assumes `http-listener-2`.

```
 get configs.config.server-config.network-config.protocols.protocol.http-listener-2.*
```

7. Restart the Controller.

Upgrade Platform Components

The information and instructions below are intended for On-Premise deployments only. SaaS deployments are managed by AppDynamics.

Related pages:

- Upgrade the Controller Using the Enterprise Console
- Upgrade the Events Service
- Upgrade the Production EUM Server
- Upgrade the Controller Using the Enterprise Console
- Discovery and Upgrade Quick Start

Before Upgrading

Before you upgrade to a newer version of the platform, complete the following tasks:

- Review the Product Announcements and Alerts page.
- Review the latest Release Notes, as well as the release notes for intermediate versions between the current version and the version you are upgrading to.
- Review the compatibility support page.
- Verify that you meet the requirements described in the Platform Requirements for the components you use.

Upgrade Order

Follow the instructions for each service in this order:

1. Upgrade the Enterprise Console
2. Upgrade the Events Service
3. Upgrade the EUM Server
4. Upgrade the Controller

Discover Existing Components

Related pages:

- Discovery and Upgrade Quick Start
- Administer the Enterprise Console

You can use the Enterprise Console to discover and upgrade existing AppDynamics components, such as a Controller or Events Service. After you discover the components, they can be added to the platform and be managed by the application. This process can be performed with the GUI or command line. You can discover Controllers that are version 4.1 or later.

Before you discover existing components, you need the following information:

Controller

- Controller root user password
- Installation directory
- Database password
- Host name or IP address

Events Service

- Host names or IP addresses
- Installation directory. Note that if you have an Events Service cluster, this directory is the same on every node.
- Host credentials (SSH key file and username)

Using the GUI

You can use the Custom Install Discover & Upgrade option in the GUI to create a platform and discover a Controller and Events Service. Alternatively, if you have already created a platform, you must add credentials and hosts to the platform before you can perform discovery. Then, discover the Controller or Events Service on the page for that component. For more information about how to add credentials and hosts, see Administer the Enterprise Console.

When the Enterprise Console discovers a component, it also checks to see if an upgrade is available and performs the upgrade.

Using the Command Line

Controller

Use the discover-upgrade-controller command to discover and upgrade a Controller:

```
bin/platform-admin.sh discover-upgrade-controller --host <host> --controller-root-password <root user password
for Controller> --installation-dir <Controller installation directory> --db-root-password <password for
Controller database>
```

If your upgrade fails, you can resume by passing the flag `useCheckpoint=true` as an argument after `--args.`

Events Service

The discover-events-service command can discover and upgrade an Events Service.

```
bin/platform-admin.sh discover-events-service --hosts <host 1> <host 2> <host 3> --installation-dir <Events
Service installation directory>
```

Instead of listing each host, you can specify a line-separated list in a text file:

```
bin/platform-admin.sh discover-events-service -n <file path for the host file> --installation-dir <Events
Service installation directory>
```

If your upgrade fails, you can resume by passing the flag `useCheckpoint=true` as an argument after `--args.`

Extend AppDynamics

This page describes how to extend and customize the AppDynamics Application Performance Management (APM) Platform.

The AppDynamics Community Exchange includes many pre-built extensions you can use to customize AppDynamics and integrate with other systems.

You can also use AppDynamics REST APIs to create custom integrations and automation tasks.

AppDynamics REST APIs

- AppDynamics APIs
- Using the Controller APIs

AppDynamics Extensions

- Integration Modules
- Integrate AppDynamics with Splunk
- Integrate AppDynamics with Scalyr

AppDynamics APIs

Related pages:

- Extensions and Custom Metrics
- AppDynamics Exchange

This page provides an overview of AppDynamics APIs, which let you extend and customize various aspects of the AppDynamics Application Performance Monitoring (APM) Platform.

Overview of AppDynamics APIs

The AppDynamics APM Platform exposes various APIs for customizing and extending its features on the platform-side, which are served by the Controller and Events Service, and on the agent-side.

The AppDynamics platform server components and agents offer the following APIs:

- Controller APIs: Administer the Controller, configure, monitor, query metrics, and more. See the Platform API Index
- Accounts APIs: Manage and monitor accounts, users, and other aspects of AppDynamics licensing. Accounts APIs are made up of these modules:
    - actionsuppressions
    - businesstransactions
    - healthrules
    - licensemodules
    - mdsconfig
    - nodes
    - policies

  See License Management.
- Analytics Events API: Send custom analytics events from your data sources to the Events Service. See the Analytics Events API section in the Platform API Index.
- Standalone Machine Agent APIs: HTTP APIs available at the machine agent for uploading custom metrics. See Machine Agent HTTP Listener.
- Database Agent APIs: Get, create, update, and delete Database Monitoring database Collectors. See Database Visibility API.
- Application Agent Instrumentation APIs: Control and customize transaction detection and correlation, along with exit point detection. Agent APIs include:
    - PHP Agent API
    - Python Agent API
    - Node.js Agent API Reference
    - C/C++ SDK
- Java Agent API: Customize agent instrumentation. See the SDK folder in the agent home directory.
- Mobile RUM: Instrument mobile applications for real user performance monitoring. See Instrument iOS Applications.

Platform API Index

These methods are Below is a list of all the methods in the AppDynamics Controller and Events Service APIs:

- **Accounts API**
    - Retrieve Controller Audit History
    - Configure Metric Retention by Account
    - Configure Metric Retention by Application
- **Application Model API**
    - Retrieve All Business Applications
    - Retrieve All Business Transactions in a Business Application
    - Retrieve All Tiers in a Business Application
    - Retrieve All Registered Backends in a Business Application with Their Properties
    - Retrieve Node Information for All Nodes in a Business Application
    - Retrieve Node Information by Node Name
    - Retrieve Node Information for All Nodes in a Tier
    - Retrieve Tier Information by Tier Name
- **Metric and Snapshot API**
    - Retrieve Metric Hierarchy
    - Retrieve Metric Data
    - Retrieve Transaction Snapshots
- **Alert and Respond APIs**
    - **Health Rule API**
        - Create a Health Rule
        - Retrieve a List of Health Rules for an Application
        - Retrieve Details of a Specified Health Rule
        - Update a Health Rule
        - Delete a Health Rule
    - **Schedule API**
        - Create a New Schedule
        - Retrieve a List of Schedules for a Given Application
        - Retrieve the Details of a Specified Schedule
        - Update a Schedule
        - Delete a Schedule

    - Create Role
    - Add Role to User
    - Remove Role from User
    - Add Role to Group
    - Remove Role from Group
    - Get Role by ID
    - Get Role by Name
    - Get All Roles
    - Update Role
    - Delete Role
- **License Rules API**
    - Creates a New License Rule
    - Returns a Summary of All License Rules for the Current Account
    - Updates a License Rule
    - Deletes a License Rule
    - Retrieve a License Rule via its Id
    - Retrieves a License Rule by Access Key
    - Retrieve a License Rule by Name

API Clients

This page describes how to create and use the API Clients identity type to provide secure access to the Controller through AppDynamics Controller REST API calls. These calls use Open Authorization (OAuth) token-based authentication.

Open Authorization (OAuth) Mechanisms

OAuth is an open protocol to allow secure authorization in a simple and standard method from web, mobile, and desktop applications. See https://oauth.net/

It acts as the intermediary on your behalf by providing third-party applications with an access token that authorizes sharing specific account information. Using the OAuth protocol with AppDynamics Controller REST APIs is the best way to securely grant access to your Controller information.

The OAuth authentication process authenticates a request token and uses it to obtain an encrypted access token from your Controller. Once the access token is available, you can use it to make requests to your Controller until the token expires or is revoked.

The tokens are based on JSON Web Tokens (JWT) authentication format, which is the industry standard RFC 7519 method for representing claims securely between two parties.

Accessing Authentication Provider Settings

Users with the **Account Owner** role or the **Administer users, groups, roles ...** permission can view API Clients settings in the **Settings** > **Administration** page.

Creating API Clients

You can create new API Client identity types that can be used to generate OAuth tokens.

1. Log in to the Controller UI as an **Account Owner** or other roles with the **Administer users, groups, roles ...** permission.
2. Click  (gear icon) > **Administration**.
3. Click the API Clients tab to view the list of existing clients.
4. Click **+ Create**.
5. Enter the Client Name and Description.
6. Click **Generate Secret** to populate the Client Secret.
   This will generate a UUID as the secret of the API Client.

   This API Client secret acts as a password. It does not generate the authentication token.

7. Set the Default API-generated Token Expiration. This expiration only applies to authentication tokens generated through the `/controller/api/oauth/access_token` REST API, not to Temporary Access Tokens generated from the UI. See Using the Access Token.

   Every API-generated access token has an expiration. Although the default is 5 minutes, you can set it to any second, minute, or hour limit. The Default API-generated Token has a shorter expiration than the authentication token generated through the Administration UI.

8. Add the **Roles** you would like to associate with this API Client. You can add or remove roles at any time. See Roles and Permissions.

   The REST APIs will use the identity which the access token represents to pull up RBAC permissions and check those permissions at the underlying API level.

9. Click **Save** at the top right.

Using the Access Token

You can generate and use the access token for each API access call into your Controller by generating the token through one of the following means:

- Administration UI: These tokens usually have a longer expiration. The account administrator can generate and distribute to parties/teams who need Controller access, but do not want to refresh such tokens frequently.
- API: These tokens usually have a relatively short expiration. The program generates and refreshes regularly before expiration. These tokens are visible from the UI and are not individually tracked and managed.

Generate the Token Through the UI

When you are ready to use the access token, you will generate it through the Administration UI:

1. Click **Generate Temporary Access Token** when you are ready to use the access token.

   > Remember that temporary token expirations are usually longer than the Default API-generated Token Expiration.

2. Copy and paste the access token from the **Temporary Access Token** field into the TOKEN portion of your curl command. For example:

   ```
   curl -H "Authorization:Bearer <AUTH_TOKEN>" "http://master-onprem-controller.e2e.appd-test.com:8090
   /controller/rest/applications"
   ```

3. Check that your call is successful. It should look like this:

Generate the Token Through an API

You can use REST APIs to generate a short-lived access token. This default, on-demand token is not tracked on the UI.

Use POST `/controller/api/oauth/access_token`. You will need the API Client name and Client Secret to generate the token. For example:

```
curl -X POST -H “Content-Type: application/vnd.appd.cntrl+protobuf;v=1” “https://<controller address>/controller
/api/oauth/access_token” -d
‘grant_type=client_credentials&client_id=<apiClientName>@<accountName>&client_secret=10d5-573e-4a75-8396-
afa006fd8f19’
```

> Be aware that:
>
> - For Controllers < 4.5.9, an authentication header is required in the request as shown below:
>
> ```
> curl -X POST -H "Content-Type: application/vnd.appd.cntrl+protobuf;v=1" -u 'TestApiClient@customer:
> face10d5-573e-4a75-8396-afa006fd8f19' "https://<controller address>/controller/api/oauth/access_token" -
> d 'grant_type=client_credentials&client_id=TestApiClient@customer&client_secret=face10d5-573e-4a75-8396-
> afa006fd8f19'
>
> {
> "access_token": "eyJraWQiOiIxIiwiYWxnIjoiSFMyNTYifQ.
> eyJpc3MiOiJBcHBEeW5hbWljcyIsImF1ZCI6IkFwcERfQVBJcyIsImV4cCI6MTUzNjI3MjI1NiwianRpIjoiT2twZHVDdmduSDdwMmdu
> Mnc4MFRtQSIsImlhdCI6MTUzNjI3MTk1NiwibmJmIjoxNTM2MjcxODM2LCJzdWIiOiJUZXN0QXBpQ2xpZW50IiwidHlwZSI6IkFQSV9D
> TElFTlQiLCJpZCI6ImFmZTQxMzg5LTJlNjMtNDQyYS1hY2U2LTEyYzU5NGFlOGM2OCIsImFjY3RJZCI6IjhlMGQ3NjI1LTY4YzEtNDE4
> Mi1hMmFmLWFhMTY1MzllZDg0OCIsImFjY3ROYW1lIjoiZTJlLWN1c3RvbWVyIn0.
> 95EyDNV5muTN_4zGOXIPQQHOdVDSiEynKSgk08UHlh0",
> "expires_in": 300
> }
> ```
>
> - The token generation API requires an authentication header with username and password and the oAuth request body to successfully request a token.

Use the generated token to list all applications. For example:

```
curl -H "Authorization:Bearer <AUTH_TOKEN>" https://master-controller.e2e.appd.com/controller/rest/applications
```

- In the case of success: all applications are returned.
- In the case of failure: `HTTP 401 Unauthorized "Failed to authenticate: invalid access token."` is returned.

Managing Access Tokens

Access tokens are based on JWT, therefore decoding them will not show sensitive information.

However, if you believe that your token has become compromised, you can revoke it by clicking **Revoke** or by deleting the API Client to invalidate the token. Calls using revoked access tokens fail to authenticate with a 401 Unauthorized error HTTP status code. You can click **Regenerate** to refresh a token.

Regenerated tokens do not disable older tokens. The older tokens will remain active until they expire.

When you regenerate a token, you can set the Temporary Token Expiration.

The default value is 1 day and the maximum limit is 30 days.

There is no way to retrieve previous or currently valid tokens. Therefore, only the current token can be revoked.

Also, API generated tokens that have Default API-generated Token Expiration, cannot be viewed nor revoked through the UI or REST API.

Using the Controller APIs

This page describes API usage information. The Controller APIs are served by the Controller instance, rather than by the Events Service or an agent component. They include:

- Accounts API
- Application Model API
- Metric and Snapshot API
- Alert and Respond API
- Configuration API
- Configuration Import and Export APIs
- Analytics Events API

Controller API Base URI

Except as indicated in the format listing for a particular method, URIs in the Controller API use the following base URI:

```
http://<controller_host>:<controller_port>/controller/rest/<REST_URI>
```

The port that serves the API is the same primary port for the Controller used by Controller UI and agents.

Retrieving Data in JSON Format

The AppDynamics Controller APIs return data in eXtensible Markup Language (XML) or JavaScript Object Notation (JSON). The default output format is XML.

Any Controller API with a URI in the `/controller/rest/` format shown in Controller API Base URI can return data in JSON format.

To retrieve data in JSON, call the API with the output query parameter set to JSON, as follows:

```
curl --user user1@customer1:secret http://demo.appdynamics.com/controller/rest/applications?output=JSON
[
    {
    "description": "",
    "id": 5,
    "name": "ECommerce_E2E"
  },
    {
    "description": "",
    "id": 8,
    "name": "ECommerce_E2E-Fulfillment"
  },
]
```

You can specify JSON output format for any of the Controller APIs.

When a client uses HTTP 1.1 and accepts gzip content-encoding, the Controller returns JSON responses using `gzip` compression.

Authentication

You can use OAuth identity types for authentication. See API Clients.

To invoke the REST APIs using basic HTTP authentication, you must provide the authentication credentials as well as your account information. These are:

- Account: the AppDynamics tenant account name
- Username: a user in that account
- Password: the password for that account

Pass the credentials in the following form:

```
<your_username>@<your_accountname>:<your_password>
```

Most on-premises Controllers are single-tenant Controllers that use `customer1` as the primary default account name. The account name should be left as default. For example:

```
<your_username>@customer1:<your_password>
```

Most SaaS Controllers are multi-tenant Controllers and allow you to replace `customer1` with your own, instance-specific account name. See License Management.

Invalid Characters for Usernames and Passwords

REST API calls will not authenticate usernames and passwords that contain these characters:

\ / " [ ] : | < > + = ; , ? * , ' % tab space @

If you have already created user credentials that contain any of the disallowed characters, such as "user:customer66", create new credentials without the disallowed character for the purpose of accessing the REST APIs.

For usernames or passwords containing the "@" symbol, URL encode the "@" character as %40.

Copying a Metric URL in the Metric Browser

1. Right-click a metric in the **Metric Browser**.
2. Copy the full REST URL of the metric.
3. Paste the REST URL into your code or onto the command line.

For security reasons, AppDynamics only supports making API calls programmatically or at the command line. Do not attempt to paste the REST URL into a browser.

Application Model API

This page describes how application APIs let you retrieve information about the monitored environment as modeled in AppDynamics. This information includes, for example, the names and IDs of the business applications, business transactions, tiers, and nodes in the modeled environment.

Retrieve All Business Applications

The application API method returns the business application names and internal numeric identifier. Many of the operations in the Controller APIs occur in the context of a business application. Use this method to discover the application names or IDs to use before invoking other methods.

Format

```
GET /controller/rest/applications
```

Input parameters

Parameter Name	Parameter Type	Value	Mandatory
`output`	Query	HTTP Request parameter included as part of the URL to change the output format. Valid values are XML (default) or `JSON`.	No
`time-range-type`	Time	A time parameter to filter data based on time range options (`time-range-type`, `startTime`, `endTime`). If a time-range-type option is specified, the query returns alive applications on that GMT day, otherwise, the query returns all applications. • Case 1 : If `time-range-type` is last T mins and the `time-range-type` just falls into one GMT day then the API returns all the applications which are alive on that GMT day. • Case 2 : If `time-range-type` is last T mins and the `time-range-type` falls into 2 GMT days (for example if the current time is 4:05 PST and `time-range-type` specified is last 10 mins then the API returns applications which are alive on this and the previous GMT day). This feature is available for SaaS only and the API returns all applications for on-premises. For more information see Metric and Snapshot API.	No

An *alive applicatio*n is an application with at least one node that submits at least one metric to the Controller in the provided time range.

Example

```
curl --user user1@customer1:your_password http://demo.appdynamics.com/controller/rest/applications

<applications>
   <application>
     <id>5</id>
     <name>ECommerce_E2E</name>
   </application>
   <application>
     <id>8</id>
     <name>ECommerce_E2E-Fulfillment</name>
   </application>
   <application>
     <id>11</id>
     <name>jimix12110919</name>
     <description></description>
     <accountGuid>429c7884-3f36-4b5a-9412-fdf827e6c86e</accountGuid>
   </application>
</applications>
```

Retrieve All Business Transactions in a Business Application

Format

```
GET /controller/rest/applications/application_name/business-transactions
```

Input parameters

Parameter Name	Parameter Type	Value	Mandatory
`application_name`	URI	The application name or application ID.	Yes
`exclude`	Query	• `false`: the query retrieves only the business transactions that are included for monitoring. • `true`: the query retrieves only the excluded business transactions. Excluded business transactions have been configured to be excluded from monitoring either from the UI or through the REST interface. • The default is `false`.	No
`output`	Query	HTTP Request parameter included as part of the URL to change the output format. Valid values are `XML` (default) or `JSON`.	No
`time-range-type`	Time	Time parameter to filter data based on time range options (`time-range-type`, `startTime`, `endTime`). If a `time-range-type` option is specified, the query returns alive business transactions on that GMT day, otherwise, the query returns all business transactions. • Case 1 : If `time-range-type` is last T mins and the `time-range-type` just falls into one GMT day then the API returns all the business transactions which are alive on that GMT day. • Case 2 : If `time-range-type` is last T mins and the `time-range-type` falls into 2 GMT days (for example if the current time is 4:05 PST and `time-range-type` specified is last 10 mins then the API returns business transactions which are alive on this and the previous GMT day). This feature is available for SaaS only and the API returns all business transactions for on-premises. See Metric and Snapshot API.	No

An *alive business transaction* is a transaction that submits at least one metric to the Controller in the provided time range.

Example

```
curl --user user1@customer1:your_password http://demo.appdynamics.com/controller/rest/applications/5/business-transactions

<business-transactions>
   <business-transaction>
     <id>92</id>
     <name>/user/.POST</name>
     <entryPointType>WEB_SERVICE</entryPointType>
     <internalName>/user/.POST</internalName>
     <tierId>9</tierId>
     <tierName>ECommerce-Services</tierName>
     <background>false</background>
   </business-transaction>
   ...
   <business-transaction>
     <id>184</id>
     <name>OrderServiceImplService.createOrder</name>
     <entryPointType>WEB_SERVICE</entryPointType>
     <internalName>OrderServiceImplService.createOrder</internalName>
     <tierId>12</tierId>
     <tierName>Inventory-Services</tierName>
     <background>false</background>
   </business-transaction>
</business-transactions>
```

Retrieve All Tiers in a Business Application

Format

```
GET /controller/rest/applications/application_name/tiers
```

Input parameters

Parameter Name	Parameter Type	Value	Mandatory

`application_name`	URI	The application name or application ID.	Yes
`output`	Query	HTTP Request parameter included as part of the URL to change the output format. Valid values are `XML` (default) or `JSON`.	No
`time-range-type`	Time	Time parameter to filter data based on time range options (`time-range-type`, `startTime`, `endTime`). If a `time-range-type` option is specified, the query returns alive tiers on that GMT day, otherwise, the query returns all tiers. • Case 1 : If `time-range-type` is last T mins and the `time-range-type` just falls into one GMT day then the API returns all the tiers which are alive on that GMT day. • Case 2 : If `time-range-type` is last T mins and the `time-range-type` falls into 2 GMT days (for example if the current time is 4:05 PST and `time-range-type` specified is last 10 mins then the API returns tiers which are alive on this and the previous GMT day). This feature is available for SaaS only and the API returns all tiers for on-premises. See Metric and Snapshot API.	No

An *alive tier* is a tier with at least one node in this tier that submits at least one metric to the Controller in the provided time range.

Example

```
curl --user user1@customer1:your_password http://demo.appdynamics.com/controller/rest/applications/5/tiers

<tiers>
<tier>
  <id>8</id>
  <name>Address-Services</name>
  <type>Application Server</type>
  <agentType>APP_AGENT</agentType>
  <numberOfNodes>1</numberOfNodes>
</tier>
<tier>
  <id>16</id>
  <name>Customer-Survey-Services</name>
  <type>Application Server</type>
  <agentType>APP_AGENT</agentType>
  <numberOfNodes>1</numberOfNodes>
</tier>
<tier>
  <id>9</id>
  <name>ECommerce-Services</name>
  <type>Application Server</type>
  <agentType>APP_AGENT</agentType>
  <numberOfNodes>2</numberOfNodes>
</tier>
<tier>
  <id>12</id>
  <name>Inventory-Services</name>
  <type>Application Server</type>
  <agentType>APP_AGENT</agentType>
  <numberOfNodes>1</numberOfNodes>
</tier>
<tier>
  <id>17</id>
  <name>Order-Processing-Services</name>
  <type>Application Server</type>
  <agentType>APP_AGENT</agentType>
  <numberOfNodes>1</numberOfNodes>
</tier>
<tier>
  <id>18</id>
  <name>Web-Tier-Services</name>
  <type>Web Server</type>
  <agentType>NATIVE_WEB_SERVER</agentType>
  <numberOfNodes>1</numberOfNodes>
</tier>
</tiers>
```

Retrieve All Registered Backends in a Business Application With Their Properties

Format

```
GET /controller/rest/applications/application_name/backends
```

Input parameters

Parameter Name	Parameter Type	Value	Mandatory
`application_name`	URI	Provide either the application name or application id.	Yes
`output`	Query	HTTP Request parameter included as part of the URL to change the output format. Valid values are `XML` (default) or `JSON`.	No

Example

```
curl --user user1@customer1:your_password http://demo.appdynamics.com/controller/rest/applications/5/backends

<backends><backend>
  <id>10</id>
  <name>INVENTORY-MySQL DB-DB</name>
  <exitPointType>JDBC</exitPointType>
  <properties>
    <name-value>
      <id>0</id>
      <name>HOST</name>
      <value>DB</value>
    </name-value>
    <name-value>
      <id>0</id>
      <name>MAJOR_VERSION</name>
      <value>5.5.44-0ubuntu0.14.04.1</value>
    </name-value>
    <name-value>
      <id>0</id>
      <name>PORT</name>
      <value>3306</value>
    </name-value>
    <name-value>
      <id>0</id>
      <name>SCHEMA</name>
      <value>INVENTORY</value>
    </name-value>
    <name-value>
      <id>0</id>
      <name>URL</name>
      <value>jdbc:mysql://db:3306/inventory?useUnicode=true&amp;characterEncoding=UTF-8&amp;autoReconnect=true<
/value>
    </name-value>
    <name-value>
      <id>0</id>
      <name>VENDOR</name>
      <value>MySQL DB</value>
    </name-value>
  </properties>
  <applicationComponentNodeId>0</applicationComponentNodeId>
  <tierId>0</tierId>
</backend>
 ...
<backend>
  <id>14</id>
  <name>Active MQ-OrderQueue</name>
  <exitPointType>JMS</exitPointType>
  <properties>
    <name-value>
      <id>0</id>
      <name>DESTINATION_NAME</name>
      <value>OrderQueue</value>
    </name-value>
    <name-value>
      <id>0</id>
      <name>DESTINATION_TYPE</name>
      <value>QUEUE</value>
    </name-value>
    <name-value>
      <id>0</id>
      <name>VENDOR</name>
      <value>Active MQ</value>
    </name-value>
  </properties>
  <applicationComponentNodeId>0</applicationComponentNodeId>
  <tierId>0</tierId>
</backend>
</backends>
```

Retrieve Node Information for All Nodes in a Business Application

Format

```
GET /controller/rest/applications/application_name/nodes
```

Input parameters

Parameter Name	Parameter Type	Value	Mandatory
`application_name`	URI	Provide either the application name or application id.	Yes
`output`	Query	HTTP Request parameter included as part of the URL to change the output format. Valid values are `XML` (default) or `JSON`.	No
`time-range-type`	Time	Time parameter to filter data based on time range options (`time-range-type`, `startTime`, `endTime`). If a `time-range-type` option is specified, the query returns alive nodes on that GMT day, otherwise, the query returns all nodes. • Case 1 : If `time-range-type` is last T mins and the `time-range-type` just falls into one GMT day then the API returns all the nodes which are alive on that GMT day. • Case 2 : If `time-range-type` is last T mins and the `time-range-type` falls into 2 GMT days (for example if the current time is 4:05 PST and `time-range-type` specified is last 10 mins then the API returns nodes which are alive on this and the previous GMT day). This feature is available for SaaS only and the API returns all nodes for on-premises. See Metric and Snapshot API.	No

An *alive node* is a node that submits at least one metric to the Controller in the provided time range.

Example

```
curl --user user1@customer1:welcome http://demo.appdynamics.com:8090/controller/rest/applications/5/nodes


<nodes><node>
  <id>7</id>
  <name>Node_8000</name>
  <type>Tomcat 5.x</type>
  <tierId>12</tierId>
  <tierName>ECommerce Server</tierName>
  <machineId>3</machineId>
  <machineName>TIER1TOMCAT</machineName>
  <machineOSType>Linux</machineOSType>
  <machineAgentPresent>true</machineAgentPresent>
  <machineAgentVersion>Machine Agent v4.2.0.0 GA Build Date 2015-12-18 18:47:15</machineAgentVersion>
  <appAgentPresent>true</appAgentPresent>
  <appAgentVersion>Server Agent v4.2.0.0 GA #10145 r514d60d3122bd992e7152820d2ca5fb5ff4e45c1 8409-master-build<
/appAgentVersion>
  <agentType>APP_AGENT</agentType>
</node>
...
<node>
  <id>10</id>
  <name>Node_8002</name>
  <type>Tomcat 5.x</type>
  <tierId>14</tierId>
  <tierName>Inventory Server</tierName>
  <machineId>6</machineId>
  <machineName>TIER3TOMCAT</machineName>
  <machineOSType>Linux</machineOSType>
  <machineAgentPresent>true</machineAgentPresent>
  <machineAgentVersion>Machine Agent v4.2.0.0 GA Build Date 2015-12-18 18:47:15</machineAgentVersion>
  <appAgentPresent>true</appAgentPresent>
  <appAgentVersion>Server Agent v4.2.0.0 GA #10145 r514d60d3122bd992e7152820d2ca5fb5ff4e45c1 8409-master-build<
/appAgentVersion>
  <agentType>APP_AGENT</agentType>
</node>
</nodes>
```

Retrieve Node Information by Node Name

Format

`GET /controller/rest/applications/application_name/nodes/node_name`

Input parameters

Parameter Name	Parameter Type	Value	Mandatory
`application_name`	URI	The application name or application ID.	Yes
`node_name`	URI	The node name or ID	Yes
`output`	Query	HTTP Request parameter included as part of the URL to change the output format. Valid values are `XML` (default) or `JSON`.	No

Example

```
curl --user user1@customer1:welcome http://demo.appdynamics.com:8090/controller/rest/applications/5/nodes/10

<nodes><node>
  <id>10</id>
  <name>Node_8002</name>
  <type>Tomcat 5.x</type>
  <tierId>14</tierId>
  <tierName>Inventory Server</tierName>
  <machineId>6</machineId>
  <machineName>TIER3TOMCAT</machineName>
  <machineOSType>Linux</machineOSType>
  <machineAgentPresent>true</machineAgentPresent>
  <machineAgentVersion>Machine Agent v4.2.0.0 GA Build Date 2015-12-18 18:47:15</machineAgentVersion>
  <appAgentPresent>true</appAgentPresent>
  <appAgentVersion>Server Agent v4.2.0.0 GA #10145 r514d60d3122bd992e7152820d2ca5fb5ff4e45c1 8409-master-build<
/appAgentVersion>
  <ipAddresses>
    <ipAddress>10.0.32.138</ipAddress>
  </ipAddresses>
  <agentType>APP_AGENT</agentType>
</node>
</nodes>
```

Retrieve Node Information for All Nodes in a Tier

Format

`GET /controller/rest/applications/application_name/tiers/tier_name/nodes`

Input parameters

Parameter Name	Parameter Type	Value	Mandatory
`application_name`	URI	The application name or application ID.	Yes
`tier_name`	URI	The tier name or ID.	Yes
`output`	Query	HTTP Request parameter included as part of the URL to change the output format. Valid values are `XML` (default) or `JSON`.	No
`time-range-type`	Time	Time parameter to filter data based on time range options (time-range-type, `startTime`, `endTime`). If a time-range-type option is specified, the query returns alive nodes on that GMT day, otherwise, the query returns all nodes. • Case 1 : If `time-range-type` is last T mins and the `time-range-type` just falls into one GMT day then the API returns all the nodes which are alive on that GMT day. • Case 2 : If `time-range-type` is last T mins and the `time-range-type` falls into 2 GMT days (for example if the current time is 4:05 PST and `time-range-type` specified is last 10 mins then the API returns nodes which are alive on this and the previous GMT day). This feature is available for SaaS only and the API returns all nodes for on-premises. See Metric and Snapshot API.	No

Example

```
curl --user user1@customer1:welcome http://demo.appdynamics.com:8090/controller/rest/applications/25/tiers/70
/nodes

<nodes><node>
  <id>81</id>
  <name>PHP_Node</name>
  <type>Other</type>
  <tierId>70</tierId>
  <tierName>PHP_Tier</tierName>
  <machineId>65</machineId>
  <machineName>232fe50b8f9c</machineName>
  <machineOSType>Linux</machineOSType>
  <machineAgentPresent>false</machineAgentPresent>
  <appAgentPresent>true</appAgentPresent>
  <appAgentVersion>Proxy v4.2.0.0 GA SHA-1:.c86ec090f4ff77195df065fe56dade4dfc3913aa #9909 8869-master-build<
/appAgentVersion>
  <ipAddresses>
    <ipAddress>fe80:0:0:0:42:acff:fe11:2%eth0</ipAddress>
    <ipAddress>172.17.0.2</ipAddress>
  </ipAddresses>
  <agentType>PHP_APP_AGENT</agentType>
</node>
</nodes>
```

Retrieve Tier Information by Tier Name

Format

`GET /controller/rest/applications/application_name/tiers/tier_name`

Input parameters

Parameter Name	Parameter Type	Value	Mandatory
`application_name`	URI	The application name or application ID.	Yes
`tier_name`	URI	Tier name or ID.	Yes
`output`	Query	HTTP Request parameter included as part of the URL to change the output format. Valid values are `XML` (default) or `JSON`.	No

Example

```
curl --user user1@customer1:welcome http://demo.appdynamics.com:8090/controller/rest/applications/5/tiers/14

<tiers><tier>
  <id>14</id>
  <name>Inventory Server</name>
  <type>Application Server</type>
  <agentType>APP_AGENT</agentType>
  <numberOfNodes>1</numberOfNodes>
</tier>
</tiers>
```

Metric and Snapshot API

This page describes the Controller Metrics and Events API methods that allow you to retrieve information on metric data and various types of activities in your monitored environment. You can also configure how long you retain the metrics.

> The AppDynamics Dexter Data Extraction Enhanced Reporting (DEXTER) extension provides an alternative to using a REST client to get metric data by makes AppDynamics data queryable in the manner of a data warehouse.

Retrieve Metric Hierarchy

This API returns information about the metric tree structure. Because it retrieves the first generation of child elements, you can only expand the children of the folder type.

- If a child element is a container item in the response, its type value is `folder`.
- Otherwise, the type value for the child element is `leaf`.

You can recurse the metric tree structure further by using the `metric-path` parameter as described in the Metric Data API.

Format

```
GET /controller/rest/applications/application_name/metrics
```

Input parameters

Parameter Name	Parameter Type	Value	Mandatory
`application_name`	URI	The name or ID of the business or EUM (`browser/mobile/IoT`) application. Use the call to get the application ID in the Application Model API.	Yes
`output`	Query	HTTP Request parameter included as part of the URL to change the output format. Valid values are `XML` (default) or `JSON`.	No
`metric-path`	Query	The path to the metric in the metric hierarchy. > When including the pipe (`|`) or backslash (`\`) special characters in the `metric-path`, you must include an additional backslash to indicate escape.	No

Example

```
curl --user user1@customer1:your_password "http://demo.appdynamics.com/controller/rest/applications
/ECommerce_E2E/metrics"

<metric-items><metric-item>
  <type>folder</type>
  <name>Backends</name>
</metric-item>
<metric-item>
  <type>folder</type>
  <name>Service Endpoints</name>
</metric-item>
<metric-item>
  <type>folder</type>
  <name>End User Experience</name>
</metric-item>
<metric-item>
  <type>folder</type>
  <name>Errors</name>
</metric-item>
<metric-item>
  <type>folder</type>
  <name>Business Transaction Performance</name>
</metric-item>
<metric-item>
  <type>folder</type>
  <name>Information Points</name>
</metric-item>
<metric-item>
  <type>folder</type>
  <name>Overall Application Performance</name>
</metric-item>
<metric-item>
  <type>folder</type>
  <name>Application Infrastructure Performance</name>
</metric-item>
<metric-item>
  <type>folder</type>
  <name>Mobile</name>
</metric-item></metric-items>
```

Retrieve Metric Data

The metric data method lets you get values generated for metrics. To use this method, specify the following parameters to the API:

- The path of the metric to retrieve.
- The time frame for the data.

Using the Controller UI is the simplest way to learn how to construct the metric path and time range-related parameters.

1. Right-click on the metric in the Metric Browser.
2. Select **Copy REST URL**. The copied URL includes the path to this metric and time range selected in the UI.
3. Certain clients can accept and properly encode the full path value as the metric path parameter.
4. Hover over the metric in the tree or copy it using the **Copy Full Path** option in the right-click menu.

Certain examples below are shown with the **full path value** rather than the fully encoded URL value. If you test calls with the full path, avoid using a pipe character at the start or end of the path.

These sections provide additional details and examples for the metric data method:

- Metric Response Values
- Using Wildcards
- Using Time Ranges
- Retrieving All Other Traffic Business Transaction Metrics

Format

`GET /controller/rest/applications/`*`application_name`*`/metric-data`

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory
`metric-path`	Query	The path to the metric in the metric hierarchy. When including the pipe (|) or backslash (\ ) special characters in the `metric-path`, you must include an additional backslash to indicate escape.	Yes
`rollup`	Query	By default, the values of the returned metrics are rolled up into a single data point (`rollup=true`). To get separate results for all values within the time range, set the **Rollup** parameter to `false` in the query.	No

Additional mandatory parameters for specifying time ranges are described in Using Time Ranges.

Example

Retrieve metric values for a metric at an absolute path:

```
curl --user user1@customer1:your_password "http://demo.appdynamics.com/controller/rest/applications
/ECommerce_E2E/metric-data?metric-path=Overall%20Application%20Performance%7CAverage%20Response%20Time%20%28ms%
29&time-range-type=BEFORE_NOW&duration-in-mins=15"

<metric-datas><metric-data>
  <metricId>2339</metricId>
  <metricPath>Overall Application Performance|Average Response Time (ms)</metricPath>
  <metricName>BTM|Application Summary|Average Response Time (ms)</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450562160000</startTimeInMillis>
      <value>302</value>
      <min>0</min>
      <max>15212</max>
      <current>15212</current>
      <sum>97800</sum>
      <count>324</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>true</useRange>
    </metric-value>
  </metricValues>
</metric-data>
</metric-datas>
```

Metric Response Values

Metrics results include the following fields:

Name	Definition
`current`	The value for the current minute. This is used only when the time rollup type used by the Controller is current.
`count`	The number of times the agent collected the metric over the selected time period.
`min, max`	The minimum and maximum values reported across the selected time period. These are not used for all metric types.
`occurrences`	The number of data samples taken by the Controller to calculate the standard deviation.
`standardDeviation`	The intermediate values calculated by the Controller during time rollup used to calculate standard deviation. See Dynamic Baselines.
`startTimeInMillis`	The `startTimeInMillis` is the start time of the time range to which the result metric data applies in UNIX epoch time.
`sum`	The total accumulated value for the metric over the selected time period.
`useRange`	Used internally by the Controller to process the metric.

`value`	The `value` value is one of the following for all metric values reported across the configured evaluation time length: • Arithmetic average: if the metric time rollup type is average. • Sum: if the metric time rollup type is sum. • Latest: if the metric time rollup type is current.

`min` and `max` values are not available for any count-based or sum-based metric except when the metric is rolled up to hourly or daily data points. These metrics include errors per minute, calls per minute, and so on.

Using Wildcards

When you copy the REST URL in the Metric Browser, you get the path to a specific metric within a specific application and tier. Alternatively, you can use wildcard characters in one or more steps in the URL path to get metric data for entities, including multiple business transactions, tiers, or nodes.

The following format examples show where to put wildcard characters in various metric paths to achieve particular results. For reading clarity, these format examples use the **Full Path** for the metric rather than the REST URL. For a full working example, click the expanding link under each format listing:

- Retrieve the **app agent availability time** for all tiers in the application using a wildcard for the tier name:

```
/controller/rest/applications/ECommerce_E2E-Fulfillment/metric-data?metric-path=Application
Infrastructure Performance|*|Agent|App|Availability&time-range-type=BEFORE_NOW&duration-in-mins=15
```

Full Example:

```
curl --user user1@customer1:your_password "http://demo.appdynamics.com/controller/rest/applications
/ECommerce_E2E-Fulfillment/metric-data?metric-path=Application%20Infrastructure%20Performance%7C*%
7CAgent%7CApp%7CAvailability&time-range-type=BEFORE_NOW&duration-in-mins=15"

<metric-datas><metric-data>
  <metricId>2329</metricId>
  <metricPath>Application Infrastructure Performance|Fulfillment-Services|Agent|App|Availability<
/metricPath>
  <metricName>Agent|App|Availability</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450562460000</startTimeInMillis>
      <value>1</value>
      <min>0</min>
      <max>0</max>
      <current>1</current>
      <sum>15</sum>
      <count>15</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>false</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>2329</metricId>
  <metricPath>Application Infrastructure Performance|Fulfillment-Client-Services|Agent|App|Availability<
/metricPath>
  <metricName>Agent|App|Availability</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450562460000</startTimeInMillis>
      <value>1</value>
      <min>0</min>
      <max>0</max>
      <current>1</current>
      <sum>15</sum>
      <count>15</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>false</useRange>
    </metric-value>
  </metricValues>
</metric-data>
</metric-datas>
```

- Retrieve the **CPU % Busy** metric for all the nodes in all tiers using a wildcard for the tier and node names:

```
/controller/rest/applications/ECommerce_E2E-Fulfillment/metric-data?metric-path=Application
Infrastructure Performance|*|Individual Nodes|*|Hardware Resources|CPU|%Busy&time-range-
type=BEFORE_NOW&duration-in-mins=15
```

Full Example:

```
curl --user user1@customer1:your_password "http://demo.appdynamics.com/controller/rest/applications
/ECommerce_E2E-Fulfillment/metric-data?metric-path=Application%20Infrastructure%20Performance%7C*%
7CIndividual%20Nodes%7C*%7CHardware%20Resources%7CCPU%7C%25Busy&time-range-type=BEFORE_NOW&duration-in-
mins=15"

<metric-datas><metric-data>
  <metricId>2231</metricId>
  <metricPath>Application Infrastructure Performance|Fulfillment-Client-Services|Individual
Nodes|FulfillmentClient|Hardware Resources|CPU|%Busy</metricPath>
  <metricName>Hardware Resources|CPU|%Busy</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450563000000</startTimeInMillis>
      <value>10</value>
      <min>2</min>
      <max>82</max>
      <current>6</current>
      <sum>4474</sum>
      <count>450</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>true</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>2231</metricId>
  <metricPath>Application Infrastructure Performance|Fulfillment-Services|Individual
Nodes|Fulfillment|Hardware Resources|CPU|%Busy</metricPath>
  <metricName>Hardware Resources|CPU|%Busy</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450563000000</startTimeInMillis>
      <value>10</value>
      <min>2</min>
      <max>82</max>
      <current>6</current>
      <sum>4478</sum>
      <count>450</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>true</useRange>
    </metric-value>
  </metricValues>
</metric-data>
</metric-datas>
```

- Retrieve the **Calls per Minute** metric for all the business transactions on the ECommerce tier using a wildcard for the business transaction name:

```
/controller/rest/applications/ACME Book Store Application/metric-data?metric-path=Business Transaction
Performance|Business Transactions|ECommerce Server|*|Calls per Minute&time-range-
type=BEFORE_NOW&duration-in-mins=15
```

Full Example:

```
curl --user user1@customer1:your_password "http://demo.appdynamics.com/controller/rest/applications
/ECommerce_E2E/metric-data?metric-path=Business%20Transaction%20Performance%7CBusiness%20Transactions%
7CECommerce-Services%7C*%7CCalls%20per%20Minute&time-range-type=BEFORE_NOW&duration-in-mins=15"
<metric-datas><metric-data>
  <metricId>4042</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|/json/cart/all.
GET|Calls per Minute</metricPath>
  <metricName>BTM|BTs|BT:125|Component:9|Calls per Minute</metricName>
  <frequency>ONE_MIN</frequency>
```

```
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450563420000</startTimeInMillis>
      <value>0</value>
      <min>0</min>
      <max>0</max>
      <current>0</current>
      <sum>5</sum>
      <count>30</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>false</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>9784</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|/appdynamicspilot
/WEB-INF|Calls per Minute</metricPath>
  <metricName>BTM|BTs|BT:183|Component:9|Calls per Minute</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450563420000</startTimeInMillis>
      <value>0</value>
      <min>2147483647</min>
      <max>-2147483648</max>
      <current>0</current>
      <sum>0</sum>
      <count>0</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>false</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>5574</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|/appdynamicspilot
/404.jsp|Calls per Minute</metricPath>
  <metricName>BTM|BTs|BT:140|Component:9|Calls per Minute</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450563420000</startTimeInMillis>
      <value>0</value>
      <min>2147483647</min>
      <max>-2147483648</max>
      <current>0</current>
      <sum>0</sum>
      <count>0</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>false</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>4033</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|/json/items/all.
GET|Calls per Minute</metricPath>
  <metricName>BTM|BTs|BT:124|Component:9|Calls per Minute</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450563420000</startTimeInMillis>
      <value>0</value>
      <min>0</min>
      <max>0</max>
      <current>0</current>
      <sum>5</sum>
```

```
      <count>30</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>false</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>4060</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|/user/login.
POST|Calls per Minute</metricPath>
  <metricName>BTM|BTs|BT:127|Component:9|Calls per Minute</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450563420000</startTimeInMillis>
      <value>0</value>
      <min>0</min>
      <max>0</max>
      <current>0</current>
      <sum>5</sum>
      <count>30</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>false</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>5592</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|/cart/{id}.
DELETE|Calls per Minute</metricPath>
  <metricName>BTM|BTs|BT:142|Component:9|Calls per Minute</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450563420000</startTimeInMillis>
      <value>0</value>
      <min>2147483647</min>
      <max>-2147483648</max>
      <current>0</current>
      <sum>0</sum>
      <count>0</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>false</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>5583</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|/cart/{id}.
GET|Calls per Minute</metricPath>
  <metricName>BTM|BTs|BT:141|Component:9|Calls per Minute</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450563420000</startTimeInMillis>
      <value>0</value>
      <min>2147483647</min>
      <max>-2147483648</max>
      <current>0</current>
      <sum>0</sum>
      <count>0</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>false</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
```

```
  <metricId>4024</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|/json/cart/co.
GET|Calls per Minute</metricPath>
  <metricName>BTM|BTs|BT:123|Component:9|Calls per Minute</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450563420000</startTimeInMillis>
      <value>0</value>
      <min>0</min>
      <max>0</max>
      <current>0</current>
      <sum>5</sum>
      <count>30</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>false</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>2477</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|/user/.
POST|Calls per Minute</metricPath>
  <metricName>BTM|BTs|BT:92|Component:9|Calls per Minute</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450563420000</startTimeInMillis>
      <value>5</value>
      <min>0</min>
      <max>0</max>
      <current>3</current>
      <sum>71</sum>
      <count>30</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>false</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>5601</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|/cart/co.
GET|Calls per Minute</metricPath>
  <metricName>BTM|BTs|BT:143|Component:9|Calls per Minute</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450563420000</startTimeInMillis>
      <value>0</value>
      <min>2147483647</min>
      <max>-2147483648</max>
      <current>0</current>
      <sum>0</sum>
      <count>0</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>false</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>4099</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|UserLogin.
memberLogin|Calls per Minute</metricPath>
  <metricName>BTM|BTs|BT:129|Component:9|Calls per Minute</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450563420000</startTimeInMillis>
```

```
      <value>0</value>
      <min>2147483647</min>
      <max>-2147483648</max>
      <current>0</current>
      <sum>0</sum>
      <count>0</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>false</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>4138</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|/appdynamicspilot
/|Calls per Minute</metricPath>
  <metricName>BTM|BTs|BT:132|Component:9|Calls per Minute</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450563420000</startTimeInMillis>
      <value>0</value>
      <min>0</min>
      <max>0</max>
      <current>0</current>
      <sum>4</sum>
      <count>30</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>false</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>4108</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|ViewItems.
getAllItems|Calls per Minute</metricPath>
  <metricName>BTM|BTs|BT:130|Component:9|Calls per Minute</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450563420000</startTimeInMillis>
      <value>0</value>
      <min>0</min>
      <max>0</max>
      <current>0</current>
      <sum>4</sum>
      <count>30</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>false</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>4129</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|ViewCart.
sendItems|Calls per Minute</metricPath>
  <metricName>BTM|BTs|BT:131|Component:9|Calls per Minute</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450563420000</startTimeInMillis>
      <value>0</value>
      <min>0</min>
      <max>0</max>
      <current>0</current>
      <sum>2</sum>
      <count>30</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
```

```
      <useRange>false</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>4051</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|/json/cart/{id}.
GET|Calls per Minute</metricPath>
  <metricName>BTM|BTs|BT:126|Component:9|Calls per Minute</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450563420000</startTimeInMillis>
      <value>1</value>
      <min>0</min>
      <max>0</max>
      <current>0</current>
      <sum>13</sum>
      <count>30</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>false</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>4156</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|ViewCart.
addToCart|Calls per Minute</metricPath>
  <metricName>BTM|BTs|BT:134|Component:9|Calls per Minute</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450563420000</startTimeInMillis>
      <value>0</value>
      <min>0</min>
      <max>0</max>
      <current>0</current>
      <sum>3</sum>
      <count>30</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>false</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>4147</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|/json/fault
/getfaults.GET|Calls per Minute</metricPath>
  <metricName>BTM|BTs|BT:133|Component:9|Calls per Minute</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450563420000</startTimeInMillis>
      <value>9</value>
      <min>0</min>
      <max>0</max>
      <current>9</current>
      <sum>130</sum>
      <count>30</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>false</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>2630</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|/items/all.
GET|Calls per Minute</metricPath>
```

```
  <metricName>BTM|BTs|BT:93|Component:9|Calls per Minute</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450563420000</startTimeInMillis>
      <value>5</value>
      <min>0</min>
      <max>0</max>
      <current>0</current>
      <sum>76</sum>
      <count>30</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>false</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>4090</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|UserLogOut.
memberLogOut|Calls per Minute</metricPath>
  <metricName>BTM|BTs|BT:128|Component:9|Calls per Minute</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450563420000</startTimeInMillis>
      <value>0</value>
      <min>2147483647</min>
      <max>-2147483648</max>
      <current>0</current>
      <sum>0</sum>
      <count>0</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>false</useRange>
    </metric-value>
  </metricValues>
</metric-data>
</metric-datas>
```

- Retrieve data for multiple metrics for the `ViewCart.addToCart` transaction on the ECommerce-Services server:

```
/controller/rest/applications/ECommerce_E2E/metric-data?metric-path=Business Transaction
Performance|Business Transactions|ECommerce Server|ViewCart.addToCart|*&time-range-
type=BEFORE_NOW&duration-in-mins=15
```

Full Example:

```
curl --user user1@customer1:your_password "http://demo.appdynamics.com/controller/rest/applications
/ECommerce_E2E/metric-data?metric-path=Business%20Transaction%20Performance%7CBusiness%20Transactions%
7CECommerce-Services%7CViewCart.addToCart%7C*&time-range-type=BEFORE_NOW&duration-in-mins=15"

<metric-datas><metric-data>
  <metricId>4155</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|ViewCart.
addToCart|Average Response Time (ms)</metricPath>
  <metricName>BTM|BTs|BT:134|Component:9|Average Response Time (ms)</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450566420000</startTimeInMillis>
      <value>28</value>
      <min>0</min>
      <max>32</max>
      <current>0</current>
      <sum>84</sum>
      <count>3</count>
```

```
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>true</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>4159</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|ViewCart.
addToCart|Number of Very Slow Calls</metricPath>
  <metricName>BTM|BTs|BT:134|Component:9|Number of Very Slow Calls</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450566420000</startTimeInMillis>
      <value>0</value>
      <min>2147483647</min>
      <max>-2147483648</max>
      <current>0</current>
      <sum>0</sum>
      <count>0</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>false</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>4157</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|ViewCart.
addToCart|Errors per Minute</metricPath>
  <metricName>BTM|BTs|BT:134|Component:9|Errors per Minute</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450566420000</startTimeInMillis>
      <value>0</value>
      <min>2147483647</min>
      <max>-2147483648</max>
      <current>0</current>
      <sum>0</sum>
      <count>0</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>false</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>4161</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|ViewCart.
addToCart|Average CPU Used (ms)</metricPath>
  <metricName>BTM|BTs|BT:134|Component:9|Average CPU Used (ms)</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450566420000</startTimeInMillis>
      <value>18</value>
      <min>0</min>
      <max>20</max>
      <current>0</current>
      <sum>54</sum>
      <count>3</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>true</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>4160</metricId>
```

```
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|ViewCart.
addToCart|Stall Count</metricPath>
  <metricName>BTM|BTs|BT:134|Component:9|Stall Count</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450566420000</startTimeInMillis>
      <value>0</value>
      <min>2147483647</min>
      <max>-2147483648</max>
      <current>0</current>
      <sum>0</sum>
      <count>0</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>false</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>4411</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|ViewCart.
addToCart|95th Percentile Response Time (ms)</metricPath>
  <metricName>BTM|BTs|BT:134|Component:9|95th Percentile Response Time (ms)</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450566420000</startTimeInMillis>
      <value>28</value>
      <min>0</min>
      <max>32</max>
      <current>0</current>
      <sum>84</sum>
      <count>3</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>true</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>4335</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|ViewCart.
addToCart|Normal Average Response Time (ms)</metricPath>
  <metricName>BTM|BTs|BT:134|Component:9|Normal Average Response Time (ms)</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450566420000</startTimeInMillis>
      <value>28</value>
      <min>0</min>
      <max>32</max>
      <current>0</current>
      <sum>84</sum>
      <count>3</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>true</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>4162</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|ViewCart.
addToCart|Average Block Time (ms)</metricPath>
  <metricName>BTM|BTs|BT:134|Component:9|Average Block Time (ms)</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450566420000</startTimeInMillis>
      <value>0</value>
```

```
      <min>0</min>
      <max>0</max>
      <current>0</current>
      <sum>0</sum>
      <count>3</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>true</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>4163</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|ViewCart.
addToCart|Average Wait Time (ms)</metricPath>
  <metricName>BTM|BTs|BT:134|Component:9|Average Wait Time (ms)</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450566420000</startTimeInMillis>
      <value>0</value>
      <min>0</min>
      <max>0</max>
      <current>0</current>
      <sum>0</sum>
      <count>3</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>true</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>4156</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|ViewCart.
addToCart|Calls per Minute</metricPath>
  <metricName>BTM|BTs|BT:134|Component:9|Calls per Minute</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450566420000</startTimeInMillis>
      <value>0</value>
      <min>0</min>
      <max>0</max>
      <current>0</current>
      <sum>3</sum>
      <count>30</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>false</useRange>
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>4331</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|ViewCart.
addToCart|Average Request Size</metricPath>
  <metricName>BTM|BTs|BT:134|Component:9|Average Request Size</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450566420000</startTimeInMillis>
      <value>740</value>
      <min>0</min>
      <max>1057</max>
      <current>0</current>
      <sum>2221</sum>
      <count>3</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>true</useRange>
```

```
    </metric-value>
  </metricValues>
</metric-data>
<metric-data>
  <metricId>4158</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce-Services|ViewCart.
addToCart|Number of Slow Calls</metricPath>
  <metricName>BTM|BTs|BT:134|Component:9|Number of Slow Calls</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450566420000</startTimeInMillis>
      <value>0</value>
      <min>2147483647</min>
      <max>-2147483648</max>
      <current>0</current>
      <sum>0</sum>
      <count>0</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>false</useRange>
    </metric-value>
  </metricValues>
</metric-data>
</metric-datas>
```

Disabling Data Rollup

By default, metric data is rolled up for the time frame you request. You can set the rollup parameter to false to get all data points within the time frame. For example:

```
curl --user user1@customer1:your_password "http://demo.appdynamics.com/controller/rest/applications
/ECommerce_E2E/metric-data?rollup=false&metric-path=Overall%20Application%20Performance%7CAverage%20Response%
20Time%20%28ms%29&time-range-type=BEFORE_NOW&duration-in-mins=15"

<metric-datas><metric-data>
  <metricId>2339</metricId>
  <metricPath>Overall Application Performance|Average Response Time (ms)</metricPath>
  <metricName>BTM|Application Summary|Average Response Time (ms)</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450503540000</startTimeInMillis>
      <value>334</value>
      <min>0</min>
      <max>3340</max>
      <current>2</current>
      <sum>6678</sum>
      <count>20</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>true</useRange>
    </metric-value>
    <metric-value>
      <startTimeInMillis>1450503600000</startTimeInMillis>
      <value>771</value>
      <min>1</min>
      <max>11235</max>
      <current>4113</current>
      <sum>15424</sum>
      <count>20</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>true</useRange>
    </metric-value>
    <metric-value>
      <startTimeInMillis>1450503660000</startTimeInMillis>
      <value>215</value>
      <min>0</min>
      <max>4249</max>
      <current>3</current>
      <sum>4306</sum>
      <count>20</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>true</useRange>
    </metric-value>
  ...
  </metricValues>
</metric-data>
</metric-datas>
```

Using Time Ranges

You can fetch metric data for any time range, including for a range between specific points, such as from 2:00 to 2:15 pm Monday, or for a relative time range.

Time-based input parameters for the metric data API method let you specify a time range in several ways, as described in the following table.

Time Range Input Parameters

Parameter Name	Parameter Type	Value	Mandatory

`time-range-type`	Query	Possible values are: • `BEFORE_NOW`—must specify the duration-in-mins parameter. • `BEFORE_TIME`—must specify the duration-in-mins and end-time parameters. • `AFTER_TIME`—must specify the duration-in-mins and start-time parameters. • `BETWEEN_TIMES`—must specify the start-time and end-time parameters. The `BETWEEN_TIMES` range includes the start- time and excludes the end-time.	Yes
`duration-in-mins`	Query	Duration (in minutes) to return the metric data.	If time-range-type is `BEFORE_NOW`, `BEFORE_TIME`, or `AFTER_TIME`
`start-time`	Query	Start time (in milliseconds) from which the metric data is returned in UNIX epoch time.	If time-range-type is `AFTER_TIME` or `BETWEEN_TIMES`
`end-time`	Query	End time (in milliseconds) until which the metric data is returned in UNIX epoch time.	If time-range-type is `BEFORE_TIME` or `BETWEEN_TIMES`

Examples

Most examples on this page use the previous 15 minutes as the request time range. The following format examples show other ways you can define the time range for the request.

- Time range of the 15 minutes after December 19, 2015 5:40:00 AM GMT:

```
?time-range-type=AFTER_TIME&start-time=1450532400000&duration-in-mins=15
```

- Time range of the 15 minutes before December 19, 2015 6:00:00 AM GMT.

```
?time-range-type=BEFORE_TIME&end-time=1450533600000&duration-in-mins=15
```

- Time range between December 19, 2015 6:00:00 AM GMT and December 19, 2015 6:30:00 AM GMT:

```
?time-range-type=BETWEEN_TIMES&start-time=1450533600000&end-time=1450535400000
```

Retrieving All Other Traffic Business Transaction Metrics

The **All Other Traffic** business transaction is a type of business transaction that aggregates traffic for new transactions once the business transaction registration limits are reached and uses the special identifier, `_APPDYNAMICS_DEFAULT_TX_`, in API URI paths. See Business Transactions.

The following shows an example of retrieving the average CPU used by the **All Other Traffic** business transaction:

```
curl --user user1@customer1:your_password "http://demo.appdynamics.com:8090/controller/rest/applications/ACME%
20Book%20Store%20Application/metric-data?metric-path=Business%20Transaction%20Performance%7CBusiness%
20Transactions%7CECommerce%20Server%7C_APPDYNAMICS_DEFAULT_TX_%7CAverage%20CPU%20Used%20%28ms%29&time-range-
type=BEFORE_NOW&duration-in-mins=15"

<metric-datas><metric-data>
  <metricId>4000</metricId>
  <metricPath>Business Transaction Performance|Business Transactions|ECommerce
Server|_APPDYNAMICS_DEFAULT_TX_|Average CPU Used (ms)</metricPath>
  <metricName>BTM|BTs|BT:78|Component:12|Average CPU Used (ms)</metricName>
  <frequency>ONE_MIN</frequency>
  <metricValues>
    <metric-value>
      <startTimeInMillis>1450570800000</startTimeInMillis>
      <value>22</value>
      <min>0</min>
      <max>50</max>
      <current>20</current>
      <sum>3140</sum>
      <count>146</count>
      <standardDeviation>0.0</standardDeviation>
      <occurrences>0</occurrences>
      <useRange>true</useRange>
    </metric-value>
  </metricValues>
</metric-data>
</metric-datas>
```

Retrieve Transaction Snapshots

Snapshots contain details on transactions, by request segment. The time range parameters are the same for snapshots as for retrieving metrics. You can similarly specify a relative time range or a specific range. See Using Time Ranges.

Format

`GET /controller/rest/applications/`*`application_name`*`/request-snapshots`

Input parameters

Parameter Name	Parameter Type	Value	Mandatory
`application_name`	URI	Provide either the application name or application id.	Yes
`time-range-type`	Query	Possible values are: • `BEFORE_NOW`—must specify the duration-in-mins parameter. • `BEFORE_TIME`—must specify the duration-in-mins and end-time parameters. • `AFTER_TIME`—must specify the duration-in-mins and start-time parameters. • `BETWEEN_TIMES`—must specify the start-time and end-time parameters. The `BETWEEN_TIMES` range includes the start- time and excludes the end-time.	Yes
`duration-in-mins`	Query	Duration (in minutes) to return the data.	If time-range-type is BEFORE_NOW, BEFORE_TIME, or AFTER_TIME
`start-time`	Query	Start time (in milliseconds) from which the data is returned.	If time-range-type is AFTER_TIME or BETWEEN_TIMES
`end-time`	Query	End time (in milliseconds) until which the data is returned.	If time-range-type is BEFORE_TIME or BETWEEN_TIMES
`guids`	Query	Array of comma-separated GUIDs for the transaction snapshots. If not specified, retrieves all snapshots in the specified time range.	No
`archived`	Query	True to retrieve archived snapshots. Default is false.	No

`deep-dive-policy`	Query	Array of comma-separated snapshot policy filters to apply. Valid values are: • `SLA_FAILURE` • `TIME_SAMPLING` • `ERROR_SAMPLING` • `OCCURRENCE_SAMPLING` • `ON_DEMAND` • `APPLICATION_STARTUP` • `SLOW_DIAGNOSTIC_SESSION` • `ERROR_DIAGNOSTIC_SESSION` • `POLICY_FAILURE_DIAGNOSTIC_SESSION` • `DIAGNOSTIC_SESSION` • `INFLIGHT_SLOW_SESSION`	No
`application-component-ids`	Query	Array of comma-separated tier IDs to filter. The default is all the tiers in the application.	No
`application-component-node-ids`	Query	Array of comma-separated node ID filters. Default is all the nodes in the application	No
`business-transaction-ids`	Query	Array of comma-separated business transaction ID filters. Default is all the business transactions in the application.	No
`user-experience`	Query	Array of comma-separated user experiences filters. Valid values are: • `NORMAL` • `SLOW` • `VERY_SLOW` • `STALL` • `ERROR`	No
`first-in-chain`	Query	If `true`, retrieve only the first request from the chain. Default is `false`.	No
`need-props`	Query	If `true`, the values of the following snapshot properties are included in the output. These values correspond to the values of the `data-collector-type` parameter. If `false`, the default, these values are empty in the output. • `errorDetails` • `errorIDs` • `httpParameters` • `businessData` • `cookies` • `httpHeaders` • `sessionKeys` • `responseHeaders` • `logMessages` • `transactionProperties` • `transactionEvents` • `dotnetProperty`	No
`need-exit-calls`	Query	If `true`, exit calls are included in the result. Default is `false`.	No
`execution-time-in-milis`	Query	If set, retrieves only data for requests with execution times greater than this value.	No
`session-id`	Query	If set, retrieves data only for this session id.	No
`user-principal-id`	Query	If set, retrieves data only for this user login.	No
`error-ids`	Query	Array of comma-separated error codes to filter by. The default is to retrieve all error codes.	No
`starting-request-id, ending-request-id`	Query	If set, retrieves data only for this range of request IDs.	No
`error-occurred`	Query	If `true`, retrieves only error requests. Default is `false`.	No
`diagnostic-snapshot`	Query	If `true`, retrieves only diagnostic snapshots. Default is `false`.	No
`bad-request`	Query	If `true`, retrieves only slow and error requests. Default is `false`.	No
`diagnostic-session-guid`	Query	Array of comma-separated diagnostic session GUIDs to filter.	No
`data-collector-name`	Query	Used with `data-collector-value` to filter snapshot collection based on the value of a data collector.	No
`data-collector-value`	Query	Used with `data-collector-name` to filter snapshot collection based on the value of a data collector.	If data-collector-name is set.

`data-collector-type`	Query	Used with `data-collector-name` and `data-collector-value` to filter snapshot collection based on the value of a data collector. Some of the values contain spaces. All are case-sensitive and where indicated the spaces are required. Valid values are: • `Error IDs` • `Stack Traces` • `Error Detail` • `Http Parameter` • `Business Data` (This type is a method invocation data collector.) • `Cookie` • `Http Header` • `Session Key` • `Response Header` • `Log Message` • `Transaction Property` • `Transaction Event` • `Dotnet Property` • `isProtoBuf` • `EUM Request GUID`	
`output`	Query	HTTP Request parameter included as part of the URL to change the output format Valid values are `XML` (default) or `JSON`.	No
`maximum-results`	Query	If specified, the number of maximum results will be returned. If not specified, a default of 600 results can be returned at most.	No

Examples

- Retrieve list of transaction snapshots for the ACME Book Store:

```
/controller/rest/applications/ECommerce_E2E-Fulfillment/request-snapshots?time-range-
type=BEFORE_NOW&duration-in-mins=5
```

Full Example:

```
curl --user user1@customer1:your_password "http://demo.appdynamics.com/controller/rest/applications
/ECommerce_E2E-Fulfillment/request-snapshots?time-range-type=BEFORE_NOW&duration-in-mins=5"

<request-segment-datas><request-segment-data>
  <id>0</id>
  <archived>false</archived>
  <requestGUID>18a9ae17-33a8-4d24-b3fa-558fe42b98b5</requestGUID>
  <businessTransactionId>113</businessTransactionId>
  <applicationId>8</applicationId>
  <applicationComponentId>14</applicationComponentId>
  <applicationComponentNodeId>13</applicationComponentNodeId>
  <async>false</async>
  <threadID>58</threadID>
  <threadName>http-nio-8080-exec-8</threadName>
  <localStartTime>1450574075422</localStartTime>
  <serverStartTime>1450574075422</serverStartTime>
  <firstInChain>true</firstInChain>
  <callChain>Component:14</callChain>
  <localID>0</localID>
  <errorOccured>false</errorOccured>
  <hasDeepDiveData>true</hasDeepDiveData>
  <userExperience>NORMAL</userExperience>
  <timeTakenInMilliSecs>3693</timeTakenInMilliSecs>
  <cpuTimeTakenInMilliSecs>19</cpuTimeTakenInMilliSecs>
  <warningThreshold>5318 ms. 3.0x of standard deviation [453.7 ms] for moving average [3956.8 ms]
(minimum baseline: 200 ms) for the last 11617 minutes.</warningThreshold>
  <criticalThreshold>5772 ms. 4.0x of standard deviation [453.7 ms] for moving average [3956.8 ms]
(minimum baseline: 600 ms) for the last 11617 minutes.</criticalThreshold>
  <summary>Scheduled Snapshots: one every 10 minutes.</summary>
  <errorSummary></errorSummary>
  <diagnosticSessionGUID></diagnosticSessionGUID>
  <deepDivePolicy>TIME_SAMPLING</deepDivePolicy>
  <delayedDeepDive>false</delayedDeepDive>
  <delayedDeepDiveOffSet>0</delayedDeepDiveOffSet>
  <exitCallsDataTruncated>false</exitCallsDataTruncated>
  <URL>/appdynamicspilot/rest/fulfillment</URL>
  <errorIDs/>
```

```
  <errorDetails/>
  <httpParameters/>
  <businessData/>
  <cookies/>
  <httpHeaders/>
  <sessionKeys/>
  <responseHeaders/>
  <logMessages/>
  <transactionProperties/>
  <transactionEvents/>
  <unresolvedCallInCallChain>false</unresolvedCallInCallChain>
  <dotnetProperty/>
  <endToEndLatency>-1</endToEndLatency>
</request-segment-data>
...
<request-segment-data>
  <id>0</id>
  <archived>false</archived>
  <requestGUID>bfce5066-2409-4a4b-a869-6afcc06614d6</requestGUID>
  <businessTransactionId>113</businessTransactionId>
  <applicationId>8</applicationId>
  <applicationComponentId>14</applicationComponentId>
  <applicationComponentNodeId>13</applicationComponentNodeId>
  <async>false</async>
  <threadID>60</threadID>
  <threadName>http-nio-8080-exec-10</threadName>
  <localStartTime>1450574082926</localStartTime>
  <serverStartTime>1450574082926</serverStartTime>
  <firstInChain>true</firstInChain>
  <callChain>Component:14</callChain>
  <localID>0</localID>
  <errorOccured>false</errorOccured>
  <hasDeepDiveData>true</hasDeepDiveData>
  <userExperience>NORMAL</userExperience>
  <timeTakenInMilliSecs>3634</timeTakenInMilliSecs>
  <cpuTimeTakenInMilliSecs>16</cpuTimeTakenInMilliSecs>
  <warningThreshold>5318 ms. 3.0x of standard deviation [453.7 ms] for moving average [3956.8 ms]
(minimum baseline: 200 ms) for the last 11617 minutes.</warningThreshold>
  <criticalThreshold>5772 ms. 4.0x of standard deviation [453.7 ms] for moving average [3956.8 ms]
(minimum baseline: 600 ms) for the last 11617 minutes.</criticalThreshold>
  <summary>[null]</summary>
  <errorSummary></errorSummary>
  <diagnosticSessionGUID></diagnosticSessionGUID>
  <deepDivePolicy>CROSS_APP_POLICY</deepDivePolicy>
  <delayedDeepDive>false</delayedDeepDive>
  <delayedDeepDiveOffSet>0</delayedDeepDiveOffSet>
  <exitCallsDataTruncated>false</exitCallsDataTruncated>
  <URL>/appdynamicspilot/rest/fulfillment</URL>
  <errorIDs/>
  <errorDetails/>
  <httpParameters/>
  <businessData/>
  <cookies/>
  <httpHeaders/>
  <sessionKeys/>
  <responseHeaders/>
  <logMessages/>
  <transactionProperties/>
  <transactionEvents/>
  <unresolvedCallInCallChain>false</unresolvedCallInCallChain>
  <dotnetProperty/>
  <endToEndLatency>-1</endToEndLatency>
</request-segment-data>
</request-segment-datas>
```

- Retrieve list of transaction snapshots including the snapshot fields that are associated with an HTTP parameter data collector:

```
/controller/rest/applications/ECommerce_E2E-Fulfillment/request-snapshots?time-range-
type=BEFORE_NOW&duration-in-mins=5&data-collector-type=Http Parameter&data-collector-name=param1&data-
collector-value=%5B100%5D&need-props=true
```

Full Example:

```
curl --user user1@customer1:your_password "http://demo.appdynamics.com/controller/rest/applications
/ECommerce_E2E-Fulfillment/request-snapshots?time-range-type=BEFORE_NOW&duration-in-mins=5&data-
collector-type=Http%20Parameter&data-collector-name=param1&data-collector-value=%5B100%5D&need-
props=true"

<request-segment-datas><request-segment-data>
   <id>0</id>
   <archived>false</archived>
   <requestGUID>07532d68-42b8-4a79-877a-dedf2912a2cf</requestGUID>
   <businessTransactionId>128</businessTransactionId>
   <applicationId>2</applicationId>
   <applicationComponentId>5</applicationComponentId>
   <applicationComponentNodeId>4</applicationComponentNodeId>
   <async>false</async>
   <threadID>60</threadID>
   <threadName>http-8000-Processor24</threadName>
   <localStartTime>1389164292752</localStartTime>
   <serverStartTime>1389164292752</serverStartTime>
   <firstInChain>true</firstInChain>
   <callChain>Component:5</callChain>
   <localID>0</localID>
   <errorOccured>true</errorOccured>
   <hasDeepDiveData>true</hasDeepDiveData>
   <userExperience>ERROR</userExperience>
   <timeTakenInMilliSecs>105</timeTakenInMilliSecs>
   <cpuTimeTakenInMilliSecs>3839000</cpuTimeTakenInMilliSecs>
   <summary>[Manual Diagnostic Session] - org.hibernate.util.JDBCExceptionReporter : Cannot create
PoolableConnectionFactory (Unknown database 'appdy') </summary>
   <errorSummary/>
   <diagnosticSessionGUID>d70a41d9-a96f-46e8-9fbc-31061c6e452f</diagnosticSessionGUID>
   <deepDivePolicy>ON_DEMAND</deepDivePolicy>
   <delayedDeepDive>false</delayedDeepDive>
   <delayedDeepDiveOffSet>0</delayedDeepDiveOffSet>
   <exitCallsDataTruncated>false</exitCallsDataTruncated>
   <URL>/appdynamicspilot/1.bookslist</URL>
   <httpSessionID>088B2A2DD0EF77424DD0EB3346A441F9</httpSessionID>
   <errorIDs>
   <long>29</long>
   </errorIDs>
   <errorDetails>
   <name-value>
   <id>0</id>
   <name>1. org.hibernate.util.JDBCExceptionReporter</name>
   <value>org.hibernate.util.JDBCExceptionReporter : Cannot create PoolableConnectionFactory (Unknown
database 'appdy')</value>
   </name-value>
   </errorDetails>
   <httpParameters>
   <name-value>
   <id>0</id>
   <name>param1</name>
   <value>[100]</value>
   </name-value>
   </httpParameters>
   <businessData/>
   <cookies/>
   <httpHeaders/>
   <sessionKeys/>
   <responseHeaders/>
   <logMessages/>
   <transactionProperties>
   <name-value>
```

```
   <id>0</id>
   <name>Servlet URI</name>
   <value>/appdynamicspilot/WEB-INF/presentation/bookslist.jsp</value>
   </name-value>
   <name-value>
   <id>0</id>
   <name>ProcessID</name>
   <value>65331</value>
   </name-value>
   </transactionProperties>
   <transactionEvents/>
   <unresolvedCallInCallChain>false</unresolvedCallInCallChain>
   <dotnetProperty/>
</request-segment-data></request-segment-datas>
```

Retrieve Controller Audit History

The Controller audit history is a record of the configuration and user activities in the Controller configuration. The `ControllerAuditHistory` API method returns the audit log for the time range specified. The output format can be `JSON` or `CSV`. This information is the same as that found in the `audit.log` file. See Platform Log Files and Log File Information by Platform

Format

```
GET /controller/ ControllerAuditHistory?startTime=<start-time>&endTime=<end-time>&include=<field>:
<value>&exclude=<field>:<value>
```

Input parameters

Parameter Name	Parameter Type	Value	Mandatory
`start-time`	Query	Start time in the format: `yyyy-MM-dd'T'HH:mm:ss.SSSZ`	Yes
`end-time`	Query	End time in the format: `yyyy-MM-dd'T'HH:mm:ss.SSSZ`	Yes
`time-zone-id`	Query	Time zone	No
`include`	Query	Restricted information in the Controller audit history	No
`exclude`	Query	Restricted information in the Controller audit history	No

- To control the size of the output, the range between the `start-time` and `end-time` cannot exceed 24 hours. For periods longer than 24 hours, use multiple queries with consecutive time parameters.
- Multiple filters of the same type are allowed.
  - The backend API treats included filters with the same `<field>` with relationship `OR`
  - Filters with different `<field>` with relationship `AND`. There is no direct interaction between `include` and `exclude` filters.
- Each filter needs to be a parameter, e.g. `include=filterName1:filterValue1&include=filterName2:filterValue2`. See the below examples:

```
http://localhost:8080/controller/ControllerAuditHistory?startTime=yyyy-MM-dd&HH:mm:ss.SSSZ&endTime=yyyy-MM-
dd&HH:mm:ss.SSSZ?include=filterName1:filterValue1&include=filterName1:filterValue1&exclude=filterName1:
filterValue1&exclude=filterName1:filterValue1
```

```
curl --user user1@customer1:welcome "http://demo.appdynamics.com:8090/controller/ControllerAuditHistory?
startTime=2015-12-19T10:50:03.607-0700&endTime=2015-12-19T17:50:03.607-0700&timeZoneId=America%2FSan%
20Francisco&include=userName:user1&include=action:LOGIN&exclude=accountName:system&exclude=action:OBJECT_UPDATE"

[{"timeStamp":1450569821811,"auditDateTime":"2015-12-20T00:03:41.811+0000","accountName":"customer1","
securityProviderType":"INTERNAL","userName":"user1","action":"LOGIN"},{"timeStamp":1450570234518,"
auditDateTime":"2015-12-20T00:10:34.518+0000","accountName":"customer1","securityProviderType":"INTERNAL","
userName":"user1","action":"LOGIN"},{"timeStamp":1450570273841,"auditDateTime":"2015-12-20T00:11:13.841+0000","
accountName":"customer1","securityProviderType":"INTERNAL","userName":"user1","action":"OBJECT_CREATED","
objectType":"AGENT_CONFIGURATION"},
...
{"timeStamp":1450570675345,"auditDateTime":"2015-12-20T00:17:55.345+0000","accountName":"customer1","
securityProviderType":"INTERNAL","userName":"user1","action":"OBJECT_DELETED","objectType":"
BUSINESS_TRANSACTION"},{"timeStamp":1450570719240,"auditDateTime":"2015-12-20T00:18:39.240+0000","accountName":"
customer1","securityProviderType":"INTERNAL","userName":"user1","action":"APP_CONFIGURATION","objectType":"
APPLICATION","objectName":"ACME Book Store Application"},{"timeStamp":1450571834835,"auditDateTime":"2015-12-
20T00:37:14.835+0000","accountName":"customer1","securityProviderType":"INTERNAL","userName":"user1","action

curl --user user1@customer1:welcome "http://127.0.0.1:8080/controller/ControllerAuditHistory?startTime=2019-05-
28T08:00:03.607-0700&endTime=2019-05-28T11:32:03.607-0700&timeZoneId=America%2FSan%
20Francisco&include=applicationName:ACME"
[{"timeStamp":1559066415823,"auditDateTime":"2019-05-28T18:00:15.823+0000","accountName":"customer1","
securityProviderType":"INTERNAL","userName":"user1","action":"LOGIN","objectId":0,"applicationName":"ACME"}]
```

Configure Metric Retention by Account

You can configure the Controller to purge stale metrics once a day based on the account.

- Stale metrics are metrics that have not had new data reported based on the number of days configured.
- This only deletes EUM and SIM metrics that are more than two days old.

To configure this option, you must be the account owner.

Format

```
POST /controller/api/accounts/<account_id>/metricstaleduration/<number_of_days>
```

Input Parameters

Parameter Name	Parameter Type	Value
account_id	URI	The account ID.
number_of_days	Integer	The number of days you want to retain stale metrics.

Example

```
curl -X POST -u user1@customer1:your_password "http://demo.appdynamics.com:8090/controller/api/accounts/2
/metricstaleduration/3"
```

Configure Metric Retention by Application

You can configure the Controller to purge stale metrics once a day based on application.

- Stale metrics are metrics that have not had new data reported based on the number of days configured.
- This only deletes EUM and SIM metrics that are more than two days old.

To configure this option, you must have administrator permissions or higher.

Format

```
POST /controller/api/accounts/<account_id>/applications/<application_name>/metricstaleduration/<number_of_days>
```

Input Parameters

Parameter Name	Parameter Type	Value
`account_id`	URI	The account ID.
`application_id`	URI	The application ID.
`number_of_days`	Integer	The number of days you want to retain stale metrics.

Example

```
curl -X POST -u user1@customer1:your_password "http://demo.appdynamics.com:8090/controller/api/accounts/2
/applications/12/metricstaleduration/3"
```

Alert and Respond API

This page provides links to the Alert and Respond APIs that allow you to create, configure and manage:

- Health Rules
- Schedules
- Policies
- Actions
- Email Digests
- Action Suppression
- Events and Action Suppression

You can create and use the identity type, `API Clients`, to provide secure access to the AppDynamics controller using REST API calls. These calls use Open Authorization (OAuth) token-based authentication. You can create new API Client identity types that can be used to generate OAuth tokens. See API Clients.

Health Rule API

Related pages:

- Health Rules

This page describes the Health Rule API methods you can use to create, configure, update, and delete health rules for multiple applications simultaneously. This API allows you to programmatically update and maintain single or multiple health rules and migrate them across applications or Controllers.

- Minimal syntax validation of the JSON payload is done when creating the health rule.
- Path validation for a specified metric is not done. Ensure that you provide valid paths for all the metrics you define.
- If metrics are not resolved during the health rule evaluation, the health rule attains an unknown (?) state. Ensure that you provide valid metrics for all the affected entities you define.

Create a Health Rule

Creates a new health rule from the specified JSON payload. See Property Details

Resource URL

```
POST <controller_url>/controller/alerting/rest/v1/applications/<application_id>/health-rules
```

Request/Response Format

JSON

Example

This example creates a health rule with an affected entity type `business transaction performance` and defines the evaluation criteria for the health rule. See Download Examples

Retrieve a List of Health Rules for an Application

This API returns a list of all the health rule IDs and names for the specified application ID. To retrieve complete details of the health rule, use `GET //health-rule/{health-rule-id}`. See Property Details

Resource URL

```
GET <controller_url>/controller/alerting/rest/v1/applications/<application_id>/health-rules
```

Response Format

JSON

Example Response

```
[
    {
        "id": 1,
        "name": "Machine Availability is too low",
        "enabled": true
    },
    {
        "id": 2,
        "name": "Overall Disk Space Available is too low",
        "enabled": true
    },
    {
        "id": 3,
        "name": "CPU Usage is too high",
        "enabled": true
    },
    {
        "id": 4,
        "name": "Memory Usage is too high",
        "enabled": true
    },
    {
        "id": 5,
        "name": "Swap Usage is too high",
        "enabled": true
    },
    {
        "id": 6,
        "name": "Disk Usage is too high on at least one partition",
        "enabled": false
    }
]
```


Retrieve Details of a Specified Health Rule

This API Returns the health rule details for the specified health rule ID.

JMX Health Rules are not supported.

Resource URL

`GET <controller_url>/controller/alerting/rest/v1/applications/<application_id>/health-rules/{health-rule-id}`

Response Format

JSON

Example Response


```
{
    "id": 26,
    "name": "My new health rule",
    "enabled": true,
    "useDataFromLastNMinutes": 30,
    "waitTimeAfterViolation": 5,
    "scheduleName": "Always",
    "affects": {
        "affectedEntityType": "BUSINESS_TRANSACTION_PERFORMANCE",
        "affectedBusinessTransactions": {
            "businessTransactionScope": "ALL_BUSINESS_TRANSACTIONS"
        }
    },
```

```
    "evalCriterias": {
        "criticalCriteria": {
            "conditionAggregationType": "ALL",
            "conditionExpression": null,
            "conditions": [
                {
                    "name": "Condition 1",
                    "shortName": "A",
                    "evaluateToTrueOnNoData": false,
                    "evalDetail": {
                        "evalDetailType": "SINGLE_METRIC",
                        "metricAggregateFunction": "VALUE",
                        "metricPath": "Average CPU Used (ms)",
                        "metricEvalDetail": {
                            "metricEvalDetailType": "BASELINE_TYPE",
                            "baselineCondition": "WITHIN_BASELINE",
                            "baselineName": "All data - Last 15 days",
                            "compareValue": 30.5,
                            "baselineUnit": "PERCENTAGE"
                        }
                    },
                    "triggerEnabled": false,
                    "minimumTriggers": 0
                }
            ],
            "evalMatchingCriteria": {
                "matchType": "ANY_NODE",
                "value": null
            }
        },
        "warningCriteria": {
            "conditionAggregationType": "ALL",
            "conditionExpression": null,
            "conditions": [
                {
                    "name": "Condition 1",
                    "shortName": "A",
                    "evaluateToTrueOnNoData": false,
                    "evalDetail": {
                        "evalDetailType": "METRIC_EXPRESSION",
                        "metricExpression": "({metric1} + ({metric2} * 3))",
                        "metricExpressionVariables": [
                            {
                                "variableName": "metric1",
                                "metricAggregateFunction": "VALUE",
                                "metricPath": "95th Percentile Response Time (ms)"
                            },
                            {
                                "variableName": "metric2",
                                "metricAggregateFunction": "MAXIMUM",
                                "metricPath": "Average CPU Used (ms)"
                            }
                        ],
                        "metricEvalDetail": {
                            "metricEvalDetailType": "SPECIFIC_TYPE",
                            "compareCondition": "GREATER_THAN_SPECIFIC_VALUE",
                            "compareValue": 10
                        }
                    },
                    "triggerEnabled": false,
                    "minimumTriggers": 0
                }
            ],
            "evalMatchingCriteria": {
                "matchType": "ANY_NODE",
                "value": null
            }
        }
    }
}
```

Update a Health Rule

This API updates an existing health rule (required fields) with details from the specified health rule ID. See Property Details

Resource URL

```
PUT <controller_url>/controller/alerting/rest/v1/applications/<application_id>/health-rules/{health-rule-id}
```

Request/Response Format

JSON

Example

Delete a Health Rule

Deletes an existing health rule with the specified ID.

Resource URL

```
DELETE <controller_url>/controller/alerting/rest/v1/applications/<application_id>/health-rules/{health-rule-id}
```

Update a Health Rule Configuration

This API updates one or more configuration setting(s) of a health rule. See Property Details

You can enter one or both of the following field(s) in the request:

- Enable/disable the health rule.
- Update the schedule of the health rule.

Resource URL

```
PUT <controller_url>/controller/alerting/rest/v1/applications/<application_id>/health-rules/{health-rule-id}
/configuration
```

Request/Response Format

JSON

Example

Response Codes

Code	Description
200	Fetched successfully
201	Created successfully
204	Deleted successfully
400	Bad request
401	Unauthorized
403	Forbidden
404	Resource not found
409	Already exists

Property Details

HealthRule

Property Name	Type and Valid Values	Description
`id`	integer	Auto-generated by the system and returned in the response. It is a `readOnly` value.
`name`	string Minimum length: 1	Name of the health rule.
`enabled`	boolean Default value: `true`	Sets the health rule to enabled/disabled state. A health rule is evaluated only if it is in `enabled` state.
`useDataFromLastNMinutes`	integer Minimum value: 1 Maximum value: 360	The time interval (in minutes) during which the data collected is considered for health rule evaluation. Enter a value between 1 to 9 or a multiple of 10 that is less than 360.
`waitTimeAfterViolation`	integer Minimum time: 1 minute	The wait time after a violation in minutes.
`scheduleName`	string Default option: `Always`	Name of schedule to be associated with the health rule for evaluation.
`affects`*		Describes entities affected by the health rule. For example, business transactions, servers, or databases. **Affects** `affectedEntityType` (string): `OVERALL_APPLICATION_PERFORMANCE`, `BUSINESS_TRANSACTION_PERFORMANCE`, `TIER_NODE_TRANSACTION_PERFORMANCE`, `TIER_NODE_HARDWARE`, `SERVERS_IN_APPLICATION`, `BACKENDS`, `ERRORS`, `SERVICE_ENDPOINTS`, `INFORMATION_POINTS`, `CUSTOM`, `DATABASES`, `SERVERS`

evalCriterias*		Defines a condition or a set of conditions (maximum of 8) expressed as a boolean expression to evaluate the health rule. Depending on affectedEntityType a condition may or may not contain evalMatchingCriteria. Conditions are classified as: • critical or • warning

HealthRuleSummaryArray

Property	Type
id*	integer
name	string minLength: 1
enabled*	boolean

HealthRuleConfiguration

The configuration details of a health rule that you can update individually without the need to send the complete health rule JSON payload. You can mention one or more properties and all those properties are set to the new specified values.

Property	Type
enabled	boolean
scheduleName	string

MetricEvalDetail

Property	Type	Enums
metricEvalDetailType*	string	BASELINE_TYPE SPECIFIC_TYPE

BaselineMetricEvalDetail

The deviation of a metric from the baseline used to evaluate the health rule.

Property	Type	Enums
metricEvalDetailType*	string	BASELINE_TYPE SPECIFIC_TYPE
baselineCondition*	string	WITHIN_BASELINE NOT_WITHIN_BASELINE GREATER_THAN_BASELINE LESS_THAN_BASELINE
baselineName*	string minLength: 1	
compareValue*	number minimum: 0	

baselineUnit*	string	STANDARD_DEVIATIONS PERCENTAGE

SpecificValueMetricEvalDetail

The deviation of a metric from a specific value used to evaluate the health rule.

Property	Type	Enums
metricEvalDetailType*	string	BASELINE_TYPE SPECIFIC_TYPE
compareCondition*	string	GREATER_THAN_SPECIFIC_VALUE LESS_THAN_SPECIFIC_VALUE
compareValue*	number minimum: 0	

MetricAggregateFunction

Metrics aggregated to determine the deviation and evaluate the health rule.

Property	Type	Enums
MetricAggregateFunction*	string	MINIMUM MAXIMUM VALUE SUM COUNT CURRENT GROUP_COUNT

SingleMetricEvalDetail

The deviation of a single metric from the aggregated value used to evaluate the health rule.

Property	Type	Enums
evalDetailType*	string	SINGLE_METRIC METRIC_EXPRESSION
metricAggregateFunction*	string	MINIMUM MAXIMUM VALUE SUM COUNT CURRENT GROUP_COUNT
metricPath*	string minLength: 1	
metricEvalDetail*	string	BASELINE_TYPE SPECIFIC_TYPE

MetricExpressionEvalDetail

The metric expression used to evaluate the health rule.

Property	Type	Enums
`evalDetailType`*	string	`SINGLE_METRIC` `METRIC_EXPRESSION`
`metricExpression`*	string minLength: 1	
`metricExpressionVariables`*	string minItems: 2	**MetricDetailWithVariableName** `variableName`* — string, minLength: 1 `metricAggregateFunction`* — string — `MINIMUM` `MAXIMUM` `VALUE` `SUM` `COUNT` `CURRENT` `GROUP_COUNT` `metricPath`* — string, minLength: 1

`metricEvalDetail`*	string	**SpecificValueMetricEvalDetail**
		`metricEvalDetailType`* — string — `BASELINE_TYPE` `SPECIFIC_TYPE`
		`compareCondition`* — string — `GREATER_THAN_SPECIFIC_VALUE` `LESS_THAN_SPECIFIC_VALUE`
		`compareValue`* — number, minimum: 0

MetricDetailWithVariableName

Property	Type	Enums
`variableName`*	string minLength: 1	

metricAggregateFunction*	string	MINIMUM MAXIMUM VALUE SUM COUNT CURRENT GROUP_COUNT
metricPath*	string minLength: 1	

NodeEvalMatchingCriteria

Property	Type	Enums
matchType	string	AVERAGE ANY_NODE PERCENTAGE_NODES NUMBER_OF_NODES
value Enter the matching criteria only when you select PERCENTAGE_NODES or NUMBER_OF_NODES as matchType. If you select NUMBER_OF_NODES, enter an integer; else if you select PERCENTAGE_NODES, enter a number.	number minimum: 0	

Condition

A single condition that can be independently evaluated to `true` or `false`. List of conditions (maximum 8) along with other properties form a criteria.

Property	Description	Type/Enums
name*	Name of the condition.	string
shortname*	A short name used in conditionExpression to evaluate CUSTOM conditionType.	string pattern: ^[A-Z]{1,3}$ Enums A B C D E F G H
evaluateToTrueOnNoData	Controls the evaluation of the condition in cases where any metric on which the condition is based, returns no data. The condition evaluates to unknown by default when no data is returned. If the health rule is based on all the conditions evaluating to true, having no data returned may affect whether the health rule triggers an action.	boolean default: false

`evalDetail`*	Details of metric(s) considered for evaluation of the condition. Use `SINGLE_METRIC` to evaluate a single metric. Use `METRIC_EXPRESSION` to evaluate a metric expression.	**ConditionEvalDetail** **Property** / **Type** / **Enums** `evalDetailType` / string / `SINGLE_METRIC`, `METRIC_EXPRESSION`
`triggerEnabled`	If set to `true`, the value in field `minimumTriggers` is considered for evaluation.	boolean default: `false`
`minimumTriggers`	If you set a non-zero value, persistence thresholds are considered when evaluating the conditions. Ensure that you define a value less than `useDataFromLastNMinutes`.	integer default: 0 minimum: 0 maximum: 30

ConditionEvalDetail

Property	Type	Enums
`evalDetailType`	string	`SINGLE_METRIC` `METRIC_EXPRESSION`

Criteria

Property	Description	Type/Enums

`conditionAggregationType`	Condition evaluation criteria that constitute a health rule violation.	string
		default: ALL
		Enums
		`ALL`
		`ANY`
		`CUSTOM`

`conditionExpression`	Use only when you set the `conditionAggregationType` variable to `CUSTOM`. Use the `ShortName` of the condition to define the boolean expression.	String minLength: 1

<table>
<tr>
<td>conditions*</td>
<td>A single condition that can be evaluated independently to true or false. OR

A list of conditions (maximum of 8) along with other properties to form a criteria.</td>
<td>String

minItems: 1

Condition
<table>
<tr><th>Property</th><th>Description</th><th>Type/Enums</th></tr>
<tr><td>name*</td><td>Name of the condition.</td><td>string</td></tr>
<tr><td>shortname*</td><td>A short name used in conditionExpression to evaluate CUSTOM conditionType.</td><td>string

pattern: ^[A-Z]{1,3}$

Enums

A

B

C

D

E

F

G

H</td></tr>
<tr><td>evaluateToTrueOnNoData</td><td>Controls the evaluation of the condition in cases where any metric on which the condition is based, returns no data. The default when no data is returned is for the condition to evaluate to unknown. If the health rule is based on all the conditions evaluating to true, having no data returned may affect whether the health rule triggers an action.</td><td>boolean

default: false</td></tr>
<tr><td>evalDetail*</td><td>Details of metric(s) considered for evaluation of the condition. Use SINGLE_METRIC to evaluate a single metric. Use METRIC_EXPRESSION to evaluate a metric expression.</td><td>ConditionEvalDetail
<table>
<tr><th>Property</th><th>Type</th><th>Enums</th></tr>
<tr><td>evalDetailType</td><td>string</td><td>SINGLE_METRIC

METRIC_EXPRESSION</td></tr>
</table></td></tr>
<tr><td>triggerEnabled</td><td>If set to true, the value in field minimumTriggers is considered for evaluation.</td><td>boolean

default: false</td></tr>
<tr><td>minimumTriggers</td><td>If you set a non-zero value, persistence thresholds are considered when evaluating the conditions. Ensure that you define a value less than useDataFromLastNMinutes.</td><td>integer
default: 0
minimum: 0
maximum: 30</td></tr>
</table></td>
</tr>
</table>

`evalMatchingCriteria`	The criteria for evaluating a condition for the following health rule types: • Business Transaction • Node health-hardware • Node health-transaction performance It defines how many nodes in the affected entities must violate the condition before the health rule is considered to violate.	**NodeEvalMatchingCriteria** **Property** / **Description** / **Type /Enums** `matchType` — The evaluation scope type that defines how many nodes in the affected entities must violate the condition before the health rule is considered violated. Enter the matching criteria only when you select `PERCENTAGE_NODES` or `NUMBER_OF_NODES` as `matchType`. — String; Enums: `AVERAGE`, `ANY_NODE`, `PERCENTAGE_NODES`, `NUMBER_OF_NODES` `value` — The number or percentage of nodes that must violate the condition to constitute a health rule violation. If you select `NUMBER_OF_NODES`, enter an integer. If you select `PERCENTAGE_NODES`, enter a number. — number; minimum: 0

EntityMatchingPattern

Property	Type	Enums

`matchTo`*	string	`STARTS_WITH`
		`ENDS_WITH`
		`CONTAINS`
		`EQUALS`
		`MATCH_REG_EX`

`matchValue`*	string minLength: 1	
`shouldNot`	boolean	

AffectedTiers

Property	Type	Enums
`affectedTierScope`*	string	`ALL_TIERS` `SPECIFIC_TIERS`

AllTiers

The scope of affected tiers is set to all tiers for an affected entity.

Property	Type	Enums
`affectedTierScope`*	string	`ALL_TIERS` `SPECIFIC_TIERS`

SpecificTiers

The scope of affected tiers is set to specific tiers for an affected entity.

Property	Type	Enums

affectedTierScope*	string	ALL_TIERS SPECIFIC_TIERS
tiers*	string minItems: 1	

AffectedNodes

Property	Type	Enums
affectedNodeScope*	string	ALL_NODES SPECIFIC_NODES NODES_OF_SPECIFIC_TIERS NODES_MATCHING_PATTERN NODE_PROPERTY_VARIABLE_MATCHER

AllNodes

The scope of affected nodes is set to all nodes for an affected entity.

Property	Type	Enums
affectedNodeScope*	string	ALL_NODES

SpecificNodes

The scope of affected nodes is set to all nodes for an affected entity.

Property	Type	Enums
affectedNodeScope*	string	SPECIFIC_NODES
nodes*	string minItems: 1	

NodesOfSpecificTiers

The scope of affected nodes is set to nodes of specific tiers for an affected entity.

Property	Type	Enums
affectedNodeScope*	string	NODES_OF_SPECIFIC_TIERS
specificTiers*	string minItems: 1	

NodesMatchingPattern

The scope of affected nodes is set to nodes matching a pattern for an affected entity.

Property	Type	Enums
affectedNodeScope*	string	NODES_MATCHING_PATTERN

patternMatcher*		**EntityMatchingPattern** **Property** / **Type** / **Enums** matchTo* — string — STARTS_WITH, ENDS_WITH, CONTAINS, EQUALS, MATCH_REG_EX matchValue* — string, minLength: 1 shouldNot — boolean

NodePropertyVariableMatcher

The scope of affected nodes is set to nodes matching a variable property for an affected entity.

Property	Type	Enums
affectedNodeScope*	string	NODE_PROPERTY_VARIABLE_MATCHER
propVarPairs*	array minItems: 1	**Property** / **Type** / **Enums** propertyType — string — META, ENV, JVM name — string, minLength: 1 value — string, minLength: 1

OverallApplicationPerformance

The scope of the affected entity is set to overall application performance.

Property	Type	Enums
affectedEntityType*	string	OVERALL_APPLICATION_PERFORMANCE

BusinessTransactionPerformance

The scope of the affected entity is set to business transaction performance.

Property	Type	Enums
affectedEntityType*	string	BUSINESS_TRANSACTION_PERFORMANCE

affectedBusinessTransactions*	string	**AffectedBusinessTransactions** **Property** / **Type** / **Enums** businessTransactionScope* / string / ALL_BUSINESS_TRANSACTIONS, SPECIFIC_BUSINESS_TRANSACTIONS, BUSINESS_TRANSACTIONS_IN_SPECIFIC_TIERS, BUSINESS_TRANSACTIONS_MATCHING_PATTERN

AffectedBusinessTransactions

Property	Type	Enums
businessTransactionScope*	string	ALL_BUSINESS_TRANSACTIONS SPECIFIC_BUSINESS_TRANSACTIONS BUSINESS_TRANSACTIONS_IN_SPECIFIC_TIERS BUSINESS_TRANSACTIONS_MATCHING_PATTERN

AllBusinessTransactions

The scope of business transactions is set to all business transactions.

Property	Type	Enums
businessTransactionScope*	string	ALL_BUSINESS_TRANSACTIONS

SpecificBusinessTransactions

The scope of business transactions is set to specific business transactions.

Property	Type	Enums
businessTransactionScope*	string	SPECIFIC_BUSINESS_TRANSACTIONS
businessTransactions*	string minItems: 1	

BusinessTransactionsInSpecificTiers

The scope of business transactions is set to business transactions for specific tiers.

Property	Type	Enums
businessTransactionScope*	string	BUSINESS_TRANSACTIONS_IN_SPECIFIC_TIERS
specificTiers*	string minItems: 1	

BusinessTransactionsMatchingPattern

The scope of business transactions is set to business transactions matching a certain pattern.

Property	Type	Enums
businessTransactionScope*	string	BUSINESS_TRANSACTIONS_MATCHING_PATTERN

specificTiers*	string	**EntityMatchingPattern** **Property** / **Type** / **Enums** matchTo* / string / STARTS_WITH, ENDS_WITH, CONTAINS, EQUALS, MATCH_REG_EX matchValue* / string, minLength: 1 / shouldNot / boolean /

TierNodeTransactionPerformance

The type of affected entities is set to the performance of tier and node transactions.

Property	Type	Enums
affectedEntityType*	string	TIER_NODE_TRANSACTION_PERFORMANCE
affectedEntities*	string	**AffectedTierOrNodeEntities** **Property** / **Type** / **Enums** tierOrNode* / string / TIER_AFFECTED_ENTITIES, NODE_AFFECTED_ENTITIES

AffectedTierOrNodeEntities

Property	Type	Enums
tierOrNode*	string	TIER_AFFECTED_ENTITIES NODE_AFFECTED_ENTITIES

TierAffectedEntities

The scope of affected entities is set to tiers.

Property	Type	Enums
tierOrNode*	string	TIER_AFFECTED_ENTITIES NODE_AFFECTED_ENTITIES
affectedTiers*	string	**AffectedTiers** **Property** / **Type** / **Enums** affectedTierScope* / string / ALL_TIERS, SPECIFIC_TIERS

NodeAffectedEntities

The scope of affected entities is set to nodes.

Property	Type	Enums
`tierOrNode`*	string	`TIER_AFFECTED_ENTITIES` `NODE_AFFECTED_ENTITIES`

`typeofNode`*	string	`ALL_NODES`
		`JAVA_NODES`
		`DOT_NET_NODES`
		`PHP_NODES`

`affectedNodes`*	string	**AffectedNodes** **Property** / **Type** / **Enums** `affectedNodeScope`* / string / `ALL_NODES`, `SPECIFIC_NODES`, `NODES_OF_SPECIFIC_TIERS`, `NODES_MATCHING_PATTERN`, `NODE_PROPERTY_VARIABLE_MATCHER`

TierNodeHardware

The affected entity type is set to tier node hardware.

Property	Type	Enums
`affectedEntityType`*	string	`TIER_NODE_HARDWARE`
`affectedEntities`*	string	**AffectedTierOrNodeEntities** **Property** / **Type** / **Enums** `tierOrNode`* / string / `TIER_AFFECTED_ENTITIES`, `NODE_AFFECTED_ENTITIES`

ServersInApplication

The affected entity type is set to **servers in the application**.

Property	Type	Enums
`affectedEntityType`*	string	`SERVERS_IN_APPLICATION`
`affectedServers`*	string	**ApplicationAffectedServers** **Property** / **Type** / **Enums** `serversScope`* / string / `ALL_SERVERS_IN_APPLICATION`, `SPECIFIC_SERVERS_IN_APPLICATION`, `ALL_SERVERS_IN_SPECIFIC_TIERS`

ApplicationAffectedServers

Property	Type	Enums

serversScope*	string	ALL_SERVERS_IN_APPLICATION SPECIFIC_SERVERS_IN_APPLICATION ALL_SERVERS_IN_SPECIFIC_TIERS

AllServersInApplication

The scope of servers is set to all servers in the application.

Property	Type	Enums
serversScope*	string	ALL_SERVERS_IN_APPLICATION

SpecificServersInApplication

The scope of servers is set to specific servers in the application.

Property	Type	Enums
serversScope*	string	SPECIFIC_SERVERS_IN_APPLICATION
specificServers*	string minLength: 1	

AllServersInSpecificTiers

The scope of servers is set to all servers in specific tiers.

Property	Type	Enums
serversScope*	string	ALL_SERVERS_IN_SPECIFIC_TIERS
specificTiers*	string minItems: 1	

Backends

The affected entity type is set to backends.

Property	Type	Enums
affectedEntityType*	string	BACKENDS
affectedBackends*	string	**AffectedBackends** Property / Type / Enums backendScope* / string / ALL_BACKENDS, SPECIFIC_BACKENDS, BACKENDS_MATCHING_PATTERN

AffectedBackends

Property	Type	Enums
backendScope*	string	ALL_BACKENDS SPECIFIC_BACKENDS BACKENDS_MATCHING_PATTERN

AllBackends

The scope of backends is set to all backends.

Property	Type	Enums
backendScope*	string	ALL_BACKENDS

SpecificBackends

The scope of backends is set to specific backends.

Property	Type	Enums
backendScope*	string	SPECIFIC_BACKENDS
backends*	string minItems: 1	

BackendsMatchingPattern

The scope of backends is set to backends matching a specific pattern.

Property	Type	Enums
backendScope*	string	BACKENDS_MATCHING_PATTERN
patternMatcher*	string	**EntityMatchingPattern** Property / Type / Enums matchTo* — string — STARTS_WITH, ENDS_WITH, CONTAINS, EQUALS, MATCH_REG_EX matchValue* — string, minLength: 1 shouldNot — boolean

Errors

The affected entity type is set to errors.

Property	Type	Enums
affectedEntityType*	string	ERRORS
affectedErrors*	string	**AffectedErrors** Property / Type / Enums errorScope* — string — ALL_ERRORS, SPECIFIC_ERRORS, ERRORS_OF_SPECIFIC_TIERS, ERRORS_MATCHING_PATTERN

AffectedErrors

Property	Type	Enums

`errorScope`*	string	`ALL_ERRORS`
		`SPECIFIC_ERRORS`
		`ERRORS_OF_SPECIFIC_TIERS`
		`ERRORS_MATCHING_PATTERN`

AllErrors

The scope of errors is set to all errors.

Property	Type	Enums
`errorScope`*	string	`ALL_ERRORS`

SpecificErrors

The scope of errors is set to specific errors.

Property	Type	Description/Enums
`errorScope`*	string	`SPECIFIC_ERRORS`
errors*	string minItems: 1	
tiers	string minItems: 1	Use if error names belong to multiple tiers. Provide the tier names as a list in `affects.affectedErrors.tiers` in the same order as the error names. In the following example, 'Page Not Found: 404' corresponds to `Tier1`, while 'Error2' corresponds to `SomeOtherTier`. - If the error names are unique to each tier, tiers map errors one to one. You need not specify the tier name in this case. - If errors have same name but belong to different tiers, duplicate names are handled by appending tier names with the delimiter '|'. - If there is a mismatch between the number of error names and tier names, then the list `affects.affectedErrors.tiers` is ignored. - If an error belongs to multiple tiers, and you do not specify the tier names, the error name cannot be uniquely identified. Hence, an error is displayed.

```
{
    "id": 33,
    "name": "Specific Error",
    "enabled": true,
    "useDataFromLastNMinutes": 1,
    "waitTimeAfterViolation": 1,
    "scheduleName": "Always",
    "affects": {
        "affectedEntityType": "ERRORS",
        "affectedErrors": {
            "errorScope": "SPECIFIC_ERRORS",
            "errors": [
                "Page Not Found : 404", "Error2"
            ],
            "tiers": [
                "Tier1", "SomeOtherTier"
            ]
        }
    },
    "evalCriterias": {
        "criticalCriteria": {
            "conditionAggregationType": "ALL",
            "conditionExpression": null,
            "conditions": [
                {
                    "name": "Condition 1",
                    "shortName": "A",
                    "evaluateToTrueOnNoData": false,
                    "evalDetail": {
                        "evalDetailType": "SINGLE_METRIC",
                        "metricAggregateFunction": "VALUE",
                        "metricPath": "Errors per Minute",
                        "metricEvalDetail": {
                            "metricEvalDetailType": "SPECIFIC_TYPE",
                            "compareCondition": "GREATER_THAN_SPECIFIC_VALUE",
                            "compareValue": 10
                        }
                    },
                    "triggerEnabled": false,
                    "minimumTriggers": 1
                }
            ],
            "evalMatchingCriteria": {
                "matchType": "ANY_NODE",
                "value": null
            }
        },
        "warningCriteria": null
    }
}
```


ErrorsOfSpecificTiers

The scope of errors is set to errors related to specific tiers.

Property	Type	Enums
`errorScope`*	string	`ERRORS_OF_SPECIFIC_TIERS`
`specificTiers`*	string minItems: 1	

ErrorsMatchingPattern

The scope of errors is set to errors matching a specific pattern.

Property	Type	Enums
errorScope*	string	ERRORS_MATCHING_PATTERN
patternMatcher	string	**EntityMatchingPattern** Property / Type / Enums matchTo* — string — STARTS_WITH, ENDS_WITH, CONTAINS, EQUALS, MATCH_REG_EX matchValue* — string, minLength: 1 shouldNot — boolean

ServiceEndpoints

The affected entity type is set to service endpoints.

Property	Type	Enums
affectedEntityType*	string	SERVICE_ENDPOINTS
affectedServiceEndpoints*		**AffectedServiceEndpoints** Property / Type / Enums serviceEndpointScope* — string — ALL_SERVICE_ENDPOINTS, SPECIFIC_SERVICE_ENDPOINTS, SERVICE_ENDPOINTS_IN_SPECIFIC_TIERS, SERVICE_ENDPOINTS_MATCHING_PATTERN

AffectedServiceEndpoints

Property	Type	Enums
serviceEndpointScope*	string	ALL_SERVICE_ENDPOINTS SPECIFIC_SERVICE_ENDPOINTS SERVICE_ENDPOINTS_IN_SPECIFIC_TIERS SERVICE_ENDPOINTS_MATCHING_PATTERN

AllServiceEndpoints

The scope of service endpoints is set to all service endpoints.

Property	Type	Enums
serviceEndpointScope*	string	ALL_SERVICE_ENDPOINTS

SpecificServiceEndpoints

The scope of service endpoints is set to specific service endpoints.

Property	Type	Enums
serviceEndpointScope*	string	SPECIFIC_SERVICE_ENDPOINTS
serviceEndpoints*	string minItems: 1	

ServiceEndpointsInSpecificTiers

The scope of service endpoints is set to specific service endpoints.

Property	Type	Enums
serviceEndpointScope*	string	SERVICE_ENDPOINTS_IN_SPECIFIC_TIERS
specificTiers*	string minItems: 1	

ServiceEndpointsMatchingPattern

The scope of service endpoints is set to service endpoints that match a specific pattern.

Property	Type	Enums
serviceEndpointScope*	string	SERVICE_ENDPOINTS_MATCHING_PATTERN
patternMatcher*	string minItems: 1	**EntityMatchingPattern** matchTo* (string): STARTS_WITH, ENDS_WITH, CONTAINS, EQUALS, MATCH_REG_EX matchValue* (string, minLength: 1) shouldNot (boolean)

InformationPoints

The affected entity type is set to information points.

Property	Type	Enums
affectedEntityType*	string	INFORMATION_POINTS
affectedInformationPoints*	string	**AffectedInformationPoints** informationPointScope* (string): ALL_INFORMATION_POINTS, SPECIFIC_INFORMATION_POINTS, INFORMATION_POINTS_MATCHING_PATTERN

AffectedInformationPoints

Property	Type	Enums

`informationPointScope`*	string	`ALL_INFORMATION_POINTS`
		`SPECIFIC_INFORMATION_POINTS`
		`INFORMATION_POINTS_MATCHING_PATTERN`

AllInformationPoints

The scope of information points is set to all information points.

Property	Type	Enums
`informationPointScope`*	string	`ALL_INFORMATION_POINTS`

SpecificInformationPoints

The scope of information points is set to specific information points.

Property	Type	Enums
`informationPointScope`*	string	`SPECIFIC_INFORMATION_POINTS`
`informationPoints`*	string minItems: 1	

InformationPointsMatchingPattern

The scope of information points is set to information points matching a pattern.

Property	Type	Enums
`informationPointScope`*	string	`INFORMATION_POINTS_MATCHING_PATTERN`
`patternMatcher`*	string minItems: 1	**EntityMatchingPattern** `matchTo`* — string — `STARTS_WITH`, `ENDS_WITH`, `CONTAINS`, `EQUALS`, `MATCH_REG_EX` `matchValue`* — string, minLength: 1 `shouldNot` — boolean

Custom

The affected entity type is set to custom.

Property	Type	Enums
`affectedEntityType`*	string	`CUSTOM`

`affectedEntityScope`*	string	**AffectedEntityScope** **Property** / **Type** / **Enums** `entityScope`* / string / `APPLICATION_PERFORMANCE`, `SPECIFIC_ENTITY_PERFORMANCE`

AffectedEntityScope

Property	Type	Enums
`entityScope`*	string	`APPLICATION_PERFORMANCE` `SPECIFIC_ENTITY_PERFORMANCE`

ApplicationPerformance

The scope of the affected entity is set to application performance.

Property	Type	Enums
`entityScope`*	string	`APPLICATION_PERFORMANCE`

SpecificEntityPerformance

The scope of the affected entity is set to specific entity performance.

Property	Type	Enums
`entityScope`*	string	`SPECIFIC_ENTITY_PERFORMANCE`

`entityType`*	string	`BUSINESS_TRANSACTION`
		`NODE`
		`SERVER`

`affectedEntityName`*	string minLength: 1	

Databases

The affected entity type is set to databases.

Property	Type	Enums
`affectedEntityType`*	string	`DATABASES`

databaseType*	string	ALL_DATABASE_TYPES COUCHBASE DB2 MONGO_DB MICROSOFT_SQL_SERVER MYSQL ORACLE POSTGRE_SQL AZURE_SQL SYBASE
affectedDatabases*	string	**AffectedDatabases** **Property** / **Type** / **Enums** databaseScope / string / ALL_DATABASES, SPECIFIC_DATABASES

AffectedDatabases

Property	Type	Enums
databaseScope*	string	ALL_DATABASES SPECIFIC_DATABASES

AllDatabases

The scope of affected databases is set to all databases.

Property	Type	Enums
databaseScope*	string	ALL_DATABASES

SpecificDatabases

The scope of affected databases is set to specific databases.

Property	Type	Enums
databaseScope*	string	SPECIFIC_DATABASES
databases*	string minItems: 1	**DbServer** **Property** / **Type** serverName* / string, minLength: 1 collectorConfigName* / string, minLength: 1

DbServer

Property	Type

`serverName`*	string minLength: 1
`collectorConfigName`*	string minLength: 1

Servers

The affected entity type is set to servers.

Property	Type	Enums
`affectedEntityType`*	string	`SERVERS`
`serverSelectionCriteria`*	string	**AffectedServersCriteria** **Property** / **Type** / **Enums** `selectServersBy`* / string / `AFFECTED_SERVER_SUBGROUPS`, `AFFECTED_SERVERS`

AffectedServersCriteria

Property	Type	Enums
`selectServersBy`*	string	`AFFECTED_SERVER_SUBGROUPS` `AFFECTED_SERVERS`

AffectedServers

Property	Type	Enums
`selectServersBy`*	string	`AFFECTED_SERVER_SUBGROUPS` `AFFECTED_SERVERS`
`affectedServers`*	string	**ServerSelectionCriteria** **Property** / **Type** / **Enums** `severSelectionScope`* / string / `ALL_SERVERS_IN_ACCOUNT`, `SERVERS_WITHIN_SUBGROUP`, `SPECIFIC_SERVERS`, `SERVERS_MATCHING_PATTERN`

ServerSelectionCriteria

Property	Type	Enums

`severSelectionScope`*	string	`ALL_SERVERS_IN_ACCOUNT`
		`SERVERS_WITHIN_SUBGROUP`
		`SPECIFIC_SERVERS`
		`SERVERS_MATCHING_PATTERN`

AllServersInAccount

The scope of servers is set to all servers within an account.

Property	Type	Enums
severSelectionScope*	string	ALL_SERVERS_IN_ACCOUNT

ServersWithinSubGroup

The scope of servers is set to all servers within a subgroup.

Property	Type	Enums
severSelectionScope*	string	SERVERS_WITHIN_SUBGROUP
subGroups*	string minItems: 1	

SpecificServers

The scope of servers is set to specific servers.

Property	Type	Enums
severSelectionScope*	string	SPECIFIC_SERVERS
servers*	string minItems: 1	

ServersMatchingPattern

The scope of servers is set to servers matching a specific pattern.

Property	Type	Enums
severSelectionScope*	string	SERVERS_MATCHING_PATTERN
patternMatcher*		**EntityMatchingPattern** matchTo* (string): STARTS_WITH, ENDS_WITH, CONTAINS, EQUALS, MATCH_REG_EX matchValue* (string, minLength: 1) shouldNot (boolean)

AffectedServerSubGroups

The scope of affected servers is set to affected servers within a subgroup.

Property	Type	Enums
selectServersBy*	string	AFFECTED_SERVER_SUBGROUPS

subGroups*	string minItems: 1	**AffectedSubGroups** **Property** / **Type** / **Enums** subGroupScope* / string / ALL_SUBGROUPS, SPECIFIC_SERVER_SUB_GROUPS

AffectedSubGroups

Property	Type	Enums
subGroupScope*	string	ALL_SUBGROUPS SPECIFIC_SERVER_SUB_GROUPS

AllSubGroups

The scope of affected servers is set to servers within all subgroups.

Property	Type	Enums
subGroupScope*	string	ALL_SUBGROUPS

SpecificServerSubGroups

The scope of affected servers is set to **servers within specific subgroups**.

Property	Type	Enums
subGroupScope*	string	SPECIFIC_SERVER_SUB_GROUPS
subGroupNames	string minItems: 1	

Affects

Describes what entities the health rule affects. For example, business transactions, servers, or databases.

Property	Type	Enums
affectedEntityType*	string	OVERALL_APPLICATION_PERFORMANCE BUSINESS_TRANSACTION_PERFORMANCE TIER_NODE_TRANSACTION_PERFORMANCE TIER_NODE_HARDWARE SERVERS_IN_APPLICATION BACKENDS ERRORS SERVICE_ENDPOINTS INFORMATION_POINTS CUSTOM DATABASES SERVERS

EvalCriterias

Property	Type/Enums
`criticalCriteria`	**Criteria** Critical conditions are evaluated before warning conditions. If you have defined a critical condition and a warning condition in the same health rule, the warning condition is evaluated only if the critical condition is not `true`.

Property	Description	Type/Enums

`conditionAggregationType`	Condition evaluation criteria that constitute a health rule violation.	String
		
		default: ALL
		
		Enums
		
		`ALL`
		
		`ANY`
		
		`CUSTOM`

`conditionExpression`	Use only when you set the `conditionAggregationType` variable to `CUSTOM`. Use the `ShortName` of the condition to define the boolean expression.	String minLength: 1

conditions*	A single condition that can be evaluated independently to `true` or `false`. OR
	
	A list of conditions (maximum of 8) along with other properties to form a criteria.
		String
		
		minItems: 1
		
		Condition
		
		Property / **Description** / **Type/Enums**
		
		`name`* — Name of the condition. — string
		
		`shortname`* — A short name used in `conditionExpression` to evaluate `CUSTOM conditionType`. — string
		
		pattern: ^[A-Z]{1,3}$
		
		Enums
		
		A
		
		B
		
		C
		
		D
		
		E
		
		F
		
		G
		
		H
		
		`evaluateToTrueOnNoData` — Controls the evaluation of the condition in cases where any metric on which the condition is based, returns no data. The default when no data is returned is for the condition to evaluate to `unknown`. If the health rule is based on all the conditions evaluating to `true`, having no data returned may affect whether the health rule triggers an action. — boolean
		
		default: `false`
		
		`evalDetail`* — Details of metric(s) considered for evaluation of the condition. Use `SINGLE_METRIC` to evaluate a single metric. Use `METRIC_EXPRESSION` to evaluate a metric expression. — **ConditionEvalDetail**
		
		Property / **Type** / **Enums**
		
		`evalDetailType` — `string` — `SINGLE_METRIC`, `METRIC_EXPRESSION`
		
		`triggerEnabled` — If set to `true`, the value in field `minimumTriggers` is considered for evaluation. — `boolean`
		
		`default: false`
		
		`minimumTriggers` — If you set a non-zero value, persistence thresholds are considered when evaluating the conditions. Ensure that you define a value less than `useDataFromLastNMinutes`. — `integer`
		`default: 0`
		`minimum: 0`
		`maximum: 30`

`evalMatchingCriteria`	The criteria for evaluating a condition for the following health rule types:
	• Business Transaction
	• Node health-hardware
	• Node health-transaction performance
	
	It defines how many nodes in the affected entities must violate the condition before the health rule is considered to violate.
		NodeEvalMatchingCriteria
		
		Property / **Type /Enums** / **Description**
		
		`matchType` — String
		Enums
		`AVERAGE`
		`ANY_NODE`
		`PERCENTAGE_NODES`
		`NUMBER_OF_NODES`
		The evaluation scope type that defines how many nodes in the affected entities must violate the condition before the health rule is considered violated.
		Enter the matching criteria only when you select `PERCENTAGE_NODES` or `NUMBER_OF_NODES` as `matchType`.
		
		`value` — The number or percentage of nodes that must violate the condition to constitute a health rule violation. If you select `NUMBER_OF_NODES`, enter an integer. If you select `PERCENTAGE_NODES`, enter a number.

`warningCriteria`

Criteria

If you have defined a critical condition and a warning condition in the same health rule, the warning condition is evaluated only if the critical condition is not `true`.

Property	Description	Type/Enums

`conditionAggregationType`	Condition evaluation criteria that constitute a health rule violation.	String
		default: ALL
		Enums
		`ALL`
		`ANY`
		`CUSTOM`

`conditionExpression`	Use only when you set the `conditionAggregationType` variable to `CUSTOM`. Use the `ShortName` of the condition to define the boolean expression.	String minLength: 1

<table>
<tr>
<td>conditions*</td>
<td>A single condition that can be evaluated independently to true or false. OR

A list of conditions (maximum of 8) along with other properties to form a criteria.</td>
<td>String

minItems: 1

Condition
<table>
<tr><th>Property</th><th>Description</th><th>Type/Enums</th></tr>
<tr><td>name*</td><td>Name of the condition.</td><td>string</td></tr>
<tr><td>shortname*</td><td>A short name used in conditionExpression to evaluate CUSTOM conditionType.</td><td>string

pattern: ^[A-Z]{1,3}$

Enums

A

B

C

D

E

F

G

H</td></tr>
<tr><td>evaluateToTrueOnNoData</td><td>Controls the evaluation of the condition in cases where any metric on which the condition is based, returns no data. The default when no data is returned is for the condition to evaluate to unknown. If the health rule is based on all the conditions evaluating to true, having no data returned may affect whether the health rule triggers an action.</td><td>boolean

default: false</td></tr>
<tr><td>evalDetail*</td><td>Details of metric(s) considered for evaluation of the condition. Use SINGLE_METRIC to evaluate a single metric. Use METRIC_EXPRESSION to evaluate a metric expression.</td><td>ConditionEvalDetail
<table>
<tr><th>Property</th><th>Type</th><th>Enums</th></tr>
<tr><td>evalDetailType</td><td>string</td><td>SINGLE_METRIC

METRIC_EXPRESSION</td></tr>
</table></td></tr>
<tr><td>triggerEnabled</td><td>If set to true, the value in field minimumTriggers is considered for evaluation.</td><td>boolean

default: false</td></tr>
<tr><td>minimumTriggers</td><td>If you set a non-zero value, persistence thresholds are considered when evaluating the conditions. Ensure that you define a value less than useDataFromLastNMinutes.</td><td>integer
default: 0
minimum: 0
maximum: 30</td></tr>
</table></td>
</tr>
</table>

<table>
<tr>
<td>evalMatchingCriteria</td>
<td>The criteria for evaluating a condition for the following health rule types:

• Business Transaction
• Node health-hardware
• Node health-transaction performance

It defines how many nodes in the affected entities must violate the condition before the health rule is considered to violate.</td>
<td>
NodeEvalMatchingCriteria
<table>
<tr><th>Property</th><th>Description</th><th>Type /Enums</th></tr>
<tr><td>matchType</td><td>The evaluation scope type that defines how many nodes in the affected entities must violate the condition before the health rule is considered violated.

Enter the matching criteria only when you select PERCENTAGE_NODES or NUMBER_OF_NODES as matchType.</td><td>String

Enums

AVERAGE

ANY_NODE

PERCENTAGE_NODES
NUMBER_OF_NODES</td></tr>
<tr><td>value</td><td>The number or percentage of nodes that must violate the condition to constitute a health rule violation. If you select NUMBER_OF_NODES, enter an integer. If you select PERCENTAGE_NODES, enter a number.</td><td></td></tr>
</table>
</td>
</tr>
</table>

HealthRuleSummary

Property	Type
id*	integer
name	string
enabled*	boolean

ErrorResponse

Property	Type
statusCode	integer
message	string

***This property is required (mandatory).**

Download Examples

Download a set of examples that help you configure a health rule, health_rule_request_examples.zip.

Download Swagger YAML spec

Download the Swagger YAML spec health_rule_openapi.yml.

Schedule API

This page describes the Schedule API methods you can use to create, configure, and manage the evaluation time frame for the health rules of an application. The metrics associated with a health rule are evaluated according to a schedule that you control. See Health Rule Schedules.

- Syntax validation of the JSON payload is done when creating the schedule.
- Ensure that you pick a time zone the Time Zone List.

Create a New Schedule

Creates a new schedule with the specified JSON payload. See Property Details.

Resource URL

```
POST <controller_url>/controller/alerting/rest/v1
/applications/<application_id>/schedules
```

Request/Response Format

JSON

Example

This example creates a health rule schedule that evaluates the health rule once. See Download Examples.

Retrieve a List of Schedules for a Given Application

Returns a list of schedule(s) details for a health rule associated with the specified application ID. This API returns the schedule ID, name, and description of the schedule. See Property Details.

Resource URL

```
GET <controller_url>/controller/alerting/rest/v1/applications/<application_id>/schedules
```

Response Format

JSON

Example Response

This example returns a list of schedules applicable to a given application ID.

```
[
    {
        "id": 62,
        "name": "Daily Schedule",
        "description": "Daily Schedule",
        "timezone": "America/Los_Angeles"
    },
    {
        "id": 12,
        "name": "End of Business Hour: 5pm-6pm, Mon-Fri",
        "description": "This schedule is active Monday through Friday, during end of business hour",
        "timezone": "Asia/Kolkata"
    },
    {
        "id": 61,
        "name": "Schedule1",
        "description": "Custom Schedule",
        "timezone": "America/Los_Angeles"
    },
    {
        "id": 11,
        "name": "Weekday lunch: 12pm-1pm, Mon-Fri",
        "description": "This schedule is active Monday through Friday, during lunch hour",
        "timezone": "Asia/Kolkata"
    },
    {
        "id": 10,
        "name": "Weekday mornings: 8am-12pm, Mon-Fri",
        "description": "This schedule is active Monday through Friday, during morning hours",
        "timezone": "Asia/Kolkata"
    },
    {
        "id": 7,
        "name": "Weekdays: 8am-5pm, Mon-Fri",
        "description": "This schedule is active Monday through Friday, during business hours",
        "timezone": "Asia/Kolkata"
    },
    {
        "id": 9,
        "name": "Weekends: 12am-11pm, Sat-Sun",
        "description": "This schedule is active all day and night on the weekend",
        "timezone": "Asia/Kolkata"
    },
    {
        "id": 8,
        "name": "Weeknights: 11pm-6am, Mon-Fri",
        "description": "This schedule is active Monday through Friday, during night time batch runs",
        "timezone": "Asia/Kolkata"
    }
]
```


Retrieve the Details of a Specified Schedule

Retrieves a schedule with a specified ID. See Property Details.

Resource URL

`GET <controller_url>/controller/alerting/rest/v1/applications/<application_id>/schedules/{schedule-id}`

Response Format

`JSON`

Example Response

This example retrieves the details of a schedule. See Download Examples.

```
{
    "id": 12,
    "name": "End of Business Hour: 5pm-6pm, Mon-Fri",
    "description": "This schedule is active Monday through Friday, during end of business hour",
    "timezone": "Asia/Kolkata",
    "scheduleConfiguration": {
        "scheduleFrequency": "WEEKLY",
        "days": [
            "MONDAY",
            "TUESDAY",
            "WEDNESDAY",
            "THURSDAY",
            "FRIDAY"
        ],
        "startTime": "17:00",
        "endTime": "18:00"
    }
}
```

Update a Schedule

Updates an existing schedule with a specified JSON payload. See Property Details.

Resource URL

`PUT <controller_url>/controller/alerting/rest/v1/applications/<application_id>/schedules/{schedule-id}`

Request/Response Format

`JSON`

Example

This example updates a schedule that evaluates the health rule once. See Download Examples.

Delete a Schedule

Delete a schedule with the specified ID. See Property Details.

Resource URL

`DELETE <controller_url>/controller/alerting/rest/v1/applications/<application_id>/schedules/{schedule-id}`

Response Codes

Code	Description
200	Fetched successfully
201	Created successfully
204	Deleted successfully
400	Bad request
401	Unauthorized
403	Forbidden
404	Resource not found
409	Already exists

Property Details

Schedule

Payload details for the health rule schedule.

Property Name	Type	Description and Valid Values
`id`	integer	Auto-generated by the system and returned in the response. It is a `ReadOnly` value.
`name`*	string	Name of schedule to be associated with the health rule for evaluation. Minimum length: 1
`description`	string	Description of the schedule with evaluation details. Example: Health rules are evaluated daily at 12 noon. Default value: `true`
`timezone`*	string	Timezone Id - Unique identifier of the time zone. See Time Zones. Example: America/Los_Angeles

`scheduleConfiguration`*		**Property Name** / **Type** / **Description**
		`scheduleFrequency` / string / The health rule(s) evaluation frequency specified in the schedule.
		Enums:
		`ONE_TIME`
		`DAILY`
		`WEEKLY`
		`MONTHLY_SPECIFIC_DATE`
		`MONTHLY_SPECIFIC_DAY`
		`CUSTOM`

OneTimeSchedule

This schedule evaluates the health rule once.

Property Name	Type	Description
`scheduleFrequency`*	string	The frequency to evaluate the health rule(s). See Enums.
`startDate`*	string	The scheduled start date in DD/MM/YYYY format. Pattern: `^(?:(?:31(\/|-|\.)(?:0?[13578]|1[02]))\1|(?:(?:29|30)(\/|-|\.)(?:0?[13-9]|1[0-2])\2))(?:(?:1[6-9]|[2-9]\d)?\d{2})$\|^(?:29(\/\|-\|\.)0?2\3(?:(?:(?:1[6-9]\|[2-9]\d)?(?:0[48]\|[2468][048]\|[13579][26])\|(?:(?:16\|[2468][048]\|[3579][26])00))))$|^(?:0?[1-9]|1\d|2[0-8])(\/|-|\.)(?:(?:0?[1-9])|(?:1[0-2]))\4(?:(?:1[6-9]|[2-9]\d)?\d{2})$`
`startTime`*	string	The scheduled start time in a 24-hour format. Pattern: `^([01]\d|2[0-3]):([0-5]\d)$`

endDate*	string	The scheduled end date in DD/MM/YYYY format. Pattern: ^(?:(?:31(\/|-|\.)(?:0?[13578]|1[02]))\1|(?:(?:29|30)(\/|-|\.)(?:0?[13-9]|1[0-2])\2))(?:(?:1[6-9]|[2-9]\d)?\d{2})$\|^(?:29(\/\|-\|\.)0?2\3(?:(?:(?:1[6-9]\|[2-9]\d)?(?:0[48]\|[2468][048]\|[13579][26])\|(?:(?:16\|[2468][048]\|[3579][26])00))))$|^(?:0?[1-9]|1\d|2[0-8])(\/|-|\.)(?:(?:0?[1-9])|(?:1[0-2]))\4(?:(?:1[6-9]|[2-9]\d)?\d{2})$
endTime*	string	The scheduled end time in a 24-hour format. Pattern: ^([01]\d|2[0-3]):([0-5]\d)$

DailySchedule

A recurring schedule to evaluate the health rule every day.

Property Name	Type	Description
scheduleFrequency*	string	The frequency to evaluate the health rule(s). See Enums.
startTime*	string	The scheduled start time in a 24-hour format. pattern: ^([01]\d|2[0-3]):([0-5]\d)$
endTime*	string	The scheduled end time in a 24-hour format. pattern: ^([01]\d|2[0-3]):([0-5]\d)$

WeeklySchedule

A recurring schedule to evaluate the health rule every week.

Property Name	Type	Description
scheduleFrequency*	string	The frequency to evaluate the health rule(s). See Enums.
days*	string	The day of the week to evaluate the health rule. minItems: 1 maxItems: 7
startTime*	string	The scheduled start time in a 24-hour format. pattern: ^([01]\d|2[0-3]):([0-5]\d)$

Property Name	Type	Description
DayOfWeek	string	**Enums** SUNDAY MONDAY TUESDAY WEDNESDAY THURSDAY FRIDAY SATURDAY

endTime*	string	The schedule end time in 24-hour format. pattern: ^([01]\d|2[0-3]):([0-5]\d)$

MonthlySpecificDateSchedule

A recurring schedule to evaluate the health rule every month on a specific date.

Property Name	Type	Description
scheduleFrequency*	string	The frequency to evaluate the health rule(s). See Enums.
startDate*	string	The schedule start date in DD/MM/YYYY format. pattern: ^(?:(?:31(\/|-|\.)(?:0?[13578]|1[02]))\1|(?:(?:29|30)(\/|-|\.)(?:0?[13-9]|1[0-2])\2))(?:(?:1[6-9]|[2-9]\d)?\d{2})$\|^(?:29(\/\|-\|\.)0?2\3(?:(?:(?:1[6-9]\|[2-9]\d)?(?:0[48]\|[2468][048]\|[13579][26])\|(?:(?:16\|[2468][048]\|[3579][26])00))))$|^(?:0?[1-9]|1\d|2[0-8])(\/|-|\.)(?:(?:0?[1-9])|(?:1[0-2]))\4(?:(?:1[6-9]|[2-9]\d)?\d{2})$
startTime*	string	The scheduled start time in a 24-hour format. pattern: ^([01]\d|2[0-3]):([0-5]\d)$
endDate*	string	The scheduled end date in DD/MM/YYYY format. pattern: ^(?:(?:31(\/|-|\.)(?:0?[13578]|1[02]))\1|(?:(?:29|30)(\/|-|\.)(?:0?[13-9]|1[0-2])\2))(?:(?:1[6-9]|[2-9]\d)?\d{2})$\|^(?:29(\/\|-\|\.)0?2\3(?:(?:(?:1[6-9]\|[2-9]\d)?(?:0[48]\|[2468][048]\|[13579][26])\|(?:(?:16\|[2468][048]\|[3579][26])00))))$|^(?:0?[1-9]|1\d|2[0-8])(\/|-|\.)(?:(?:0?[1-9])|(?:1[0-2]))\4(?:(?:1[6-9]|[2-9]\d)?\d{2})$
endTime*	string	The scheduled end time in a 24-hour format. pattern: ^([01]\d|2[0-3]):([0-5]\d)$

MonthlySpecificDaySchedule

A recurring schedule to evaluate the health rule every month on a specific date.

Property Name	Type	Description
scheduleFrequency*	string	The frequency to evaluate the health rule(s). See Enums.
startTime*	string	The scheduled start time in a 24-hour format. pattern: ^([01]\d|2[0-3]):([0-5]\d)$
endTime*	string	The schedules end time in a 24-hour format. pattern: ^([01]\d|2[0-3]):([0-5]\d)$

<table>
<tr><td>day*</td><td>string</td><td>The day of the week to evaluate the health rule.
minItems: 1
maxItems: 7<table><tr><th>Property Name</th><th>Type</th><th>Description</th></tr><tr><td>DayOfWeek</td><td>string</td><td>Enums
SUNDAY
MONDAY
TUESDAY
WEDNESDAY
THURSDAY
FRIDAY
SATURDAY</td></tr></table></td></tr>
<tr><td>occurrence*</td><td>string</td><td>The occurrence of the day within a month.
Enums:
FIRST
SECOND
THIRD
FOURTH
LAST</td></tr>
</table>

CustomSchedule

A custom schedule to evaluate the health rule based on specific requirements and timezone. Use UNIX cron expressions to define properties.

Property Name	Type	Description
scheduleFrequency*	string	The frequency to evaluate the health rule(s). See Enums.
startCron*	string	The beginning of the UNIX cron expression.
endCron*	string	The end of the UNIX cron expression.

ScheduleSummaryElement

The details of the schedule you have defined that are returned when you retrieve the schedules API.

Property Name	Type	Description
id	integer	Auto-generated by the system and returned in the response.
name*	string	Name of the schedule associated with the health rule for evaluation.
description	string	Description of the schedule and evaluation details.
timezone*	string	Timezone ID - Unique identifier of the time zone.

***This property is required (mandatory).**

Download Examples

Download a set of examples that help you configure a schedule, Schedule Examples.zip.

Download SWAGGER YAML file

Download the Swagger YAML spec schedule_openapi.yml.

List of Time Zone IDs

Sl.No	Time Zone
1	ACT
2	AET
3	AGT
4	ART
5	AST
6	Africa/Abidjan
7	Africa/Accra
8	Africa/Addis_Ababa
9	Africa/Algiers
10	Africa/Asmara
11	Africa/Asmera
12	Africa/Bamako
13	Africa/Bangui
14	Africa/Banjul
15	Africa/Bissau
16	Africa/Blantyre
17	Africa/Brazzaville
18	Africa/Bujumbura
19	Africa/Cairo
20	Africa/Casablanca
21	Africa/Ceuta
22	Africa/Conakry
23	Africa/Dakar
24	Africa/Dar_es_Salaam
25	Africa/Djibouti
26	Africa/Douala
27	Africa/El_Aaiun
28	Africa/Freetown
29	Africa/Gaborone
30	Africa/Harare
31	Africa/Johannesburg
32	Africa/Juba
33	Africa/Kampala
34	Africa/Khartoum
35	Africa/Kigali
36	Africa/Kinshasa
37	Africa/Lagos
38	Africa/Libreville

39	Africa/Lome
40	Africa/Luanda
41	Africa/Lubumbashi
42	Africa/Lusaka
43	Africa/Malabo
44	Africa/Maputo
45	Africa/Maseru
46	Africa/Mbabane
47	Africa/Mogadishu
48	Africa/Monrovia
49	Africa/Nairobi
50	Africa/Ndjamena
51	Africa/Niamey
52	Africa/Nouakchott
53	Africa/Ouagadougou
54	Africa/Porto-Novo
55	Africa/Sao_Tome
56	Africa/Timbuktu
57	Africa/Tripoli
58	Africa/Tunis
59	Africa/Windhoek
60	America/Adak
61	America/Anchorage
62	America/Anguilla
63	America/Antigua
64	America/Araguaina
65	America/Argentina/Buenos_Aires
66	America/Argentina/Catamarca
67	America/Argentina/ComodRivadavia
68	America/Argentina/Cordoba
69	America/Argentina/Jujuy
70	America/Argentina/La_Rioja
71	America/Argentina/Mendoza
72	America/Argentina/Rio_Gallegos
73	America/Argentina/Salta
74	America/Argentina/San_Juan
75	America/Argentina/San_Luis
76	America/Argentina/Tucuman
77	America/Argentina/Ushuaia
78	America/Aruba
79	America/Asuncion

80	America/Atikokan
81	America/Atka
82	America/Bahia
83	America/Bahia_Banderas
84	America/Barbados
85	America/Belem
86	America/Belize
87	America/Blanc-Sablon
88	America/Boa_Vista
89	America/Bogota
90	America/Boise
91	America/Buenos_Aires
92	America/Cambridge_Bay
93	America/Campo_Grande
94	America/Cancun
95	America/Caracas
96	America/Catamarca
97	America/Cayenne
98	America/Cayman
99	America/Chicago
100	America/Chihuahua
101	America/Coral_Harbour
102	America/Cordoba
103	America/Costa_Rica
104	America/Creston
105	America/Cuiaba
106	America/Curacao
107	America/Danmarkshavn
108	America/Dawson
109	America/Dawson_Creek
110	America/Denver
111	America/Detroit
112	America/Dominica
113	America/Edmonton
114	America/Eirunepe
115	America/El_Salvador
116	America/Ensenada
117	America/Fort_Nelson
118	America/Fort_Wayne
119	America/Fortaleza
120	America/Glace_Bay

121	America/Godthab
122	America/Goose_Bay
123	America/Grand_Turk
124	America/Grenada
125	America/Guadeloupe
126	America/Guatemala
127	America/Guayaquil
128	America/Guyana
129	America/Halifax
130	America/Havana
131	America/Hermosillo
132	America/Indiana/Indianapolis
133	America/Indiana/Knox
134	America/Indiana/Marengo
135	America/Indiana/Petersburg
136	America/Indiana/Tell_City
137	America/Indiana/Vevay
138	America/Indiana/Vincennes
139	America/Indiana/Winamac
140	America/Indianapolis
141	America/Inuvik
142	America/Iqaluit
143	America/Jamaica
144	America/Jujuy
145	America/Juneau
146	America/Kentucky/Louisville
147	America/Kentucky/Monticello
148	America/Knox_IN
149	America/Kralendijk
150	America/La_Paz
151	America/Lima
152	America/Los_Angeles
153	America/Louisville
154	America/Lower_Princes
155	America/Maceio
156	America/Managua
157	America/Manaus
158	America/Marigot
159	America/Martinique
160	America/Matamoros
161	America/Mazatlan

162	America/Mendoza
163	America/Menominee
164	America/Merida
165	America/Metlakatla
166	America/Mexico_City
167	America/Miquelon
168	America/Moncton
169	America/Monterrey
170	America/Montevideo
171	America/Montreal
172	America/Montserrat
173	America/Nassau
174	America/New_York
175	America/Nipigon
176	America/Nome
177	America/Noronha
178	America/North_Dakota/Beulah
179	America/North_Dakota/Center
180	America/North_Dakota/New_Salem
181	America/Ojinaga
182	America/Panama
183	America/Pangnirtung
184	America/Paramaribo
185	America/Phoenix
186	America/Port-au-Prince
187	America/Port_of_Spain
188	America/Porto_Acre
189	America/Porto_Velho
190	America/Puerto_Rico
191	America/Punta_Arenas
192	America/Rainy_River
193	America/Rankin_Inlet
194	America/Recife
195	America/Regina
196	America/Resolute
197	America/Rio_Branco
198	America/Rosario
199	America/Santa_Isabel
200	America/Santarem
201	America/Santiago
202	America/Santo_Domingo

203	America/Sao_Paulo
204	America/Scoresbysund
205	America/Shiprock
206	America/Sitka
207	America/St_Barthelemy
208	America/St_Johns
209	America/St_Kitts
210	America/St_Lucia
211	America/St_Thomas
212	America/St_Vincent
213	America/Swift_Current
214	America/Tegucigalpa
215	America/Thule
216	America/Thunder_Bay
217	America/Tijuana
218	America/Toronto
219	America/Tortola
220	America/Vancouver
221	America/Virgin
222	America/Whitehorse
223	America/Winnipeg
224	America/Yakutat
225	America/Yellowknife
226	Antarctica/Casey
227	Antarctica/Davis
228	Antarctica/DumontDUrville
229	Antarctica/Macquarie
230	Antarctica/Mawson
231	Antarctica/McMurdo
232	Antarctica/Palmer
233	Antarctica/Rothera
234	Antarctica/South_Pole
235	Antarctica/Syowa
236	Antarctica/Troll
237	Antarctica/Vostok
238	Arctic/Longyearbyen
239	Asia/Aden
240	Asia/Almaty
241	Asia/Amman
242	Asia/Anadyr
243	Asia/Aqtau

244	Asia/Aqtobe
245	Asia/Ashgabat
246	Asia/Ashkhabad
247	Asia/Atyrau
248	Asia/Baghdad
249	Asia/Bahrain
250	Asia/Baku
251	Asia/Bangkok
252	Asia/Barnaul
253	Asia/Beirut
254	Asia/Bishkek
255	Asia/Brunei
256	Asia/Calcutta
257	Asia/Chita
258	Asia/Choibalsan
259	Asia/Chongqing
260	Asia/Chungking
261	Asia/Colombo
262	Asia/Dacca
263	Asia/Damascus
264	Asia/Dhaka
265	Asia/Dili
266	Asia/Dubai
267	Asia/Dushanbe
268	Asia/Famagusta
269	Asia/Gaza
270	Asia/Harbin
271	Asia/Hebron
272	Asia/Ho_Chi_Minh
273	Asia/Hong_Kong
274	Asia/Hovd
275	Asia/Irkutsk
276	Asia/Istanbul
277	Asia/Jakarta
278	Asia/Jayapura
279	Asia/Jerusalem
280	Asia/Kabul
281	Asia/Kamchatka
282	Asia/Karachi
283	Asia/Kashgar
284	Asia/Kathmandu

285	Asia/Katmandu
286	Asia/Khandyga
287	Asia/Kolkata
288	Asia/Krasnoyarsk
289	Asia/Kuala_Lumpur
290	Asia/Kuching
291	Asia/Kuwait
292	Asia/Macao
293	Asia/Macau
294	Asia/Magadan
295	Asia/Makassar
296	Asia/Manila
297	Asia/Muscat
298	Asia/Nicosia
299	Asia/Novokuznetsk
300	Asia/Novosibirsk
301	Asia/Omsk
302	Asia/Oral
303	Asia/Phnom_Penh
304	Asia/Pontianak
305	Asia/Pyongyang
306	Asia/Qatar
307	Asia/Qyzylorda
308	Asia/Rangoon
309	Asia/Riyadh
310	Asia/Saigon
311	Asia/Sakhalin
312	Asia/Samarkand
313	Asia/Seoul
314	Asia/Shanghai
315	Asia/Singapore
316	Asia/Srednekolymsk
317	Asia/Taipei
318	Asia/Tashkent
319	Asia/Tbilisi
320	Asia/Tehran
321	Asia/Tel_Aviv
322	Asia/Thimbu
323	Asia/Thimphu
324	Asia/Tokyo
325	Asia/Tomsk

326	Asia/Ujung_Pandang
327	Asia/Ulaanbaatar
328	Asia/Ulan_Bator
329	Asia/Urumqi
330	Asia/Ust-Nera
331	Asia/Vientiane
332	Asia/Vladivostok
333	Asia/Yakutsk
334	Asia/Yangon
335	Asia/Yekaterinburg
336	Asia/Yerevan
337	Atlantic/Azores
338	Atlantic/Bermuda
339	Atlantic/Canary
340	Atlantic/Cape_Verde
341	Atlantic/Faeroe
342	Atlantic/Faroe
343	Atlantic/Jan_Mayen
344	Atlantic/Madeira
345	Atlantic/Reykjavik
346	Atlantic/South_Georgia
347	Atlantic/St_Helena
348	Atlantic/Stanley
349	Australia/ACT
350	Australia/Adelaide
351	Australia/Brisbane
352	Australia/Broken_Hill
353	Australia/Canberra
354	Australia/Currie
355	Australia/Darwin
356	Australia/Eucla
357	Australia/Hobart
358	Australia/LHI
359	Australia/Lindeman
360	Australia/Lord_Howe
361	Australia/Melbourne
362	Australia/NSW
363	Australia/North
364	Australia/Perth
365	Australia/Queensland
366	Australia/South

367	Australia/Sydney
368	Australia/Tasmania
369	Australia/Victoria
370	Australia/West
371	Australia/Yancowinna
372	BET
373	BST
374	Brazil/Acre
375	Brazil/DeNoronha
376	Brazil/East
377	Brazil/West
378	CAT
379	CET
380	CNT
381	CST
382	CST6CDT
383	CTT
384	Canada/Atlantic
385	Canada/Central
386	Canada/Eastern
387	Canada/Mountain
388	Canada/Newfoundland
389	Canada/Pacific
390	Canada/Saskatchewan
391	Canada/Yukon
392	Chile/Continental
393	Chile/EasterIsland
394	Cuba
395	EAT
396	ECT
397	EET
398	EST
399	EST5EDT
400	Egypt
401	Eire
402	Etc/GMT
403	Etc/GMT+0
404	Etc/GMT+1
405	Etc/GMT+10
406	Etc/GMT+11
407	Etc/GMT+12

408	Etc/GMT+2
409	Etc/GMT+3
410	Etc/GMT+4
411	Etc/GMT+5
412	Etc/GMT+6
413	Etc/GMT+7
414	Etc/GMT+8
415	Etc/GMT+9
416	Etc/GMT-0
417	Etc/GMT-1
418	Etc/GMT-10
419	Etc/GMT-11
420	Etc/GMT-12
421	Etc/GMT-13
422	Etc/GMT-14
423	Etc/GMT-2
424	Etc/GMT-3
425	Etc/GMT-4
426	Etc/GMT-5
427	Etc/GMT-6
428	Etc/GMT-7
429	Etc/GMT-8
430	Etc/GMT-9
431	Etc/GMT0
432	Etc/Greenwich
433	Etc/UCT
434	Etc/UTC
435	Etc/Universal
436	Etc/Zulu
437	Europe/Amsterdam
438	Europe/Andorra
439	Europe/Astrakhan
440	Europe/Athens
441	Europe/Belfast
442	Europe/Belgrade
443	Europe/Berlin
444	Europe/Bratislava
445	Europe/Brussels
446	Europe/Bucharest
447	Europe/Budapest
448	Europe/Busingen

449	Europe/Chisinau
450	Europe/Copenhagen
451	Europe/Dublin
452	Europe/Gibraltar
453	Europe/Guernsey
454	Europe/Helsinki
455	Europe/Isle_of_Man
456	Europe/Istanbul
457	Europe/Jersey
458	Europe/Kaliningrad
459	Europe/Kiev
460	Europe/Kirov
461	Europe/Lisbon
462	Europe/Ljubljana
463	Europe/London
464	Europe/Luxembourg
465	Europe/Madrid
466	Europe/Malta
467	Europe/Mariehamn
468	Europe/Minsk
469	Europe/Monaco
470	Europe/Moscow
471	Europe/Nicosia
472	Europe/Oslo
473	Europe/Paris
474	Europe/Podgorica
475	Europe/Prague
476	Europe/Riga
477	Europe/Rome
478	Europe/Samara
479	Europe/San_Marino
480	Europe/Sarajevo
481	Europe/Saratov
482	Europe/Simferopol
483	Europe/Skopje
484	Europe/Sofia
485	Europe/Stockholm
486	Europe/Tallinn
487	Europe/Tirane
488	Europe/Tiraspol
489	Europe/Ulyanovsk

490	Europe/Uzhgorod
491	Europe/Vaduz
492	Europe/Vatican
493	Europe/Vienna
494	Europe/Vilnius
495	Europe/Volgograd
496	Europe/Warsaw
497	Europe/Zagreb
498	Europe/Zaporozhye
499	Europe/Zurich
500	GB
501	GB-Eire
502	GMT
503	GMT0
504	Greenwich
505	HST
506	Hongkong
507	IET
508	IST
509	Iceland
510	Indian/Antananarivo
511	Indian/Chagos
512	Indian/Christmas
513	Indian/Cocos
514	Indian/Comoro
515	Indian/Kerguelen
516	Indian/Mahe
517	Indian/Maldives
518	Indian/Mauritius
519	Indian/Mayotte
520	Indian/Reunion
521	Iran
522	Israel
523	JST
524	Jamaica
525	Japan
526	Kwajalein
527	Libya
528	MET
529	MIT
530	MST

531	MST7MDT
532	Mexico/BajaNorte
533	Mexico/BajaSur
534	Mexico/General
535	NET
536	NST
537	NZ
538	NZ-CHAT
539	Navajo
540	PLT
541	PNT
542	PRC
543	PRT
544	PST
545	PST8PDT
546	Pacific/Apia
547	Pacific/Auckland
548	Pacific/Bougainville
549	Pacific/Chatham
550	Pacific/Chuuk
551	Pacific/Easter
552	Pacific/Efate
553	Pacific/Enderbury
554	Pacific/Fakaofo
555	Pacific/Fiji
556	Pacific/Funafuti
557	Pacific/Galapagos
558	Pacific/Gambier
559	Pacific/Guadalcanal
560	Pacific/Guam
561	Pacific/Honolulu
562	Pacific/Johnston
563	Pacific/Kiritimati
564	Pacific/Kosrae
565	Pacific/Kwajalein
566	Pacific/Majuro
567	Pacific/Marquesas
568	Pacific/Midway
569	Pacific/Nauru
570	Pacific/Niue
571	Pacific/Norfolk

572	Pacific/Noumea
573	Pacific/Pago_Pago
574	Pacific/Palau
575	Pacific/Pitcairn
576	Pacific/Pohnpei
577	Pacific/Ponape
578	Pacific/Port_Moresby
579	Pacific/Rarotonga
580	Pacific/Saipan
581	Pacific/Samoa
582	Pacific/Tahiti
583	Pacific/Tarawa
584	Pacific/Tongatapu
585	Pacific/Truk
586	Pacific/Wake
587	Pacific/Wallis
588	Pacific/Yap
589	Poland
590	Portugal
591	ROK
592	SST
593	Singapore
594	SystemV/AST4
595	SystemV/AST4ADT
596	SystemV/CST6
597	SystemV/CST6CDT
598	SystemV/EST5
599	SystemV/EST5EDT
600	SystemV/HST10
601	SystemV/MST7
602	SystemV/MST7MDT
603	SystemV/PST8
604	SystemV/PST8PDT
605	SystemV/YST9
606	SystemV/YST9YDT
607	Turkey
608	UCT
609	US/Alaska
610	US/Aleutian
611	US/Arizona
612	US/Central

613	US/East-Indiana
614	US/Eastern
615	US/Hawaii
616	US/Indiana-Starke
617	US/Michigan
618	US/Mountain
619	US/Pacific
620	US/Pacific-New
621	US/Samoa
622	UTC
623	Universal
624	VST
625	W-SU
626	WET
627	Zulu

Policy API

This page describes the Policy API methods you can use to create, configure, and manage policies for an application.

A policy consists of a *trigger* based on one or more events and an *action* in response to that trigger. You use policies to automate monitoring, alerting, and problem remediation.

- Syntax validation of the JSON payload is done when creating the policy.
- Objects/Entities corresponding to EUM applications are not supported.

Create a Policy

Creates a new policy using the given JSON payload.

Resource URL

```
POST <controller_url>/controller/alerting/rest
/v1/applications/<application_id>/policies
```

Request/Response Format

JSON

Example

Retrieve a list of Policies associated with an Application

This API returns a list of all policies associated with the given application, policy names, IDs, and enable flag details. To fetch the complete details of a policy, use `GET /policies/{policy-id}`.

Resource URL

```
GET <controller_url>/controller/alerting/rest/v1/applications/<application_id>/policies
```

Response Format

JSON

Example Response

```
[
    {
        "id": 1,
        "name": "Policy json update example",
        "enabled": true
    },
    {
        "id": 2,
        "name": "Policy json create example",
        "enabled": true
    }
]
```

Retrieve Details of a Specified Policy

Returns a `JSON` representation of a policy for the given policy ID.

Resource URL

```
GET <controller_url>/controller/alerting/rest/v1/applications/<application_id>/policies/{policy-id}
```

Response Format

JSON

Example Response

This example retrieves the configuration details of a policy.


```
{
    "id": 1,
    "name": "Policy REST get json example",
    "enabled": true,
    "executeActionsInBatch": true,
    "actions": [
        {
            "actionName": "your@email.com",
            "actionType": "EMAIL",
            "notes": "example notes"
        },
        {
            "actionName": "1234567890",
            "actionType": "SMS"
        },
        {
            "actionName": "Thread dump action",
            "actionType": "THREAD_DUMP",
            "specifiedEntityActionDetails": {
                "specifiedEntityActionScope": "SPECIFIC_NODES",
                "nodes": [
                    "node1"
                ]
            }
        }
    ],
    "events": {
        "healthRuleEvents": {
            "healthRuleEventTypes": [
                "HEALTH_RULE_OPEN_CRITICAL",
                "HEALTH_RULE_UPGRADED",
                "HEALTH_RULE_CONTINUES_CRITICAL",
                "HEALTH_RULE_CONTINUES_WARNING"
            ],
            "healthRuleScope": {
                "healthRuleScopeType": "SPECIFIC_HEALTH_RULES",
                "healthRules": [
                    "JVM Garbage Collection Time is too high",
                    "Memory utilization is too high"
                ]
            }
        },
        "otherEvents": [
            "VERY_SLOW",
            "STALL",
            "SLOW",
            "ERROR"
        ],
        "anomalyEvents": null,
        "customEvents": []
    },
    "selectedEntities": {
        "selectedEntityType": "SPECIFIC_ENTITIES",
        "entities": [
            {
                "entityType": "BUSINESS_TRANSACTION",
                "selectedBusinessTransactions": {
                    "businessTransactionScope": "SPECIFIC_BUSINESS_TRANSACTIONS",
                    "businessTransactions": [
                        "/home/auctions"
                    ]
                }
            },
```

```
            {
                "entityType": "TIER_NODE",
                "tierOrNode": {
                    "tierOrNodeScope": "NODE_SELECTED_ENTITIES",
                    "typeofNode": "ALL_NODES",
                    "selectedNodes": {
                        "selectedNodeScope": "SPECIFIC_NODES",
                        "nodes": [
                            "node1"
                        ]
                    }
                }
            }
        ]
    }
}
```

Update a Policy

Updates the configuration settings of an existing policy with the values of a specified policy ID.

This request requires a complete JSON payload as input. AppDynamics recommends

1. Retrieving the JSON payload using `GET /policies/{policy-id}` and update the required fields.
2. Send the modified payload as part of the `PUT` request.

Resource URL

```
PUT <controller_url>/controller/alerting/rest/v1/applications/<application_id>/policies/{policy-id}
```

Request/Response Format

JSON

Example

Delete a Policy

Deletes an existing policy with the specified ID.

Ensure that you provide a valid existing policy ID.

Resource URL

```
DELETE <controller_url>/controller/alerting/rest/v1/applications/<application_id>/policies/{policy-id}
```

Update a Policy Configuration

Updates one or more specific configuration setting(s) of a policy. You can update the **Name** and **Enabled** fields using this API.

Resource URL

```
PUT <controller_url>/controller/alerting/rest/v1/applications/<application_id>/policies/{policy-id}/configuration
```

Request/Response Format

JSON

Example

Example Response

Response Codes

Code	Description
200	Fetched successfully
201	Created successfully
204	Deleted successfully
400	Bad request
401	Unauthorized
403	Forbidden
404	Resource not found
409	Already exists

Property Details

Policy

Payload details of a policy.

Property Name	Type	Description and Valid Values
`id`	integer	This is auto-generated by the system and returned in the response. It is a `readOnly` value.
`name`*	string	Name of the policy. Minimum length: 1
`enabled`	boolean	Sets the policy to enabled/disabled state. Default value: `true`
`executeActionsInBatch`*	boolean	Executes the actions configured for a policy, once for all the triggering events that occurred during the last minute. Default value: `true`

<table>
<tr>
<td>actions*</td>
<td>minItems : 1</td>
<td>
Action
Describes the predefined, reusable, and automated response to an event to be taken when the policy is triggered.
<table>
<tr><th>Property Name</th><th>Type</th><th>Description</th></tr>
<tr><td>actionName*</td><td>string</td><td>Name of the action defined for a policy.</td></tr>
<tr><td>actionType*</td><td>string</td><td>ActionType Enum
Creates the following types of actions:
SMS
EMAIL
CUSTOM_EMAIL
THREAD_DUMP
HTTP_REQUEST
CUSTOM
RUN_SCRIPT_ON_NODES
DIAGNOSTIC_BUSINESS_TRANSACTIONS
CREATE_UPDATE_JIRA</td></tr>
</table>
</td>
</tr>
<tr>
<td>events*</td>
<td></td>
<td>
Events
Describes the events that trigger the policy.
<table>
<tr><th>Property Name</th><th>Description</th></tr>
</table>
</td>
</tr>
</table>

`healthRuleEvents`	**HealthRuleEvents**
	
	Describes the trigger event type(s) generated due to health rule issues.
	
	Property Name / **Description**
	
	`healthRuleEventTypes`* — HealthRuleEventTypes
	
	Describes the event type related to health rule that triggers the policy.
	
	minItems: 1
	
	HealthRuleEventTypeEnum
	
	`HEALTH_RULE_CONTINUES_CRITICAL`
	
	`HEALTH_RULE_OPEN_CRITICAL`
	
	`HEALTH_RULE_OPEN_WARNING`
	
	`HEALTH_RULE_UPGRADED`
	
	`HEALTH_RULE_DOWNGRADED`
	
	`HEALTH_RULE_CONTINUES_WARNING`
	
	`HEALTH_RULE_CLOSE_WARNING`
	
	`HEALTH_RULE_CLOSE_CRITICAL`
	
	`HEALTH_RULE_CANCELED_WARNING`
	
	`HEALTH_RULE_CANCELED_CRITICAL`
	
	`healthRuleScope`* — **HealthRuleScope**
	
	Describes the scope of the health rule based on which, the events are triggered.
	
	Enums
	
	`ALL_HEALTH_RULES`
	
	`SPECIFIC_HEALTH_RULES`

otherEvents	**OtherEvents** Describes the trigger event type(s) generated due to other issues.

Property Name	Description
OtherEventType	Lists the event type that triggers an action. Enums CLR_CRASH APPLICATION_CRASH DEADLOCK RESOURCE_POOL_LIMIT APPLICATION_DEPLOYMENT APP_SERVER_RESTART APPLICATION_CONFIG_CHANGE AGENT_CONFIGURATION_ERROR APPLICATION_DISCOVERED TIER_DISCOVERED NODE_DISCOVERED MACHINE_DISCOVERED BT_DISCOVERED SERVICE_ENDPOINT_DISCOVERED BACKEND_DISCOVERED EUM_CLOUD_SYNTHETIC_HEALTHY_EVENT EUM_CLOUD_SYNTHETIC_WARNING_EVENT EUM_CLOUD_SYNTHETIC_CONFIRMED_WARNING_EVENT EUM_CLOUD_SYNTHETIC_ONGOING_WARNING_EVENT EUM_CLOUD_SYNTHETIC_ERROR_EVENT EUM_CLOUD_SYNTHETIC_CONFIRMED_ERROR_EVENT EUM_CLOUD_SYNTHETIC_ONGOING_ERROR_EVENT EUM_CLOUD_SYNTHETIC_PERF_HEALTHY_EVENT EUM_CLOUD_SYNTHETIC_PERF_WARNING_EVENT EUM_CLOUD_SYNTHETIC_PERF_CONFIRMED_WARNING_EVENT EUM_CLOUD_SYNTHETIC_PERF_ONGOING_WARNING_EVENT EUM_CLOUD_SYNTHETIC_PERF_CRITICAL_EVENT EUM_CLOUD_SYNTHETIC_PERF_CONFIRMED_CRITICAL_EVENT EUM_CLOUD_SYNTHETIC_PERF_ONGOING_CRITICAL_EVENT MOBILE_NEW_CRASH_EVENT, SLOW, VERY_SLOW, STALL ERROR

<table>
<tr><td></td><td></td><td>
<table>
<tr><td>anomalyEvents</td><td>
AnomalyEvents
Describes the trigger event type(s) generated due to anomaly detection.
minItems: 1
<table>
<tr><th>Property Name</th><th>Description</th></tr>
<tr><td>AnomalyEventType</td><td>Lists the event type that triggers an action.
Enums
ANOMALY_OPEN_WARNING
ANOMALY_OPEN_CRITICAL
ANOMALY_UPGRADED
ANOMALY_DOWNGRADED
ANOMALY_CLOSE_WARNING
ANOMALY_CLOSE_CRITICAL
ANOMALY_CANCELED_WARNING
ANOMALY_CANCELED_CRITICAL</td></tr>
</table>
</td></tr>
<tr><td>customEvents</td><td>
Describes the custom event type(s) generated.
<table>
<tr><th>Property Name</th><th>Type</th><th>Description</th></tr>
<tr><td>eventName*</td><td>string</td><td>Custom event name.</td></tr>
<tr><td>propertyMatchCriteria</td><td>string</td><td>Enums
ANY
ALL</td></tr>
<tr><td>keyValuePairArray</td><td>string</td><td>KeyValuePair
key*
value*</td></tr>
</table>
</td></tr>
</table>
</td></tr>
<tr><td>selectedEntities</td><td></td><td>
SelectedEntityType
Scope of entities considered for the policy evaluation.
Entities corresponding to EUM applications are not supported.
<table>
<tr><th>Property Name</th><th>Type</th><th>Description</th></tr>
<tr><td>SelectedEntityType*</td><td>string</td><td>Enums
ANY_ENTITY
SPECIFIC_ENTITIES</td></tr>
</table>
</td></tr>
</table>

SelectedEntityType

Scope of entities considered for the policy evaluation.

Entities corresponding to EUM applications are not supported.

Property Name	Type	Description

`SelectedEntityType`*	string	Enums
		`ANY_ENTITY`
		`SPECIFIC_ENTITIES`

SpecificEntities

Scope of specific entities considered for the policy evaluation.

Property Name	Type	Description

`selectedEntityType`*	string	Enums
		`ANY_ENTITY`
		`SPECIFIC_ENTITIES`

`entities`	minItems: 1	**Entity** **Property name** / **Type** / **Description** `entityType`* — string, minItems: 1 — Enums: `BUSINESS_TRANSACTION`, `TIER_NODE`, `ERRORS`, `SERVICE_ENDPOINTS`, `INFORMATION_POINTS`, `DATABASES_IN_APPLICATION`, `SERVERS_IN_APPLICATION`

Entity

Property name	Type	Description

`entityType`*	string
	minItems: 1
		Enums
		`BUSINESS_TRANSACTION`
		`TIER_NODE`
		`ERRORS`
		`SERVICE_ENDPOINTS`
		`INFORMATION_POINTS`
		`DATABASES_IN_APPLICATION`
		`SERVERS_IN_APPLICATION`

AnyEntity

Scope of entities considered for the policy evaluation.

> Entities corresponding to EUM applications are not supported.

Property Name	Type	Description

SelectedEntityType*	string	Enums ANY_ENTITY SPECIFIC_ENTITIES

BusinessTransaction

All entities of type BUSINESS_TRANSACTION are considered for policy evaluation.

Property Name	Type	Description
entityType*	string minItems: 1	Enums BUSINESS_TRANSACTION TIER_NODE ERRORS SERVICE_ENDPOINTS INFORMATION_POINTS DATABASES_IN_APPLICATION SERVERS_IN_APPLICATION
selectedBusinessTransactions*	string	**BusinessTransactionScope** Property Name: businessTransactionScope* — Description: Enums ALL_BUSINESS_TRANSACTIONS SPECIFIC_BUSINESS_TRANSACTIONS BUSINESS_TRANSACTIONS_IN_SPECIFIC_TIERS BUSINESS_TRANSACTIONS_MATCHING_PATTERN

SelectedBusinessTransactions

Property Name	Description
businessTransactionScope*	Enums ALL_BUSINESS_TRANSACTIONS SPECIFIC_BUSINESS_TRANSACTIONS BUSINESS_TRANSACTIONS_IN_SPECIFIC_TIERS BUSINESS_TRANSACTIONS_MATCHING_PATTERN

AllBusinessTransactions

The scope of business transactions is set to all business transactions.

Property Name	Description
businessTransactionScope*	Enums ALL_BUSINESS_TRANSACTIONS SPECIFIC_BUSINESS_TRANSACTIONS BUSINESS_TRANSACTIONS_IN_SPECIFIC_TIERS BUSINESS_TRANSACTIONS_MATCHING_PATTERN

SpecificBusinessTransactions

The scope of business transactions is set to select business transactions.

Property Name	Type	Description
businessTransactionScope*		Enums ALL_BUSINESS_TRANSACTIONS SPECIFIC_BUSINESS_TRANSACTIONS BUSINESS_TRANSACTIONS_IN_SPECIFIC_TIERS BUSINESS_TRANSACTIONS_MATCHING_PATTERN
businessTransactions*	string minItems: 1	Name(s) of the business transactions.

BusinessTransactionsInSpecificTiers

The scope of business transactions is set to business transactions associated with a specific tier.

Property Name	Type	Description
businessTransactionScope*		Enums ALL_BUSINESS_TRANSACTIONS SPECIFIC_BUSINESS_TRANSACTIONS BUSINESS_TRANSACTIONS_IN_SPECIFIC_TIERS BUSINESS_TRANSACTIONS_MATCHING_PATTERN
specificTiers*	string minItems: 1	Name of the specified tier.

BusinessTransactionsMatchingPattern

The scope of business transactions is set to business transactions that match a specific pattern.

Property Name	Description
businessTransactionScope*	Enums ALL_BUSINESS_TRANSACTIONS SPECIFIC_BUSINESS_TRANSACTIONS BUSINESS_TRANSACTIONS_IN_SPECIFIC_TIERS BUSINESS_TRANSACTIONS_MATCHING_PATTERN

`patternMatcher`*	**EntityMatchingPattern**
	Business transactions that match the specified pattern are included in the scope.
	Property Name / **Type** / **Description**
	`matchTo`* — Enums: `STARTS_WITH`, `ENDS_WITH`, `CONTAINS`, `EQUALS`, `MATCH_REG_EX`
	`matchValue`* — string, minLength: 1
	`shouldNot` — boolean, default: `false`

SelectedTierOrNodeEntities

Specific tiers or nodes are considered for policy evaluation.

Property Name	Type	Description
`entityType`*	string	Enums `BUSINESS_TRANSACTION` TIER_NODE ERRORS SERVICE_ENDPOINTS INFORMATION_POINTS DATABASES_IN_APPLICATION SERVERS_IN_APPLICATION
`tierOrNode`*	string minLength: 1	**TierOrNode** **Property Name** / **Type** / **Description** `tierOrNodeScope`* — string — **Enums**: `TIER_SELECTED_ENTITIES`, `NODE_SELECTED_ENTITIES`

TierOrNode

Property Name	Type	Description

`tierOrNodeScope`*	string	**Enums**
		`TIER_SELECTED_ENTITIES`
		`NODE_SELECTED_ENTITIES`

TierSelectedEntities

Property Name	Type	Description

`tierOrNodeScope`*	string	**Enums**
		`TIER_SELECTED_ENTITIES`
		`NODE_SELECTED_ENTITIES`

`selectedTiers`*		**SelectedTiers** **Property Name** / **Type** / **Description** `selectedTierScope`* / string / Enums: `ALL_TIERS`, `SPECIFIC_TIERS`

SelectedTiers

Property Name	Type	Description

`selectedTierScope`*	string	Enums
		`ALL_TIERS`
		`SPECIFIC_TIERS`

AllTiers

Property Name	Type	Description

`selectedTierScope`*	string	Enums
		`ALL_TIERS`
		`SPECIFIC_TIERS`

SpecificTiers

Property Name	Type	Description

`selectedTierScope`*	string	Enums
		`ALL_TIERS`
		`SPECIFIC_TIERS`

`tiers`*	string minItems: 1	Name(s) of the specified tier(s).

NodeSelectedEntities

Property Name	Type	Description

`tierOrNodeScope`*	string	Enums
		`TIER_SELECTED_ENTITIES`
		`NODE_SELECTED_ENTITIES`

typeofNode*	string	Enums ALL_NODES JAVA_NODES DOT_NET_NODES PHP_NODES
selectedNodes*		**SelectedNodes** **Property Name** / **Type** / **Description** selectedNodeScope* — string — Enums: ALL_NODES, SPECIFIC_NODES, NODES_OF_SPECIFIC_TIERS, NODES_MATCHING_PATTERN, NODE_PROPERTY_VARIABLE_MATCHER

SelectedNodes

Property Name	Type	Description
selectedNodeScope*	string	Enums ALL_NODES SPECIFIC_NODES NODES_OF_SPECIFIC_TIERS NODES_MATCHING_PATTERN NODE_PROPERTY_VARIABLE_MATCHER

AllNodes

Property Name	Type	Description
selectedNodeScope*	string	Enums ALL_NODES SPECIFIC_NODES NODES_OF_SPECIFIC_TIERS NODES_MATCHING_PATTERN NODE_PROPERTY_VARIABLE_MATCHER

SpecificNodes

Property Name	Type	Description

`selectedNodeScope`*	string	Enums
		`ALL_NODES`
		`SPECIFIC_NODES`
		`NODES_OF_SPECIFIC_TIERS`
		`NODES_MATCHING_PATTERN`
		`NODE_PROPERTY_VARIABLE_MATCHER`

`nodes`*	string minItems: 1	Name(s) of the specified node(s).

NodesOfSpecificTiers

Property Name	Type	Description

`selectedNodeScope`*	string	Enums
		`ALL_NODES`
		`SPECIFIC_NODES`
		`NODES_OF_SPECIFIC_TIERS`
		`NODES_MATCHING_PATTERN`
		`NODE_PROPERTY_VARIABLE_MATCHER`

`specificTiers`*	string minItems: 1	Name(s) of tier with the associated nodes.

NodesMatchingPattern

Property Name	Type	Description

`selectedNodeScope`*	string	Enums
		`ALL_NODES`
		`SPECIFIC_NODES`
		`NODES_OF_SPECIFIC_TIERS`
		`NODES_MATCHING_PATTERN`
		`NODE_PROPERTY_VARIABLE_MATCHER`

`patternMatcher`*	string
	minItems: 1
		EntityMatchingPattern
		Nodes that match a specified pattern are included in the scope.
		Property Name / **Type** / **Description**
		`matchTo`* — Enums: `STARTS_WITH`, `ENDS_WITH`, `CONTAINS`, `EQUALS`, `MATCH_REG_EX`
		`matchValue`* — string, minLength: 1
		`shouldNot` — boolean, default: `false`

NodePropertyVariableMatcher

Property Name	Type	Description

`selectedNodeScope`*	string	Enums
		`ALL_NODES`
		`SPECIFIC_NODES`
		`NODES_OF_SPECIFIC_TIERS`
		`NODES_MATCHING_PATTERN`
		`NODE_PROPERTY_VARIABLE_MATCHER`

`propVarPairs`*	minItems: 1	**propVarPairs** **Property Name** / **Type** / **Description** `propertyType`* — string — NodePropertyTypeEnum: `META`, `ENV`, `JVM` `name`* — string, minLength: 1 `value`* — string, minLength: 1

Errors

Specific errors are considered for policy evaluation.

Property Name	Type	Description

entityType*	string	Enums BUSINESS_TRANSACTION TIER_NODE ERRORS SERVICE_ENDPOINTS INFORMATION_POINTS DATABASES_IN_APPLICATION SERVERS_IN_APPLICATION
selectedErrors*		**SelectedErrors** **Property Name** / **Type** / **Description** errorScope* — string — Enums: ALL_ERRORS, SPECIFIC_ERRORS, ERRORS_OF_SPECIFIC_TIERS, ERRORS_MATCHING_PATTERN

SelectedErrors

Property Name	Type	Description
errorScope*	string	Enums ALL_ERRORS SPECIFIC_ERRORS ERRORS_OF_SPECIFIC_TIERS ERRORS_MATCHING_PATTERN

AllErrors

Property Name	Type	Description
errorScope*	string	Enums ALL_ERRORS SPECIFIC_ERRORS ERRORS_OF_SPECIFIC_TIERS ERRORS_MATCHING_PATTERN

SpecificErrors

Property Name	Type	Description

`errorScope`*	string	Enums
		`ALL_ERRORS`
		`SPECIFIC_ERRORS`
		`ERRORS_OF_SPECIFIC_TIERS`
		`ERRORS_MATCHING_PATTERN`

`errors`*	string minItems: 1 Example: `NullPointerException`	

ErrorsOfSpecificTiers

Property Name	Type	Description

`errorScope`*	string	Enums
		`ALL_ERRORS`
		`SPECIFIC_ERRORS`
		`ERRORS_OF_SPECIFIC_TIERS`
		`ERRORS_MATCHING_PATTERN`

`specificTiers`*	string minItems: 1	

ErrorsMatchingPattern

Property Name	Type	Description

`errorScope`*	string	Enums
		`ALL_ERRORS`
		`SPECIFIC_ERRORS`
		`ERRORS_OF_SPECIFIC_TIERS`
		`ERRORS_MATCHING_PATTERN`

`patternMatcher`*		**EntityMatchingPattern**
		Property Name / **Type** / **Description**
		`matchTo`* — string — Enums: `STARTS_WITH`, `ENDS_WITH`, `CONTAINS`, `EQUALS`, `MATCH_REG_EX`
		`matchValue`* — string, minLength: 1
		`shouldNot` — boolean, default: `false`

ServiceEndpoints

Specific service endpoints are considered for policy evaluation.

Property Name	Type	Description

`entityType`*	string	Enums
		`BUSINESS_TRANSACTION`
		`TIER_NODE`
		`ERRORS`
		`SERVICE_ENDPOINTS`
		`INFORMATION_POINTS`
		`DATABASES_IN_APPLICATION`
		`SERVERS_IN_APPLICATION`

`selectedServiceEndpoints`*		**SelectedServiceEndpoints** See table below

Property Name	Type	Description

`serviceEndpointScope`*	string	Enums
		`ALL_SERVICE_ENDPOINTS`
		`SPECIFIC_SERVICE_ENDPOINTS`
		`SERVICE_ENDPOINTS_IN_SPECIFIC_TIERS`
		`SERVICE_ENDPOINTS_MATCHING_PATTERN`

SelectedServiceEndpoints

Property Name	Type	Description

`serviceEndpointScope`*	string	Enums
		`ALL_SERVICE_ENDPOINTS`
		`SPECIFIC_SERVICE_ENDPOINTS`
		`SERVICE_ENDPOINTS_IN_SPECIFIC_TIERS`
		`SERVICE_ENDPOINTS_MATCHING_PATTERN`

AllServiceEndpoints

Property Name	Type	Description

`serviceEndpointScope`*	string	Enums
		`ALL_SERVICE_ENDPOINTS`
		`SPECIFIC_SERVICE_ENDPOINTS`
		`SERVICE_ENDPOINTS_IN_SPECIFIC_TIERS`
		`SERVICE_ENDPOINTS_MATCHING_PATTERN`

SpecificServiceEndpoints

Property Name	Type	Description

`serviceEndpointScope`*	string	Enums
		`ALL_SERVICE_ENDPOINTS`
		`SPECIFIC_SERVICE_ENDPOINTS`
		`SERVICE_ENDPOINTS_IN_SPECIFIC_TIERS`
		`SERVICE_ENDPOINTS_MATCHING_PATTERN`

`serviceEndpoints`*	string minItems: 1	

ServiceEndpointsInSpecificTiers

Property Name	Type	Description
`serviceEndpointScope`*	string	Enums ALL_SERVICE_ENDPOINTS SPECIFIC_SERVICE_ENDPOINTS SERVICE_ENDPOINTS_IN_SPECIFIC_TIERS SERVICE_ENDPOINTS_MATCHING_PATTERN
`specificTiers`*	string minItems: 1	

ServiceEndpointsInSpecificTiers

Property Name	Type	Description
`serviceEndpointScope`*	string	Enums ALL_SERVICE_ENDPOINTS SPECIFIC_SERVICE_ENDPOINTS SERVICE_ENDPOINTS_IN_SPECIFIC_TIERS SERVICE_ENDPOINTS_MATCHING_PATTERN

`patternMatcher`*		**EntityMatchingPattern**
		Property Name / **Type** / **Description**
		`matchTo`* — string — Enums: `STARTS_WITH`, `ENDS_WITH`, `CONTAINS`, `EQUALS`, `MATCH_REG_EX`
		`matchValue`* — string, minLength: 1
		`shouldNot` — boolean, default: `false`

InformationPoints

Specific information points are considered for policy evaluation.

Property Name	Type	Description

`entityType`*	string	Enums
		`BUSINESS_TRANSACTION`
		`TIER_NODE`
		`ERRORS`
		`SERVICE_ENDPOINTS`
		`INFORMATION_POINTS`
		`DATABASES_IN_APPLICATION`
		`SERVERS_IN_APPLICATION`

`selectedInformationPoints`*		**SelectedInformationPoints** See the SelectedInformationPoints table below.

SelectedInformationPoints

Property Name	Type	Description
`informationPointScope`*	string	Enums ALL_INFORMATION_POINTS SPECIFIC_INFORMATION_POINTS INFORMATION_POINTS_MATCHING_PATTERN

AllInformationPoints

Property Name	Type	Description

`informationPointScope`*	string	Enums
		`ALL_INFORMATION_POINTS`
		`SPECIFIC_INFORMATION_POINTS`
		`INFORMATION_POINTS_MATCHING_PATTERN`

SpecificInformationPoints

Property Name	Type	Description

`informationPointScope`*	string	Enums
		`ALL_INFORMATION_POINTS`
		`SPECIFIC_INFORMATION_POINTS`
		`INFORMATION_POINTS_MATCHING_PATTERN`

`informationPoints`*	string minItems: 1	

InformationPointsMatchingPattern

Property Name	Type	Description

`informationPointScope`*	string	Enums
		`ALL_INFORMATION_POINTS`
		`SPECIFIC_INFORMATION_POINTS`
		`INFORMATION_POINTS_MATCHING_PATTERN`

`patternMatcher`*		**EntityMatchingPattern**
		Property Name / **Type** / **Description**
		`matchTo`* — string — Enums: `STARTS_WITH`, `ENDS_WITH`, `CONTAINS`, `EQUALS`, `MATCH_REG_EX`
		`matchValue`* — string, minLength: 1
		`shouldNot` — boolean, default: `false`

DatabasesInApplication

Specific databases associated with the application are considered for policy evaluation.

Property Name	Type	Description

`entityType`*	string	Enums
		`BUSINESS_TRANSACTION`
		`TIER_NODE`
		`ERRORS`
		`SERVICE_ENDPOINTS`
		`INFORMATION_POINTS`
		`DATABASES_IN_APPLICATION`
		`SERVERS_IN_APPLICATION`

selectedApplicationDatabases*		**SelectedApplicationDatabases** **Property Name** / **Type** / **Description** applicationDatabaseScope — string — Enums: ALL_APPLICATION_DATABASES, SPECIFIC_APPLICATION_DATABASES, APPLICATION_DATABASES_MATCHING_PATTERN

SelectedApplicationDatabases

Property Name	Type	Description
applicationDatabaseScope*	string	Enums ALL_APPLICATION_DATABASES SPECIFIC_APPLICATION_DATABASES APPLICATION_DATABASES_MATCHING_PATTERN

AllApplicationDatabases

Property Name	Type	Description
applicationDatabaseScope*	string	Enums ALL_APPLICATION_DATABASES SPECIFIC_APPLICATION_DATABASES APPLICATION_DATABASES_MATCHING_PATTERN

SpecificApplicationDatabases

Property Name	Type	Description
applicationDatabaseScope*	string	Enums ALL_APPLICATION_DATABASES SPECIFIC_APPLICATION_DATABASES APPLICATION_DATABASES_MATCHING_PATTERN
applicationDatabases*	string minItems: 1	

ApplicationDatabasesMatchingPattern

Property Name	Type	Description
applicationDatabaseScope*	string	Enums ALL_APPLICATION_DATABASES SPECIFIC_APPLICATION_DATABASES APPLICATION_DATABASES_MATCHING_PATTERN

`patternMatcher`*		**EntityMatchingPattern**
		Property Name / **Type** / **Description**
		`matchTo`* — string — Enums: `STARTS_WITH`, `ENDS_WITH`, `CONTAINS`, `EQUALS`, `MATCH_REG_EX`
		`matchValue`* — string, minLength: 1
		`shouldNot` — boolean, default: `false`

ServersInApplication

Specific servers associated with the application, are considered for the policy evaluation.

Property Name	Type	Description

`entityType`*	string	Enums
		`BUSINESS_TRANSACTION`
		`TIER_NODE`
		`ERRORS`
		`SERVICE_ENDPOINTS`
		`INFORMATION_POINTS`
		`DATABASES_IN_APPLICATION`
		`SERVERS_IN_APPLICATION`

`selectedServers`*	string	**ApplicationSelectedServers** **Property Name** / **Type** / **Description** `serversScope`* — string — Enums: `ALL_SERVERS_IN_APPLICATION`, `SPECIFIC_SERVERS_IN_APPLICATION`, `ALL_SERVERS_IN_SPECIFIC_TIERS`

ApplicationSelectedServers

Property Name	Type	Description

`serversScope`*	string	Enums
		`ALL_SERVERS_IN_APPLICATION`
		`SPECIFIC_SERVERS_IN_APPLICATION`
		`ALL_SERVERS_IN_SPECIFIC_TIERS`

AllServersInApplication

Property Name	Type	Description

`serversScope`*	string	Enums
		`ALL_SERVERS_IN_APPLICATION`
		`SPECIFIC_SERVERS_IN_APPLICATION`
		`ALL_SERVERS_IN_SPECIFIC_TIERS`

SpecificServersInApplication

Property Name	Type	Description

`serversScope`*	string	Enums
		`ALL_SERVERS_IN_APPLICATION`
		`SPECIFIC_SERVERS_IN_APPLICATION`
		`ALL_SERVERS_IN_SPECIFIC_TIERS`

`specificServers`*	string minLength: 1 minItems: 1	

AllServersInSpecificTiers

Property Name	Type	Description

`serversScope`*	string	Enums
		`ALL_SERVERS_IN_APPLICATION`
		`SPECIFIC_SERVERS_IN_APPLICATION`
		`ALL_SERVERS_IN_SPECIFIC_TIERS`

`specificTiers`*	string minItems: 1	

Events

Different types of events that trigger a policy.

Property Name	Type	Description

`healthRuleEvents`		**HealthRuleEvent**
		
		Events associated with health rules that trigger the policy.
		
		Property Name / **Type** / **Description**
		
		`healthRuleEventTypes`* — string, minItems: 1 — **HealthRuleEventType**
		Enums
		`HEALTH_RULE_CONTINUES_CRITICAL`
		`HEALTH_RULE_OPEN_CRITICAL`
		`HEALTH_RULE_OPEN_WARNING`
		`HEALTH_RULE_UPGRADED`
		`HEALTH_RULE_DOWNGRADED`
		`HEALTH_RULE_CONTINUES_WARNING`
		`HEALTH_RULE_CLOSE_WARNING`
		`HEALTH_RULE_CLOSE_CRITICAL`
		`HEALTH_RULE_CANCELED_WARNING`
		`HEALTH_RULE_CANCELED_CRITICAL`
		
		`healthRuleScope`* — string — Events associated with specific health rules or all health rules that trigger the policy.
		HealthRuleScopeType
		Property Name / **Type** / **Description**
		`healthRuleScopeType` — string — `ALL_HEALTH_RULES`, `SPECIFIC_HEALTH_RULES`

otherEvents	string	**OtherEventType** Enums CLR_CRASH APPLICATION_CRASH DEADLOCK RESOURCE_POOL_LIMIT APPLICATION_DEPLOYMENT APP_SERVER_RESTART APPLICATION_CONFIG_CHANGE AGENT_CONFIGURATION_ERROR APPLICATION_DISCOVERED TIER_DISCOVERED NODE_DISCOVERED MACHINE_DISCOVERED BT_DISCOVERED SERVICE_ENDPOINT_DISCOVERED BACKEND_DISCOVERED EUM_CLOUD_SYNTHETIC_HEALTHY_EVENT EUM_CLOUD_SYNTHETIC_WARNING_EVENT EUM_CLOUD_SYNTHETIC_CONFIRMED_WARNING_EVENT EUM_CLOUD_SYNTHETIC_ONGOING_WARNING_EVENT EUM_CLOUD_SYNTHETIC_ERROR_EVENT EUM_CLOUD_SYNTHETIC_CONFIRMED_ERROR_EVENT EUM_CLOUD_SYNTHETIC_ONGOING_ERROR_EVENT EUM_CLOUD_SYNTHETIC_PERF_HEALTHY_EVENT EUM_CLOUD_SYNTHETIC_PERF_WARNING_EVENT EUM_CLOUD_SYNTHETIC_PERF_CONFIRMED_WARNING_EVENT EUM_CLOUD_SYNTHETIC_PERF_ONGOING_WARNING_EVENT EUM_CLOUD_SYNTHETIC_PERF_CRITICAL_EVENT EUM_CLOUD_SYNTHETIC_PERF_CONFIRMED_CRITICAL_EVENT EUM_CLOUD_SYNTHETIC_PERF_ONGOING_CRITICAL_EVENT MOBILE_NEW_CRASH_EVENT SLOW VERY_SLOW STALL ERROR

<table>
<tr>
<td>anomalyEvents</td>
<td>minItems: 1</td>
<td>Events triggered due to anomaly detection.

AnomalyEventType

Enums

ANOMALY_OPEN_WARNING

ANOMALY_OPEN_CRITICAL

ANOMALY_UPGRADED

ANOMALY_DOWNGRADED

ANOMALY_CLOSE_WARNING

ANOMALY_CLOSE_CRITICAL

ANOMALY_CANCELED_WARNING

ANOMALY_CANCELED_CRITICAL</td>
</tr>
<tr>
<td>customEvents</td>
<td></td>
<td>Custom-defined events that trigger the policy.

CustomEvent

<table>
<tr><th>Property Name</th><th>Type</th><th>Description</th></tr>
<tr><td>eventName*</td><td>string</td><td></td></tr>
<tr><td>PropertyMatchCriteria</td><td>string

default: ANY</td><td>Enums

ANY

ALL</td></tr>
<tr><td>keyValuePairArray</td><td></td><td>KeyValuePair

<table>
<tr><th>Property Name</th><th>Type</th></tr>
<tr><td>Key*</td><td>string</td></tr>
<tr><td>Value*</td><td>string</td></tr>
</table>
</td></tr>
</table>
</td>
</tr>
</table>

CustomEvent

Details of custom-defined event that triggers the policy.

<table>
<tr><th>Property Name</th><th>Type</th><th>Description</th></tr>
<tr><td>eventName*</td><td>string</td><td></td></tr>
<tr><td>PropertyMatchCriteria</td><td>string

default: ANY</td><td>Enums

ANY

ALL</td></tr>
<tr><td>keyValuePairArray</td><td></td><td>KeyValuePair

<table>
<tr><th>Property Name</th><th>Type</th></tr>
<tr><td>Key*</td><td>string</td></tr>
<tr><td>Value*</td><td>string</td></tr>
</table>
</td></tr>
</table>

HealthRuleEvent

Events associated with health rules that trigger the policy.

Property Name	Type	Description

`healthRuleEventTypes`*	string
	minItems: 1
		HealthRuleEventType
		Enums
		`HEALTH_RULE_CONTINUES_CRITICAL`
		`HEALTH_RULE_OPEN_CRITICAL`
		`HEALTH_RULE_OPEN_WARNING`
		`HEALTH_RULE_UPGRADED`
		`HEALTH_RULE_DOWNGRADED`
		`HEALTH_RULE_CONTINUES_WARNING`
		`HEALTH_RULE_CLOSE_WARNING`
		`HEALTH_RULE_CLOSE_CRITICAL`
		`HEALTH_RULE_CANCELED_WARNING`
		`HEALTH_RULE_CANCELED_CRITICAL`

`healthRuleScope`*	string	Events associated with specific health rules or all health rules that trigger the policy. healthRuleScopeType **Property Name** / **Type** / **Description** `healthRuleScopeType`* — string — Enums: `ALL_HEALTH_RULES`, `SPECIFIC_HEALTH_RULES`

HealthRuleScope

Events associated with specific health rules or all health rules that trigger the policy.

Property Name	Type	Description

`healthRuleScopeType`*	string	Enums
		`ALL_HEALTH_RULES`
		`SPECIFIC_HEALTH_RULES`

AllHealthRules

Events associated with all health rules within an application, trigger the policy.

Property Name	Type	Description

`healthRuleScopeType`*	string	Enums
		`ALL_HEALTH_RULES`
		`SPECIFIC_HEALTH_RULES`

AllHealthRules

Events associated with specific health rules within an application, trigger the policy.

Property Name	Type	Description

`healthRuleScopeType`*	string	Enums
		`ALL_HEALTH_RULES`
		`SPECIFIC_HEALTH_RULES`

`healthRules`*	string minItems: 1	

HealthRuleEventTypes

Property Name	Type	Description

`healthRuleEventTypes`*	string	Enums
		`HEALTH_RULE_CONTINUES_CRITICAL`
		`HEALTH_RULE_OPEN_CRITICAL`
		`HEALTH_RULE_OPEN_WARNING`
		`HEALTH_RULE_UPGRADED`
		`HEALTH_RULE_DOWNGRADED`
		`HEALTH_RULE_CONTINUES_WARNING`
		`HEALTH_RULE_CLOSE_WARNING`
		`HEALTH_RULE_CLOSE_CRITICAL`
		`HEALTH_RULE_CANCELED_WARNING`
		`HEALTH_RULE_CANCELED_CRITICAL`

OtherEvents

Property Name	Type	Description

otherEvents	string	**OtherEventType** Enums CLR_CRASH APPLICATION_CRASH DEADLOCK RESOURCE_POOL_LIMIT APPLICATION_DEPLOYMENT APP_SERVER_RESTART APPLICATION_CONFIG_CHANGE AGENT_CONFIGURATION_ERROR APPLICATION_DISCOVERED TIER_DISCOVERED NODE_DISCOVERED MACHINE_DISCOVERED BT_DISCOVERED SERVICE_ENDPOINT_DISCOVERED BACKEND_DISCOVERED EUM_CLOUD_SYNTHETIC_HEALTHY_EVENT EUM_CLOUD_SYNTHETIC_WARNING_EVENT EUM_CLOUD_SYNTHETIC_CONFIRMED_WARNING_EVENT EUM_CLOUD_SYNTHETIC_ONGOING_WARNING_EVENT EUM_CLOUD_SYNTHETIC_ERROR_EVENT EUM_CLOUD_SYNTHETIC_CONFIRMED_ERROR_EVENT EUM_CLOUD_SYNTHETIC_ONGOING_ERROR_EVENT EUM_CLOUD_SYNTHETIC_PERF_HEALTHY_EVENT EUM_CLOUD_SYNTHETIC_PERF_WARNING_EVENT EUM_CLOUD_SYNTHETIC_PERF_CONFIRMED_WARNING_EVENT EUM_CLOUD_SYNTHETIC_PERF_ONGOING_WARNING_EVENT EUM_CLOUD_SYNTHETIC_PERF_CRITICAL_EVENT EUM_CLOUD_SYNTHETIC_PERF_CONFIRMED_CRITICAL_EVENT EUM_CLOUD_SYNTHETIC_PERF_ONGOING_CRITICAL_EVENT MOBILE_NEW_CRASH_EVENT SLOW VERY_SLOW STALL ERROR

AnomalyEvents

Events generated due to anomaly detection that trigger the policy.

Property Name	Type	Description

anomalyEvents	string minItems: 1	**AnomalyEventType** Enums ANOMALY_OPEN_WARNING ANOMALY_OPEN_CRITICAL ANOMALY_UPGRADED ANOMALY_DOWNGRADED ANOMALY_CLOSE_WARNING ANOMALY_CLOSE_CRITICAL ANOMALY_CANCELED_WARNING ANOMALY_CANCELED_CRITICAL

Action

A list of actions that are taken when a policy is triggered.

Property Name	Type	Description
actionName*	string	
actionType*	string	**ActionType** Enums SMS EMAIL CUSTOM_EMAIL THREAD_DUMP HTTP_REQUEST CUSTOM RUN_SCRIPT_ON_NODES DIAGNOSTIC_BUSINESS_TRANSACTIONS CREATE_UPDATE_JIRA

SimpleActionType

A simple action that is taken when the policy is triggered.

Property Name	Type	Description
actionName*	string	

actionType*	string	**ActionType** Enums SMS EMAIL CUSTOM_EMAIL THREAD_DUMP HTTP_REQUEST CUSTOM RUN_SCRIPT_ON_NODES DIAGNOSTIC_BUSINESS_TRANSACTIONS CREATE_UPDATE_JIRA

EmailActionType

An email is sent when the policy is triggered.

Property Name	Type	Description
actionName*	string	
actionType*	string	**ActionType** Enums SMS EMAIL CUSTOM_EMAIL THREAD_DUMP HTTP_REQUEST CUSTOM RUN_SCRIPT_ON_NODES DIAGNOSTIC_BUSINESS_TRANSACTIONS CREATE_UPDATE_JIRA
notes	string	

ActionOnSpecifiedEntities

A simple action that is taken for specific entities when the policy is triggered.

Property Name	Type	Description
actionName*	string	

actionType*	string	**ActionType** Enums SMS EMAIL CUSTOM_EMAIL THREAD_DUMP HTTP_REQUEST CUSTOM RUN_SCRIPT_ON_NODES DIAGNOSTIC_BUSINESS_TRANSACTIONS CREATE_UPDATE_JIRA
specifiedEntityActionDetails*	string	**SpecifiedEntityActionDetails** Property Name / Type / Description specifiedEntityActionScope — string — **SpecifiedEntityActionScope** Enums PERCENTAGE ABSOLUTE SPECIFIC_NODES

SpecifiedEntityActionDetails

Property Name	Type	Description
specifiedEntityActionScope	string	**SpecifiedEntityActionScope** Enums PERCENTAGE ABSOLUTE SPECIFIC_NODES

ActionOnPercentageEntities

The scope of entities on which the action is performed is set to 'percentage'.

Property Name	Type	Description
specifiedEntityActionScope	string	**SpecifiedEntityActionScope** Enums PERCENTAGE ABSOLUTE SPECIFIC_NODES
value*	integer	

ActionOnPercentageEntities

The scope of entities on which the action is performed is set to **absolute**.

Property Name	Type	Description
specifiedEntityActionScope	string	**SpecifiedEntityActionScope** Enums PERCENTAGE ABSOLUTE SPECIFIC_NODES
value*	integer	

ActionOnPercentageEntities

A list of nodes on which the action is performed.

Property Name	Type	Description
specifiedEntityActionScope	string	**SpecifiedEntityActionScope** Enums PERCENTAGE ABSOLUTE SPECIFIC_NODES
nodes*	string minItems: 1	

PolicySummaryArray

Property Name	Type
id*	integer
name*	string minLength: 1
enabled*	boolean

KeyValuePair

Property Name	Type
key*	string
value*	string

PolicySummary

Property Name	Type
id*	integer
name*	string minLength: 1
enabled*	boolean

PolicyConfiguration

Property Name	Type
`enabled`*	boolean
`policyName`	string

EntityMatchingPattern

Entities that match the specified pattern.

Property Name	Type	Description

`matchTo`*	string	Enums
		`STARTS_WITH`
		`ENDS_WITH`
		`CONTAINS`
		`EQUALS`
		`MATCH_REG_EX`

`matchValue`*	string minLength: 1	
`shouldNot`	boolean default: `false`	

ErrorResponse

Property Name	Type
`statusCode`	integer
`message`	string

PropertyMatchCriteria

Property Name	Type	Description

`propertyMatchCriteria`	string
	default: ANY
		Enums
		`ANY`
		`ALL`

EntityMatchingPatternEnum

Property Name	Type	Description

`EntityMatchingPatternEnum`	string	Enums
		`STARTS_WITH`
		`ENDS_WITH`
		`CONTAINS`
		`EQUALS`
		`MATCH_REG_EX`

BusinessTransactionScopeEnum

Property Name	Type	Description
businessTransactionScope	string	Enums ALL_BUSINESS_TRANSACTIONS SPECIFIC_BUSINESS_TRANSACTIONS BUSINESS_TRANSACTIONS_IN_SPECIFIC_TIERS BUSINESS_TRANSACTIONS_MATCHING_PATTERN

TierOrNodeScopeEnum

Property Name	Type	Description
TierOrNodeScope	string	Enums TIER_SELECTED_ENTITIES NODE_SELECTED_ENTITIES

SelectedTierScopeEnum

Property Name	Type	Description
SelectedTierScope	string	Enums ALL_TIERS SPECIFIC_TIERS

TypeOfNodeEnum

Property Name	Type	Description
typeofNode	string	Enums ALL_NODES JAVA_NODES DOT_NET_NODES PHP_NODES

SelectedNodesScopeEnum

Property Name	Type	Description
selectedNodeScope	string	Enums ALL_NODES SPECIFIC_NODES NODES_OF_SPECIFIC_TIERS NODES_MATCHING_PATTERN NODE_PROPERTY_VARIABLE_MATCHER

NodePropertyTypeEnum

Property Name	Type	Description
NodePropertyTypeEnum	string	Enums META ENV JVM

ErrorScopeEnum

Property Name	Type	Description
ErrorScopeEnum	string	Enums ALL_ERRORS SPECIFIC_ERRORS ERRORS_OF_SPECIFIC_TIERS ERRORS_MATCHING_PATTERN

ServiceEndpointScopeEnum

Property Name	Type	Description
ServiceEndpointScopeEnum	string	Enums ALL_SERVICE_ENDPOINTS SPECIFIC_SERVICE_ENDPOINTS SERVICE_ENDPOINTS_IN_SPECIFIC_TIERS SERVICE_ENDPOINTS_MATCHING_PATTERN

InformationPointScopeEnum

Property Name	Type	Description
InformationPointScopeEnum	string	ALL_INFORMATION_POINTS SPECIFIC_INFORMATION_POINTS INFORMATION_POINTS_MATCHING_PATTERN

DatabaseTypeEnum

Property Name	Type	Description

DatabaseTypeEnum	string	**Enums** ALL_DATABASE_TYPES COUCHBASE DB2 MONGO_DB MICROSOFT_SQL_SERVER MYSQL ORACLE POSTGRE_SQL AZURE_SQL SYBASE

ApplicationDatabaseScopeEnum

Property Name	Type	Description
ApplicationDatabaseScopeEnum	string	**Enums** ALL_APPLICATION_DATABASES SPECIFIC_APPLICATION_DATABASES APPLICATION_DATABASES_MATCHING_PATTERN

ServersScopeEnum

Property Name	Type	Description
ServersScopeEnum	string	**Enums** ALL_SERVERS_IN_APPLICATION SPECIFIC_SERVERS_IN_APPLICATION ALL_SERVERS_IN_SPECIFIC_TIERS

SpecifiedEntityActionScopeEnum

Property Name	Type	Description
SpecifiedEntityActionScopeEnum	string	**Enums** PERCENTAGE ABSOLUTE SPECIFIC_NODES

AnomalyEventType

Property Name	Type	Description

AnomalyEventType	string	Enums ANOMALY_OPEN_WARNING ANOMALY_OPEN_CRITICAL ANOMALY_UPGRADED ANOMALY_DOWNGRADED ANOMALY_CLOSE_WARNING ANOMALY_CLOSE_CRITICAL ANOMALY_CANCELED_WARNING ANOMALY_CANCELED_CRITICAL

HealthRuleEventTypeEnum

Property Name	Type	Description
HealthRuleEventTypeEnum	string	Enums HEALTH_RULE_CONTINUES_CRITICAL HEALTH_RULE_OPEN_CRITICAL HEALTH_RULE_OPEN_WARNING HEALTH_RULE_UPGRADED HEALTH_RULE_DOWNGRADED HEALTH_RULE_CONTINUES_WARNING HEALTH_RULE_CLOSE_WARNING HEALTH_RULE_CLOSE_CRITICAL HEALTH_RULE_CANCELED_WARNING HEALTH_RULE_CANCELED_CRITICAL

HealthRuleScopeType

Property Name	Type	Description
HealthRuleScopeType	string	ALL_HEALTH_RULES SPECIFIC_HEALTH_RULES

OtherEventType

Property Name	Type	Description

OtherEventType	string	Enums CLR_CRASH APPLICATION_CRASH DEADLOCK RESOURCE_POOL_LIMIT APPLICATION_DEPLOYMENT APP_SERVER_RESTART APPLICATION_CONFIG_CHANGE AGENT_CONFIGURATION_ERROR APPLICATION_DISCOVERED TIER_DISCOVERED NODE_DISCOVERED MACHINE_DISCOVERED BT_DISCOVERED SERVICE_ENDPOINT_DISCOVERED BACKEND_DISCOVERED EUM_CLOUD_SYNTHETIC_HEALTHY_EVENT EUM_CLOUD_SYNTHETIC_WARNING_EVENT EUM_CLOUD_SYNTHETIC_CONFIRMED_WARNING_EVENT EUM_CLOUD_SYNTHETIC_ONGOING_WARNING_EVENT EUM_CLOUD_SYNTHETIC_ERROR_EVENT EUM_CLOUD_SYNTHETIC_CONFIRMED_ERROR_EVENT EUM_CLOUD_SYNTHETIC_ONGOING_ERROR_EVENT EUM_CLOUD_SYNTHETIC_PERF_HEALTHY_EVENT EUM_CLOUD_SYNTHETIC_PERF_WARNING_EVENT EUM_CLOUD_SYNTHETIC_PERF_CONFIRMED_WARNING_EVENT EUM_CLOUD_SYNTHETIC_PERF_ONGOING_WARNING_EVENT EUM_CLOUD_SYNTHETIC_PERF_CRITICAL_EVENT EUM_CLOUD_SYNTHETIC_PERF_CONFIRMED_CRITICAL_EVENT EUM_CLOUD_SYNTHETIC_PERF_ONGOING_CRITICAL_EVENT MOBILE_NEW_CRASH_EVENT, SLOW, VERY_SLOW, STALL ERROR

SelectedEntityType

Property Name	Type	Description
SelectedEntityType	string	Enums ANY_ENTITY SPECIFIC_ENTITIES

EntityType

Property Name	Type	Description

`EntityType`	string	Enums
		`BUSINESS_TRANSACTION`
		`TIER_NODE`
		`ERRORS`
		`SERVICE_ENDPOINTS`
		`INFORMATION_POINTS`
		`DATABASES_IN_APPLICATION`
		`SERVERS_IN_APPLICATION`

ActionTypeEnum

Property Name	Type	Description

`ActionTypeEnum`	string	**Enums**
		`SMS`
		`EMAIL`
		`CUSTOM_EMAIL`
		`THREAD_DUMP`
		`HTTP_REQUEST`
		`CUSTOM`
		`RUN_SCRIPT_ON_NODES`
		`DIAGNOSTIC_BUSINESS_TRANSACTIONS`
		`CREATE_UPDATE_JIRA`

***This property is required.**

Download Examples

Download a set of examples that help you configure a policy, `AppDynamicsPoliciesExamples.zip`.

Download SWAGGER YAML Spec

Download the Swagger YAML spec `policies_openapi.yml`.

Actions API

This page describes the Action API methods you can use to create, configure, and manage various actions that are to be triggered as a response to events. Use this API to create these types of actions:

- Notification
- Diagnostic
- Remediation
- JIRA
- HTTP Request
- Custom

Syntax validation of the JSON payload is done when creating the action.

Create a New Action

Creates a new action with the specified JSON payload. See Property Details.

Resource URL

```
POST <controller_url>/controller/alerting/rest/v1
/applications/<application_id>/actions
```

Request/Response Format

JSON

Example

Retrieve a List of Actions for a Given Application

Returns the action ID, name, and description of the action pertaining to a specified application ID. See Property Details.

Resource URL

```
GET <controller_url>/controller/alerting/rest/v1/applications/<application_id>/actions
```

Response Format

JSON

Example Response

This example returns a list of actions pertaining to a given application ID.

```
[{"id":1,"name":"Thread Dump Action","actionType":"THREAD_DUMP"}]
```

Retrieve Details of a Specified Action

Retrieves the details of action with a specified ID. See Property Details.

Ensure that you provide a valid action ID.

Resource URL

```
GET <controller_url>/controller/alerting/rest/v1/applications/<application_id>/actions/{action-id}
```

Response Format

JSON

Example Response

This example retrieves the details of an action. See Download Examples.

```
{
 "id": 1,
 "actionType": "THREAD_DUMP",
 "name": "Thread Dump Action",
 "numberOfThreadDumps": 2,
 "intervalInMs": 500,
 "approvalBeforeExecution": {
 "requireApproval": true,
 "approverEmail": "email@website.com"
 }
}
```

Update an Action

Updates an existing action with a specified JSON payload. See Property Details.

> This request requires a complete JSON payload as input. Hence, it is recommended that you retrieve the JSON payload using, `GET /action/{action-id}` and update the required fields. Then, send the modified payload as part of a `PUT` request.

Resource URL

```
PUT <controller_url>/controller/alerting/rest/v1/applications/<application_id>/action/{action-id}
```

Request/Response Format

JSON

Example

This example updates an action. See Download Examples.

Delete an Action

Deletes an action with the specified ID. See Property Details.

> Ensure that you provide a valid action ID.

Resource URL

```
DELETE <controller_url>/controller/alerting/rest/v1/applications/<application_id>/action/{action-id}
```

Response Codes

Code	Description
200	Fetched successfully
201	Created successfully
204	Deleted successfully
400	Bad request
401	Unauthorized
403	Forbidden

404	Resource not found
409	Already exists

Property Details

Action

Payload details for an action triggered as a response to an event.

Property Name	Type	Description and Valid Values
`id`	integer	This is auto-generated by the system and returned in the response. It is a `readOnly` value.
`actionType`*	string	The type of action triggered as a response to an event. **Enums** SMS EMAIL CUSTOM_EMAIL THREAD_DUMP HTTP_REQUEST CUSTOM RUN_SCRIPT_ON_NODES DIAGNOSE_BUSINESS_TRANSACTIONS CREATE_UPDATE_JIRA

SmsAction

An SMS notification is triggered as a response to an event.

> **Note**
>
> Ensure that you have configured the email and SMS settings for AppDynamics. See Configure the SMTP Server.

Property Name	Type	Description and Valid Values
`id`	integer	This is auto-generated by the system and returned in the response. It is a `readOnly` value.

`actionType`*	string	The type of action triggered as a response to an event. **Enums** SMS EMAIL CUSTOM_EMAIL THREAD_DUMP HTTP_REQUEST CUSTOM RUN_SCRIPT_ON_NODES DIAGNOSE_BUSINESS_TRANSACTIONS CREATE_UPDATE_JIRA
`phoneNumber`*	string pattern: `^\d{10,}$`	

EmailAction

An email notification is triggered as a response to an event.

Property Name	Type	Description and Valid Values
`id`	integer	This is auto-generated by the system and returned in the response. It is a `readOnly` value.
`actionType`*	string	The type of action triggered as a response to an event. **Enums** SMS EMAIL CUSTOM_EMAIL THREAD_DUMP HTTP_REQUEST CUSTOM RUN_SCRIPT_ON_NODES DIAGNOSE_BUSINESS_TRANSACTIONS CREATE_UPDATE_JIRA
`emails`*	string pattern `^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$`	

CustomEmailAction

An email notification based on a predefined template is triggered as a response to an event.

The template must already exist before you can use it in an action. See Email Templates.

Property Name	Type	Description and Valid Values
`id`	integer	This is auto-generated by the system and returned in the response. It is a `readOnly` value.
`actionType`*	string	The type of action triggered as a response to an event. **Enums** SMS EMAIL CUSTOM_EMAIL THREAD_DUMP HTTP_REQUEST CUSTOM RUN_SCRIPT_ON_NODES DIAGNOSE_BUSINESS_TRANSACTIONS CREATE_UPDATE_JIRA
`name`*	string minLength: 1	**ActionName** The name you assign to the action.
`emailTemplateName`*	string minLength: 1	The name of the template to be used for email notification.
`to`*	string pattern: `^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$`	**EmailArray** A list of email IDs.
`cc`*	string pattern: `^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$`	**EmailArray** A list of email IDs.
`bcc`*	string pattern: `^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$`	**EmailArray** A list of email IDs.
`customTemplateVariables`*	string	**KeyValuePair** Property Name / Type `key`* — string `value`* — string

ThreadDumpAction

Property Name	Type	Description and Valid Values
`id`	integer	This is auto-generated by the system and returned in the response. It is a `readOnly` value.

`actionType`*	string	The type of action triggered as a response to an event. **Enums** SMS EMAIL CUSTOM_EMAIL THREAD_DUMP HTTP_REQUEST CUSTOM RUN_SCRIPT_ON_NODES DIAGNOSE_BUSINESS_TRANSACTIONS CREATE_UPDATE_JIRA
`name`*	string minLength: 1	**ActionName** The name you assign to the action.
`numberOfThreadDumps`*	integer minimum: 1 maximum: 50	The number of thread dump samples you want the 'action' to collect.
`intervalInMs`*	integer minimum: 500	The time interval in milliseconds between the thread dump samples collected.
`approvalBeforeExecution`*		**ApprovalBeforeExecution** Mandate an approval before the thread dump action is started. **Property Name** / **Type** `requireApproval` — boolean `approverEmail` — string pattern: `^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$`

HttpRequestAction

Property Name	Type	Description and Valid Values
`id`	integer	This is auto-generated by the system and returned in the response. It is a `readOnly` value.

`actionType`*	string	The type of action triggered as a response to an event. **Enums** SMS EMAIL CUSTOM_EMAIL THREAD_DUMP HTTP_REQUEST CUSTOM RUN_SCRIPT_ON_NODES DIAGNOSE_BUSINESS_TRANSACTIONS CREATE_UPDATE_JIRA
`name`*	string minLength: 1	**ActionName** The name you assign to the action.
`httpRequestTemplateName`	string minLength: 1	An existing HTTP request template to be used in an HTTP request action.
`customTemplateVariables`		**KeyValuePair** **Property Name** / **Type** `key`* — string `value`* — string

CustomAction

Property Name	Type	Description and Valid Values
`id`	integer	This is auto-generated by the system and returned in the response. It is a `readOnly` value.
`actionType`*	string	The type of action triggered as a response to an event. **Enums** SMS EMAIL CUSTOM_EMAIL THREAD_DUMP HTTP_REQUEST CUSTOM RUN_SCRIPT_ON_NODES DIAGNOSE_BUSINESS_TRANSACTIONS CREATE_UPDATE_JIRA
`name`*	string minLength: 1	**ActionName** The name you assign to the action.
`customActionName`*	string minLength: 1	

ScriptAction

Property Name	Type	Description and Valid Values
`id`	integer	This is auto-generated by the system and returned in the response. It is a `readOnly` value.
`actionType`*	string	The type of action triggered as a response to an event. **Enums** SMS EMAIL CUSTOM_EMAIL THREAD_DUMP HTTP_REQUEST CUSTOM RUN_SCRIPT_ON_NODES DIAGNOSE_BUSINESS_TRANSACTIONS CREATE_UPDATE_JIRA
`name`*	string minLength: 1	**ActionName** The name you assign to the action.
`scriptPath`*	string minLength: 1	The relative path of the script. Enter the part after `${machine.agent.directory}/local-scripts/`
`logFilesPath`		The absolute path of the log file(s).
`scriptTimeout`*	string integer minimum: 1 maximum: 1440	
`approvalBeforeExecution`*		Mandate approval before the script action is started. **Property Name** / **Type** `requireApproval` — boolean `approverEmail` — string pattern: `^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$`

DiagnosticAction

Property Name	Type	Description and Valid Values
`id`	integer	This is auto-generated by the system and returned in the response. It is a `readOnly` value.

`actionType`*	string	The type of action triggered as a response to an event. **Enums** SMS EMAIL CUSTOM_EMAIL THREAD_DUMP HTTP_REQUEST CUSTOM RUN_SCRIPT_ON_NODES DIAGNOSE_BUSINESS_TRANSACTIONS CREATE_UPDATE_JIRA
`name`*	string minLength: 1	**ActionName** The name you assign to the action.
`duration`*	integer minimum: 1 maximum: 10	The duration in minutes to run the diagnostic session.
`snapshotRate`*	integer minimum: 1 maximum: 10	The rate at which diagnostic snapshots are captured.
`businessTransactions`*	string	Runs the diagnostic session on the specified Business Transactions. **Enums** ALL_AFFECTED_BUSINESS_TRANSACTIONS SPECIFIC_BUSINESS_TRANSACTIONS

JiraAction

Property Name	Type	Description and Valid Values
`id`	integer	This is auto-generated by the system and returned in the response. It is a `readOnly` value.
`actionType`*	string	The type of action triggered as a response to an event. **Enums** SMS EMAIL CUSTOM_EMAIL THREAD_DUMP HTTP_REQUEST CUSTOM RUN_SCRIPT_ON_NODES DIAGNOSE_BUSINESS_TRANSACTIONS CREATE_UPDATE_JIRA
`name`*	string minLength: 1	**ActionName** The name you assign to the action.

`jiraActionDetails`*	string	**jiraActionType** The JIRA action type triggered as a response to an event. **Enums** CREATE_JIRA UPDATE_JIRA

JiraActionDetails

The JIRA action type triggered as a response to an event.

Property Name	Type	Description and Valid Values
`jiraActionType`	string	**Enums** CREATE_JIRA UPDATE_JIRA

JiraCreateAction

The Jira action type `create JIRA` is triggered as a response to an event.

Property Name	Type	Description and Valid Values
`jiraActionType`*	string	**Enums** CREATE_JIRA UPDATE_JIRA
`assignee`*	string minLength: 1	
`project`*	string minLength: 1	
`priority`*	string minLength: 1	
`issueType`*	string minLength: 1	

JiraUpdateAction

The Jira action type `update JIRA` is triggered as a response to an event.

Property Name	Type	Description and Valid Values
`jiraActionType`*	string	**Enums** CREATE_JIRA UPDATE_JIRA
`changePriorityTo`*	string minLength: 1	

ActionName

The name you assign to the action.

Property Name	Type	Description and Valid Values
`ActionName`	string minLength: 1	

ApprovalBeforeExecution

Mandate email approval before the action execution is initiated.

Property Name	Type
`requireApproval`	boolean
`approverEmail`	string pattern: `^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$`

BusinessTransactions

Run the diagnostic session on the specified Business Transactions.

Property Name	Type	Description and Valid Values
`businessTransactionScope`	string	**Enums** ALL_AFFECTED_BUSINESS_TRANSACTIONS SPECIFIC_BUSINESS_TRANSACTIONS

AllAffectedBusinessTransactions

Run the diagnostic session on all Business Transactions.

Property Name	Type	Description and Valid Values
`businessTransactionScope`	string	**Enums** ALL_AFFECTED_BUSINESS_TRANSACTIONS SPECIFIC_BUSINESS_TRANSACTIONS

SpecificBusinessTransactions

Run the diagnostic session on the Business Transactions that match the specified criteria.

Property Name	Type	Description and Valid Values
`businessTransactionScope`	string	**Enums** ALL_AFFECTED_BUSINESS_TRANSACTIONS SPECIFIC_BUSINESS_TRANSACTIONS
`businessTransactionNames`*	string minItems: 1	

EmailArray

Property Name	Type
`EmailArray`	string pattern: `^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$`

KeyValuePair

Property Name	Type
`key`*	string
`value`*	string

Email

Property Name	Type
`Email`	string pattern: ^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$

ActionSummaryArray

Property Name	Type	Description and Valid Values
`id`	integer	This is auto-generated by the system and returned in the response. It is a `readOnly` value.
`name`*	string minLength: 1	**ActionName** The name you assign to the action.
`actionType`*	string	The type of action triggered as a response to an event. **Enums** SMS EMAIL CUSTOM_EMAIL THREAD_DUMP HTTP_REQUEST CUSTOM RUN_SCRIPT_ON_NODES DIAGNOSE_BUSINESS_TRANSACTIONS CREATE_UPDATE_JIRA

ActionSummary

Property Name	Type	Description and Valid Values
`id`	integer	This is auto-generated by the system and returned in the response. It is a `readOnly` value.
`name`*	string minLength: 1	**ActionName** The name you assign to the action.

`actionType`*	string	The type of action triggered as a response to an event. **Enums** SMS EMAIL CUSTOM_EMAIL THREAD_DUMP HTTP_REQUEST CUSTOM RUN_SCRIPT_ON_NODES DIAGNOSE_BUSINESS_TRANSACTIONS CREATE_UPDATE_JIRA

StringIntegerPair

Property Name	Type
`id`	integer
`name`*	string minLength: 1

JiraActionTypeEnum

Property Name	Type	Description and Valid Values
`jiraActionTypeEnum`	string	CREATE_JIRA UPDATE_JIRA

BusinessTransactionScopeEnum

Property Name	Type	Description and Valid Values
`BusinessTransactionScopeEnum`	string	ALL_AFFECTED_BUSINESS_TRANSACTIONS SPECIFIC_BUSINESS_TRANSACTIONS

ActionTypeEnum

Property Name	Type	Description and Valid Values
`ActionTypeEnum`	string	**Enums** SMS EMAIL CUSTOM_EMAIL THREAD_DUMP HTTP_REQUEST CUSTOM RUN_SCRIPT_ON_NODES DIAGNOSE_BUSINESS_TRANSACTIONS CREATE_UPDATE_JIRA

ErrorResponse

Property Name	Type
`statusCode`	integer
`message`	string

***This property is required.**

Download Examples

Download actions_api.zip for a set of examples that help you configure an action.

Download SWAGGER YAML file

Download the Swagger actions_openapi.yml.

Email Digest API

This page provides Email Digest API methods you can use to report a summary of a specific event(s) to a recipient list configured on a health rule schedule. Email digests are triggered as a response to a health rule violation event.

> ⚠ Syntax validation of the JSON payload is done when creating the email digest.

Create a New Email Digest

Creates a new email digest with the specified JSON payload. See Property Details.

Resource URL

```
POST <controller_url>/controller/alerting/rest/v1
/applications/<application_id>/email-digests
```

Request/Response Format

JSON

Example

Retrieve a List of Email Digests for an Application

This API returns the email digest names, IDs, and enable flag details pertaining to the specified application. See Property Details.

Resource URL

```
GET <controller_url>/controller/alerting/rest/v1/applications/<application_id>/email-digests
```

Response Format

JSON

Example Response

This example returns a list of email digests pertaining to the specified application ID.

```
[
  {
    "id": 12,
    "name": "some email digest name",
    "enabled": true
  }
]
```

Retrieve Details of an Email Digest

Retrieves the details of an email digest with a specified ID. See Property Details.

> ⓘ Ensure that you provide a valid email digest ID.

Resource URL

```
GET <controller_url>/controller/alerting/rest/v1/applications/<application_id>/email-digests/{email-digest-id}
```

Response Format

JSON

Example Response

```
{
  "id": 0,
  "name": "My new email digest",
  "enabled": true,
  "executeActionsInBatch": true,
  "frequency": 2,
  "actions": [
    {
      "actionName": "My action 1",
      "actionType": "SMS"
    }
  ],
  "events": {
    "healthRuleEvents": {
      "healthRuleEventTypes": [
        "HEALTH_RULE_CONTINUES_CRITICAL",
        "HEALTH_RULE_UPGRADED"
      ],
      "healthRuleScope": {
        "healthRuleScopeType": "ALL_HEALTH_RULES"
      }
    },
    "otherEvents": [
      "CLR_CRASH",
      "DEADLOCK"
    ],
    "anomalyEvents": [
      "ANOMALY_OPEN_WARNING",
      "ANOMALY_CLOSE_CRITICAL"
    ],
    "customEvents": [
      {
        "eventName": "string",
        "propertyMatchCriteria": "ANY",
        "keyValuePairArray": [
          {
            "key": "key1",
            "value": "value1"
          }
        ]
      }
    ]
  },
  "selectedEntities": {
    "selectedEntityType": "ANY_ENTITY"
  }
}
```

Update an Email Digest

Updates an existing email digest with a specified JSON payload. See Property Details.

This request requires a complete JSON payload as input. It is recommended that you retrieve the JSON payload using, `GET email-digests/{email-digest-id}` and update the required fields. Then, send the modified payload as part of `PUT` request.

Resource URL

`PUT <controller_url>/controller/alerting/rest/v1/applications/<application_id>/email-digests/{email-digest-id}`

Request/Response Format

JSON

Example

Delete an Email Digest

Deletes an email digest with the specified ID. See Property Details.

> Ensure that you provide a valid email digest ID.

Resource URL

```
DELETE <controller_url>/controller/alerting/rest/v1/applications/<application_id>/email-digests/{email-digest-id}
```

Update one or more properties of an Email Digest

Updates the properties of an existing email digest with a specified JSON payload. See Property Details

Resource URL

```
PUT <controller_url>/controller/alerting/rest/v1/applications/<application_id>/email-digests/{email-digest-id}
/configuration
```

Request/Response Format

JSON

Example

Response Codes

Code	Description
200	Fetched successfully
201	Created successfully
204	Deleted successfully
400	Bad request
401	Unauthorized
403	Forbidden
404	Resource not found
409	Already exists

Property Details

Email Digest

Payload details of an email digest.

Property Name	Type	Description and Valid Values
`id`	integer	This is auto-generated by the system and returned in the response. It is a `readOnly` value.
`name`*	string Minimum length: 1	Name of the email digest.

<table>
<tr><td>enabled</td><td>boolean

Default value: true</td><td>Sets the email digest to enabled/disabled state.</td></tr>
<tr><td>frequency</td><td>integer
minimum: 1
maximum: 168</td><td>The frequency in hours at which emails are sent as a response to an event.</td></tr>
<tr><td>actions*</td><td>minItems: 1</td><td>Action

Describes the predefined, reusable, and automated response to an event to be taken when the event is triggered.

<table><tr><th>Property Name</th><th>Type</th><th>Description</th></tr><tr><td>actionName*</td><td>string</td><td>Name of the action defined for an email digest.</td></tr><tr><td>actionType*</td><td>string</td><td>ActionTypeEnum

Creates the following types of actions:

SMS

EMAIL

CUSTOM_EMAIL

THREAD_DUMP

HTTP_REQUEST

CUSTOM

RUN_SCRIPT_ON_NODES

DIAGNOSTIC_BUSINESS_TRANSACTIONS

CREATE_UPDATE_JIRA</td></tr></table></td></tr>
<tr><td>events*</td><td></td><td>Events

Describes the events that trigger the email digest.

<table><tr><th>Property Name</th><th>Description</th></tr></table></td></tr>
</table>

healthRuleEvents	**HealthRuleEvents** Describes the trigger event type(s) generated due to health rule issues. **Property Name** / **Description** healthRuleEventTypes* — HealthRuleEventTypes Describes the event type related to health rule that triggers the email digest. minItems: 1 HealthRuleEventTypeEnum HEALTH_RULE_CONTINUES_CRITICAL HEALTH_RULE_OPEN_CRITICAL HEALTH_RULE_OPEN_WARNING HEALTH_RULE_UPGRADED HEALTH_RULE_DOWNGRADED HEALTH_RULE_CONTINUES_WARNING HEALTH_RULE_CLOSE_WARNING HEALTH_RULE_CLOSE_CRITICAL HEALTH_RULE_CANCELED_WARNING HEALTH_RULE_CANCELED_CRITICAL healthRuleScope* — **HealthRuleScope** Describes the scope of the health rule based on which, the events are triggered. Enums ALL_HEALTH_RULES SPECIFIC_HEALTH_RULES

otherEvents	**OtherEvents** Describes the trigger event type(s) generated due to other issues.

Property Name	Description
OtherEventType	Lists the event type that triggers an action. Enums CLR_CRASH APPLICATION_CRASH DEADLOCK RESOURCE_POOL_LIMIT APPLICATION_DEPLOYMENT APP_SERVER_RESTART APPLICATION_CONFIG_CHANGE AGENT_CONFIGURATION_ERROR APPLICATION_DISCOVERED TIER_DISCOVERED NODE_DISCOVERED MACHINE_DISCOVERED BT_DISCOVERED SERVICE_ENDPOINT_DISCOVERED BACKEND_DISCOVERED EUM_CLOUD_SYNTHETIC_HEALTHY_EVENT EUM_CLOUD_SYNTHETIC_WARNING_EVENT EUM_CLOUD_SYNTHETIC_CONFIRMED_WARNING_EVENT EUM_CLOUD_SYNTHETIC_ONGOING_WARNING_EVENT EUM_CLOUD_SYNTHETIC_ERROR_EVENT EUM_CLOUD_SYNTHETIC_CONFIRMED_ERROR_EVENT EUM_CLOUD_SYNTHETIC_ONGOING_ERROR_EVENT EUM_CLOUD_SYNTHETIC_PERF_HEALTHY_EVENT EUM_CLOUD_SYNTHETIC_PERF_WARNING_EVENT EUM_CLOUD_SYNTHETIC_PERF_CONFIRMED_WARNING_EVENT EUM_CLOUD_SYNTHETIC_PERF_ONGOING_WARNING_EVENT EUM_CLOUD_SYNTHETIC_PERF_CRITICAL_EVENT EUM_CLOUD_SYNTHETIC_PERF_CONFIRMED_CRITICAL_EVENT EUM_CLOUD_SYNTHETIC_PERF_ONGOING_CRITICAL_EVENT MOBILE_NEW_CRASH_EVENT, SLOW, VERY_SLOW, STALL ERROR

<table>
<tr><td></td><td></td><td>
<table>
<tr><td>anomalyEvents</td><td>

AnomalyEvents

Describes the trigger event type(s) generated due to anomaly detection.

minItems: 1

<table>
<tr><th>Property Name</th><th>Description</th></tr>
<tr><td>AnomalyEventType</td><td>Lists the event type that triggers an action.
Enums
ANOMALY_OPEN_WARNING
ANOMALY_OPEN_CRITICAL
ANOMALY_UPGRADED
ANOMALY_DOWNGRADED
ANOMALY_CLOSE_WARNING
ANOMALY_CLOSE_CRITICAL
ANOMALY_CANCELED_WARNING
ANOMALY_CANCELED_CRITICAL</td></tr>
</table>

</td></tr>
<tr><td>customEvents</td><td>

The custom event type(s) you define.

<table>
<tr><th>Property Name</th><th>Type</th><th>Description</th></tr>
<tr><td>eventName*</td><td>string</td><td>Custom event name.</td></tr>
<tr><td>propertyMatchCriteria</td><td>string</td><td>Enums
ANY
ALL</td></tr>
<tr><td>keyValuePairArray</td><td>string</td><td>KeyValuePair
key*
value*</td></tr>
</table>

</td></tr>
</table>
</td></tr>
<tr><td>selectedEntities</td><td></td><td>

SelectedEntityType

Scope of entities considered for the email digest.

Entities corresponding to EUM applications are not supported.

<table>
<tr><th>Property Name</th><th>Type</th><th>Description</th></tr>
<tr><td>SelectedEntityType*</td><td>string</td><td>Enums
ANY_ENTITY
SPECIFIC_ENTITIES</td></tr>
</table>

</td></tr>
</table>

SelectedEntityType

Scope of entities considered for the email digest.

Property Name	Type	Description

`SelectedEntityType`*	string	Enums
		`ANY_ENTITY`
		`SPECIFIC_ENTITIES`

SpecificEntities

Scope of specific entities considered for the email digest.

Property Name	Type	Description
`selectedEntityType`*	string	Enum `SPECIFIC_ENTITIES`
`entities`	minItems: 1	**Entity** **Property name** / **Type** / **Description** `entityType`* — string, minItems: 1 — Enums: `BUSINESS_TRANSACTION`, `TIER_NODE`, `ERRORS`, `SERVICE_ENDPOINTS`, `INFORMATION_POINTS`, `DATABASES_IN_APPLICATION`, `SERVERS_IN_APPLICATION`, `PAGE`, `AJAX_REQUEST`, `SYNTHETIC_JOBS`, `IFRAME`, `VIRTUAL_PAGE`, `MOBILE_APPS`, `MOBILE_NETWORK_REQUESTS`

Entity

Property name	Type	Description

`entityType`*	string
	minItems: 1
		Enums
		`BUSINESS_TRANSACTION`
		`TIER_NODE`
		`ERRORS`
		`SERVICE_ENDPOINTS`
		`INFORMATION_POINTS`
		`DATABASES_IN_APPLICATION`
		`SERVERS_IN_APPLICATION`
		`PAGE`
		`AJAX_REQUEST`
		`SYNTHETIC_JOBS`
		`IFRAME`
		`VIRTUAL_PAGE`
		`MOBILE_APPS`
		`MOBILE_NETWORK_REQUESTS`

AnyEntity

Scope of entities considered for the email digest.

Property Name	Type	Description
`SelectedEntityType`*	string	Enums `ANY_ENTITY`

SelectedMobileApps

Property name	Type	Description

`mobileAppsScope`*	string	Enums
		`ALL_MOBILE_APPS`
		`SPECIFIC_MOBILE_APPS`
		`MOBILE_APPS_MATCHING_PATTERN`

AllMobileApps

Property name	Type	Description
`mobileAppsScope`*	string	Enum `ALL_MOBILE_APPS`

SpecificMobileApps

Property name	Type	Description
`mobileAppsScope`*	string	Enum `SPECIFIC_MOBILE_APPS`
`mobileApps`	string minItems: 1	

MobileAppsMatchingPattern

Property name	Type	Description
`mobileAppsScope`*	string	Enum MOBILE_APPS_MATCHING_PATTERN

`patternMatcher`*		**EntityMatchingPattern**
		Business transactions that match the specified pattern are included in the scope.
		
		Property Name / **Type** / **Description**
		`matchTo`* — Enums: `STARTS_WITH`, `ENDS_WITH`, `CONTAINS`, `EQUALS`, `MATCH_REG_EX`
		`matchValue`* — string, minLength: 1
		`shouldNot` — boolean, default: `false`

MobileNetworkRequests

Property name	Type	Description
`entityType`*	string minItems: 1	Enum `MOBILE_NETWORK_REQUESTS`

`selectedMobileNetworkRequests`*	string	**MobileNetworkRequestsScope**
		Enums
		`ALL_MOBILE_NETWORK_REQUESTS`
		`SPECIFIC_MOBILE_NETWORK_REQUESTS`
		`SPECIFIC_MOBILE_APPS_NETWORK_REQUESTS`
		`MOBILE_NETWORK_REQUESTS_MATCHING_PATTERN`

SelectedMobileNetworkRequests

Property name	Type	Description

`mobileNetworkRequestsScope`*	string	**MobileNetworkRequestsScope**
		Enums
		`ALL_MOBILE_NETWORK_REQUESTS`
		`SPECIFIC_MOBILE_NETWORK_REQUESTS`
		`SPECIFIC_MOBILE_APPS_NETWORK_REQUESTS`
		`MOBILE_NETWORK_REQUESTS_MATCHING_PATTERN`

AllMobileNetworkRequests

The scope of mobile network requests is `ALL_MOBILE_NETWORK_REQUESTS`.

Property name	Type	Description
`mobileNetworkRequestsScope`*	string	**MobileNetworkRequestsScope** Enum `ALL_MOBILE_NETWORK_REQUESTS`

SpecificMobileNetworkRequests

The scope of mobile network requests is `SPECIFIC_MOBILE_NETWORK_REQUESTS`.

Property name	Type	Description
`mobileNetworkRequestsScope`*	string	**MobileNetworkRequestsScope** Enum `SPECIFIC_MOBILE_NETWORK_REQUESTS`
`mobileNetworkRequests`*	string minItems:1	

SpecificMobileAppsNetworkRequests

The scope of mobile network requests is `SPECIFIC_MOBILE_APPS_NETWORK_REQUESTS`.

Property name	Type	Description
`mobileNetworkRequestsScope`*	string	**MobileNetworkRequestsScope** Enum `SPECIFIC_MOBILE_APPS_NETWORK_REQUESTS`
`mobileApps`*	string minItems:1	

MobileNetworkRequestsMatchingPattern

The scope of mobile network requests is `MOBILE_NETWORK_REQUESTS_MATCHING_PATTERN`.

Property name	Type	Description
`mobileNetworkRequestsScope`*	string	**MobileNetworkRequestsScope** Enum `MOBILE_NETWORK_REQUESTS_MATCHING_PATTERN`

patternMatcher*		**EntityMatchingPattern** Business transactions that match the specified pattern are included in the scope. **Property Name** / **Type** / **Description** matchTo* / / Enums: STARTS_WITH, ENDS_WITH, CONTAINS, EQUALS, MATCH_REG_EX matchValue* / string, minLength: 1 / shouldNot / boolean, default: false /

Page

The scope of entities considered for the email digest is page for EUM applications.

Property name	Type	Description
entityType*	string minItems: 1	Enum PAGE
selectedPages*	string	**pageScope** Enums ALL_PAGES SPECIFIC_PAGES PAGES_MATCHING_PATTERN

SelectedPages

The scope of entities considered for the email digest is selected pages for EUM applications.

Property name	Type	Description
pageScope*	string	Enums ALL_PAGES SPECIFIC_PAGES PAGES_MATCHING_PATTERN

AllPages

The scope of entities considered for the email digest is ALL_PAGES for EUM applications.

Property name	Type	Description
pageScope*	string	Enum ALL_PAGES

SpecificPages

The scope of entities considered for the email digest is `SPECIFIC_PAGES` for EUM applications.

Property name	Type	Description
`pageScope`*	string	Enum `SPECIFIC_PAGES`
`pages`*	array of strings minItems: 1	

PagesMatchingPattern

The scope of entities considered for the email digest is `SPECIFIC_PAGES` for EUM applications.

Property name	Type	Description
`pageScope`*	string	Enum `PAGES_MATCHING_PATTERN`
`pages`*		**EntityMatchingPattern** Business transactions that match the specified pattern are included in the scope. **Property Name** / **Type** / **Description** `matchTo`* / / Enums: `STARTS_WITH`, `ENDS_WITH`, `CONTAINS`, `EQUALS`, `MATCH_REG_EX` `matchValue`* / string, minLength: 1 / `shouldNot` / boolean, default: `false` /

VirtualPage

Property name	Type	Description
`entityType`*	string minItems: 1	Enum `VIRTUAL_PAGE`

`selectedVirtualPages`*	string	**SelectedVirtualPages**
		Enums
		`ALL_VIRTUAL_PAGES`
		`SPECIFIC_VIRTUAL_PAGES`
		`VIRTUAL_PAGES_MATCHING_PATTERN`

SelectedVirtualPages

Property name	Type	Description

`virtualPageScope`*	string	**SelectedVirtualPages**
		Enums
		`ALL_VIRTUAL_PAGES`
		`SPECIFIC_VIRTUAL_PAGES`
		`VIRTUAL_PAGES_MATCHING_PATTERN`

AllVirtualPages

Property name	Type	Description
`virtualPageScope`*	string	**SelectedVirtualPages** Enum `ALL_VIRTUAL_PAGES`

SpecificVirtualPages

Property name	Type	Description
`virtualPageScope`*	string	**SelectedVirtualPages** Enum `SPECIFIC_VIRTUAL_PAGES`
`virtualPages`	string minItems: 1	

VirtualPagesMatchingPattern

Property name	Type	Description
`virtualPageScope`*	string	**SelectedVirtualPages** Enum `VIRTUAL_PAGES_MATCHING_PATTERN`

patternMatcher		**EntityMatchingPattern** Business transactions that match the specified pattern are included in the scope. **Property Name** / **Type** / **Description** matchTo* — Enums: STARTS_WITH, ENDS_WITH, CONTAINS, EQUALS, MATCH_REG_EX matchValue* — string, minLength: 1 shouldNot — boolean, default: false

SyntheticJob

Property name	Type	Description
entityType*	string minItems: 1	Enum SYNTHETIC_JOBS
selectedSyntheticJobs*	string	**SelectedSyntheticJobs** Enums ALL_SYNTHETIC_JOBS SPECIFIC_SYNTHETIC_JOBS SYNTHETIC_JOBS_MATCHING_PATTERN

SelectedSyntheticJobs

Property name	Type	Description
syntheticJobScope*	string	**Enums** ALL_SYNTHETIC_JOBS SPECIFIC_SYNTHETIC_JOBS SYNTHETIC_JOBS_MATCHING_PATTERN

AllSyntheticJobs

Property name	Type	Description
syntheticJobScope*	string	**Enum** ALL_SYNTHETIC_JOBS

SpecificSyntheticJobs

Property name	Type	Description
`syntheticJobScope`*	string	**Enum** `SPECIFIC_SYNTHETIC_JOBS`
`syntheticJobs`*	string minItems: 1	

SyntheticJobsMatchingPattern

Property name	Type	Description
`syntheticJobScope`*	string	**Enum** `SYNTHETIC_JOBS_MATCHING_PATTERN`
`patternMatcher`*		**EntityMatchingPattern** Business transactions that match the specified pattern are included in the scope. **Property Name** / **Type** / **Description** matchTo* — Enums: `STARTS_WITH`, `ENDS_WITH`, `CONTAINS`, `EQUALS`, `MATCH_REG_EX` matchValue* — string, minLength: 1 shouldNot — boolean, default: `false`

IFrame

Property name	Type	Description
`entityType`*	string minItems: 1	Enum `IFRAME`

`selectedIFrames`*	string	**iFrameScope**
		Enums
		`ALL_IFRAMES`
		`SPECIFIC_IFRAMES`
		`IFRAMES_MATCHING_PATTERN`

SelectedIFrames

Property name	Type	Description

`iFrameScope`*	string	Enums
		`ALL_IFRAMES`
		`SPECIFIC_IFRAMES`
		`IFRAMES_MATCHING_PATTERN`

AllIFrames

Property name	Type	Description
`iFrameScope`*	string	Enum `ALL_IFRAMES`

SpecificIFrames

Property name	Type	Description
`iFrameScope`*	string	Enum `SPECIFIC_IFRAMES`
`iFrames`*	string minItems: 1	

IFramesMatchingPattern

Property name	Type	Description
`iFrameScope`*	string	Enum `IFRAMES_MATCHING_PATTERN`
`patternMatcher`*		**EntityMatchingPattern** Business transactions that match the specified pattern are included in the scope.

Property Name	Type	Description

`matchTo`*		Enums
		`STARTS_WITH`
		`ENDS_WITH`
		`CONTAINS`
		`EQUALS`
		`MATCH_REG_EX`

`matchValue`*	string minLength: 1	
`shouldNot`	boolean default: `false`	

AjaxRequest

Property name	Type	Description

`entityType`*	string minItems: 1	Enum `AJAX_REQUEST`

`selectedAjaxRequests`*	string	**AjaxRequestsScope**
		Enums
		`ALL_AJAX_REQUESTS`
		`SPECIFIC_AJAX_REQUESTS`
		`AJAX_REQUESTS_MATCHING_PATTERN`

SelectedAjaxRequests

Property name	Type	Description

`selectedAjaxRequests`*	string	**Enums**
		`ALL_AJAX_REQUESTS`
		`SPECIFIC_AJAX_REQUESTS`
		`AJAX_REQUESTS_MATCHING_PATTERN`

AllAjaxRequests

Property name	Type	Description
`selectedAjaxRequests`*	string	**Enum** `ALL_AJAX_REQUESTS`

SpecificAjaxRequests

Property name	Type	Description
`selectedAjaxRequests`*	string	**Enum** `SPECIFIC_AJAX_REQUESTS`
`ajaxRequests`*	array of strings minItems: 1	

AjaxRequestsMatchingPattern

Property name	Type	Description
`ajaxRequestScope`*	string	**Enum** `AJAX_REQUESTS_MATCHING_PATTERN`

`patternMatcher`*		**EntityMatchingPattern**
		Business transactions that match the specified pattern are included in the scope.
		Property Name / **Type** / **Description**
		`matchTo`* — Enums: `STARTS_WITH`, `ENDS_WITH`, `CONTAINS`, `EQUALS`, `MATCH_REG_EX`
		`matchValue`* — string, minLength: 1
		`shouldNot` — boolean, default: `false`

BusinessTransaction

All entities of type `BUSINESS_TRANSACTION` are considered for email digest.

Property Name	Type	Description
`entityType`*	string	Enum BUSINESS_TRANSACTION
`selectedBusinessTransactions`*	string	**BusinessTransactionScope** **Property Name** / **Description** `businessTransactionScope`* — Enums: `ALL_BUSINESS_TRANSACTIONS`, `SPECIFIC_BUSINESS_TRANSACTIONS`, `BUSINESS_TRANSACTIONS_IN_SPECIFIC_TIERS`, `BUSINESS_TRANSACTIONS_MATCHING_PATTERN`

SelectedBusinessTransactions

Property Name	Description

`businessTransactionScope`*	Enums
	`ALL_BUSINESS_TRANSACTIONS`
	`SPECIFIC_BUSINESS_TRANSACTIONS`
	`BUSINESS_TRANSACTIONS_IN_SPECIFIC_TIERS`
	`BUSINESS_TRANSACTIONS_MATCHING_PATTERN`

AllBusinessTransactions

The scope of business transactions is set to all business transactions.

Property Name	Description

`businessTransactionScope`*	Enum `ALL_BUSINESS_TRANSACTIONS`

SpecificBusinessTransactions

The scope of business transactions is set to select business transactions.

Property Name	Type	Description
`businessTransactionScope`*	string	Enum `SPECIFIC_BUSINESS_TRANSACTIONS`
`businessTransactions`*	array of strings minItems: 1	Name(s) of the business transactions.

BusinessTransactionsInSpecificTiers

The scope of business transactions is set to business transactions associated with a specific tier.

Property Name	Type	Description
`businessTransactionScope`*		Enum `BUSINESS_TRANSACTIONS_IN_SPECIFIC_TIERS`
`specificTiers`*	string minItems: 1	Name of the specified tier.

BusinessTransactionsMatchingPattern

The scope of business transactions is set to business transactions that match a specific pattern.

<table>
<tr><th>Property Name</th><th>Description</th></tr>
<tr><td><code>businessTransactionScope</code>*</td><td>Enum
<code>BUSINESS_TRANSACTIONS_MATCHING_PATTERN</code></td></tr>
<tr><td><code>patternMatcher</code>*</td><td>EntityMatchingPattern
Business transactions that match the specified pattern are included in the scope.
<table>
<tr><th>Property Name</th><th>Type</th><th>Description</th></tr>
<tr><td><code>matchTo</code>*</td><td></td><td>Enums
<code>STARTS_WITH</code>
<code>ENDS_WITH</code>
<code>CONTAINS</code>
<code>EQUALS</code>
<code>MATCH_REG_EX</code></td></tr>
<tr><td><code>matchValue</code>*</td><td>string
minLength: 1</td><td></td></tr>
<tr><td><code>shouldNot</code></td><td>boolean
default: <code>false</code></td><td></td></tr>
</table>
</td></tr>
</table>

SelectedTierOrNodeEntities

Specific tiers or nodes are considered for the email digest.

Property Name	Type	Description
`entityType`*	string	Enum `TIER_NODE`

`tierOrNode`*	string
	minLength: 1
		TierOrNode
		Property Name / **Type** / **Description**
		`tierOrNodeScope`* — string — **Enums**
		`TIER_SELECTED_ENTITIES`
		`NODE_SELECTED_ENTITIES`

TierOrNode

Property Name	Type	Description

`tierOrNodeScope`*	string	**Enums**
		`TIER_SELECTED_ENTITIES`
		`NODE_SELECTED_ENTITIES`

TierSelectedEntities

Property Name	Type	Description
`tierOrNodeScope`*	string	**Enums** `TIER_SELECTED_ENTITIES`

`selectedTiers`*		**SelectedTiers**
		Property Name / **Type** / **Description**
		`selectedTierScope`* — string — Enums
		`ALL_TIERS`
		`SPECIFIC_TIERS`

SelectedTiers

Property Name	Type	Description

`selectedTierScope`*	string	Enums
		`ALL_TIERS`
		`SPECIFIC_TIERS`

AllTiers

Property Name	Type	Description
`selectedTierScope`*	string	Enum `ALL_TIERS`

SpecificTiers

Property Name	Type	Description
`selectedTierScope`*	string	Enum `SPECIFIC_TIERS`
`tiers`*	string minItems: 1	Name(s) of the specified tier(s).

NodeSelectedEntities

Property Name	Type	Description
`tierOrNodeScope`*	string	Enum NODE_SELECTED_ENTITIES
`typeofNode`*	string	Enums ALL_NODES JAVA_NODES DOT_NET_NODES PHP_NODES
`selectedNodes`*		**SelectedNodes** **Property Name** / **Type** / **Description** `selectedNodeScope`* — string — Enums: `ALL_NODES`, `SPECIFIC_NODES`, `NODES_OF_SPECIFIC_TIERS`, `NODES_MATCHING_PATTERN`, `NODE_PROPERTY_VARIABLE_MATCHER`

SelectedNodes

Property Name	Type	Description

`selectedNodeScope`*	string	Enums
		`ALL_NODES`
		`SPECIFIC_NODES`
		`NODES_OF_SPECIFIC_TIERS`
		`NODES_MATCHING_PATTERN`
		`NODE_PROPERTY_VARIABLE_MATCHER`

AllNodes

Property Name	Type	Description
`selectedNodeScope`*	string	Enum `ALL_NODES`

SpecificNodes

Property Name	Type	Description
`selectedNodeScope`*	string	Enum `SPECIFIC_NODES`
`nodes`*	string minItems: 1	Name(s) of the specified node(s).

NodesOfSpecificTiers

Property Name	Type	Description
`selectedNodeScope`*	string	Enum `NODES_OF_SPECIFIC_TIERS`
`specificTiers`*	string minItems: 1	Name(s) of tier with the associated nodes.

NodesMatchingPattern

Property Name	Type	Description
`selectedNodeScope`*	string	Enum `NODES_MATCHING_PATTERN`
`patternMatcher`*		**EntityMatchingPattern** Nodes that match a specified pattern are included in the scope.

Property Name	Type	Description

`matchTo`*		Enums
		`STARTS_WITH`
		`ENDS_WITH`
		`CONTAINS`
		`EQUALS`
		`MATCH_REG_EX`

`matchValue`*	string minLength: 1	
`shouldNot`	boolean default: `false`	

NodePropertyVariableMatcher

Property Name	Type	Description
`selectedNodeScope`*	string	Enum `NODE_PROPERTY_VARIABLE_MATCHER`

propVarPairs*	minItems: 1	**propVarPairs** **Property Name** / **Type** / **Description** propertyType* — string — NodePropertyTypeEnum: META, ENV, JVM name* — string, minLength: 1 value* — string, minLength: 1

Errors

Specific errors are considered for email digest.

Property Name	Type	Description
entityType*	string	Enum ERRORS
selectedErrors*		**SelectedErrors** **Property Name** / **Type** / **Description** errorScope* — string — Enums: ALL_ERRORS, SPECIFIC_ERRORS, ERRORS_OF_SPECIFIC_TIERS, ERRORS_MATCHING_PATTERN

SelectedErrors

Property Name	Type	Description
errorScope*	string	Enums ALL_ERRORS SPECIFIC_ERRORS ERRORS_OF_SPECIFIC_TIERS ERRORS_MATCHING_PATTERN

AllErrors

Property Name	Type	Description
errorScope*	string	Enum ALL_ERRORS

SpecificErrors

Property Name	Type	Description
`errorScope`*	string	Enum `SPECIFIC_ERRORS`
`errors`*	string minItems: 1 Example: `NullPointerException`	

ErrorsOfSpecificTiers

Property Name	Type	Description
`errorScope`*	string	Enum `ERRORS_OF_SPECIFIC_TIERS`
`specificTiers`*	string minItems: 1	

ErrorsMatchingPattern

Property Name	Type	Description
`errorScope`*	string	Enum ERRORS_MATCHING_PATTERN
`patternMatcher`*		**EntityMatchingPattern** See table below

Property Name	Type	Description

`matchTo`*	string	Enums
		`STARTS_WITH`
		`ENDS_WITH`
		`CONTAINS`
		`EQUALS`
		`MATCH_REG_EX`

`matchValue`*	string minLength: 1	
`shouldNot`	boolean default: `false`	

ServiceEndpoints

Specific service endpoints are considered for email digest.

Property Name	Type	Description
`entityType`*	string	Enum `SERVICE_ENDPOINTS`

`selectedServiceEndpoints`*		**SelectedServiceEndpoints** **Property Name** / **Type** / **Description** `serviceEndpointScope`* — string — Enums: `ALL_SERVICE_ENDPOINTS`, `SPECIFIC_SERVICE_ENDPOINTS`, `SERVICE_ENDPOINTS_IN_SPECIFIC_TIERS`, `SERVICE_ENDPOINTS_MATCHING_PATTERN`

SelectedServiceEndpoints

Property Name	Type	Description

`serviceEndpointScope`*	string	Enums
		`ALL_SERVICE_ENDPOINTS`
		`SPECIFIC_SERVICE_ENDPOINTS`
		`SERVICE_ENDPOINTS_IN_SPECIFIC_TIERS`
		`SERVICE_ENDPOINTS_MATCHING_PATTERN`

AllServiceEndpoints

Property Name	Type	Description
`serviceEndpointScope`*	string	Enums `ALL_SERVICE_ENDPOINTS`

SpecificServiceEndpoints

Property Name	Type	Description
`serviceEndpointScope`*	string	Enum `SPECIFIC_SERVICE_ENDPOINTS`
`serviceEndpoints`*	string minItems: 1	

ServiceEndpointsInSpecificTiers

Property Name	Type	Description
`serviceEndpointScope`*	string	Enum `SERVICE_ENDPOINTS_IN_SPECIFIC_TIERS`
`specificTiers`*	string minItems: 1	

ServiceEndpointsMatchingPattern

Property Name	Type	Description

`serviceEndpointScope`*	string	Enum `SERVICE_ENDPOINTS_MATCHING_PATTERN`

`patternMatcher`*		**EntityMatchingPattern**
		Property Name / **Type** / **Description**
		`matchTo`* — string — Enums: `STARTS_WITH`, `ENDS_WITH`, `CONTAINS`, `EQUALS`, `MATCH_REG_EX`
		`matchValue`* — string, minLength: 1
		`shouldNot` — boolean, default: `false`

InformationPoints

Specific information points are considered for email digest.

Property Name	Type	Description
`entityType`*	string	Enum INFORMATION_POINTS
`selectedInformationPoints`*		**SelectedInformationPoints** **Property Name** / **Type** / **Description** `informationPointScope`* — string — Enums: `ALL_INFORMATION_POINTS`, `SPECIFIC_INFORMATION_POINTS`, `INFORMATION_POINTS_MATCHING_PATTERN`

SelectedInformationPoints

Property Name	Type	Description

`informationPointScope`*	string	Enums
		`ALL_INFORMATION_POINTS`
		`SPECIFIC_INFORMATION_POINTS`
		`INFORMATION_POINTS_MATCHING_PATTERN`

AllInformationPoints

Property Name	Type	Description

`informationPointScope`*	string	Enums `ALL_INFORMATION_POINTS`

SpecificInformationPoints

Property Name	Type	Description
`informationPointScope`*	string	Enum `SPECIFIC_INFORMATION_POINTS`
`informationPoints`*	string minItems: 1	

InformationPointsMatchingPattern

Property Name	Type	Description
`informationPointScope`*	string	Enum `INFORMATION_POINTS_MATCHING_PATTERN`
`patternMatcher`*		**EntityMatchingPattern** See table below

EntityMatchingPattern

Property Name	Type	Description

`matchTo`*	string	Enums
		`STARTS_WITH`
		`ENDS_WITH`
		`CONTAINS`
		`EQUALS`
		`MATCH_REG_EX`

`matchValue`*	string minLength: 1	
`shouldNot`	boolean default: `false`	

DatabasesInApplication

Specific databases associated with the application are considered for email digest.

Property Name	Type	Description
`entityType`*	string	Enums `DATABASES_IN_APPLICATION`

selectedApplicationDatabases*		**SelectedApplicationDatabases** **Property Name** / **Type** / **Description** applicationDatabaseScope — string — Enums: ALL_APPLICATION_DATABASES, SPECIFIC_APPLICATION_DATABASES, APPLICATION_DATABASES_MATCHING_PATTERN

SelectedApplicationDatabases

Property Name	Type	Description
applicationDatabaseScope*	string	Enums ALL_APPLICATION_DATABASES SPECIFIC_APPLICATION_DATABASES APPLICATION_DATABASES_MATCHING_PATTERN

AllApplicationDatabases

Property Name	Type	Description
applicationDatabaseScope*	string	Enum ALL_APPLICATION_DATABASES

SpecificApplicationDatabases

Property Name	Type	Description
applicationDatabaseScope*	string	Enum SPECIFIC_APPLICATION_DATABASES
applicationDatabases*	string minItems: 1	

ApplicationDatabasesMatchingPattern

Property Name	Type	Description
applicationDatabaseScope*	string	Enum APPLICATION_DATABASES_MATCHING_PATTERN

patternMatcher*		**EntityMatchingPattern** **Property Name** / **Type** / **Description** matchTo* — string — Enums: STARTS_WITH, ENDS_WITH, CONTAINS, EQUALS, MATCH_REG_EX matchValue* — string, minLength: 1 shouldNot — boolean, default: false

ServersInApplication

Specific servers associated with the application are considered for the email digest.

Property Name	Type	Description
entityType*	string	Enum SERVERS_IN_APPLICATION
selectedServers*	string	**ApplicationSelectedServers** **Property Name** / **Type** / **Description** serversScope* — string — Enums: ALL_SERVERS_IN_APPLICATION, SPECIFIC_SERVERS_IN_APPLICATION, ALL_SERVERS_IN_SPECIFIC_TIERS

ApplicationSelectedServers

Property Name	Type	Description
serversScope*	string	Enums ALL_SERVERS_IN_APPLICATION SPECIFIC_SERVERS_IN_APPLICATION ALL_SERVERS_IN_SPECIFIC_TIERS

AllServersInApplication

Property Name	Type	Description
serversScope*	string	Enum ALL_SERVERS_IN_APPLICATION

SpecificServersInApplication

Property Name	Type	Description
`serversScope`*	string	Enum `SPECIFIC_SERVERS_IN_APPLICATION`
`specificServers`*	string minLength: 1 minItems: 1	

AllServersInSpecificTiers

Property Name	Type	Description
`serversScope`*	string	Enum `ALL_SERVERS_IN_SPECIFIC_TIERS`
`specificTiers`*	string minItems: 1	

Events

Different types of events considered for an email digest.

Property Name	Type	Description
`healthRuleEvents`		**HealthRuleEvent** Events associated with health rules that trigger the email digest.

HealthRuleEvent

Events associated with health rules that trigger the email digest.

Property Name	Type	Description

`healthRuleEventTypes`*	string
	minItems: 1
		HealthRuleEventType
		Enums
		`HEALTH_RULE_CONTINUES_CRITICAL`
		`HEALTH_RULE_OPEN_CRITICAL`
		`HEALTH_RULE_OPEN_WARNING`
		`HEALTH_RULE_UPGRADED`
		`HEALTH_RULE_DOWNGRADED`
		`HEALTH_RULE_CONTINUES_WARNING`
		`HEALTH_RULE_CLOSE_WARNING`
		`HEALTH_RULE_CLOSE_CRITICAL`
		`HEALTH_RULE_CANCELED_WARNING`
		`HEALTH_RULE_CANCELED_CRITICAL`

healthRuleScope*	string	Events associated with specific health rules or all health rules that trigger the email digest. HealthRuleScopeType

HealthRuleScopeType

Property Name	Type	Description
`healthRuleScopeType`	string	`ALL_HEALTH_RULES` `SPECIFIC_HEALTH_RULES`

otherEvents	string	**OtherEventType** Enums CLR_CRASH APPLICATION_CRASH DEADLOCK RESOURCE_POOL_LIMIT APPLICATION_DEPLOYMENT APP_SERVER_RESTART APPLICATION_CONFIG_CHANGE AGENT_CONFIGURATION_ERROR APPLICATION_DISCOVERED TIER_DISCOVERED NODE_DISCOVERED MACHINE_DISCOVERED BT_DISCOVERED SERVICE_ENDPOINT_DISCOVERED BACKEND_DISCOVERED EUM_CLOUD_SYNTHETIC_HEALTHY_EVENT EUM_CLOUD_SYNTHETIC_WARNING_EVENT EUM_CLOUD_SYNTHETIC_CONFIRMED_WARNING_EVENT EUM_CLOUD_SYNTHETIC_ONGOING_WARNING_EVENT EUM_CLOUD_SYNTHETIC_ERROR_EVENT EUM_CLOUD_SYNTHETIC_CONFIRMED_ERROR_EVENT EUM_CLOUD_SYNTHETIC_ONGOING_ERROR_EVENT EUM_CLOUD_SYNTHETIC_PERF_HEALTHY_EVENT EUM_CLOUD_SYNTHETIC_PERF_WARNING_EVENT EUM_CLOUD_SYNTHETIC_PERF_CONFIRMED_WARNING_EVENT EUM_CLOUD_SYNTHETIC_PERF_ONGOING_WARNING_EVENT EUM_CLOUD_SYNTHETIC_PERF_CRITICAL_EVENT EUM_CLOUD_SYNTHETIC_PERF_CONFIRMED_CRITICAL_EVENT EUM_CLOUD_SYNTHETIC_PERF_ONGOING_CRITICAL_EVENT MOBILE_NEW_CRASH_EVENT SLOW VERY_SLOW STALL ERROR

<table>
<tr><td>anomalyEvents</td><td>minItems: 1</td><td>Events triggered due to anomaly detection.
AnomalyEventType
Enums
ANOMALY_OPEN_WARNING
ANOMALY_OPEN_CRITICAL
ANOMALY_UPGRADED
ANOMALY_DOWNGRADED
ANOMALY_CLOSE_WARNING
ANOMALY_CLOSE_CRITICAL
ANOMALY_CANCELED_WARNING
ANOMALY_CANCELED_CRITICAL</td></tr>
<tr><td>customEvents</td><td></td><td>Custom-defined events that trigger the email digest.
CustomEvent
<table>
<tr><th>Property Name</th><th>Type</th><th>Description</th></tr>
<tr><td>eventName*</td><td>string</td><td></td></tr>
<tr><td>PropertyMatchCriteria</td><td>string
default: ANY</td><td>Enums
ANY
ALL</td></tr>
<tr><td>keyValuePairArray</td><td></td><td>KeyValuePair
<table>
<tr><th>Property Name</th><th>Type</th></tr>
<tr><td>Key*</td><td>string</td></tr>
<tr><td>Value*</td><td>string</td></tr>
</table></td></tr>
</table></td></tr>
</table>

CustomEvent

Details of custom-defined event considered for an email digest.

Property Name	Type	Description
eventName*	string	
PropertyMatchCriteria	string default: ANY	Enums ANY ALL
keyValuePairArray		KeyValuePair

Property Name	Type
Key*	string
Value*	string

HealthRuleEvent

Events associated with health rules considered for an email digest.

Property Name	Type	Description

`healthRuleEventTypes`*	string
	minItems: 1
		HealthRuleEventType
		Enums
		`HEALTH_RULE_CONTINUES_CRITICAL`
		`HEALTH_RULE_OPEN_CRITICAL`
		`HEALTH_RULE_OPEN_WARNING`
		`HEALTH_RULE_UPGRADED`
		`HEALTH_RULE_DOWNGRADED`
		`HEALTH_RULE_CONTINUES_WARNING`
		`HEALTH_RULE_CLOSE_WARNING`
		`HEALTH_RULE_CLOSE_CRITICAL`
		`HEALTH_RULE_CANCELED_WARNING`
		`HEALTH_RULE_CANCELED_CRITICAL`

`healthRuleScope`*	string	Events associated with specific health rules or all health rules that trigger the email digest. healthRuleScopeType **Property Name** / **Type** / **Description** `healthRuleScopeType`* — string — Enums: `ALL_HEALTH_RULES`, `SPECIFIC_HEALTH_RULES`

HealthRuleScope

Events associated with specific health rules or all health rules considered for an email digest.

Property Name	Type	Description

`healthRuleScopeType`*	string	Enums
		`ALL_HEALTH_RULES`
		`SPECIFIC_HEALTH_RULES`

AllHealthRules

Events associated with all health rules within an application considered for an email digest.

Property Name	Type	Description
`healthRuleScopeType`*	string	Enum `ALL_HEALTH_RULES`

SpecificHealthRules

Events associated with specific health rules within an application considered for an email digest.

Property Name	Type	Description

`healthRuleScopeType`*	string	Enum `SPECIFIC_HEALTH_RULES`
`healthRules`*	string minItems: 1	

HealthRuleEventTypes

Property Name	Type	Description

`healthRuleEventTypes`*	string	Enums
		`HEALTH_RULE_CONTINUES_CRITICAL`
		`HEALTH_RULE_OPEN_CRITICAL`
		`HEALTH_RULE_OPEN_WARNING`
		`HEALTH_RULE_UPGRADED`
		`HEALTH_RULE_DOWNGRADED`
		`HEALTH_RULE_CONTINUES_WARNING`
		`HEALTH_RULE_CLOSE_WARNING`
		`HEALTH_RULE_CLOSE_CRITICAL`
		`HEALTH_RULE_CANCELED_WARNING`
		`HEALTH_RULE_CANCELED_CRITICAL`

OtherEvents

Property Name	Type	Description

otherEvents	string	**OtherEventType** Enums CLR_CRASH APPLICATION_CRASH DEADLOCK RESOURCE_POOL_LIMIT APPLICATION_DEPLOYMENT APP_SERVER_RESTART APPLICATION_CONFIG_CHANGE AGENT_CONFIGURATION_ERROR APPLICATION_DISCOVERED TIER_DISCOVERED NODE_DISCOVERED MACHINE_DISCOVERED BT_DISCOVERED SERVICE_ENDPOINT_DISCOVERED BACKEND_DISCOVERED EUM_CLOUD_SYNTHETIC_HEALTHY_EVENT EUM_CLOUD_SYNTHETIC_WARNING_EVENT EUM_CLOUD_SYNTHETIC_CONFIRMED_WARNING_EVENT EUM_CLOUD_SYNTHETIC_ONGOING_WARNING_EVENT EUM_CLOUD_SYNTHETIC_ERROR_EVENT EUM_CLOUD_SYNTHETIC_CONFIRMED_ERROR_EVENT EUM_CLOUD_SYNTHETIC_ONGOING_ERROR_EVENT EUM_CLOUD_SYNTHETIC_PERF_HEALTHY_EVENT EUM_CLOUD_SYNTHETIC_PERF_WARNING_EVENT EUM_CLOUD_SYNTHETIC_PERF_CONFIRMED_WARNING_EVENT EUM_CLOUD_SYNTHETIC_PERF_ONGOING_WARNING_EVENT EUM_CLOUD_SYNTHETIC_PERF_CRITICAL_EVENT EUM_CLOUD_SYNTHETIC_PERF_CONFIRMED_CRITICAL_EVENT EUM_CLOUD_SYNTHETIC_PERF_ONGOING_CRITICAL_EVENT MOBILE_NEW_CRASH_EVENT SLOW VERY_SLOW STALL ERROR

AnomalyEvents

Events generated due to anomaly detection considered for an email digest.

Property Name	Type	Description

anomalyEvents	string minItems: 1	**AnomalyEventType** Enums ANOMALY_OPEN_WARNING ANOMALY_OPEN_CRITICAL ANOMALY_UPGRADED ANOMALY_DOWNGRADED ANOMALY_CLOSE_WARNING ANOMALY_CLOSE_CRITICAL ANOMALY_CANCELED_WARNING ANOMALY_CANCELED_CRITICAL

Action

A list of actions that are taken in response to an event.

Property Name	Type	Description
actionName*	string	
actionType*	string	**ActionType** Enums SMS EMAIL CUSTOM_EMAIL THREAD_DUMP HTTP_REQUEST CUSTOM RUN_SCRIPT_ON_NODES DIAGNOSTIC_BUSINESS_TRANSACTIONS CREATE_UPDATE_JIRA

SimpleActionType

A simple action that is taken when the policy is triggered.

Property Name	Type	Description
actionName*	string	

actionType*	string	**ActionType** Enums SMS EMAIL CUSTOM_EMAIL THREAD_DUMP HTTP_REQUEST CUSTOM RUN_SCRIPT_ON_NODES DIAGNOSTIC_BUSINESS_TRANSACTIONS CREATE_UPDATE_JIRA

EmailActionType

An email is sent when the policy is triggered.

Property Name	Type	Description
actionName*	string	
actionType*	string	**ActionType** Enums SMS EMAIL CUSTOM_EMAIL THREAD_DUMP HTTP_REQUEST CUSTOM RUN_SCRIPT_ON_NODES DIAGNOSTIC_BUSINESS_TRANSACTIONS CREATE_UPDATE_JIRA
notes	string	

ActionOnSpecifiedEntities

A simple action that is taken for specific entities when the policy is triggered.

Property Name	Type	Description
actionName*	string	

`actionType`*	string	**ActionType**
		Enums
		`SMS`
		`EMAIL`
		`CUSTOM_EMAIL`
		`THREAD_DUMP`
		`HTTP_REQUEST`
		`CUSTOM`
		`RUN_SCRIPT_ON_NODES`
		`DIAGNOSTIC_BUSINESS_TRANSACTIONS`
		`CREATE_UPDATE_JIRA`

`specifiedEntityActionDetails`*	string	**SpecifiedEntityActionDetails**
		Property Name / **Type** / **Description**
		`specifiedEntityActionScope` / string / **SpecifiedEntityActionScope**
		Enums
		`PERCENTAGE`
		`ABSOLUTE`
		`SPECIFIC_NODES`

SpecifiedEntityActionDetails

Property Name	Type	Description

`specifiedEntityActionScope`	string	**SpecifiedEntityActionScope**
		Enums
		`PERCENTAGE`
		`ABSOLUTE`
		`SPECIFIC_NODES`

ActionOnPercentageEntities

The scope of entities on which the action is performed is set to `percentage`.

Property Name	Type	Description
`specifiedEntityActionScope`	string	**SpecifiedEntityActionScope** Enum `PERCENTAGE`
`value`*	integer	

ActionOnAbsoluteEntities

The scope of entities on which the action is performed is set to `absolute`.

Property Name	Type	Description

specifiedEntityActionScope	string	**SpecifiedEntityActionScope** Enum ABSOLUTE
value*	integer	

ActionOnSpecificNodes

A list of nodes on which the action is performed.

Property Name	Type	Description
specifiedEntityActionScope	string	**SpecifiedEntityActionScope** Enum SPECIFIC_NODES
nodes*	string minItems: 1	

EmailDigestSummaryArray

Property Name	Type
id*	integer
name*	string minLength: 1
enabled*	boolean

KeyValuePair

Property Name	Type
key*	string
value*	string

EmailDigestSummary

Property Name	Type
id*	integer
name*	string minLength: 1
enabled*	boolean

EmailDigestConfiguration

Property Name	Type
enabled*	boolean
EmailDigestName	string

`frequency`	integer minimum: 1 maximum: 168

EntityMatchingPattern

Entities that match the specified pattern.

Property Name	Type	Description

`matchTo`*	string	Enums
		`STARTS_WITH`
		`ENDS_WITH`
		`CONTAINS`
		`EQUALS`
		`MATCH_REG_EX`

`matchValue`*	string minLength: 1	
`shouldNot`	boolean default: `false`	

Enums

PropertyMatchCriteria

Property Name	Type	Description

`propertyMatchCriteria`	string
	default: ANY
		Enums
		`ANY`
		`ALL`

EntityMatchingPatternEnum

Property Name	Type	Description

`EntityMatchingPatternEnum`	string	Enums
		`STARTS_WITH`
		`ENDS_WITH`
		`CONTAINS`
		`EQUALS`
		`MATCH_REG_EX`

PageScopeEnum

Property Name	Type	Description

`PageScopeEnum`	string	Enums
		`ALL_PAGES`
		`SPECIFIC_PAGES`
		`PAGES_MATCHING_PATTERN`

VirtualPageScopeEnum

Property Name	Type	Description
PageScopeEnum	string	Enums ALL_VIRTUAL_PAGES SPECIFIC_VIRTUAL_PAGES VIRTUAL_PAGES_MATCHING_PATTERN

AjaxRequestScopeEnum

Property Name	Type	Description
AjaxRequestScopeEnum	string	Enums ALL_AJAX_REQUESTS SPECIFIC_AJAX_REQUESTS AJAX_REQUESTS_MATCHING_PATTERN

AjaxRequestScopeEnum

Property Name	Type	Description
AjaxRequestScopeEnum	string	Enums ALL_AJAX_REQUESTS SPECIFIC_AJAX_REQUESTS AJAX_REQUESTS_MATCHING_PATTERN

SyntheticJobScopeEnum

Property Name	Type	Description
SyntheticJobScopeEnum	string	Enums ALL_SYNTHETIC_JOBS SPECIFIC_SYNTHETIC_JOBS SYNTHETIC_JOBS_MATCHING_PATTERN

IFrameScopeEnum

Property Name	Type	Description
IFrameScopeEnum	string	Enums ALL_IFRAMES SPECIFIC_IFRAMES IFRAMES_MATCHING_PATTERN

BusinessTransactionScopeEnum

Property Name	Type	Description

businessTransactionScope	string	Enums ALL_BUSINESS_TRANSACTIONS SPECIFIC_BUSINESS_TRANSACTIONS BUSINESS_TRANSACTIONS_IN_SPECIFIC_TIERS BUSINESS_TRANSACTIONS_MATCHING_PATTERN

MobileAppsScopeEnum

Property Name	Type	Description
MobileAppsScopeEnum	string	Enums ALL_MOBILE_APPS SPECIFIC_MOBILE_APPS MOBILE_APPS_MATCHING_PATTERN

MobileNetworkRequestsScopeEnum

Property Name	Type	Description
MobileAppsScopeEnum	string	Enums ALL_MOBILE_NETWORK_REQUESTS SPECIFIC_MOBILE_NETWORK_REQUESTS SPECIFIC_MOBILE_APPS_NETWORK_REQUESTS MOBILE_NETWORK_REQUESTS_MATCHING_PATTERN

TierOrNodeScopeEnum

Property Name	Type	Description
TierOrNodeScope	string	Enums TIER_SELECTED_ENTITIES NODE_SELECTED_ENTITIES

SelectedTierScopeEnum

Property Name	Type	Description
SelectedTierScope	string	Enums ALL_TIERS SPECIFIC_TIER

TypeOfNodeEnum

Property Name	Type	Description

typeofNode	string	Enums ALL_NODES JAVA_NODES DOT_NET_NODES PHP_NODES

SelectedNodesScopeEnum

Property Name	Type	Description
selectedNodeScope	string	Enums ALL_NODES SPECIFIC_NODES NODES_OF_SPECIFIC_TIERS NODES_MATCHING_PATTERN NODE_PROPERTY_VARIABLE_MATCHER

NodePropertyTypeEnum

Property Name	Type	Description
NodePropertyTypeEnum	string	Enums META ENV JVM

ErrorScopeEnum

Property Name	Type	Description
ErrorScopeEnum	string	Enums ALL_ERRORS SPECIFIC_ERRORS ERRORS_OF_SPECIFIC_TIERS ERRORS_MATCHING_PATTERN

ServiceEndpointScopeEnum

Property Name	Type	Description
ServiceEndpointScopeEnum	string	Enums ALL_SERVICE_ENDPOINTS SPECIFIC_SERVICE_ENDPOINTS SERVICE_ENDPOINTS_IN_SPECIFIC_TIERS SERVICE_ENDPOINTS_MATCHING_PATTERN

InformationPointScopeEnum

Property Name	Type	Description
InformationPointScopeEnum	string	ALL_INFORMATION_POINTS SPECIFIC_INFORMATION_POINTS INFORMATION_POINTS_MATCHING_PATTERN

DatabaseTypeEnum

Property Name	Type	Description
DatabaseTypeEnum	string	ALL_DATABASE_TYPES COUCHBASE DB2 MONGO_DB MICROSOFT_SQL_SERVER MYSQL ORACLE POSTGRE_SQL AZURE_SQL SYBASE

ApplicationDatabaseScopeEnum

Property Name	Type	Description
ApplicationDatabaseScopeEnum	string	ALL_APPLICATION_DATABASES SPECIFIC_APPLICATION_DATABASES APPLICATION_DATABASES_MATCHING_PATTERN

ServersScopeEnum

Property Name	Type	Description
ServersScopeEnum	string	ALL_SERVERS_IN_APPLICATION SPECIFIC_SERVERS_IN_APPLICATION ALL_SERVERS_IN_SPECIFIC_TIERS

SpecifiedEntityActionScopeEnum

Property Name	Type	Description
SpecifiedEntityActionScopeEnum	string	PERCENTAGE ABSOLUTE SPECIFIC_NODES

AnomalyEventType

Property Name	Type	Description

AnomalyEventType	string	Enums ANOMALY_OPEN_WARNING ANOMALY_OPEN_CRITICAL ANOMALY_UPGRADED ANOMALY_DOWNGRADED ANOMALY_CLOSE_WARNING ANOMALY_CLOSE_CRITICAL ANOMALY_CANCELED_WARNING ANOMALY_CANCELED_CRITICAL

HealthRuleEventTypeEnum

Property Name	Type	Description
HealthRuleEventTypeEnum	string	Enums HEALTH_RULE_CONTINUES_CRITICAL HEALTH_RULE_OPEN_CRITICAL HEALTH_RULE_OPEN_WARNING HEALTH_RULE_UPGRADED HEALTH_RULE_DOWNGRADED HEALTH_RULE_CONTINUES_WARNING HEALTH_RULE_CLOSE_WARNING HEALTH_RULE_CLOSE_CRITICAL HEALTH_RULE_CANCELED_WARNING HEALTH_RULE_CANCELED_CRITICAL

HealthRuleScopeType

Property Name	Type	Description
HealthRuleScopeType	string	ALL_HEALTH_RULES SPECIFIC_HEALTH_RULES

OtherEventType

Property Name	Type	Description

OtherEventType	string	Enums CLR_CRASH APPLICATION_CRASH DEADLOCK RESOURCE_POOL_LIMIT APPLICATION_DEPLOYMENT APP_SERVER_RESTART APPLICATION_CONFIG_CHANGE AGENT_CONFIGURATION_ERROR APPLICATION_DISCOVERED TIER_DISCOVERED NODE_DISCOVERED MACHINE_DISCOVERED BT_DISCOVERED SERVICE_ENDPOINT_DISCOVERED BACKEND_DISCOVERED EUM_CLOUD_SYNTHETIC_HEALTHY_EVENT EUM_CLOUD_SYNTHETIC_WARNING_EVENT EUM_CLOUD_SYNTHETIC_CONFIRMED_WARNING_EVENT EUM_CLOUD_SYNTHETIC_ONGOING_WARNING_EVENT EUM_CLOUD_SYNTHETIC_ERROR_EVENT EUM_CLOUD_SYNTHETIC_CONFIRMED_ERROR_EVENT EUM_CLOUD_SYNTHETIC_ONGOING_ERROR_EVENT EUM_CLOUD_SYNTHETIC_PERF_HEALTHY_EVENT EUM_CLOUD_SYNTHETIC_PERF_WARNING_EVENT EUM_CLOUD_SYNTHETIC_PERF_CONFIRMED_WARNING_EVENT EUM_CLOUD_SYNTHETIC_PERF_ONGOING_WARNING_EVENT EUM_CLOUD_SYNTHETIC_PERF_CRITICAL_EVENT EUM_CLOUD_SYNTHETIC_PERF_CONFIRMED_CRITICAL_EVENT EUM_CLOUD_SYNTHETIC_PERF_ONGOING_CRITICAL_EVENT MOBILE_NEW_CRASH_EVENT, SLOW, VERY_SLOW, STALL ERROR

SelectedEntityType

Property Name	Type	Description
SelectedEntityType	string	ANY_ENTITY SPECIFIC_ENTITIES

EntityType

Property Name	Type	Description

EntityType	string	Enums BUSINESS_TRANSACTION TIER_NODE ERRORS SERVICE_ENDPOINTS INFORMATION_POINTS DATABASES_IN_APPLICATION SERVERS_IN_APPLICATION

ActionTypeEnum

Property Name	Type	Description
ActionTypeEnum	string	SMS EMAIL CUSTOM_EMAIL THREAD_DUMP HTTP_REQUEST CUSTOM RUN_SCRIPT_ON_NODES DIAGNOSTIC_BUSINESS_TRANSACTIONS CREATE_UPDATE_JIRA

ErrorResponse

Property Name	Type
statusCode	integer
message	string

***This property is required.**

Download Examples

Download Appdynamics Email Digest Examples.zip to get a set of examples that help you configure an email digest,

Download SWAGGER YAML file

Download the Swagger email_digests_openapi.yml file.

Action Suppression API

This page describes the Action Suppression API methods you can use to temporarily suspend the automatic trigger of actions and alerts by a policy in response to an event. This API is useful when performing maintenance activities or troubleshooting a component.

> **Note**
>
> Syntax validation of the JSON payload is done when creating action suppression.

Create a New Action Suppression

Creates a new action suppression with the specified JSON payload. See Property Details.

Resource URL

```
POST <controller_url>/controller/alerting/rest/v1/applications/<application_id>/action-suppressions
```

Request/Response Format

JSON

Example

Retrieve a List of Action Suppressions Configured for a Given Application

Returns all the action suppressions that are configured for a given application. The required fields are action suppression IDs and names. Details such as `timezone`, `startTime`, `endTime` and `recurringSchedule` of configured action suppressions are returned. See Property Details.

Resource URL

```
GET <controller_url>/controller/alerting/rest/v1/applications/<application_id>/action-suppressions
```

Response Format

JSON

Example Response


```
[
 {
        "id": 11,
        "name": "TestAS1",
        "timezone": "Asia/Kolkata",
        "startTime": "2020-08-19T12:06:00",
        "endTime": "2020-08-19T13:06:00"
 }
 {
        "id": 12,
        "name": "TestAS2",
        "timezone": "Asia/Kolkata",
        "startTime": "2020-08-19T12:27:00",
        "endTime": "2020-08-19T13:27:00"
 }
 {
        "id": 13,
        "name": "TestAS3",
        "timezone": "Asia/Kolkata",
        "startTime": "2020-08-19T12:08:00",
        "endTime": "2020-08-19T13:08:00"
 }
]
```

Retrieve the Details of an Action Suppression

Returns JSON representation of action suppression for the given action suppression ID. See Property Details.

Resource URL

```
GET <controller_url>/controller/alerting/rest/v1/applications/<application_id>/action-suppressions/{action-suppression-id}
```

Response Format

JSON

Example Response

```
{
  "id": 9,
  "name": "TestAS",
  "disableAgentReporting": false,
  "suppressionScheduleType": "ONE_TIME",
  "timezone": "Asia/Kolkata",
  "startTime": "2020-08-19T11:43:00",
  "endTime": "2020-08-19T12:43:00",
  "recurringSchedule": null,
  "affects": {
    "affectedInfoType": "APPLICATION"
  },
  "healthRuleScope": null
}
```

Update an Action Suppression

Updates an existing action suppression with a specified JSON payload. See Property Details.

Resource URL

```
PUT <controller_url>/controller/alerting/rest/v1/applications/<application_id>/action-suppressions/{action-suppression-id}
```

Request/Response Format

JSON

Example

Delete an Action Suppression

Deletes an action suppression with the specified ID. See Property Details.

Resource URL

```
DELETE <controller_url>/controller/alerting/rest/v1/applications/<application_id>/action-suppressions/{action-suppression-id}
```

Retrieve the Details of an Action Suppression by Name

Returns JSON representation of action suppression for the given action suppression name. See Property Details.

Resource URL

```
GET <controller_url>/controller/alerting/rest/v1/applications/<application_id>/action-suppressions/action-suppression-by-name/?name=<ActionSuppressionName>
```

Replace `<ActionSuppressionName>` with a name you specified for the action suppression. For example, `ACTION_SUP_15`.

Response Format

JSON

Example Response

```
{
    "id": 9,
    "name": "TestAS1",
    "timezone": "Asia/Kolkata",
    "startTime": "2020-08-19T12:05:00",
    "endTime": "2020-08-19T13:05:00",
    "recurringSchedule": null
  }
```

Response Codes

Code	Description
`200`	Fetched successfully
`201`	Created successfully
`204`	Deleted successfully
`400`	Bad request
`401`	Unauthorized
`403`	Forbidden
`404`	Resource not found
`409`	Already exists

Property Details

Action Suppression

Payload details for action suppression.

Property Name	Type	Description and Valid Values
`id`	integer	This is auto-generated by the system and returned in the response. It is a `readOnly` value.
`name`*	string minLength: 1 maxLength: 100	The name of action suppression.
`disableAgentReporting`	boolean default: `false`	If enabled, the agents defined in the scope do not report any metric data during the specified time frame.
 You cannot define a recurring action suppression if you set `disableAgentReporting` to `true`.		
`suppressionScheduleType`	string default: ONE_TIME	The available scheduling options for action suppression. **Enums**

<table>
<tr><td></td><td></td><td>ONE_TIME

RECURRING</td></tr>
<tr><td>timezone</td><td>string</td><td>The timezone ID. The default time zone is the controller timezone.</td></tr>
<tr><td>startTime</td><td>string

format: yyyy-MM-ddTHH:mm:ss</td><td>The time at which the action suppression is initiated. Specify startTime only if suppressionScheduleType is set to ONE_TIME. If you do not specify the startTime, action suppression starts from the current time. The expected format is yyyy-MM-ddTHH:mm:ss conforming to rfc3339.</td></tr>
<tr><td>endTime</td><td>string

format: yyyy-MM-ddTHH:mm:ss</td><td>The time at which the ongoing action suppression ends. Specify startTime only if suppressionScheduleType is set to ONE_TIME. If not specified, action suppression ends at 60 minutes from the current time. The expected format is yyyy-MM-ddTHH:mm:ss conforming to rfc3339.</td></tr>
<tr><td>recurringSchedule</td><td>string</td><td>The recurring schedule details to initiate an action suppression.

<table><tr><th>Property Name</th><th>Type</th><th>Description</th></tr><tr><td>scheduleFrequency*</td><td>string</td><td>The frequency of invoking action suppression.

Enums

DAILY

WEEKLY

MONTHLY_SPECIFIC_DATE

MONTHLY_SPECIFIC_DAY</td></tr></table></td></tr>
<tr><td>affects*</td><td></td><td>Describes entities affected by action suppression. For example, applications, business transactions, servers, or databases.

<table><tr><th>Property Name</th><th>Type</th><th>Description</th></tr><tr><td>affectedInfoType*</td><td>string</td><td>The affected entity type.

Enums

APPLICATION

BUSINESS_TRANSACTIONS

TIERS_NODES

SERVERS

DATABASES</td></tr></table></td></tr>
<tr><td>healthRuleScope*</td><td></td><td>The scope of the health rules applicable to action suppression.

<table><tr><th>Property Name</th><th>Type</th><th>Description</th></tr><tr><td>healthRuleScopeType*</td><td>string</td><td>The health rule types affected by action suppression.

Enums

SPECIFIC_HEALTH_RULES</td></tr></table></td></tr>
</table>

Action Suppression Schedule Type

Use this property to schedule a one-time or recurring action suppression.

Property Name	Type	Description
ActionSuppressionScheduleType	string	**Enums**

		ONE_TIME RECURRING

Example for one-time action suppression

```
{
    "name": "Action Suppression With One Time Schedule",
    "disableAgentReporting": false,
    "suppressionScheduleType": "ONE_TIME",
    "timezone": "Asia/Kolkata",
    "startTime": "2020-06-18T13:33:37",
    "endTime": "2021-06-30T16:48:37",
    "recurringSchedule": null,
    "affects": {
        "affectedInfoType": "APPLICATION"
    },
    "healthRuleScope": null
}
```

Recurring Schedule Frequency Details

Use this property to define the frequency of invoking action suppression on a recurring basis.

Property Name	Type	Description

`scheduleFrequency`*	string	Describes how often action suppression is invoked.
		
		Enums
		
		`DAILY`
		
		`WEEKLY`
		
		`MONTHLY_SPECIFIC_DATE`
		
		`MONTHLY_SPECIFIC_DAY`

Example for recurring action suppression scheduled on a daily basis

```
{
    "name": "Action Suppression With Daily Schedule",
    "disableAgentReporting": false,
    "suppressionScheduleType": "RECURRING",
    "timezone": "Asia/Kolkata",
    "startTime": null,
    "endTime": null,
    "recurringSchedule": {
        "scheduleFrequency": "DAILY",
        "startTime": "01:00",
        "endTime": "01:22"
    },
    "affects": {
        "affectedInfoType": "APPLICATION"
    },
    "healthRuleScope": null
}
```

Daily Schedule - Configuration Details

Use the following properties to configure the details of an action suppression that is scheduled on a daily basis.

Property Name	Type	Description
`scheduleFrequency`*	string	Action suppression is invoked daily.

		Enum `DAILY`
`startTime`*	Time string pattern: ^([01]\d|2[0-3]):([0-5]\d)$	The time at which the action suppression is initiated. The time in 24 hour format.
`endTime`*	Time string pattern: ^([01]\d|2[0-3]):([0-5]\d)$	The time at which the ongoing action suppression ends. The time in 24 hour format.

Example

```
{
    "name": "Action Suppression With Daily Schedule",
    "disableAgentReporting": false,
    "suppressionScheduleType": "RECURRING",
    "timezone": "Asia/Kolkata",
    "startTime": null,
    "endTime": null,
    "recurringSchedule": {
        "scheduleFrequency": "DAILY",
        "startTime": "01:00",
        "endTime": "01:22"
    },
    "affects": {
        "affectedInfoType": "APPLICATION"
    },
    "healthRuleScope": null
}
```

Weekly Schedule - Configuration Details

Use the following properties to configure the details of an action suppression that is scheduled on a weekly basis.

<table>
<tr><th>Property Name</th><th>Type</th><th>Description</th></tr>
<tr><td><code>scheduleFrequency</code>*</td><td>string</td><td>Action suppression is invoked on a specific day of the week.
Enum
<code>WEEKLY</code></td></tr>
<tr><td><code>days</code>*</td><td>string
minItems: 1
maxItems: 7</td><td>The day(s) of the week to suppress actions.
<table>
<tr><th>Property Name</th><th>Type</th><th>Description</th></tr>
<tr><td>DayOfWeek</td><td>string</td><td>Enums
<code>SUNDAY</code>
<code>MONDAY</code>
<code>TUESDAY</code>
<code>WEDNESDAY</code>
<code>THURSDAY</code>
<code>FRIDAY</code>
<code>SATURDAY</code></td></tr>
</table></td></tr>
<tr><td><code>startTime</code>*</td><td>Time string
pattern: <code>^([01]\d|2[0-3]):([0-5]\d)$</code></td><td>The time at which the action suppression is initiated.
The time in 24 hour format.</td></tr>
<tr><td><code>endTime</code>*</td><td>Time string
pattern: <code>^([01]\d|2[0-3]):([0-5]\d)$</code></td><td>The time at which the ongoing action suppression ends.
The time in 24 hour format.</td></tr>
</table>

Example

```
{ "name": "Action Suppression With Weekly Schedule", "disableAgentReporting": false,
    "suppressionScheduleType": "RECURRING",
    "timezone": "Asia/Kolkata",
    "startTime": null,
    "endTime": null,
    "recurringSchedule": {
        "scheduleFrequency": "WEEKLY",
        "startTime": "06:00",
        "endTime": "18:00",
        "days": [
            "SUNDAY",
            "MONDAY"
        ]
    },
    "affects": {
        "affectedInfoType": "APPLICATION"
    },
    "healthRuleScope": null
}
```

Monthly Schedule, Specific Date - Configuration Details

Use the following properties to configure the details of an action suppression that is scheduled on a monthly basis, on a specific date.

Property Name	Type	Description
`scheduleFrequency`*	string	Action suppression is invoked on a specific date every month. **Enum** `MONTHLY_SPECIFIC_DATE`
startDate*	string pattern: `^(0[1-9]|[12][0-9]|3[01])$\|^Last Day$`	The start date of the month to suppress actions.
`startTime`*	Time string pattern: `^([01]\d|2[0-3]):([0-5]\d)$`	The time at which action suppression is initiated. The time in 24 hour format.
`endDate`*	string pattern: `^(0[1-9]|[12][0-9]|3[01])$\|^Last Day$`	The end date of the month to suppress actions.
`endTime`*	Time string pattern: `^([01]\d|2[0-3]):([0-5]\d)$`	The time at which the ongoing action suppression ends. The time in 24 hour format.

Example

```
{
    "name": "Action Suppression With Monthly Specific Date Schedule",
    "disableAgentReporting": false,
    "suppressionScheduleType": "RECURRING",
    "timezone": "Asia/Kolkata",
    "startTime": null,
    "endTime": null,
    "recurringSchedule": {
        "scheduleFrequency": "MONTHLY_SPECIFIC_DATE",
        "startDate": "01",
        "startTime": "06:00",
        "endDate": "Last Day",
        "endTime": "18:00"
    },
    "affects": {
        "affectedInfoType": "APPLICATION"
    },
```

```
    "healthRuleScope": null
}
```

Monthly Schedule, Specific Day - Configuration Details

Use the following properties to configure the details of an action suppression that is scheduled on a monthly basis, on a specific date.

Property Name	Type	Description
`scheduleFrequency`*	string	Action suppression is invoked on a specific day of the month. **Enum** `MONTHLY_SPECIFIC_DAY`
`startTime`*	Time string pattern: `^([01]\d|2[0-3]):([0-5]\d)$`	The time at which action suppression is initiated. The time in 24 hour format.
`endTime`*	Time string pattern: `^([01]\d|2[0-3]):([0-5]\d)$`	The time at which the ongoing action suppression ends. The time in 24 hour format.
`days`*	string	The day of the month to suppress actions.

`occurrence`*	string	**Enums**
		`FIRST`
		`SECOND`
		`THIRD`
		`FOURTH`
		`LAST`

`days`* — nested properties:

Property Name	Type	Description

DayOfWeek	string	**Enums**
		`SUNDAY`
		`MONDAY`
		`TUESDAY`
		`WEDNESDAY`
		`THURSDAY`
		`FRIDAY`
		`SATURDAY`

Example

```
{
    "name": "Action Suppression With Monthly Specific Day Schedule",
    "disableAgentReporting": false,
    "suppressionScheduleType": "RECURRING",
    "timezone": "Asia/Kolkata",
    "startTime": null,
    "endTime": null,
    "recurringSchedule": {
        "scheduleFrequency": "MONTHLY_SPECIFIC_DAY",
        "startTime": "06:00",
        "endTime": "18:00",
        "day": "SUNDAY",
        "occurrence": "FIRST"
    },
```

```
    "affects": {
        "affectedInfoType": "APPLICATION"
    },
    "healthRuleScope": null
}
```

Schedule Frequency Details

Use this property to define the frequency of invoking an action suppression.

Property Name	Type	Description

`scheduleFrequency`*	string	**Enums**
		`DAILY`
		`WEEKLY`
		`MONTHLY_SPECIFIC_DATE`
		`MONTHLY_SPECIFIC_DAY`

Example

```
{
   "recurringSchedule":{
      "scheduleFrequency":"WEEKLY",
      "startTime":"06:00",
      "endTime":"18:00",
      "days":[
         "SUNDAY",
         "MONDAY"
      ]
   }
```

Occurrence Details

The occurrence of the day of the month to suppress actions.

Property Name	Type	Description

`occurrence`	string	**Enums**
		`FIRST`
		`SECOND`
		`THIRD`
		`FOURTH`
		`LAST`

Example

```
{
   "recurringSchedule":{
      "scheduleFrequency":"MONTHLY_SPECIFIC_DAY",
      "startTime":"06:00",
      "endTime":"18:00",
      "day":"SUNDAY",
      "occurrence":"FIRST"
   }
}
```

Monthly Action Suppression - Day

The day of the month to suppress actions.

Property Name	Type	Description

`DayOfWeek`	string	**Enums**
		`SUNDAY`
		`MONDAY`
		`TUESDAY`
		`WEDNESDAY`
		`THURSDAY`
		`FRIDAY`
		`SATURDAY`

Example

```
{
   "recurringSchedule":{
      "scheduleFrequency":"MONTHLY_SPECIFIC_DAY",
      "startTime":"06:00",
      "endTime":"18:00",
      "day":"SUNDAY",
      "occurrence":"FIRST"
   }
}
```

Action Suppression Time Details

Use this property to define the time details to suppress an action.

Property Name	Type	Description
`Time`	string pattern: `^([01]\d|2[0-3]):([0-5]\d)$`	The time in 24 hour format.

Example

```
{
        "recurringSchedule": {
        "scheduleFrequency": "MONTHLY_SPECIFIC_DATE",
        "startDate": "01",
        "startTime": "06:00",
        "endDate": "Last Day",
        "endTime": "18:00"
    }
}
```

Timezone

Property Name	Type	Description
`Timezone`	string	Timezone Id. The default time zone is the controller timezone.

Example

```
{
    "name": "Action Suppression With One Time Schedule",
    "disableAgentReporting": false,
    "suppressionScheduleType": "ONE_TIME",
    "timezone": "Asia/Kolkata",
    "startTime": "2020-06-18T13:33:37",
    "endTime": "2021-06-30T16:48:37",
    "recurringSchedule": null,
    "affects": {
        "affectedInfoType": "APPLICATION"
    },
    "healthRuleScope": null
}
```

Entities Affected by Action Suppression

Information pertaining to entities affected by action suppression.

Property Name	Type	Description and Valid Values

`affectedInfoType`*	string	**Enums**
		`APPLICATION`
		`BUSINESS_TRANSACTIONS`
		`TIERS_NODES`
		`SERVERS`
		`DATABASES`

Example

```
{
            "affectedInfoType": "TIERS_NODES",
            "affectedEntities": {
                "tierNodeType": "NODE",
                "nodeType": "ALL_NODES",
                "affectedNodes": {
                    "affectedNodeScope": "NODES_OF_SPECIFIC_TIERS",
                    "specificTiers": ["Tier1"]
                }
            }
  }
```

Application-level Entities Affected by Action Suppression

Use to suppress actions for entities at the application-level.

Property Name	Type	Description and Valid Values
`affectedInfoType`*	string	**Enum** `APPLICATION`

Example

```
{
    "affectedInfoType": "APPLICATION"
}
```

Business Transactions Affected by Action Suppression

Use this to suppress actions for entities at the business transaction (BT) level.

Property Name	Type	Description and Valid Values
`affectedInfoType`*	string	**Enum** `BUSINESS_TRANSACTIONS`
`affectedBusinessTransactions`*		The scope of the business transactions affected by action suppression.

Property Name	Type	Description and Valid Values

`businessTransactionScope`*	string	**Enums**
		`ALL_BUSINESS_TRANSACTIONS`
		`SPECIFIC_BUSINESS_TRANSACTIONS`
		`BUSINESS_TRANSACTIONS_IN_SPECIFIC_TIERS`
		`BUSINESS_TRANSACTIONS_MATCHING_PATTERN`

Example

```
{
   "affectedInfoType":"BUSINESS_TRANSACTIONS",
   "affectedBusinessTransactions":{
         "businessTransactionScope":"BUSINESS_TRANSACTIONS_IN_SPECIFIC_TIERS",
         "specificTiers":["DefaultTier1"]
      }
}
```

Scope of Business Transactions **Affected**

Use this to suppress actions for entities at the BT level for the selected BT types.

Property Name	Type	Description and Valid Values

`businessTransactionScope`*	string	**Enums**
		`ALL_BUSINESS_TRANSACTIONS`
		`SPECIFIC_BUSINESS_TRANSACTIONS`
		`BUSINESS_TRANSACTIONS_IN_SPECIFIC_TIERS`
		`BUSINESS_TRANSACTIONS_MATCHING_PATTERN`

All Types of Business Transactions

Use this to suppress actions for entities at the BT level for all BTs.

Property Name	Type	Description and Valid Values
`businessTransactionScope`*	string	**Enum** `ALL_BUSINESS_TRANSACTIONS`

Example

```
{
                "affectedInfoType": "BUSINESS_TRANSACTIONS",
                "affectedBusinessTransactions": {
                        "businessTransactionScope": "ALL_BUSINESS_TRANSACTIONS"
                }
}
```

Specific Business Transactions

Use this to suppress actions for entities at the BT level for the specific BTs only.

Property Name	Type	Description and Valid Values
`businessTransactionScope`*	string	**Enum** `SPECIFIC_BUSINESS_TRANSACTIONS`
`businessTransactions`*	array of strings minItems: 1	Specific business transactions affected by action suppression. For example: `[CheckoutBt, LoginBt]`

Example

```
{
                "affectedInfoType": "BUSINESS_TRANSACTIONS",
                "affectedBusinessTransactions": {
                        "businessTransactionScope": "SPECIFIC_BUSINESS_TRANSACTIONS",
                        "businessTransactions": ["/BT/", "/BT/rest"]
}
```

Business Transactions Associated with Specific Tiers

Use this to suppress actions for entities at the BT level for BTs within specific tiers only.

Property Name	Type	Description and Valid Values
`businessTransactionScope`*	string	**Enum** `BUSINESS_TRANSACTIONS_IN_SPECIFIC_TIERS`
`specificTiers`*	array of strings minItems: 1	Business transactions associated with specific tiers affected by action suppression. For example: `[CheckoutTier, LoginTier]`

Example

```
{
   "affectedInfoType":"BUSINESS_TRANSACTIONS",
   "affectedBusinessTransactions":{
         "businessTransactionScope":"BUSINESS_TRANSACTIONS_IN_SPECIFIC_TIERS",
         "specificTiers":["DefaultTier1"]
      }
}
```

Business Transactions Matching a Pattern

Use this to suppress actions for entities at the BT level for BTs with properties that match a given pattern.

Property Name	Type	Description and Valid Values
`businessTransactionScope`*	string	**Enums** `BUSINESS_TRANSACTIONS_MATCHING_PATTERN`
`patternMatcher`*		**EntityMatchingPattern** `matchTo`* (string) — **Enums**: `STARTS_WITH`

		ENDS_WITH CONTAINS EQUALS MATCH_REG_EX
matchValue*	string minLength: 1	The pattern match value.
shouldNot	boolean default: false	Select this to reverse the pattern match condition.

Example

```
{
                "affectedInfoType": "BUSINESS_TRANSACTIONS",
                "affectedBusinessTransactions": {
                        "businessTransactionScope": "BUSINESS_TRANSACTIONS_MATCHING_PATTERN",
                        "patternMatcher": {
                                "matchTo": "STARTS_WITH",
                                "matchValue": "E",
                                "shouldNot": false
                        }
                }
        }
```

Tiers/Nodes Entities Affected

Use this to suppress actions for entities at the Tier/Node level.

Property Name	Type	Description and Valid Values
affectedInfoType*	string	**Enum** TIERS_NODES
affectedEntities*		**TierNodeEntities** **Property Name** / **Type** / **Description and Valid Values** tierNodeType* / string / **Enums**: TIER, NODE

Example

```
{
            "affectedInfoType": "TIERS_NODES",
            "affectedEntities": {
                "tierNodeType": "NODE",
                "nodeType": "ALL_NODES",
                "affectedNodes": {
                    "affectedNodeScope": "NODES_OF_SPECIFIC_TIERS",
                    "specificTiers": ["Tier1"]
                }
            }
  }
```

Tier or Node Entities Affected

Use this to suppress actions for entities at the tier level or node level.

Property Name	Type	Description and Valid Values

`tierNodeType`*	string	**Enums**
		`TIER`
		`NODE`

Tier-level Entities **Affected**

Property Name	Type	Description and Valid Values
`tierNodeType`*	string	**Enum** `TIER`
`affectedTiers`*		**AffectedTiers** **Property Name** / **Type** / **Description and Valid Values** `affectedTierScope`* — string — **Enums**: `ALL_TIERs`, `SPECIFIC_TIERS`

Example

```
{
                "affectedInfoType": "TIERS_NODES",
                "affectedEntities": {
                        "tierNodeType": "TIER",
                        "affectedTiers": {
                                "affectedTierScope": "SPECIFIC_TIERS",
                                "tiers": ["ECommerce-Services", "Inventory-Services"]
                                        }
                                }
}
```

Node-level Entities Affected

Property Name	Type	Description and Valid Values
`tierNodeType`*	string	**Enum** `NODE`

`nodeType`*	string	**Enums**
		`ALL_NODES`
		`JAVA_NODES`
		`DOT_NET_NODES`
		`PHP_NODES`

`affectedNodes`*		**AffectedNodes** **Property Name** / **Type** / **Description and Valid Values** `affectedNodeScope`* — string — **Enums**

				ALL_NODES SPECIFIC_NODES NODES_OF_SPECIFIC_TIERS NODES_MATCHING_PATTERN NODE_PROPERTY_VARIABLE_MATCHER

Example

```
{
            "affectedInfoType": "TIERS_NODES",
            "affectedEntities": {
                "tierNodeType": "NODE",
                "nodeType": "ALL_NODES",
                "affectedNodes": {
                    "affectedNodeScope": "NODES_OF_SPECIFIC_TIERS",
                    "specificTiers": ["Tier1"]
                }
            }
  }
```

Affected Tier Scope

Property Name	Type	Description and Valid Values
affectedTierScope*	string	**Enums** ALL_TIERs SPECIFIC_TIERS

All Tiers

Use this to suppress actions for entities at the tier level for all tiers.

Property Name	Type	Description and Valid Values
affectedTierScope*	string	**Enum** ALL_TIERS

Example

```
{
                "affectedInfoType": "TIERS_NODES",
                "affectedEntities": {
                        "tierNodeType": "TIER",
                        "affectedTiers": {
                                "affectedTierScope": "ALL_TIERS"
                                }
                        }
}
```

Specific Tiers

Use this to suppress actions for entities at the tier level for specific tiers only.

Property Name	Type	Description and Valid Values
affectedTierScope*	string	**Enum** SPECIFIC_TIERS

`tiers`*	array of strings minItems: 1	

Example

```
{
                "affectedInfoType": "TIERS_NODES",
                "affectedEntities": {
                        "tierNodeType": "TIER",
                        "affectedTiers": {
                                "affectedTierScope": "SPECIFIC_TIERS",
                                "tiers": ["ECommerce-Services", "Inventory-Services"]
                                        }
                                }
}
```

Affected Nodes

Property Name	Type	Description and Valid Values

`affectedNodeScope`*	string	**Enums**
		
		`ALL_NODES`
		
		`SPECIFIC_NODES`
		
		`NODES_OF_SPECIFIC_TIERS`
		
		`NODES_MATCHING_PATTERN`
		
		`NODE_PROPERTY_VARIABLE_MATCHER`

Example

```
{
   "affectedNodeScope": "NODES_OF_SPECIFIC_TIERS",
   "specificTiers": ["Tier1"]
 }
```

All Nodes

Use this to suppress actions for entities at the node level for all nodes.

Property Name	Type	Description and Valid Values
`affectedNodeScope`*	string	**Enum** `ALL_NODES`

Example

```
{
                "affectedInfoType": "TIERS_NODES",
                "affectedEntities": {
                        "tierNodeType": "NODE",
                        "nodeType": "ALL_NODES",
                        "affectedNodes": {
                                "affectedNodeScope": "ALL_NODES"
                        }
                }
        }
```

Specific Nodes

Use this to suppress actions for entities at the node level for specific nodes only.

Property Name	Type	Description and Valid Values
`affectedNodeScope`*	string	**Enum** `SPECIFIC_NODES`
`nodes`*	array of strings minItems: 1	A list of nodes considered as affected entities for action suppression. For example: `[Node1, Node2]`

Example

```
{
                "affectedInfoType": "TIERS_NODES",
                "affectedEntities": {
                        "tierNodeType": "NODE",
                        "nodeType": "ALL_NODES",
                        "affectedNodes": {
                                "affectedNodeScope": "SPECIFIC_NODES",
                                "nodes": ["Node1", "Node2"]
                        }
                }
        }
```

Nodes within Specific Tiers

Use this to suppress actions for entities at the node level for nodes within specific tiers only.

Property Name	Type	Description and Valid Values
`affectedNodeScope`*	string	**Enum** `NODES_OF_SPECIFIC_TIERS`
`specificTiers`*	array of strings minItems: 1	

Example

```
{
                "affectedInfoType": "TIERS_NODES",
                "affectedEntities": {
                        "tierNodeType": "NODE",
                        "nodeType": "ALL_NODES",
                        "affectedNodes": {
                                "affectedNodeScope": "NODES_OF_SPECIFIC_TIERS",
                                "specificTiers": ["ECommerce-Services", "Inventory-Services"]
                        }
                }
        }
```

Nodes that Match a Pattern

Use this to suppress actions for entities at the node level for nodes with properties that match a pattern.

Property Name	Type	Description and Valid Values
`affectedNodeScope`*	string	**Enum** `NODES_MATCHING_PATTERN`

`patternMatcher`*

EntityMatchingPattern

Property Name	Type	Description

`matchTo`*	string	**Enums**
		`STARTS_WITH`
		`ENDS_WITH`
		`CONTAINS`
		`EQUALS`
		`MATCH_REG_EX`

`matchValue`*	string minLength: 1	The pattern match value.
`shouldNot`	boolean default: `false`	Select this if you want to reverse the pattern match condition.

Example

```
{
                "affectedInfoType": "TIERS_NODES",
                "affectedEntities": {
                        "tierNodeType": "NODE",
                        "nodeType": "ALL_NODES",
                        "affectedNodes": {
                                "affectedNodeScope": "NODES_MATCHING_PATTERN",
                                "patternMatcher": {
                                        "matchTo": "STARTS_WITH",
                                        "matchValue": "A",
                                        "shouldNot": false
                                }
                        }
                }
}
```

Node Property Variable Matcher

Use this to suppress actions for entities at the node level for nodes that match specified environment variables.

Property Name	Type	Description and Valid Values
`affectedNodeScope`*	string	**Enum** `NODE_PROPERTY_VARIABLE_MATCHER`
`propVarPairs`*		

Property Name	Type	Description

`propertyType`*	string	**Enums**
		`META`
		`ENV`
		`JVM`

`name`*	string minLength: 1	
`value`*	string minLength: 1	

Example

```
{
        "affectedInfoType": "TIERS_NODES",
        "affectedEntities": {
                "tierNodeType": "NODE",
                "nodeType": "ALL_NODES",
                "affectedNodes": {
                        "affectedNodeScope": "NODE_PROPERTY_VARIABLE_MATCHER",
                        "propVarPairs": [{
                                "propertyType": "ENV",
                                "name": "CLASSPATH",
                                "value": "C:\\Users\\Java\\Classes"
                        }]
                }
        }
}
```

Servers Affected

Use this to suppress actions for entities at the server level.

Property Name	Type	Description and Valid Values
`affectedInfoType`*	string	**Enum** SERVERS
`affectedServers`*		**ApplicationAffectedServers** **Property Name** / **Type** / **Description** `serversScope`* / string / **Enum** `SPECIFIC_SERVERS_IN_APPLICATION`

Example

```
{
   "affectedInfoType":"SERVERS",
    "affectedServers":{
       "serversScope":"SPECIFIC_SERVERS_IN_APPLICATION",
       "specificServers":["DropWizardTestApplicationDropWizardDefaultNode1"]
     }
}
```

Specific Servers within an Application

Use this property to suppress actions for entities at the server level for specific servers within an application.

Property Name	Type	Description
`serversScope`*	string	**Enum** `SPECIFIC_SERVERS_IN_APPLICATION`
`specificServers`*	array of strings minItems: 1 MinLength: 1	A list of servers considered as affected entities for action suppression. For example: `[server1, server2]`

Example

```
{
   "affectedInfoType":"SERVERS",
    "affectedServers":{
       "serversScope":"SPECIFIC_SERVERS_IN_APPLICATION",
       "specificServers":["DropWizardTestApplicationDropWizardDefaultNode1"]
     }
}
```

Databases Affected

Use this property to suppress actions for entities at the database level.

Property Name	Type	Description and Valid Values
`affectedInfoType`*	string	**Enum** `DATABASES`
`affectedDatabases`*		**AffectedDatabases** **Property Name** / **Type** / **Description** `databaseScope`* — string — **Property Name** / **Type** / **Description** AffectedDatabaseType — string — **Enums**: `ALL_DATABASES`, `SPECIFIC_DATABASES`

Example

```
{
   "affects":{
      "affectedInfoType":"DATABASES",
      "affectedDatabases":{
         "databaseScope":"ALL_DATABASES"
}
```

Scope of Affected Databases

Use this property to define the scope of action suppression at the database level.

Property Name	Type	Description
`databaseScope`*	string	**Property Name** / **Type** / **Description** `AffectedDatabaseType` — string — **Enums**: `ALL_DATABASES`, `SPECIFIC_DATABASES`

Scope of Affected Databases—All Databases

Use this property to suppress actions for entities of all the databases.

Property Name	Type	Description
`databaseScope`*	string	**Property Name** / **Type** / **Description**

		AffectedDatabaseType	string	**Enums** ALL_DATABASES

Scope of Affected Databases—Specific Databases

Use this property to suppress actions for entities of specific databases.

Property Name	Type	Description
databaseScope*	string	**Enums** SPECIFIC_DATABASES
databases*	array of strings minItems: 1	**DbServer** serverName* (string, minLength: 1) collectorConfigName* (string, minLength: 1)

Example

```
{
"affects":{
      "affectedInfoType":"DATABASES",
      "affectedDatabases":{
         "databaseScope":"SPECIFIC_DATABASES",
         "databases":[
            {
               "serverName":"MongoNewKC",
               "collectorConfigName":"MongoNewKC"
            },
            {
               "serverName":"Collector41",
               "collectorConfigName":"Collector41"
            }
         ]
      }
   }
}
```

Database Server Details

Use this property to define database server details.

Property Name	Type
serverName*	string minLength: 1
collectorConfigName*	string minLength: 1

Affected Database Types

Use this property to define the scope of affected databases.

Property Name	Type	Description

`AffectedDatabaseType`	string	**Enums**
		`ALL_DATABASES`
		`SPECIFIC_DATABASES`

Health Rule Scope

Use this to suppress actions triggered as a response to health rule violation events.

Property Name	Type	Description
`healthRuleScopeType`*	string	**HealthRuleScopeType** Enum `SPECIFIC_HEALTH_RULES`

Example

```
{
   "healthRuleScopeType": "SPECIFIC_HEALTH_RULES",
   "healthRules": ["CPU utilization is too high", "JVM Garbage Collection Time is too high"]
}
```

Health Rule Scope Type

Property Name	Type	Description
`healthRuleScope`*	string	**Enum** `SPECIFIC_HEALTH_RULES`

Specific Health Rules

Property Name	Type	Description
`healthRuleScopeType`*	string	**HealthRuleScopeType** Enum `SPECIFIC_HEALTH_RULES`
`healthRules*`	array of strings minItems: 1	

Example

```
{
   "healthRuleScopeType": "SPECIFIC_HEALTH_RULES",
   "healthRules": ["CPU utilization is too high", "JVM Garbage Collection Time is too high"]
}
```

Action Suppression Summary

Property Name	Type
`id`*	integer

`name`*	string minLength: 1

Error Response

Property Name	Type
`statusCode`	integer
`message`	string

***This property is required.**

Download Examples

Download Appdynamics Action Suppression examples.zip to get a set of examples that help you configure a schedule.

Download SWAGGER YAML file

Download the Swagger YAML spec action_suppression_openapi.yml.

Events and Action Suppression API

This page describes the Events and Suppression API methods you can use to create, manage, and monitor events and action suppression.

Retrieve All Health Rule Violations in a Business Application

Returns all health rule violations that have occurred in an application within a specified time frame.

URI

`/controller/rest/applications/application_id/problems/healthrule-violations`

Input parameters

Parameter Name	Parameter Type	Value	Mandatory
`application_id`	URI	Provide either the application name or application id.	Yes

`time-range-type`	Query	Possible values are:
		`BEFORE_NOW`
		To use the "`BEFORE_NOW`" option, you must also specify the "`duration-in-mins`" parameter.
		`BEFORE_TIME`
		To use the "`BEFORE_TIME`" option, you must also specify the "`duration-in-mins`" and "`end-time`" parameters.
		`AFTER_TIME`
		To use the "`AFTER_TIME`" option, you must also specify the "`duration-in-mins`" and "`start-time`" parameters.
		
		`BETWEEN_TIMES`
		
		To use this option, you must also specify the "`start-time`" and "`end-time`" parameters. The "`BETWEEN_TIMES`" range includes the start-time and excludes the end-time.
			Yes

`duration-in-mins`	Query	Duration (in minutes) to return the metric data.	If time-range-type is `BEFORE_NOW, BEFORE_TIME, or AFTER_TIME`
`start-time`	Query	Start time (in milliseconds) from which the metric data is returned.	If time-range-type is `AFTER_TIME` or `BETWEEN_TIMES`
`end-time`	Query	End time (in milliseconds) until which the metric data is returned.	If time-range-type is `BEFORE_TIME` or `BETWEEN_TIMES`
`output`	Query	HTTP Request parameter included as part of the URL to change the output format. Valid values are `XML` (default) or `JSON`.	No

```
http://demo.appdynamics.com/controller/rest/applications/7/problems/healthrule-violations?time-range-
type=BEFORE_NOW&duration-in-mins=15
<policy-violations><policy-violation>
  <id>266</id>
  <name>CPU utilization is too high</name>
  <startTimeInMillis>1452630655000</startTimeInMillis>
  <detectedTimeInMillis>0</detectedTimeInMillis>
  <endTimeInMillis>1452630715000</endTimeInMillis>
  <incidentStatus>RESOLVED</incidentStatus>
  <severity>WARNING</severity>
  <triggeredEntityDefinition>
    <entityType>POLICY</entityType>
    <entityId>30</entityId>
    <name>CPU utilization is too high</name>
  </triggeredEntityDefinition>
  <affectedEntityDefinition>
    <entityType>APPLICATION_COMPONENT_NODE</entityType>
    <entityId>16</entityId>
    <name>Fulfillment</name>
  </affectedEntityDefinition>
  <deepLinkUrl>http://demo.appdynamics.com/controller/#location=APP_INCIDENT_DETAIL&amp;incident=266<
/deepLinkUrl>
  <description>AppDynamics has detected a problem.<br><b>errorAbhi</b> is violating.
</description>
</policy-violation>
<policy-violation>
  <id>268</id>
  <name>CPU utilization is too high</name>
  <startTimeInMillis>1452630655000</startTimeInMillis>
  <detectedTimeInMillis>0</detectedTimeInMillis>
  <endTimeInMillis>1452630715000</endTimeInMillis>
  <incidentStatus>RESOLVED</incidentStatus>
  <severity>WARNING</severity>
  <triggeredEntityDefinition>
    <entityType>POLICY</entityType>
    <entityId>30</entityId>
    <name>CPU utilization is too high</name>
  </triggeredEntityDefinition>
  <affectedEntityDefinition>
    <entityType>APPLICATION_COMPONENT_NODE</entityType>
    <entityId>20</entityId>
    <name>FulfillmentClient</name>
  </affectedEntityDefinition>
  <deepLinkUrl>http://demo.appdynamics.com/controller/#location=APP_INCIDENT_DETAIL&amp;incident=268<
/deepLinkUrl>
  <description>AppDynamics has detected a problem with Node &lt;b&gt;FulfillmentClient&lt;/b&gt;.&lt;br&gt;&lt;
b&gt;CPU utilization is too high&lt;/b&gt; started violating and is now &lt;b&gt;warning&lt;/b&gt;.&lt;br&gt;
All of the following conditions were found to be violating&lt;br&gt;For Node &lt;b&gt;FulfillmentClient&lt;
/b&gt;:&lt;br&gt;1) Hardware Resources|CPU|%Busy Condition&lt;br&gt;&lt;b&gt;%Busy's&lt;/b&gt; value &lt;b&gt;
76.0&lt;/b&gt; was &lt;b&gt;greater than&lt;/b&gt; the threshold &lt;b&gt;75.0&lt;/b&gt; for the last &lt;b&gt;
30&lt;/b&gt; minutes&lt;br&gt;</description>
</policy-violation>
</policy-violations>
```

Retrieve Event Data

You can capture data for the event types listed in the `eventtypes` parameter.

URI

`/controller/rest/applications/application_id/events`

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory

`application_id`	URI	Provides either the application name or application id.	Yes
`summary`	Query	Provides the summary for the event.	Yes
`comment`	Query	Provides the comments (if any) for the event.	No
`eventtype`	Query	`APPLICATION_DEVELOPMENT`	Yes

`time-range-type`	Query	Possible values are:
		`BEFORE_NOW`
		To use the "`BEFORE_NOW`" option, you must also specify the "`duration-in-mins`" parameter.
		`BEFORE_TIME`
		To use the "`BEFORE_TIME`" option, you must also specify the "`duration-in-mins`" and "`end-time`" parameters.
		`AFTER_TIME`
		To use the "`AFTER_TIME`" option, you must also specify the "`duration-in-mins`" and "`start-time`" parameters.
		
		`BETWEEN_TIMES`
		
		To use this option, you must also specify the "start-time" and "end-time" parameters. The "BETWEEN_TIMES" range includes the start-time and excludes the end-time.
			Yes

`duration-in-mins`	Query	Specify the duration (in minutes) to return the metric data.	If time-range-type is `BEFORE_NOW, BEFORE_TIME,` or `AFTER_TIME`
`start-time`	Query	Specify the start time (in milliseconds) from which the metric data is returned.	If time-range-type is `AFTER_TIME` or `BETWEEN_TIMES`
`end-time`	Query	Specify the end time (in milliseconds) until which the metric data is returned.	If time-range-type is `BEFORE_TIME` or `BETWEEN_TIMES`
`event-types`	Query	Specify the comma-separated list of event types for which you want to retrieve event information. See Events Reference.	Yes
`severity`	Query	Provides the severity level. Specify the comma-separated list of severities for which you want to retrieve event information. Allowed values are: • `INFO` • `WARN` • `ERROR` In the UI these values become `Info, Warning, and Critical`.	Yes
`output`	Query	HTTP Request parameter included as part of the URL to change the output format. Valid values are `XML` (default) or `JSON`.	No
`tier`	Query	Name of the tier in the application	No

This API can retrieve 600 events at a time.

Example

Retrieve the list of events of type `APPLICATION_ERROR` or `DIAGNOSTIC_SESSION` of any severity that occurred in the specified time range:

```
curl --user user1@customer1:your_password http://demo.appdynamics.com//controller/rest/applications/6/events?
time-range-type=BEFORE_NOW\&duration-in-mins=30\&event-types=%20APPLICATION_ERROR,
DIAGNOSTIC_SESSION\&severities=INFO,WARN,ERROR

<events><event>
  <id>44658</id>
  <type>DIAGNOSTIC_SESSION</type>
  <subType>ERROR_DIAGNOSTIC_SESSION</subType>
  <eventTime>1451343453085</eventTime>
  <severity>WARN</severity>
  <summary>Starting Diagnostic Session after series of errors for a Business Transaction 18% (2/11) of requests
had errors in the last minute starting 12/28/15 10:57 PM local time</summary>
  <affectedEntities>
    <entity-definition>
      <entityType>APPLICATION</entityType>
      <entityId>6</entityId>
      <name>ECommerce</name>
    </entity-definition>
    <entity-definition>
      <entityType>APPLICATION_COMPONENT</entityType>
      <entityId>11</entityId>
      <name>ECommerce-Services</name>
    </entity-definition>
    <entity-definition>
      <entityType>APPLICATION_COMPONENT_NODE</entityType>
      <entityId>19</entityId>
      <name>ECommerce_WEB2</name>
    </entity-definition>
    <entity-definition>
      <entityType>BUSINESS_TRANSACTION</entityType>
      <entityId>35</entityId>
      <name>/items/all.GET</name>
    </entity-definition>
    <entity-definition>
      <entityType>MACHINE_INSTANCE</entityType>
      <entityId>8</entityId>
      <name>ECommerce-web1</name>
    </entity-definition>
  </affectedEntities>
  <triggeredEntity>
    <entityType>APPLICATION_COMPONENT_NODE</entityType>
    <entityId>19</entityId>
    <name>ECommerce_WEB2</name>
  </triggeredEntity>
  <markedAsRead>false</markedAsRead>
  <markedAsResolved>false</markedAsResolved>
  <archived>false</archived>
  <deepLinkUrl>http://demo.appdynamics.com:8090/controller/#location=APP_EVENT_VIEWER_MODAL&amp;
eventSummary=44658</deepLinkUrl>
</event>
</events>
```

Create Events

Application deployment events notify AppDynamics when you upgrade your application, push new code, etc. This lets you correlate these application deployment activities with other data inside AppDynamics. This is useful for regression analysis, root cause analysis, and performance studies. It is beneficial to inject your application deployment event into AppDynamics as part of the build process for deploying a new version of your application.

The AppDynamics REST API lets you integrate events of type `APPLICATION_DEPLOYMENT` with other systems.

For example, to create an event automatically in your AppDynamics monitored system for every new release you would integrate these systems and use the following REST API to create an event of type "`APPLICATION_DEPLOYMENT`" in your managed environment.

You should receive the event ID after the successful invocation of the request.

Roles and Permissions

Creating events requires the **Create Events** permission. See Application Permissions.

URI

```
POST /controller/rest/applications/application_id/events
```

Input parameters

Parameter Name	Parameter Type	Value	Mandatory
`application_id`	URI	Provide either application name or application id.	Yes
`summary`	Query	Provide a summary describing the event.	Yes
`comment`	Query	Provide the comments (if any) for the event.	No
`eventtype`	Query	`APPLICATION_DEPLOYMENT`	Yes
`severity`	Query	Provide a severity level. Allowed values include: • `"INFO"` • `"WARN"` • `"ERROR"` In the UI, these become `"Info", "Warning"`, and `"Critical"`	Yes

Create a Custom Event

You can create custom events to be reported in the AppDynamics event viewer and in the event panels on the AppDynamics dashboards. See Monitor Events to learn how to filter on your custom events. Then you can create alerts triggered by these events as you do for AppDynamics standard events.

You should receive the event ID after the successful invocation of the request.

Roles and Permissions

Creating a custom event requires the **Create Events** permission. See Application Permissions.

URI

```
POST /controller/rest/applications/application_id/events
```

Input parameters

Parameter Name	Parameter Type	Value	Mandatory
`application_id`	URI	Provide either application name or application id.	Yes
`summary`	Query	Provide a summary describing the event.	Yes
`comment`	Query	Provide a comment for the event.	No
`severity`	Query	Provide a severity level. Allowed values include: • `"INFO"` • `"WARN"` • `"ERROR"` In the UI, these become `"Info", "Warning"`, and `"Critical"`	Yes
`eventtype`	Query	CUSTOM	Yes
`customeventtype`	Query	Provide a name for the `"type"`. For example, the source could be `"nagios"`.	No
`node`	Query	Provide the affected node name.	No
`tier`	Query	Provide the affected tier name.	Yes, if `node` and `bt` are specified
`bt`	Query	Provide the affected business transaction name.	No
`propertynames`	Query	Provide a property name as a pair, i.e., the `"key"`.	No, but if one element of the pair is defined, the other must be defined also

`propertyvalues`	Query	Provide the property value as a pair, i.e., the "`value`".	No, but if one element of the pair is defined, the other must be defined also

Example

```
curl -X POST --user user1@customer1:your_password 'http://demo.appdynamics.com/controller/rest/applications/5
/events?
severity=INFO&summary=test1&eventtype=CUSTOM&customeventtype=mycustomevent&propertynames=key1&propertynames=key2
&propertyvalues=value1&propertyvalues=value'
```


Notice the pattern for custom properties: `propertynames` and `propertyvalues` get matched up by order position, so to set N property values, you need N occurrences of `propertynames` and N occurrences of `propertyvalues`.

Create Custom URLs for Notifications

Single tenants in a multi-tenant Controller instance should use this API method to specify a custom or vanity URL for notification purposes. Instead of a URL such as `paid8.appdynamics.com` being displayed as the host, the custom URL can be displayed as `yourcompany.appdynamics.com` in the notification.

URI

`POST /controller/rest/accounts/customer_name/update-controller-url`

Input parameters

Parameter Name	Parameter Type	Value	Mandatory
`customer_name`	URI	The customer account name	Yes

Body Parameter

As Application/JSON content:

```
{
   "controllerURL": "http://<my-custom-hostname:port>"
}
```

If the URL in the alerts is invalid, you can set it using the following curl command:

```
curl -k --basic --user root@system --header "Content-Type: application/json" --data '{ "controllerURL":
"http://<controller>:<port>" }' http://<controller>:<port>/controller/rest/accounts/<ACCOUNT-NAME>
/update-controller-url
```

```
curl -k --basic --user root@system --header "Content-Type: application/json" --data '{ "controllerURL":
"http://<controller>:<port>" }' http://<controller>:<port>/controller/rest/accounts/<ACCOUNT-NAME>
/update-controller-url
```

For example:

```
curl -k --basic --user root@system --header "Content-Type: application/json" --data '{ "controllerURL":
"https://myVIP:443" }' https://myhost:8181/controller/rest/accounts/customer1/update-controller-url
```

There is no need to reset the Controller as upgrading it will reset the deep link URL settings.

Create and Delete Action Suppressions

By default any response is in JSON, although XML can be requested by using the following header:
`Name : Accept`
`Value : application/vnd.appd.cntrl+xml;v=1`

Retrieve All Existing Action Suppressions

Use this to retrieve a list of all existing action suppressions.

URI

`GET /controller/api/accounts/account_id/applications/application_id/actionsuppressions`

Input parameters

Parameter Name	Parameter Type	Value	Mandatory
`account_id`	URI	The account ID	Yes
`application_id`	URI	The application ID	Yes

Example request to get all action suppressions:

```
/controller/api/accounts/2/applications/9/actionsuppressions
```

Example response:

```
Status : 200 ok
Output Data :
{"actionSuppressions": [{"id": "15","name": "App-ASW","timeRange": {"startTimeMillis": "2014-10-25T04:16:
30+0000","endTimeMillis": "2014-10-25T06:16:30+0000"},"affects": {"type": "APP"}},{"id": "16","name": "Node-
ASW","timeRange": {"startTimeMillis": "2014-10-25T04:16:57+0000","endTimeMillis": "2014-10-25T05:16:57+0000"},"
healthRuleIds": [60,61],"affects": {"type": "NODE","nodeAffectedEntities": {"type": "SPECIFIC","nodeType":
"ALL","nodes": [17,18]}}}],"actions": [{"href": "http://demo.appdynamics.com:8090/controller/api/accounts/2
/applications/9/actionsuppressions/%7Bactionsuppressions.id%7D/%7Bactions.name%7D","method": ["POST","DELETE"],"
name": "enabled"}],"links": [{"href": "http://ec2-54-80-163-175.compute-1.amazonaws.com:8090/controller/api
/accounts/2/applications/9/actionsuppressions/%7Bactionsuppressions.id%7D","name": "actionsuppressions"}]}
```

Retrieve a Specific Action Suppression by ID

Use this to get an action suppression by the specified ID.

URI

`/controller/api/accounts/account_id/applications/application_id/actionsuppressions/actionsuppression_id`

Input parameters

Parameter Name	Parameter Type	Value	Mandatory
`account_id`	URI	The account id.	Yes
`application_id`	URI	The application id.	Yes
`actionsuppressions_id`	URI	The action suppression id.	Yes

Example request:

```
/controller/api/accounts/2/applications/9/actionsuppressions/15
```

Example response:

```
Status : 200 ok
Output Data :
{"id": "15","name": "App-ASW","timeRange": {"startTimeMillis": "2014-10-25T04:16:30+0000","endTimeMillis":
"2014-10-25T06:16:30+0000"},"affects": {"type": "APP"}}
```

Create a New Action Suppression

This is a POST request that should return a 201 - created response.

URI

```
POST /controller/api/accounts/account_id/applications/application_id/actionsuppressions
```

Input parameters

Parameter Name	Parameter Type	Value	Mandatory
`account_id`	URI	The account ID.	Yes
`application_id`	URI	The application ID.	Yes
`name`	body key	The name of the action suppression window.	Yes
`timeRange`	body key	The start and end time of the window. Includes: `startTimeMillis`, eg 2014-10-25T04:16:57+0000 `endTimeMillis`, eg 2014-10-25T05:16:57+0000	Yes
`healthRuleIds`	body key	The ids of the affected health rules. If not provided, all rules affected	No

<table>
<tr>
<td><code>affects</code></td>
<td>body key</td>
<td>Type of entity and corresponding IDs:
<table>
<tr><th>Scope</th><th>Type</th><th>Value</th><th>Example</th></tr>
<tr><td>Application</td><td>APP</td><td>Covers the entire application</td><td><code>"affects": {"type": "APP"}</code></td></tr>
<tr><td>Business Transaction</td><td>BT</td><td>Covers one or more Business Transactions</td><td></td></tr>
<tr><td></td><td></td><td>All Business Transactions</td><td><code>"affects": {"type": "BT","btAffectedEntities": {"type": "ALL"}}</code></td></tr>
<tr><td></td><td></td><td>Business Transactions from specific tiers</td><td><code>"affects": {"type": "BT","btAffectedEntities": {"type": "WITHIN_TIERS","tiers": [11,12]}}</code> where 11,12 are Tier Ids</td></tr>
<tr><td></td><td></td><td>Specific Business Transactions by id</td><td><code>"affects": {"type": "BT","btAffectedEntities": {"type": "SPECIFIC","bts": [1,2]}}</code> where 1,2 are BT Ids</td></tr>
<tr><td></td><td></td><td>Business Transactions that match criteria</td><td><code>"affects": {"type": "BT","btAffectedEntities": {"type": "CRITERIA","matchesOperator": "CONTAINS","matchesValue": "pojo"}}</code> where "<code>matchesOperator</code>" can be:
CONTAINS
EQUALS
STARTS
ENDS
REGEX_VALUE</td></tr>
<tr><td>Tier</td><td>TIER</td><td>Covers one or more tiers</td><td></td></tr>
<tr><td></td><td></td><td>All tiers</td><td><code>"affects": {"type": "TIER","tierAffectedEntities": {"type": "ALL"}}</code></td></tr>
<tr><td></td><td></td><td>Specific tiers</td><td><code>"affects": {"type": "TIER","tierAffectedEntities": {"type": "SPECIFIC","tiers": [11,12]}}</code> where 11,12 are Tier Ids</td></tr>
<tr><td>Node</td><td>NODE</td><td>Covers one or more nodes</td><td></td></tr>
<tr><td></td><td></td><td>All nodes</td><td><code>"affects": {"type": "NODE","nodeAffectedEntities": {"type": "ALL","nodeType": "ALL"}}</code></td></tr>
<tr><td></td><td></td><td>Nodes belonging to specific tiers</td><td><code>"affects": {"type": "NODE","nodeAffectedEntities": {"type": "WITHIN_TIERS","nodeType": "ALL","tiers": [11,12]}}</code> where 11,12 are Tier Ids</td></tr>
<tr><td></td><td></td><td>Specific nodes</td><td>"affects": {"type": "NODE","nodeAffectedEntities": {"type": "SPECIFIC","nodeType": "ALL","nodes": [9,10]}} where 9,10 are Node Ids</td></tr>
<tr><td></td><td></td><td>Nodes that match criteria</td><td><code>"affects": {"type": "NODE","nodeAffectedEntities": {"type": "NAME_CRITERIA","nodeType": "ALL","nameMatchesOperator": "EQUALS","nameMatchesValue": "Node"}}</code> - String match
<code>"affects": {"type": "NODE","nodeAffectedEntities": {"type": "PROPERTY_CRITERIA","nodeType": "ALL","metaInfoProperties": [{"name": "ProcessID","value": 12343}]}}</code> - Meta Info Properties match

where "matchesOperator" can be:
<code>CONTAINS</code>
<code>EQUALS</code>
<code>STARTS</code>
<code>ENDS</code>
<code>REGEX_VALUE</code></td></tr>
<tr><td>Machine</td><td>MACHINE</td><td>Covers one or more machines</td><td><code>"affects": {"type": "MACHINE","machineAffectedEntities": {"type": "SPECITIC","machines": [4,5]}}</code> where 4,5 are Machine Ids
You must spell the value as "SPECITIC" due to a typo in the underlying code.</td></tr>
</table>
</td>
<td>Yes</td>
</tr>
</table>

Example request

```
/controller/api/accounts/2/applications/9/actionsuppressions
```

Header

- `Name: Content-Type`
- `Value: application/vnd.appd.cntrl+json;v=1`

Body

```
{"name": "App-ASW_2","timeRange": {"startTimeMillis": "2014-10-25T04:16:30+0000","endTimeMillis": "2014-10-25T06:16:30+0000"},"affects": {"type": "APP"}}
```

or

```
{"name": "Node-ASW_1","timeRange": {"startTimeMillis": "2014-10-25T04:16:57+0000","endTimeMillis": "2014-10-25T05:16:57+0000"},"healthRuleIds": [60,61],"affects": {"type": "NODE","nodeAffectedEntities": {"type": "SPECIFIC","nodeType": "ALL","nodes": [17,18]}}}
```

Delete a Specific Action Suppression by ID

This is a `DELETE` request that should return a `204 - No Content` message.

URI

`DELETE /controller/api/accounts/account_id/applications/application_id/actionsuppressions/actionsuppression_id`

Input parameters

Parameter Name	Parameter Type	Value	Mandatory
`account_id`	URI	The account ID	Yes
`application_id`	URI	The application ID	Yes
`actionsuppression_id`	URI	The ID of the action suppression to be deleted	Yes—The Alert and Respond APIs let you manage and monitor events, health rules, and other aspects of the AppDynamics alert and respond features.

Configuration API

This page describes the Configuration API methods you can use to read and modify selected Controller configuration settings programmatically. You can use it to script or automate tasks that must be performed frequently or in large batches, such as adding users.

> The Configuration Export and Import API provides the ability to perform select configuration changes as well as you can edit and import Controller configuration definition files.

Create and Modify AppDynamics Users

Use this to create or modify user accounts in the Controller.

You pass the user configuration settings as query parameters to the API call. The format of the create and modify user calls are identical except for the `user-id` parameter, which is not passed for the create operation. The `user-id` is generated by the `create` operation.

Format

```
POST /controller/rest/users
```

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory
`user-name`	Query	user name	Yes
`user-id`	Query	user id	No for a `create`; yes for an `update`
`user-display-name`	Query	display name	Yes
`user-roles`	Query	comma-separated list of roles	No
`user-password`	Query	user password	Yes for a `create`; optional for an `update`
`user-email`	Query	user email	Yes

Example

```
curl -X POST --user user1@customer1:your_password http://demo.appdynamics.com/controller/rest/users?user-
name=user2\&user-display-name=User%20Two\&user-password=welcome\&user-email=user2\@example.com
```

Include or Exclude a Business Transaction from Monitoring

You can exclude or include business transactions for monitoring by passing the exclude parameter to the business-transactions retrieval API described in the Application Model API.

To exclude a business transaction, pass the XML-represented ID of the business transaction to be excluded with the exclude parameter set to `true`. To turn on monitoring for a currently excluded business transaction, set the exclude parameter to `false`.

Send the list of business transactions to be excluded or re-included as the XML-formatted `POST` payload. A sample business-transaction list is:

```
<business-transactions>
    <business-transaction>
        <id>15</id>
    </business-transaction>
    <business-transaction>
        <id>16</id>
    </business-transaction>
</business-transactions>
```

> Ensure that the `Content-Type` header is set to `application/xml`.

Format

```
POST /controller/rest/applications/application_id/business-transactions
```

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory
`application_id`	URI	Provide either the application name or application id.	Yes
`exclude`	Post	`true|false`	Yes

Example

```
curl -X POST -H "Content-Type:text/xml" --user user1@customer1:your_password http://demo.appdynamics.com
/controller/rest/applications/6/business-transactions\?exclude=true -d @businesstransaction.xml
```

Retrieve All Controller Settings

The Controller global configuration values are made up of the Controller settings that are presented in the Administration Console.

Format

`GET /controller/rest/configuration`

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory
`output`	Query	HTTP Request parameter included as part of the URL to change the output format. Valid values are `XML` (default) or `JSON`.	No

Example

```
curl --user admin@customer1:your_password http://demo.appdynamics.com/controller/rest/configuration

<configuration-items><configuration-item>
  <name>eventsvc.request.segment.data.max.retrieval.size</name>
  <value>5000</value>
  <description>Max number of request segment data that can be retrieved from event service in a query<
/description>
  <updateable>true</updateable>
  <scope>cluster</scope>
</configuration-item>
<configuration-item>
  <name>machine.agent.max.new.actions.per.min</name>
  <value>15</value>
  <description>Maximum number of new actions dispatched per minute for each machine agent</description>
  <updateable>true</updateable>
  <scope>cluster</scope>
</configuration-item>
...
<configuration-item>
  <name>tss.retention.period</name>
  <value>336</value>
  <description>Time (in hours) to retain 12 hour tss data values before they are purged from the system.<
/description>
  <updateable>true</updateable>
  <scope>cluster</scope>
</configuration-item>
<configuration-item>
  <name>snapshots.retention.period</name>
  <value>336</value>
  <description>Time (in hours) to retain snapshots before they are purged from the system.</description>
  <updateable>true</updateable>
  <scope>cluster</scope>
</configuration-item>
<configuration-item>
  <name>metrics.min.retention.period</name>
  <value>4</value>
  <description>Time (in hours) to retain minute metric data values before they are purged from the system.<
/description>
  <updateable>true</updateable>
  <scope>cluster</scope>
</configuration-item>
<configuration-item>
  <name>system.notification.event.types</name>
  <value>LICENSE,DISK_SPACE,CONTROLLER_AGENT_VERSION_INCOMPATIBILITY,CONTROLLER_EVENT_UPLOAD_LIMIT_REACHED,
CONTROLLER_RSD_UPLOAD_LIMIT_REACHED,CONTROLLER_METRIC_REG_LIMIT_REACHED,CONTROLLER_METRIC_DATA_BUFFER_OVERFLOW,
CONTROLLER_ERROR_ADD_REG_LIMIT_REACHED,CONTROLLER_ASYNC_ADD_REG_LIMIT_REACHED,
AGENT_ADD_BLACKLIST_REG_LIMIT_REACHED,AGENT_METRIC_BLACKLIST_REG_LIMIT_REACHED,
CONTROLLER_STACKTRACE_ADD_REG_LIMIT_REACHED,CONTROLLER_SEP_ADD_REG_LIMIT_REACHED,
CONTROLLER_MEMORY_ADD_REG_LIMIT_REACHED,CONTROLLER_TRACKED_OBJECT_ADD_REG_LIMIT_REACHED,
CONTROLLER_COLLECTIONS_ADD_REG_LIMIT_REACHED</value>
  <description>Comma separated list of Event Types (with no spaces between each) that will shown as System
Notifications in the UI.</description>
  <updateable>true</updateable>
  <scope>cluster</scope>
</configuration-item>
</configuration-items>
```

Retrieve a Controller Setting by Name

Use this to get the value of a given Controller configuration setting.

Format

```
GET /configuration?name=controller_setting_name
```

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory
`name`	Query	Name of the Controller setting to retrieve	Yes
`output`	Query	HTTP Request parameter included as part of the URL to change the output format. Valid values are `XML` (default) or `JSON`.	No

Example

```
curl --user admin@customer1:your_password http://demo.appdynamics.com/controller/rest/configuration?
name=metrics\.min\.retention\.period

<configuration-items><configuration-item>
  <name>metrics.min.retention.period</name>
  <value>4</value>
  <description>Time (in hours) to retain minute metric data values before they are purged from the system.<
/description>
  <updateable>true</updateable>
  <scope>cluster</scope>
</configuration-item>
</configuration-items>
```

Configure Global Controller Settings

Use this to set a Controller setting to a specified value.

You cannot use this REST API to modify Controller settings on SaaS.

Format

`POST /controller/rest/configuration`

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory
`name`	Query	Name of the Controller setting to get.	Yes
`value`	Query	Value to set.	Yes

Mark Nodes as Historical

Use this to mark nodes as historical, which directs AppDynamics to stop collecting metrics for the nodes. By default, AppDynamics marks as historical (soft deletes) a node that has lost contact with the Controller for the number of hours configured in the `node.retention.period` Controller setting. The default is 500 hours.

Pass one or more identifiers of the node to be marked as historical, up to a maximum of 25 nodes. Multiple IDs should be comma-separated.

Format

`POST /controller/rest/mark-nodes-historical?application-component-node-ids=value`

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory
`application-component-node-ids`	Query	Comma-separated list of node IDs	Yes

Example

```
curl -X POST --user admin@customer1:your_password http://demo.appdynamics.com/controller/rest/mark-nodes-historical?application-component-node-ids=44,45

<application-component-node-id>
<44/>
<45/>
</application-component-node-id>
```

Configuration Import and Export API

This page describes the AppDynamics API methods you can use to import and export various types of configuration settings in the Controller.

About the Configuration Import/Export APIs

The Configuration Import/Export APIs enable you to migrate configuration settings across Controller accounts, business applications, or Controller instances. You can also use it to add configuration artifacts like transaction detection rules, health rules, or custom dashboards to an existing configuration programmatically.

An exported configuration is an XML or JSON representation of the configuration artifact. After exporting the file, you can upload it to another account or application, optionally modifying the configuration.

Export Actions from an Application

Use this to export all actions in the specified application to a JSON file.

The user account you use to make the API call must have permission to view an action or action template in the application you are exporting from.

Format

```
GET /controller/actions/application_id
```

Input parameters

Parameter Name	Parameter Type	Value	Mandatory
`application_id`	URI	The application name or application ID.	Yes

Example

```
curl --user user1@customer1:your_password http://demo.appdynamics.com/controller/actions/7

[
   {
      actionType: "EmailAction",
      name: "6DA8942B-DF4A-417A-E1NF-59F14231D670",
      priority: 1,
      description: null,
      toAddress: "user1@example.com",
      subject: "",
      timeZone: null
   },
   {
      actionType: "DiagnosticSessionAction",
      name: "MyDiagnotic",
      priority: 0,
      description: null,
      businessTransactionTemplates: [ ],
      numberOfSnapshotsPerMinute: 5,
      durationInMinutes: 10,
      adjudicate: false,
      adjudicatorEmail: null
   }
]
```

Import Actions into an Application

After you have exported actions, you can import them to a different application passing the JSON file created by the export operation as the payload to a POST request.

The user account you use to make the API call must have permission to create an action or action template in the account.

Actions in the import file that have conflicting names with actions in the existing configuration are not imported. The import for those actions fails, while new actions are imported successfully.

This call takes data as multipart/form-data content. Use UTF-8 URL encoding of the URI before posting; for example, do not replace a space (" ") with "%20" in the URI.

Format

```
POST /controller/actions/application_id
```

Input parameters

Parameter Name	Parameter Type	Value	Mandatory
`application_id`	URI	The application name or application ID.	Yes

Example

```
curl  -X POST --user user1@customer1:your_password http://demo.appdynamics.com/controller/actions/38 -F
file=@ExportActions.json

{"success":true,"errors":[],"warnings":[]}
```

If there are actions in the file with the same name as ones in the configuration, those actions are not imported, and the response indicates the success of the request as `false`. For example:

```
{"success":false,"errors":["Not importing Action with name: DuplicateExportedDiagnosticAction, since it already
exists."],"warnings":["Imported 1 out of 2 actions"]}
```

Export Email Action Templates from an Account

This API exports all the email action templates in the current account in JSON format.

Format

```
GET /controller/actiontemplate/email
```

Example

```
curl --user user1@customer1:your_password http://demo.appdynamics.com/controller/actiontemplate/email

[ {
  "actionPlanType" : "email",
  "name" : "MyCustomEmailTemplate",
  "oneEmailPerEvent" : true,
  "eventClampLimit" : 100,
  "defaultCustomProperties" : [ {
    "id" : 0,
    "version" : 0,
    "name" : "env",
    "value" : "%OS"
  } ],
  "allowCustomRecipients" : true,
  "toRecipients" : [ ],
  "ccRecipients" : [ ],
  "bccRecipients" : [ ],
  "headers" : [ ],
  "subject" : "We've got a situation...",
  "includeTextBody" : true,
  "textBody" : "<h1>Summary of events occurring during the ${policy.digestDurationInMins}+ minute(s) prior to
${action.triggerTime}:</h1> <table> #foreach(${eventList} in ${fullEventsByTypeMap.values()}) #foreach(${event}
in ${eventList}) <tr> <td> <!-- Event icon --> <img src="${event.severityImage.mimeContentRef}" alt="${event.
severity}" /> </td> <td> <!-- Event name with event link --> <a href="${event.deepLink}">${event.displayName}<
/a> </td> <td> <!-- Event message --> ${event.eventMessage} </td> </tr> #end #end </table>"",
  "includeHtmlBody" : true,
  "htmlBody" : "<p>Please look into it.</p>",
  "testLogLevel" : "DEBUG",
  "testPropertiesPairs" : [ ],
  "testToRecipients" : [ ],
  "testCcRecipients" : [ ],
  "testBccRecipients" : [ ],
  "eventTypeCountPairs" : [ ]
} ]
```

Import Email Action Templates

Use this to import email action templates to an account as a JSON file.

> The import will fail if you attempt to import a template with the same name as an existing template of the same type in the destination account.

Data for this call should be in the form of multipart/form-data. Use UTF-8 URL encoding of the URI before posting; for example, do not replace a space (" ") with "%20" in the URI.

Format

```
POST /controller/actiontemplate/email
```

Example

```
curl  -X POST --user user1@customer1:your_password http://demo.appdynamics.com/controller/actiontemplate/email -
F file=@emailactiontemplate.json

{"success":true,"errors":[],"warnings":[]}
```

Export HTTP Request Action Templates from an Account

This API exports all the HTTP request action templates in the current account to a JSON file.

Format

```
GET /controller/actiontemplate/httprequest/
```

Example:

```
curl --user user1@customer1:your_password http://demo.appdynamics.com/controller/actiontemplate/httprequest

[ {
  "actionPlanType" : "httprequest",
  "name" : "MyCustomHTTPTemplate",
  "oneRequestPerEvent" : false,
  "eventClampLimit" : -1,
  "defaultCustomProperties" : [ ],
  "method" : "GET",
  "scheme" : "HTTP",
  "host" : "http",
  "port" : 0,
  "path" : "//demo.appdynamics.com//controller/rest/applications/${latestEvent.application.name}/nodes
/${latestEvent.node.name}",
  "query" : "",
  "urlCharset" : "UTF_8",
  "authType" : "BASIC",
  "authUsername" : "user1",
  "authPassword" : "your_password",
  "headers" : [ ],
  "payloadTemplate" : {
    "httpRequestActionMediaType" : "text/plain",
    "charset" : "UTF_8",
    "formDataPairs" : [ ],
    "payload" : ""
  },
  "connectTimeoutInMillis" : 5000,
  "socketTimeoutInMillis" : 15000,
  "maxFollowRedirects" : 0,
  "responseMatchCriteriaAnyTemplate" : [ ],
  "responseMatchCriteriaNoneTemplate" : [ ],
  "testLogLevel" : "DEBUG",
  "testPropertiesPairs" : [ ],
  "eventTypeCountPairs" : [ ]
} ]
```

Import HTTP Action Templates into an Account

After you have exported HTTP request action templates, you can import them to a different account by logging into the destination account and passing the JSON file created by the export operation as the payload to the `POST` request.

You can modify the exported file before you import it. You might want to do this to remove one or more template configurations or to change their names.

The import will fail if you attempt to import a template with the same name as an existing template of the same type in the destination account.

Use UTF-8 URL encoding of the URI before posting; for example, do not replace a space (" ") with "%20" in the URI.

Format

```
GET /controller/actiontemplate/httprequest
```

Example

```
curl  -X POST --user user1@customer1:your_password http://demo.appdynamics.com/controller/actiontemplate
/httprequest -F file=@httpactiontemplate.json

{"success":true,"errors":[],"warnings":[]}
```

Export Custom Dashboards and Templates

You can export and import custom dashboards and custom dashboard templates interactively from the Controller UI or by using this API call. See Import and Export Custom Dashboards and Templates Using the UI.

To export the dashboard, the user making the API call must have permission to view the custom dashboard.

In the export call, you must identify the dashboard to export by its ID. When you open the dashboard in the UI, the ID appears as the dashboard parameter at the end of the URL.

For example, in this URL snippet, the custom dashboard ID is 3: `location=CDASHBOARD_DETAIL&mode=MODE_DASHBOARD&dashboard=3`

Format

```
GET /controller/CustomDashboardImportExportServlet?dashboardId=dashboard_id
```

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory
`dashboardId`	Query	The numeric ID of the custom dashboard.	Yes

Example

```
curl --user user1@customer1:your_password http://demo.appdynamics.com/controller
/CustomDashboardImportExportServlet?dashboardId=8

{
  "schemaVersion" : null,
  "dashboardFormatVersion" : "3.0",
  "name" : "Analytics-BrowserData",
   ...
  "warRoom" : false,
  "template" : false
}
```

View a complete response example:

```
{
  "schemaVersion" : null,
  "dashboardFormatVersion" : "3.0",
  "name" : "Analytics-BrowserData",
  "description" : null,
  "properties" : null,
  "templateEntityType" : "APPLICATION_COMPONENT_NODE",
  "associatedEntityTemplates" : null,
  "minutesBeforeAnchorTime" : 15,
  "startDate" : null,
  "endDate" : null,
  "refreshInterval" : 120000,
  "backgroundColor" : 15395562,
  "color" : 15395562,
  "height" : 768,
  "width" : 1024,
  "canvasType" : "CANVAS_TYPE_GRID",
  "layoutType" : "",
  "widgetTemplates" : [ {
    "widgetType" : "AnalyticsWidget",
    "title" : "Browser_data",
    "height" : 4,
    "width" : 4,
    "x" : 0,
    "y" : 0,
    "label" : "",
    "description" : "",
    "drillDownUrl" : "",
    "useMetricBrowserAsDrillDown" : false,
```

```
    "backgroundColor" : 16777215,
    "backgroundColors" : null,
    "backgroundColorsStr" : null,
    "color" : 4210752,
    "fontSize" : 12,
    "useAutomaticFontSize" : false,
    "borderEnabled" : false,
    "borderThickness" : 0,
    "borderColor" : 0,
    "backgroundAlpha" : 1.0,
    "showValues" : false,
    "compactMode" : false,
    "showTimeRange" : false,
    "renderIn3D" : false,
    "showLegend" : false,
    "legendPosition" : null,
    "legendColumnCount" : null,
    "startTime" : null,
    "endTime" : null,
    "minutesBeforeAnchorTime" : 0,
    "isGlobal" : true,
    "propertiesMap" : null,
    "dataSeriesTemplates" : null,
    "adqlQueries" : [ "SELECT appkey, pageexperience, distinctcount(pageurl) AS \"URL (Count Distinct)\" FROM
browser_records LIMIT 100,100" ],
    "analyticsWidgetType" : "COLUMN",
    "maxAllowedYAxisFields" : 3,
    "maxAllowedXAxisFields" : 2,
    "min" : null,
    "interval" : 98,
    "max" : null,
    "intervalType" : "By Fixed Number",
    "showMinExtremes" : null,
    "showMaxExtremes" : null,
    "displayPercentileMarkers" : null,
    "percentileValue1" : null,
    "percentileValue2" : null,
    "percentileValue3" : null,
    "percentileValue4" : null,
    "resolution" : "1m",
    "dataFetchSize" : null,
    "percentileLine" : null,
    "timeRangeInterval" : null,
    "pollingInterval" : null,
    "unit" : null
  } ],
  "warRoom" : false,
  "template" : false
}o
```


Import Custom Dashboards and Templates

You can import custom dashboards and templates based on a previously exported JSON definition, which has optionally been modified. Import the definition as an `application/json` content type.

The user making the API call must have permission to create dashboards in the Controller.

Data for this call should be in the form of multipart/form-data. Use UTF-8 URL encoding of the URI before posting; for example, do not replace a space (" ") with "%20" in the URI.

Prior to version 4.1, exported custom dashboards were in XML format. You can import custom dashboards previously exported as XML data into the current Controller; however, custom dashboards can only be exported as JSON data.

Format

```
POST /controller/CustomDashboardImportExportServlet
```

Example

```
curl  -X POST --user user1@customer1:your_password http://demo.appdynamics.com/controller
/CustomDashboardImportExportServlet -F file=@customdashboards.json

{"success":true,"errors":[],"warnings":[],"createdDashboardName":"Uploaded-Analytics-BrowserData"}
```

Export Health Rules from an Application

Returns all health rules in XML format.

The user account you use to make the API call must have permission to view the health rule.

Format

```
GET /controller/healthrules/application_id?name=health_rule_name
```

Input parameters

Parameter Name	Parameter Type	Value	Mandatory
`application_id`	URI	The application name or application ID.	Yes
`name`	Query	The name of the health rule to export. If not specified, exports all health rules.	No

Example

```
curl --user user1@customer1:your_password http://demo.appdynamics.com/controller/healthrules/38?
name=MyCustomHealthRule

<health-rules controller-version="004-002-000-000">
    <health-rule>
        <name>MyCustomHealthRule</name>
        <type>BUSINESS_TRANSACTION</type>
        <description/>
        <enabled>true</enabled>
        <is-default>false</is-default>
        <always-enabled>true</always-enabled>
        <duration-min>30</duration-min>
        <wait-time-min>30</wait-time-min>
        <affected-entities-match-criteria>
            <affected-bt-match-criteria>
                <type>ALL</type>
            </affected-bt-match-criteria>
        </affected-entities-match-criteria>
        <warning-execution-criteria>
            <entity-aggregation-scope>
                <type>ANY</type>
                <value>0</value>
            </entity-aggregation-scope>
            <policy-condition>
                <type>leaf</type>
                <display-name>CPU</display-name>
                <condition-value-type>BASELINE_STANDARD_DEVIATION</condition-value-type>
                <condition-value>2.0</condition-value>
                <operator>GREATER_THAN</operator>
                <condition-expression/>
                <use-active-baseline>true</use-active-baseline>
                <metric-expression>
                    <type>leaf</type>
                    <function-type>VALUE</function-type>
                    <value>0</value>
                    <is-literal-expression>false</is-literal-expression>
                    <display-name>null</display-name>
                    <metric-definition>
                        <type>LOGICAL_METRIC</type>
                        <logical-metric-name>Average CPU Used (ms)</logical-metric-name>
                    </metric-definition>
                </metric-expression>
            </policy-condition>
        </warning-execution-criteria>
    </health-rule>
</health-rules>
```

Import Health Rules into an Application

You can import health rules defined in an XML file into a business application.

Data for this call should be in the form of multipart/form-data. In the POST request, use UTF-8 URL encoding for the URI; for example, do not replace a space (" ") with "%20" in the URI.

By default, a health rule in the posted data with an identical name to one in the existing configuration does not overwrite the existing health rule. If you want to overwrite an existing health rule of the same name, use the overwrite parameter.

The syntax is the same for importing one health rule configuration or several. All the health rule configurations in the posted XML file are imported.

Format

```
POST /controller/healthrules/application_id?overwrite=true_or_false
```

Input parameters

Parameter Name	Parameter Type	Value	Mandatory

application_id	URI	The application name or application ID.	Yes
overwrite	Query	Set to true to have health rules in the posted data overwrite existing health rules with the same name. The default is false.	No

Example

```
curl -X POST --user user1@customer1:your_password http://demo.appdynamics.com/controller/healthrules/38 -F
file=@uploadhealthrule.xml

Imported 1 health rules successfully.
```

If the health rule exists and you have not enabled the overwrite parameters, you will get the following response:

```
Not importing the health rule: healthrulename since it already exists.
```

Export Transaction Detection Rules

Use this to get all transaction detection rules in XML format. This call returns different types of detection rule configurations when MDS is enabled.

You can get transaction detection rules from a number of different configurations, including the configuration for a specific scope by name.

The URI used by clients for this call should be UTF-8 encoded.

Format

```
GET /controller/transactiondetection/application_id/[scope_name]/rule_type/[entry_point_type]/[rule_name]
>> xml_name.xml
```

Input parameters

Parameter Name	Parameter Type	Value	Mandatory
application_id	URI	The application name or application ID. It can be found in the URL in the address bar if you click the specified application. It is an Integer value. It will return an error if you do not specify.	Yes
scope_name	URI	The name of the scope from which you are exporting the entry point configuration. The scope name cannot be custom and auto. If the scope name contains a space, type %20 instead of space. For example: curl -X GET --user user1@customer1:welcome http://localhost:8080/controller/transactiondetection/10/default%20scope/custom >> result.xml It will export all the rules under all the scopes if you do not specify.	No
rule_type	URI	The type of rule to export, from these options: • auto: Automatic detection rules • custom: Custom detection rules in the configuration It will return an error if you do not specify.	Yes
entry_point_type	URI	The POJO, Servlet, EJB, Spring Bean, etc. It will export the rules which belong to all the entry point types if you do not specify.	No
rule_name	URI	The name of the rule which you are exporting. It will export all the rules under a scope or all the scopes if you do not specify.	No

- The order of the parameters cannot be changed. The order of the parameters should be: application_id/[scope_name]/rule_type/[entry_point_type]/[rule_name].
- It will return an XML file with error content if you use the wrong name(s) in all the parameters.
- Tier information is provided in the scope list.
- If you do not specify the scope when exporting, then you should also not specify it when importing.
- Please check the server.log file if you do not get the expected result after exporting.

Scenarios:

- Export the rules from all the scopes.

```
curl -X GET --user user1@customer1:welcome http://localhost:8080/controller/transactiondetection
/applicationID/{rule_type} >> {xml_name}.xml
```

Example:

```
curl -X GET --user user1@customer1:welcome http://localhost:8080/controller/transactiondetection/10
/custom >> result.xml
```

If you do not include `scope_name`, the output is divided into three parts under `<mds-config-data>: scope-list, rule-list,` and `scope-rule-mapping-list.`

```
<mds-data>
    <mds-config-data>
        <scope-list>
            <scope scope-description="" scope-name="scope0"
                scope-type="ALL_TIERS_IN_APP" scope-version="0"/>
            <scope scope-description="" scope-name="scope1"
                scope-type="SELECTED_TIERS" scope-version="0"/>
        </scope-list>
        <rule-list>
            <rule agent-type="APPLICATION_SERVER" enabled="true"
                priority="1"
                rule-description="ruleInScope1_SERVLET"
                rule-name="ruleInScope1_SERVLET" rule-type="TX_MATCH_RULE" version="0">
                <tx-match-rule>{"type":"CUSTOM","txautodiscoveryrule":{"autodiscoveryconfigs":[]},"
txcustomrule":{"type":"INCLUDE","txentrypointtype":"SERVLET","matchconditions":[{"type":"HTTP","
httpmatch":{"uri":{"type":"IS_NOT_EMPTY","matchstrings":[""]},"parameters":[],"headers":[],"cookies":
[]}}],"actions":[],"properties":[]},"agenttype":"APPLICATION_SERVER"}</tx-match-rule>
            </rule>
            <rule agent-type="DOT_NET_APPLICATION_SERVER"
                enabled="true" priority="0"
                rule-description="ASP.NET MVC5 Resource Handler"
                rule-name="ASP.NET MVC5 Resource Handler"
                rule-type="TX_MATCH_RULE" version="0">
                <tx-match-rule>{"type":"CUSTOM","txcustomrule":{"type":"EXCLUDE","txentrypointtype":"
POJO","matchconditions":[{"type":"HTTP","httpmatch":{"parameters":[],"headers":[],"classmatch":{"type":"
MATCHES_CLASS","classnamecondition":{"type":"EQUALS","matchstrings":["System.Web.Optimization.
BundleHandler"],"isnot":false}},"cookies":[]}}],"actions":[{"type":"HTTP_SPLIT","httpsplit":{}}],"
properties":[]},"agenttype":"DOT_NET_APPLICATION_SERVER"}</tx-match-rule>
            </rule>
            <rule agent-type="APPLICATION_SERVER"
                enabled="true" priority="0"
                rule-description="testPOJO"
                rule-name="testPOJO"
                rule-type="TX_MATCH_RULE" version="0">
                <tx-match-rule>{"type":"CUSTOM","txcustomrule":{"type":"EXCLUDE","txentrypointtype":"
POJO","matchconditions":[{"type":"HTTP","httpmatch":{"parameters":[],"headers":[],"classmatch":{"type":"
MATCHES_CLASS","classnamecondition":{"type":"EQUALS","matchstrings":["System.Web.Optimization.
BundleHandler"],"isnot":false}},"cookies":[]}}],"actions":[{"type":"HTTP_SPLIT","httpsplit":{}}],"
properties":[]},"agenttype":"APPLICATION_SERVER"}</tx-match-rule>
            </rule>
            <rule agent-type="APPLICATION_SERVER"
                enabled="true" priority="0"
                rule-description="whatever_SERVLET"
                rule-name="whatever_SERVLET"
                rule-type="TX_MATCH_RULE" version="0">
                <tx-match-rule>{"type":"CUSTOM","txcustomrule":{"type":"EXCLUDE","txentrypointtype":"
SERVLET","matchconditions":[{"type":"HTTP","httpmatch":{"parameters":[],"headers":[],"classmatch":
{"type":"MATCHES_CLASS","classnamecondition":{"type":"EQUALS","matchstrings":["System.Web.Optimization.
BundleHandler"],"isnot":false}},"cookies":[]}}],"actions":[{"type":"HTTP_SPLIT","httpsplit":{}}],"
```

```
properties":[]},"agenttype":"APPLICATION_SERVER"}</tx-match-rule>
            </rule>
        </rule-list>
        <scope-rule-mapping-list>
            <scope-rule-mapping scope-name="scope1">
                <rule rule-description="ruleInScope1_SERVLET" rule-name="ruleInScope1_SERVLET"/>
            </scope-rule-mapping>
            <scope-rule-mapping scope-name="scope0">
                <rule rule-description="ASP.NET MVC5 Resource Handler" rule-name="ASP.NET MVC5 Resource
Handler"/>
                <rule rule-description="whatever_SERVLET" rule-name="whatever_SERVLET"/>
                <rule rule-description="testPOJO" rule-name="testPOJO"/>
            </scope-rule-mapping>
        </scope-rule-mapping-list>
    </mds-config-data>
</mds-data>
```

- Export the rules under the specified scope:

```
curl -X GET --user user1@customer1:welcome http://localhost:8080/controller/transactiondetection
/applicationID/scope_name/{rule_type} >> {xml_name}.xml
```

Example:

```
curl -X GET --user user1@customer1:welcome http://localhost:8080/controller/transactiondetection/10
/scope0/custom >> result.xml
```

If you include `scope_name` then only `rule-list` is included in the output.

```
<mds-data>
    <mds-config-data>
        <rule-list>
            <rule agent-type="DOT_NET_APPLICATION_SERVER"
                enabled="true" priority="0"
                rule-description="ASP.NET MVC5 Resource Handler"
                rule-name="ASP.NET MVC5 Resource Handler"
                rule-type="TX_MATCH_RULE" version="0">
                <tx-match-rule>{"type":"CUSTOM","txcustomrule":{"type":"EXCLUDE","txentrypointtype":"
ASP_DOTNET","matchconditions":[{"type":"HTTP","httpmatch":{"parameters":[],"headers":[],"classmatch":
{"type":"MATCHES_CLASS","classnamecondition":{"type":"EQUALS","matchstrings":["System.Web.Optimization.
BundleHandler"],"isnot":false}},"cookies":[]}}],"actions":[{"type":"HTTP_SPLIT","httpsplit":{}}],"
properties":[]},"agenttype":"DOT_NET_APPLICATION_SERVER"}</tx-match-rule>
            </rule>
            <rule agent-type="APPLICATION_SERVER"
                enabled="true" priority="0"
                rule-description="whatever_SERVLET"
                rule-name="whatever_SERVLET"
                rule-type="TX_MATCH_RULE" version="0">
                <tx-match-rule>{"type":"CUSTOM","txcustomrule":{"type":"EXCLUDE","txentrypointtype":"
SERVLET","matchconditions":[{"type":"HTTP","httpmatch":{"parameters":[],"headers":[],"classmatch":
{"type":"MATCHES_CLASS","classnamecondition":{"type":"EQUALS","matchstrings":["System.Web.Optimization.
BundleHandler"],"isnot":false}},"cookies":[]}}],"actions":[{"type":"HTTP_SPLIT","httpsplit":{}}],"
properties":[]},"agenttype":"APPLICATION_SERVER"}</tx-match-rule>
            </rule>
            <rule agent-type="APPLICATION_SERVER"
                enabled="true" priority="0"
                rule-description="testPOJO"
                rule-name="testPOJO"
                rule-type="TX_MATCH_RULE" version="0">
                <tx-match-rule>{"type":"CUSTOM","txcustomrule":{"type":"EXCLUDE","txentrypointtype":"
POJO","matchconditions":[{"type":"HTTP","httpmatch":{"parameters":[],"headers":[],"classmatch":{"type":"
MATCHES_CLASS","classnamecondition":{"type":"EQUALS","matchstrings":["System.Web.Optimization.
BundleHandler"],"isnot":false}},"cookies":[]}}],"actions":[{"type":"HTTP_SPLIT","httpsplit":{}}],"
properties":[]},"agenttype":"APPLICATION_SERVER"}</tx-match-rule>
            </rule>
```

```
        </rule-list>
    </mds-config-data>
</mds-data>
```

- Export the rules belonging to the specified entry point under all the scopes.

```
curl -X GET --user user1@customer1:welcome http://localhost:8080/controller/transactiondetection
/applicationID/{rule_type}/{entry_point_type} >> {xml_name}.xml
```

Example:

```
curl -X GET --user user1@customer1:welcome http://localhost:8080/controller/transactiondetection/10
/custom/servlet >> {xml_name}.xml
```

If you do not include `scope_name` then the output is divided into three parts under `<mds-config-data>: scope-list, rule-list, and scope-rule-mapping-list.`

```
<mds-data>
    <mds-config-data>
        <scope-list>
            <scope scope-description="" scope-name="scope0"
                scope-type="ALL_TIERS_IN_APP" scope-version="0"/>
            <scope scope-description="" scope-name="scope1"
                scope-type="SELECTED_TIERS" scope-version="0"/>
        </scope-list>
        <rule-list>
            <rule agent-type="APPLICATION_SERVER" enabled="true"
                priority="1"
                rule-description="ruleInScope1_SERVLET"
                rule-name="ruleInScope1_SERVLET" rule-type="TX_MATCH_RULE" version="0">
                <tx-match-rule>{"type":"CUSTOM","txautodiscoveryrule":{"autodiscoveryconfigs":[]},"
txcustomrule":{"type":"INCLUDE","txentrypointtype":"SERVLET","matchconditions":[{"type":"HTTP","
httpmatch":{"uri":{"type":"IS_NOT_EMPTY","matchstrings":[""]},"parameters":[],"headers":[],"cookies":
[]}}],"actions":[],"properties":[]},"agenttype":"APPLICATION_SERVER"}</tx-match-rule>
            </rule>
            <rule agent-type="APPLICATION_SERVER"
                enabled="true" priority="0"
                rule-description="whatever_SERVLET"
                rule-name="whatever_SERVLET"
                rule-type="TX_MATCH_RULE" version="0">
                <tx-match-rule>{"type":"CUSTOM","txcustomrule":{"type":"EXCLUDE","txentrypointtype":"
SERVLET","matchconditions":[{"type":"HTTP","httpmatch":{"parameters":[],"headers":[],"classmatch":
{"type":"MATCHES_CLASS","classnamecondition":{"type":"EQUALS","matchstrings":["System.Web.Optimization.
BundleHandler"],"isnot":false}},"cookies":[]}}],"actions":[{"type":"HTTP_SPLIT","httpsplit":{}}],"
properties":[]},"agenttype":"APPLICATION_SERVER"}</tx-match-rule>
            </rule>
        </rule-list>
        <scope-rule-mapping-list>
            <scope-rule-mapping scope-name="scope1">
                <rule rule-description="ruleInScope1_SERVLET" rule-name="ruleInScope1_SERVLET"/>
            </scope-rule-mapping>
            <scope-rule-mapping scope-name="scope0">
                <rule rule-description="whatever_SERVLET" rule-name="whatever_SERVLET"/>
            </scope-rule-mapping>
        </scope-rule-mapping-list>
    </mds-config-data>
</mds-data>
```

- Export the rules belonging to the specified entry point under the specified scope.

```
curl -X GET --user user1@customer1:welcome http://localhost:8080/controller/transactiondetection
/applicationID/scope_name/{rule_type}/{entry_point_type} >> {xml_name}.xml
```

Example:

```
curl -X GET --user user1@customer1:welcome http://localhost:8080/controller/transactiondetection/10
/scope0/custom/servlet >> {xml_name}.xml
```

```
<mds-data>
    <mds-config-data>
        <rule-list>
            <rule agent-type="APPLICATION_SERVER" enabled="true"
                priority="1"
                rule-description="ruleInScope1_SERVLET"
                rule-name="ruleInScope1_SERVLET" rule-type="TX_MATCH_RULE" version="0">
                <tx-match-rule>{"type":"CUSTOM","txautodiscoveryrule":{"autodiscoveryconfigs":[]},"
txcustomrule":{"type":"INCLUDE","txentrypointtype":"SERVLET","matchconditions":[{"type":"HTTP","
httpmatch":{"uri":{"type":"IS_NOT_EMPTY","matchstrings":[""]},"parameters":[],"headers":[],"cookies":
[]}}],"actions":[],"properties":[]},"agenttype":"APPLICATION_SERVER"}</tx-match-rule>
            </rule>
        </rule-list>
    </mds-config-data>
</mds-data>
```

- Export the single rule belonging to the specified entry point under the specified scope.

```
curl -X GET --user user1@customer1:welcome http://localhost:8080/controller/transactiondetection
/applicationID/scope_name/{rule_type}/{entry_point_type}/{rule_name} >> {xml_name}.xml
```

Example:

```
curl -X GET --user user1@customer1:welcome http://localhost:8080/controller/transactiondetection/10
/scope0/custom/servlet/rule_name >> {xml_name}.xml
```

```
<mds-data>
    <mds-config-data>
        <rule-list>
            <rule agent-type="APPLICATION_SERVER" enabled="true"
                priority="1" rule-description=""
                rule-name="ruleInScope1" rule-type="TX_MATCH_RULE" version="0">
                <tx-match-rule>{"type":"CUSTOM","txautodiscoveryrule":{"autodiscoveryconfigs":[]},"
txcustomrule":{"type":"INCLUDE","txentrypointtype":"SERVLET","matchconditions":[{"type":"HTTP","
httpmatch":{"uri":{"type":"IS_NOT_EMPTY","matchstrings":[""]},"parameters":[],"headers":[],"cookies":
[]}}],"actions":[],"properties":[]},"agenttype":"APPLICATION_SERVER"}</tx-match-rule>
            </rule>
        </rule-list>
    </mds-config-data>
</mds-data>
```

Import Transaction Detection Rules

Use this to import automatic detection rules in XML format. This call returns different types of detection rule configurations when MDS is enabled.

Importing action will overwrite a rule or add the new rule to the all/specified scopes.

Data for this call should be in the form of multipart/form-data. The URI for this call should be UTF-8 encoded.

Format

```
POST /controller/transactiondetection/application_id/[scope_name]/rule_type/[entry_point_type]/[rule_name] -F
file=@exported_file_name.xml
```

Input parameters

Parameter Name	Parameter Type	Value	Mandatory

`application_id`	URI	The application name or application ID. It can be found in the URL in address bar if you click the specified application. It is an Integer value. It will return an error if you do not specify.	Yes
`scope_name`	URI	The name of the scope from which you are importing the entry point configuration. It will import the rules to all the scopes if you do not specify.	No
`rule_type`	URI	The type of rule to import, from these options: • auto: Automatic detection rules • custom: Custom detection rules in the configuration • exclude: Custom exclude rules for transaction detection It will return an error if you do not specify.	Yes
`entry_point_type`	URI	The POJO, Servlet, EJB, Spring Bean, etc	No
`rule_name`	URI	The name of the rule which you are importing. It will import all the rules under a scope or all the scopes if you do not specify.	No

- When you import, if the XML file does not have `<scope-list>`, then you should type the scope name, otherwise, it will fail, and vice versa.
- It will return an error if you use the wrong name(s) in all the parameters.
- If you do not specify the scope when importing, then you should not specify it when exporting.
- Check the `server.log` file if you do not get the expected result after exporting.

Examples

- Import the rule(s) to the application without scopes specified:

```
curl -X POST --user user1@customer1:welcome http://localhost:8080/controller/transactiondetection/
{application_id}/{rule_type} -F file=@{exported_file_name}.xml
```

- Import the rule(s) to the application with scopes specified:

```
curl -X POST --user user1@customer1:welcome http://localhost:8080/controller/transactiondetection/
{application_id}/{scope_name}/{rule_type} -F file=@{exported_file_name}.xml
```

- For more scenarios to import, see Export Transaction Detection Rules.

Export Policies

You can export policies to a JSON file. Before you export policies, export any actions or health rules with their respective APIs.

Format

`GET /controller/policies/application_id`

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory
`application_id`	URI	The application name or application ID.	Yes

Example

```
curl --user user1@customer1:your_password http://demo.appdynamics.com/controller/policies/application_id

[ {
  "applicationName" : "ECommerce-Books",
  "name" : "My Policy",
  "reactorType" : "IMMEDIATE",
  "enabled" : true,
  "batchActionsPerMinute" : true,
  "durationInMin" : 1,
```

```
  "eventFilterTemplate" : {
    "applicationName" : "ECommerce-E2E",
    "healthRuleNames" : null,
    "eventTypes" : [ "POLICY_OPEN_WARNING", "POLICY_OPEN_CRITICAL", "POLICY_CONTINUES_WARNING",
"POLICY_CONTINUES_CRITICAL" ],
    "rsdTypes" : null,
    "customEventFilters" : null,
    "specificEntityNamesByType" : null
  },
  "entityFilterTemplates" : [ ],
  "actionWrapperTemplates" : [ {
    "actionTag" : "ops_viewer@acme.com",
    "type" : null,
    "value" : 0,
    "notes" : "Policy: My Policy",
    "entityIdentifierTemplates" : [ ]
  } ]
} ]
```

Import Policies

You can import policies that you exported with the Export Policies API. Before you import policies, import any actions or health rules with their respective APIs.

You can import a policy after modifying the defined parameter(s) and overwrite the existing policy with the updated one.

Format

`POST /controller/policies/application_id ?overwrite=true_or_false`

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory
`application_id`	URI	The application name or application ID.	Yes
`overwrite`	Query	Set to true to have updates to a policy overwrite the existing policy with the same name. The default is `false` .	No

Example

```
curl  -X POST --user user1@customer1:your_password http://demo.appdynamics.com/controller/policies/38 -F
file=@ImportPolicies.json

{"success":true,"errors":[],"warnings":[]}
```

Example to Overwrite a Policy

```
curl  -X POST --user user1@customer1:your_password http://demo.appdynamics.com/controller/policies/38\?
overwrite\=true -F file=@ImportPolicies.json

{"success":true,"errors":[],"warnings":[]}
```

Export Application Analytics Dynamic Service Configuration

The Analytics Dynamic Service is an AppDynamics app agent plugin that performs Analytics client functions for the agent. Enabling the Dynamic Service enables AppDynamics Analytics for an app agent type. You can export the Dynamic Service configuration to back up the configuration or for later import into another Controller.

Format

`GET /controller/analyticsdynamicservice/application_id`

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory
application_id	URI	The application name or application ID.	Yes
filename	Query	The name of a file to which the configuration will be exported.	No

Example

```
curl -i --user user1@customer1:your_password http://demo.appdynamics.com/controller/analyticsdynamicservice/10

<analytics-dynamic-service-configurations controller-version="004-003-000-000">
    <analytics-dynamic-service-configuration>
        <override>true</override>
        <agent-type>APP_AGENT</agent-type>
        <enabled>true</enabled>
    </analytics-dynamic-service-configuration>
    <analytics-dynamic-service-configuration>
        <override>true</override>
        <agent-type>DOT_NET_APP_AGENT</agent-type>
        <enabled>true</enabled>
    </analytics-dynamic-service-configuration>
    <analytics-dynamic-service-configuration>
        <override>true</override>
        <agent-type>NODEJS_APP_AGENT</agent-type>
        <enabled>true</enabled>
    </analytics-dynamic-service-configuration>
</analytics-dynamic-service-configurations>
```

Import Application Analytics Dynamic Service Configuration

The Analytics Dynamics Service configuration determines whether AppDynamics Analytics is enabled for an app agent type. You use this API to import a previously exported configuration to another Controller.

Data for this call should be in the form of multipart/form-data.

Format

```
POST /controller/analyticsdynamicservice/application_id
```

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory
application_id	URI	The name or ID identifier of the application to which the Analytics Dynamic Service configuration should be applied.	Yes

Example

```
curl -i -X POST --user user1@customer1:your_password http://demo.appdynamics.com/controller
/analyticsdynamicservice/10 -F file=@dynamicservice.xml
```

The following example shows sample contents of the `dynamicservice.xml` file. Notice that the `agent-type` element indicates the type of app server agent to which the enabled state of the dynamics service applies. The `APP_AGENT` type represents the Java Agent, the `DOT_NET_APP_AGENT` the .NET Agent, and so on.

```
<analytics-dynamic-service-configurations controller-version="004-003-000-000">
    <analytics-dynamic-service-configuration>
        <override>true</override>
        <agent-type>APP_AGENT</agent-type>
        <enabled>false</enabled>
    </analytics-dynamic-service-configuration>
    <analytics-dynamic-service-configuration>
        <override>true</override>
        <agent-type>DOT_NET_APP_AGENT</agent-type>
        <enabled>false</enabled>
    </analytics-dynamic-service-configuration>
```

```
    <analytics-dynamic-service-configuration>
        <override>true</override>
        <agent-type>NODEJS_APP_AGENT</agent-type>
        <enabled>true</enabled>
    </analytics-dynamic-service-configuration>
</analytics-dynamic-service-configurations>
```

Database Visibility API

This page describes the Database Visibility API methods you can use to get, create, update, and delete Database Visibility Collectors.

Include the following headers for all Database Visibility API requests:

```
Accept: application/json; Content-type: application/json
```

JSON is currently the only supported format.

Get All Collectors

```
GET /controller/rest/databases/collectors
```

Get a Specific Collector

```
GET /controller/rest/databases/collectors/{configurationId}
```

Create a Collector

```
POST /controller/rest/databases/collectors/create
```

The JSON you send must contain the relevant Collector information. The required fields describing the Collector vary based on the type of database. See UI Collector versus JSON Collector Configuration Field Names

Example JSON Request

```
{
    "type":"MYSQL",
    "name":"localdocker_dbagent-MySQLCollector",
    "hostname":"mysql",
    "port":"3306",
    "username":"root",
    "password":"appdynamics_redacted_password",
    "enabled":true,
    "excludedSchemas":null,
    "databaseName":null,
    "failoverPartner":null,
    "connectAsSysdba":false,
    "useServiceName":false,
    "sid":null,
    "customConnectionString":null,
    "enterpriseDB":false,
    "useSSL":false,
    "enableOSMonitor":false,
    "hostOS":null,
    "useLocalWMI":false,
    "hostDomain":null,
    "hostUsername":null,
    "hostPassword":"",
    "dbInstanceIdentifier":null,
    "region":null,
    "certificateAuth":false,
    "removeLiterals":true,
    "sshPort":0,
    "agentName":"localdocker_dbagent",
    "dbCyberArkEnabled":false,
    "dbCyberArkApplication":null,
    "dbCyberArkSafe":null,
    "dbCyberArkFolder":null,
    "dbCyberArkObject":null,
```

```
    "hwCyberArkEnabled":false,
    "hwCyberArkApplication":null,
    "hwCyberArkSafe":null,
    "hwCyberArkFolder":null,
    "hwCyberArkObject":null,
    "orapkiSslEnabled":false,
    "orasslClientAuthEnabled":false,
    "orasslTruststoreLoc":null,
    "orasslTruststoreType":null,
    "orasslTruststorePassword":"",
    "orasslKeystoreLoc":null,
    "orasslKeystoreType":null,
    "orasslKeystorePassword":"",
    "ldapEnabled":false,
    "customMetrics":null,
    "subConfigs":[
        {
            "type":"MYSQL",
            "name":"localdocker_dbagent-MySQLCollector sub-collector",
            "hostname":"mysql-remote",
            "port":"3388",
            "username":"root",
            "password":"different-password",
            "enabled":true,
            "excludedSchemas":null,
            "databaseName":null,
            "failoverPartner":null,
            "connectAsSysdba":false,
            "useServiceName":false,
            "sid":null,
            "customConnectionString":null,
            "enterpriseDB":false,
            "useSSL":false,
            "enableOSMonitor":false,
            "hostOS":null,
            "useLocalWMI":false,
            "hostDomain":null,
            "hostUsername":null,
            "hostPassword":"",
            "dbInstanceIdentifier":null,
            "region":null,
            "certificateAuth":false,
            "removeLiterals":true,
            "sshPort":0,
            "agentName":"localdocker_dbagent",
            "dbCyberArkEnabled":false,
            "dbCyberArkApplication":null,
            "dbCyberArkSafe":null,
            "dbCyberArkFolder":null,
            "dbCyberArkObject":null,
            "hwCyberArkEnabled":false,
            "hwCyberArkApplication":null,
            "hwCyberArkSafe":null,
            "hwCyberArkFolder":null,
            "hwCyberArkObject":null,
            "orapkiSslEnabled":false,
            "orasslClientAuthEnabled":false,
            "orasslTruststoreLoc":null,
            "orasslTruststoreType":null,
            "orasslTruststorePassword":"",
            "orasslKeystoreLoc":null,
            "orasslKeystoreType":null,
            "orasslKeystorePassword":"",
            "ldapEnabled":false,
            "customMetrics":null
        }
    ]
}
```

extraProperties: To configure the frequency for sampling queries from the database (except Cassandra), you can use the `extraProperties` parameter in the JSON request payload. The following JSON sample can be used in the request payload to configure the sampling interval:

```
{
    "type":"POSTGRESQL",
    "agentName":"UpgradeTest204",
    "name":"SamplingInterval",
    "hostname":"ec2-54-202-140-213.us-west-2.compute.amazonaws.com",
    "port":"5432",
    "username":"postgres",
    "password":"Appd123",
    "removeLiterals":"true",
    "enableOSMonitor":"true",
    "hostOS":"LINUX",
    "sshPort":"22",
    "hostUsername":"ec2-user",
    "hostPassword":"",
    "certificateAuth":true,
    "enabled":"true",
    "enterpriseDB":"false",
    "extraProperties":[
        {
            "key":"dbagent.sampling.interval",
            "value":"10",
            "sensitive":false
        }
    ]
}
```

The following JSON key-value pair is used in `extraProperties` for configuring the interval:

key: `dbagent.sampling.interval`

value: positive integer

The value can be 1, 2, 5, 10, 20, or 30.

You can configure this property through the agent. See Configure the Agent Settings for Monitoring Database.

Update a Collector

1. Make a `GET` request for the collector that you want to update.
2. Copy the JSON response body that is returned by the `GET` request to a text editor, and modify the fields that you want to update.
3. Make a `POST` request for the collector that you want to update, and include the updated JSON.

```
POST /controller/rest/databases/collectors/update
```

Example JSON Request

```
{
    "id":1,
    "type":"MYSQL",
    "name":"localdocker_dbagent-MySQLCollector",
    "hostname":"mysql",
    "port":"3306",
    "username":"root",
    "password":"appdynamics_redacted_password",
    "enabled":true,
    "excludedSchemas":null,
    "databaseName":null,
    "failoverPartner":null,
    "connectAsSysdba":false,
    "useServiceName":false,
    "sid":null,
    "customConnectionString":null,
    "enterpriseDB":false,
    "useSSL":false,
```

```
"enableOSMonitor":false,
"hostOS":null,
"useLocalWMI":false,
"hostDomain":null,
"hostUsername":null,
"hostPassword":"",
"dbInstanceIdentifier":null,
"region":null,
"certificateAuth":false,
"removeLiterals":true,
"sshPort":0,
"agentName":"localdocker_dbagent",
"dbCyberArkEnabled":false,
"dbCyberArkApplication":null,
"dbCyberArkSafe":null,
"dbCyberArkFolder":null,
"dbCyberArkObject":null,
"hwCyberArkEnabled":false,
"hwCyberArkApplication":null,
"hwCyberArkSafe":null,
"hwCyberArkFolder":null,
"hwCyberArkObject":null,
"orapkiSslEnabled":false,
"orasslClientAuthEnabled":false,
"orasslTruststoreLoc":null,
"orasslTruststoreType":null,
"orasslTruststorePassword":"",
"orasslKeystoreLoc":null,
"orasslKeystoreType":null,
"orasslKeystorePassword":"",
"ldapEnabled":false,
"customMetrics":null,
"subConfigs":[
    {
        "id":2,
        "type":"MYSQL",
        "name":"localdocker_dbagent-MySQLCollector sub-collector",
        "hostname":"mysql",
        "port":"3388",
        "username":"root",
        "password":"appdynamics-redacted-password",
        "enabled":true,
        "excludedSchemas":null,
        "databaseName":null,
        "failoverPartner":null,
        "connectAsSysdba":false,
        "useServiceName":false,
        "sid":null,
        "customConnectionString":null,
        "enterpriseDB":false,
        "useSSL":false,
        "enableOSMonitor":false,
        "hostOS":null,
        "useLocalWMI":false,
        "hostDomain":null,
        "hostUsername":null,
        "hostPassword":"",
        "dbInstanceIdentifier":null,
        "region":null,
        "certificateAuth":false,
        "removeLiterals":true,
        "sshPort":0,
        "agentName":"localdocker_dbagent",
        "dbCyberArkEnabled":false,
        "dbCyberArkApplication":null,
        "dbCyberArkSafe":null,
        "dbCyberArkFolder":null,
        "dbCyberArkObject":null,
        "hwCyberArkEnabled":false,
        "hwCyberArkApplication":null,
        "hwCyberArkSafe":null,
```

```
            "hwCyberArkFolder":null,
            "hwCyberArkObject":null,
            "orapkiSslEnabled":false,
            "orasslClientAuthEnabled":false,
            "orasslTruststoreLoc":null,
            "orasslTruststoreType":null,
            "orasslTruststorePassword":"",
            "orasslKeystoreLoc":null,
            "orasslKeystoreType":null,
            "orasslKeystorePassword":"",
            "ldapEnabled":false,
            "customMetrics":null
        }
    ]
}
```

The JSON you send must contain all the details of the existing collector with only the fields that you want to modify changed. To ensure you have all the fields, use the Get a Specific Collector call.

To add a new sub-collector to an existing collector, provide the sub-collector details without the `id` field.

> To configure the interval for sampling queries, refer to the `extraProperties` parameter in the JSON request payload as mentioned in extraProperties under Create a Collector.

Delete a Specific Collector

```
DELETE /controller/rest/databases/collectors/{configurationId}
```

Example Delete Request

```
DELETE /controller/rest/databases/collectors/{1}
```

Batch Delete Multiple Collectors

```
POST /controller/rest/databases/collectors/batchDelete
```

Send an array of the configuration Ids of the Collectors.

Below is an example of a batch delete command.

```
curl --user {username}@{account_name}:{password} -H "Accept: application/json" -H "Content-type: application
/json" -X POST -d '[1,2,3]' {Controller_URL}/controller/rest/databases/collectors/batchDelete
```

Get All Monitored Database Servers

```
GET /controller/rest/databases/servers
```

Example

```
curl --user {username}@{account_name}:{password} {Controller_URL}/controller/rest/databases/servers
```

The output is a list of database servers and their details.

Get Database Server Details

```
GET /controller/rest/databases/servers/{dbserver_id}
```

Example

```
curl --user {username}@{account_name}:{password} {Controller_URL}/controller/rest/databases/servers/
{dbserver_id}
```

The output contains a list of the database's details, including name, node ID, and database type.

Get all Database Agent Events

```
GET /controller/rest/applications/_dbmon/events
```

For a list of query string parameters, see Retrieve Event Data.

Example

```
curl --user {username}@{account_name}:{password} {Controller_URL}/controller/rest/applications/_dbmon/events?
time-range-type=BEFORE_NOW&duration-in-mins=30&event-types=%20AGENT_EVENT,
DB_SERVER_PARAMTER_CHANGE&severities=INFO,WARN,ERROR
```

The output gives you a list of events. For each event element, you can determine the node that the event is mapped to by looking for the entity-definition element.

Get all Database Monitoring Application Nodes

```
GET /controller/rest/applications/_dbmon/nodes
```

Example

```
curl --user {username}@{account_name}:{password} {Controller_URL}/controller/rest/applications/_dbmon/nodes
```

UI Collector versus JSON Collector Configuration Field Names

Use the table below to ensure you use the correct field names for your API calls. The Collector configuration field names are described in Configure the Database Agent to Monitor Server Hardware and Add Database Collectors.

Section	UI Collector Configuration Field Name	JSON Collector Configuration Field Name
		id (AppDynamics assigns this ID to the Collector when you configure the Collector. You need this ID when doing a batch delete.)
	Database Type	`type`
	Database Agent	`agentName`
	Database	`name`
Connection Details	**Hostname/IP Address**	`hostname`
	EnterpriseDB	`enterpriseDB`
	Failover Partner	`failoverPartner`
	Listener Port	`port`

	Custom JDBC Connection String	`customConnectionString`
	Use Service Name	`useServiceName`
	SID or SERVICE_NAME	`sid`
	Connect as a sysdba	`connectAsSysdba`
	Username	`username`
	Password	`password`
	Logging Enabled	
Hardware Monitoring	**Monitor Operating System**	`enableOSMonitor`
	Operating System	`hostOS`
	Use Local WMI	`useLocalWMI`
	Domain	`hostDomain`
	SSH Port	`sshPort`
	Use certificate	`certificateAuth`
	Username	`hostUsername`
	Password	`hostPassword`

SSL field

In addition to JSON Configuration Fields, there is the SSL field. SSL is a configurable property for the Database Agent. If the Database Agent has been configured to use SSL, then you must also provide the SSL field and its value in your Database Visibility API calls.

Get Health Rule Violations for a Collector

```
GET /controller/rest/databases/servers/healthrule-violations/<server-id>
```

Input parameters

Parameter Name	Parameter Type	Value	Mandatory
`server_id`	URI	Provide the databse server name or ID.	Yes

`time-range-type`	Query	Possible values are:
		`BEFORE_NOW`
		To use the "`BEFORE_NOW`" option, you must also specify the "`duration-in-mins`" parameter.
		`BEFORE_TIME`
		To use the "`BEFORE_TIME`" option, you must also specify the "`duration-in-mins`" and "`end-time`" parameters.
		`AFTER_TIME`
		To use the "`AFTER_TIME`" option, you must also specify the "`duration-in-mins`" and "`start-time`" parameters.
		
		`BETWEEN_TIMES`
		
		To use this option, you must also specify the "`start-time`" and "`end-time`" parameters. The "`BETWEEN_TIMES`" range includes the start-time and excludes the end-time.
			Yes

`duration-in-mins`	Query	Duration (in minutes) to return the metric data.	If time-range-type is `BEFORE_NOW, BEFORE_TIME, or AFTER_TIME`
`start-time`	Query	Start time (in milliseconds) from which the metric data is returned.	If time-range-type is `AFTER_TIME` or `BETWEEN_TIMES`
`end-time`	Query	End time (in milliseconds) until which the metric data is returned.	If time-range-type is `BEFORE_TIME` or `BETWEEN_TIMES`
`output`	Query	HTTP Request parameter included as part of the URL to change the output format. Valid values are `XML` (default) or `JSON`.	No

Example

```
curl --user {username}@{account_name}:{password} {Controller_URL}/controller/rest/databases/servers/healthrule-violations/<server-id>?time-range-type=BEFORE_NOW&duration-in-mins=<required_duration>
```

Analytics Events API

AppDynamics Analytics provides built-in support for collecting analytics data from various types of sources, such as agent-instrumented Java applications, .NET applications, browser applications, and more. The Analytics Events API supplements built-in analytics data sources with your custom data sources and event types.

In addition to the transaction events published by the app agents, custom events collected by the Analytics Events API are metered.

Publishing custom events requires Transaction Analytics licensing. Transaction Analytics license units determine the limit on the volume of custom e publish.

About the Analytics Events API

In Analytics, an event encapsulates a unit of analytics data. In APM, for example, each event corresponds to a method or service invocation, whether it be an entry point service or downstream service.

With the Analytics API, you define the structure of your own custom event in the data store, capture the event records as they occur in your custom source, and send them to the Events Service, the data store for Analytics. Once your data is in the Events Store, users can query your data through the Controller UI or the Analytics Events API.

Analytics Events API uses a shared key to authenticate clients to the Events Service. As a Controller or Analytics Administrator, you can generate API keys from the Controller UI. See Manage API Keys.

A Transaction Analytics license is required to use the Events Service API. The same licensing model that applies to business transactions for custom events (based on the number of events per unit/day) applies to the API.

Addressing the Events Service Data Store

Unlike most AppDynamics REST APIs, which are presented at the Controller, you access the Analytics Events API by addressing the Events Service instance in the AppDynamics platform.

You address the Events Service at one of the following URLs:

For an on-premises Events Service, address the Events Service instance host (or more likely, the virtual IP presented by a load balancer for the Events Service cluster). Use the primary default listening port for the Events Service, 9080.

Calls to the Analytics Events API need to specify the <global_accountp_name> for the Controller account being address, and the <api_key> generated by the administrator for this client. The API expects the values as headers, property-value pairs that are separated by a colon. As cURL arguments, for example, the values would be passed with curl through the `-H` or `--header option` as follows:

```
-H"X-Events-API-AccountName:<global_account_name>" -H"X-Events-API-Key:<api_key>"
```

You can get the global account name to use from the License page in the Controller UI. The API keys are described in Manage API Keys.

The content type, also as a cURL argument, is:

```
-H"Content-type: application/vnd.appd.events+json;v=2"
```

AppDynamics strongly recommends the use of SSL/HTTPS to access the API. Otherwise, the API key is sent in plain text.

For security reasons, the Analytics Events API, by default, does not accept cross-origin HTTP requests. For example, from links embedded directly in web pages.

Data Format

The Analytics Events API takes events as JSON-formatted name-value pair data.

Before sending data that conforms to a custom events schema, you need to define the data structure for the custom schema. The Events Service matches incoming events data to the appropriate schema.

Supported Data Types

- Boolean
- Date – Supported time formats include:
  - ISO 8601 format: `yyyy-MM-dd'T'HH:mm:ss.SSSZZ`.
  - UNIX epoch date format: A 13-digit number representing the number of seconds/milliseconds since UNIX epoch time (Jan 1, 1970). For example, (GMT): Mon, 17 Apr 2017 23:46:22 GMT would be 1492472782000.
- Float
- Integer
- Object
- String

Naming Restrictions

Custom event names and field names must conform to the following:

- Contain only a-z, A-Z, _ (underscore), 0-9.
- Names can not start with a number.

Timestamp Fields

Two implicit timestamp fields are automatically added to custom schemas:

- `eventTimestamp`
- `pickupTimestamp`

The `eventTimestamp` field represents the time an event occurred. An API client can specify a value for the timestamp field when it creates an event. If it does not, the Analytics Events API uses the same value for `eventTimestamp` as it uses for another implicit field, `pickupTimestamp`.The `pickupTimestamp` field, which is always populated by the Events Service, represents the time the event was received by the Events Service.

When configuring a milestone in Business Journeys, you need to supply a date in the `eventTimestamp` field in order for the event to register. In order to use Custom Events to define Business Journey, the custom events should send the `eventTimestamp` field. If not, the Business Journeys will not work.

AppDynamics uses another timestamp, `eventcompletionTimestamp`, which is not automatically added to custom schemas. The `eventcompletionTimestamp` field represents the time an event ends. Generally, AppDynamics uses this field internally to accurately report long-running events.

You can express all timestamp fields using ISO 8601 or UNIX epoch time (64-bit milliseconds) format.

Example API Call Flow

The following steps take you through an on-premises API call workflow for using the Analytics Events API. The steps show cURL examples for creating a schema, publishing an event to that schema, and then querying the event.

For SaaS deployments, replace the value for the URL and port in the examples (`<events_service_endpoint>:9080`) to your SaaS URL.

1. Define the schema by associating field names with data types. For example, the following defines a Purchase event type:

```
curl -X POST "<events_service_endpoint>:9080/events/schema/myProducts" -H"X-Events-API-AccountName:
customer1_1234-567a-bccc-123" -H"X-Events-API-Key:a123b456-c789-1d23-e456-nnn" -H"Content-type:
application/vnd.appd.events+json;v=2" -d '{"schema" : { "id": "string", "productBrand": "string",
"userRating": "integer", "price": "float", "productName": "string", "description": "string" } }'
```

2. Publish an event based on the schema you created:

```
curl -X POST "<events_service_endpoint>:9080/events/publish/myProducts" -H"X-Events-API-AccountName:
customer1_1234-567a-bccc-123" -H"X-Events-API-Key:a123b456-c789-1d23-e456-nnn" -H"Content-type:
application/vnd.appd.events+json;v=2" -d '[{"id": "5653b879ab33a","productBrand": "ACME","userRating":
3,"price": 2006.41,"productName": "Watch","description": "new watch"},{"id": "5653b879700","
productBrand": "Widget","userRating": 1,"price": 3800.13,"productName": "Watch","description": "2015
watch"}]'
```

3. Query the event data:

```
curl -X POST "http://<events_service_endpoint>:9080/events/query" -H"X-Events-API-AccountName:
customer1_7xxx-467a-bccc-xxx" -H"X-Events-API-Key:a123b456-c789-1d23-e456-nnn" -H"Content-type:
application/vnd.appd.events+text;v=2" -d 'SELECT * FROM myProducts'
```

If including fields with ADQL keywords, enclose the keywords in single quotes. These keywords include, for example, `between`, `in`, `select`, and others.

For a single query request, use this content type:

```
-H"Content-type: application/vnd.appd.events+text;v=2"
```

In a multi-query request, the queries are passed as JSON body text. In this case, use the following content type header:

```
-H"Content-type: application/vnd.appd.events+json;v=2"
```

Custom Event Ingestion Limits

Controller ingestion of custom events has the following limits:

- Fields: 255 maximum per event type
- String attributes: 4 kb maximum length
- Batch total count: 1000 events per call
- Batch total size: 5 Mb maximum per call
- Max custom events for an account: 20

Custom Events Limit Increase

> Custom events limit increases are available for SaaS Events Service version 4.5.13 and later. On-premises Events Services cannot exceed a maximum of 1500 custom field events.

By default, the Events Service allows you to create up to 1500 custom fields. If you reach 80% capacity of custom fields (1200 fields), the maximum field limit increases by 500 fields at the next index rollover.

For example, if you reach 1400 custom fields, your maximum field limit will increase from 1500 to 2000 fields. If you later reach 1800 custom fields, and maximum increases to 2500 fields.

You can create an absolute maximum of 3000 custom fields. Reaching or exceeding the maximum fields could result in lag or outages.

Publish Events

The Publish Events API call takes an array of events and stores them in the Events Service storage. The data must comply with an existing schema. A single request cannot publish to multiple event types.

If the event data doesn't match an event schema, the Events Service makes a best-effort attempt to match the data to the schema and returns a 400 bad request if unsuccessful.

Format

Query Params

N/A

Path Parameters

Name	Description
`accountId`	Account ID
`schemaName`	Event schema name

Headers

Name	Description
`X-Events-API-AccountName`	The global account name, as shown in the Controller UI **License** page.
`X-Events-API-Key`	The Analytics API key. See Manage API Keys
`Content-Type`	The `Content-Type` of the request body. The default is `application/vnd.appd.events+json;v=2` which also versions the resource representation (v=2).

Example SaaS Publish

Error Codes

Error Code	Description
400	The given request was invalid.
401	The given authentication information provided in the authorization header was invalid.
404	No event type could be found for this account.
406	The `Accept` header was not `application/vnd.appd.events+json;v=2` .
413	The request body is larger than the max allowed size.
415	The `Content-Type` header was not `application/vnd.appd.events+json;v=2` .
429	Too many requests. Returned when `account` or `event` reaches limits.

Create Event Schema

Use this API to create your own event schema. The schema defines the overall structure of an event type by field and type.

You only need to use this API if the event you are uploading does not match an existing schema for first-class event types (such as logs or transactions). Events that conform to an existing schema automatically match that schema. Review the supported data types and naming restrictions described prior to this topic.

Format

Path Params

Name	Description
`accountId`	Account ID
`schemaName`	Event schema name

Query Params

N/A

Headers

Name	Description
`X-Events-API-AccountName`	The global account name, as shown in the Controller UI **License** page.
`X-Events-API-Key`	The Analytics API key. See Manage API Keys
`Accept`	The `Content-Type` of the response body. The supported value is `application/vnd.appd.events+json;v=2`.
`Content-type`	The `Content-Type` of the request body. The default is `application/vnd.appd.events+json;v=2` which also versions the resource representation (v=2).

Example SaaS Create

Retrieve Event Schema

Use this API to retrieve an existing event schema.

Format

Path Params

Name	Description
`accountId`	Account ID
`schemaName`	Event schema name

Query Params

N/A

Headers

Name	Description
`X-Events-API-AccountName`	The global account name, as shown in the Controller UI **License** page.
`X-Events-API-Key`	The Analytics API key. See Manage API Keys
`Accept`	The `Content-Type` of the response body. The supported value is `application/vnd.appd.events+json;v=2`.

Example SaaS Retrieve

Update Event Schema

Use this API to update an existing event schema by field. The request body defines the updates to be applied to the event schema.

As shown in the following example, you specify each field update action as a named section in the request body. The actions are represented by these fields:

- Add field
- Rename field

For the Add field definition, you need to specify the data format for the new field, as you would when creating the event schema.

The response to this call should be the complete event schema as modified.

Format

Path Parameters

Name	Description
`accountId`	Account id
`schemaName`	Event schema name

Query Params

N/A

Headers

Name	Description
`X-Events-API-AccountName`	The global account name, as shown in the Controller UI **License** page.
`X-Events-API-Key`	The Analytics API key. See Manage API Keys.
`Accept`	The `Content-Type` of the response body. The supported value is `application/vnd.appd.events+json;v=2`.
`Content-type`	The `Content-Type` of the request body. The default is `application/vnd.appd.events+json;v=2` which also versions the resource representation (v=2).

Example SaaS Update

Delete Event Schema

Use this API to delete an existing event schema.

Format

Path Params

Name	Description
`accountId`	Account id
`eventType`	Event schema name

Query Params

N/A

Headers

Name	Description
`X-Events-API-AccountName`	The global account name, as shown in the Controller UI **License** page.
`X-Events-API-Key`	The Analytics API key. See Manage API Keys
`Accept`	The `Content-Type` of the response body. The following is the supported value: `application/vnd.appd.events+json;v=2`

Example SaaS Delete Request

```
DELETE http://analytics.api.example.com/events/schema/{schemaName} HTTP/1.1
X-Events-API-AccountName:<global_account_name>
X-Events-API-Key:<api_key>
Accept: application/vnd.appd.events+json;v=2
```

Querying Events

When querying analytics events data, the following applies:

- Every Events Service API has a limit of 200 searches per minute by each account on each event type.
- The Multi-Query Events API is limited to twenty queries per HTTP request.
- The Analytics Query API can return a maximum of 10,000 results. If you would like to retrieve and paginate over the data, you can use the scroll mode. Scroll mode is limited to 1,000 results per batch.

  ⚠ Scroll mode cannot be used in Query Events (Multiple Queries).

- Limits work differently for aggregation and non-aggregation queries. Because we use the limits specified in the ADQL query as the bucket count limit, it is not possible to also use it as the overall result count limit. Therefore, the URL parametric limit is used for the overall limit. In the case of non-aggregation queries, there is no bucket limit, so the limit specified in the ADQL query is taken as the row count limit and the URL parametric limit becomes the second place to look for it if the ADQL query does not specify a limit.
  - Aggregation queries: the total returned row count is limited by the limit in the URL query parameter, and is not directly related to the limits specified in the ADQL query statement itself. The limits in the ADQL query apply only to bucket counts in aggregations.
  - Non-aggregation queries: If `LIMIT` is not specified in the `SELECT` statement, the value specified in the URL query parameter is used. If the limit query parameter is also absent, the default is 100.

Query Events (Single Query)

This API can be used either as a simple text-formatted query or as a JSON-formatted query. The JSON-formatted query can accommodate multiple queries per call and is described in Query Events (Multiple Queries).

An event type might search against multiple event types. Therefore, the event type is not provided in the URL path or as a query parameter, but as part of the ADQL query provided in the request body itself. Your ADQL queries must adhere to the syntax described in the ADQL Reference.

This section describes the single query form for querying events.

Format

Query Params

Name	Description
`start`	Filter results based on the minimum event timestamp, specified in ISO 8601 time (https://en.wikipedia.org/wiki/ISO_8601) or Unix time (http://en.wikipedia.org/wiki/Unix_time). If not specified, then the default is no minimum timestamp filtering. Note that data returned will always be limited by the data retention. Specify the time in combined UTC date and time format or epoch milliseconds. Start time is inclusive of limiting timestamps.
`end`	Filter results based on the maximum event timestamp, specified in ISO 8601 time (https://en.wikipedia.org/wiki/ISO_8601) or Unix time (http://en.wikipedia.org/wiki/Unix_time). If not specified, then the default is no maximum timestamp filtering. ⓘ Data returned will always be limited by the data retention. Specify the time in combined UTC date and time format or epoch milliseconds. End time is inclusive of limiting timestamps.
`limit`	Limits the number of results returned. The default value is 100. The upper limit on the results that can be fetched is 1,000.
`mode`	The default mode is none. You can use the scroll mode to fetch bulk data and paginate over 10,000 ADQL results. Unless a limit is specified in the query, the API enables the Events Service to fetch beyond the maximum limit. The default mode supports aggregation but the scroll mode does not support aggregation or math operations. Order by queries is supported in scroll mode. Scroll queries returns results in batches. Every request response contains results along with a `scrollid`, which needs to be passed in the next request to fetch the next batch of results.

Headers

Name	Description
`X-Events-API-AccountName`	The global account name, as shown in the Controller UI **License** page.
`X-Events-API-Key`	The Analytics API key. See Manage API Keys.
`Accept`	The `Content-Type` of the response body. The supported value is `application/vnd.appd.events+json;v=2`.
`Content-type`	The `Content-Type` of the request body. The default is `application/vnd.appd.events+json;v=2` which also versions the resource representation (v=2).

Example SaaS Query

Query Events (Multiple Queries)

Use this API to execute multiple queries against a particular account and event type in parallel. A query event that specifies multiple queries does so by including multiple ADQL queries in the body of the request. Your ADQL queries must adhere to the syntax described in the ADQL Reference.

Using queries in this form takes advantage of certain backend optimizations in query performance. Query filter criteria, such as time range and limit, can be overridden by each inner query.

ⓘ The Multi-Query Events API is limited to twenty queries per HTTP request.

Format

Path Params

None

Query Params

Name	Description
`start`	Filter results based on the minimum event timestamp, specified in ISO 8601 time or Unix time. If not specified, then the default is no minimum timestamp filtering. Specify the time in combined UTC date and time format or epoch milliseconds. Start time is inclusive of limiting timestamps.
`end`	Filter results based on the maximum event timestamp, specified in ISO 8601 time or Unix time. If not specified, then the default is no maximum timestamp filtering. Specify the time in combined UTC date and time format or epoch milliseconds. End time is inclusive of limiting timestamps.
`limit`	Limits the number of results returned. The default value is 100. The upper limit on the results that can be fetched is 10,000.

Headers

Name	Description
`X-Events-API-AccountName`	The global account name, as shown in the Controller UI **License** page.
`X-Events-API-Key`	The Analytics API key. See Manage API Keys.
`Accept`	The `Content-Type` of the response body. The supported value is `application/vnd.appd.events+json;v=2`.
`Content-type`	The `Content-Type` of the request body. The default is `application/vnd.appd.events+json;v=2` which also versions the resource representation (v=2).

Payload

Field	Description
label	(Optional) Friendly name to identify the query.
query	ADQL query to execute.
start	(Optional) Overrides the start parameter value provided as a query parameter.
end	(Optional) Overrides the end parameter value provided as a query parameter.
limit	(Optional) Overrides the limit provided as a query parameter.

Example SaaS Multiple Query

RBAC API

This page describes the Role-Based Access Control (RBAC) API methods you can use to manage users, groups, and roles for AppDynamics features. These operations provide more flexibility and automation with RBAC management. Relationship settings such as `addUserToGroup` and `removeUserToGroup` are supported.

You must be the account owner or have **administer user** permissions to use the RBAC API.

SAML and LDAP user creations are not supported. You can only create permissions through the UI. See Manage Users and Groups.

Create User

Use this to create users in the current account. The request payload should specify `name`, `security_provider_type`, `displayName`, and `password`. The user ID is generated by the server.

Format

```
POST /controller/api/rbac/v1/users
```

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory
`name`	Request payload		Yes
`security_provider_type`	Request payload	`"INTERNAL"`	Yes
`displayName`	Request payload		Yes
`password`	Request payload		Yes

Example

```
curl -H "Content-Type: application/vnd.appd.cntrl+json;v=1" -X POST -d '{"name": "user10",
"security_provider_type": "INTERNAL", "displayName": "user10", "password": "welcome"}' -u user1@customer1
http://localhost:8080/controller/api/rbac/v1/users

Response status code 200 :
{
  "id": 10,
  "name": "user10",
  "displayName": "user10",
  "security_provider_type": "INTERNAL"
}
```

Get User by ID

Use this to get full user information, including a summary of affiliated groups and roles, using the `userId` in the current account.

Format

```
GET /controller/api/rbac/v1/users/userId
```

Example

```
curl -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/users/4

Response status code 200 :
{
   "id": 4,
   "name": "user1",
   "email": "user1@customer1.com",
   "displayName": "user1",
   "security_provider_type": "INTERNAL",
   "roles":
      [
        {"id": 17,"name": "Workflow Executor"},
        {"id": 18,"name": "DB Monitoring Administrator"},
        {"id": 19,"name": "DB Monitoring User"},
        {"id": 20,"name": "Analytics Administrator"},
        {"id": 21,"name": "Server Monitoring Administrator"},
        {"id": 22,"name": "Server Monitoring User"},
        {"id": 23,"name": "Universal Agent Administrator"},
        {"id": 24,"name": "Universal Agent User"},
        {"id": 13,"name": "Account Administrator"},
        {"id": 14,"name": "Administrator"},
        {"id": 15,"name": "User"},
        {"id": 16,"name": "Dashboard Viewer"}
      ],
   "groups":
      [
        {"id": 1,"name": "group_01"}
      ]
}
```

This API only supports retrieving internal users and not SAML or LDAP.

Get User by Name

Use this to get full user information, including a summary of affiliated groups and roles, using the `userName` in the current account.

Format

`GET /controller/api/rbac/v1/users/name/name`

Example

```
curl -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/users/name/user1

Response status code 200 :
{
   "id": 4,
   "name": "user1",
   "email": "user1@customer1.com",
   "displayName": "user1",
   "security_provider_type": "INTERNAL",
   "roles":
      [
        {"id": 17,"name": "Workflow Executor"},
        {"id": 18,"name": "DB Monitoring Administrator"},
        {"id": 19,"name": "DB Monitoring User"},
        {"id": 20,"name": "Analytics Administrator"},
        {"id": 21,"name": "Server Monitoring Administrator"},
        {"id": 22,"name": "Server Monitoring User"},
        {"id": 23,"name": "Universal Agent Administrator"},
        {"id": 24,"name": "Universal Agent User"},
        {"id": 13,"name": "Account Administrator"},
        {"id": 14,"name": "Administrator"},
        {"id": 15,"name": "User"},
        {"id": 16,"name": "Dashboard Viewer"}
      ],
   "groups":
      [
        {"id": 1,"name": "group_01"}
      ]
```

- This API only supports retrieving internal users and not SAML or LDAP.
- You have to include an optional parameter (securityProviderType**)** to find SAML/LDAP users.

Get All Users

Use this to get a list of all users in the current account. The list includes user summaries, which includes `userId` and `userName`.

Format

```
GET /controller/api/rbac/v1/users
```

Example

```
curl -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/users

Response status code 200 :
{
   "users":
      [
         {"id": 4,"name": "user1"},
         {"id": 10,"name": "user10"}
      ]
}
```

Update User

Use this to update a user by `userId` in the current account. Only the user object itself is updated, with the relationship to roles and groups remaining unaffected.

Format

```
PUT /controller/api/rbac/v1/users/userId
```

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory
id	Request payload		Yes
name	Request payload		Yes
displayName	Request payload		Yes
security_provider_type	Request payload	"INTERNAL"	Yes

Example

```
curl -H "Content-Type: application/vnd.appd.cntrl+json;v=1" -X PUT -d '{"id": 11,"name": "updated_user9","
displayName": "user9","security_provider_type": "INTERNAL"}' -u user1@customer1 http://localhost:8080/controller
/api/rbac/v1/users/11

Response status code 200 :
{
   "id": 11,
   "name": "updated_user9",
   "displayName": "user9",
   "security_provider_type": "INTERNAL"
}
```

Delete User

Use this to delete a user by `userId` in the current account.

Format

`DELETE /controller/api/rbac/v1/users/userId`

Example

```
curl -X DELETE -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/users/11

Response status code 200 :
```

Create Group

Use this to create a group in the current account. The `groupId` is generated by the server.

Format

`POST /controller/api/rbac/v1/groups`

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory
name	Request payload		Yes
description	Request payload		No
security_provider_type	Request payload	"INTERNAL"	Yes

Example

```
curl -H "Content-Type: application/vnd.appd.cntrl+json;v=1" -X POST -d '{"name": "group100","description": "new
description", "security_provider_type": "INTERNAL"}' -u user1@customer1 http://localhost:8080/controller/api
/rbac/v1/groups

Response status code 200 :
{
   "id": 2,
   "name": "group100",
   "security_provider_type": "INTERNAL",
   "description": "new description"
}
```

Get Group by ID

Use this to get full group information by `groupId` in the current account.

Format

`GET /controller/api/rbac/v1/groups/groupId`

Example

```
curl -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/groups/1

Response status code 200 :
{
   "id": 1,
   "name": "group_03",
   "security_provider_type": "INTERNAL"
   "description": "",
   "roles":
      [
         {"id": 19,"name": "DB Monitoring User"},
         {"id": 20,"name": "Analytics Administrator"},
         {"id": 21,"name": "Server Monitoring Administrator"},
         {"id": 22,"name": "Server Monitoring User"},
         {"id": 23,"name": "Universal Agent Administrator"},
         {"id": 13,"name": "Account Administrator"},
         {"id": 16,"name": "Dashboard Viewer"}
      ]
}
```

Get Group by Name

Use this to get full group information by `groupName` in the current account.

Format

`GET /controller/api/rbac/v1/groups/name/name`

Example

```
curl -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/groups/name/group_03

Response status code 200 :
{
   "id": 1,
   "name": "group_03",
   "security_provider_type": "INTERNAL"
   "description": "",
   "roles":
      [
         {"id": 19,"name": "DB Monitoring User"},
         {"id": 20,"name": "Analytics Administrator"},
         {"id": 21,"name": "Server Monitoring Administrator"},
         {"id": 22,"name": "Server Monitoring User"},
         {"id": 23,"name": "Universal Agent Administrator"},
         {"id": 13,"name": "Account Administrator"},
         {"id": 16,"name": "Dashboard Viewer"}
      ]
}
```

Get All Groups

Use this to get all groups in the current account. This only returns group summaries, which includes `groupId` and `groupName`.

Format

`GET /controller/api/rbac/v1/groups`

Example

```
curl -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/groups

Response status code 200 :
{
   "groups":
      [
         {"id": 1,"name": "group_03"},
         {"id": 2,"name": "group100"}
      ]
}
```

Update Group

Use this to update a group by `groupId` in the current account. Only the group itself is updated, while the relationships with users and roles remain unaffected.

Format

`PUT /controller/api/rbac/v1/groups/groupId`

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory
`id`	Request payload		Yes
`name`	Request payload		Yes
`description`	Request payload		No
`security_provider_type`	Request payload	`"INTERNAL"`	Yes

Example

```
curl -H "Content-Type: application/vnd.appd.cntrl+json;v=1" -X PUT -d '{"id": 1, "name": "group2","
description": "new description", "security_provider_type": "INTERNAL"}' -u user1@customer1 http://localhost:8080
/controller/api/rbac/v1/groups/1

Response status code 200 :
{
   "id": 1,
   "name": "group2",
   "security_provider_type": "INTERNAL",
   "description": "new description",
   "roles":
      [
         {"id": 19,"name": "DB Monitoring User"},
         {"id": 20,"name": "Analytics Administrator"},
         {"id": 21,"name": "Server Monitoring Administrator"},
         {"id": 22,"name": "Server Monitoring User"},
         {"id": 23,"name": "Universal Agent Administrator"},
         {"id": 13,"name": "Account Administrator"},
         {"id": 16,"name": "Dashboard Viewer"}
      ]
}
```

Delete Group

Use this to delete a group by `groupId` in the current account.

Format

```
DELETE /controller/api/rbac/v1/groups/groupId
```

Example

```
curl -X DELETE -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/groups/1


Response status code 200 :
```

Add User to Group

Use this to add a user to a group by `userId` and `groupId`.

Format

```
PUT /controller/api/rbac/v1/groups/groupId/users/userId
```

Example

```
curl -H "Content-Type: application/vnd.appd.cntrl+json;v=1" -X PUT -u user1@customer1 http://localhost:8080
/controller/api/rbac/v1/groups/2/users/10


Response status code 200 :
```

Remove User from Group

Use this to remove a user from a group by `userId` and `groupId`.

Format

```
DELETE /controller/api/rbac/v1/groups/groupId/users/userId
```

Example

```
curl -X DELETE -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/groups/2/users/10


Response status code 200 :
```

Create Role

Use this to create a role in the current account. The ID is generated by the server.

Format

`POST /controller/api/rbac/v1/roles`

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory
`name`	Request payload		Yes
`description`	Request payload		No
permissions	Request payload		No

Example

```
curl -X POST /controller/api/rbac/v1/roles \
-H 'Content-Type: application/vnd.appd.cntrl+json;v=1' \
-d '{
 "name": "SampleRole2",
  "permissions": [
   {
     "entityType": "APPLICATION",
     "entityId": 24,
     "action": "CONFIG_ACTIONS"
   },
   {
     "entityType": "APPLICATION",
     "entityId": 24,
     "action": "CONFIG_BASELINES"
   },
   {
     "entityType": "APPLICATION",
     "entityId": 24,
     "action": "CONFIG_BUSINESS_TRANSACTIONS"
   },
   {
     "entityType": "APPLICATION",
     "entityId": 24,
     "action": "CONFIG_ERROR_DETECTION"
   },
   {
     "entityType": "APPLICATION",
     "entityId": 24,
     "action": "CONFIG_EUM"
   },
   {
     "entityType": "APPLICATION",
     "entityId": 24,
     "action": "CONFIG_EVENT_REACTOR"
   },
   {
     "entityType": "APPLICATION",
     "entityId": 24,
     "action": "CONFIG_POLICIES"
   },
   {
     "entityType": "APPLICATION",
     "entityId": 24,
     "action": "CONFIG_TRANSACTION_DETECTION"
   },
   {
     "entityType": "APPLICATION",
     "entityId": 24,
     "action": "VIEW"
   }
 ]
}’
```

```
200 OK
{
  "id": 87,
  "name": "SampleRole2"
}
```

Add Role to User

Use this to add a role to a user by `roleId` and `userId`.

Format

`PUT /controller/api/rbac/v1/roles/roleId/users/userId`

Example

```
curl -H "Content-Type: application/vnd.appd.cntrl+json;v=1" -X PUT -u user1@customer1 http://localhost:8080
/controller/api/rbac/v1/roles/50/users/10


Response status code 200 :
```

Remove Role from User

Use this to remove a role from a user by `roleId` and `userId`.

Format

```
DELETE /controller/api/rbac/v1/roles/roleId/users/userId
```

Example

```
curl -X DELETE -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/roles/50/users/10

Response status code 200 :
```

Add Role to Group

Use this to add a role to a group by `roleId` and `groupId`.

Format

```
PUT /controller/api/rbac/v1/roles/roleId/groups/groupId
```

Example

```
curl -H "Content-Type: application/vnd.appd.cntrl+json;v=1" -X PUT -u user1@customer1 http://localhost:8080
/controller/api/rbac/v1/roles/50/groups/2


Response status code 200 :
```

Remove Role from Group

Use this to remove a role from a group by `roleId` and `groupId`.

Format

```
DELETE /controller/api/rbac/v1/roles/roleId/groups/groupId
```

Example

```
curl -X DELETE -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/roles/50/groups/2

Response status code 200 :
```

Get Role by ID

Use this to get full role information by `roleId` in the current account. This only returns the `role` object.

Format

```
GET /controller/api/rbac/v1/roles/[roleId]?include-permissions=true
```

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory
`id`	Request payload		Yes
`include-permissions (>= v4.5.14)`	Request payload	`"true"`	No

Example

```
curl -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/roles/15?include-permissions=true


Response status code 200 :

{
 "id": 15,
 "name": "SampleRole",
 "permissions": [
   {
     "id": 2619,
     "entityType": "APPLICATION",
     "entityId": 27,
     "action": "CONFIG_ACTIONS"
   },
   {
     "id": 2621,
     "entityType": "APPLICATION",
     "entityId": 27,
     "action": "CONFIG_BASELINES"
   },
   {
     "id": 2620,
     "entityType": "APPLICATION",
     "entityId": 27,
     "action": "CONFIG_BUSINESS_TRANSACTIONS"
   },
   {
     "id": 2610,
     "entityType": "APPLICATION",
     "entityId": 27,
     "action": "CONFIG_ERROR_DETECTION"
   },
   {
     "id": 2615,
     "entityType": "APPLICATION",
     "entityId": 27,
     "action": "CONFIG_EUM"
   },
   {
     "id": 2618,
     "entityType": "APPLICATION",
     "entityId": 27,
     "action": "CONFIG_EVENT_REACTOR"
   },
   {
     "id": 2617,
     "entityType": "APPLICATION",
     "entityId": 27,
     "action": "CONFIG_POLICIES"
   },
   {
     "id": 2608,
     "entityType": "APPLICATION",
     "entityId": 27,
     "action": "CONFIG_TRANSACTION_DETECTION"
   },
   {
     "id": 2606,
     "entityType": "APPLICATION",
     "entityId": 27,
     "action": "VIEW"
   }
 ]
}
```

Get Role by Name

Use this to get full role information by `roleName` in the current account.

Format

```
GET /controller/api/rbac/v1/roles/name/[RoleName]?include-permissions=true
```

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory
name	Request payload		Yes
`include-permissions (>= v4.5.14)`	Request payload	`"true"`	No

Example

```
curl -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/roles/name/SampleRole?include-
permissions=true


Response status code 200 :

{
 "id": 15,
 "name": "SampleRole",
 "permissions": [
   {
     "id": 2619,
     "entityType": "APPLICATION",
     "entityId": 27,
     "action": "CONFIG_ACTIONS"
   },
   {
     "id": 2621,
     "entityType": "APPLICATION",
     "entityId": 27,
     "action": "CONFIG_BASELINES"
   },
   {
     "id": 2620,
     "entityType": "APPLICATION",
     "entityId": 27,
     "action": "CONFIG_BUSINESS_TRANSACTIONS"
   },
   {
     "id": 2610,
     "entityType": "APPLICATION",
     "entityId": 27,
     "action": "CONFIG_ERROR_DETECTION"
   },
   {
     "id": 2615,
     "entityType": "APPLICATION",
     "entityId": 27,
     "action": "CONFIG_EUM"
   },
   {
     "id": 2618,
     "entityType": "APPLICATION",
     "entityId": 27,
     "action": "CONFIG_EVENT_REACTOR"
   },
   {
     "id": 2617,
     "entityType": "APPLICATION",
     "entityId": 27,
     "action": "CONFIG_POLICIES"
   },
   {
     "id": 2608,
     "entityType": "APPLICATION",
     "entityId": 27,
     "action": "CONFIG_TRANSACTION_DETECTION"
   },
   {
     "id": 2606,
     "entityType": "APPLICATION",
     "entityId": 27,
     "action": "VIEW"
   }
 ]
}
```

Get All Roles

Use this to get all roles in the current account. This only returns role summaries, which includes `roleId` and `roleName`.

Format

```
GET /controller/api/rbac/v1/roles
```

Example

```
curl -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/roles

Response status code 200 :
{
   "roles":
      [
         {"id": 13,"name": "Account Administrator"},
         {"id": 14,"name": "Administrator"},
         {"id": 20,"name": "Analytics Administrator"},
         {"id": 16,"name": "Dashboard Viewer"},
         {"id": 18,"name": "DB Monitoring Administrator"},
         {"id": 19,"name": "DB Monitoring User"},
         {"id": 21,"name": "Server Monitoring Administrator"},
         {"id": 22,"name": "Server Monitoring User"},
         {"id": 23,"name": "Universal Agent Administrator"},
         {"id": 24,"name": "Universal Agent User"},
         {"id": 15,"name": "User"},
         {"id": 17,"name": "Workflow Executor"}
      ]
}
```

Update Role

Use this to update a role by `roleId` in the current account. This only updates the `role` object itself, while leaving the relationship with users and groups unaffected.

You cannot update permissions within a role through this API. You can only update the `name` and `description` parameters.

Format

```
PUT /controller/api/rbac/v1/roles/roleId
```

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory
`id`	Request payload		Yes
`name`	Request payload		Yes
`description`	Request payload		No

Example

```
curl -H "Content-Type: application/vnd.appd.cntrl+json;v=1" -X PUT -d '{"id": 49, "name": "role1","
description": "new description" }' -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/roles/49

Response status code 200 :
{
   "id": 49,
   "name": "role1",
   "description": "new description"
}
```

Delete Role

Use this to delete a role in the current account.

Format

```
DELETE /controller/api/rbac/v1/roles/roleId
```

Example

```
curl -X DELETE -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/roles/49


Response status code 200 :
```

Create Central Identity User API

This page describes the Create Central Identity User API methods you can use to create central identity(ci) users in the current account. This API is only available when the current account is configured with the `CI_USER_CREATION` property.

You must be the account owner or have the **administer user** permissions to use the Create Central Identity User API.

SAML and LDAP user creations are not supported. You can only create permissions through the UI. See Create and Manage Custom Roles.

Create User

The request payload should specify `email`, `security_provider_type`, and `displayName`. The server generates the user ID.

Format

```
POST /controller/api/rbac/v1/ci-user
```

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory
`email`	Request payload		Yes
`security_provider_type`	Request payload	`"INTERNAL"`	Yes
`displayName`	Request payload		Yes

Example

```
curl -H "Content-Type: application/vnd.appd.cntrl+json;v=1" -X POST -d '{"email": "user10@domain.com",
"security_provider_type": "INTERNAL", "displayName": "user10"}' -u user1@customer1 http://localhost:8080
/controller/api/rbac/v1/ci-user

Response status code 200 :
{
  "id": 10,
  "name": "user10@domain.com",
  "email": "user10@domain.com",
  "displayName": "user10",
  "security_provider_type": "INTERNAL"
}
```

Get User by ID

Use this to get full user information, including a summary of affiliated groups and roles, using the `userId` in the current account.

Format

```
GET /controller/api/rbac/v1/users/userId
```

Example

```
curl -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/users/4

Response status code 200 :
{
   "id": 4,
   "name": "user1",
   "email": "user1@customer1.com",
   "displayName": "user1",
   "security_provider_type": "INTERNAL",
   "roles":
      [
        {"id": 17,"name": "Workflow Executor"},
        {"id": 18,"name": "DB Monitoring Administrator"},
        {"id": 19,"name": "DB Monitoring User"},
        {"id": 20,"name": "Analytics Administrator"},
        {"id": 21,"name": "Server Monitoring Administrator"},
        {"id": 22,"name": "Server Monitoring User"},
        {"id": 23,"name": "Universal Agent Administrator"},
        {"id": 24,"name": "Universal Agent User"},
        {"id": 13,"name": "Account Administrator"},
        {"id": 14,"name": "Administrator"},
        {"id": 15,"name": "User"},
        {"id": 16,"name": "Dashboard Viewer"}
      ],
   "groups":
      [
        {"id": 1,"name": "group_01"}
      ]
}
```

Get User by Name

Use this to get full user information, including a summary of affiliated groups and roles, using the `userName` in the current account.

Format

`GET /controller/api/rbac/v1/users/name/name`

Example

```
curl -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/users/name/user1

Response status code 200 :
{
   "id": 4,
   "name": "user1",
   "email": "user1@customer1.com",
   "displayName": "user1",
   "security_provider_type": "INTERNAL",
   "roles":
      [
        {"id": 17,"name": "Workflow Executor"},
        {"id": 18,"name": "DB Monitoring Administrator"},
        {"id": 19,"name": "DB Monitoring User"},
        {"id": 20,"name": "Analytics Administrator"},
        {"id": 21,"name": "Server Monitoring Administrator"},
        {"id": 22,"name": "Server Monitoring User"},
        {"id": 23,"name": "Universal Agent Administrator"},
        {"id": 24,"name": "Universal Agent User"},
        {"id": 13,"name": "Account Administrator"},
        {"id": 14,"name": "Administrator"},
        {"id": 15,"name": "User"},
        {"id": 16,"name": "Dashboard Viewer"}
      ],
   "groups":
      [
        {"id": 1,"name": "group_01"}
      ]
```

Get All Users

Use this to get a list of all users in the current account. The list includes user summaries, which includes `userId` and `userName`.

Format

`GET /controller/api/rbac/v1/users`

Example

```
curl -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/users

Response status code 200 :
{
   "users":
      [
         {"id": 4,"name": "user1"},
         {"id": 10,"name": "user10"}
      ]
}
```

Update User

Use this to update a user by `userId` in the current account. Only the user object itself is updated, with the relationship to roles and groups remaining unaffected.

Format

`PUT /controller/api/rbac/v1/users/userId`

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory
`id`	Request payload		Yes

name	Request payload		Yes
displayName	Request payload		Yes
security_provider_type	Request payload	"INTERNAL"	Yes

Example

```
curl -H "Content-Type: application/vnd.appd.cntrl+json;v=1" -X PUT -d '{"id": 11,"name": "updated_user9","
displayName": "user9","security_provider_type": "INTERNAL"}' -u user1@customer1 http://localhost:8080/controller
/api/rbac/v1/users/11

Response status code 200 :
{
   "id": 11,
   "name": "updated_user9",
   "displayName": "user9",
   "security_provider_type": "INTERNAL"
}
```

Delete User

Use this to delete a user by `userId` in the current account.

Format

`DELETE /controller/api/rbac/v1/users/userId`

Example

```
curl -X DELETE -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/users/11

Response status code 200 :
```

Create Group

Use this to create a group in the current account. The `groupId` is generated by the server.

Format

`POST /controller/api/rbac/v1/groups`

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory
name	Request payload		Yes
description	Request payload		No
security_provider_type	Request payload	"INTERNAL"	Yes

Example

```
curl -H "Content-Type: application/vnd.appd.cntrl+json;v=1" -X POST -d '{"name": "group100","description": "new
description", "security_provider_type": "INTERNAL"}' -u user1@customer1 http://localhost:8080/controller/api
/rbac/v1/groups

Response status code 200 :
{
   "id": 2,
   "name": "group100",
   "security_provider_type": "INTERNAL",
   "description": "new description"
}
```

Get Group by ID

Use this to get full group information by `groupId` in the current account.

Format

`GET /controller/api/rbac/v1/groups/groupId`

Example

```
curl -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/groups/1

Response status code 200 :
{
   "id": 1,
   "name": "group_03",
   "security_provider_type": "INTERNAL"
   "description": "",
   "roles":
      [
         {"id": 19,"name": "DB Monitoring User"},
         {"id": 20,"name": "Analytics Administrator"},
         {"id": 21,"name": "Server Monitoring Administrator"},
         {"id": 22,"name": "Server Monitoring User"},
         {"id": 23,"name": "Universal Agent Administrator"},
         {"id": 13,"name": "Account Administrator"},
         {"id": 16,"name": "Dashboard Viewer"}
      ]
}
```

Get Group by Name

Use this to get full group information by `groupName` in the current account.

Format

`GET /controller/api/rbac/v1/groups/name/name`

Example

```
curl -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/groups/name/group_03

Response status code 200 :
{
   "id": 1,
   "name": "group_03",
   "security_provider_type": "INTERNAL"
   "description": "",
   "roles":
      [
         {"id": 19,"name": "DB Monitoring User"},
         {"id": 20,"name": "Analytics Administrator"},
         {"id": 21,"name": "Server Monitoring Administrator"},
         {"id": 22,"name": "Server Monitoring User"},
         {"id": 23,"name": "Universal Agent Administrator"},
         {"id": 13,"name": "Account Administrator"},
         {"id": 16,"name": "Dashboard Viewer"}
      ]
}
```

Get All Groups

Use this to get all groups in the current account. This only returns group summaries, which includes `groupId` and `groupName`.

Format

```
GET /controller/api/rbac/v1/groups
```

Example

```
curl -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/groups

Response status code 200 :
{
   "groups":
      [
         {"id": 1,"name": "group_03"},
         {"id": 2,"name": "group100"}
      ]
}
```

Update Group

Use this to update a group by `groupId` in the current account. Only the group itself is updated, while the relationships with users and roles remain unaffected.

Format

```
PUT /controller/api/rbac/v1/groups/groupId
```

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory
`id`	Request payload		Yes
`name`	Request payload		Yes
`description`	Request payload		No
`security_provider_type`	Request payload	`"INTERNAL"`	Yes

Example

```
curl -H "Content-Type: application/vnd.appd.cntrl+json;v=1" -X PUT -d '{"id": 1, "name": "group2","
description": "new description", "security_provider_type": "INTERNAL"}' -u user1@customer1 http://localhost:8080
/controller/api/rbac/v1/groups/1

Response status code 200 :
{
   "id": 1,
   "name": "group2",
   "security_provider_type": "INTERNAL",
   "description": "new description",
   "roles":
      [
         {"id": 19,"name": "DB Monitoring User"},
         {"id": 20,"name": "Analytics Administrator"},
         {"id": 21,"name": "Server Monitoring Administrator"},
         {"id": 22,"name": "Server Monitoring User"},
         {"id": 23,"name": "Universal Agent Administrator"},
         {"id": 13,"name": "Account Administrator"},
         {"id": 16,"name": "Dashboard Viewer"}
      ]
}
```

Delete Group

Use this to delete a group by `groupId` in the current account.

Format

```
DELETE /controller/api/rbac/v1/groups/groupId
```

Example

```
curl -X DELETE -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/groups/1


Response status code 200 :
```

Add User to Group

Use this to add a user to a group by `userId` and `groupId`.

Format

```
PUT /controller/api/rbac/v1/groups/groupId/users/userId
```

Example

```
curl -H "Content-Type: application/vnd.appd.cntrl+json;v=1" -X PUT -u user1@customer1 http://localhost:8080
/controller/api/rbac/v1/groups/2/users/10


Response status code 200 :
```

Remove User from Group

Use this to remove a user from a group by `userId` and `groupId`.

Format

```
DELETE /controller/api/rbac/v1/groups/groupId/users/userId
```

Example

```
curl -X DELETE -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/groups/2/users/10


Response status code 200 :
```

Create Role

Use this to create a role in the current account. The ID is generated by the server.

Format

```
POST /controller/api/rbac/v1/roles
```

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory
`name`	Request payload		Yes
`description`	Request payload		No

Example

```
curl -H "Content-Type: application/vnd.appd.cntrl+json;v=1" -X POST -d '{"name": "role2","description": "new
description"}' -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/roles

Response status code 200 :
{
   "id": 49,
   "name": "role2",
   "description": "new description"
}
```

Add Role to User

Use this to add a role to a user by `roleId` and `userId`.

Format

```
PUT /controller/api/rbac/v1/roles/roleId/users/userId
```

Example

```
curl -H "Content-Type: application/vnd.appd.cntrl+json;v=1" -X PUT -u user1@customer1 http://localhost:8080
/controller/api/rbac/v1/roles/50/users/10


Response status code 200 :
```

Remove Role from User

Use this to remove a role from a user by `roleId` and `userId`.

Format

```
DELETE /controller/api/rbac/v1/roles/roleId/users/userId
```

Example

```
curl -X DELETE -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/roles/50/users/10

Response status code 200 :
```

Add Role to Group

Use this to add a role to a group by `roleId` and `groupId`.

Format

```
PUT /controller/api/rbac/v1/roles/roleId/groups/groupId
```

Example

```
curl -H "Content-Type: application/vnd.appd.cntrl+json;v=1" -X PUT -u user1@customer1 http://localhost:8080
/controller/api/rbac/v1/roles/50/groups/2


Response status code 200 :
```

Remove Role from Group

Use this to remove a role from a group by `roleId` and `groupId`.

Format

`DELETE /controller/api/rbac/v1/roles/roleId/groups/groupId`

Example

```
curl -X DELETE -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/roles/50/groups/2

Response status code 200 :
```

Get Role by ID

Use this to get full role information by `roleId` in the current account. This only returns the role object.

Format

`GET /controller/api/rbac/v1/roles/roleId`

Example

```
curl -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/roles/15

Response status code 200 :
{
   "id": 15,
   "name": "User",
   "description": "Can view applications and dashboards but not modify their configuration"
}
```

Get Role by Name

Use this to get full role information by `roleName` in the current account.

Format

`GET /controller/api/rbac/v1/roles/name/name`

Example

```
curl -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/roles/name/User

Response status code 200 :
{
   "id": 15,"name":
   "User",
   "description": "Can view applications and dashboards but not modify their configuration"
}
```

Get All Roles

Use this to get all roles in the current account. This only returns role summaries, which includes `roleId` and `roleName`.

Format

`GET /controller/api/rbac/v1/roles`

Example

```
curl -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/roles

Response status code 200 :
{
   "roles":
      [
         {"id": 13,"name": "Account Administrator"},
         {"id": 14,"name": "Administrator"},
         {"id": 20,"name": "Analytics Administrator"},
         {"id": 16,"name": "Dashboard Viewer"},
         {"id": 18,"name": "DB Monitoring Administrator"},
         {"id": 19,"name": "DB Monitoring User"},
         {"id": 21,"name": "Server Monitoring Administrator"},
         {"id": 22,"name": "Server Monitoring User"},
         {"id": 23,"name": "Universal Agent Administrator"},
         {"id": 24,"name": "Universal Agent User"},
         {"id": 15,"name": "User"},
         {"id": 17,"name": "Workflow Executor"}
      ]
}
```

Update Role

Use this to update a role by `roleId` in the current account. This only updates the role object itself, while leaving the relationship with users and groups unaffected.

Format

`PUT /controller/api/rbac/v1/roles/roleId`

Input Parameters

Parameter Name	Parameter Type	Value	Mandatory
`id`	Request payload		Yes
`name`	Request payload		Yes
`description`	Request payload		No

Example

```
curl -H "Content-Type: application/vnd.appd.cntrl+json;v=1" -X PUT -d '{"id": 49, "name": "role1","
description": "new description" }' -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/roles/49

Response status code 200 :
{
   "id": 49,
   "name": "role1",
   "description": "new description"
}
```

Delete Role

Use this to delete a role in the current account.

Format

`DELETE /controller/api/rbac/v1/roles/roleId`

Example

```
curl -X DELETE -u user1@customer1 http://localhost:8080/controller/api/rbac/v1/roles/49


Response status code 200 :
```

License API

Related pages:

- License Management
- Licensing
- License Entitlements and Restrictions

This page describes the License APIs you can use to perform functions related to licensing and accounts including uploading license files, allocating licenses, getting license and account usage information as well as creating, updating, and deleting license rules.

The License APIs below are separated by licensing model. To learn about licenses within each model, see License Entitlements and Restrictions.

Integration Modules

Related pages:

- AppDynamics Exchange

The AppDynamics Community Exchange includes many extensions that supplement the capabilities of the AppDynamics Application Performance Monitoring (APM) Platform. Documentation on setting up and using the integration module appears with the module.

Some integrations listed here do not display on the Controller UI Integrations page. This is because not all integrations require configuration at the Controller (e.g., Rookout, Apica) level or they can only be configured at the underlying Glassfish Admin Console level (e.g., DB CAM).

The following integration modules are built into the AppDynamics platform and described on the pages linked here.

Integration Module	More information
Rookout	Deep Code Insights powered by Rookout
Apica	Apica & AppDynamics
DB CAM	Integrate AppDynamics with DB CAM
Scalyr	Integrate AppDynamics with Scalyr
Splunk	Integrate AppDynamics with Splunk
Compuware Strobe	Integrate AppDynamics with Compuware Strobe
AppDynamics for Databases	Integrate AppDynamics for Databases with AppDynamics Pro 3.7 and higher
Cisco® Application Centric Infrastructure (Cisco ACI™)	Integrate AppDynamics with Cisco ACI™
ServiceNow®	Integrate AppDynamics with ServiceNow®CMDB and Event Management

Integrate AppDynamics with Cisco Application Centric Infrastructure

> ⚠ AppDynamics is ending support for Application Centric Infrastructure (ACI). The feature will not work once customers upgrade their Controllers beyond > 21.1. See Support Advisory: Application Centric Infrastructure (ACI) End of Life (EOL) Notice.

This page describes the advantages of the integration of AppDynamics with Cisco® Application Centric Infrastructure (Cisco ACI™). This solution provides a unified view from the application code to underlying network layers for business-critical applications running in a data center. This enables the network operations Admin and the application operations Admin to quickly troubleshoot issues at the application and the network levels.

> ⚠ You can integrate the Cisco ACI™ solution with select AppDynamics Controllers only. If you are interested in using the integrated solution, contact help@appdynamics.com. See Troubleshoot Using the Integrated Solution.

Getting Started with AppDynamics- Cisco ACI Integration

> AppDynamics is ending support for Application Centric Infrastructure (ACI). The feature will not work once customers upgrade their Controllers beyond > 21.1. See Support Advisory: Application Centric Infrastructure (ACI) End of Life (EOL) Notice.

This page provides instructions for integrating AppDynamics with Cisco ACI™ solution and verifying interoperability between the applications. This integration provides end-to-end visibility into the application and network layers within a data center.

Before You Begin

Ensure that your environment meets these minimum system requirements:

- Controller >= 4.5.2 with a Network Visibility license per OS instance. See Controller System Requirements.
- Hosts with Network Visibility Agents >= 4.5.2. See Network Visibility Requirements and Supported Environments.
- Hosts with Java Agents >= 4.5.2 or .NET Agent for Windows >= 4.5.15 with an App Agent license. See Install Java Agent or .NET Agent.
- `sudo` or `root` access permissions on the agent host to install the Network Agent.
- Network Visibility is supported on Linux and Windows operating systems. See Network Visibility Supported Environments.

> If your applications are running in a Kubernetes environment, enable the Kubernetes-ACI integration before you proceed with the AppDynamics and Cisco ACI™ integration. See Cisco ACI and Kubernetes Integration.

Enable SSL for Cisco ACI Integration

The Cisco Application Policy Infrastructure Controller (APIC) supports the import and storage of an SSL certificate and private key into the Controller to create a secure and trusted environment with AppDynamics.

1. Access the AppDynamics Controller using the SSH.
2. At a command prompt, change directories to the following location:

```
<controller_home>/appserver/glassfish/domains/domain1/config
```

3. Import the self-signed certificate from the appropriate directory as follows:

```
keytool -import -keystore cacerts.jks -file /path/to/RootCA.crt
```

Enable Cisco ACI Integration (Controller <= 4.5.10)

> The Cisco ACI Integration is enabled by default for >= 4.5.11.

1. Log in to the Administration Console:

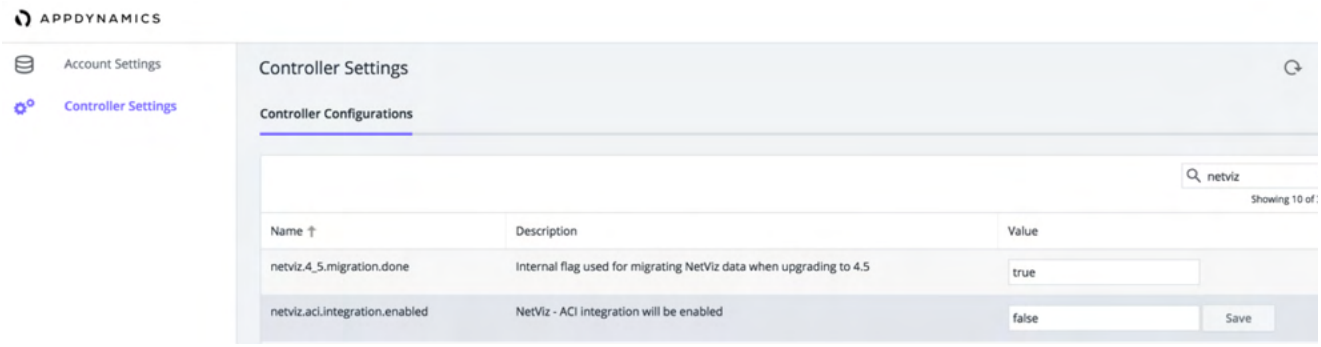
```
http:<controller-hostname>:<controller-port>/controller/admin.jsp
```

   For example:

```
https:<controller-hostname>:8800/controller/admin.jsp
https:<controller-hostname>:443/controller/admin.jsp
```

> You must use the password that was set for the root user of the Controller when the Controller was installed. See User Manageme

2. In **Controller Settings**, search for **netviz.aci.integration.enabled** and set this field to **true**.
3. Click **Save**.



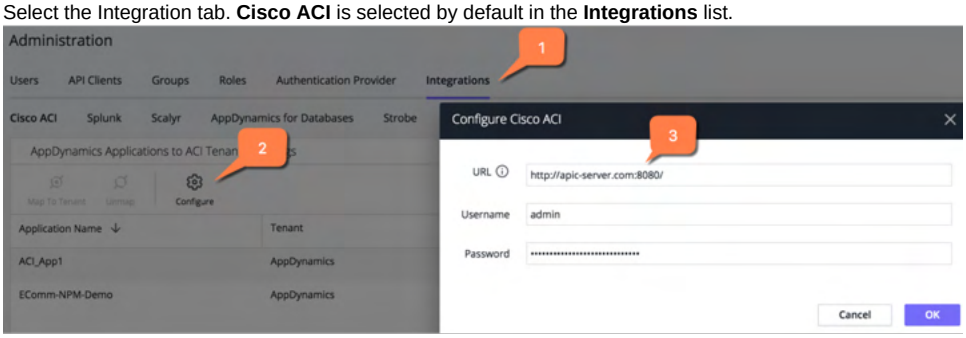
Configure Cisco ACI Credentials

If you are using an AppDynamics Controller >= 4.5.11, the Cisco ACI Integration feature is enabled by default. If you are using an AppDynamics Controller <= 4.5.10, you must enable Cisco ACI Integration manually before you configure your credentials.

1. Log in to the AppDynamics Controller as an administrator:

   ```
   http:<controller-hostname>:<controller-port>/controller
   ```

2. Click **Settings > Administration**.
3. Select the Integration tab. **Cisco ACI** is selected by default in the **Integrations** list.



4. Click **Configure** and enter these details:
   a. Full URL along with the port details to the Cisco APIC Server. For example,

      ```
      http://apic-server.com:8080/
      ```

   b. User credentials to the Cisco APIC.
5. Click **OK**.

AppDynamics initiates the background processes required to integrate with the Cisco ACI solution.

Correlate AppDynamics and Cisco ACI Components

The AppDynamics integration with Cisco ACI brings together the logical constructs of both solutions. This enables the network operations admin and the application operations admin to obtain a comprehensive end-to-end view from the application to the network. Correlating or mapping the components of both solutions helps the admins triage and troubleshoot issues quickly.

AppDynamics Components

The AppDynamics application model serves as the framework around which AppDynamics organizes and presents application performance information. A typical application environment consists of different components that interact in multiple ways.

- Application—The top-level container in the AppDynamics model. A business application contains a set of related services and business transactions.
- Tier—A grouping of one or more nodes in the AppDynamics application model. There is no interaction among nodes within a single tier.
- Node—The smallest unit of the modeled environment. A node corresponds to a monitored server or JVM in the application environment. A node may correspond to an individual application server, JVM, CLR, PHP application, and Apache Web server.

AppDynamics components are logically arranged as Application > Tiers > Nodes.

Cisco ACI Components

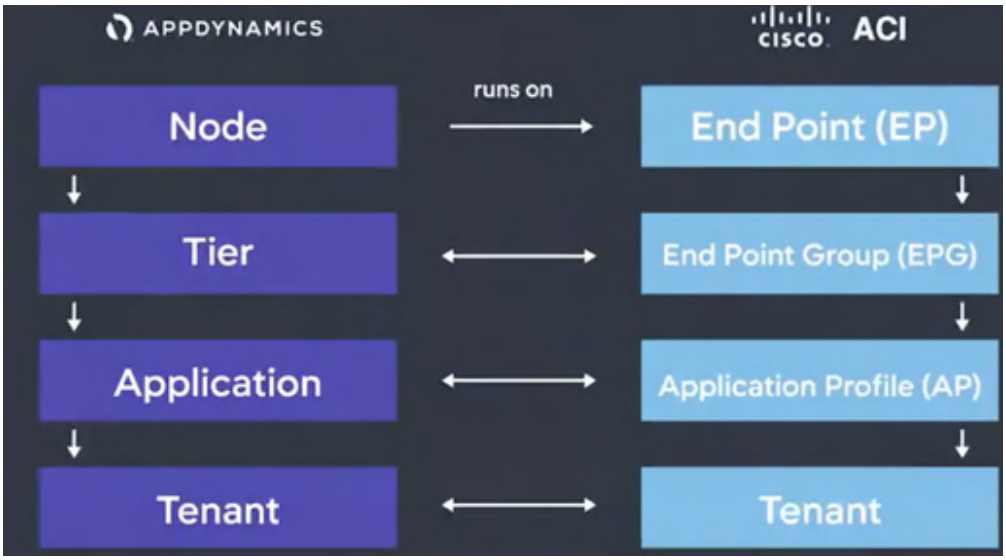
Cisco ACI is a Software-Defined Networking (SDN) solution that simplifies, optimizes, and accelerates infrastructure deployment and governance, and expedites the application deployment lifecycle.

Cisco ACI implements Cisco's intent-based networking framework. It captures higher-level business and user intent in the form of a policy and converts this intent into the network constructs necessary to dynamically provision the network, security, and infrastructure services.

- Cisco APIC—The infrastructure Controller is the main architectural component of the Cisco ACI solution. It is the unified point of automation and management for the Cisco ACI fabric, policy enforcement, and health monitoring.
    - The APIC appliance is a centralized, clustered controller that optimizes performance and unifies the operation of physical and virtual environments. The controller manages and operates a scalable multi-tenant Cisco ACI fabric.
- Tenant—A logical container for policies that enable an administrator to exercise domain-based access control.
- Application Profile—Defines the policies, services, and relationships between endpoint groups (EPGs).
- EPG—A managed object that is a named logical entity. It is a collection of endpoints (EPs).
- Endpoints—Devices that are connected to the network directly or indirectly. They have an address (identity), a location, and attributes (such as version).

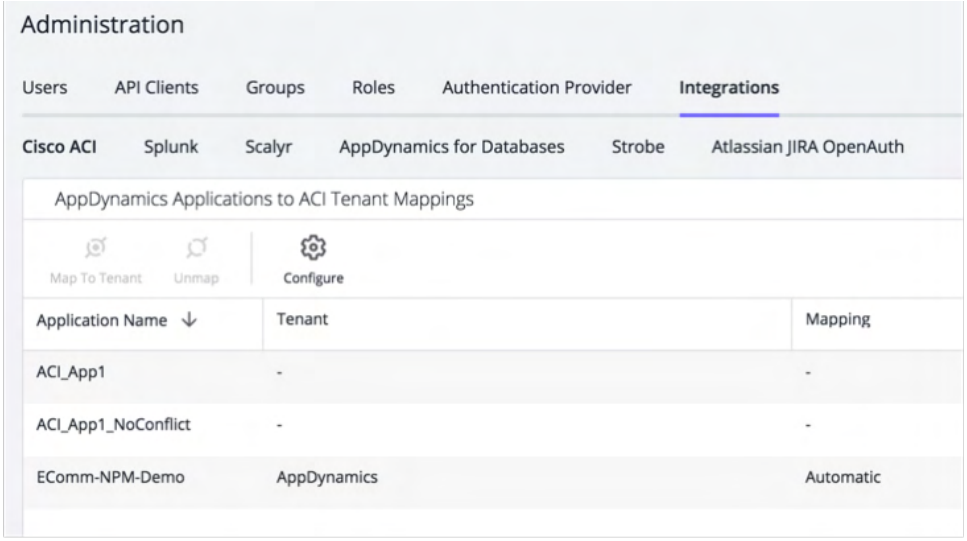
Cisco APIC components are logically arranged as Application profile > EPGs > EPs.

This figure shows the correlation of AppDynamics and Cisco ACI components.



AppDynamics Applications to ACI Tenant Mapping

After you configure Cisco ACI credentials, all AppDynamics applications display in the **AppDynamics Applications to ACI Tenant Mappings** page.



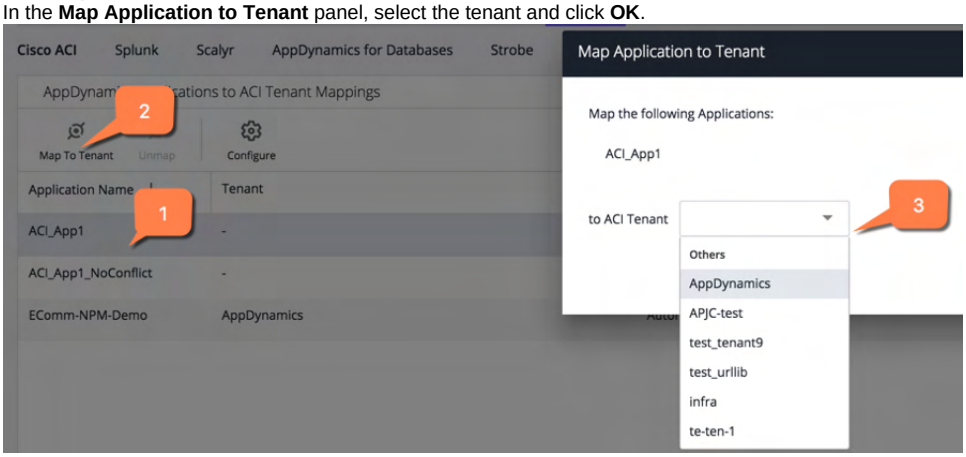
These mappings are based on the mappings between AppDynamics nodes and Cisco ACI endpoints.

When the endpoints in different tenants have the same IP address, a node-to-endpoint mapping conflict occurs. AppDynamics resolves these conflicts based on heuristic data. However, if the conflicts persist, you must resolve these conflicts manually.

Map AppDynamics Application to Cisco ACI Tenant Manually

The application to Cisco ACI tenant map is left blank in **AppDynamics Applications to ACI Tenant Mappings** page when there is a conflict in auto-mapping. You can map applications to Cisco ACI tenants manually.

1. Select the AppDynamics application you want to map manually. Multiple applications can be mapped to a tenant at a given time.
2. Click **Map to Tenant**.
3. In the **Map Application to Tenant** panel, select the tenant and click **OK**.



Override Auto-Mapping

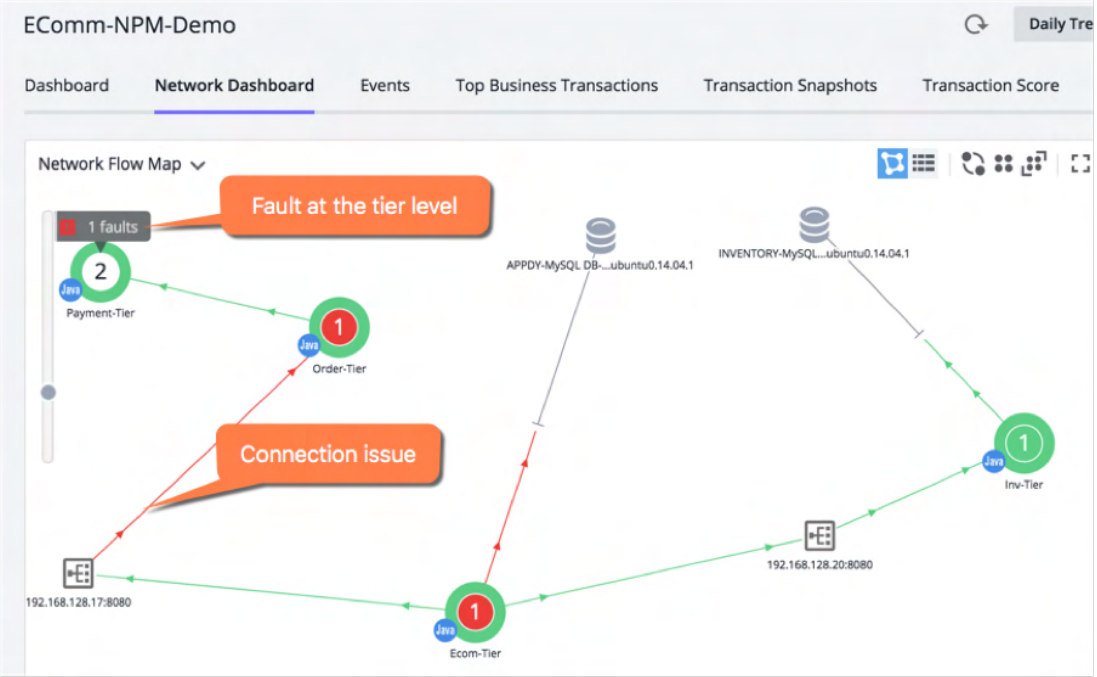
You can override the auto-map and map the application to another Cisco ACI tenant using the **Map Application to Tenant** panel. To override the auto-map, **Unmap** the application to tenant map and map it to another Cisco ACI tenant.

To revert to auto-map, **Unmap** the application to tenant map and refresh the page. AppDynamics auto-maps the application to the corresponding Cisco ACI tenant.

Troubleshoot Using the Integrated Solution

> AppDynamics is ending support for Application Centric Infrastructure (ACI). The feature will not work once customers upgrade their Controllers > 21.1. See Support Advisory: Application Centric Infrastructure (ACI) End of Life (EOL) Notice.

This page describes how to troubleshoot applications using the Cisco APIC integration. The AppDynamics Network Dashboard displays a network-layer view of the monitored application. Based on the runtime state of the application, AppDynamics automatically fetches the faults and connection issues and displays them in the network dashboard:



Faults display at the tier level. The number of faults displayed indicates an aggregate of faults occurring at all EPGs within the tier. You can view the number of faults at a given point of time in the Network Dashboard.

A connection is the set of all traffic for an application that has the same Source IP, Destination IP, Destination Port, and Protocol. AppDynamics detects if there is an issue in the traffic flow and indicates the connection issue on the network dashboard.

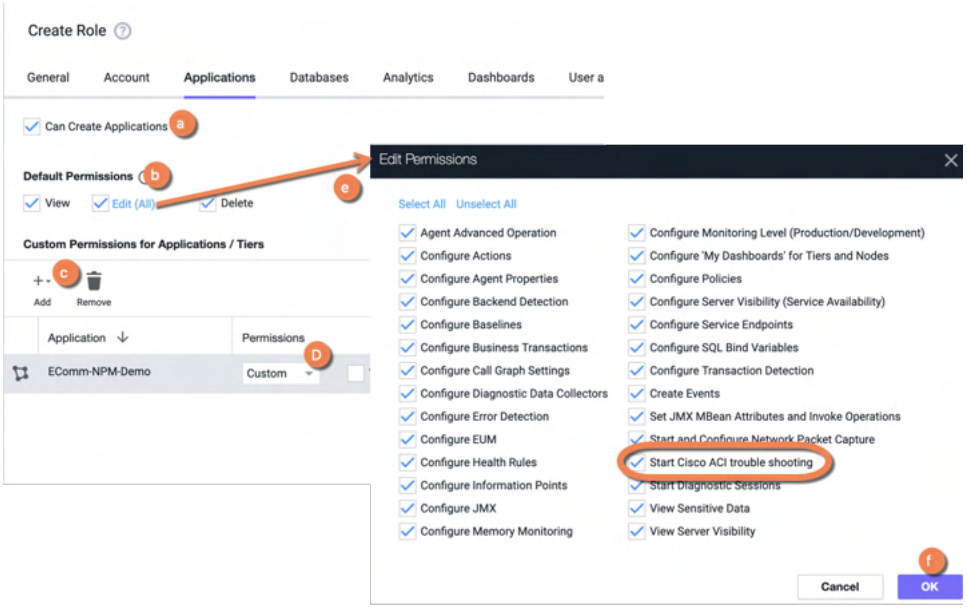
You can initiate a diagnosis to determine the exact location of the connection issue within the network by cross-launching the connection in Cisco APIC Controller.

Set Custom Permission to Cross-Launch Cisco APIC Controller

To include permission to cross-launch Cisco APIC Controller:

1. Log in to the Controller UI as an Administrator or Account Owner.
2. Click the gear icon ( ) > **Administration**.
3. Click the Roles tab.
4. Click **+ Create**.
5. Specify a **Name** and a **Description** for the newly created role.
6. Customize the business application level permission according to these guidelines:
    a. On the **Applications** tab, click **Can Create Applications**.
    b. Under **Default Permissions**, select **View**, **Edit,** and **Delete**.
    c. Click **+ Add** to add a new custom permission and click **Done**.
    The custom permission is added.
    d. Select **Custom** from the **Permissions** menu for the application (replacing the value of Inherited).
    e. Click **Edit(All)** and ensure that **Start Cisco ACI trouble shooting** is selected.

f. Click **OK**.



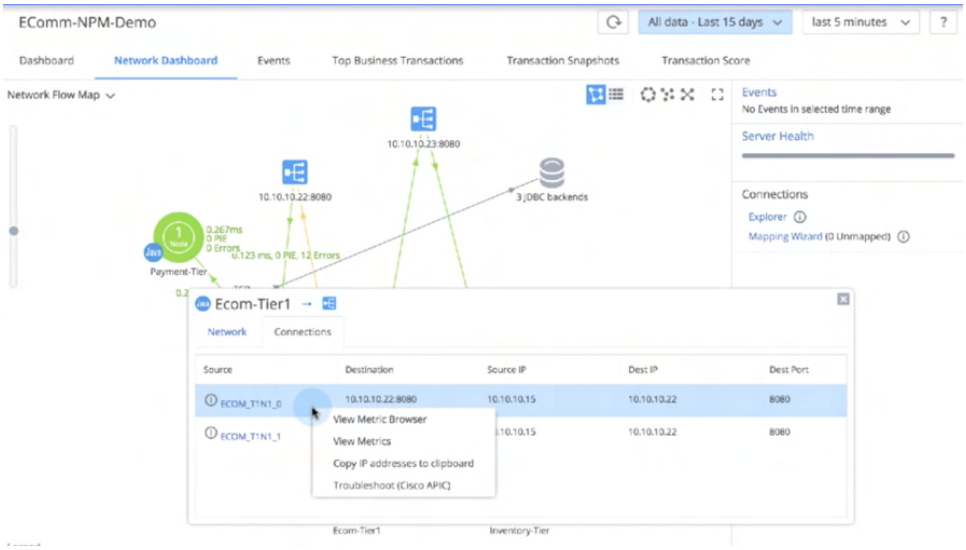
Customize Permissions to Cross-Launch a Connection Issue in Cisco APIC Controller

As an AppDynamics administrator, you can now customize the permissions for various roles to include permission to cross-launch a connection issue in the Cisco APIC Controller. See Set Custom Permission to Cross-Launch Cisco APIC Controller.

Launch Affected Connection in Cisco APIC

After collecting the connection issue information, you can cross-launch the connection in the Cisco APIC Controller.

1. Click on the affected connection on the **Network flow Map**.
2. Click the Connections tab.
3. Select the connection you want to troubleshoot.
4. Right-click on the selection and select **Troubleshoot (Cisco APIC)**.



5. Enter your Cisco APIC login credentials. After the initial entry, your credentials will be cached. The connection opens in the Cisco APIC Controller with the correct contracts (Visibility & Troubleshooting wizard) and the right set of endpoints.

> ⚠ If you are using the Cisco ACI Release 4.0, you need not enter the login credentials. The authentication happens automatically. You must enter the login credentials only if you are using a lower version of the Cisco ACI.

6. Troubleshoot the issue in Cisco APIC.

Integrate AppDynamics with Compuware Strobe

This page describes how to integrate AppDynamics with Compuware Strobe. This integration allows you to drill down from a SQL Call of a transaction snapshot to view and analyze the calls in Strobe.

Configure Compuware Strobe Integration

1. Log in to the Controller UI as an administrator.
2. Select **Settings > Administration**.
3. Select **Integration > Strobe**.
4. Click the **Enabled** checkbox.
5. For the **URL**, enter the Strobe URL and port number.
6. For the **Strobe LPAR**, enter the name of the logical partition where Strobe is running.
7. Click **Save**.

Launch Strobe from AppDynamics

You can access the **View in Strobe** option from a transaction snapshot.

1. Navigate to the Business Transactions dashboard.
2. From the **Transaction Snapshot** tab, select a transaction snapshot containing DB2 z/OS database access.
3. Select the **SQL Calls** tab.
4. Right-click a SQL query and select **View in Strobe**.

Integrate AppDynamics with DB CAM

Related pages:

- Access the Administration Console
- Monitor Databases

This page describes how to integrate AppDynamics with DB CAM. You can link to DB CAM for any DB CAM-monitored database that is discoverable by AppDynamics. This integration provides access to the database performance metrics provided by DB CAM.

Before You Begin

To use this integration you must have a DB CAM license. DB CAM must be configured to monitor the databases that you want to link to from AppDynamics.

Configure AppDynamics to Interface with DB CAM

You configure DB CAM integration at the account level and app agent level.

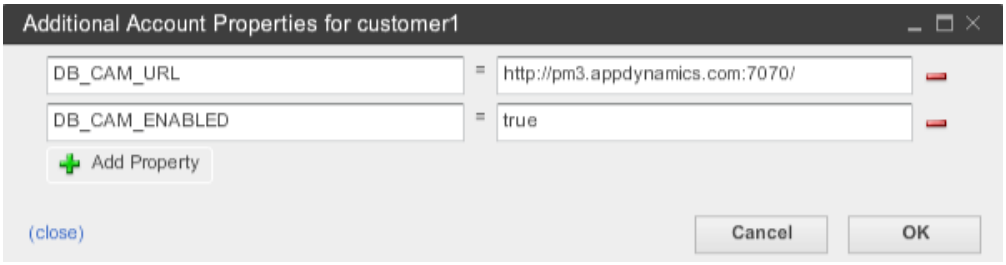
Configure DB CAM at the Account Level

Configure the integration at the account level using the Administration Console at `<host>:<port>/controller/admin.jsp`.

Create two new properties as name-value pairs in each account for which you want to enable DB CAM integration.

Configure One AppDynamics Account for DB CAM Integration

1. Log in to the Administrator Console with the administrator root password.
2. Select **Accounts**.
3. Double-click the account for which you want to configure DB CAM integration.
4. Click **Additional Account Properties** in the upper right.
5. Click **Add Property**.
6. In the left dialog, enter "DB_CAM_URL".
7. In the right dialog, enter the URL of the AppDynamics Controller that you are configuring using the syntax: http[s]://<host>:<port>
8. Click **Add Property** again.
9. In the left dialog, enter "DB_CAM_ENABLED".
10. In the right dialog, enter "true".
11. Click **OK**.
12. Log out of the Administrator Console.



Configure DB CAM at the Agent Level

Perform this procedure for each app agent for which you want to enable access to deep diagnostics from DB CAM:

1. Open the `AppServerAgent/conf/app-agent-config.xml` file for the app agent.
2. Locate the `TransactionMonitoringService` element, `<agent-service name="TransactionMonitoringService" enabled="true">`.
3. Add the `jdbc-dbcam-integration-enabled` property for the service:

```
<agent-service name="TransactionMonitoringService" enabled="true">
    <service-dependencies>BCIEngine,SnapshotService</service-dependencies>
        <configuration-properties>
            <property name="jdbc-dbcam-integration-enabled" value="true"/>
```

4. Save the file.

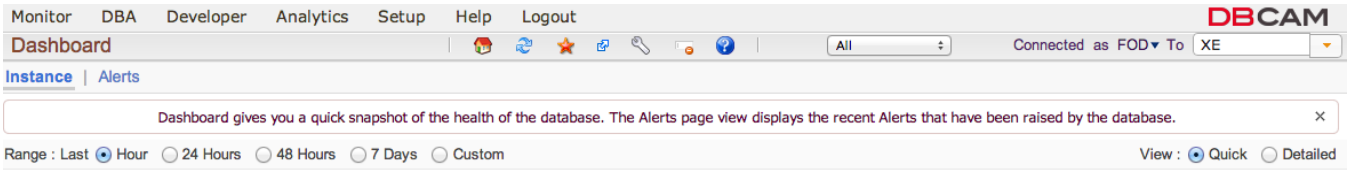
Link to DB CAM from AppDynamics

You can link to DB CAM from any AppDynamics flow map that displays a discovered DB CAM-monitored database. The flow map could be in a dashboard or a transaction snapshot.

If you link to DB CAM from a dashboard, you gain access to the DB CAM instance dashboard. If you link to be DB CAM from a transaction snapshot, you gain access to the DB CAM Session Drill Down pane.

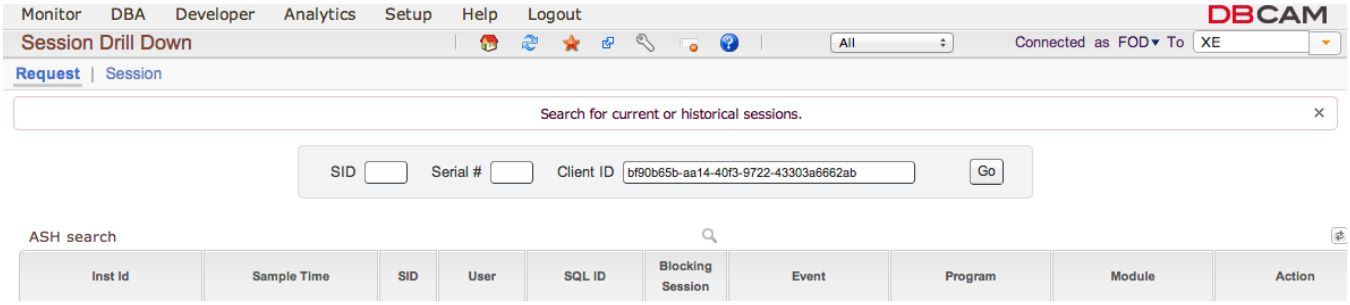
To Link to DB CAM from a Dashboard

1. In the flow map of a dashboard, right-click on the link below a database icon.
2. Select **Search in DBCam**. DB CAM launches and displays the instance dashboard for the selected database.



To Link to DB CAM from a Transaction Snapshot

1. In the flow map of a transaction snapshot, right-click on the link below the database icon.
2. Select **Search in DBCam**. DB CAM launches and displays the session drill down for the selected database.



Integrate AppDynamics with Scalyr

This page describes how to integrate AppDynamics with the Scalyr module. This integration gives you a single entry point for viewing and analyzing data gathered by AppDynamics and Scalyr. With the Scalyr integration module enabled, you can launch Scalyr searches directly from the AppDynamics Console.

The Scalyr search includes context from the AppDynamics UI session, including the time range and IP or hostname of the node being investigated in the AppDynamics UI.

Enable the Scalyr Integration Module

To enable and configure the Scalyr integration:

1. Log in to the Controller UI as an administrator.
2. Select **Settings > Administration**.
3. Select **Integration > Scalyr**.
4. Click the **Enabled** checkbox.
5. For the **URL**, enter `https://www.scalyr.com`.
6. Click **Save**.

Search Scalyr Data from AppDynamics

To launch a Scalyr search, Controller UI users must have:

- An active browser session in the Scalyr UI.
- Scalyr credentials.
- Popups permitted in the browser.

Controller users can search Scalyr logs from the node dashboard, business transaction dashboard, or from segments of a transaction snapshot. The time frame passed to Scalyr varies for each context.

Node Dashboard Access

1. Navigate to a node dashboard.
2. Select **Actions > Search Scalyr**.

This passes a search for that node with the timeframe currently selected in the Controller UI.

Business Transaction Dashboard

1. Select the Transaction Snapshot tab.
2. Select **Actions** or right-click a transaction snapshot**.**
3. Select **Search Scalyr**.

This generates a search on the nodes where the transaction occurred within the general time frame of the transaction execution. The time frame spans from 30 minutes before the transaction start time to 30 minutes after.

Call Drilldown Access

1. Select a **Call** in the drilldown view.
2. Select **Actions > Search Scalyr**.

This generates a search on the machine on which the snapshot was generated for the time frame of that segment.

Integrate AppDynamics with ServiceNow CMDB and Event Management

This page describes how to integrate the AppDynamics for ServiceNow® application with ServiceNow®.

Download

To download, visit the AppDynamics Community Exchange.

Version 20.7

AppDynamics has released a new version of the sync utility to integrate with the Orlando Release of ServiceNow®. To review version-specific configuration details, see New Version 20.7.

Please review this version compatibility matrix to guide your installation and upgrade decision:

Integration Version	AppDynamics	ServiceNow®
20.7	>= 4.5.0.1	Orlando and later
3.1	>= 4.5.0.1	New York and earlier

Use Case

AppDynamics Application Performance Monitoring traces every transaction and builds real-time application topology. The ServiceNow® integration provides these capabilities:

- AppDynamics application topology feeds data into the *AppDynamics for ServiceNow®* application as custom tables in the CMDB.

  > In this version, monitored servers can be reconciled with the ServiceNow® appropriate CMDB Server CIs.

- AppDynamics can send alerts to the *ServiceNow® Event Management* application and use the AppDynamics created entities. The AppDynamics for ServiceNow® application imports the AppDynamics alerts as entities. These entities can be used only when they are added to the ServiceNow® table. The AppDynamics for ServiceNow® application then creates a ServiceNow® table that maps with the entities.

This table lists these entities along with the ServiceNow® table that is created to represent them:

AppDynamics Entity Installed	ServiceNow® Table Created
Application	`x_apd_appdynamics_application`
Business Transaction	`x_apd_appdynamics_business_transaction`
Controller	`x_apd_appdynamics_controller`
Database	`x_apd_appdynamics_database`
Node	`x_apd_appdynamics_node`
Remote Service	`x_apd_appdynamics_remote_service`
Tier	`x_apd_appdynamics_tier`

Before You Begin the Integration

This integration requires that you download, install, and complete the following tasks, in this order, before starting the integration.

1. Download and install the AppDynamics for ServiceNow® application from the ServiceNow® Store.
   This application creates the custom AppDynamics tables in the Configuration Management Database (CMDB).
2. Download, install, and configure the Data Sync Utility.
   This utility makes API calls to AppDynamics and ServiceNow® to push our topology into the CMDB tables.
3. Once the data sync utility has been configured, go back to the utility and choose to download and install the Alerting template.
   The template contains the proper settings for AppDynamics to send alerts to the event service in ServiceNow®.
4. Install Java >= 1.8 on the machine that runs this program.
5. Start Java with 512 MB of initial memory (`-Xms512m`).

Add AppDynamics Discovery Source to ServiceNow®

You must create an AppDynamics Discovery Source before installing the integration from the ServiceNow® Store:

1. Log in to the ServiceNow® store.
2. In the left menu, select **System Definition** > **Choice Lists**.
3. Click **New**.
4. Specify the following details only. Do not change the values of other fields.

Table	`Configuration Item [cmdb_ci]`
Element	`discovery_source`
Label	`AppDynamics`
Value	`AppDynamics`

Download AppDynamics for ServiceNow® Application

Download the AppDynamics for ServiceNow® application from the ServiceNow® store and install it on your ServiceNow® instance.

> Ensure that you follow the instructions under the *System Requirements* and *Other Requirements* sections of the ServiceNow® download page.

To use Node-to-Server CI Reconciliation in ServiceNow® >= 3.1, you must have:

- AppDynamics Machine Agents with Server Visibility enabled installed and running on each host supporting AppDynamics Nodes to be imported.
- AppDynamics Controller and Machine Agent >= 4.5.0.1 with Server Visibility enabled.
- If you have issues, see Troubleshooting.

Download AppDynamics-ServiceNow® Data Sync Utility

Download the Data Sync Utility zip file from the AppDynamics Exchange.

You can download this utility either as a standalone Java application or as a JAR file. The standalone Java application can run on a Windows or Linux host and the JAR file can run as a Java program on any host machine.

> This host must have connectivity to both the target ServiceNow® instance and the target AppDynamics Controller.

The downloaded file includes a lightweight web server that has a basic web application to configure the integration. You can also run the integration in CLI mode only.

Server Options

Download and unzip the `appdynamics-cmdb-service-$version.zip` Data Sync Utility file.

1. Unzip the `appdynamics-cmdb-service-$version.zip` file.
2. Start the Server.
    a. **init.d service (System V)**

    ```
    sudo ln -s /path/to/appdynamics-cmdb-service/appdynamics-cmdb-service.jar /etc/init.d/appdynamics-cmdb-service
    sudo /etc/init.d/appdynamics-cmdb-service start
    ```

    The java options can be configured in the `$HOME/appdynamics-cmdb-service.conf` file.
    The application logs are created in the `$HOME/logs` directory.
    The system out logs are created in the `/var/log/appdynamics-cmdb-service.log` file.

    b. **Windows (winsw):**

    Open a terminal and navigate to the `appdynamics-cmdb-service` folder where you extracted the zip file:

```
cd C:\Path-to\appdynamics-cmdb-service
```

Run the following command:

```
javaw -jar appdynamics-cmdb-service.jar
```

The Java options can be configured in the `%$HOME%/bin/appdynamics-cmdb-service-win.xml` file.

The logs are created in the `$HOME/logs` directory.

c. **Other platforms:**

```
java -Xms512m -jar /path/to/appdynamics-cmdb-service.jar
```

The application logs are created in the `$HOME/logs` directory.

Upgrades

1. Backup the data directory `data/` where the java sync utility is installed.
2. Overwrite the existing installation with the new file.

Setup

1. init.d service: The configuration file is located at `$HOME/appdynamics-cmdb-service.conf`. Add the params to the `JAVA_OPTS` property.
2. Windows service: The configuration file is located at `$HOME/bin/appdynamics-cmdb-service-win.xml`. Add the params to the `<arguments>` property.
3. jar: Add the params directly to the command preceding `-jar ...`

Options

- Use a different port:

```
-Dserver.port=8080
```

- Enable authentication:

```
-Dplatform.security.enabled=true  # This will create a local login with the following credentials.
-Dplatform.username=user
-Dplatform.password="deFAultPwd4P1atfrm"
-Dplatform.security.encryption-key=mykey
-Dplatform.security.encryption-salt=mysalt
```


The `encryption-key` and `encryption-salt` are optional.

Configure the Data Sync Utility

User Roles and Permissions

The Sync Utility needs to communicate with both an AppDynamics Controller and a ServiceNow® instance. We recommend using a service account for this access:

- **AppDynamics Controller:**

The service account user requires these roles with rights to all applications to be synchronized.

  - Applications and Dashboards Viewer

    - Server Monitoring User—Optional if you want to enable node-to-server CI reconciliation

- **ServiceNow®**: These roles should be assigned to the ServiceNow® user account that will be posting data to ServiceNow®:

  > This user can be set to **web service access only** on the user form.

- If Event Management is activated in your ServiceNow® instance, add these roles:
    - `x_apd_appdynamics.appdynamics_role`
    - `evt_mgmt_user`
    - `evt_mgmt_integration`
    - `mid_server`—Optional if you want to do server CI reconciliation.
    - `app_service_admin`—Required for sync utility v20.7.
- If Event Management is *not* activated, add these roles:
    - `x_apd_appdynamics.appdynamics_role`
    - `itil`
    - `mid_server`—Optional if you want to do server CI reconciliation.
    - `app_service_admin`—Required for sync utility v20.7.

Domain Separated ServiceNow® Instances

If your ServiceNow® instance is domain-separated, a unique user account is required for each domain you want to update in the CMDB. For each domain:

1. Create a user account as described in User Roles and Permissions with the required roles.
2. Assign this user to the appropriate domain for the applications that it needs to access.
3. Create a new ServiceNow® instance with the appropriate user for each domain.
4. Set up synchronization for the applications appropriate for each domain user.

Synchronize with the CMDB

The integration supports the synchronization of one or more ServiceNow® instances with one or more AppDynamics Controllers.

1. Log in to the Sync Utility.
    a. Open `http://<host>:8080` on a browser.
    b. The default username is `user` and the default password is `deFAultPwd4P1atfrm`.

    > This can be overridden by the java startup property `-Dplatform.password=welcome`. See Server Options.

2. Select the **Service Model Integration** left menu bar.
3. Click **AppDynamics Controllers** and add your Controller(s).

    > If your Controller is on-premises or dedicated SaaS (single-tenant), the **Account Name** will be `customer1.`

4. Click **ServiceNow® Instances** and add your ServiceNow® instance(s).
5. Click **Synchronize** from the top menu.
6. Choose applications from the **Applications** list to export to ServiceNow®.
7. Select which **AppDynamics Relationships to Synchronize** to include:
    - **Tier to Tier**
    - **Tier to Remote** (Remote Services)
    - **Tier to Application**
    - **Node-to-Server CI**
        - Reconciles the node host with an existing Server CI and builds a **Runs** relationship with the `x_apd_appdynamics_node` runni ng on it. See Node-to-Server CI Reconciliation.
    - **Create Business Service**
        - Creates a Business Service CI (`cmdb_ci_service`) and builds a **Runs** relationship to the `x_apd_appdynamics_applicati` on CI. See Create Business Service CI.
8. Click **Run Diagnostics** to begin verification without creating any data.
    - If successful, click **Synchronize** to run a one-time synchronization.
    - If not successful, address the issues in the diagnostic messages provided.

Most integration users may like the synchronization to run at regular intervals to keep the service models. The integrated scheduler function allows for the creation and running of synchronization automatically.

UI Tab	Description

Schedules	Allows schedule creation using a cron expression. The **Synchronize** options on this page are the same as the other configuration panes.
History	Provides a graphical view of prior runs. You can expand each run to get details of the run, including error messages.
Settings	Provides additional settings for connection timeouts, SSL, proxy configuration, and additional static values that can be added to fields in various CMDB tables.

After the synchronization is complete, log in to ServiceNow® instance and select the AppDynamics menu item to view the tables and data associated with the integration.

Node-to-Server CI Reconciliation

In >= 3.1, the **Node-to-Server CI** checkbox in the sync utility builds a relationship between an AppDynamics Node (`x_apd_appdynamics_node`) and the underlying host Server CI (`cmdb_ci_linux_server, cmdb_ci_windows_server`) in the ServiceNow® CMDB.

To use this feature, a Machine Agent with Server Visibility enabled must be installed and running on each host that supports an imported AppDynamics Node. See Machine Agent with Server Visibility.

The integration uses the default ServiceNow® hardware identification rules for reconciliation:

- **Name**: AppDynamics uses the operating system-reported hostname as the value for this field, which is subject to the following **Discovery Property** settings in ServiceNow®, whether Discovery is installed or not:

  The `mid_server` role must be assigned to the ServiceNow® user for the integration to read these properties.

  - `glide.discovery.hostname.case`
  - `glide.discovery.hostname.include_domain`
  - `glide.discovery.fqdn.regex`

  Refer to the ServiceNow® product documentation for more information about these properties.

- **IP Address + MAC address** (Network Adapter Table): AppDynamics uses any adapter with a valid IPv4 or IPv6 address and a MAC address.

These attributes are provided by Machine Agents with Server Visibility enabled for Controller and Machine Agent >= 4.5.0.1.

1. To avoid duplicate Server CIs being created by this integration, any discovery sources employed must adhere to the same hardware identification rules:
   a. **Name** in the Server CI should match the system reported hostname subject to the Discovery properties described above.
   b. For each **Server Network Adapter**, the `ip_address` and `mac_address` must be populated in the Network Adapter table and related to the Server CI.

   ServiceNow® CMDB Identification and Reconciliation will not consider IP Address and MAC Address values populated on the Server CI form directly.

2. This integration identifies Linux and Windows servers and reconciles them to their respective CI classes on update or create (`cmdb_ci_linux_server & cmdb_ci_windows_server`).

   If a server is not Windows or Linux, no CMDB relationships will be created.

Fields Populated in ServiceNow® by Node-to-Server CI Reconciliation

For the appropriate Server Class (`cmdb_ci_*_server`) and related to the parent AppDynamics Node:

ServiceNow® Server CI Field	AppDynamics Server Visibility Attribute Used
Name (`name`)	The reported Hostname—the Sync Utility reads and uses the discovery properties listed above to determine the hostname.
Hostname (`host_name`)	The reported Hostname.

Each Network Adapter reported populates the Network Adapter CI (`cmdb_ci_network_adapter`) and related parent Server CI:

ServiceNow® Network Adapter CI Field	AppDynamics Server Visibility Attribute Used
IP Address (`ip_address`)	IP Address with /x removed. Using either IPv4 or IPv6 Address as reported. If neither is populated, the adapter is not synced.
MAC Address (`mac_address`)	The reported MAC address.

Add Static Values to Required Fields

If there are required fields in your CMDB that AppDynamics does not populate, you can use the **CMDB Additional Fields** to add multiple static values per table.

1. Navigate to the AppDynamics-ServiceNow® Sync Utility.
2. Select **Settings.**
3. Select **CMDB Additional Fields.**
4. For the desired CI table, click **Add Field**.
5. Enter the ServiceNow® **Field Name** (not the label).
6. Enter the static value to populate all records of this type. For example, if you are required to populate the `hardware_status` field on all `cmdb_ci_server` and child class records, enter "`hardware_status`" and then "`Installed`".
7. Click **Save**.

Considerations for Dynamic Application Environments

If the application environments you are monitoring with AppDynamics change frequently, you may end up with artifacts in the ServiceNow® CMDB that no longer reflect the current state of your application topology.

In order to address this, ServiceNow® provides documentation on how to configure "data refresh rules" in the CMDB so that irrelevant AppDynamics CIs can be removed. For the New York release, see Create data refresh rules.

Create Business Service CI

AppDynamics Application to ServiceNow® Application Service Relationships

Health rule violation alerts related to a ServiceNow® Application Service are shown in the Event Management Dashboard. There are two ways to build this relationship based on the current state of the ServiceNow® CMDB:

If Business Services do not exist:

1. If there are no Business Service CIs in the CMDB, ensure that the **Create Business Service** checkbox is checked when initially synchronizing an application. Follow the instructions in the Sync Utility documentation that explain how to convert that CI to an Application Service (`cmdb_ci_service_discovered`) CI that can be shown on the ServiceNow® Event Management Dashboard.
   You must select six (6) levels of CIs to include in the model.

This step requires the `app_service_admin` role. See the ServiceNow® Convert a business service to an application service.

If Business Services exist:

If the Business Service entities are Application Service class CIs, a relationship can be manually created between the AppDynamics Application and the ServiceNow® Application Service. To create the relationship:

1. Open the AppDynamics Application CI (`x_apd_appdynamics_application`).
2. Click **Add CI relationship.**
3. In the filter, set the "**Class**" - "**is**" - "**Application Service**".
4. Click **Run Filter**.
5. Select the **Application Service CI**.
6. Select the **Runs On::Runs** relationship.
7. Click **Create new relationships with selected configuration item(s)**.

AppDynamics Tier to ServiceNow® Application Service Relationships

In some cases, an AppDynamics Tier may be the correct entity to relate to a ServiceNow® Application Service. To build this relationship in the ServiceNow® CMDB:

1. Open the **AppDynamics Tier CI** (`x_apd_appdynamics_tier`).
2. Click **Add CI relationship.**

3. In the filter, set the "**Class**" - "**is**" - "**Application Service**".
4. Click **Run Filter**.
5. Select the **Application Service CI**.
6. Select the **Runs On::Runs** relationship.
7. Click **Create new relationships with selected configuration item(s)**.

Additional Application Relationships

The AppDynamics Application description and Tier description fields are populated in the AppDynamics Application CI (`x_apd_appdynamics_application`) and AppDynamics Tier CI (`x_apd_appdynamics_tier`) respectively:

Table	Application CI Field	AppDynamics Field Used
`x_apd_appdynamics_application`	Description (`short_description`)	Application description

Table	Tier CI Field	AppDynamics Field Used
`x_apd_appdynamics_tier`	Description (`short_description`)	Tier description

Troubleshooting

Logs

The logs are generated in the `logs/` directory where the application is installed.

Debug Logs:

To enable debug logging, edit the file `conf/log4j2.xml` and change the level of the logger `com.appdynamics` to `DEBUG`.

Read Timeouts

If you encounter any Read Timeout errors while doing the sync:

1. Increase the **Socket Timeout** value in the Settings tab up to `60,000` initially and higher if needed.
2. Reduce the **Sync Batch Size** of entities to ServiceNow® Instance.
3. Navigate to the `Installation` directory.
4. Find the `conf/application.properties` file and adjust the size value.
5. Restart the AppDynamics ServiceNow® Sync Utility service or daemon (Required).

Events Integration

This section assumes that you are familiar with the configuration of HealthRules, Policies, and Actions in AppDynamics. See Alert and Respond.

The Events integration uses the HTTP templates feature in the Controller to push events to the ServiceNow® Events API. The details of the ServiceNow® Events API can be found here. See HTTP Request Actions and Templates.

Prerequisites

- Download and install the AppDynamics for ServiceNow® application on your ServiceNow® instance.
- Run the AppDynamics-ServiceNow® Data Sync Utility and verify that the entities are imported into your ServiceNow® instance. See Data Sync Utility.
- Activate the ServiceNow® Event Management plugin.

Events Installation and Configuration

In this section, you will create an Action, configure Health Rule Violations, and create binding Policies.

Create an Action on the AppDynamics Controller

1. Log in to the web UI for the AppDynamics-ServiceNow® Data Sync Utility.
2. Select **Service Model Integration** > **Event Integration.**
3. Select the Controller you would like to configure. If one is not configured, see Data Sync Utility Configuration.
4. Click **Download Template.**

Use the downloaded file to configure the HTTP Request template in AppDynamics. You will then create an Action and apply that action to a policy that specifies a health rule and action. To create the new Action:

1. Navigate to the **Alert & Respond** top menu in your AppDynamics Controller.

2. Click **Alert & Respond** > **HTTP Request Templates** in the left navigation.
3. Select the resource on which the action is to be created.
4. Click **New** and fill out the following fields:
    a. **Name**: Any name to uniquely identify the template
    b. **Request URL**
        i. **Method**: `POST`
        ii. **Raw URL**: `https://<your-instance>.service-now.com/em_event.do?JSONv2&sysparm_action=insertMultiple.` Replace `<your-instance>` with the id of the actual instance.
    c. **Authentication**
        i. **Type**: `BASIC`
        ii. Enter the `user name` and `password` of your ServiceNow® instance.

        > Ensure you have already configured the ServiceNow® User and Roles as described in User Roles and Permissions.

    d. **Payload**
        i. **MIME Type**: `application/json`
        ii. Copy the contents from the downloaded file, `event-request-template.txt`, and then paste it in the **Payload** text area
        iii. Update the `controllerName` in the first line of the file. It must be the same value that you set while adding the `AppDynamics Controller Name` in the Sync Utility.
5. Save the HTTP Template.

Configure Health Rules on the AppDynamics Controller

1. Select the Alert & Respond tab.
2. Select **Health Rules**.
3. Use the **Select** dropdown to choose an application on which you want to alert.
4. Review the list of **Health Rules** configured out-of-the-box for that application. Add more **Health Rules** if needed.

Create a Policy on the AppDynamics Controller to Bind Health Rules to Actions

1. Select **Alert & Respond** > **Policies**.
2. Select **Create a Policy Manually** if there are no policies configured already.
3. Select the type of scenarios on which you want to receive alerts.

    > Checking all the boxes might lead to an alert storm on your ServiceNow® instance.

4. Click **Save**.
5. Select the **Health Rule Scope** defining which tiers and nodes the Health Rule should cover and click **Save**.
6. Select the **Object Scope** and choose the **Tiers and Nodes** to be covered in this policy and click **Save**.
7. Select **Actions**.
8. Click **Add +** sign to create a new **Action**.
9. Select the ServiceNow® **Action** you created Create an Action on the AppDynamics Controller and click **Save**.

Create a Manual Entry Point for the AppDynamics Monitored Application on the Application Service CI

> If you are connecting to a ServiceNow® instance running Orlando or later, skip this section and refer to the New Version 20.7 instructions.

For AppDynamics alerts to impact the health of an application service in ServiceNow® Event Management, you must create a manual entry point.

1. If you chose to have the Sync Utility create a Business Service:
    a. Make sure you convert the service to an application service by following the prompts in the sync utility after synchronizing the application.
    b. This conversion will add the manual entry point automatically.
2. If you already have Application Service CIs that you want to associate with an AppDynamics monitored application (or application tier):
    a. Open the **Application Service CI** record.
    b. Click **Add Entry Point**.
    c. Select the AppDynamics monitored **Application** or **Tier CI Type**.
    d. Select the name of the AppDynamics monitored application or tier to associate.
    e. Click **Save**.
    f. Click the **Additional Info** link.
    g. Click the **Update with changes from CMDB UI Action** link.
    h. Select the number of levels of related CIs to include in the new application service.
    i. Click **OK**.

New Version 20.7

Requirements

Versions

- AppDynamics >= 4.5.0.1 is required.
- ServiceNow® instances to be synchronized must be running Orlando or greater.

New Roles

- No additional roles are needed for the service account in AppDynamics.
- The service account in ServiceNow® will also need the `app_service_admin` role.

Upgrade to Version 20.7

To upgrade an existing sync utility, follow the upgrade section. Below are some answers to questions about upgrading an existing sync utility:

What happens to existing applications that have already been synchronized?

Existing AppDynamics monitored applications and related entities that have been synchronized to a ServiceNow® CMDB from previous versions of the sync utility will remain in the CMDB. When the same application is re-synchronized using 20.7, a calculated service CI is created for those applications (or tiers) as configured. See Creating Calculated Application Service CIs.

What happens to existing manual entry points?

If you created manual entry points for Application Service CIs using AppDynamics monitored application CIs `x_apd_appdynamics_application` will remain as-is. After synchronizing using 20.7 of the sync utility, a new calculated application service CI will be added to the relationships.

AppDynamics recommends that you remove any previously created entry points and create a manual entry point to the new Calculated Application Service `cmdb_ci_service_calculated` CI. To do this:

1. To remove the existing manual entry point:
    a. Open the **Application Service CI** you are monitoring with Event Management
    b. Click - next to the **Manually Added CI** link and click **Remove**.
2. Add a manual entry point for the calculated application service:
    a. Open the application service CI you are monitoring in ServiceNow.
    b. Click **Add Entry Point**.
    c. Select **Manually Created**.
    d. For **CI Type**, select **Calculated Application Service**.
    e. For **CI Name**, select the name for the application you synched.

Create Calculated Application Service CIs

Orlando introduces a new CI class called Calculated Application Service `cmdb_ci_service_calculated`. This new class is dynamic and provides additional capability to automatically update a service map when relationships in the CMDB change for CIs that are part of the application service.

Some AppDynamics customers monitor applications as Tiers. With the new version 20.7, you can map AppDynamics Applications or AppDynamics Tiers to `cmdb_ci_service_calculated` CIs in ServiceNow®.

Select Tiers to Monitor as Application Services in ServiceNow®

By default, all applications monitored by an AppDynamics controller relate to a new calculated application service when synchronized for the first time with 20.7 with a **Used By::Depends On** relationship.

To sync monitored tiers to a calculated Application Service CI in ServiceNow®:

1. Navigate to **Application Mappings**.
2. Add a Controller if none exists, or select the Controller from the dropdown. A list of all the AppDynamics monitored applications in the selected Controller display in the left panel.
3. To map each tier of a monitored application to a Calculated Application Service CI, move that application to the right bucket as an **Application Group**.

Synchronize the Application Services in ServiceNow®

ServiceNow® requires that each Calculated Application Service CI (`cmdb_ci_service_calculated`) has a unique name. When synchronized for the first time using 20.7, a new Calculated Application Service CI is created with this naming convention:

- By default, each monitored application will have a Calculated Application Service CI named:
    - AppD - AppNameGoesHere (`controllerName`)
- If an application is designated as an Application Group, each monitored tier will have a Calculated Application Service CI named:
    - AppD - AppnameGoesHere:TierNameGoesHere (`controllerName`)

AppDynamics Application & ServiceNow Calculated Application CI Dashboard

Use the **View Mapped Applications** dashboard to view the mappings between AppDynamics Applications (and Tiers) and ServiceNow® Calculated Application CI.

To delete a mapping from ServiceNow®, click the red '**X**' to delete the mapping from this panel. This will delete both the Calculated Application CI and the relationship displayed. The AppDynamics entities in the CMDB will remain.

Event Integration Enhancements

The Event Integration Template has several enhancements in the 20.7 release. Alerts are more easily bound to Business Transaction and Node CIs.

Support

For questions or feature requests, please contact AppDynamics Help.

Integrate AppDynamics with Splunk

Related pages:

- Splunkbase Apps
- Splunk documentation

This page describes how to integrate AppDynamics and Splunk. This integration provides a single, cohesive view of data and allows you to:

- Launch Splunk searches using auto-populated queries from the AppDynamics Console based on criteria such as time ranges and the node IP address.
- Push notifications on policy violations and events from AppDynamics to Splunk.
- Mine performance data from AppDynamics using the Controller REST API and push it into Splunk.

Configure Splunk Integration

1. Log in to the Controller UI as an administrator.
2. Select **Settings > Administration**.
3. Select **Integration > Splunk**.
4. Click the **Enabled** checkbox.
5. For the **URL**, enter the Splunk URL and port number.
6. Optionally, enter **Extra Query Parameters**. These parameters are appended to each Splunk search initiated from AppDynamics.
7. Click **Save**.

Launch a Splunk Search from AppDynamics

You can launch a search of Splunk logs for a specific time frame associated with a transaction snapshot from several places in AppDynamics.

To launch a Splunk search:

- You need Splunk credentials. You will only enter your credentials the first time that you launch a Splunk search. Your credentials are cached by the browser after the first login.
- Ensure the Splunk Server is running.
- Configure your browser to allow popups.

> **Enable Pop-ups**
>
> If you do not see a login prompt at first login, either your browser is blocking the Splunk login popup or the Splunk Server is not running.

You can access the **Search Splunk** option from the node dashboard or the business transaction dashboard.

Node Dashboard Access

1. Navigate to a node dashboard.
2. Select **Actions** > **Search Splunk**.

Business Transaction Dashboard

1. Select the **Transaction Snapshot** tab.
2. Right-click a transaction snapshot.
3. Select **More Actions**.
4. Select **Search Splunk**.

Product and Release Announcements

This table lists the agent, Controller, and on-premises releases. Note that some releases will not have resolved issues or enhancements. For other types of announcements, see Product Announcements and Alerts Home.

Date	Announcement
May 31, 2021	21.4.0 (Synthetic Server) released. See 21.4 On-premises Platform Enhancements.
May 28, 2021	21.5.0 (.NET Agent) released. See 21.5 Agent Enhancements.
May 28, 2021	21.5.0 (Node.js Agent) released. See 21.5 Agent Enhancements.
May 27, 2021	21.5.0 (Analytics Agent) released. See 21.5 Agent Enhancements.
May 26, 2021	21.5.0-2052 (Cluster Agent) released. See 21.5 Agent Enhancements.
May 25, 2021	21.5.0 (Private Synthetic Agent - K8S Container-Based Agent) released. See Past On-premises Platform Releases.
May 18, 2021	21.5.1 (Xamarin Agent) released. See 21.5 Agent Enhancements.
May 18, 2021	21.5.0 (Cordova Plugin) released. See 21.5 Agent Enhancements.
May 18, 2021	21.5.0-3130 (Machine Agent) released. See 21.5 Agent Enhancements.
May 17, 2021	21.5.0 (Android Agent) released. See 21.5 Agent Enhancements.
May 17, 2021	21.5.0 (General) released. See 21.5 Controller Enhancements and Controller Resolved Issues.
May 12, 2021	21.5.0 (iOS Agent) released. See 21.5 Agent Enhancements.
May 10, 2021	AppDynamics AWS Lambda Extension for Serverless APM. See Use the AppDynamics AWS Lambda Extension to Instrument Serverless APM at Runtime.
May 10, 2021	21.5.286 (Node.js Serverless Tracer) released. See 21.5 Agent Enhancements.
May 3, 2021	21.2.7-24387 (Enterprise Console) released. See 21.2 On-premises Platform Enhancements and 21.2 On-premises Platform Resolved Issues.
April 29, 2021	21.4.3-24599 (Enterprise Console) released. See 21.4 On-premises Platform Enhancements and 21.4 On-premises Platform Resolved Issues.
April 28, 2021	21.4.0 (.NET Agent) released. See 21.4 Agent Enhancements.
April 25, 2021	21.4.0 (Experience Journey Map) UIC released. See 21.4 Controller Enhancements.
April 26, 2021	21.4.0 (ThousandEyes Integration with AppDynamics) released. See 21.4 Controller Enhancements.
April 23, 2021	21.2.6-24377 (Enterprise Console) released. See 21.2 On-premises Platform Enhancements and 21.2 On-premises Platform Resolved Issues.
April 22, 2021	21.4.2-24588 (Enterprise Console) released. See 21.4 On-premises Platform Enhancements and 21.4 On-premises Platform Resolved Issues.
April 22, 2021	21.4.0-385 (Analytics Agent) released. See 21.4 Agent Enhancements.
April 22, 2021	21.4.0-3075 (Machine Agent) released. See 21.4 Agent Enhancements.
April 21, 2021	21.4.0-2358 (Database Agent) released. See 21.4 Agent Enhancements.
April 21, 2021	21.4.0 (Java Agent) released. See 21.4 Agent Enhancements.
April 16, 2021	21.4.0 (Node.js Agent) released. See 21.4 Agent Enhancements.
April 16, 2021	21.4.0 (JavaScript Agent) released. See 21.4 Agent Enhancements.
April 12, 2021	21.3.1 (.NET Agent) released. See Past Resolved and Known Issues by Release.
April 7, 2021	21.4.0 (Apache Agent) released. See 21.4 Agent Enhancements.
April 5, 2021	21.4.0 (IBM Integration Bus Agent (IIB) Agent) released. See 21.4 Agent Enhancements.
April 5, 2021	21.4.0 (General) released. See 21.4 Controller Enhancements and 21.4 Controller Resolved Issues.
April 5, 2021	21.4.0 (React Native) released. See 21.4 Agent Enhancements.
March 29, 2021	21.3.0 (.NET Agent) released. See 21.3 Agent Enhancements and 21.3 Agent Resolved Issues.
March 26, 2021	21.3.278 (Node.js Serverless Tracer) released. See 21.3 Agent Enhancements and 21.3 Agent Resolved Issues.

March 26, 2021	21.3.1-2042 (Cluster Agent) released. See 21.3 Agent Enhancements and 21.3 Agent Resolved Issues.
March 25, 2021	21.3.0-3059 (Machine Agent) released. See 21.3 Agent Enhancements and 21.3 Agent Resolved Issues.
March 25, 2021	21.3.0-2038 (Cluster Agent) released. See 21.3 Agent Enhancements and 21.3 Agent Resolved Issues.
March 25, 2021	21.3.0-9590 (Analytics Agent) released. See 21.3 Agent Enhancements and 21.3 Agent Resolved Issues.
March 25, 2021	21.3.0-2181 (Network Agent) released. See 21.3 Agent Enhancements and 21.3 Agent Resolved Issues.
March 15, 2021	21.2.3-24315 (Enterprise Console) released. See 21.2 On-premises Platform Enhancements and 21.2 On-premises Platform Resolved Issues.
March 8, 2021	21.3.0 (Dash Studio) UIC release. See 21.3 Controller Enhancements.
March 8, 2021	21.2.2-24308 (Enterprise Console) released. See 21.2 On-premises Platform Enhancements and 21.2 On-premises Platform Resolved Issues.
February 26, 2021	21.2.0 (Synthetic Server) released. See 21.2 On-premises Platform Enhancements and 21.2 On-premises Platform Resolved Issues.
February 25, 2021	21.2.0-3052 (Machine Agent) released. See 21.2 Agent Enhancements and 21.2 Agent Resolved Issues.
February 25, 2021	21.2.0 (Node.js Agent) Released. See 21.2 Agent Enhancements and 21.2 Agent Resolved Issues.
February 25, 2021	21.2.1 (Analytics Agent) released. See 21.2 Agent Enhancements and 21.2 Agent Resolved Issues.
February 25, 2021	21.2.0 (Network Agent) released. See 21.2 Agent Enhancements and 21.2 Agent Resolved Issues.
February 24, 2021	21.2.0-1997 (Cluster Agent) released. See 21.2 Agent Enhancements and 21.2 Agent Resolved Issues.
February 24, 2021	21.2.1-24293 (Enterprise Console) released. See 21.2 On-premises Platform Enhancements and 21.2 On-premises Platform Resolved Issues.
February 24, 2021	21.2.0 (.NET Agent) released. See 21.2 Agent Enhancements and 21.2 Agent Resolved Issues.
February 24, 2021	21.2.0 (PHP Agent) released. See 21.2 Agent Enhancements and 21.2 Agent Resolved Issues.
February 22, 2021	21.2.0 (Enterprise Console) released. See 21.2 On-premises Platform Enhancements and 21.2 On-premises Platform Resolved Issues.
February 22, 2021	21.2.0 (General) released. See 21.2 Controller Enhancements.
February 19, 2021	21.2.1 (React Native Agent) released. See 21.2 Agent Enhancements and 21.2 Agent Resolved Issues.
February 19, 2021	21.2.1 (Cordova Plugin) released. See 21.2 Agent Enhancements and 21.2 Agent Resolved Issues.
February 19, 2021	21.2.1351 (Xamarin Agent) released. See 21.2 Agent Enhancements and 21.2 Agent Resolved Issues.
February 18, 2021	21.2.0 (Android Agent) released. See 21.2 Agent Enhancements and 21.2 Agent Resolved Issues.
February 18, 2021	20.11.9-23919 (Enterprise Console) released. See 20.x On-premises Platform Enhancements and 20.x Resolved and Known Issues.
February 17, 2021	21.2.0 (JavaScript Agent) released. See 21.2 Agent Enhancements and 21.2 Agent Resolved Issues.
February 17, 2021	21.2.0-2285 (Database Agent) released. See 21.2 Agent Enhancements and 21.2 Agent Resolved Issues.
February 16, 2021	21.2.0 (Dash Studio) UIC release. See 21.2 Controller Enhancements.
February 16, 2021	21.2.0 (iOS Agent) released. See 21.2 Agent Enhancements and 21.2 Agent Resolved Issues.
February 16, 2021	21.2.0 (Synthetic Monitoring: Hosted agents) released. See 21.2 Agent Enhancements and 21.2 Agent Resolved Issues.

February 15, 2021	21.2.0 (Java Agent) released. See 21.2 Agent Enhancements and 21.2 Agent Resolved Issues.
February 4, 2021	20.11.8-23902 (Enterprise Console) released. See 20.x Past On-premises Platform Releases and 20.x Past Resolved and Known Issues.
February 2, 2021	21.2.0 (Python Agent) released. See 21.2 Agent Enhancements and 21.2 Agent Resolved Issues.
January 29, 2021	21.1.0 (Node.js Agent) released. See 21.1 Agent Resolved Issues.
January 29, 2021	20.11.7-23894 (Enterprise Console) released. See 20.x Past On-premises Platform Releases and 20.x Past Resolved and Known Issues.
January 29, 2021	21.1.0 (.NET Agent) released. See 21.1 Agent Resolved Issues.
January 28, 2021	21.1.0 (Dash Studio) UIC release. See 21.1 Controller Enhancements.
January 28, 2021	21.1.1 (Java Agent) released. See 21.1 Agent Enhancements and 21.1 Agent Resolved Issues.
January 26, 2021	20.11.1-3044 (Machine Agent) released. See 20.x Past Agent Releases.
January 25, 2021	20.11.6-23887 (Enterprise Console) released. See 20.x Past On-premises Platform Releases and 20.x Past Resolved and Known Issues.
January 22, 2021	21.1.0-3041 (Machine Agent) released. See 21.1 Agent Enhancements and 21.1 Agent Resolved Issues.
January 18, 2021	21.1.0 (Java Agent) released. See 21.1 Agent Enhancements and 21.1 Agent Resolved Issues.
January 8, 2021	20.10.10-23638 (Enterprise Console) released. See 20.x Past On-premises Platform Releases and 20.x Past Resolved and Known Issues.
December 18, 2020	20.12.1-1948 (Cluster Agent) released. See 20.x Past Resolved and Known Issues.
December 18, 2020	20.11.5-23850 (Enterprise Console) released. 20.x Past On-premises Platform Releases and 20.x Past Resolved and Known Issues.
December 18, 2020	20.10.9-23633 (Enterprise Console) released. See 20.x Past On-premises Platform Releases and 20.x Past Resolved and Known Issues.
December 18, 2020	20.12.0 (.NET Agent) released. See 20.x Past Resolved and Known Issues.
December 18, 2020	20.12.0 (Python Agent) released. See 20.x Past Resolved and Known Issues.
December 17, 2020	20.12.0-1937 (Cluster Agent) released. See 20.x Past Resolved and Known Issues.
December 17, 2020	20.11.4-23846 (Enterprise Console) released. See 20.x Past On-premises Platform Releases and 20.x Past Resolved and Known Issues.
December 16, 2020	20.12.0-3016 (Machine Agent) released. See 20.x Past Agent Releases and 20.x Past Resolved and Known Issues.
December 16, 2020	20.12.0-2183 (Database Agent) released. See 20.x Past Agent Releases and 20.x Past Resolved and Known Issues.
December 15, 2020	20.12.0 (iOS Agent) released. See 20.x Past Agent Releases and 20.x Past Resolved and Known Issues.
December 11, 2020	20.12.0 (JavaScript Agent) released. See 20.x Past Agent Releases and 20.x Past Resolved and Known Issues.
December 8, 2020	20.12.0 (Node.js Agent) released. See 20.x Past Agent Releases and 20.x Past Resolved and Known Issues.
December 8, 2020	20.11.3-23827 (Enterprise Console) released. See 20.x Past On-premises Platform Releases and 20.x Past Resolved and Known Issues.
December 3, 2020	20.10.8-23600 (Enterprise Console) released. See 20.x Past On-premises Platform Releases and 20.x Past Resolved and Known Issues.

December 2, 2020	20.11.2-23819 (Enterprise Console) released. See 20.x Past On-premises Platform Releases and 20.x Past Resolved and Known Issues.
November 30, 2020	20.11.1-23806 (Enterprise Console) released. See 20.x Past On-premises Platform Releases and 20.x Past Resolved and Known Issues.
November 27, 2020	20.11.0 (Java Agent) released. See 20.x Past Agent Releases and 20.x Past Resolved and Known Issues.
November 25, 2020	20.11.0 (Python Agent) released. See 20.x Past Agent Releases and 20.x Past Resolved and Known Issues.
November 25, 2020	20.11.0-1877 (Cluster Agent) released. See 20.x Past Agent Releases and 20.x Past Resolved and Known Issues.
November 24, 2020	20.11.0 (PHP Agent) released. See 20.x Past Agent Releases and 20.x Past Resolved and Known Issues.
November 23, 2020	20.11.0 (SAP Agent) released. See 20.x Past Agent Releases and 20.x Past Resolved and Known Issues.
November 20, 2020	20.11.0 (React Native Agent) released. See 20.x Past Agent Releases and 20.x Past Resolved and Known Issues.
November 20, 2020	20.11.0 (.NET Agent) released. See 20.x Past Agent Releases and 20.x Past Resolved and Known Issues.
November 19, 2020	20.11.0 (Cordova Plugin) released. SSee 20.x Past Agent Releases and 20.x Past Resolved and Known Issues.
November 19, 2020	20.11.0-2915 (Machine Agent) released. See 20.x Past Agent Releases and 20.x Past Resolved and Known Issues.
November 19, 2020	20.11.0 (Xamarin Agent) released. See 20.x Past Agent Releases and 20.x Past Resolved and Known Issues.
November 18, 2020	20.11.0 (Android Agent) released. See 20.x Agent Enhancements and 20.x Resolved and Known Issues.
November 16, 2020	20.11.0 (General) released. See 20.x Past Controller Releases and Past Resolved and Known Issues by Release.
November 13, 2020	20.10.7-23585 (Enterprise Console) released. See 20.x Past Agent Releases and 20.x Past Resolved and Known Issues.
November 10, 2020	20.10.0 (Xamarin Agent) released. See 20.x Agent Enhancements and 20.x Resolved and Known Issues.
November 10, 2020	20.10.6-23577 (Enterprise Console) released. See 20.x Past On-premises Platform Releases and 20.x Past Resolved and Known Issues.
November 2, 2020	20.11-1400 (Java Serverless Tracer) released. See 20.x Past Agent Releases and 20.x Past Resolved and Known Issues.
November 2, 2020	20.11-242 (Node.js Serverless Tracer) released. See 20.x Past Agent Releases and 20.x Past Resolved and Known Issues.
November 2, 2020	20.11-411 (Python Serverless Tracer) released. See 20.x Past Agent Releases and 20.x Past Resolved and Known Issues.
November 2, 2020	20.10.5-23565 (Enterprise Console) released. See 20.x Past Agent Releases and 20.x Past Resolved and Known Issues.
November 2, 2020	20.10.5-23565 (Application Analytics) released. See Past On-premises Platform Releases and Past Resolved and Known Issues by Release.
November 2, 2020	20.10.0 (Cloud Native Visualization) released. See 20.x Past Controller Releases.

Release Notes

On this page:

Related pages:

- Product Announcements and Alerts Home
- Past Releases

AppDynamics provides release information for Controller and agent releases, resolved issues, system requirements, and all current product updates.

Resolved and Known Issues

These are the resolved issues for the agents and the Controller (SaaS) in the 21.6 release.

When artifacts are updated, they are listed with their new version numbers in the tables below. Version numbers are also shown in the AppDynamics Downloads Portal.

You can sort the table of resolved issues by key, version, and product, or use the search field to find resolved issues. The most recent releases appear at the top of the page.

Agent Resolved Issues

There are no agent resolved issues for 21.6 release yet.

Controller (SaaS) Resolved Issues

There are no Controller resolved issues for 21.6 release yet.

Known Issues

There are no known issues in the 21.6 release as of yet.

Agent Enhancements

Filter table:

Agent	Summary	Version/Date
Analytics Agent	There is no release for 21.6.0 yet.	
Android Agent	There is no release for 21.6.0 yet.	
Apache Web Server Agent	There is no release for 21.6.0 yet.	
AppDynamics AWS Lambda Extension for Serverless APM	There is no release for 21.6.0 yet.	
C/C++ SDK	There is no release for 21.6.0 yet.	
Cluster Agent	There is no release for 21.6.0 yet.	
Cordova Plugin	There is no release for 21.6.0 yet.	
Database Agent	There is no release for 21.6.0 yet.	
Go SDK	There is no release for 21.6.0 yet.	
IBM Integration Bus Agent	There is no release for 21.6.0 yet.	
iOS Agent	There is no release for 21.6.0 yet.	

Java Agent	There is no release for 21.6.0 yet.	
JavaScript Agent	There is no release for 21.6.0 yet.	
Java Serverless Tracer	There is no release for 21.6.0 yet.	
Machine Agent	There is no release for 21.6.0 yet.	
.NET Agent	There is no release for 21.6.0 yet.	
Network Agent	There is no release for 21.6.0 yet.	
Node.js Agent	There is no release for 21.6.0 yet.	
Node.js Serverless Tracer	There is no release for 21.6.0 yet.	
PHP Agent	There is no release for 21.6.0 yet.	
Python Agent	There is no release for 21.6.0 yet.	
Python Serverless Tracer	There is no release for 21.6.0 yet.	
React Native Agent	There is no release for 21.6.0 yet.	
SAP	There is no release for 21.6.0 yet.	
Synthetic Hosted Agent	There is no release for 21.6.0 yet.	
Xamarin Agent	There is no release for 21.6.0 yet.	

Controller Enhancements

There are no new features for 21.6.0 release yet.

System Requirements and Supported Environments

See the relevant page for the component you want to install or upgrade.

Platform Requirements	Agent Requirements

• Controller System Requirements
• Enterprise Console Requirements
• Platform Requirements
• Events Service Requirements
• EUM Server Requirements
	• App Agents:
	  • Java Supported Environments
	  • .NET Supported Environments
	  • .NET Core Microservices Agent Support
	  • .NET Core for Linux SDK Supported Environments
	  • Node.js Supported Environments
	  • PHP Supported Environments
	  • Python Supported Environments
	  • Supported Apache Web Servers
	  • C/C++ Agent Supported Platforms
	  • Go SDK
	• Analytics Agent Environments
	• Browser RUM Supported Environments
	• Mobile RUM Supported Environments
	• Database Visibility Supported Environments
	• Machine Agent Requirements and Supported Environments
	• Network Visibility Supported Environments

Calendar Versioning

This page describes the versioning format used for product releases.

We use a calendar-based version format with a syntax of *YY.M.X* to reflect the date of release. For example, a release in October 2020 was 20.10.0, November was 20.11.0, and so on. If another version releases in the same month, the last number of the version increments by one, for example 20.11.1.

Controller and Agent Releases

The versioning of AppDynamics language agents is done independently of each other and the Controller. Starting in release 4.5.0, agents are backward-compatible with any Controller 4.4.1 and later, so you can download and install the latest version of the agent without having to keep track of your agent versions. See Agent and Controller Compatibility.

Maintenance Support for Software Versions

AppDynamics maintains and supports every software version for one calendar year after the next major software release is generally available. For example, support and maintenance of the 20.3 release will continue for 12 months after the 20.4 release.

See Product Announcements and Alerts.

For specific maintenance and support terms, refer to your license agreement(s).

Controller GA End of Maintenance and Support

This table lists generally available AppDynamics versions, release dates, and end of maintenance and support dates:

Controller Version	Release Date	End of Maintenance and Support
21.5	May 17, 2021	To be determined
21.4	April 5, 2021	May 17, 2022
21.2	February 22, 2021	April 5, 2022
20.11	November 16, 2020	February 22, 2022
20.10	October 5, 2020	November 16, 2021
20.8	August 24, 2020	October 5, 2021
20.7	July 16, 2020	August 24, 2021
20.6	June 2, 2020	July 16, 2021
20.4	April 21, 2020	June 2, 2021
20.3	March 9, 2020	April 21, 2021
4.5	July 11, 2018	March 9, 2021
4.4	November 1, 2017	July 11, 2019
4.3	April 5, 2017	November 1, 2018

AppDynamics Artifacts End of Maintenance and Support

Artifacts are generally available AppDynamics agent and platform components on the AppDynamics Downloads Portal or AppDynamics Accounts Downloads.

Artifact end of maintenance and support dates are one calendar year after the next generally available artifact release. For example, the .NET Agent 20.11 has an end of maintenance and support date of December 18, 2021 because 20.12 released on December 18, 2020.

Hotfixes are not considered major releases, and therefore do not affect the end of maintenance and support dates. Certain AppDynamics platform components have diverged from calendar versioning, such as the Events Service.

These components operate on semantic calendar versioning and will continue to release in the 4.5.x version convention. For example, the Events Service version 4.5.2.20640 will be maintained and supported for 12 months after the next major release.

Agent and Controller Compatibility

This page describes the compatibility mapping between Agent and Controller versions.

Agent and Controller Compatibility Mapping

Please refer to the Language Agent Backward Compatibility table for AppDynamics Agents and AppDynamics Controller compatibility.

As long as all agents are compatible with the Controller version, a monitored environment may have different agent versions deployed in it at a given time. However, the oldest agents should be on originating tiers of any business transactions. This ensures that agents on downstream nodes can process the correlation header created by the originating tier.

When rolling out agent upgrades, be sure to start upgrading the agents on the nodes of downstream tiers first, and then upgrade the agents on the originating tier nodes last.

Language Agent Backward Compatibility

As of release 4.5, AppDynamics language agents are backward-compatible with any Controller >= 4.4.1. This enables you to upgrade your language agents and take advantage of the latest agent-side enhancements, features, and bug fixes, without having to upgrade your 4.4 Controller. This table outlines the Controller and language agent releases that are compatible.

Starting in release 20.2, the AppDynamics language agents are backward-compatible with any Controller >= 4.5.0, including Controllers using Calendar Versioning >= 20.3.

> The Database Agent >= 20.7 is compatible only with the Controller >= 4.5.2.

Controller Release	Language Agent Release	Notes
>= 20.3	>= 4.4 >= 20.2 >= 20.3	Controllers >= 20.3 can accept connections from Agents >= 4.4, as well as Agents >= 20.2 and Agents >= 20.3 that use Calendar Versioning. As of 20.3, Controllers cannot accept connections from 4.1 Agents.
4.5.x	>= 4.1 >= 20.2 >= 20.3	Controller 4.5.x can accept connections Agents >= 4.1, including agents that use Calendar Versioning.
>= 4.4.1	>= 4.1	Controllers >= 4.4.1 can accept connections from newer agents.
4.3.x	4.1.x – 4.3.x	Controllers 4.3.x cannot accept connections from newer agents.

> The above table is only a reference for general software compatibility. For specific maintenance and support terms, refer to your specific EULA.

Backward compatibility is supported on these agents since the 4.5 release:

- C/C++ SDK
- Cluster
- Go SDK
- Java
- IBM Integration Bus (IIB)
- Machine
- .NET
- Node.js
- PHP
- Python

> While using the Enterprise Console to upgrade the Controller, several folders (including the `appagent` folder) are removed and reinstalled. If you deployed a Java Agent with a newer version than the Controller in the `controller/appserver/glassfish/domains/domain1/appagent` folder, the upgrade will fail and internal monitoring will be lost. To prevent this, you must update the `domain.xml` file to point to the new Java Agent or avoid using customized installation for Java Agents.

For notes on SSL protocol compatibility between versions of the agent and Controller, see SSL Compatibility Matrix for App Agent for Java - Controller.

Agent Versions Supported by Controller 20.x

Agent Version /Agent Type	Languages Agents								Cluster Agent	Database Agent	AppDynamics for Databases	Machine Agent	Network Agent	Analytics Agent
	Java Agent	.NET Agent	Node.js Agent	PHP Agent	Python Agent	Web Server Agent	Go SDK	C /C++ SDK						
>= 20.x	✓	✓	✓	✓	✓		✓	✓	✓	✓		✓	✓	✓
4.5.x	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓
4.4.x	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓	✓	✓

The ability to customize Business Transaction detection, supported by .NET Agent for Linux 4.5.9, requires Controller >= 4.5.2. The .NET Agent 4.5.9 works with Controller >= 4.4.1, but the customizable transaction detection and configuration capabilities require Controller >= 4.5.2. See .NET Agent for Linux Business Transaction Configuration.

SSL Compatibility between Java Agent and Controller

The default protocol used by the Controller is TLSv1.2. See Secure the Platform for information on changing the default security protocol used by the Controller or agent.

For the Java Agent, the default protocols are:

- For Java 8 SE applications: TLSv1.2.
- For Java >= 7 applications: TLSv1.2.

SSLv3 has been disabled for SaaS Controllers because of the CVE-2014-3566 vulnerability. SSLv3 connections are no longer supported.

Introduction of New Agent Features

Even when new and old agents are supported by the same up-to-date Controller, new functionality introduced by later agents is not available on the older agents.

Past Releases

See the following for the release notes of past versions:

- Past Resolved and Known Issues by Release
- Past Agent Releases
- Past Controller Releases
- Past On-premises Platform Releases

Past Resolved and Known Issues by Release

This page lists the resolved issues for the agents, the Controller (on-premises or SaaS), and the on-premises platform since the 20.2 release.

When artifacts are updated, they are listed with their new version numbers in the tables below. Version numbers are also shown in the AppDynamics Downloads Portal.

You can sort the table of resolved issues by key, version, and product, or use the search field to find resolved issues. The most recent releases appear at the top of the page.

21.5 Resolved and Known Issues

Agent Resolved Issues

Filter table:

Key	Product	Summary
CDM-7188	Android Agent	**Apply code changes** UI function is not working in Android studio when the `appdynamicsGeneratedBuildId_` field is added or removed
CLUSTERMON-2663	Cluster Agent	Even though the auto-instrumentation fails for an application, the environment is updated with some changes
DOTNET-4246	.NET Agent	"Console" as a logging configuration directs logs to STDERR instead of to STDOUT
DOTNET-4793	.NET Agent	HTTP request body removed for certain cases when ASP.NET Legacy instrumentation is enabled in .NET Agent for Alpine Linux
DOTNET-5344	.NET Agent	By default, the Unique Host ID should be set to the Container ID
DOTNET-5377	.NET Agent	There is a `System.TypeLoadException` exception on POCOs with nullable arguments
DOTNET-5490	.NET Agent	The Linux Agent does not correctly instrument generic value types
DOTNET-5551	.NET Agent	When the `site-regex` attribute is set to `true` for an IIS site configuration in the `config.xml` file, the full Windows agent now applies the cc to all site names matching the provided regular expression, instead of to just one site name.
JAVA-9254	Java Agent	Instrumentation point gets disabled for Netty exits
JAVA-8827	Java Agent	Agent does not sync with Controller appropriately for the purged application diagnostic ids
NODEJS-501	Node.js Agent	Agent does not show the correct number of nodes when starting the process manager.
PHP-1396	PHP Agent	One of the third-party libraries for the PHP Agent is upgraded
PHP-1357	PHP Agent	The `ADRUM_BT` cookie is sent across only for secure connections (HTTPS)

Controller (SaaS/On-Premises) Resolved Issues

Filter table:

Key	Product	Summary
ANALYTICS-13291	Analytics	Selecting multiple source to create Business Journey Maps prevents the retrieving of data for selectors
ANLYTCS_ES-4881	Event Service	Resolve the discrepancy between Metric Processor parsing of control characters and Elasticsearch
ANLYTCS_ES-5121	Event Service	Elasticsearch analyzer does not match the current Metric Processor Lucene analyzer
DBMON-7665	Database Visibility	The health rule violation details are not displayed when you click the health status in "Databases">"Dashboard" tab
DBMON-7879	Database Visibility	For some database, the "Object Browser">"Error log" screen does not render the log data
DBMON-7925	Database Visibility	JMX connection not working for Cassandra cluster with SSL enabled
DBMON-7931	Database Visibility	Database Agent naming convention prevents collectors from reporting data

DBMON-8032	Database Visibility	Add Controller flag to control metric filtering for configurations such as Cassandra's keyspace metrics and Couchbase's bucket m
DBMON-8037	Database Visibility	`DBMonEventBus` POST event caused thread pool saturation
DIAGPLAT-1130	APM	Snapshots filter for data collectors when collector type is "Any" is not working
DIAGPLAT-1151	APM	Missing end-to-end latency time in the "Business Transaction" view
METADATA-9610	APM	Corrected caching issue to ensure the setting of the Strict-Transport-Security header

Known Issues

There are no known issues in the 21.5 release.

21.4 Resolved and Known Issues

Agent Resolved Issues

Filter table:

Key	Product	Summary	Version
BRUM-6565	JavaScript Agent	Error in Visually Complete Time calculation when element is detached from DOM in IE11	21.4.0
BRUM-6613	JavaScript Agent	Calling the `ADRUM` command `addUserData` adds an entry even when the key/value pair is null or undefined	21.4.0
DBMON-7950	Database Agent	The `pg_database_size` metric for Postgres system statistics was computed every minute causing high CPU utilization. With this release, the `pg_database_size` value is computed at an interval of 1hr approximately	21.4.0
DBMON-8026	Database Agent	The Pluggable Database monitoring (PDB) does not work with the Database Agent 21.2	21.4.0
DOTNET-5406	.NET Agent	.NET Agent correctly reports all HTTP errors	21.4.0
DOTNET-5555	.NET Agent	When restarting the application using the MicroServices Agent, the node reuse environment variables now use the correct values	21.4.0
JAVA-9448	Java Agent	After upgrading Java Agent from 4.5.9 to 20.x or 21.x, the servlet business transactions are not displayed. The node property `osb-enable-webservice-entry` value has to be set to false (true by default)	21.4.0
JAVA-9433	Java Agent	An upgrade required for the agent such that the legacy agent uses `commons-io-2.5` and non-legacy agent uses `commons-io-2.8` library for `singularity-commons` dependency	21.4.0
JAVA-9361	Java Agent	When you use action suppression for disabling agent reporting, the suppression is not performed for the complete configured duration	21.4.0
JAVA-9261	Java Agent	Update the agent to allow splitting of business transaction on Spring Web Service entry by creating a custom configuration	21.4.0
JAVA-9235	Java Agent	The name of a node is generated multiple times for ephemeral nodes	21.4.0
JAVA-9110	Java Agent	Snapshot reporting stops after a few minutes post the agent-reset	21.4.0
LIBAGENT-442	Node.js Agent	Node.js Agent does not report HTTP data collector to Analytics	21.4.0
NODEJS-245	Node.js Agent	Node.js Agent fails HTTP tunneling for secure connections	21.4.0
WEBSRV-398	Apache Agent	`HttpClient` library for the Apache Agent is upgraded	21.4.0

Controller (SaaS/On-Premises) Resolved Issues

Filter table:

Key	Product	Summary
ALERT-8667	Alert & Respond	**Affects** column not available in the **Health Rule** widget
ALERT-86678	Alert & Respond	Behavior of the Controller's **Email Template Editor** is confusing
ANALYTICS-13434	Analytics	Optimize the Analytics Agent status call by using AnalyticsAgentConfigurationCache
BRUM-6548	Browser RUM	Null Point Error preventing the Controller UI from loading the dashboard data
DBMON-7397	Database Visibility	Remove incorrect max heap size information from the **Getting Started Wizard - Databases**
DBMON-7533	Database Visibility	HTTP thread pool saturation preventing access to the Controller UI
DBMON-7843	Database Visibility	No auditing for database collectors (update/delete/create)
DBMON-8033	Database Visibility	Add Controller flag to control metric filtering for configurations such as Cassandra's keyspace metrics and Couchbase's metrics
DBMON-8038	Database Visibility	`DBMonEventBus` POST event caused thread pool saturation
DIAGPLAT-1280	APM	Service Endpoints returns an HTTP 500 status and emits the error message **Error handling snapshot data**
L4A-17233	License Management	Usage views not displaying proper data granularity
METADATA-9564	Dashboard	**Export Custom Dashboard** should check for null widgets and not include them in the export
METADATA-9614	Dashboard	Corrected caching issue to ensure the setting of the Strict-Transport-Security header
PLATSE-132	Platform Engineering	Display the correct node information in the flow map
SVCMON-1253	APM	Null Pointer Error is preventing the deletion of business transactions and applications
SVCMON-1273	APM	**Service Endpoints** page displays **No Data Available** message or blank screen after scrolling
SVCMON-1277	APM	Improved the loading of **Custom Dashboards** by optimizing business transaction queries
SVCMON-1278	APM	New **Service Endpoints** table does not fit on the screen size
SYNTH-7058	Synthetic Browser Monitoring	Synthetic Agent is not displaying in the **Job Editor**

On-Premises Platform Resolved Issues

Filter table:

Key	Product	Summary	Version
ECONSOLE-6426	Enterprise Console	The upgrade process after the glassfish password change is fixed	21.4.0
ECONSOLE-6436	Enterprise Console	The HA Module final replication process is fixed	21.4.0
ECONSOLE-6438	Enterprise Console	The Database data directory downtime is fixed	21.4.0
ECONSOLE-6440	Enterprise Console	The HA Module now shows rsync errors for the Database replication	21.4.0
SYNTH-7154	Synthetic Server	Synthetic jobs get deleted when there are connection issues between the Synthetic Server and EUM database	21.4.0

Known Issues

There are no known issues in the 21.4 release.

21.3 Resolved and Known Issues

Agent Resolved Issues

Filter table:

Key	Product	Summary	Version
			Controller (SaaS/On-

DOTNET-4992	.NET Agent	Fixed process hang when capturing the thread snapshots under some conditions	21.3.1
DOTNET-5382	.NET Agent	(For Full Windows .NET Agent) You can now use the `APPDYNAMICS_CONTROLLER_HOST_NAME` , `APPDYNAMICS_CONTROLLER_PORT`, and `APPDYNAMICS_CONTROLLER_SSL_ENABLED` configuration environment variables to override properties in the `config.json` file for "controller host", "controller port", and "enable controller ssl connection", respectively.	21.3.1
DOTNET-5417	.NET Agent	Sensitive URL filter has been fixed for stalled transactions in .NET Core MicroServices Agent	21.3.0
DOTNET-5453	.NET Agent	.NET Agent for full Windows now reports correct Average CPU Time	21.3.0
DOTNET-5455	.NET Agent	Able to switch to different DMM modes from KPI for .NET application through Netviz Agent on Windows	21.3.0
DOTNET-5456	.NET Agent	Can start pcap for .NET application through Netviz Agent on Windows	21.3.0
DOTNET-5492	.NET Agent	Memory leak due to NetViz has been fixed	21.3.0
JAVA-9297	Java Agent	Issue with the commons-io library that is packaged with the Java Agent	21.3.0
DOTNET-5527	.NET Agent	TypeCache implementation is now safe for threading	21.3.1
DOTNET-5534	.NET Agent	Fix for proper handling of empty application configuration in .NET Agent configuration	21.3.1
DOTNET-5536	.NET Agent	Optimization for CPU usage when NetViz is not enabled	21.3.1
JAVA-8997	Java Agent	Servlet Split rule based on the selected parameter in the request does not work for the Reactor Netty framework	21.3.0
JAVA-8748	Java Agent	Duplicate backends are created with Java Agent API	21.3.0
LIBAGENT-442	Node.js Agent	Node.js Agent does not report HTTP data collector to Analytics	21.3.0
LIBAGENT-494	C/C++ SDK	Frame calls are abruptly failing	21.3.0
NETWORK-6264	Network Agent	The Network Agent Npcap version is upgraded to 1.20	21.3.0
NETWORK-6269	Network Agent	The Network Agent installation of Visual C++ is fixed	21.3.0
SRVLESS-1537	Node.js Serverless Tracer	Updated appdynamics-lambda-trace package dependencies	21.3.0

Premises) Resolved Issues

There was no Controller release for 21.3.

On-Premises Platform Resolved Issues

There was no on-premises platform release for 21.3.

21.2 Resolved and Known Issues

Agent Resolved Issues

Filter table:

Key	Product	Summary	Version
ANALYTICS-13226	Java Agent	Enabling recommended data collection intermittently affects Transaction Analytics reporting	21.2.0
BRUM-6539	JavaScript Agent	`ReferenceError`: `hideCookieBanner` is not defined when page is injected with `adrum-latest.js`	21.2.0
BRUM-6552	JavaScript Agent	Extra value at the end of the app URL set by the ADRUM cookie	21.2.0
BRUM-6553	JavaScript Agent	JavaScript statements inside `setTimeout` and `setInterval` run in local scope when adrum is enabled	21.2.0

CDM-7032	React Native Agent	Networking request responses are corrupted	21.2.0
CDM-7033	iOS Agent	iOS Agent disables SwiftUI previews	21.2.0
CDM-7175	Android Agent	Events are stored on `EventBus` even when agent is not initialized	21.2.0
DBMON-7552	Database Agent	The Database Agent sends the health rule data of a MongoDB replica node even after the node is stopped	21.2.0
DBMON-7568	Database Agent	The Database Agent log file includes the error, invalid USERENV parameter, for an Oracle collector	21.2.0
DBMON-7739	Database Agent	The custom metrics value drops to zero intermittently	21.2.0
DBMON-7654	Database Agent	The Oracle collector log displays an error when Sysmetrics query is used	21.2.0
DBMON-7745	Database Agent	The **Explain another query:** field under **Execution Plan** displays an error without any details for MariaDB	21.2.0
DOTNET-4538	.NET Agent	Optimizations for Async tracking in .NET Core and .NET Framework applications	21.2.0
DOTNET-4834	.NET Agent	Fix for high thread contention in certain Async-heavy applications	21.2.0
DOTNET-5175	.NET Agent	Fix for analytics data loss in certain cases when the Analytics Agent is recycled and loses communication with the .NET Agent	21.2.0
JAVA-9093	Java Agent	The **User Experience** value based on the set threshold limit is incorrect only for those records that do not have a matching transaction snapshot on the APM	21.2.0
JAVA-9037	Java Agent	Upgrading the Java Agent to 20.11 displays an exception `Error instantiating DispatcherServletInterceptor` in the logs	21.2.0
LIBAGENT-490	Node.js Agent	Unregistered backend metrics appear in the **Metric Browser**	21.2.0
METADATA-9624	APM	Ensure that the HTTP header `Strict-Transport-Security` is set	21.2.5
NETWORK-6227	Network Agent	Network Agent does not support host metrics on Windows	21.2.0
NODEJS-324	Node.js Agent	HTTP Request Data Collectors collect **HTTP Parameters Name** instead of **Display Name**	21.2.0
PYTHON-498	Python Agent	Java proxy caches stale agent data	21.2.0
PHP-1170	PHP Agent	PHP Agent does not support using an environment variable for defining a unique host ID	21.2.0

Controller (SaaS/On-Premises) Resolved Issues

Filter table:

Key	Product	Summary
ALERT-7111	Alert & Respond	Garbled characters appear in email template instead of cyrillic string
ALERT-7786	Alert & Respond	Health rules not available in cache after Controller upgrade
AMCSEV-1260	App Monitoring Cloud Services	Tiers Page does not show node details for the nodes after the 10th registered node
ANALYTICS-13205	Analytics	From the **Dashboard**, double-clicking "business journey" transactions directs user to **Overview** page instead of **Details** dialog
ANALYTICS-13219	Analytics	When changing the date format in **My Preferences** to "DD/MM/YY" and then opening a transaction snapshot in Analytics the er **has illegal timeRange** is displayed
ANLYTCS_ES-3474	Events Service	Some logically incorrect ADQL is validated
BRUM-6410	Browser RUM	Review of health rules violations from **User Experience** applications redirects to health rules for **Application Performance**
CLUSTERMON-2060	Cluster Monitoring	Agent registration is prevented while storing Cluster Agent properties in the database
DBMON-7131	Database Visibility	An Incorrect Database Agent version is displayed in the **Database Agents** setting after upgrading the agent and starting the ne the same name

DBMON-7222	Database Visibility	When configuring **Monitoring Operating System** for an AWS RDS collector, the required region is not displayed in the list. Now, the required regions are included and some unnecessary regions are removed from the list.
DBMON-7302	Database Visibility	**Disk Usage** within the database **Queries** displays an incorrect error message for the PostgreSQL database
DBMON-7308	Database Visibility	The **Live view** session list displays all the active sessions when the **View** filter is set to **All Sessions**, but active sessions are n when the filter is set to **All Active**
DBMON-7354	Database Visibility	The Invalid DB error is displayed when saving an action suppression on one of the nodes of the database cluster
DBMON-7373	Database Visibility	A blank window is displayed when you click **Drill Down** for a database on the **Transactions** snapshot
DBMON-7517	Database Visibility	Inconsistent **Custom Metric** permissions
DBMON-7535	Database Visibility	The **Exclude Schemas** option gets checked after upgrading from old Controller to new Controller
DBMON-7549	Database Visibility	Duplicate IO Metric entries are displayed for Microsoft SQL Server on the **Health Rule** configuration page
DBMON-7604	Database Visibility	Custom Metric not displayed in **Custom Metrics** tab even though data is displayed in **Metric Browser**
DBMON-8034	Database Visibility	Added the `dbmon.config.metrics.filtering.enabled` Controller flag to control the Database metric filtering for Cassar Keyspace and Couchbase's bucket-level metrics
DBMON-8039	Database Visibility	`DBMonEventBus` post-event call results in thread pool saturation
DIAGPLAT-1080	APM	Searching for snapshots yields inaccurate results for **Standard Time Range** from **Transaction Snapshot**
DIAGPLAT-1229	APM	Correct the data population for the "Snapshots Summary View"
LIC-648	Licensing	Database Monitoring license usage reporting does not work with Agent-based license rules
L4A-17796	Accounts	Increase the container-app version in the Controller
L4A-17838	Accounts	Multiple entries with same key
L4A-17959	Accounts	SAML and LDAP users without email address are not able to edit or add roles
PLATSE-120	Platform Maintenance	Reset counter when restarting Developer mode
SVCMON-1303	APM	NPE in BtServiceImpl#registerDeleteMdsBtEntityTransactionListener does not allow purge of applications and Business Transa
UISVCS-1063	Accessibility	The color of blue links in data grids, such as the **Business Transactions** list, has been updated to #0040B6 for improved contr
USERIMPACT-714	Experience Journey Map	Bad Link when clicking **View All Applications** in Experience Journey Map

On-Premises Platform Resolved Issues

Filter table:

Key	Product	Summary	Version
ANLYTCS_ES-5166	Enterprise Console	Update the Enterprise Controller to use Events Service 20.9	21.2.4
ECONSOLE-6395	Enterprise Console	The Controller monitoring setting in the AppServer configuration is fixed	21.2.0
ECONSOLE-6518	Enterprise Console	Correct the Enterprise Console's discovery for the latest version of the Events Service	21.2.7

Known Issues

Synthetic Services (Version 21.2.0)

Change in Synthetic job failure events behaviour (in the 20.7.0. version) affects the alerting configurations. This change triggers alerts for wrong events.

In this release:

- The events behaviour is reverted to the standard behaviour in Chrome, IE, and Firefox browsers. The issue still persists in Mobile Browser Emulation.
- Non-alphanumeric characters are not displayed in script logs of the Synthetic sessions.

21.1 Resolved and Known Issues

Agent Resolved Issues

Filter table:

Key	Product	Summary	Version
DOTNET-5316	.NET Agent	Sensitive Message Filter has been fixed when including parameters message-pattern FOUND but redaction-regex NOT MATCHED	21.1.0
DOTNET-5372	.NET Agent	Improvements to Azure Service Bus async error tracking	21.1.0
JAVA-7603	Java Agent	An error is reported when the maximum analytics collector limit exceeds	21.1.0
JAVA-8137	Java Agent	A Null Pointer Exception occurs if no regex pattern is provided in the custom backend naming config	21.1.0
JAVA-8255	Java Agent	Agent reset through the controller does not reset Dynamics Service (ADS) runtime state	21.1.0
JAVA-8380	Java Agent	EUM does not work appropriately with request match rules in non-servlet environments	21.1.0
JAVA-8407	Java Agent	Snapshots are not generated due to SQL parsing, even though the snapshots are reported to the controller	21.1.0
JAVA-8595	Java Agent	The transformation limit is reached due to the Analytics data collection interceptor	21.1.0
JAVA-8878	Java Agent	Business transaction data is not displayed if the business transaction lockdown is enabled in a service-proxy enabled environment	21.1.0
NODEJS-378	Node.js Agent	Unhandled `TypeError` exception terminates the Node.js Agent	21.1.0
PHP-1210	PHP Agent	PHP Agent does not report metrics after the Business Transaction lockdown in the Controller	21.1.0
PYTHON-640	Python Agent	Applied changes from JAVA-8878 to the Python Java proxy	21.1.0
SERVER-8458	Machine Agent	Server Agent no longer purges machines inadvertently before the expected purge time	21.1.0

Controller (SaaS/On-Premises) Resolved Issues

Filter table:

Key	Product	Summary	
ALERT-7111	Alert & Respond	Garbled characters are appearing in email template instead of a Cyrillic string	2
ALERT-7682	Alert & Respond	Action suppression time selector value is reset once you navigate away from the tab	2(1.
AMCSEV-575	App Monitoring Cloud Services	Not able to create custom **Service Endpoint** for JMS type with property type **Integer**	2
AMCSEV-927	App Monitoring Cloud Services	Long thread names are difficult to read	2
AMCSEV-1075	App Monitoring Cloud Services	Popup on flow map does not show the calls per minute/response as per grouped JDBC	2
AMCSEV-1099	App Monitoring Cloud Services	Filter UI For **Tiers & Nodes** in application need to close on **Cancel** or **Apply**	2
AMCSEV-1187	App Monitoring Cloud Services	Machine Agents screen displays the error `unnecessary keys are empty error`	2
ANALYTICS-12060	Analytics	Automatic field extraction does not permit adding a definer sample for more than one field	2
ANALYTICS-12912	Analytics	Missing start tier causes TypeError in Business Transaction flow map with EUM and BT node	2
ANALYTICS-12846	Analytics	Better handling of fields to display from originating tier in Analytics data table	2
ANALYTICS-12938	Analytics	Funnel Dashboard UI: Access to the specified resource forbidden	2
ANALYTICS-12953	Analytics	When editing the **Analytics Search** widget on the **Dashboard**, the toggle **Use Dashboard Time Range** is not responding	2
ANALYTICS-13003	Analytics	From **Analytics Search**, the array of Booleans and numbers display incorrect values	2
ANALYTICS-13031	Analytics	From **Analytics Search**, the **Tier** filter is incorrectly initialized by the **Node** filter	2

ANALYTICS-13070	Analytics	From **Transaction Snapshots**, the Analytics timestamp is incorrect if the customer changed **Display Time Zone** in **Preferences**	2
ANALYTICS-13090	Analytics	NodeJS does not receive Analytics HTTP config	2
BRUM-6390	Browser RUM	From **Timing Breakdown**, the SSL/TLS bar is not rendering correctly	2
CDM-7016	Mobile RUM	Accessing warning snapshots from **Code Issues** causes a java.lang.exception	2
DIAGPLAT-1001	APM	Snapshots are sometimes not displaying the potential issue information	2
DIAGPLAT-1043	APM	Query executed in `getSummariesAndAppendBlobsIfNeeded` times out	2
DIAGPLAT-1081	APM	Searching for snapshots yields inaccurate results for **Standard Time Range** from **Transaction Snapshot**	2
L4A-16565	Accounts	After migrating to Infrastructured-based Licensing, the Premium package had the wrong license consumption	2
METADATA-9467	APM	The security header response (HSTS) is fixed	2
METADATA-9483	APM	Health rule status precedence is fixed	2
REPORTS-799	Reports	Scheduled reports are sent on clicking Save	2
REPORTS-811	Reports	Reports (scheduled or one time) with a custom time range are delivered with the Login page	2
TMNT-952	APM	Customer time range does not display time as per the timezone configured in the preferences	2
UISVCS-1011	UI Services	Creation of error detection configuration field for the HTTP return codes for the PHP Agent	2

Filter table:

Key	Product	Summary	Version
ECONSOLE-6162	Enterprise Console	The Enterprise Console now selectively calls the create-xxx commands and sets the alias equal to s1as (the default for the Controller)	21.1.0
ECONSOLE-6245	Enterprise Console	`check-ec-dir-owner.orcha` now runs a task on all inventory hosts	21.1.0
ECONSOLE-6303	Enterprise Console	The lsof package check on controller.sh has been updated with a non-pipe (|) implementation	21.1.0
ECONSOLE-6304	Enterprise Console	Function to check privileged ports in install-init.sh has been fixed	21.1.0
ECONSOLE-6344	Enterprise Console	EC Installer copies mysql 5.5 binaries to null in 4.5.17+ versions	4.5.17, 20.3.0, 20.4.0, 20.7.0 20.6.10, 21.1.0
ECONSOLE-6368	Enterprise Console	The Controller diagnosis job has been fixed and no longer fails in a Windows implementation	21.1.0
ECONSOLE-6370	Enterprise Console	Installation of Enterprise Console fails on some time zone settings	21.1.0
EUMPLAT-1280	EUM Server	HMAC key is not available for Synthetic Hosted Agent license	21.1.1
EUMPLAT-1320	EUM Server	Unified license terms not properly provisioned on Controller with license file	21.1.0

Known Issues

There are no known issues in the 21.1 release.

20.12 Resolved and Known Issues

Agent Resolved Issues

Filter table:

Key	Product	Summary	Version
ANALYTICS-13154	Java Agent	`toString()` method displays `["[Ljava.lang.String;@23377943"]` error for Java arrays in **Data Collectors**	20.12.0
BRUM-6403	JavaScript Agent	Page renders slowly for AngularJS apps if there is no `onreadystatechange` handler for AJAX calls	20.12.0
BRUM-6438	JavaScript Agent	ADRUM cookie triggers `sameSite` attribute warning in Firefox	20.12.0
CLUSTERMON-2259	Cluster Agent	The `cluster-agent-operator-openshift.yaml` file fails to deploy the pod on OpenShift and the log displays the `no matches for kind "Clustercollector" in version` error	20.12.1
CLUSTERMON-2088	Cluster Agent	The auto-instrumentation does not work when there is a space in the `appname` configuration within the `cluster-agent.yaml` file	20.12.0
DBMON-7391	Database Agent	The Network IO metrics value displays an unreasonable number for the AIX servers	20.12.0
DBMON-7561	Database Agent	When configuring **Monitoring Operating System** for an AWS RDS collector, the **eu-north-1** region is not displayed	20.12.0
DBMON-7495	Database Agent	When the IP address of the host machine keeps changing, the database metrics drop to zero and the **Events** page displays that the collectors are reinitialized	20.12.0
DBMON-7444	Database Agent	When a database uses the Database Partitioning Feature (DPF), the Calls per Minute metric on the Metric Browser displays incorrect data	20.12.0
DBMON-7428	Database Agent	The hardware monitoring stops working and the log displays the `com.jcraft.jsch.JSchException: session is down` error	20.12.0
DOTNET-5047	.NET Agent	Ability to download latest .NET Agent from the Controller	20.12.0
DOTNET-5108	.NET Agent	Fixed issue of incorrect CPU% reporting in certain cases	20.12.0
DOTNET-5111	.NET Agent	Download link for .NET has been updated on the Getting Started Wizard page	20.12.0
DOTNET-5270	.NET Agent	Ability to add headers to ASP.NET Core HTTP responses is now available	20.12.0
DOTNET-5300	.NET Agent	Agent MSI MA registration request with vCPU information was extended to support IBL licensing	20.12.0
DOTNET-5340	.NET Agent	Unhandled exception in timer thread may cause application crash in some situations	20.12.0
NODEJS-199	Node.js Agent	Standalone 64-bit Node.js Agent versions 10 and 12 fail in Linux with a `Segmentation Fault` error	20.12.0
NODEJS-284	Node.js Agent	Node.js Agent errors in Alpine-based Docker images	20.12.0
PHP-999	PHP Agent	PHP Agent does not detect ports when a client uses `CURLOPT_PORT` instead of placing it in the URL	20.12.0
PHP-1187	PHP Agent	Analytics snapshot data is missing for continuing and terminating the PHP Tiers snapshots	20.12.0
PYTHON-518	Python Agent	Upstream application is not visible on Controller when the downstream application is Python	20.12.0
SERVER-8351	Machine Agent	Machine Agent works with user-defined `JAVA_HOME` with `-j` flag	20.12.0

Controller (SaaS/On-Premises) Resolved Issues

There are no Controller releases for 20.12 yet.

On-Premises Platform Resolved Issues

There are no on-premises platform releases for 20.12.

Known Issues

There are no known issues in the 20.12 release as of yet.

Filter table:

Key	Product	Summary	Version

Resolved Issues

BRUM-6350	JavaScript Agent	Visually Complete Time sometimes reports as a Page Complete Time	20.11.0
CDM-6991	Android Agent	Issue parsing the mobile agent token that registers the agent with the Controller, causing extra licenses to be consumed	20.11.0
CLUSTERMON-2089	Cluster Agent	Agent registration is prevented while storing Cluster Agent properties in the database	20.11.0
CLUSTERMON-2057	Cluster Agent	The terminologies blacklist and whitelist are replaced with blocklist and allowlist to follow neutral usage of words	20.11.0
DOTNET-4986	.NET Agent	Fix for race condition in tracking CLR instances for IIS in-processes hosted by .NET Core 3.1 applications	20.11.0
DOTNET-5159	.NET Agent	To prevent correlation headers from becoming too long, the Linux Agent now limits the number of in-process call chain segments for thread correlation to ten segments per node. To disable this limit, set the node property track-all-async = true.	20.11.0
DOTNET-5244	.NET Agent	Fix for application crash if more than one URL filter is configured	20.11.0
JAVA-8599	Java Agent	Upgrading the Java Agent to use Agentless Analytics results in custom data not getting reported to analytics	20.11.0
JAVA-8536	Java Agent	A significant number of threads are blocked in the Java Agent UUID generation	20.11.0
JAVA-8456	Java Agent	Issue with the snapshot generation post-JVM restarts	20.11.0
JAVA-8441	Java Agent	When JVM is restarted before the end of Action suppression rule created with disable agent reporting, the agent is not re-enabled automatically and the metrics are not reported	20.11.0
JAVA-8283	Java Agent	A `Null Pointer Exception` error is reported when retrieving JMX attribute	20.11.0
JAVA-7944	Java Agent	Issue with the Average Response Time and reporting of Thread Task metrics	20.11.0
JAVA-7603	Java Agent	An error is reported when the maximum analytics collector limit is exceeded	20.11.0
PYTHON-517	Python Agent	Python agent does not group all methods (GET, POST, PUT, and DELETE) when custom transaction detection rules are applied without HTTP method criteria	20.11.0
PYTHON-581	Python Agent	Individual Business Transactions are formed even in the presence of Custom Transaction Rule	20.11.0
SERVER-8419	Machine Agent	Machine Agent installed with RPM does not honor settings for user and user group	20.11.0
SERVER-8465	Machine Agent	JVM options provided via service installation is set in `vmoptions` file	20.11.0
SERVER-8471	Machine Agent	CPU usage reporting fix for Solaris 11	20.11.0
SERVER-8530	Machine Agent	Correct CPU used percentage reported for each processor group	20.11.0
SRVLESS-1305	Serverless Tracer	The Java, Node.js, and Python Serverless Tracers now default HTTP calls made by the tracer to timeout after `2000ms`	20.11.0

Filter table:

Key	Product	Summary
ALERT-6830	Alert & Respond	Incorrect link destination during remediation script action creation
ALERT-6990	Alert & Respond	`Only application object scope is supported for application` error displays when you suppress an action (one an EUM enabled APM application
ALERT-7088	Alert & Respond	Health Rule for the User Experience Application does not automatically include new pages
ALERT-7828	Alert & Respond	Health rules not available in cache after Controller upgrade
AMCSEV-727	App Monitoring Cloud Services	Automatic Leak detection session duration shows 60 minutes when the agent default is 30 minutes
AMCSEV-1023	App Monitoring Cloud Services	Unable to export the call graph
AMCSEV-1328	App Monitoring Cloud Services	Tiers page does not display node details after the 10th node is registered

ANALYTICS-12780	Application Analytics	ADQL basic search filter does not escape double quotation
ANALYTICS-13089	Application Analytics	Incorrect Browser Sessions/Mobile Sessions details in the **Analytics Home** page
ANALYTICS-13095	Application Analytics	Node.js Agent does not receive the Analytics HTTP configuration
ANALYTICS-13067	Application Analytics	From **Analytics Search**, the **Tier** filter is initialized incorrectly by the **Node** filter
CLUSTERMON-2089	Cluster Monitoring	Agent registration is prevented while storing Cluster Agent properties in the database
DBMON-7170	Database Visibility	The database health rule is not evaluated for the custom metric name that has more than two | (pipe) symbols
DBMON-7256	Database Visibility	Some Sybase custom metrics do not display when the SQL query returns multiple result sets
DBMON-7268	Database Visibility	You cannot update the Advanced Options for a collector because the collector is automatically updated when you select the **Ad Options** on the Collector configuration page
DBMON-7269	Database Visibility	The values cannot be added in **Advanced Options** on the **Edit Collector** page
DBMON-6620	Database Visibility	The Oracle collector cannot be created with a custom connection string and sub-collectors
DBMON-7615	Database Visibility	Custom Metric appears in the Metric Tree for all the configurations regardless of the configuration it belongs to in the Metric Bro
DBMON-7618	Database Visibility	Unable to create action suppression on single node for database with multiple nodes (topology)
DIAGPLAT-879	APM	Retrieve Event Data API is not filtered by the tier ID
DIAGPLAT-891	APM	Event creation locked because of a thread pool saturation
DIAGPLAT-907	APM	Filtering is incorrect for the Service Endpoints in getRequestSegmentsByFilterHandle API
DIAGPLAT-947	APM	Some TSS bound its detail string with the wrong ID
DIAGPLAT-950	APM	Providing JavaScript to purge bad TSS detail IDs from the cache
DIAGPLAT-1061	APM	Query executed in `getSummariesAndAppendBlobsIfNeeded` timed out
DIAGPLAT-1082	APM	Search for snapshots gives inaccurate results for **Standard Time Range** under **Transaction Snapshot**
JAVA-8865	APM	ServiceProxy feature for Controllers upgraded to 20.11.0 are disabled
L4A-14347	Accounts	Improper Authorization allows access to `admin.jsp`
METADATA-9402	APM	REST API has added load historical liveness data for a tier, its node, and its parent app based on a anchor metric ID provided b
METADATA-9485	APM	Ensure that the HTTP header `Strict-Transport-Security` is set
METADATA-9488	APM	Health rule status precedence is incorrect
MQS-1247	Metric Query Service	**Metric Browser** does not display data for some random window sizes
REPORTS-813	Reports	Reports with custom time ranges getting delivered with login page
SVCMON-1301	APM	Improve queries for business transactions to optimize performance
SYNTH-5864	Synthetic Browser Monitoring	Synth Credential vault shows EUM App name in the dropdown instead of the app name
TMNT-707	APM	When exporting metric data, the browser does not show data points after the Daylight Savings Time (DST) change

Filter table:

Key	Product	Summary	Version
ECONSOLE-5932	Enterprise Console	The `.properties` files are no longer included in the Enterprise Console Installer	20.11.0
ECONSOLE-6190	Enterprise Console	Enterprise Console successfully changes the `http-listener-2.port` after the `update_config` job completes	20.11.0
ECONSOLE-6196	Enterprise Console	The Controller DB user password has been updated with Aurora DB	20.11.0
ECONSOLE-6201	Enterprise Console	Customizations to the `domain.xml` file are no longer lost during upgrade to 20.4	20.11.0
ECONSOLE-6216	Enterprise Console	The Enterprise Console recognizes timeouts in Windows Stop Service from playbooks	20.11.0
ECONSOLE-6344	Enterprise Console	MySQL 5.5 and 5.6 binaries are fixed in the Enterprise Console packaging for Windows	20.11.0

Known Issues

There were no known issues in the 20.11 release.

20.10 Resolved and Known Issues

Agent Resolved Issues

Filter table:

Key	Product	Summary
BRUM-6375	JavaScript Agent	Invalid pointer error for JavaScript Agent in Internet Explorer/Edge is caused by `HTMLImageElement` URI length
BRUM-6230	JavaScript Agent	`updateFetchEventResponseHeaders` throws the error message "forEach is not a function.." intermittently
BRUM-6351	JavaScript Agent	A virtual page event is triggered when there is a call to push or replace states, even when there is no URL change
CDM-5817	iOS Agent	MAT is retained across calls when `config.collectorChannel` is set
DBMON-7057	Database Agent	The database events summary displays an error, `Maximum custom metric limit 40 is reached`, even when the number of custom met than 40
DBMON-7280	Database Agent	**Metric Browser** does not display the performance metrics for Oracle Pluggable Database (PDB)
DBMON-7423	Database Agent	The Topology tab displays incorrect port numbers for the non-default ports of Microsoft SQL and Oracle clusters
DBMON-7255	Database Agent	Oracle Database failover issue with the SCAN IP in a custom JDBC string
DBMON-6900	Database Agent	Even when an SQL query is updated to include a label for a column within a custom metric, the **Details** tab of the event displays the column nam the label
DOTNET-4860	.NET Agent	.NET Agent logging has been fixed and no longer shows extraneous Socket error messages
DOTNET-4990	.NET Agent	When you split a URI using a custom exit call naming rule with method parameters, the string length has been expanded to 100 characters
DOTNET-5082	.NET Agent	Fix for 20.4 .NET Agent for Windows causing high CPU in some scenarios
DOTNET-5141	.NET Agent	Added a new configuration option (set `APPDYNAMICS_AGENT_MASK_ENV_VARS environment variable = true`) to mask environment va the Linux agent before sending information to the Controller
DOTNET-5171	.NET Agent	Fix for .NET Agent for Windows where diagnostic messages from the agent were failing on malformed Regex rules
DOTNET-5205	.NET Agent	Fix for BT level error metrics not including HTTP errors in some situations
JAVA-8537	Java Agent	After upgrading the Java Agent version 4.5.9 to 20.8, the OSB Rules are not applied correctly
JAVA-8436	Java Agent	When Java Agent 20.9 is instrumented with JDK 15, the agent does not start and throws a Null Pointer Exception error
JAVA-8399	Java Agent	Snapshots are not generated, even though they are reported to the controller
JAVA-8009	Java Agent	The agent is unable to find the matching `AgentBackendDiscoveryConfig` as it is unable to extract the database names for the Oracle RAC c
JAVA-7898	Java Agent	The Java Agent cannot be dynamically attached to the Java 11 process
JAVA-7775	Java Agent	The JVM tab displays empty contents for the Java Agent installed on Docker and instrumented with JDK11
JAVA-7713	Java Agent	`java.lang.IllegalStateException` is reported due to incomplete instrumentation and a race condition seen under heavy load
NODEJS-10	Node.js Agent	`Mark Nodes as Historical` API causes C++ Rest SDK errors on shutdown

NODEJS-245	Node.js Agent	Remove `request` package from the Node.js Agent
NODEJS-261	Node.js Agent	Upgrade Java proxy to version 20.7.0.30393
NODEJS-285	Node.js Agent	Remove `console.log(process.env);` from Node.js Installer script
PHP-1044	PHP Agent	PHP Agent does not capture Postgres queries when an explicit connection handle is passed to the query functions
PHP-1040	PHP Agent	The PHP Agent crashes due to a null reference, when accessing a call graph node on the request shutdown
PHP-1001	PHP Agent	The PHP Agent crashes applications during the call graph collection
PHP-987	PHP Agent	There is no way to filter expected error codes resulting from an outgoing service call using `drupal_http_request`
PYTHON-324	Python Agent	Python Agent collects all the errors in one bucket when custom error codes are configured
SERVER-8435	Machine Agent	Resolved high CPU usage caused while executing `WmiPrvSE.exe` process commands
SERVER-8466	Machine Agent	Reports correct CPU usage date for AIX systems
SERVER-8469	Machine Agent	English Locale set for AIX collector to parse output correctly
WEBSRV-298	Apache Agent	Issue with the backend naming for a large number of virtual hosts.

Controller (SaaS/On-Premises) Resolved Issues

Filter table:

Key	Product	Summary
ALERT-6669	Alert and Respond	Drill down for health rule violation and affected entities display blank results
ALERT-6767	Alert and Respond	`Application with id:-1 does not exist` error is displayed when you access the health rules using the dash click on **Critical** or **Warning** tab
ALERT-6781	Alert and Respond	Support **Overall Application Performance** type health rules through **Alerting Templates** for EUM-enabled APM app
ALERT-7379	Alert and Respond	Unable to save action suppression for EUM-enabled applications
AMCSEV-743	App Monitoring Cloud Services	Node count on the Dashboard does not match the node count on the **Node** panel
AMCSEV-768	App Monitoring Cloud Services	Custom transaction match rules cannot be created from **Snapshot** screen
AMCSEV-808	App Monitoring Cloud Services	Java Agent version not displayed correctly on **Tier & Nodes**
AMCSEV-862	App Monitoring Cloud Services	Backend discovery rules losing data upon enabling or disabling
AMCSEV-885	App Monitoring Cloud Services	Java Agent version not displayed correctly on **Tier & Nodes**. Backport to 20.8
AMCSEV-1170	App Monitoring Cloud Services	Enable **Automated Transaction Diagnostics** in the Controller by default
ANALYTICS-11516	Application Analytics	UI not setting start and end timestamps correctly in ADQL aggregation query
ANALYTICS-12809	Application Analytics	Selected options do not attach the correct options in **Add Business Journeys**
ANALYTICS-12926	Application Analytics	Limit error messages and stack traces in Business Transactions to 8191 characters
ANALYTICS-13094	Application Analytics	Node.js Agent does not receive the Analytics HTTP configuration
BRUM-6243	Browser RUM	Creating EUM Application using **Getting Started Wizard** fails with error
BRUM-6223	Browser RUM	EURT column on Pages and AJAX request UI is too slim and column sizing does not persist
CDM-6544	Mobile RUM	Correct `Fragment Start Event` type typo in the Collector
CDM-6571	Mobile RUM	Create unit test cases for IP Header and Geo resolution handling for mobile beacons
CLUSTERMON-1948	Cluster Monitoring	The **Pods** tab does not display data

CLUSTERMON-2090	Cluster Monitoring	Cluster Agent runs out of database space when persisting the agent registration request
DBMON-6275	Database Visibility	When you edit the query text of a custom metric, the changes get saved without clicking **Save**
DBMON-6816	Database Visibility	The Metric Browser displays a `504` Error when you launch it for the first time
DBMON-6937	Database Visibility	The **Custom Metrics** tab displays a `500 internal` error
DBMON-6947	Database Visibility	The database health rules are not evaluated for the collectors with the view permissions
DBMON-7055	Database Visibility	The memory metrics are not displayed for the Oracle 11g database
DBMON-7060	Database Visibility	The Transaction Snapshot is not displayed for the Oracle Cluster
DBMON-7600	Database Visibility	Only add custom metrics to the configuration which it belongs to in the **Metric Browser**
DBMON-7617	Database Visibility	Unable to create action suppression on single node for database with multiple nodes (topology)
DIAGPLAT-767	APM	No RSD found for filter error in `/snapshot/getRequestSegmentDatav2`
DIAGPLAT-925	APM	Filtering incorrect for the Service Endpoints in the `getRequestSegmentsByFilterHandle` API
DIAGPLAT-961	APM	Some top summary stats (TSS) are bound to the wrong ID
DIAGPLAT-994	APM	Added a filter to the REST endpoint `/restui/events/query` to curtail unexpected results
DIAGPLAT-1018	APM	Incorrect join for archive queries when getting potential problems for snapshots
DIAGPLAT-1060	APM	Query executed in `getSummariesAndAppendBlobsIfNeeded` timed out
DIAGPLAT-1083	APM	Search for snapshots gives inaccurate results for **Standard Time Range** under **Transaction Snapshot**
L4A-8473	Accounts	When passing multiple groups, the `SAML Access Attribute` does not work
L4A-13281	Accounts	Account key is appearing in the Controller audit report
METADATA-9355	APM	MDS connection pools no longer ignore JDBC parameters in domain.xml
METADATA-9372	APM	MDS connection pools no longer ignore JDBC parameters in domain.xml
METADATA-9486	APM	Ensure that the HTTP header `Strict-Transport-Security` is set
METADATA-9489	APM	Health rule status precedence is incorrect
MOBILE-1619	Mobile App	Fetch push server URL from the Configuration Manager in each notification request
METADATA-5428	Dashboards	Dashboard health status widget appears green even when the health rule shows grey
REPORTS-799, REPORTS-814	Reports	Scheduled reports are sent on clicking **Save**
REPORTS-812	Reports	Reports with a custom time range are delivered with the login page
SVCMON-963	APM	Exit calls to resolved backends now display on **Flow Maps**
SVCMON-1252	APM	Query updating a flow map preference causes high CPU usage

On-Premises Platform Resolved Issues

Filter table:

Key	Product	Summary
ECONSOLE-6221	Enterprise Console	Enterprise Console now honors timeouts in Windows Stop Service during upgrade
ECONSOLE-6343	Enterprise Console	Upgrading the Enterprise Console from 20.8 to 20.10 no longer hinders the JRE upgrade on Controller nodes during an orcha upg
EUMPLAT-1104	EUM Server	Upgrade MySQL to 5.7.31
EUMPLAT-1231	EUM Server	Threads blocked in JSON parsing in the `gson` library

Known Issues

There are no known issues in the 20.10 release.

20.9 Resolved and Known Issues

Agent Resolved Issues

Filter table:

Key	Product	Summary
BRUM-6321	JavaScript Agent	`promise.then()` throws an error when receiving non-function parameters
CLUSTERMON-1910	Cluster Agent	Cluster Agent is unable to monitor the namespaces that include the `config` string
CLUSTERMON-1869	Cluster Agent	The auto-instrumentation of Java applications fail when using the 20.6 version of the Redhat Java Agent image
DBMON-7160	Database Agent	The **Disk IO** hardware metric is not displayed for a Linux machine
DBMON-7231	Database Agent	The monitoring of Oracle database is affected when the `v$osstat` permission is missing
DBMON-7232	Database Agent	Issues with Oracle topology resolution when using a specific LDAP connection string or a specific `local_listener` value
DBMON-7127	Database Agent	The **CPU** metric value is displayed incorrectly when the instance caging is enabled in Oracle
DOTNET-4397	.NET Agent	Business Transaction name now registers as the `ControllerName.ActionName`
DOTNET-4942	.NET Agent	Business Transaction correlation for EUM has been fixed so that the .NET Agent appends the Business Transaction ID cookie
DOTNET-4954	.NET Agent	Windows service does not stall after a few days of instrumentation
DOTNET-4973	.NET Agent	The .NET Linux Agent no longer crashes with a `seqfault` error. To avoid potential crashes, you can disable snapshots by entering the following code: `"profiler": { "eventMask": 2149842980 }`
IIB-316	IBM Integration Bus Agent (IIB)	Business Transactions are missing for some of the message flows on the **AppDynamics Business Transaction** view
JAVA-8110	Java Agent	Java Agent is unable to register an application and send metrics because an `InstantiationException` with `NettyMarkerInter` observed on the console or agent logs when the Servlet monitoring is disabled
JAVA-8052	Java Agent	A `Snapshot call graph contains CPU times - This is a Hotspot call graph` message is displayed on some of the snapshots
JAVA-8047	Java Agent	A large number of AppDynamics threads are found on the applications using Vert.x
JAVA-7947	Java Agent	A `ClassCastException` is observed when the Live Preview session is started with the Custom Servlet rule
JAVA-7601	Java Agent	Ignored errors or exceptions are still counted towards Service Endpoint errors per minute
LIBAGENT-46	C/C++ SDK	**Thread Name** shows `-1` for C/C++ SDK segments in the **Waterfall View** tab
LIBAGENT-53	C/C++ SDK	Cross-application correlation displays incorrectly in the primary **Application Flow Map** dashboard
LIBAGENT-54	C/C++ SDK	C/C++ SDK Agent on Windows fails to connect and logs an `Open failed` error
PHP-988	PHP Agent	PHP Drupal application crashes when handling an outgoing HTTP request
PHP-985	PHP Agent	PHP packages fail the Aqua security scan
PHP-974	PHP Agent	PHP Agent crashes PHP applications on Drupal7 HTTP interception
PYTHON-493	Python Agent	Jackson jars dependency removed from Python Agent
PYTHON-473	Python Agent	Python Agent reports misleading error format
SERVER-8437	Machine Agent	Machine Agent now extracts numerical addresses for network interfaces without symbolic host, port or user names
SERVER-8443	Machine Agent	Machine Agent for AIX no longer gets stuck on remote volumes command and reports only local volumes metrics

SERVER-8446	Machine Agent	Removed `perfstat_partition_config()` call due to AIX bug crashing the Machine Agent
WEBSRV-256	Apache Agent	Agent crashes the worker process when the Backend ID is `NULL`

Controller (SaaS/On-Premises) Resolved Issues

Filter table:

Key	Product	Summary	Version
ALERT-6835	Alert and Respond	Alerting template popup status does not change as per the template progress status	uic-20.9.0
SYNTH-6536	Synthetic Browser Monitoring	Cookies not visible in request headers of Chrome v83	20.9.0

20.8 Resolved and Known Issues

Agent Resolved Issues

Filter table:

Key	Product	Summary
BRUM-6191	JavaScript Agent	`adrum.js` beacon response fails due to multiple Access-Control-Allow-Origins headers being set
BRUM-6216	JavaScript Agent	Console error cannot read property url of undefined on a fetch call
BRUM-6207	JavaScript Agent	`adrum.js` beacon contains more events than the graphql events call
CDM-6144	iOS Agent	Inconsistent crash report time stamp in Controller
DOTNET-4089	.NET Agent	Applications no longer generate invalid token exceptions
DOTNET-4579	NET Agent	Access violation in Redis backend resolution attempt to retrieve host name
DOTNET-4599	NET Agent	Two lines of the console output that appeared when the Windows agent started up no longer appear. To restore them, create and set an `appdynamics.outputMode` environment variable.
DOTNET-4659	.NET Agent	Reporting service from the Microsoft Power BI Tools no longer fails
DOTNET-4707	.NET Agent	Analytics transaction metrics are not starting unless Micro Services APM Agent is restarted
DOTNET-4811	.NET Agent	OWIN cookies no longer lost after agent upgrade
DOTNET-4898	.NET Agent	IPC timeout configurability between coordinator and .NET Agent
JAVA-7625	Java Agent	A null message passed causes a Null Pointer Exception in the `ALoggerMessageInterceptor`
JAVA-7569	Java Agent	Sensitive message filter does not work appropriately when the message-pattern is `FOUND` but redaction-regex is `NOT MATCHED`
JAVA-7939	Java Agent	Vertx exit interceptor fails to create exit call due to a bad URI
JAVA-7950	Java Agent	In certain scenarios, `ServerTransactions`, which are started before all required classes get instrumented, remain in the agent memor they get aborted
JAVA-8007	Java Agent	After a tier is deleted, JMX rules are not re-registered and therefore the JMX metrics are not displayed on the Metric Browser
JAVA-8013	Java Agent	The auto-discovered spring web service names display multiple duplicate names with different `ns` numbers when auto-detection is enabl
NETWORK-6171	Network Agent	Network Agent now reports data in low throughput environments
PHP-984	PHP Agent	A large number of errors are displayed in the agent logs due to `No URL filter config found in filterUrl`
PYTHON-244	Python Agent	MongoDB Transaction snapshots expose sensitive data
SERVER-8361	Machine Agent	AIX CPU metrics calculation now uses proper timestamp
SERVER-8367	Machine Agent	The Controller has been enhanced to not allow registration of misconfigured container type Server Visibility or Docker Visibility Agents (r

		mode)
SERVER-8420	Machine Agent	Add default heap JVM options in `InstallService.vbs` for Windows MA installer
SERVER-8450	Machine Agent	Default heap JVM options now default to -Xms256m and -Xmx256m in the `InstallService.vbs` script for the Windows Machine Age

Controller (SaaS/On-Premises) Resolved Issues

Filter table:

Key	Product	Summary
ALERT-6676	Alert and Respond	The third-party Bzip2 library used by the Controller has been updated to version 1.0.8
ALERT-6761	Alert and Respond	Upon clicking the **View Dashboard During Health Rule Violation** for a BT, you are redirected to a non-existing entit
ALERT-6860	Alert and Respond	When you access a health rule using the health status widget on the dashboard, `Application with id:-1 does` displayed
ALERT-6916	Alert and Respond	Upgrade to Kafka Streams 2.3.1
ALERT-6996	Alert and Respond	Actions are not getting triggered after **Recurring Action Suppression** time has expired
AMCSEV-62	App Monitoring Cloud Services	Unable to save an exception match condition
AMCSEV-90	App Monitoring Cloud Services	**Create Method Invocation Data Collector** dialog from call graph does not autofill
AMCSEV-519	App Monitoring Cloud Services	Unable to save **Java Error Detection Configuration** because of an `OptimisticLockException`
AMCSEV-543	App Monitoring Cloud Services	Unable to delete archived machine snapshots
AMCSEV-607	App Monitoring Cloud Services	Error counts for the URLs in the **Error Occurrence Details** table are not getting refreshed for different time ranges
AMCSEV-755	App Monitoring Cloud Services	Failed to create a backend detection `Custom Exit Point` via the right-right-click
AMCSEV-885	App Monitoring Cloud Services	Java Agent version is not displayed correctly with the Controller version on **Tier & Nodes**
ANALYTICS-12711	Application Analytics	Overloaded method checkbox becomes unchecked in **Create Data Collector** dialog box
ANALYTICS-12730	Application Analytics	Analytics widget size is setting wrong when add to absolute layout dashboard
ANALYTICS-12900	Application Analytics	Analytics Home page displays inconsistent EUM charts
ANALYTICS-12912	Application Analytics	Missing start tier causes `TypeError` in Business Transaction flow map with EUM and BT node
ANALYTICS-12935	Application Analytics	When you add a Business Journey and select a Business Transaction, the BT name is replaced with the BT ID upon s Journey
ANALYTICS-13024	Application Analytics	The Business Transactions are not filtered based on the tier selected in the **Business Journey** page
ANALYTICS-13093	Application Analytics	Node.js Agent does not receive the Analytics HTTP configuration
ANLYTCS_ES-3766	Events Service	The third-party netty library used by the Controller has been updated to 4.1.48
ANLYTCS_ES-3804	Events Service	User is unable to see data in the Metric Browser because of the error message `last run failed`
ANLYTCS_ES-3861	Events Service	The `third-party search-guard-ssl` library used by the Event Service is updated to version 24.0 and kibana pl version 5.6.8.7
APMPLAT-11061	APM	App and Machine Agent Status now correctly display percentages for Nodes and Tiers in the user interface
BRUM-6161	Browser RUM	Analyze records can fail to load
CLUSTERMON-1870	Cluster Monitoring	Collect Cluster Agent adoption and usage data
COGENG-901	Cognition Engine	The third-party `netty` library used by the Controller has been updated to 4.1.48
DBMON-6596	Database Monitoring	The Oracle Collector does not work when the **Oracle Wallet** option is enabled during the Oracle Collector configuratio
DBMON-6764	Database Monitoring	The **Hardware Monitoring** option includes an outdated region for **Amazon RDS**
DBMON-7005	Database Monitoring	The third-party `netty` library used by the Controller has been updated to 4.1.48
DBMON-7111	Database Monitoring	The Controller does not display any data when you open the Metric Browser for the first time
DBMON-7140	Database Monitoring	Upgrade the JAR `singularity-xml` to include Apache Xerces 2.12.0

DIAGPLAT-798	APM	Snapshots do not get loaded, `500 Internal Server Error` is displayed
DIAGPLAT-828	APM	Call to `/snapshot/getRequestSegmentDatav2` returns the error `No RSD found for filter`
DIAGPLAT-893	APM	Correct the exit call mapper RSD v2
DIAGPLAT-916	APM	When reviewing a Business Transaction Snapshot, the display of **Potential Issues** takes time of 30 to 60 seconds
DIAGPLAT-926	APM	Service Endpoints do not filter corresponding transaction snapshots when you use the `getRequestSegmentsByFil`
DIAGPLAT-942	APM	Correct the Service Endpoints filter for snapshots at the front-end layer
DIAGPLAT-960	APM	Modify query for loading the TSS detail string to cache
DIAGPLAT-963	APM	Some TSS bound its detail string with the wrong ID
JAVA-8021	Java Agent - Controller	Upgrade the Java Agent to 20.6.0-30555 in the Controller
L4A-12908	Accounts	Reset password via email does not work for single tenant
L4A-14101	Accounts	Upgrade `platform-rbac` to 20.4.0-170
L4A-14454	Accounts	Upgrade the dropwizard library to 20.4.0-170
L4A-14543	Authentication	LDAP group sync fails due to deleted groups being included in results
METADATA-9008	APM	Socket timeout and number of connections in Jedis pool are configurable
METADATA-9384	APM	MDS connection pools no longer ignore JDBC parameters in `domain.xml`
METADATA-9397	APM	Upgrade the JAR `singularity-xml` to include Apache Xerces 2.12.0
METADATA-9397	APM	The context is not reset post an exception in some dashboard flows
METADATA-9453	APM	REST API loads historical liveness data for a tier, its node, and its parent app based on an anchor metric ID provided
METADATA-9487	APM	Ensure that the HTTP header Strict-`Transport-Security` is set
METRICSVC-5021	Metric Service	App and Machine Agent Status now correctly display percentages for Nodes and Tiers in the user interface
METRICSVC-5758	Metric Service	Upgrade the PostreSQL JDBC Driver library to 42.2.14
MQS-1139	Metric Query Service	App and Machine Agent Status now correctly display percentages for Nodes and Tiers in the user interface
NETWORK-5403	Network Visibility	User with **Administration, Agents, Getting Started Wizard** permission is not allowed to view the **Network Visibility**
REPORTS-791	Reports	Controller changes made all reports fail if any had scheduling fails on startup
REPORTS-795	Reports	NodeJs used by controller has been updated to v14.8
REPORTS-777	Reports	Upgrade the version of Node.js in the Controller to 8.17.0
SERVER-8174	Machine Agent	App and Machine Agent Status now correctly display percentages for Nodes and Tiers in the user interface
SVCMON-827	APM	In Controller UI, **Web Service Exit Call** showing as `ui:ms_TransactionExitPointType_null`
SVCMON-864	APM	Controller logs full of the Redis error `Cache unavailable issue`
SYNTH-5965	Synthetic Browser Monitoring	`Problem Ended` event should be generated only after Error/Warning is confirmed
TMNT-720	APM	Time changes randomly in the **Custom Time** range component

On-Premises Platform Resolved Issues

Filter table:

Key	Product	Summary	Version
ECONSOLE-862	Enterprise Console	User who installed Enterprise Console is the only user permitted to upgrade the Enterprise Console	20.8.0
ECONSOLE-6083	Enterprise Console	Windows Controller upgrade fails	20.8.0

Known Issues

There were no known issues in the 20.8 release.

20.7 Resolved and Known Issues

Agent Resolved Issues

Filter table:

Key	Product	Summary
CDM-6426	Android Agent	Screenshot tiles are not properly sent to the collector and as a result, screenshots may not be rendered correctly in the c
CLUSTERMON-1758	Cluster Agent	The Cluster Agent Pod does not start and remains in the **Pending** state when taint is defined in the nodepool This release adds support to set tolerations. See *tolerations* in the Configure the Cluster Agent YAML File section.
DBMON-7036	Database Agent	The literals are unmasked for a CREATE statement even after selecting **Remove literals from the queries** in the **Secu** ation
DOTNET-4728	.NET Agent	Unable to see complete stacktrace after upgrading to MSI-based 20.5 agent from .NET Microservice agent
IIB-253	IBM Integration Bus (IIB) Agent	IIB Agent log files grow continuously for un-instrumented execution groups
IIB-280	IBM Integration Bus (IIB) Agent	Nodes are not getting discovered by the IIB Agent on ACE (IIB 11)
IIB-294	IBM Integration Bus (IIB) Agent	Shutdown of the broker that runs multiple execution groups might result in a crash on some execution groups
IIB-287	IBM Integration Bus (IIB) Agent	The Broker on AIX crashes during startup when instrumented with IIB Agent version 20.7.0
JAVA-7825	Java Agent	Java Agent stops reporting analytics data on the agent version 20.4
JAVA-7820	Java Agent	Running Corretto with Java Agent results in an `Unrecognized Vendor` message
JAVA-7800	Java Agent	End User Monitoring (EUM) correlation with ADRUM headers does not work for Akka HTTP
JAVA-7569	Java Agent	The slowest database and remote calls are not displayed for any time slice (current or historical) for one hour or less
JAVA-7504	Java Agent	The `ApacheLog4` library in appserver agent shipped with controller installer is upgraded
JAVA-7234	Java Agent	The Servlet filter API is displayed as a POJO type in the call graph in place of Servlet entry point
NODEJS-39	Node.js Agent	Agent detects proxy server as HTTP Backend and displays it in the **Flow Map**
PHP-947	PHP Agent	PHP Agent reports incorrect HTTP response code when exceptions are thrown
PYTHON-323	Python Agent	Python Agent affects the request header of Application API when using `pyagent`
PYTHON-20	Python Agent	`def get_proxysupport_dir()` error is displayed when instrumenting ODOO application using the Python Agent
PYTHON-228	Python Agent	CVE-2019-17571 vulnerability found in the Python AppDynamics proxy
SERVER-8322	Machine Agent	Windows: OSHI process CPU utilization metric displays correctly when compared to WMI-based metrics
SERVER-8348	Machine Agent	CPU User metric now shown in KPI mode
SYNTH-5620	Synthetic Agent	Resource errors are not excluded when authentication protocol determines synthetic job session status
SYNTH-5313	Synthetic Agent	In certain cases of network failure in Chrome, the Synthetic Agent records retried network requests as two distinct pages session
SYNTH-6495	Synthetic Agent	Synthetic job results in the error `Invalid SSL/TLS certificate: INET_E_INVALID_CERTIFICATE' in IE11` for Azure Agents
WEBSRV-241	Apache Agent	The `install.sh` script unnecessarily creates an entry in the `/etc/ld.so.conf.d` directory

Controller (SaaS/On-Premises) Resolved Issues

Filter table:

Key	Product	Summary
ALERT-3800	Alert and Respond	The audit.log and REST API response include a typo, `helathRuleIds`
ALERT-4781	Alert and Respond	When you update an existing action, `OBJECT_UPDATED` events in the audit log file show old values as `0` instead of the previously conf value
ALERT-5940	Alert and Respond	When you update a global health rule checkbox for an application, the application name is missing in the audit log
ALERT-6106	Alert and Respond	Action suppression does not work for health rules with status `Ended – Critical`; this happens for events of the type `Database`
ALERT-6139	Alert and Respond	Alerting template does not get applied to applications, it gets stuck with a status `In Queue`

ALERT-5349	Alert and Respond	When you open an Event UI and reload the background UI, the refreshed background UI display is distorted
ALERT-6028	Alert and Respond	Remove Alerting Template RBAC constraints
ALERT-6684	Alert and Respond	The third-party Bzip2 library used by the Controller has been updated to version 1.0.8
ALERT-6764	Alert and Respond	Deep link for **Affected Entity** and **View Dashboard During Health Rule Violation** is taking -1 instead of the correct entity ID
ALERT-6857	Alert and Respond	Accessing the **Health Status** widget returns the `Application with id:-1 does not exist` error
ALERT-6915	Alert and Respond	Upgrade to Kafka Streams 2.3.1
AMCSEV-49	App Monitoring Cloud Services	Missing information when exporting JMX Rule
AMCSEV-191	App Monitoring Cloud Services	Customization of **Automatic Leak Detection** settings transforms `APP_AGENT` configuration
AMCSEV-470	App Monitoring Cloud Services	500 Internal Server Error occurs while loading **Application Dashboard**
AMCSEV-512	App Monitoring Cloud Services	**User Getter Chain** not working correctly when `\.` operator is found
AMCSEV-518	App Monitoring Cloud Services	Deleting backend throws the error message `Invalid unresolved backend call id`
AMCSEV-521	App Monitoring Cloud Services	Snapshots link broken in **Transaction Summary Statistics** UI with RSD V2 enabled
ANALYTICS-11038	Applications Analytics	ADQL with substring function does not work for fields with hyphen
ANALYTICS-11634	Applications Analytics	MIDC configuration incorrectly stores method chains from APM configurations
ANALYTICS-12263	Applications Analytics	BO definition in draft state with duplicate names will loop forever in UI
ANALYTICS-12441	Applications Analytics	Analytics: UI Fails To Handle Long Queries
ANALYTICS-12571	Applications Analytics	Page Name filter in Add Business Journey dialog does not display by App Key and does not use ADQL
ANALYTICS-12675	Applications Analytics	BO grid view API call should include an additional param `flowmapDetails` set to false
ANALYTICS-12878	Applications Analytics	Selected options do not attach the correct options in **Add Business Journeys**
ANLYTCS_ES-3415	Events Service	Cannot create Analytics metrics from an ADQL query that contains a comma
ANLYTCS_ES-3803	Events Service	Unable to see metric data on the Metric Browser, last run failed message is displayed
ANLYTCS_ES-3860	Events Service	The third-party `search-guard-ssl` library used by the Event Service is updated to v24.0 and kibana plugin is updated to v5.6.8.7
ANLYTCS_ES-3938	Events Service	Upgrade the Controller HTTP Client, HTTP Core, and AsyncHTTP Client library versions
APMPLAT-11934	APM	JMX `create metric` operation to auto-fill domain name in UI popup and JMX rule using the `create metric` operation have been
APMPLAT-13274	APM	This event warning has been updated: "Automatic Service Proxy Discovery functionality has minimum agent version requirements for upstream of discovered proxies. Please check the documentation for details."
CLUSTERMON-1358	Cluster Monitoring	The **Pods** tab does not display the pod list after the re-registration of Cluster Agent
DBMON-6529	Database Visibility	The `Connect backends to Database Visibility to track health status` message is displayed every time, even wher required
DBMON-6628	Database Visibility	The **Network Dashboard** flowmap does not display the backend nodes of the database
DBMON-6601	Database Visibility	The **Events** tab for the MySQL collector displays the error, `Error processing InnoDB metrics for config`
DBMON-7141	Database Visibility	The third-party Apache xerces library used by the Database Agent is updated to 2.12.0
DIAGPLAT-807	APM	Snapshots do not get loaded, 500 Internal Server Error is displayed
DIAGPLAT-809	APM	`ArrayIndexOutOfBoundsException` is displayed for RSD V2 enabled Controllers

DIAGPLAT-815	APM	Allow a list of GUIDs in `SnapshotListFilter` to view multiple GUIDs in the snapshot list view
DIAGPLAT-857	APM	No RSD found for filter error is displayed for the REST API call `/snapshot/getRequestSegmentDatav2?`
DIAGPLAT-892	APM	Correct the exit call mapper RSD v2
DIAGPLAT-915	APM	Correction of a query for the RSD v2
DIAGPLAT-927	APM	Service Endpoints do not filter corresponding transaction snapshots when you use the `getRequestSegmentsByFilterHandle` AP
DIAGPLAT-943	APM	Correct the Service Endpoints filter for snapshots at the front-end layer
EUMPLAT-593	EUM	Archived Browser Snapshot does not open when the linked Transaction Snapshot does not exist
EUMPLAT-831	EUM	Account credential is not updated when hitting `renew-license-key` endpoint
JAVA-8020	APM	The third-party Apache xerces library used by the Java Agent is updated to 2.12.0
L4A-12097	Accounts	No audit log entry created for license changes
L4A-12444	Accounts	`NullPointerException` when using REST API to generate an access token for API Client
L4A-12447	Accounts	Encountered LDAP sync failure while updating user outside Hibernate session
L4A-13981	Accounts	The third-party Jackson `databind` library used by `platform-rbac` is updated to 2.9.10.5
L4A-14453	Accounts	Upgrade the dropwizard to 20.7
L4A-14542	Accounts	LDAP group sync fails due to deleted groups being included in results
METADATA-7833	APM	Anchor metrics for business transactions are getting lost causing issues in entity liveness
METADATA-8834	Metric Browser	ADDs display correctly in the Metric Browser without a name conflict
METADATA-9035	APM	Able to use native query to find Business Transaction IDs and their tier IDs from the database correctly
METADATA-9171	APM	No snapshots are created because the `ADD` stacktrace purger takes more than 20 hours to complete and holds the read-only lock for t metadata table
METADATA-9381	APM	MDS connection pools ignore the JDBC parameters in `domain.xml` including `useSSL`, which impacts the the AppDynamics GovAPM deployment
METADATA-9393	APM	Upgrade the JAR `singularity-xml` to include Apache Xerces 2.12.0
METADATA-9430	APM	Context is not reset in some dashboard flows post an exception
METRICSVC-5472	Metric Service	Reader client version was updated to version 2.12.0.5959 for Controller primary
METRICSVC-5776	Metric Service	The third-party PostgreSQL JDBC Driver library used by the Controller has been updated to version 4.2.13
METRICSVC-5972	Metric Service	Client connection pool saturated
MQS-1100	Metric Query Service	Missing business transaction specific graphs in the **Expression Dashboard**
REPORTS-792	Reports	Scheduled reports not working after Controller upgrade if any invalid report definitions exist when `appserver` is restarted
REPORTS-796	Reports	NodeJS used by controller has been updated to v14.7.0
SERVER-8232	Server Visibility	Service availability check interval in the Controller UI now displays minutes instead of seconds
SERVER-8271	Server Visibility	Server metrics no longer disappear in the UI when scrolling
SVCMON-883	APM	`WebService` type exit call is displayed as `ui:ms_TransactionExitPointType_null`
SVCMON-900	Controller UI	ResourceManager renders package:key strings when a key is not found instead of the locale strings, with a log message `package o string not found`
TMNT-580	Machine Agent	Unable to download the Machine Agent from **Getting Started Wizard - Servers**

On-Premises Platform Resolved Issues

Filter table:

Key	Product	Summary
ECONSOLE-5645	Enterprise Console	Controller Upgrade button has been fixed and is now responsive after Enterprise Console upgrade
ECONSOLE-6070	Enterprise Console	Add secondary job now verifies JRE version and updates the JRE configuration on both primary and secondary Controllers
SYNTH-4759	Synthetic Server	On-premises Controller with hosted license is unable to create jobs due to SaaS API call failures
SYNTH-5678	Synthetic Server	Private synthetic agent jobs do not consume synthetic license units if webpages are not loaded
SYNTH-5685	Synthetic Agent	TEMP directory `C:\users\agent_user\AppData\Local\Temp` does not get deleted automatically
SYNTH-5636	Synthetic Server	Synthetic jobs are auto-disabled even when sufficient license units are left
SYNTH-5554	Synthetic Server	Synthetic job editor becomes unresponsive when it receives errors from AJAX calls
SYNTH-5458	Synthetic Server	Capacity-related error messages displayed for private agents are now different from the error messages displayed for SaaS agent
SYNTH-5445	Synthetic Server	Waterfall graph displays duplicate tick labels in the timeline on the **Session Summary** page
ALERT-4182	Alert and Respond	`NullPointerException` is displayed when configuring the Email SMS for an on-premises Controller

Known Issues

There were no known issues in the 20.7 release.

20.6 Resolved and Known Issues

Agent Resolved Issues

Filter table:

Key	Product	Summary
ANALYTICS-4256	Analytics Agent	Analytics Agent inside Machine Agent does not drop PID file
ANALYTICS-12717	Analytics Agent	Recommended Data Collectors field name appears as `null`
ANALYTICS-12739	Analytics Agent	Upgraded SCS Tool library
ANALYTICS-12740	Analytics Agent	Upgraded common-codec-1.14 library
DBMON-6873	Database Agent	The TOTAL_LOG_USED metrics value displayed on the Metric browser does not match with the value in Database server
DBMON-6074	Database Agent	The agent logs display `org.apache.http.NoHttpResponseException` when connecting to the Controller
DBMON-7037	Database Monitoring	The Database Agent fails to monitor Oracle version 11
DBMON-4943	Database Monitoring	**Execution Plan** is not displayed on the dashboard for MySQL Server 8
DBMON-6936	Database Monitoring	The hardware metrics are not displayed on the dashboard when enhanced monitoring is enabled on Microsoft SQL Server on A
DOTNET-3457	.NET Agent	Integer and Boolean type of SQL data gatherer for Analytics is not working
DOTNET-3640	.NET Agent	Unable to configure apps running on .NET Core due to incorrect assembly reference
DOTNET-4410	.NET Agent	IIS Ping failure/hang in presence of AppDynamics Agent
DOTNET-4513	.NET Agent	Sensitive URL filters not working on HTTP exits
DOTNET-4574	.NET Agent	Agent Coordinator service throwing unhandled exception
DOTNET-4678	.NET Agent	Controller application name in config.xml is case sensitive
DOTNET-4710	.NET Agent	Service End Point is not identifying in .NET Agent version 20.5
DOTNET-4717	.NET Agent	Exclude rules not working with .NET Agent 20.3.0 and later
DOTNET-4723	.NET Agent	AppDynamics coordinator service no longer fails with a system management exception
DOTNET-4745	.NET Agent	HTTP request collection fails for all transactions once a stalled transaction is identified

DOTNET-4711	.NET Agent	Error in CLR Version shown in Controller UI
NODEJS-91	Node.js Agent	Agent capturing disabled snapshots with node-level properties crashes
NODEJS-103	Node.js Agent	MongoDB queries fail
SERVER-6143	Machine Agent	No longer runs multiple `awk` commands resulting in system resource exhaustion
SERVER-8091	Machine Agent	Registered Machine Agents no longer report metrics after default account access keys are changed
SERVER-8207	Machine Agent	AIX Machine Agent should not have `JavaHardwareMonitor` enabled
SERVER-8272	Machine Agent	AIX collector was reviewed and produces well-formed JSON to prevent script errors
SERVER-8279	Machine Agent	Windows: OSHI reported processes now list multiple processes sharing the same name
SERVER-8318	Machine Agent	`java.net.UnknownHostException: kubernetes.default.svc` exception is fixed

Controller (SaaS/On-Premises) Resolved Issues

Filter table:

Key	Product	Summary
ALERT-5349	Alert & Respond	When you open an Event UI and re-load the background UI, the refreshed background UI display is distorted
ALERT-5731	Alert & Respond	Alert link in the email notification is broken and does not display the alert details, post Controller upgrade
ALERT-6220	Alert & Respond	When you apply an alerting template, it gets stuck in a queue and is not applied
ALERT-6355	Alert & Respond	Alerting Templates not showing in 20.6 Controllers
ALERT-6377	Alert & Respond	Editing or deleting custom roles for Alerting Templates causes the error `CONFIG_ALERTING_TEMPLATES_PLANS does exists`
ALERT-6518	Alert & Respond	Unable to add dashboards to the admin role, error `permission action is not specified` due to missing `CONFIG CHEDULE` permission
ALERT-6914	Alert & Respond	Upgrade to Kafka Streams 2.3.1
AMCSEV-979	App Monitoring Cloud Services	Customization of **Automatic Leak Detection** settings transforms `APP_AGENT` configuration
ANALYTICS-12073	Analytics Events Service (SaaS)	Inconsistency in counting business transaction segments for Analytics licensing
ANALYTICS-12775	Analytics	BizOutcome Grid View API call should include the parameter flowmapDetails set to false
ANALYTICS-12644	Analytics	Business Journeys are failing to load
ANLYTCS_ES-3536	Events Service	Update the Events Service version in Controller to get library updates for Apache Kafka, Google Guava, and `snakeyml`
ANLYTCS_ES-3859	Events Service	Remove the search-guard library from the Events Service
ANLYTCS_ES-3937	Events Service	Upgrade the Controller HTTP Client, HTTP Core, and AsyncHTTP Client library versions
BRUM-6022	Browser RUM	Grid columns appear when user sorts some EUM paginated grids
CLUSTERMON-1257	Cluster Monitoring	The **Succeeded** pod count includes the number of deleted pods on the **Pods** page
DBMON-6477	Database Monitoring	The queries that correlate with standalone collectors are not displayed when you drill-down **Business Transactions**
DBMON-6628	Database Monitoring	The **Network Dashboard** flowmap does not display the backend nodes of the database
DBMON-4676	Database Monitoring	Collectors display multiple nodes with the same name when creating a health rule
DBMON-6718	Database Monitoring	The `Connect backends to Database Visibility to track health status` message is displayed every tim when it is not required
DBMON-5508	Database Monitoring	Sorting does not work as expected for the **Queries, Time in database,** or **CPU Usage(%)** metrics in the **Databases** pag
DBMON-6948	Database Monitoring	Removed a singleton from an Enterprise JavaBean (EJB) to prevent a Database Monitoring thread pool saturation
DBMON-6983	Database Monitoring	After upgrading to 20.6, the Database Dashboard is missing the CPU and Network I/O graphs
DBMON-7142	Database Monitoring	Upgrade the Apache Xerces third-party library to 2.12.0

EUMPLAT-601	End User Monitoring	Update the Neustar map library to correct geodata
EUMPLAT-776	End User Monitoring	Metrics missing intermittently due to an SocketTimeoutException when EUM tries to download metrics
JAVA-8019	Controller - APM	Update the Java Agent in the Controller
L4A-14102	Accounts	Upgrade `platform-rbac` to 20.4.0-170
L4A-14452	Accounts	Upgrade dropwizard to 20.6
L4A-14541	Accounts	Customization of **Automatic Leak Detection** settings transforms `APP_AGENT` configuration
METADATA-9220	APM	No snapshots are created because the `ADD` stack trace purger takes more than 20 hours to complete and holds the read- the `metadata` table
METADATA-9399	APM	Upgrade the Apache Xerces third-party library in the Controller to 2.12.0
METRICSVC-5901	Metric Service	Client connection pool saturated
MQS-1131	Metric Query Service	**Server|Component:tearName** metrics not supported by new implementation of **Custom Dashboards**
MQS-1163	Metric Query Service	Missing graphs specific to business transactions in the **Expression Dashboard**
SERVER-6833	Server Visibility	Server page displays the total number of servers in the bottom right footer
SERVER-7785	Server Visibility	Searching using the Hierarchy field is now possible
SERVER-8250	Server Visibility	Upgraded Apache `httpclient` to 5.0
SERVER-8279	Server Visibility	The dashboard now displays all processes, even if they share the same name
WEBSRV-6	Apache Agent	Apache Agent injects ADRUM headers when Business Transaction correlation is disabled
WEBSRV-22	Apache Agent	Using `AppDynamicsLaunchProxy` directive with the apachectl reload (graceful) option does not delete a previous proxy
WEBSRV-228	Apache Agent	The Webserver Agent version is displayed in place of the proxy version on the Controller UI

On-Premises Platform Resolved Issues

Filter table:

Key	Product	Summary	V
ECONSOLE-5466	Enterprise Console	The `check-controller-health` CLI command from the Enterprise Console now reads results correctly	20.
ECONSOLE-5825	Enterprise Console	The `appserver-admin-console-password` is no longer used nor displays in setup.xml	20.
ECONSOLE-5832	Enterprise Console	Enterprise Console no longer allows an older Enterprise Console platform installer to be used to upgrade to a newer version	20.
ECONSOLE-5889	Enterprise Console	MySQL root permissions have been fixed to allow upgrade	20.
ECONSOLE-5906	Enterprise Console	Controller upgrade from 4.5.x to 20.3 no longer fails with `mysql_upgrade` version	20.
ECONSOLE-6183	Enterprise Console	Upgrade MySQL to 5.7.31	20.
EUMPLAT-903	EUM Server	Use jetty-util 9.4.18 to resolve the issue upgrading SSLContextFactory	20.

Known Issues

There were no known issues in the 20.6 release.

20.5 Resolved and Known Issues

Agent Resolved Issues

Filter table:

Key	Product	Summary
ANALYTI	Analytics	Health Check commands failed due to deprecated method

CS-12608	Agent	
CDM-5998	Android Agent	Server is returning null response headers for correlated requests
DBMON-6508	Database Monitoring	After upgrading the Database Agent, the database collectors fail to retrieve data. Hence, the summary of each event on the **Events** page displ `.sql.SQLException: Connection property: format error: Property is 'v$session.program` error
DBMON-6805	Database Monitoring	For Mongo database, Database Agent includes its own monitoring query
DLNATIVE-2544	Node.js Agent	Customer Transaction Naming based on HTTP headers and cookies is now enabled.
DOTNET-4608	.NET Agent	.NET Agent Nuget package for Azure App Service does not support .NET Core 2.2+
DOTNET-4465	.NET Agent	BT detection stopped working after upgrade to .NET core 3.1 version and disable MVC naming
DOTNET-3654	.NET Agent	MVC Controller/Action BT naming with Linux full agent is not working with endpoint-based routing
DOTNET-4063	.NET Agent	`A request container already exists on the request` error when implementing .NET Core Agent with oData
DOTNET-4286	.NET Agent	ASP_DOTNET Core Interceptors reporting sporadic stalled/abandoned transaction
DOTNET-4621	.NET Agent	Azure WebApp is crashing with agent version 20.4.1
DOTNET-4442	.NET Agent	AccessViolationException crashes applications with .NET Agent 4.5.18.1
IIB-256	IBM Integration Bus Agent	IIB Broker starts up with the instrumented Appdynamics Agent but crashes after sometime
JAVA-6431	Java Agent	The correlation between client and server is not reported for Spring Boot RMI
JAVA-6793	Java Agent	Applications using the SAP wily Agent along with Java Agent fail to start
JAVA-6876	Java Agent	Akka Http Exit call does not report correct `average response time`
JAVA-6993	Java Agent	Agent fails to collect logs through the Controller in the **Debug** level mode
JAVA-7005	Java Agent	A verification error results in the application failure to run the database calls
JAVA-7164	Java Agent	Java Agent throws the exception `java.lang.reflect.InaccessibleObjectException` with `jdk-11.0.6.jdk`
JAVA-7257	Java Agent	AppDynamics Opentracer library is incorrectly published to Maven
JAVA-7338	Java Agent	Java Agent reports insecure Deserialization Arbitrary CodeExecution due to htpclient-cache third party library vulnerability
JAVA-7368	Java Agent	ASM is updated to the latest version, 8.0
JAVA-7389	Java Agent	The following Apache commons packages have been upgraded: • `apache-fileupload` to 1.4 • `apache-commons-codec` and `httpclient` to 4.5.12
JAVA-7471	Java Agent	Running an application server with Java Agent 20.4+ throws a `NullPointerException` in the Java Agent logs
NODEJS-83	Node.js Agent	Node.js Agent fails when variables pass through the Express middleware
NODEJS-89	Node.js Agent	Node.js Agent does not capture the cookie value for HTTP Data Collector configuration if the specific cookie is not first in the cookie header.
PHP-171	PHP Agent	PHP Agent causes a CLI process to crash during batch processing
PHP-822	PHP Agent	SQL query does not get masked when a comment is included in the actual query
PHP-852	PHP Agent	Agent shows increased Response Time for the socket connection with JAVA proxy
PHP-863	PHP Agent	PHP RPM files do not have the relevant version numbers
PHP-891	PHP Agent	Wordpress and Drupal Frameworks are not detected in the downstream app when the downstream app receives a call from the upstream app `le_get_contents` function

SERVER-6824	Machine Agent	Machine Agent ignores IPv6 network interface if it is not available
SERVER-8182	Machine Agent	OSHI CPU utilization metric displays correctly when compared to WMI-based metrics
SERVER-8205	Machine Agent	The Windows collector script exits gracefully if it times out
SERVER-8273	Machine Agent	Support of runtime system properties for logging functionality
SERVER-8284	Machine Agent	Machine Agent for Linux no longer gets stuck on remote volumes command and continues to report metrics
WEBSRV-20	Apache Agent	Upgrade Java version for proxy to fulfil security audit requirements
WEBSRV-24	Apache Agent	Crash in the destruction of the exit call under heavy load
WEBSRV-63	Apache Agent	Apache aborts under high load and worker thread counts
WEBSRV-66	Apache Agent	Unix Socket Limitation in Apache Agent
WEBSRV-69	Apache Agent	Apache worker thread crashes within ZeroMQ calls

Cloud Monitoring Resolved Issues

Filter table:

Key	Product	Summary
CLUSTERMON-1576	Cluster Agent	Cluster Pods list is populating correctly for SaaS Controllers
SERVER-8264	Controller UI	When navigating to the cluster dashboard, the dashboard no longer displays a 500 Internal Server Error while fetching High Metric Reten level data for more than 1 hour
SERVER-8271	Machine Agent	Server metrics no longer disappear in the UI when scrolling

Controller (SaaS/On-Premises) Resolved Issues

There are no resolved issues for the Controller in this release.

Known Issues

Android Agent (Version 20.5.2)

Screenshot tiles are not properly sent to the collector and as a result, screenshots may not be rendered correctly in the collector.

20.4 Resolved and Known Issues

Agent Resolved Issues

Filter table:

Key	Product	Summary
CLUSTERMON-168	Cluster Agent	After Cluster Agent pod restarts, it does not preserve the configuration for monitored namespaces
CLUSTERMON-1164	Cluster Agent	ClusterAgent v4.5.16.780 on a K8s Cluster v1.11 crashes with `panic: runtime error: integer divide by zero`
CLUSTERMON-	Cluster Agent	Cluster Error Graph: Shows an incorrect number of Error Events

6628		
DBMON-6219	Database Monitoring	Database Agent cannot connect to Sybase Adaptive Server Enterprise 16.0 SP03
DBMON-6553	Database Monitoring	Database Agent does not capture details from Sybase 15.5 database.
DBMON-6303	Database Monitoring	The metric browser does not capture the buffer and the cache memory metrics for a Linux collector
DBMON-6460	Database Monitoring	Database Agent cannot resolve the different nodes of Oracle RAC by using LDAP authentication
DOTNET-4521	.NET Agent	Agent does not report Http Status codes in Asp.Net core in-process hosted app
DOTNET-4448	.NET Agent	Coordinator taking longer to start when set as `Earlystartservices`
DOTNET-4508	.NET Agent	.NET Core 3.1 based application is failing to instrument with .NET Agent version 20.3
BRUM-5979	JavaScript Agent	`node.type` not supported in IE11 and throws an error
JAVA-6355	Java Agent	Server Transaction times out and is abandoned in `projectreactor.io` based application with executor mode thread tracing enabl
JAVA-6560	Java Agent	Java Agent does not start when trying to instrument an application running with JDK1.6 due to log4j
JAVA-6965	Java Agent	In the dynamic agent attach scenario, the agent throws a `NullPointerException` and is unable to start due to a cached system p
JAVA-6617 and JAVA-7263	Java Agent	JMX attributes are marked as ignored if they are de-registered from the Mbean server. The attributes are not registered by the agent registered again with the Mbean server
JAVA-7139	Java Agent	Object Instance Tracking (OIT) does not work for all the Adopt JDK versions with both Hotspot and OpenJ9 distributions, the OIT wor AdoptJDK9-Hotspot
JAVA-7225	Java Agent	Fixed the VerifyError that is displayed on the application startup
JAVA-7279	Java Agent	Thread dump files captured are reported with a length of zero bytes
JAVA-7312	Java Agent	Upgrade the `commons-io:commons-io` library to 2.6 version
JAVA-7322	Java Agent	An exception is reported when Java Agent runs with an application that has JDK9+ and the application makes SQL calls
JAVA-7342	Java Agent	An error is reported in the agent when instrumenting apps deployed on jetty 6.x because the class `org.mortbay.jetty.Response#getHeaders` returns an enumeration object
SERVER-6167	Machine Agent	Machine Agent shows wrong IPV4 address for the network interface card

Controller (SaaS/On-Premises) Resolved Issues

Filter table:

Key	Product	Summary
ALERT-2307	Alert & Respond	Import of Action API fails with a `Could not import Actions: Transaction marked for rollback` error
ALERT-4482	Alert & Respond	Action suppression with object scope set to `node` resets to scope `application` when saved
ALERT-6028	Alert & Respond	Remove Alerting Template RBAC constraints
ALERT-6366	Alert & Respond	Editing or deleting custom roles for Alerting Templates causes the `CONFIG_ALERTING_TEMPLATES_PLANS doesn` error
AMCSEV-544	App Monitoring Cloud Services	Snapshots link broken in **Top Summary Lists** UI when RSD V2 is enabled
AMCSEV-941	App Monitoring Cloud Services	Customization of **Automatic Leak Detection** settings transforms `APP_AGENT` configuration
ANALYTICS-10310	Application Analytics	SQL data collectors are not enabled for Analytics when saved from new HTML screens
ANALYTICS-12272	Application Analytics	Cannot save Analytics searches if the search name exists in another account
ANALYTICS-12322	Application Analytics	No access to Mobile Code Issues event type on Analytics search screen
ANALYTICS-12397	Application Analytics	Option to create SQL Data collector from APM snapshot is missing from 4.5.10 Controller version
APMPLAT-11662	APM	Excessive logging of error messages has been fixed
APMPLAT-13124	APM	Filters for Transaction Snapshots has been fixed and is working correctly
CDM-6119	iOS	AppD sessions do not end post 10 mins inactivity as per the defined timeout, leading to long-running events

CDM-6198	Mobile RUM	**Crash Snapshot Details** dialog fails to open
CLUSTERMON-868	Controller	Some Cluster Agent metrics are listed even when there are not enough Server Visibility licenses
CLUSTERMON-1177	Controller	High load on Controller when monitoring the deleted and the succeeded pods in a cluster
COGENG-110	Cognition Engine	Selecting the business transaction in filters does not change the list of violations
COGENG-438	Cognition Engine	Variables are not being replaced with values intermittently, within an email template for anomaly events
COGENG-703	Cognition Engine	UI does not invoke the API to plot the abstract syntax tree (AST) graph intermittently
DBMON-3906	Database Visibility	The hardware metrics are not captured on the **MS SQL Database** dashboard or on the **Metric Browser** when using cluster
DBMON-6115	Database Visibility	The controller displays access key mismatch event when the database agents start with license rule access key
DBMON-6119	Database Visibility	The details pane on the **Queries Statistics Report** page is too small to view the data
DBMON-6307	Database Visibility	Remote DB Service with meta-info `jdbc|db|adodotnet` cannot be mapped to a database collector
DBMON-6529	Database Visibility	The `Connect backends to Database Visibility to track health status` message is displayed ever when it is not required
DBMON-6628, DBMON-6641	Controller	Missing the database nodes in the **Network Dashboard** flow map
DIAGPLAT-58	Controller	Optimizing `EventManagerBean.getEventDetailsInBatch` to prevent Internal Service and Event details errors
DIAGPLAT-425	Controller	Receiving 500 internal errors when attempting to view snapshots in an application
L4A-10645	Authentication	SAML encryption key is updated when a role is deleted
L4A-11010	Authentication	LDAP TLS client contexts should default to the newest TLS version supported by the Controller's bundled JRE
L4A-11361	Authentication	Nested group functionality not working
L4A-11846	Authentication	Reset password button is disabled
L4A-12066	Authentication	Optimizing LDAP sync
L4A-13322	Authentication	Unable to set an expiry date for account license
L4A-14540	Authentication	LDAP group sync fails due to deleted groups being included in results
METADATA-8112	Dashboards and Reports	Show error in Controller UI when a user doesn't have permission to create dashboards
METADATA-8228	Controller - Platform	The `CustomDashboardImportExportServlet` no longer writes a reply to the client before committing the transa
METADATA-9221	APM	No snapshots are created because the ADD stacktrace purger takes more than 20 hours to complete and holds the r lock for the metadata table
METRICSVC-5902	Metric Service	The client connection pool gets saturated as the connections to the Controller are not cleared after a task is performe
MQS-1028	Metadata	Expression widget with business transactions with `MetricHierarchy` shows no data
MQS-1203	Metric Query Service	Availability metric appears to be different in MySQL when querying for 10 minutes after the 10 minute boundary
MQS-1201	Metric Query Service	MQDC may not return the right metric data for a given time range
MQS-1207	Metric Query Service	Fix availability is below 100% for high retention enabled case
MQS-1209	Metric Query Service	Availability metric appears less than 100% even when it is 100%
REPORTS-591	Reports Service	Application Health Reports show `APPDYNAMICS_INTERNAL_DIAGNOSTICS` events
SERVER-6833	Controller UI	Server page displays total number of servers in the bottom right footer
SERVER-7505	Controller - Machine Agent	AppDynamics Agents UI displays `java.lang.NullPointerException` error for Machine Agents
SERVER-7725	Controller - Machine Agent	Machine Agent reports data continuously, but health rules on those metrics remain in a `Cancelled` status
SERVER-7855	Controller - Machine Agent	Metric hierarchy REST API returns empty response for `Root|*` in hierarchy path
SERVER-8249	Controller UI	OS icons are not displaying properly on on-premises Controller UI
SVCMON-639	Controller	Creating a business transaction group causes a `NullPointException`
SYNTH-5024	Synthetic Browser Monitoring	Synthetic Private Agent license usage for SaaS customers is incorrect
SYNTH-5027	Synthetic Browser Monitoring	500 Internal Server Error caused by null pointer exception
UIPLATF-9947	Custom Dashboards	Custom Time Range seems to be broken
UIPLATF-6208	Custom Dashboards	Absolute Layout: Widgets aren't saved correctly after creation
USERIMPACT-258	Controller	Read-only users cannot access the **Experience Journey Maps** UI

On-Premises Platform Resolved Issues

Filter table:

Key	Product	Summary	Ve
BRUM-5973	EUM Server	Method needed to specify domains or subdomains when setting the `access-control-allow-origin` HTTP header	20.4
CDM-6234	EUM Server	Change default ANR threshold value when `AnrSeverityThresholdConfig` is missing	20.4
ECONSOLE-2824	Enterprise Console	If error occurs, Enterprise Console job fails immediately instead of waiting 45 minutes	20.4
ECONSOLE-5553	Enterprise Console	A check was added to the AWS Controller installation with Aurora to help prevent database misconfiguration	20.4
ECONSOLE-5779	Enterprise Console	Controller installation no longer fails due to timeout	20.4
ECONSOLE-5906	Enterprise Console	Controller upgrade from 4.5.x to 20.3 no longer fails with `mysql_upgrade` version	20.4
ECONSOLE-5933	Enterprise Console	Upgrading Controller from 4.5.x to 20.3 fails with `mysql_upgrade` version check failure	20.4
EUMPLAT-778	EUM Server	Mobile Dashboard fails to load	20.4
EUMPLAT-790	EUM Server	License Service keeps making calls for individual account usage and terms	20.4
EUMPLAT-800	EUM Server	More usage data needs to be included into the `EUM Usage Trend` API	20.4
SYNTH-5140	Synthetic Private Agent	Firefox measurements fail for some specific URLs	20.4

Known Issues

End User Monitoring

License service makes individual accounts' usage and term calls to the Events Service.

20.3 Resolved and Known Issues

Agent Resolved Issues

Filter table:

Key	Product	Summary
ANALYTICS-12329	Analytics Agent	Denial of Service vulnerability found in Jackson libraries
ANALYTICS-12448	Java Agent	Node CPU spikes when running in Agentless Analytics mode after hitting Transaction Analytics license limit
CDM-5998	Android Agent	Add null protection while processing null response headers from the server
DBMON-6131	Database Monitoring	The ASC keyword in a custom metric query causes a query validation error
DBMON-6004	Database Monitoring	The Metric Browser displays incorrect hardware memory usage when the hardware monitoring for MS SQL server is disal
DBMON-4515	Database Monitoring	DB Agent does not capture the cursor metrics from Mongo DB
DBMON-5628	Database Monitoring	The Metric Browser does not capture the buffer and the cache memory metrics for a Linux collector
DOTNET-4309	.NET Agent	MSMQ backends resolving to downstream tiers
DOTNET-4000	.NET Agent	.NET Agent reports MSMQ Entrypoint as an exit call
DOTNET-4242	.NET Agent	.NET Agent throws excessive errors on ADO.NET exit calls
DOTNET-3536	.NET Agent	.NET Agent does not report WCF transactions if completion of asynchronous tasks result in an exception
DOTNET-4452	.NET Agent	Applications crash in HTTP data collection due to changes in .NET Core 3.0.3 and 3.1.2
DOTNET-4524	.NET Agent	Agent is logging for Netviz `Controller is not supported` even when customer is not using Network Agent
DOTNET-4230	.NET Agent	POCO instrumentation crashing instrumented application
IIB-228	IBM Integration Bus Agent	The App Agent Version column lists the release number of the agent as 0.0.0.0 in place of 4.5.5.0

IIB-36	IBM Integration Bus Agent	Agent initialization is extremely slow when instrumenting IIB9
JAVA-6554	Java Agent	Length of a string is limited to only 50 characters, when a URI is split using custom exit call naming rule with method para
JAVA-6631	Java Agent	Snapshot data added using the Agent API that is omitted from snapshots
JAVA-6923	Java Agent	E2E latency column missing for snapshots in the executor mode
JAVA-6975	Java Agent	Defer bytecode transformation logging after class loader lock is released
JAVA-7010	Java Agent	Java Agent API snapshot data collectors do not report statistics after the agent upgrade from 4.5.13 to 4.5.18
JAVA-7031	Java Agent	BT level error statistics exclude HTTP errors
JAVA-7091	Java Agent	Enabling EUM for Java Agent 4.5.18 for the `undertow-io` throws an error
JAVA-7160	Java Agent	False stall transactions reported with webflux and Reactor netty
JSNODE-19	Node.js Agent	`jackson -databind` library upgrade for the Node.js Agent
NETWORK-6054	Network Agent	Network Agent installed on Windows installs the NPCAP loopback adapter that can sometimes affect user traffic
NETWORK-6108	Network Agent	Network Agent installation on Windows Server 2008 R2 fails
NETWORK-6109	Network Agent	Network Agent crashes if port 3898 is being listened to by another application
NETWORK-6146	Network Agent	Request to fix read timeout and provide option to customize logging changes
NETWORK-6156	Network Agent	Network Agent installation fails if a higher version of the Visual C++ Redistributable is already installed on the machine
PHP-812	PHP Agent	PHP Agent installation on Apache logs irrelevant error messages
PHP-806	PHP Agent	PHP Agent does not detect DB queries with PHP Data Objects (PDO)
PHP-766	PHP Agent	Custom Match Regex Rule does not work with PHP 7.3
PHP-757	PHP Agent	Intermittent crash caused by SOAP client call interception in the PHP-FPM process
PYTHON-6	Python Agent	Avoid issues with functools.wraps on Python 2
PYTHON- 39	Python Agent	Remove unused third-party Java library from python-agent-proxy
SERVER-8069	Machine Agent	Update the JRE bundled with Machine Agent from 1.8.0_212 to 1.8.0_241
SERVER-6366	Machine Agent	Machine Agent RPM upgrade from 4.5.6 to later versions is now working correctly
SERVER-6705	Machine Agent	Machine Agent init.d start up scripts error is fixed
SERVER-8100	Machine Agent	`jackson-databind` library upgrade for the Machine Agent

Controller (SaaS/On-Premises) Resolved Issues

Filter table:

Key	Product	Summary
ALERT-3645	Health Rules	No audit log is captured when you disable or enable the health rule from Controller UI
ALERT-4482	Alerting	Action suppression with object scope set to `node` resets to scope `application` when saved
ALERT-4489	Health Rules	`OBJECT_UPDATED` events show old values as 0 when a condition is updated in health rule
ALERT-4581	Health Rules	X of Y health rule based on wildcard custom metrics does not work when one metric is not reported
ALERT-4919	Alerting	Logs from alerts is noisy and affecting the connection to node
ALERT-5753	Alerting	Alert/event links do not open and display an error, post controller upgrade
ALERT-5800	Alerting	Upgrade the Java library `C3P0` to version 0.9.5.2
ALERT-6064	Alerting	Upgraded the `C3P0` library in the Controller to version 0.9.5.5
ALERT-6489	Alerting	Hibernate session cache becomes excessively large when the event reactor index is refreshed
ALERT-6503	Alerting	Enable writing to digest tables based on flags
AMCSEV-145	Browser RUM	ZDT Controller restarts trigger re-transformation
AMCSEV-45	App Monitoring Cloud Services	Intermediate page is loaded briefly when user double-clicks an exception
AMCSEV-47	App Monitoring Cloud Services	Process snapshots are not filtered by a custom time range
AMCSEV-433	App Monitoring Cloud	Search option is not available the first time in **Transaction Snapshots** and filter options will not display

	Services	
AMCSEV-942	App Monitoring Cloud Services	Customization of **Automatic Leak Detection** settings transforms `APP_AGENT` configuration
ANALYTICS-12426	Analytics	SQL data collectors are not enabled for analytics when saved from new HTML screens
ANALYTICS-12489	Analytics	Option to create SQL Data collector from APM snapshot is missing from 4.5.10 Controller version
ANALYTICS-12485	Analytics	Cannot collect SQL parameters from Analytics when you use the new HTML screens
ANALYTICS-12121	Analytics	Page Name control in Browser Request Business Journey milestone unresponsive
ANALYTICS-12157	Analytics	XLM export does not have values in daily reporting.
ANALYTICS-12183	Analytics	Cannot remove Business transactions from Transaction Analytics UI
ANALYTICS-12202	Analytics	Table widget is missing progress bar and toolbar
ANALYTICS-12273	Analytics	Missing widget titles in Analytics Search Visualization
ANALYTICS-12519	Analytics	Time series chart won't remove data series which exists in the previous page but disappears in new response
ANLYTCS_ES-3484	Analytics	Client connection pool gets corrupted and cannot recover, requiring Controller restart
ANLYTCS_ES-3635	Analytics	Recreation of the client when the connection pool is corrupted
APMPLAT-12551	APM	Error graph shows correct information for time range greater than four hours
APMPLAT-13084	APM	Libagent-based agents reporting version as `0.0.0.0` are unable to register with the Controller
APMPLAT-13154	APM	Customer time range selection is incorrect
APMPLAT-13154	APM	Unable to filter the transaction snapshots list using the filter option in the UI
BRUM-5926	Browser RUM	**Page Limit Ajax Request Warning** documentation link does not work
CDM-5729	Mobile RUM	Health rule violation summaries are not rendered when the health rule name contains HTML/XML elements
CDM-5818	Mobile RUM	Health rule violation descriptions display incorrect health rule name containing HTML/XML elements
CLUSTERMON-74	Cluster Monitoring	Metrics are now available when the time range filter is set to `8 days`
CLUSTERMON-866, CLUSTERMON-34, CLUSTERMON-342	Cluster Monitoring	Cluster Agent should respect pod limit set by the Controller
CLUSTERMON-1137	Cluster Monitoring	Incoming Cluster Agent requests are directed to the wrong threadpool
CLUSTERMON-1173	Cluster Monitoring	AppDynamics Metric Browser stops responding when cluster monitoring is enabled and a large number of nodes registered
CLUSTERMON-1183	Cluster Monitoring	Default SIM health rules are being evaluated for clusters and pods leading to incorrect evaluation and health rule depicted in the UI
CLUSTERMON-1199	Cluster Monitoring	Normal pods count is negative
CUSTOMDASH-9325	Custom Dashboards	Custom widget title is no longer displaying after upgrading to 4.5.16
DBMON-6193	Database Monitoring	None of the DB collectors are reporting after Controller upgrade to 4.5.17
DBMON-6672	Database Monitoring	Removed the DBMon Service from the Controller
DBMON-6694	Database Monitoring	Upgrade the `mysql-connector-java:8.0.13` library for the DBMon Service to 8.0.20
DIAGPLAT-516	Health Rules	**Event Summary** page shows exception when accessed via link from email
DIAGPLAT-554	Events	Several ID blockers are skipped during the activation of the Controller
DIAGPLAT-556	Events	`Invalid Account: Principal` error is displayed when you load the events, post controller upgrade
DIAGPLAT-664	Events	Upgrade the H2 database engine library in the Controller
DIAGPLAT-667	Events	Upgrade the Apache Commons library `beanutil:1.8.3` in the Controller to 1.9.4
DIAGPLAT-681	Events	Upgrade the `mysql-connector-java` library in the Controller to 8.0.20
EUMPLAT-811, EUMPLAT-823	Controller - EUM	Upgrade the library `jackson-databind` to 2.11.0, and the libraries `dropwizard-core` and `dropwizard-val` 1.3.23 in the EUM client for the Controller
JAVA-7529	Controller	Update the Java Agent in the Controller to 20.4
L4A-7824	Accounts	New permissions added in a later release are not saved when added to existing roles
L4A-9790	RBAC	Cannot assign roles to the LDAP users when user attribute is updated on LDAP
L4A-9822	Accounts	Read-only user can access admin pages and see some admin entities
L4A-10035	Accounts	Error saving LDAP configuration if nested group attribute is an empty string in DB
L4A-10463	Accounts	User update operation leads to extra references in `AuthEntityHelper.CacheRepository`
L4A-12140	Admin Console	Optimize account listing for the Controller Admin Console

L4A-12311	Accounts	Email alerts are not generated as the notification configuration for AppDynamics account moves to invalid state
L4A-12292	Authentication	Including SAML config in Account DTO causes performance problems on controllers with many accounts
L4A-12065	Authentication	LDAP sync takes a lot of time in case there are more than 1,000 users with multiple groups assigned
L4A-12508	Authentication	LDAP sync fails while updating user outside of session
L4A-12712	Authentication	Update the OneLogin SAML version to 2.5.0 in the Controller
L4A-13321	Authentication	Unable to set an expiry date for the account license
L4A-13321	Authentication	LDAP group sync fails due to deleted groups being included in results
METADATA-7800	APM	Flowmap view and Detail screen no longer show mismatched data
METADATA-7922	Accounts	The accounts to application cache is now consistent with applications cache and database
METADATA-7956	APM	Server Health Status now shows correctly when the same dashboard is exported and imported
METADATA-7980	APM	BT liveness data now loads correctly from Blitz
METADATA-8305	Accounts	Admins can now enable the HSTS header as a setting
METADATA-8433	Dashboards	Metric Expression corrected in Custom dashboard
METADATA-8903	Metrics	Downgrade metric ID log level to debug to 20.3.0
METADATA-8999	Metrics	Upgraded the libraries `netty` to version 4.1.48.Final and `lettuce-core` to 5.1.8.RELEASE
METADATA-9120	Metrics	Upgrade the libraries `apache-tomcat` to 9.0.31, `dropwizard-validation` to 1.3.2, and `mysql-connector-` 8.0.20 for the Controller
METADATA-9149	Metrics	Upgrade the `mysql-connector` Java library to 8.0.20 in modules of the Controller
METADATA-9168	Metrics	Upgrade the `dom4j` library to 2.1.3 in the GlassFish application server of the Controller
MQS-1032	Metric Query Service	No data available when the custom time range is less than 30 minutes
MQS-1161	Metric Query Service	Split an insert statement for the Metric Hierarchy Store into smaller partitions
NETWORK-5276	Network Visibility	Error message needs to improve when user does not have permission to view Network agents tab
PLATCPS-14052	Controller	jsessionid no longer appears in URL and server log
REPORTS-715	Reports	Audit Report shows inconsistency when data is displayed in PDF and CSV formats
REPORTS-738	Reports	Unable to save the custom time range using the schedule report calendar fields
REPORTS-754	Reports	Remove older unused version of the AppDynamics Node.js Agent from the Reports Service bundle
SERVER-7017	Server Visibility	Server Visibility health rules work correctly with .NET compatible-mode enabled
SERVER-8254	Server Visibility	Upgrade the Apache `commons-httpclient` library used by the Controller to version 5.0
SVCMON-552	APM	Divide by Zero exception creates 500 error in Controller flowmap
SVCMON-597	APM	Slow application dashboard flowmap
SERVER-7017	Server Visibility	SIM API not working for .NET agent in compatibility mode
SERVER-7345	Server Visibility	ID column in SIM table changed from int type to BigInt type
SYNTH-4791	Browser Synthetic Monitoring	Business Transactions Missing Scroll Bar
SYNTH-5403	Browser Synthetic Monitoring	Synthetic Private Agent license usage for SaaS customers is incorrect
TMNT-27	APM	Functionality to set default landing screen is not working

On-Premises Platform Resolved Issues

Filter table:

Key	Product	Summary
ANALYTICS-12472	Enterprise Console	Stacktrace propagated in status code while handling ADQL exceptions
ANLYTCS_ES-3496	Events Service	Index compaction causing client timeouts

APMPLAT-13185	Enterprise Console - Controller	Upgrade of the Controller 20.3.0 fails when attempting to modify the database table `unresolved_backend_call_in`
ECONSOLE-4479	Enterprise Console	tzdata check is now correct
ECONSOLE-5691	Enterprise Console	Replication script no longer fails if the file group name contains whitespace
ECONSOLE-5718	Enterprise Console	MySql configuration no longer disables the recovery of InnoDB pages
ECONSOLE-5720	Enterprise Console	Running the final replication after incremental replication from the Enterprise Console UI causes the Enterprise Console
ECONSOLE-5804	Enterprise Console	Upgrade/Add Secondary HA directory validation fails because of a trailing forward slash in Controller path
ECONSOLE-5828	Enterprise Console	Add secondary job no longer fails when the volume mounted has a wider name
ECONSOLE-5934	Enterprise Console	Upgrading the Controller from version 4.5.x to version 20.3 fails with `mysql_upgrade` version check failure
ECONSOLE-5978	Enterprise Console	Upgrade the `mysql-connector` Java library to 8.0.20
ECONSOLE-5982	Enterprise Console	Upgrade the `dropwizard-validation` and `dropwizard-core` libraries to 2.0.8 for the Enterprise Console
ECONSOLE-5986	Enterprise Console	Upgrade the `jinjava` library to 2.5.3 for the Enterprise Console
ECONSOLE-5990	Enterprise Console	Upgrade the `okhttp` Java library to 4.5.0
ECONSOLE-5994	Enterprise Console	Upgrade the Apache `tomcat-jdbc` Java library to 9.0.31
ECONSOLE-5999	Enterprise Console	Upgrade the `dom4j` Java library to 2.1.3
ECONSOLE-6013	Enterprise Console	Retain the `internal_tmp_disk_storage_engine=MYISAM` setting in the `db.cnf` file through the upgrades of the and MySQL
ECONSOLE-6044	Enterprise Console	Purge script to remove unused artifacts from a prior Enterprise Console installation
EUMPLAT-572	EUM Server	Updated mysql-connector-java to 8.0.19 and mysql-version to 5.7.29
EUMPLAT-901	EUM Server	Use `jetty-util 9.4.18` to resolve the issue upgrading `SSLContextFactory`
SYNTH-4959	Synthetic Server	Prevent schedules from being deleted when MySQL is down in on-prem environments
SYNTH-5291	Synthetic Private Agent	The Page Load Time metric is sometimes captured in events.json but not captured in the HAR
SYNTH-5339	Synthetic Private Agent	Revert Chrome to version 64

20.2 Resolved and Known Issues

Controller Resolved Issues

There are no resolved issues for the Controller in this release.

Agent Resolved Issues

Filter table:

Key	Summary	Vers
BRUM-5889	`wrap()` changes the context of Event Listener while calling `pushApplyPopFromCausalityChain`()	20.2.0
CDM-5873	Fixed a bug in the iOS dSYM upload script that would have caused an error when attempting to set the correct timestamp for the uploaded metadata	20.2.0

On-Premises Resolved Issues

Filter table:

Key	Product	Summary	Version
SYNTH-4919	Page count error for Synthetic Private Agent license consumption	20.2.0	Synthetic Private Agent
SYNTH-5291	**Page Load Time** metric is sometimes captured in events.json but not in the HAR	20.2.1	Synthetic Private Agent
SYNTH-5333	Chrome upgrade to version 79 broke many non-W3C selenium commands	20.2.1	Synthetic Private Agent

Known Issues

There were no known issues in the 20.2 release.

Past Agent Releases

This page lists the agent enhancements for all calendar-versioned agent releases.

21.x Agent Releases

Analytics Agent

Version 21.5.0 - May 27, 2021

This release includes the third-party library upgrades:

- Jetty upgrade to version 2.34.
- Netty upgrade to version 4.1.63.Final.

Version 21.4.0 - April 22, 2021

This release includes the Jetty third-party library upgrade to version 9.4.39.v20210325.

Version 21.3.0 – March 25, 2021

This release includes the following enhancements:

- The Analytics Agent bundled in Machine Agent 20.8 has enhanced improvements
- The Analytics Agent Jetty libraries are upgraded
- The Analytics Agent Jersey is upgraded
- The Dropwizard and Hibernate Validator libraries are upgraded
- The unused HTTP methods in Analytics Agent have been disabled

Version 21.2.1 – February 25, 2021

The following third-party libraries have been upgraded.

- `com.fasterxml.jackson.core:jackson-databind:2.12.0`
- `com.google.guava:guava:30.0-jre`
- `commons-io:commons-io:2.8.0`
- `guava: 30.0-jre`
- `httpclient: 4.5.13`
- `jackson-core: 2.12.0`
- `jackson-databind: 2.12.0`
- `jetty-server to 9.4.30.v20200611.jar`
- `jetty-webapp to 9.4.33.v20201020.jar`
- `junit:junit:4.13.1`
- `netty-all: 4.1.55`
- `org.apache.ant:ant:1.10.9`
- `org.apache.httpcomponents:httpclient:4.5.13`
- `org.bouncycastle: bcprov-jdk15on:1.67`
- `org.eclipse.jetty:jetty-server:9.4.35.v20201120`
- `org.eclipse.jetty:jetty-webapp:9.4.35.v20201120`
- `org.exist-db.thirdparty.xerces:xercesImpl:2.12.1`
- `org.hibernate: hibernate-validator:6.1.6.Final`

Version 21.1.0

There was no release for 21.1.

Android Agent

Version 21.5.0 - May 17, 2021

Android Agent 21.5.0 released with minor enhancements and fixes.

Version 21.4.0

There was no release for 21.4.0.

Version 21.3.0

There was no release for 21.3.0.

Version 21.2.0 – February 18, 2021

Android Agent 21.2.0 released with minor bug fixes and API enhancements:

- Disable the agent to stop sending user data to the collector
- Disable crash reporting

Manual and automatic screenshots are limited to one screenshot per 10-second interval.

Version 21.1.0

There was no release for 21.1.

Apache Web Server Agent

Version 21.5.0

There was no release for 21.5.0.

Version 21.4.0

There was no release for 21.4.0.

Version 21.3.0

There was no release for 21.3.

Version 21.2.0

There was no release for 21.2.

Version 21.1.0

There was no release for 21.1.

AppDynamics AWS Lambda Extension for Serverless APM

Version 21.5.0 - May 10, 2021

To use AppDynamics AWS Lambda Extension for Serverless APM, choose ARN 10 which corresponds to these versions

- Node (npm) 21.5.286
- Python (PyPI) 20.11.411

Version 21.4.0

There was no release for 21.4.0.

Version 21.3.0

There was no release for 21.3.

Version 21.2.0

There was no release for 21.2.

Version 21.1.0

There was no release for 21.1.

C/C++ SDK

Version 21.5.0

There was no release for 21.5.0.

Version 21.4.0

There was no release for 21.4.0.

Version 21.3.0

There was no release for 21.3.

Version 21.2.0

There was no release for 21.2.

Version 21.1.0

There was no release for 21.1.

Cluster Agent

Version 21.5.0 - May 25, 2021

You can now use `resourcesNetViz` parameter within the `infraviz.yaml` filet to set the resources for the Network Visibility container.

Version 21.4.0

There was no release for 21.4.0.

Version 21.3.1 – March 26, 2021

This release includes the support for Infrastructure Visibility with Cluster Agent. You can now use the `infraviz.yaml` file with Cluster Agent Operator. For more information, see Install Infrastructure Visibility with the Cluster Agent Operator.

Version 21.3.0 – March 25, 2021

This release of Cluster Agent does not include any enhancement but fixes some minor issues.

Version 21.2.0 – February 24, 2021

This release of Cluster Agent includes the following updates:

- Cluster Agent is available with the 0.6.7 version of the operator. You can find the list of supported operators for each Cluster Agent at Cluster Agent and the Operator Compatibility Matrix.
- The deployment modes for the Cluster Agent Helm chart are now `PRIMARY` and `NAMESPACED` instead of `MASTER` and `NAMESPACED`. See Install the Cluster Agent with Helm Charts.

Version 21.1.0

There was no release for 21.1.

Cordova Plugin

Version 21.5.0 - May 18, 2021

Cordova Plugin 21.5.0 release includes iOS and Android Agents 21.5.0

Version 21.4.0

There was no release for 21.4.0.

Version 21.3.0

There was no release for 21.3.

Version 21.2.1 – February 19, 2021

Cordova Plugin 21.2.1 released with minor fixes and enhancements, and includes the latest version of the iOS and Android Agents.

Version 21.1.0

There was no release for 21.1.

Database Agent

Version 21.5.0

There was no release for 21.5.0.

Version 21.4.0 - April 21, 2021

This release includes the following enhancements:

You can now view:

- Costly operations in execution plan along with the warnings in the execution plan. See Database Query Execution Plan Window
- Additional Cassandra metrics such as Dropped Mutations, Max Partition Size, Messaging Latency, Read Failures, Repair Age Percentage, and Write Failures. See Database Monitoring Metrics.

Version 21.3.0

There was no release for 21.3.

Version 21.2.0 – February 17, 2021

This release of Database Agent collects additional Couchbase metrics. For the list of resolved issues, see Past Agent Resolved Issues.

Version 21.1.0

There was no release for 21.1.

Go SDK

Version 21.5.0

There was no release for 21.5.0.

Version 21.4.0

There was no release for 21.4.0.

Version 21.3.0

There was no release for 21.3.

Version 21.2.0

There was no release for 21.2.

Version 21.1.0

There was no release for 21.1.

IBM Integration Bus Agent

Version 21.5.0 - May 27, 2021

The IIB Agent 21.5.0 is released with bug fixes for cross-app BT correlation.

Version 21.4.1 - April 27, 2021

The IIB Agent 21.4.1 release introduces the Flow Visibility feature. The timings are collected only for flow start and end and any external calls made from within a flow. Therefore, the expected CPU cost of instrumenting the flows with the agent is reduced, especially for complex flows. See IIB Agent Flow Level Visibility.

Version 21.4.0 - April 5, 2021

The IIB Agent 21.4.0 release introduces the cross-application Business Transaction correlation feature that helps you to correlate the business transactions passing through an IIB application.

Version 21.3.0

There was no release for 21.3.

Version 21.2.0

There was no release for 21.2.

Version 21.1.0

There was no release for 21.1.

iOS Agent

Version 21.5.0 - May 12, 2021

iOS Agent 21.5.0 released with minor enhancements and fixes.

Version 21.4.0

There was no release for 21.4.0.

Version 21.3.0

There was no release for 21.3.

Version 21.2.0 – February 16, 2021

Manual and automatic screenshots are limited to one screenshot per 10-second interval. See See Past Resolved and Known Issues by Release. and iOS SDK Documentation.

Version 21.1.0

There was no release for 21.1.

Java Agent

Version 21.5.0 - May 26, 2021

Java Agent 21.5.0 released with minor enhancements and fixes.

Version 21.4.0 - April 20, 2021

- Java Agent 21.4.0 adds entry and exit support for Vert.x 4.0. See Java Supported Environments.
- This release adds exit support for WSO2 API Microgateway. See Java Supported Environments.
- An option to rediscover MBean servers periodically using a node property is introduced. See App Agent Node Properties.
- Support has been added for JDK16. See Java Supported Environments.

Version 21.3.0 – March 23, 2021

- Java Agent 21.3.0 adds entry and exit support for Micronaut 2.0. See Java Supported Environments.
- This release adds entry support for WSO2 API Microgateway. See Java Supported Environments.

Version 21.2.0 – February 15, 2021

The Java Agent 21.2.0 adds a new property to truncate the length of SQL statements. For more information, see App Agent Node Properties.

Version 21.1.1 - January 28, 2021

- Java Agent 21.1.1 adds support for Tomcat 10. For more information, see Java Supported Environments.
- This release adds support for Ktor and Netty frameworks. For more information, see Java Supported Environments.
- This release adds exit support for Couchbase database. For more information, see Java Backend Detection.

Version 21.1.0 - January 18, 2021

Java Agent 21.1.0 adds support for OOTB Detection for Maria DB. For more information, see Java Supported Environments.

JavaScript Agent

Version 21.5.0

There was no release for 21.5.0.

Version 21.4.0

There was no release for 21.4.0.

Version 21.3.0

There was no release for 21.3.

Version 21.2.0 – February 17, 2021

See Past Resolved and Known Issues by Release.

Version 21.1.0

There was no release for 21.1.

Java Serverless Tracer

Version 21.5.0

There was no release for 21.5.0.

Version 21.4.0

There was no release for 21.4.0.

Version 21.3.0

There was no release for 21.3.

Version 21.2.0

There was no release for 21.2.

Version 21.1.0

There was no release for 21.1.

Machine Agent

Version 21.5.0-3130 - May 19, 2021

Machine Agent 21.5.0 release:

- Deprecates the `metric.http.listener.host` property.

Upgrades these third-party libraries:

- `Ant 1.10.10`
- `Dropwizard 2.0.21`
- `Fabric8-Kubernetes 4.13.3`
- `Jackson 2.12.3`
- `Jetty 9.4.40.20210413`
- `OSHI 5.7.2`

Version 21.4.0-3075 - April 22, 2021

This release of Machine Agent :

- has updated Java 8 Azul to build version 8.52 and Java 11 Azul to build version 11.45.
- supports `diskstats` output for Linux kernel versions >= 4.18

The following third-party libraries have been upgraded.

- `dropwizard: 2.0.20`
- `guava: 30.0-jre`
- `Fabric8 Kubernetes: 4.13.2`
- `jackson-core: 2.12.0`
- `jackson-databind: 2.12.0`
- `jetty-server to 9.4.39.20210325`
- `Log4j2: 2.14.1`

Version 21.3.0-3059 – March 25, 2021

The Machine Agent release is bundled with the latest releases of the Analytics Agent and Network Agent.

Version 21.2.0 – February 25, 2021

The Machine Agent has been updated, tested, and certified to work with the following:

- `org.eclipse.jetty:jetty-server 9.4.36.v20210114`
- `org.eclipse.jetty:jetty-servlet 9.4.36.v20210114`
- `org.eclipse.jetty:jetty-servlets 9.4.36.v20210114`

- `org.bouncycastle:bcprov-jdk15on 1.68`
- `commons-net:commons-net 3.7.2`
- `commons-codec:commons-codec 1.15`
- `commons-validator:commons-validator 1.7`
- `io.dropwizard:dropwizard-jetty 2.0.18`
- `io.dropwizard:dropwizard-util 2.0.18`
- `com.google.guava:guava 30.1-jre`
- `com.fasterxml.jackson.* 2.12.1`
- `io.fabric8:kubernetes-client 4.13.1`

Version 21.1.0 – January 22, 2021

- The Machine Agent and Server Visibility on Linux bundles have been updated to Java 11 Azul JRE. If you upgrade the Linux Machine Agent to 21.1.0, then you must also upgrade all extensions to their latest versions to support Machine Agent 21.1.0. See Machine Agent Requirements and Supported Environments for details.
- The Machine Agent has updated the bundled Analytics Agent to version 20.10.0-7428.

.NET Agent

Version 21.5.0 - May 28, 2021

.NET Agent for Linux 21.5 incorporates significant architectural changes to accelerate the development of missing features in the Linux Agent relative to .NET Agent for Windows. These architectural changes do not affect .NET Agent for Windows (MSI and Microservices).

Because .NET Agent for Linux 21.5 is a significant architectural update, some issues may arise despite extensive testing performed. If you experience any blocking issues, please open a high priority support ticket where AppDynamics will provide workarounds or alternate options.

Generally, .NET Agent for Linux 21.5 maintains parity with existing features in .NET Agent for Linux < 21.5. However due to architectural updates, there are differences between .NET Agent for Linux >= 21.5 and .NET Agent for Linux < 21.5. Before you upgrade to .NET Agent for Linux 21.5, see Upgrade the .NET Agent for Linux.

Enhancements:

- Sensitive data filters for Linux. See Filter Sensitive Data with the .NET Agent.
- Custom exit points for Linux. See Exit Point Detection Rules.
- Instrumentation configuration enhancements for: `classIgnore`, `typeMatch 'Signature'`, `'IsSubClass' list` and `regex for Linux`.
- ASP.NET instrumentation enhancements for Linux. See ASP.NET Core Instrumentation Options.
- Redis backend detection for IP address endpoint for all platforms.
- EUM for .NET Core for Linux. See Browser RUM Supported Environments.
- .NET extends support for `BasicPublish` and `BasicConsume` in the RabbitMQ client >= 6.0.0 for all platforms.
- .NET Agent supports SSL mutual authentication for Azure PaaS app services for Windows only.
- The download package options for .NET Agent on Linux have been updated. The ".NET Agent - 64-bit alpine-linux" package has been discontinued and is replaced with the ".NET Agent - 64-bit linux" package.
- .NET Agent supports custom trusted certificate validation for all .NET Core Linux and Windows.

Version 21.4.0 - April 28, 2021

.NET Agent supports mutual authentication (mTLS) for Windows Agent for both .NET Full Framework and .NET Core applications; excluding Azure PaaS environments such as Azure App Services. See Resolved and Known Issues.

Version 21.3.0 – March 29, 2021

- .NET Agent for Windows running on .NET Core >= 3.1 supports async exit calls over HTTP to WCF services. The remote address, full URL, host, and port identifying properties and correlation are supported.
- .NET Agent supports the Controller service proxy feature.
- New instrumentation options for ASP.NET Core.

See Past Resolved and Known Issues by Release for details

Version 21.2.0 – February 24, 2021

he .NET 21.2.0 release includes:

- .NET Agent for Windows on .NET Core now automatically captures errors for loggers implementing the Microsoft.Extensions.Logging.ILogger API. The standard Error Detection configuration options apply.
- **CosmosDB for .NET Core on Windows is supported.**
- Added optimizations for Async Task tracking and Netviz socket tracking.

See Past Resolved and Known Issues by Release.

Version 21.1.0 – January 29, 2021

- Windows Agents now support End User Monitoring (EUM) for ASP.NET Core Web applications running on .NET Core 3.1 and .NET 5, including automatic Business Transaction correlation and automatic JavaScript Agent injection. Both MVC and Razor Pages are supported.
- Redis instrumentation is now supported on both .NET Full Framework and Core Framework on Windows.

- The Windows Agents can now automatically set the `HttpOnly` flag to `false` for EUM cookies on ASP.NET. To enable, set the `eum-auto-httponly-false` boolean node property to `true`.

Network Agent

Version 21.5.0

There was no release for 21.5.0.

Version 21.4.0

There was no release for 21.4.0.

Version 21.3.0 – March 25, 2021

This release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release.

Version 21.2.0 – February 25, 2021

This release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release for details.

Version 21.1.0

There was no release for 21.1.

Node.js Agent

Version 21.5.0 - May 28, 2021

This release includes:

- Support for gRPC. See gRPC Support for the Node.js Agent.
- Availability for cross-application Business Transactions.

Version 21.4.0 - April 16, 2021

This release includes no new features, but bug fixes. See Resolved and Known Issues.

Version 21.3.0

There was no release for 21.3.

Version 21.2.0 – February 25, 2021

This release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release for details.

Version 21.1.0 – January 29, 2021

- You can now use Node.js 15.
- You can also tune Analytics data limits with the agent settings.

Node.js Serverless Tracer

Version 21.5.286 - May 10, 2021

The Node.js Serverless Tracer has been updated to 21.5.286

Version 21.4.0

There was no release for 21.4.

Version 21.3.278 – March 26, 2021

The Node.js Serverless Tracer has removed `request 2.88.0` library and added axios `0.21.1` library. See Past Resolved and Known Issues by Release for details.

Version 21.2.0

There was no release for 21.2.

Version 21.1.0

There was no release for 21.1.

PHP Agent

Version 21.5.0 - May 26, 2021

PHP Agent 21.5.0 released with bug fixes.

Version 21.4.0

There was no release for 21.4.

Version 21.3.0

There was no 21.3 release.

Version 21.2.0 – February 24, 2021

This release introduces an environment variable to set a unique host ID for the PHP Agent. See PHP Agent Configuration Settings.

Version 21.1.0 – January 27, 2021

This release includes no new features, but bug fixes. See Past Resolved and Known Issues for details.

Python Agent

Version 21.5.0

There was no release for 21.5.0.

Version 21.4.0

There was no release for 21.4.

Version 21.3.0

There was no 21.3 release.

Version 21.2.0 – February 2, 2021

This release includes no new features, but bug fixes. See Past Resolved and Known Issues for details.

Version 21.1.0

There was no 21.1 release.

Python Serverless Tracer

Version 21.5.0

There was no release for 21.5.0.

Version 21.4.0

There was no release for 21.4.

Version 21.3.0

There was no 21.3 release.

Version 21.2.0

There was no 21.2 release.

Version 21.1.0

There was no 21.1 release.

React Native Agent

Version 21.5.0

There was no release for 21.5.0.

Version 21.4.0 - April 5, 2021

This release includes an API to manually track custom HTTP requests. See Customize the React Native Instrumentation.

Version 21.3.0

There was no 21.3 release.

Version 21.2.1 – February 19, 2021

React Native Agent 21.2.1 supports `CrashReportCallback` to receive a report for native crashes. See Customize the React Native Instrumentation.

Version 21.1.0

There was no 21.1 release.

SAP

Version 21.5.0

There was no release for 21.5.0.

Version 21.4.0

There was no release for 21.4.

Version 21.3.0

There was no 21.3 release.

Version 21.2.0 – February 26, 2021

See Release Notes Version 21.2.0.

Version 21.1.0

There was no 21.1 release.

Synthetic Hosted Agent

Version 21.5.0

There was no release for 21.5.0.

Version 21.4.0

There was no release for 21.4.

Version 21.3.0

There was no 21.3 release.

Version 21.2.0 – February 16, 2021

Chrome browser version 86 is now available. See Chrome Version 86 for details.

Xamarin Agent

Version 21.5.1 - May 18, 2021

The Xamarin 21.5.1 Agent includes newly exposed APIs to:

- Configure a semi-automatic HTTP tracker using the `HttpMessageHandler`
- Transform URLs for network requests
- Configure, disable, block, and unblock screenshots

See Customize the Xamarin Instrumentation.

Other enhancements:

- When setting up the Xamarin Agent, you must add the AppDynamics.Agent package and call an extra line of code in `MainActivity.cs` under `OnCreate`. See Instrument Xamarin Applications.

Version 21.4.0

There was no release for 21.4.

Version 21.3.0

There was no 21.3 release.

Version 21.2.0 – February 19, 2021

Xamarin Agent 21.2.0 release includes fixes from the 21.2.0 JavaScript Agent, iOS, and Android Agent.

Version 21.1.0

There was no 21.1 release.

20.x Agent Releases

Analytics Agent

Version 20.12.0

There was no release for 20.12.

Version 20.11.0

There was no release for 20.11.

Version 20.10.0

There was no release for 20.10.

Version 20.9.0

There was no release for 20.9.

Version 20.8.0

There was no release for 20.8.

Version 20.7.0

There was no release for 20.7.

Version 20.6.0 – June 25, 2020

This release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.5.0 – May 26, 2020

- Oracle JRE has been replaced with Azul's Zulu JRE in the Window Analytics Agent Bundle.
- Third-party libraries have been upgraded.

Version 20.4.0

There was no 20.4 release for the Analytics Agent.

Version 20.3.0 – March 4, 2020

This release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.2.0

There was no release for the Analytics Agent.

Android Agent

Version 20.12.0

There was no release for 20.12.

Version 20.11.1 - November 17, 2020

Android Agent 20.11.0 released with minor enhancements and fixes.

Version 20.10.0 – October 9, 2020

Android Agent 20.10.0 captures mobile device metrics with which you can filter performance data, such as network requests, crashes and code issues, based on device specifications, such as CPU type, memory, disk space, and battery. See Mobile RUM Metrics for details.

Version 20.9.0

There was no 20.9 release for Android Agent.

Version 20.8.0

There was no 20.8 release for Android Agent.

Version 20.7.1 – July 2, 2020

Android Agent 2020.7.0.1710 is released. See Past Resolved and Known Issues by Release.

Recommendations

- If your project uses Android Gradle Plugin version < 4.5.1, use Android Agent 20.4.0.
- If your project uses Android Gradle Plugin version >= 4.5.1, use Android Agent >=20.7.1.

Version 20.6.0

There was no 20.6.0 release for the Android Agent.

Version 20.5.0 – May 13, 2020

The Android Agent has updated the Android Gradle plugin dependency to version 3.4.1 or higher. If you are unable to update to Android Gradle Plugin 3.4.1 or higher, you are recommended to use Android Agent 20.4.0 or earlier.

Version 20.4.0

There was no 20.4.0 release for the Android Agent.

Version 20.3.1 - March 31, 2020

This release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release.

Version 20.3.0 – March 4, 2020

This release includes optimizations and small code changes.

Version 20.2.0

There was no 20.2.0 release for the Android Agent.

Apache Web Server Agent

Version 20.12.0

There was no release for 20.12.

Version 20.11.0

There was no release for 20.11.

Version 20.10.0

The release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release.

Version 20.9.0 – September 21, 2020

The release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release.

Version 20.8.0

There was no 20.8 release for the Apache Web Server Agent.

Version 20.7.0 – July 27, 2020

Apache Agent 20.7.0 is released with small code changes and optimizations. See Past Resolved and Known Issues by Release.

Version 20.6.0 – June 24, 2020

This release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release.

Version 20.5.0 – May 14, 2020

This release includes optimizations and small code changes.

Version 20.4.0

There was no 20.4.0 release for the Apache Web Server Agent.

Version 20.3.0

There was no 20.3.0 release for the Apache Web Server Agent.

Version 20.2.0

There was no 20.2.0 release for the Apache Web Server Agent.

AppDynamics AWS Lambda Extension for Serverless APM

Version 20.12.0

There was no release for 20.12.

Version 20.11.0

There was no 20.10 release for the AppDynamics AWS Lambda Extension for Serverless APM.

Version 20.10.0 – October 8, 2020

AppDynamics AWS Lambda Extension for Serverless APM released. See Use the AppDynamics AWS Lambda Extension to Instrument Serverless APM at Runtime.

C/C++ SDK

Version 20.12.0

There was no release for 20.12.

Version 20.11.0

There was no 20.11 release for the C/C++ SDK.

Version 20.10.0

There was no 20.10 release for the C/C++ SDK.

Version 20.9.0 – September 30, 2020

This release includes new features and bug fixes.

- The C/C++ SDK is now compatible with calendar versioning.
- `appd_sdk_add_app_context` now logs Controller and application information. See C/C++ SDK Reference.
- The C/C++ SDK now supports Custom Events. You can create a custom event, add data to the event, and report the event to the Controller. See C/C++ Agent SDK Resource Management.
- You can enable/disable Analytics reporting. When disabled, the Analytics-related logging messages are suppressed. See C/C++ SDK Reference.

Version 20.8.0

There was no 20.8 release for the C/C++ SDK.

Version 20.7.0

There was no 20.7.0 release for C/C++ SDK.

Version 20.6.0

There was no 20.6.0 release for the C/C++ SDK.

Version 20.5.0

There was no 20.5.0 release for the C/C++ SDK.

Version 20.4.0

There was no 20.4.0 release for the C/C++ SDK.

Version 20.3.0

There was no 20.3.0 release for the C/C++ SDK.

Version 20.2.0

There was no 20.2.0 release for the C/C++ SDK.

Cluster Agent

Version 20.12.1-1948 – December 18, 2020

There are no enhancements in this release. For the resolved issue, see Past Resolved and Known Issues by Release.

Version 20.12.0-1937 – December 17, 2020

There are no enhancements in this release. For the resolved issue, see Past Resolved and Known Issues by Release.

Version 20.11.0-1877 – November 25, 2020

This release provides the following enhanced capabilities:

- The Cluster Agent can now monitor 1500 pods and 3000 containers (for 2 containers per pod) or 2250 pods and 2250 containers (for 1 container per pod). See Cluster Agent Performance Certification.
- You can configure the requests and limits of CPU and memory resources for the Cluster Agent. See the `resources` parameter at Configure the Cluster Agent.
- The `instrumentationRules` parameter is now mandatory to enable auto-instrumentation of supported applications. This ensures auto-instrumentation is enabled for the required application deployment. See Enable Auto-Instrumentation for the Supported Applications.
  If you do not specify any value for the `matchString` parameter within the `instrumentationRules`, the Cluster Agent takes into account the value specified in the `defaultInstrumentMatchString` and instruments all the deployments that satisfy the default value. See Cluster Agent Configuration for Instrumentation Rules.

Version 20.10.0 – October 23, 2020

- This release of Cluster Agent provides enhanced configuration for auto-instrumentation.
  - You can use the custom configuration for instrumenting the supported applications by using the `customAgentConfigSource` parameter within instrumentation rules. See Cluster Agent Configuration for Instrumentation Rules.
  - The `imagePullPolicy` option of the `imageInfo` parameter can now be used within the instrumentation rules. See Cluster Agent Configuration for Instrumentation Rules.
- This release supports custom SSL for instrumenting the supported applications. For Node.js you must use the custom configuration to use the custom SSL certificate. See, Supported Applications for Auto-Instrumentation.

Version 20.9.1-1675 – September 30, 2020

The AppDynamics Operator that is to be used by the Cluster Agent has been updated to version 0.6.2.

Version 20.9.0-1662 – September 25, 2020

This release of Cluster Agent provides the following enhancements:

- You can now auto-instrument the `Node.js` and the `.Net` Core on Linux applications along with the Java applications. See Enable Auto-Instrumentation of Supported Applications.
- You can now limit the number of auto-instrumentation tasks that Cluster Agent can run simultaneously. See the `numberOfTaskWorkers` property at Cluster Agent Configuration for Auto-Instrumentation. By default, Cluster Agent can perform two auto-instrumentation deployments simultaneously.

Version 20.8.0 – August 26, 2020

Cluster Agent released with minor enhancements and fixes.

Version 20.7.0 – July 23, 2020

With this release of Cluster Agent, you can:

- Use auto-instrumentation for the Java applications that are deployed using the DeploymentConfig resource. See Supported Resources for Instrumentation.
- Set tolerations on Cluster Agent pod. See tolerations in Configure the Cluster Agent YAML File.
- Deploy the Cluster Agent with Helm Charts. See Deploy the Cluster Agent with Helm Charts.
- Monitor the cluster-specific events from the **Clusters** dashboard. See the **Events** tab in Monitor Cluster Health.

Version 20.6.0 – June 26, 2020

- Cluster Agent now supports instrumenting the Java applications that are deployed as StatefulSet. This release introduces the parameter `resourcesToInstrument` to configure auto-instrumentation based on the resource that is used for deploying the Java application. See Cluster Agent Configuration for Auto-Instrumentation.
- You can now configure `runAsUser` and `runAsGroup` as instrumentation rules. This configuration takes precedence over the default configuration. See Cluster Agent Configuration for Instrumentation Rules.
- You can use the `appNameStrategy` parameter to choose the option for naming the AppDynamics Application. See Options for Naming the Appdynamics Application.

Version 20.5.0 – May 29, 2020

Cluster Agent can now enable auto-instrumentation for Java applications. See Enable Auto-Instrumentation of Java Applications.

Version 20.4 – April 28, 2020

This release includes the following enhancements:

- Support for Cluster Agent on Rancher Kubernetes Engine (RKE) 1.0.4 with Kubernetes 1.17. See Cluster Agent Requirements and Supported Environments.
- Support for Cluster Agent on Google GKE 1.14
- Support for monitoring Kubernetes stateful sets
- Changes have been made to preserve the configuration for monitored namespaces. See the `nstoMonitor` property in Configure the Cluster Agent and Use the Cluster Agent.

Version 20.3.0 – March 6, 2020

- The Cluster Agent can blocklist and allowlist pods based on regular expression and domains.
- You can configure the maximum number of Cluster Agents that can report to an account.
- You can configure the maximum number of pods that can be associated with any Cluster Agent.

Version 20.2.0

There was no release for the Cluster Agent.

Cordova Plugin

Version 20.12.0

There was no release for 20.12.

Version 20.11.0 – November 19, 2020

Cordova Plugin 20.11.0 released with minor fixes and optimizations. Cordova Plugin 20.11.0 is integrated with Android Agent 20.11.1.

Version 20.10.0 – October 16, 2020

This release includes optimizations and small code changes.

Version 20.9.0

There was no 20.9 release for the Cordova Plugin.

Version 20.8.0

There was no 20.8 release for the Cordova Plugin.

Version 20.7.0 – July 13, 2020

Cordova Agent 20.7.0 captures `hybridAgentType` and `hybridAgentVersion`.

Version 20.6.0

There was no 20.6.0 release for the Cordova Plugin.

Version 20.5.0

There was no 20.5.0 release for the Cordova Plugin.

Version 20.4.0

There was no 20.4.0 release for the Cordova Plugin.

Version 20.3.0 – March 18, 2020

The Cordova Plugin has now been tested and certified to work with the following:

- Cordova 9.0.0
- Cordova iOS 4.5.5/5.0.0
- Cordova 8.0.0
- Ionic CLI 6.1.0
- Node 12.16.1
- npm 6.13.4

Version 20.2.0

There was no 20.2.0 release for the Cordova Plugin.

Database Agent

Version 20.12.0-2183 – December 16, 2020

- Database Agent now supports mutual (client and server) authentication. See Mutual Authentication.
- You can now use the service record (SRV record) within the connection string while configuring the MongoDB collector. See **SRV Record** at Configure MongoDB Collectors.

Version 20.11.0

There was no 20.11 release for the Database Agent.

Version 20.10.0 – October 27, 2020

This release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.9.0 – September 16, 2020

The release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release.

Version 20.8.0

There was no 20.8 release for the Database Agent.

Version 20.7.0 – July 28, 2020

The Database Agent 20.7 and later are not compatible with Controller versions 4.5.0 and 4.5.1. Therefore, some of the features may not work when the Controller version is 4.5.0 or 4.5.1.

For the resolved issue, see Past Resolved and Known Issues by Release.

Version 20.6.0

There was no 20.6.0 release for Database Agent.

Version 20.5.0 – May 26, 2020

- You can now create multiple Database Agents as different Windows services with different names. For more information, see Start the Database Agent Automatically on Windows.
- Database Agent adds support for Windows 64-bit. See Database Visibility Supported Environments.
  From 20.5 release onwards, Database Visibility is not supported on Windows 32-bit.

Version 20.4 – April 28, 2020

This release includes the following new features:

- Support for Microsoft SQL Server 2019
- Support for Cassandra database
- Customize the sampling interval
- Capture details to integrate Database Visibility with the Application Flow Map

For the issues that are fixed in this release, seePast Resolved and Known Issues by Release.

Version 20.3.0 – March 20, 2020

This release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.2.0

There was no 20.2.0 release for Database Agent.

Go SDK

Version 20.12.0

There was no release for 20.12.

Version 20.11.0

There was no release for 20.11.

Version 20.10.0

There was no 20.10.0 release for the Go SDK.

Version 20.9.0

There was no 20.9.0 release for the Go SDK.

Version 20.8.0

There was no 20.8.0 release for the Go SDK.

Version 20.7.0

There was no 20.7.0 release for the Go SDK.

Version 20.6.0

There was no 20.5.0 release for the Go SDK.

Version 20.5.0

There was no 20.5.0 release for the Go SDK.

Version 20.4.0

There was no 20.4.0 release for the Go SDK.

Version 20.3.0

There was no 20.3.0 release for the Go SDK.

Version 20.2.0

There was no 20.2.0 release for the Go Agent.

IBM Integration Bus Agent

Version 21.1.0

There was no release for 21.1.

Version 20.12.0

There was no release for 20.12.

Version 20.11.0

There was no release for 20.11.

Version 20.10.0

There was no 20.10 release for the IBM Integration Bus Agent.

Version 20.9.0 – September 24, 2020

The release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release.

Version 20.8.0

There was no 20.8 release for the IBM Integration Bus Agent.

Version 20.7.0 – July 2, 2020

AppDynamics IIB Agent introduces a configurable setting for the log file size. See Install the IIB Agent.

Version 20.6.0

There was no 20.6.0 release for the IBM Integration Bus Agent.

Version 20.5.0 – May 14, 2020

- The Appdynamics IIB Agent now adds support for ACE 11 on the AIX platform. See IIB Supported Environments.
- This release also includes optimizations and small code changes.

Version 20.4.0

There was no 20.4.0 release for the IBM Integration Bus Agent.

Version 20.3.0 – March 31, 2020

The tracer's version is now compatible with calendar versioning. This release includes no new enhancements.

Version 20.2.0

There was no 20.2.0 release for the IBM Integration Bus Agent.

iOS Agent

Version 20.12.0 – December 15, 2020

You can now disable the iOS Agent to stop sending user data to the collector. See Customize the iOS Instrumentation.

Version 20.11.0

There was no release for the iOS Agent.

Version 20.10.0.2298 – October 7, 2020

iOS Agent 20.10.0 captures mobile device metrics with which you can filter performance data, such as network requests, crashes and code issues, based on device specifications, such as CPU type, memory, disk space, and battery. See Mobile RUM Metrics for details.

For information on the iOS 14 privacy guidelines, see iOS Data Collection Disclosure. For bug fixes, see Past Resolved and Known Issues by Release.

Version 20.9.0

There was no release for the iOS Agent.

Version 20.8.0 – August 20, 2020

The iOS Agent released with minor enhancements and fixes. See Past Resolved and Known Issues by Release.

Version 20.7.0.2209 – July 8, 2020

Filtered URLs are logged when the logging level is set to `verbose`.

Version 20.6.0

There was no release for the iOS Agent.

Version 20.5.0

There was no release for the iOS Agent.

Version 20.4.0

There was no release for the iOS Agent.

Version 2020.3.0-2006 – March 2, 2020

This release includes optimizations and small code changes.

Version 20.2.0

There was no release for the iOS Agent.

Java Agent

Version 20.12.0

There was no release for 20.12.

Version 20.11.0 - November 27, 2020

- Java Agent 20.11.0 adds exit support for Vertx 3.9. For more information, see Java Supported Environments.
- This release makes the *Executor* mode the default mode for handling asynchronous transactions. See Configure the Thread Correlation in Java Agent.
- This release adds a property for optimized heap mode enabling. See Optimize Heap Mode.

Version 20.10.0 – October 20, 2020

- This release adds an asynchronous entry and exit support for gRPC. For more information, see Java Supported Environments.
- This release certifies support for Java 15. For more information, see Java Supported Environments.
- This release adds exit support for the HTTP4 blaze client.

Version 20.9.0 – September 22, 2020

- This release adds the synchronous entry and exit support for gRPC. For more information, see Java Supported Environments.
- This release adds support for the following thread handoff across Reactive Extension for Java (RxJava 2): **Observable** in addition to the thread handoff **Single**.

Version 20.8.0 – August 21, 2020

This release verifies support for GraalVM using the Graal compiler as a HotSpot JIT.

Version 20.7.0 – July 31, 2020

- This release adds support for MYSQL 6.0 and 8.0. See Java Supported Environments.
- This release adds support for MongoDB drivers (synchronous, asynchronous, and reactivestreams) up to v4.x. See Java Supported Environments.
- This release verifies support for Spring 5.2. See Java Supported Environments.
- This release adds support for Spring Cloud Gateway.
- This release adds support for the following thread handoffs across Reactive Extension for Java (RxJava 2): **Maybe**, **Completable**, and **Flowable** in addition to the thread handoff **Single**.
- An additional artifact for Java Agent to include the security compliant versions of the third-party dependencies is included. An agent compatible with Java version 7 will no longer be available for Java applications running on PCF.

Version 20.6.0 – June 23, 2020

- Java Agent 20.6.0 adds support for Oracle 10, 11, 12, 18, and 19 drivers. See Java Supported Environments.
- Support for Spring WebFlux and WebClient 5.2 is added. See Java Supported Environments.
- A node property is added to prevent duplicate ADRUM headers from being injected in the HTTP response. See App Agent Node Properties (D-E).
- Java Agent 20.6.0 adds support to the service proxy feature. See Service Proxy Overview.

Version 20.5.0 – May 21, 2020

- Java Agent 20.5.0 adds support for Vertx 3.8. See Java Supported Environments.
- You can now capture the raw SQL queries for databases such as MongoDB, DynamoDB, and CassandraDB. See Call Graph Settings.

Version 20.4.0 – April 22, 2020

- Java Agent 20.4.0 adds support for netty version 4.x async exit. See Java Supported Environments.

- This release adds Java 14 certification for AdoptOpenJDK, Azul Zulu OpenJDK, Oracle, and Open Source OpenJDK. See Java Supported Environments.

Version 20.3.0 – March 20, 2020

- Java Agent version 20.3.0 adds support for netty version 4.x async entry.
- Support added for Rx Java 2 Single.
- Support added for Akka HTTP Non-Route DSL.
- Support to enable business transaction correlation using `SOAP:Header` with JBoss. To enable the correlation only for JBoss, you must create a Servlet exclude rule.

Agentless Transaction Analytics

- You can now disable agentless Transaction Analytics at the tier and node level.

Version 20.2.0

There was no release for the Java Agent.

JavaScript Agent

Version 20.12.0

There was no release for 20.12.

Version 20.11.0

There was no 20.11.0 release for the JavaScript Agent.

Version 20.10.0 – October 21, 2020

JavaScript Agent 20.10.0 released with minor optimizations, including to how Visually Complete Time (VCT) is calculated. For bug fixes, see Past Resolved and Known Issues by Release.

Version 20.9.0 – September 15, 2020

JavaScript Agent 20.9.0 was released. See Past Resolved and Known Issues by Release for more information.

Version 20.8.0 – August 5, 2020

JavaScript Agent released with bug fixes. See Past Resolved and Known Issues by Release for more information.

Version 20.7.0

There was no 20.7.0 release for the JavaScript Agent.

Version 20.6.0.3177 – June 17, 2020

Angular 9 is now supported.

Version 20.5.0.3144 – May 15, 2020

Images without extensions or dots (.) are now classified as "images" instead of "other."

Version 20.4.0.3090 – April 14, 2020

This release includes bug fixes but no new features. See Past Resolved and Known Issues by Release for details.

Version 20.3.0-3009 – March 18, 2020

The JavaScript Agent, by default, limits the number of Ajax requests reported per page. Starting with this version, you can configure the JavaScript Agent to report an unlimited number of Ajax requests per page.

Version 20.2.0-2928 – February 13, 2020

This release includes bug fixes but no new features. See Past Resolved and Known Issues by Release for details.

Java Serverless Tracer

Version 20.12.0

There was no release for 20.12.

Version 20.11.1400

The Java Serverless Tracer has added an environment variable, `APPDYNAMICS_HTTP_TIMEOUT_MS`, which allows you to set the timeout in milliseconds for all the HTTP calls made by the tracer. See Past Resolved and Known Issues by Release for details.

Version 20.10.0

There was no 20.10 release for the Java Serverless Tracer.

Version 20.9.0

There was no 20.9 release for the Java Serverless Tracer.

Version 20.8.0

There was no 20.8 release for the Java Serverless Tracer.

Version 20.7.0

There was no release for 20.7.

Version 20.6.0

There was no release for the Java Serverless Tracer Agent.

Version 20.5.0

There was no release for the Java Serverless Tracer Agent.

Version 20.4.0

There was no release for the Java Serverless Tracer Agent.

Version 20.3.1391 – March 17, 2020

The tracer's version is now compatible with calendar versioning. This release includes no new enhancements or resolved issues.

Version 20.2.0

There was no release for the Java Serverless Tracer Agent.

Machine Agent

Version 20.12.0-3016 - December 16, 2020

This release provides the following enhancements:

- The Machine Agent now prepends the hostname in the log files.
- The Machine Agent has been updated, tested, and certified to work with the following:
    - `com.fasterxml.jackson.core:jackson-databind:2.12.0`
    - `com.google.guava:guava:30.0-jre`
    - `commons-io:commons-io:2.8.0`
    - `junit:junit:4.13.1`
    - `org.apache.ant:ant:1.10.9`
    - `org.apache.httpcomponents:httpclient:4.5.13`
    - `org.bouncycastle: bcprov-jdk15on:1.67`
    - `org.eclipse.jetty:jetty-server:9.4.35.v20201120`
    - `org.eclipse.jetty:jetty-webapp:9.4.35.v20201120`
    - `org.exist-db.thirdparty.xerces:xercesImpl:2.12.1`
    - `org.hibernate: hibernate-validator:6.1.6.Final`

Version 20.11.1-3044 - January 26, 2021

This release provides the following enhancements:

- The Machine Agent has been updated, tested, and certified to work with the following:
    - `com.fasterxml.jackson.core:jackson-databind:2.12.0`
    - `com.google.guava:guava:30.0-jre`
    - `commons-io:commons-io:2.8.0`
    - `junit:junit:4.13.1`
    - `org.apache.ant:ant:1.10.9`
    - `org.apache.httpcomponents:httpclient:4.5.13`
    - `org.bouncycastle: bcprov-jdk15on:1.67`

- `org.eclipse.jetty:jetty-server:9.4.35.v20201120`
- `org.eclipse.jetty:jetty-webapp:9.4.35.v20201120`
- `org.exist-db.thirdparty.xerces:xercesImpl:2.12.1`
- `org.hibernate: hibernate-validator:6.1.6.Final`

Version 20.11.0.2915 - November 19, 2020

The Machine Agent Azul Zulu JRE version has been updated to 8.50 (maps to OpenJDK version 8u272). Machine Agent OSHI library upgraded to version 5.3.5.

Version 20.10.0-2813 - October 22, 2020

This release includes optimizations and small code changes. For bug fixes, see Past Resolved and Known Issues by Release.

Version 20.9.0 – September 21, 2020

Machine Agent 20.9.0-2763 was released. Machine Agent OSHI library upgraded to version 5.2.5. See Past Resolved and Known Issues by Release.

Version 20.8.0.2713 – August 20, 2020

Machine Agent released with bug fixes. See Past Resolved and Known Issues by Release for more information.

Version 20.7.0-2694 – July 23, 2020

When using Docker Visibility, if the unique host ID setting is not configured to use container ID in host network mode, the Machine Agent now automatically registers the container using the container ID as the host ID. See Using Docker Visibility with Kubernetes.

Version 20.6.0-2676 – June 25, 2020

The Machine Agent now has the ability to identify container processes by container names only. See Configuring Docker Visibility.

Version 20.5.1-2635 – May 29, 2020

The Machine Agent has been updated, tested, and certified to work with the following:

- `ant v1.10.8`
- `commons-fileupload v1.4.0`
- `hibernate-validator v6.1.5`
- `snakeyaml v1.26`

Version 20.5.0-2617 – May 20, 2020

Azul JRE 8.0.252 is replacing Oracle JRE as a component included in the Machine Agent. See Machine Agent Requirements and Supported Environments.

Version 20.4.0-2571 – April 27, 2020

The Machine Agent has been updated, tested, and certified to work with the following:

- `ch.qos.logback:logback-core:1.2.3`
- `commons-codec:commons-codec:1.14`
- `commons-io:commons-io:2.6`
- `commons-net:commons-net:3.6`
- `dropwizard-jetty 2.0.7`
- `log4j-api:2.13.1`
- `log4j-core:2.13.1`
- `org.antlr:antlr4-runtime:4.8-1`
- `org.apache.httpcomponents:httpclient:4.5.12`
- `org.bouncycastle:bcprov-jdk15on:1.62`
- `org.hibernate:hibernate-validator:6.1.2.Final`
- `org.javassist:javassist:3.21.0-GA`
- `org.jboss.logging:jboss-logging:3.4.1.Final`

Version 20.3.2 – March 27, 2020

The Machine Agent has been updated, tested and certified to work with the following:

- okhttp 3.12.6
- log4j-api 2.13.1
- log4j-core 2.13.1
- log4j-slf4j-impl 2.13.1

log4j_log4j 1.2.17 has been removed.

Version 20.3.1-2476 – March 16, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.3.0-2468 – March 9, 2020

This release includes optimizations and small code changes.

Version 20.2.0

There was no 20.2.0 release for the Machine Agent.

.NET Agent

Version 20.12.0 – December 18, 2020

There are no enhancements in this release. For the resolved issue, see Past Resolved and Known Issues by Release.

Version 20.11.0 - November 20, 2020

.NET Agent now supports the .NET 5.0 environment.

Version 20.10.0 - October 28, 2020

.NET Agent includes these enhancements:

- .NET Agent supports Agentless Analytics on Windows. See Agentless Analytics Overview.
- .NET Agent is now supported on Windows Server 2019.
- The ASP.NET business transactions configuration feature has been removed as a preview feature in the Linux Agent. You now have the ability to perform custom backend naming for HTTP backends in the Linux Agent in your production environment.

For bug fixes, see Past Resolved and Known Issues by Release.

Version 20.9.0 – September 24, 2020

Azure Cosmos DB

The .NET Linux Agent now supports the Azure Cosmos DB through the SQL API. Both of these Azure versions are supported:

- Microsoft.Azure.DocumentDB.Core (2.x)
- Microsoft.Azure.Cosmos (3.x)

Note the following:

- The backend naming of the Azure Cosmos DB is detected as `{host}.{port}`.
- The backend properties of the `exit` type shows Database Account with `host` and `port`.
- The Snapshot properties of "Query through LINQ" call does not contain the SQL Query Statement.
- The `ConnectionString` of the Azure Cosmos DB is not shown in any of the Snapshot properties, nor in the backend properties.
- The following is *not supported*:
    - Instrumentation of `BulkExecution`
    - Sensitive data filters on Query

See Past Resolved and Known Issues by Release for details.

Version 20.8.0 – August 24, 2020

The.NET Agent now supports the transport layer security (TLS) 1.2 and proxy-related environment variables for the Micro Services Agent on Windows. See Configure the Agent Using Environment Variables for details.

Version 20.7.0 - July 23, 2020

The .NET Agent for Linux now:

- Partially supports the detection of WCF exit calls on .NET Core 3.1 and later. See .NET Backend Detection for more details.
- Supports setting `APPDYNAMICS_AGENT_UNIQUE_HOST_ID` as an environment variable.

This release also includes bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.6.1 - July 17, 2020

This release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.6.0 - July 1, 2020

- The .NET Agent for Windows now correlates business transactions with End User Monitoring by default.

- Improvements have been added to instrumentation in full-no-dependency mode. You can now instrument non-NGen assemblies when MSCorlib is loaded as NGen.
- Internal library upgrades and code improvements added to the .NET Agent for Linux.
- Improvements added to the .NET Agent to detect RabbitMQ Async entry points.

This release also includes bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.5.0 – May 20, 2020

- Sensitive information for MongoDB query expressions is now filtered out by default. You can enable raw query collections for MongoDB in Call Graph Settings.
- RabbitMQ backend detection is now compatible with RabbitMQ Client version 6. See RabbitMT Backends for .NET for more details.

Version 20.4.1 (.NET Agent for Windows) - April 29, 2020

Due to a signing issue, the 20.4.0 version of the .NET Agent for Windows has been replaced with 20.4.1. It has no feature difference with 20.4.0.

Version 20.4.0 - April 28, 2020

This release includes log improvements in the Linux profiler and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.3.1 - April 7, 2020

This release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.3.0 - March 26, 2020

This release contains the .NET Agent for Linux only. The .NET Agent for Linux now supports .NET Core 3.1.

Version 20.3.0 – March 25, 2020

This release contains the .NET Agent for Windows only.

The .NET Agent MSI installer for Windows now includes support for .NET Core applications, enabling unified management of .Net Core and full framework applications on Windows. The agent monitors .NET Core applications for IIS out of the box, if automatic IIS mode is configured.

- The MSI Installer bundles .NET Core agent with the existing .NET framework agent and creates appropriate environment variables.
- Supported environments are identical to the previous .NET Agent.
- .NET Core versions 2.0 and later are supported.
- IIS-hosted .NET Core applications are monitored similar to the .NET frameworks configured in automatic mode.
- You can configure .NET Core applications in the `config.xml` file, just as you would configure .NET framework applications.

You have to manually configure .NET Core applications in the `config.xml` file in either of the following scenarios:

- You monitor ASP .NET core applications that are not hosted in process, but are hosted in `dotnet.exe`
- You currently monitor standalone applications

The .NET Agent for Windows supports Azure Functions 3.x and the following:

- Supports Netviz entry calls for non-IIS applications.
- Filters sensitive data in environment variables.
- Reports CoreCLR runtime to the Controller.
- Includes WCF entry points with the agent's network visibility support, if the WCF service is operating in ASP.NET compatibility mode. If the WCF service is not using ASP.NET compatibility mode, or if the service receives requests that do not use this mode, the agent logs a warning that network visibility is currently not supported for this setup.

Version 20.2.0

There was no 20.2.0 release for the .NET Agent.

Network Agent

Version 20.12.0

There was no 20.12.0 release for the Network Agent.

Version 20.11.0

There was no 20.11.0 release for the Network Agent.

Version 20.10.0

There was no 20.10.0 release for the Network Agent.

Version 20.9.0

There was no 20.9.0 release for the Network Agent.

Version 20.8.0.2169 – August 26, 2020

This release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.7.0

There was no 20.7.0 release for the Network Agent.

Version 20.6.0 – June 25, 2020

This release includes optimizations and small code changes.

Version 20.5.0 – May 21, 2020

This release includes optimizations and small code changes.

Version 20.4.0

There was no 20.4.0 release for the Network Agent.

Version 20.3.0-2139 – March 9, 2020

This release includes optimizations and small code changes.

Version 20.2.0

There was no 20.2.0 release for the Network Agent.

Node.js Agent

Version 20.12.0 – December 8, 2020

- You can attach existing Analytics data generated for any given transaction to an exit call. For more information, see Node.js Agent API Reference.
- Docker init container images are now available on Dockerhub for Linux and Alpine environments. For all other environments, see Build the Node.js Agent Init Container Image.

Version 20.11.0

There was no 20.11.0 release for the Node.js Agent.

Version 20.10.1 – October 16, 2020

This release includes optimizations and small code changes.

Version 20.10.0 – October 15, 2020

This release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.9.0

There was no 20.9.0 release for the Node.js Agent.

Version 20.8.0-2169 – August 27, 2020

You can now separate Business Transactions into multiple queries with GraphQL. For more information, see Custom Match Rules.

Version 20.7.0 – July 29, 2020

This release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.6.0 – June 12, 2020

This release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.5.0 – May 29, 2020

This release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.4.0 – April 29, 2020

This release includes optimizations and small code changes.

Version 20.3.1 – March 27, 2020

This release includes optimizations and small code changes.

Version 20.3.0 – March 26, 2020

This release includes optimizations and small code changes.

Version 20.2.0

There was no release for the Node.js Agent.

Node.js Serverless Tracer

Version 20.12.0

There was no 20.12.0 release for the Node.js Serverless Tracer.

Version 20.11.242

The Node.js Serverless Tracer has added an environment variable, `APPDYNAMICS_HTTP_TIMEOUT_MS`, which allows you to set the timeout in milliseconds for all the HTTP calls made by the tracer. See Past Resolved and Known Issues by Release for details.

Version 20.10.0

There was no 20.10.0 release for the Node.js Serverless Tracer.

Version 20.9.221 – September, 2020

This release includes minor enhancements and bug fixes.

Version 20.8.0

There was no 20.8.0 release for the Node.js Serverless Tracer.

Version 20.7.0

There was no 20.7.0 release for the Node.js Serverless Tracer.

Version 20.6.0

There was no 20.6.0 release for the Node.js Serverless Tracer.

Version 20.5.0

There was no 20.5.0 release for the Node.js Serverless Tracer.

Version 20.4.0

There was no 20.4.0 release for the Node.js Serverless Tracer.

Version 20.3.178 – March 16, 2020

The tracer's version is now compatible with calendar versioning. This release includes no new enhancements or resolved issues.

Version 20.2.0

There was no 20.2.0 release for the Node.js Serverless Tracer.

PHP Agent

Version 20.12.0 – December 16, 2020

This release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.11.0 - November 24, 2020

This release supports the Node Reuse feature. For more information, see Node Reuse for PHP Agent.

Version 20.10.0 – October 22, 2020

This release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.9.0 – September 15, 2020

The release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release.

Version 20.8.0.4083 – August 26, 2020

PHP Agent released with bug fixes. See Past Resolved and Known Issues by Release.

Version 20.7.0 – July 16, 2020

The PHP Agent 20.7.0 release enhances the General Data Protection Regulation (GDPR) compliance for data filtering.

This helps you to redact any sensitive data (URLs, Exit calls, or cookies), which is shown in the Controller. For more information, see Redaction of URL Segments and Query Parameters.

Version 20.6.0 – June 23, 2020

The PHP Agent 20.6.0 release implements the General Data Protection Regulation (GDPR) compliance for data filtering.

This helps you to redact any sensitive data (URLs, Exit calls, or cookies), which is shown in the Controller. For more information, see Redaction of URL Segments and Query Parameters.

Version 20.5.0 – May 29, 2020

This release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.4.0.3759 – April 29, 2020

This release adds support to CakePHP 4.0 and Magento 2.3 for the PHP Agent. See PHP Supported Environments.

Version 20.3.1 – March 26, 2020

This release supports PHP Agent for CentOS version 8 and adds support to the following frameworks:

- FuelPHP 1.8.1
- Zend 3
- Laravel 6
- CodeIgnitor4

Version 20.3.0 – March 3, 2020

- PHP 7.4 support is added to the AppDynamics PHP Agent. See PHP Supported Environments.
- The PHP Agent now supports Symfony versions 3 and 4. See PHP Supported Environments.

Version 20.2.0

There was no 20.2.0 release for the PHP Agent.

Python Agent

Version 20.12.0 – December 18, 2020

This release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.11.411

The Python Serverless Tracer has added an environment variable, `APPDYNAMICS_HTTP_TIMEOUT_MS`, which allows you to set the timeout in milliseconds for all the HTTP calls made by the tracer. See Past Resolved and Known Issues by Release for details.

Version 20.11.0 - November 25, 2020

The Python Agent has released with minor fixes and optimizations. See Past Resolved and Known Issues by Release.

Version 20.10.0 - October 21, 2020

- This release supports Node Reuse feature.
- This release supports Tornado 6 for Python Agent 3.7 and later versions.

For bug fixes, see Past Resolved and Known Issues by Release.

Version 20.9.0

This release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.8.0

There was no 20.8 release for the Python Agent.

Version 20.8.0

There was no 20.8 release for the Python Agent.

Version 20.7.0.2292 – July 7, 2020

Python Agent artifacts can now be downloaded from the AppDynamics Downloads Portal. For more information, see Install the Python Agent.

Deprecation Notice

Support for Mac OS X has been removed for Python Agent.

Version 20.6.0

There was no 20.6.0 release for the Python Agent.

Version 20.5.0

There was no 20.5.0 release for the Python Agent.

Version 20.4.0

There was no 20.4.0 release for the Python Agent.

Version 20.3.0

There was no 20.3.0 release for the Python Agent.

Version 20.2.0

There was no 20.2.0 release for the Python Agent.

Python Serverless Tracer

Version 20.12.0

There was no 20.12.0 release.

Version 20.11.411

The Python Serverless Tracer has added an environment variable, `APPDYNAMICS_HTTP_TIMEOUT_MS`, which allows you to set the timeout in milliseconds for all the HTTP calls made by the tracer. See Past Resolved and Known Issues by Release for details.

Version 20.10.0

There was no 20.10.0 release for the Python Serverless Tracer.

Version 20.9.0

There was no 20.9.0 release for the Python Serverless Tracer.

Version 20.8.0

There was no 20.8.0 release for the Python Serverless Tracer.

Version 20.7.383 - July 29, 2020

The Python Serverless Tracer is now generally available to monitor AWS Lambda functions written in Python. The tracer is available on the PyPi repository. See Python Serverless Tracer to get started.

React Native Agent

Version 20.12.0

There was no 20.12.0 release.

Version 20.11.0 - November 20, 2020

The React Native Agent has released with minor fixes and optimizations.

Version 20.10.0 – October 16, 2020

This release includes minor enhancements and bug fixes. React Native 0.60 is now supported.

Version 20.9.0

There was no 20.9.0 release for the React Native Agent.

Version 20.8.0

There was no 20.8.0 release for the React Native Agent.

Version 20.7.0 – July 8, 2020

React Native Agent 20.7.0 captures `hybridAgentType` and `hybridAgentVersion`.

Version 20.6.0

There was no 20.6.0 release for the React Native Agent.

Version 20.5.0

There was no 20.5.0 release for the React Native Agent.

Version 20.4.0

There was no 20.4.0 release for the React Native Agent.

Version 20.3.0

There was no 20.3.0 release for the React Native Agent.

Version 20.2.0

There was no 20.2.0 release for the React Native Agent.

SAP

Version 20.12.0

There was no 20.12.0 release.

Version 20.11.0 - November 23, 2020

SAP Agent has released. See Release Notes Version 20.11.0.

Version 20.10.0

There was no 20.10.0 release for SAP.

Version 20.9.0

There was no 20.9.0 release for SAP.

Version 20.8.0

SAP 20.8.0 is released. See Release Notes Version 20.8.0.

Version 20.7.0

SAP 20.7.0 is released. See Release Notes Version 20.7.0.

Version 20.6.0

There was no 20.6.0 release for SAP.

Version 20.5.0

SAP 20.5.0 is released. See Release Notes Version 20.5.0.

Version 20.4.0

There was no 20.4.0 release for SAP

Version 20.3.0

There was no 20.3.0 release for SAP.

Version 20.2.0

There was no 20.2.0 release SAP.

Synthetic Agent (Hosted Agents)

Version 20.6.0 - June 25, 2020

Synthetic monitoring: ‘os’, ‘socket’, and, ‘subprocess’ modules will not be allowed for any new job. See Import Python Packages in Synthetic Scripts for details.

Xamarin Agent

Version 20.12.0

There was no 20.12.0 release.

Version 20.11.0 - November 10, 2020

The Xamarin Agent has released with minor fixes and optimizations.

Version 20.10.0

There was no 20.10.0 release for the Xamarin Agent.

Version 20.9.0

There was no 20.9.0 release for the Xamarin Agent.

Version 20.8.0

There was no 20.8.0 release for the Xamarin Agent.

Version 20.7.0

There was no 20.7.0 release for the Xamarin Agent.

Version 20.6.0

There was no release for the Xamarin Agent.

Version 20.5.0

There was no release for the Xamarin Agent.

Version 2020.4.0 – April 6, 2020

You can set the boolean property `EnableAggregateExceptionHandling` to `true` to configure the Xamarin Agent to report aggregate exceptions as crashes. When the property is set to `false`, only unhandled exceptions are reported. The default value is `false`.

Version 2020.3.3 – March 16, 2020

This release includes optimizations and small code changes.

Version 2020.3.2 – March 13, 2020

By default, the Xamarin Agent reports exceptions as crashes. You can now set a configuration to disable the reporting of crashes as exceptions.

Version 2020.3.1 – March 10, 2020

This release includes optimizations and small code changes.

Version 2020.2.0

There was no 20.2.0 release for the Xamarin Agent.

Past Controller Releases

The Controller releases occur every six weeks. This page lists the past SaaS and on-premises Controller enhancements since 20.2.

21.5 Release

License Management

For infrastructure-based licenses in Controller 21.5.0:

- The license summary drill-down dashboard allows you to filter usage data by time and view a list of agents connected to a license rule.
- When looking at license usage and license rules, you can view and sort applications with analytics enabled.

Database Visibility

The Controller audit report can fetch the audit log for the Remove literal flag. See Controller Audit Log and Configure Query Literals Security.

The user interface is updated. You can:

- Return to **Queries** page from the **Query Details** page. See Database Query Details Window.
- Check the number of database nodes for each collector on the **Databases** page. The number of nodes is included in both card and list view. See View Overall Database and Server Performance.
- Navigate between different nodes of the cluster on the database **Dashboard**. See Database Dashboard.

21.4 Release

Account Management Enhancements

AppDynamics has enhanced user management and security for customers choosing to use AppDynamics to authenticate their users.

Enhanced Security

- You can now have a single user identity for all your SaaS Tenants and across your entire SaaS environment.
- You can have a single identity extended to the Account Management Portal.
- All first user accounts authorize using a validated email.
- Administrators can no longer create or update user passwords. Users now create and update their own passwords using an updated password security policy.

Enhanced User Management

Administrators can now:

- Use a single interface to centrally manage all their users regardless if they have rights on the Tenants themselves.
- Delete users that have an inactive status.
- View Tenants and licenses associated with users.

Experience Journey Map

- (SaaS only) You can create custom views to filter the Experience Journey Map by specified user journeys. See Create a Custom View.
- You can filter the Experience Journey Map by custom user data. See Using the Custom User Data Filter.

Database Visibility Enhancement

- You can now monitor the MS SQL cluster instances. See Configure Microsoft SQL Server Collectors.
- You can now use the Kerberos authentication for the DB2 and Oracle Collector. See Configure IBM DB2 Collectors and Configure Oracle Collectors.

License Management

- When viewing license usage, you can customize the time range on the **License Summary** dashboard.

ThousandEyes Integration with AppDynamics

AppDynamics and ThousandEyes integration delivers a full-stack solution, which enables you to identify network-related performance issues to reduce the impact on critical applications. See ThousandEyes Integration with AppDynamics.

21.3 Release

Dash Studio (21.3.1)

Simple Navigations

You can now navigate from one dashboard to the other using **Single Click Action** option. See Dash Studio Widgets.

Dash Studio (21.3.0)

You can now set performance baselines in a Time Series widget by adding a **Threshold** value. See Dash Studio Widgets.

Cisco Secure Application

AppDynamics introduces Cisco Secure Application to monitor the security of APM-managed applications. The data from the managed application is scanned for any security issues and displayed in a real-time dashboard. Secure Application is bundled within the Java Agent and is available from the 21.3.0 release onwards with the purchase of an additional SKU.

21.2 Release

Agent Installer

- Machine Agent is supported in `non-sudo` installations.
- Agent Installer supports the Infrastructure Agent.
- A new `watchdog` process is included with the Agent Installer Platform installation to monitor and ensure that agents are running properly.
- When installing the Agent Installer Platform with `systemd`, a profile is created for the `watchdog` process instead of for individual agents.
- `bash` profiles for `non-sudo` installations are no longer created.

Analytics

- You can add alternative milestone events of the same event type.
- You can configure agent-side metrics in the Metric Browser. For more information, see Troubleshoot Analytics Issues.

Cloud Native Visualization

- Cloud Native Visualization introduces options to monitor the health and performance of entities. You can create and configure health rules to monitor the health of the entities. You can use anomaly detection to automatically detect anomalies with entities. This helps initiate remediation actions quickly. See Configure Entity Health
- You can now monitor the metrics of AWS instances with both basic (5 minute granularity) and detailed (1 minute granularity) monitoring. See Cloud Native Visualization Metrics Overview

Database Monitoring

- The top query statistics report for Oracle database now displays only the following statistic type data:
  - Buffer Gets
  - Cpu Time
  - Disk Reads
  - Elapsed Time
  - Executions
  - Parse Calls
  - Rows Processed

  See Database Activity Window
- You can use the **SRV Record** option for MongoDB. See Configure MongoDB Collectors
- Additional Couchbase metrics are added for the database monitoring. See **Couchbase Metrics** at Database Monitoring Metrics

End User Monitoring

Experience Journey Map

- You can customize how you analyze user journeys.
- You can customize naming of mobile views/activities with the `SessionFrame` API.
- You can filter by Internet Service Provider (ISP) for browser apps.

License Management

Controller 21.2.0 includes a new UI for managing infrastructure-based licenses.

Dash Studio (21.2.0)

Gauge Widget

You can now present metric data on a relative scale using the Gauge widget. See Dash Studio Widgets.

Time Range Comparison

This feature is now available for Metric Number widget. See Time Range Comparisons.

21.1 Release

Dash Studio (21.1.0)

Grouped Widgets

You can now group two or more widgets on a single card or on their respective cards. See Dash Studio Widgets.

Label Widget

You can now apply these properties to modify the appearance of the Label widget:

- Font Color
- Font Size
- Horizontal Alignment
- Vertical Alignment

See Dash Studio Widgets.

20.12 Release

Dash Studio (uic-20.12.1)

Auto Refresh Interval

You can now enable the **Auto Refresh Interval** option to automatically refresh the dashboard at regular intervals. For more information, see Visual Dashboard Editor.

Group Time Range

You can now set the time range for a set of grouped widgets. For more information, see Dash Studio Widgets.

Cloud Native Visualization (uic-20.12.0)

You can now monitor the health status of entities in Cloud Native Visualization. Entities in the Relationships map, Interactions map, and the List view are grouped into healthy and unhealthy categories based on their health status. To initiate remediation actions quickly, you can click the unhealthy entity to drill down and determine the affected business transactions, services, and service instances to resolve any critical issues. For more information, see Monitor the Health of Entities.

20.11 Release

Agent Installer

Monitoring Settings

- The content of the Monitoring Settings page has been reorganized into three tabs: **Managed Processes**, **Unmanaged Processes**, and **Uninstrumented Processes**.
- The ability to paginate has been added to the Monitoring Settings page eliminating the previous 1000 record limitation.

End User Monitoring

Mobile Real User Monitoring

You can filter performance data, such as network requests, crashes and code issues, based on device specifications, such as CPU type, memory, disk space, and battery. See Mobile RUM Metrics for details.

Dash Studio

Mobile Applications Metrics

Mobile Applications metrics in the **Data** panel of the Dash Studio allows you to filter and create customized metrics to view the performance of mobile applications. See Data Binding.

Browser Applications Metrics

Browser Applications metrics in the **Data** panel of the Dash Studio allows you to filter and create customized metrics to view the performance of the web applications. See Data Binding.

Simple Navigations

Simple navigation actions in Dash Studio allow you to drill down into complex and detailed information about a selected entity. See Dash Studio Widgets.

Database Visibility

You can now integrate Database Visibility with Server Visibility. See Integrate Database Visibility with Server Visibility.

There are new Oracle Server metrics available with this Controller release. See Database Monitoring Metrics.

Service Endpoints

The Service Endpoints page includes these enhancements:

- A new grid display is available that includes a total count of entries at the bottom right of the grid. This information was previously located below the Search text box.
- The top toolbar on the page includes buttons for: **Details**,  (Delete), **Exclude**, **View Excluded**, **Configure**, and **Filter** (replacing the previous Actions sub-menu that included these buttons).
- When you select the Filter icon, the **Filter** menu panel is overlaid on top of the grid on the left side of the page and includes additional filter options.

20.10 Release

Agent Installer

Improvements to the **Agent Installer** page have been implemented to improve the customer experience:

- Overall installation process has been streamlined to reduce manual input
- Two installation options are available:
  - **Express** - for new installations
  - **Custom** - provides options you can customize

APM Platform

The Automated Transaction Diagnostics (ATD) capability is now available for SaaS. This feature exposes periods of anomalous behavior in your business transactions and pinpoints the causal factor to facilitate a more efficient root cause analysis process.

Using the already collected transaction snapshots, the diagnostic process:

- Analyzes business transaction response time data to determine if there was an issue and the time period impacted.
- Analyzes metrics data across the business transaction path to identify the suspected cause and type of issue.
- Provides a guided root cause analysis that diagnoses the business transaction anomaly allowing you to manage the issue before it becomes visible to customers.

Cloud Native Visualization

AppDynamics new Cloud Native Visualization services are generally available today. Cloud Native Visualization provides new streamlined views of your application landscape enabling you to make informed decisions. By correlating application metrics and interactions to cloud infrastructure, Cloud Native Visualization enables you to explore deployments, and examine the cloud infrastructure impact on application performance.

> As part of the launch program, early adopters have access to the Cloud Native Visualization functionality for six months free of charge. After six months, a purchase of new licenses is required to continue using the functionality. AppDynamics reserves the right to charge separately for Cloud Native Visualization after the end of the six month period.

Cloud Native Visualization is available for versions:

- SaaS Controller >= 20.10.0
- Java Agent >= 20.9.0
- .NET Agent >= 20.10.0
- Node.js Agent >= 20.10.0

Note: Java Agent, .NET Agent, and Node.js Agent are required for Amazon EC2 to APM node correlation.

Cloud Native Visualization requires:

- Applications to be deployed in Amazon Web Services (AWS).

- Configuration changes in the AppDynamics SaaS platform to be performed by AppDynamics.

> Controller configuration is being rolled out gradually to SaaS customers beginning on October 28, 2020.

Cloud Native Visualization Benefits:

- Correlates APM data fully with cloud platform data to help you make informed decisions.
- Ingests cloud platform metrics and metadata automatically using the AIOps platform, thereby introducing new holistic views of your cloud application landscape.
- Displays application services (formerly known as tiers), and related infrastructure entities such as load balancers, databases, and business transactions, in a single unified view.

Cloud Native Visualization provides full-stack correlation with these AWS services:

- Amazon Elastic Compute Cloud (EC2)
- Amazon Elastic Block Store (EBS)
- Amazon Relational Database Service (RDS)
- Amazon Aurora
- Amazon Elastic Load Balancers (ELB) including NLBs, ALBs and CLBs
- Virtual Private Cloud (VPC) metadata

End User Monitoring

- Experience Journey Map can be filtered in the Controller UI by fields such as device, OS, and location.
- Automatic business transaction correlation for .Net Agent 20.6.0 and higher is enabled by default. You can also manually enable and disable business transaction correlation with older APM agents or applications with frameworks other than .Net and Java. See Correlate Business Transactions for EUM Monitoring for more information.

Browser Real User Monitoring

- When configuring the JavaScript Agent in the Controller, you can set a custom page title to an arbitrary string. See Configure the JavaScript Agent .

Mobile Real User Monitoring

- You can filter performance data, such as network requests, crashes and code issues, based on device specifications, such as CPU type, memory, disk space, and battery. See Mobile RUM Metrics for details.

20.9 Release

There was no 20.9.0 Controller release.

20.8 Release

Alert and Respond

Action Suppression

You can now suppress a policy's automatic invocation of actions and alerts on a recurring basis. For example, if you have scheduled maintenance every month and would not like to be notified of any event violations or trigger any actions for these events during the maintenance period, you can schedule to suppress the alerts and actions. For more information, see Configure and Manage Action Suppressions. For corresponding API updates, see Action Suppression API.

HTTP/HTTPS Proxy for On-premise Controller

You can now configure an HTTP/HTTPS proxy for an on-premise Controller. When you configure a proxy, the HTTP request actions or JIRA actions are routed through the HTTP proxy if the endpoint is an HTTP endpoint or HTTPS proxy if the endpoint is an HTTPS endpoint. You can configure the HTTP /HTTPS proxy settings in the `domain.xml` global configuration file. For more information, see Configure HTTP/HTTPS Proxy.

> You can configure an HTTP/HTTPS proxy for an on-premise Controller only.

Database Monitoring

- You can now view the Database Agent telemetry on the **Metric Browser** window. See Database Agent Telemetry for details.
- For Oracle database, you can view the procedure for queries and view the container where the query is running. Also, you can view the CPU and Memory metrics without enabling hardware monitoring. For more information about containers and procedures, see Database Containers Window and Database Procedures Window respectively.
  To view these you must be using Database Agent 20.6 or later.

- You can view the Database Agent related logs for debugging and resolving issues. For more information about the agent logs, see View Database Agent Properties and Logs.

Cluster Monitoring

You can now use filters to view the required event details from the list of cluster events. See Monitor Kubernetes Events.

End User Monitoring

- You can customize Experience Journey Map by toggling performance markers on or off, and by setting values for performance thresholds and drop-off rates. See Configure Experience Journey Map for details.
- You can filter mobile crashes based on the following agent version fields: agent version, hybrid agent version, and hybrid agent type.
- All Synthetic Agent IBM SoftLayer locations have been migrated to Azure and AWS. Agents in Dallas, Texas, Houston, TX, and Quéretaro, Mexico are no longer available and have been replaced with San Antonio, TX with Azure. See Synthetic Agent Locations for details.
- Browser Synthetic Monitoring has transitioned all hosted Synthetic Agent IBM locations to Azure and AWS. See Synthetic Agent Locations for an up-to-date list of agent locations and providers.

Transaction Detection for Node.js Applications

GraphQL Business Transactions, a feature that enables transaction detection on the Node.js Agent, is now available. You can define a custom match rule that separates a business transaction into several GraphQL queries with GraphQL Business Transactions. See Node.js Business Transaction Detection.

20.7 Release

Alert and Respond

Alerting Template

You can now export a configured alerting template and save it for later use. You can import a saved template and apply it to applications. For more information, see Configure and Manage Alerting Templates.

Alert Sensitivity Tuning

You can now define a metric expression to evaluate a health rule condition and use Alert Sensitivity Tuning to fine-tune the configuration. For more information, see Create a Health Rule and Fine-tune Metric Evaluation.

Application Analytics

The End User Monitoring (EUM) filter has been added to the Analytics Home page with options to display browser and mobile sessions. See Overview of Application Analytics.

End User Monitoring

- Experience Journey Map is supported for on-prem Controller 20.7.0 and later and requires a EUM PEAK license.
- You can now custom configure the JavaScript Agent in the Controller UI. This UI-based configuration is only available for SaaS Controller 20.7.0 and later. See Configure the JavaScript Agent.

20.6 Release

Agent Installer

The Agent Installer requires microservices configuration performed by AppDynamics. The configurations will be rolled out gradually to SaaS Controllers beginning on June 8, 2020.

The Agent Installer is now available. The Agent Installer simplifies the installation process for the Java and Machine Agents. The installer provides two lines of script that:

- deploy the Java and (optionally) Machine Agents,
- automatically instrument applications, and
- assign unique names to tiers and nodes

The Agent Installer is available for SaaS deployments and Linux operating systems. See Agent Installer documentation for more information.

Analytics Events Service (SaaS only)

The `eventcompletionTimestamp` field has been introduced for ADQL queries. `eventcompletionTimestamp` indicates the time that the event was received by the Events Services. See Analytics Events API for details.

The metric processor will be rolled out gradually to SaaS Controllers. Contact AppDynamics Support for more information.

Application Analytics

Recommended Data Collectors, a feature that simplifies data collector configuration, is now available for SaaS deployments. Instead of manually configuring data collectors, you can browse and search for event data displayed in the Controller that Recommended Data Collectors automatically collects from the Java Agent.

Cluster Monitoring

The **Cluster Event** summary dialog is enhanced with the following additional details:

- The **Affects** and the **Cluster Agent Version** fields are added to provide information about the affected entity and the version of the Cluster Agent respectively.
- The **Cluster Name** field displays the name of the cluster along with an option to drill down. Therefore, you can launch the Cluster Agent dashboard by clicking the name.

End User Monitoring

Experience Journey Map visualizes real-time user journeys, providing performance metrics and traffic data for browser and mobile applications. This feature is only available for customers with a SaaS Controller and EUM PEAK license.

Service Proxy

A new service proxy feature (represented by this icon ) has been added to enable the system to discover the backend (remote service) as a service proxy, and display the correct component structure in a flow map. The service proxy feature is enabled in the Controller. Once the Controller has identified the backend as a service proxy, it sends the configuration information to an Agent (Java Agent version 20.5.0 and later, is the only Agent supported with this release).

20.5 Release

The Controller release occurs every six weeks. There was no Controller release for 20.5. For bug fixes, see Past Resolved and Unknown Issues by Release.

20.4 Release

Alert and Respond

Alerting Template

AppDynamics 20.4 introduces 'Alerting Template' that allows you to efficiently maintain and manage alerting configuration at scale. An Alerting Template is a configuration of features like Health Rules, Policies, Actions, Action Suppression, Schedules, and Email digests. Templates eliminate the need to configure or update individual applications. You can easily update a template and ensure that updates are consistently applied across multiple applications. For more information, see:

- Alert and Respond
- Alerting Templates
- Configure Alerting Templates
- Alerting Template Questions and Answers

Action Suppression and Email Digest APIs

AppDynamics introduces REST APIs to perform CRUD operations for action suppression and policy email digest. For more details, see:

- Action Suppression API
- Email Digest API

Alert Sensitivity Tuning

AppDynamics introduces 'Alert Sensitivity Tuning (AST) that helps you visualize the impact of the alerting configuration you define. AST provides historical data for the metric or the baseline being configured. Instant insights are provided in a graphical format when you configure the alerts. Based on this data, you can set up alerts with the appropriate sensitivity and avoid false alerts. For more information, see:

- Alert Sensitivity Tuning
- Create a Health Rule and Fine-tune Metric Evaluation
- AST Questions and Answers

Database Monitoring

- The controller provides the following visual context for the backend database on the flow map:

    - The database icon with green mark shows that the status of the database is healthy. If there are any health rule violations, the icon changes accordingly.
    - If there are any disconnected backend database servers, a message is displayed to connect to the appropriate server or cluster in Database Visibility.
- The Metric browser now displays the buffer and the cache memory metrics for a Linux collector.
- The controller now supports monitoring Apache Cassandra.

End User Monitoring

- Business transaction correlation is enabled by default for business applications instrumented with the Java Agent 4.5.0 or higher.
- Admin-level users can configure read-only user permissions for the Synthetic Credential Vault.
- Python 3 is supported in Synthetic Scripts to create Synthetic Jobs.
- AppDynamics offers a unified Real User Monitoring license that can be allocated to both Browser and Mobile Real User Monitoring usage.

Known Issues

- License service makes individual accounts' usage and term calls to the Events Service.

SAML Authentication

The AppDynamics Controller now supports HTTP GET as well as POST for the authentication request to the identity provider for the sign-out message to the identity provider.

20.3 Release

Alert and Respond

AppDynamics introduces REST APIs to perform CRUD operations for various actions that are triggered as a response to events. For more details, see Actions API.

Anomaly Detection

AppDynamics root cause analysis (RCA) algorithms have been refined considerably to determine and isolate faulty components while reducing false positives. This new algorithm greatly improves the time taken to fetch metrics and hence RCA response for any anomaly event. For more information, see Troubleshooting Anomalies.

My Preferences

Time Zone

The time zone controller setting helps adjust the time zone enabling you to collaborate and troubleshoot issues with more accuracy. This setting also eliminates issues such as translating timestamp of your location to another user's time zone. See Time Zone Controller, for more details. In the 20.3.0 release, enable the ui.platform.timezone.enabled flag for the time zone capability to work. The flag will be enabled by default for the upcoming releases.

20.2 Release

The Controller release occurs every six weeks. There is no Controller release for 20.2. For bug fixes, see Past Resolved and Unknown Issues by Release.

Past On-premises Platform Releases

This page lists all the past on-premises releases since the 20.2 release.

21.5 Release

Synthetic Private Agent (K8S Container Based Agent)

Version 21.5.0 - May 25, 2021

New Private Synthetic Agent (K8S Container Based Agent) is available now. See Private Synthetic Agent.

21.4 Release

Enterprise Console

Version 21.4.3 – April 29, 2021

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 21.4.2 – April 22, 2021

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 21.4.0 – April 5, 2021

The following enhancements have been added to the Enterprise Console:

- MySQL version 5.7.33 is now supported.
- Java Runtime Environment (JRE) version 8.0.28 is now supported.

EUM Server

Version 21.4.0 – April 5, 2021

MySQL version was upgraded to 5.7.33.

Events Service

There was no 21.4.0 release of the Events Service.

Geo Server

There was no 21.4.0 release of the Geo Server.

Synthetic Private Agent

There was no 21.4.0 release of the Synthetic Private Agent.

Synthetic Server

There was no 21.4.0 release of the Synthetic Server.

21.3 Release

There was no on-premises platform release for 20.3.

21.2 Release

Enterprise Console

Version 21.2.7 – May 3, 2021

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 21.2.6 – April 23, 2021

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 21.2.3 – March 15, 2021

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 21.2.2 – March 8, 2021

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 21.2.1 – February 24, 2021

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 21.2.0 – February 22, 2021

The following enhancements have been added to the Enterprise Console:

- MySQL version 5.6.50 and 5.7.32 is now supported.
- `Libncurses` and `ncurses` version 5 and 6 is now supported.
- Java Runtime Environment (JRE) version 8u172 is now supported.

EUM Server

There was no 21.2.0 release of the EUM Server.

Events Service

There was no 21.2.0 release of the Events Service.

Geo Server

There was no 21.2.0 release of the Geo Server.

Synthetic Private Agent

There was no 21.2.0 release of the Synthetic Private Agent.

Synthetic Server

Version 21.2.0 – February 26, 2021

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

21.1 Release

Enterprise Console

There was no 21.1.0 release of the Enterprise Console.

EUM Server

There was no 21.1.0 release of the EUM Server.

Events Service

There was no 21.1.0 release of the Events Service.

Geo Server

There was no 21.1.0 release of the Geo Server.

Synthetic Private Agent

There was no 21.1.0 release of the Synthetic Private Agent.

Synthetic Server

There was no 21.1.0 release of the Synthetic Server.

20.12 Release

Enterprise Console

There was no 20.12.0 release of the Enterprise Console.

EUM Server

There was no 20.12.0 release of the EUM Server.

Events Service

There was no 20.12.0 release of the Events Service.

The current version, 4.5.2.20640, is available as a standalone artifact or as a bundle with the Enterprise Console. This version contains hotfixes. See 4.5.x Hotfixes for details.

Geo Server

There was no 20.12.0 release of the Geo Server.

Synthetic Private Agent

There was no 20.12.0 release of the Synthetic Private Agent.

Synthetic Server

There was no 20.12.0 release of the Synthetic Server.

20.11 Release

Enterprise Console

Version 20.11.9 – February 18, 2021

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.11.8 – February 4, 2021

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.11.7 – January 29, 2021

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.11.6 – January 25, 2021

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.11.5 – December 18, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.11.4 – December 17, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.11.3 – December 8, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.11.2 – December 2, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.11.1 – November 30, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.11.0 – November 16, 2020

These additional operating systems (OS) now support the AppDynamics platform:

Linux (64 bit)	Microsoft Windows (64 bit)
• RHEL 8.x • CentOS 8.x • Ubuntu 18.x and 20.x • openSUSE Leap 15 • Amazon Linux 2	• Windows Server 2019

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

EUM Server

Version 20.11.0 – November 16, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Events Service

There is no 20.11.0 release of the Events Service.

The current version, 4.5.2.20640, is available as a standalone artifact or as a bundle with the Enterprise Console. This version contains hotfixes. See 4.5.x Hotfixes for details.

Geo Server

Version 20.11.0 – November 16, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Synthetic Private Agent

There is no 20.11.0 release of the Synthetic Private Agent.

Synthetic Server

There is no 20.11.0 release of the Synthetic Server.

20.10 Release

AppDynamics Cluster Manager

There is no 20.10 release of the AppDynamics Cluster Manager.

Enterprise Console

Version 20.10.10 – January 8, 2021

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.10.9 – December 18, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.10.8 – December 3, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.10.7 – November 13, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.10.6 – November 10, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.10.5 – November 2, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.10.4 – October 22, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.10.3 – October 19, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.10.2 – October 16, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.10.1 – October 13, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.10.0 – October 6, 2020

The Azul JRE version has been updated to 1.8.0_262 used by the Enterprise Console Installer.

EUM Server

Version 20.10.1-32458 – October 12, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.10.0 – October 6, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Events Service

There is no 20.10.0 release of the Events Service.

The current version, 4.5.2.20634, is available as a standalone artifact or as a bundle with the Enterprise Console. This version contains hotfixes. See 4.5.x Hotfixes for details.

Geo Server

Version 20.10.0 – October 6, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Synthetic Private Agent

There is no 20.10.0 release of the Synthetic Private Agent.

Synthetic Server

There is no 20.10.0 release of the Synthetic Server.

20.9 Release

AppDynamics Cluster Manager

There was no release of the AppDynamics Cluster Manager.

EUM Server

There was no 20.9.0 release for EUM Server.

Events Service

Version 20.9.0 – April 12, 2021

This release includes optimizations and bug fixes.

Synthetic Private Agent

Version 20.9.0 – September 30, 2020

You can use the `privateMode:true` flag to remediate the disk filling up issue on private agents. Private Synthetic Agents do not run the `webdriver-server` component if `privateMode:true` is set in the file `C:\appdynamics\synthetic-agent\synthetic-driver\conf\synthetic-driver.yml`. Note that the Internet Explorer (IE) needs the webdriver-server component to run, therefore setting the `privateMode` flag to true causes all IE measurements to fail.

Synthetic Server

There was no 20.9.0 release for Synthetic Server.

20.8 Release

AppDynamics Cluster Manager

There was no release of the AppDynamics Cluster Manager for 20.8.

Enterprise Console

Version 20.8.9-23271 – October 28, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.8.8-23269 – October 21, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.8.7-23267 – October 7, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.8.6-23263 – October 1, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.8.5-23257 – September 23, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.8.4-23255 – September 15, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.8.3-23251 – September 14, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.8.2-3834 – September 10, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.8.1-23241 – September 1, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.8.0-23236 – August 27, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

EUM Server

Version 20.8.0.31802 – August 27, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Events Service

There is no release of the Events Service for 20.8.

The current version, 4.5.2.20634, is available as a standalone artifact or as a bundle with the Enterprise Console. This version contains hotfixes. See 4.5.x Hotfixes for details.

Geo Server

Version 20.8.0-31802 – August 27, 2020

This release includes optimizations and bug fixes.

Synthetic Private Agent

There was no release of the Synthetic Private Agent for 20.8 yet.

Synthetic Server

There was no release of the Synthetic Server for 20.8 yet.

20.7 Release

AppDynamics Cluster Manager

There was no release of the AppDynamics Cluster Manager.

Enterprise Console

Version 20.7.6-22956 – September 29, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.7.5-22944 – September 14, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.7.4-22931 – August 13, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.7.3-22924 – August 7, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.7.2-22914 – July 24, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.7.1-22912 – July 21, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.7.0-22903 – July 16, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

EUM Server

Version 20.7.0-31650 – July 16, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Events Service

There was no release of the Events Service for July 2020.

The current version, 4.5.2.20634, is available as a standalone artifact or as a bundle with the Enterprise Console and contains hotfixes. See 4.5.x Hotfixes f or details.

Geo Server

Version 20.7.0-31650 – July 16, 2020

This release includes small code changes and optimizations.

Synthetic Private Agent

There is no release of the Synthetic Private Agent.

Synthetic Server

Version 20.7.0-31 – June 24, 2020

This release includes small code changes and optimizations. See Past Resolved and Known Issues by Release for details.

20.6 Release

AppDynamics Cluster Manager

There was no release of the AppDynamics Cluster Manager for June 2020.

Enterprise Console

Version 20.6.10-22689 – September 16, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.6.9-22674 – August 13, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.6.8-22664 – July 24, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.6.7-22661 – July 21, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.6.6-226586 – July 15, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Releasefor details.

Version 20.6.50-22646 – July 9, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.6.3-22635 – June 25, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.6.2-22624 – June 12, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.6.1-22620 – June 8, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.6.0-22615 – June 2, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

EUM Server

Version 20.6.1-31662 – June 30, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.6.0-31157 – June 2, 2020

This release includes small code changes and optimizations.

Events Service

Version 4.5.2.20602

There was no release of the Events Service for June 2020.

The current version, 4.5.2.20602, is available as a standalone artifact or as a bundle with the Enterprise Console. This version contains hotfixes. See 4.5.x Hotfixes for details.

Geo Server

Version 20.6.0-31157 – June 2, 2020

This release includes small code changes and optimizations.

Synthetic Private Agent

There was no release of the Synthetic Private Agent.

Synthetic Server

Version 20.6.0-31157 – June 2, 2020

This release includes small code changes and optimizations.

20.5 Release

AppDynamics Cluster Manager

Version 20.5.0 – May 29, 2020

AppDynamics Cluster Manager is an optional service that AppDynamics administrators and end users can install in an on-premises environment. It is an orchestrated set of software components which run on multiple server nodes. AppDynamics Cluster Manager provides on-premises customers improved platform, metrics, and database scalability.

AppDynamics Cluster Manager is compatible with Controller version 20.4.3 and later.

20.4 Release

Enterprise Console

Version 20.4.10-22377 – October 8, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.4.9-22374 – September 16, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.4.8-22353 – June 12, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.4.7-22350 – June 8, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.4.6-22344 – May 29, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.4.5.22338 – May 26, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.4.4.22335 – May 21, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.4.3-22332 – May 19, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.4.2-22301 – May 4, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.4.1-22292 – April 28, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.4.0-22285 – April 21, 2020

Azul JRE 1.8.0_242 is replacing Oracle JRE as a component included in the AppDynamics products for:

- Enterprise Console upgrade
- Controller upgrade
- Events Service upgrade

EUM Server

Version 20.4.2-31620 – June 30, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.4.1-31303 – May 13, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.4.0-31010 – April 21, 2020

The Oracle Java Runtime Environment (JRE) for the EUM Server has been replaced with Azul's Zulu OpenJDK 8 JRE.

Events Service

There was no release of the Events Service.

Synthetic Private Agent

Version 20.4.0 – April 21, 2020

- The Oracle Java Runtime Environment (JRE) bundled with the Synthetic Private Agent has been replaced with the OpenJDK 8 JRE.
- Support is now available for Python 3.

Synthetic Server

Version 20.4.0-30998 – April 21, 2020

This release includes optimizations and small code changes.

20.3 Release

Enterprise Console

Version 20.3.13-22001 – September 16, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.3.12-21986 – June 29, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.3.11-21983 – June 24, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.3.10-21980 – June 22, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.3.9-21977 – June 10, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.3.8-21967 – May 21, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.3.7-21963 – May 20, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.3.6-21932 – April 20, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.3.5-21927 – April 13, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.3.4-21917 – April 7, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.3.3-21908 – April 2, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.3.2-21899 – March 27, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.3.1-21882 – March 18, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.3.0 – March 9, 2020

The HA Controller Upgrade Wizard is now automatically enabled when you choose to upgrade your Controllers.

EUM Server

Version 20.3.1-31617 – June 30, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Version 20.3.0-30600 – March 9, 2020

This release includes optimizations and small code changes.

Events Service

There was no release of the Events Service.

Synthetic Private Agent

Version 20.3.0 – March 17, 2020

This release includes optimizations and bug fixes. See Past Resolved and Known Issues by Release for details.

Synthetic Server

Version 20.3.0-30600 – March 9, 2020

This release includes no new features, but bug fixes. See Past Resolved and Known Issues by Release for details.

20.2 Release

Enterprise Console

There was no release of the Enterprise Console.

EUM Server

There was no release of the EUM Server.

Events Service

There was no release of the Events Service.

Synthetic Private Agent

Version 20.2.1560 – February 11, 2020

- Synthetic Private Agent now supports Windows Server 2016.
- Synthetic Private Agent Installer has been upgraded to support regional clouds.

Synthetic Server

There was no release of the Synthetic Server.

Glossary

Familiarize yourself with a list of common terms you may come across as an AppDynamics user.

An automatic response to an event based on a policy.

Examples include sending alerts, taking diagnostic snapshots, remediation through scripts, or making a REST API call to integrate with other tools. Actions are customizable.
Related Pages

- Actions
- Alert and Respond

Code that collects and reports data.

Agents can refer to languages (Java, Node.js, Python, etc), machines (hardware and network data), and databases, etc.
Related Pages:

- Install App Server Agents

Email, SMS, or customized external notification interface that notifies you of a problem or event.
Related Pages:

- Alert and Respond

A named collection of tiers representing a monitored environment. Also referred to as Business Application.

See also: Tier, Node
Related Pages:

- Business Applications

Databases, remote services, and any detected out of process components that are involved in business transaction processing.
Related Pages:

- Backend Detection Rules

An action of your application. Business transactions represent your critical business functions.

Examples include login, browse, and checkout.
Related Pages:

- Business Transactions

Collects, stores, analyzes, and baselines performance data collected by agents.
Related Pages:

- Getting Started
- Controller UI Overview

Enables you to collect supplemental application data for business transaction snapshots and/or analytics.

There are three types of data collectors: Method Invocation (MIDC), HTTP, and SQL.
Related Pages:

- Data Collectors

Any object that AppDynamics monitors. Entities typically have associated metrics, events, and a health status.

Examples of entities are applications, tiers, nodes, and business transactions.
Related Pages:

- Overview of Application Monitoring

Unstructured data that represents an action or occurrence detected by the agent.

Examples include JVM restart, deadlock, and error.
Related Pages:

- Monitor Events

Graphical view of information flow within a business application.
Related Pages:

- Flow Maps

Allows you to select specific metrics as key to the overall health of an application and to define ranges for acceptable performance of those metrics.

You can customize default health rules or create new ones.
Related Pages:

- Health Rules
- Alert and Respond

Instruments a method in application code outside the context of any business transaction.

Used for monitoring the performance of the method itself, or for capturing numerical data from the method's input parameters or return value.
Related Pages:

- Information Points

Numerical measurement over a time interval. A data point is a numerical value that an agent measures and reports.
Examples include "calls per minute" and "average response time."
Related Pages:

- Metrics and Graphs

The smallest unit of the AppDynamics modeling environment.

See also: Application, Tier
Related Pages:

- Tiers and Nodes

Provides a mechanism to automate monitoring, alerting, and problem remediation.

Consists of:

1. A trigger based on events
2. An action in response to the trigger

Related Pages:

- Policies
- Alert and Respond

Provides basic metrics for one application service, or group of services, provided by a tier.
Related Pages:

- Service Endpoints

A named collection of nodes, typically representing a service.

See also: Application, Node
Related Pages:

- Tiers and Nodes

A collection of detailed diagnostic information captured during a single invocation of a business transaction.
Examples include call graphs and timing during a single user login, where login is the business transaction.
Related Pages:

- Call Graphs
- Diagnostic Sessions
- Browser Snapshots
- Network Request Snapshots