



APPDYNAMICS

# AppDynamics for Node.js Beta

AppDynamics Pro Documentation

Version 3.8.x

1. AppDynamics for Node.js Beta	3
1.1 Requirements for Node.js Beta	3
1.2 AppDynamics for Node.js Architecture	3
1.3 Supported Environments and Versions for Node.js	4
1.4 Install the App Agent for Node.js	5
1.5 Uninstall the App Agent for Node.js	8
1.6 Configure AppDynamics for Node.js	8
1.6.1 Configure Transaction Detection for Node.js	8
1.6.2 Configure Error Detection for Node.js	12
1.7 Monitor Node.js Applications	13
1.7.1 Monitor Node.js Business Transactions	14
1.7.2 Monitor Node.js Processes	14
1.7.3 Monitor Node.js Backends	19
1.8 Troubleshoot Node.js Application Problems	20
1.8.1 Troubleshoot Slow Response Times for Node.js	20

# AppDynamics for Node.js Beta

This information covers using and configuring the Agent for Node.js.

For general information about AppDynamics see [AppDynamics Essentials](#) and [AppDynamics Features](#).

---

## Monitor Node.js Applications

---

## Troubleshoot Node.js Application Problems

---

## Install AppDynamics for Node.js

---

## Configure AppDynamics for Node.js

---

## Supported Environments and Versions for Node.js

---

## Requirements for Node.js Beta

The Node.js agent requires a 3.8 AppDynamics on-premise Controller. See [Controller System Requirements](#) for information on sizing your Controller.

You can download the controller from [the AppDynamics download site](#).

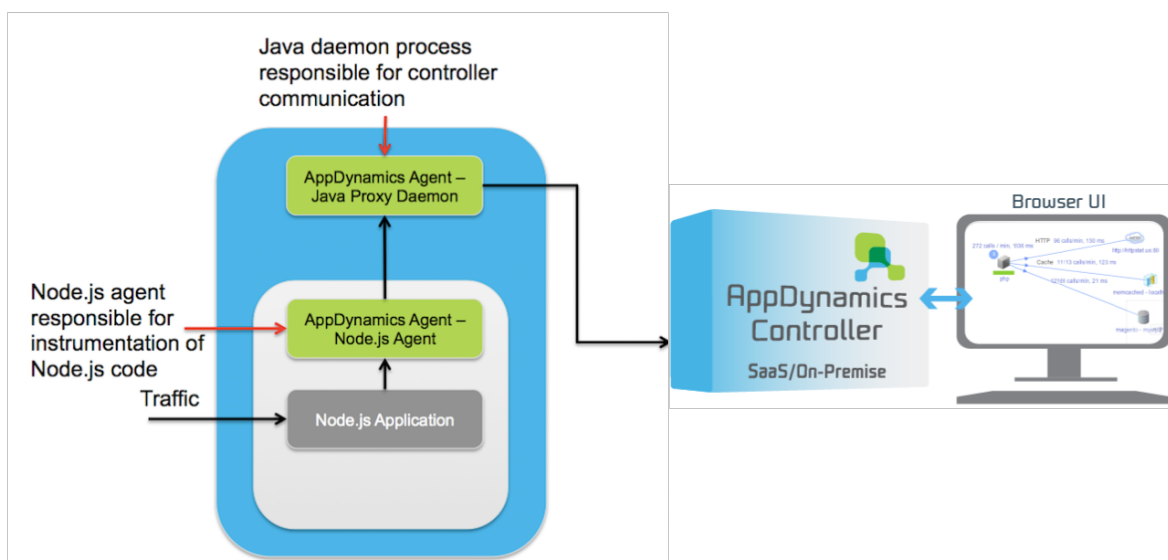
A license file is required. Contact the Node.js beta manager at [omed@appdynamics.com](mailto:omed@appdynamics.com) about obtaining a license and accessing the agent download site.

## AppDynamics for Node.js Architecture

The AppDynamics agent for Node.js instruments a single Node.js process. Instrumentation enables the agent to discover, map and track metrics for business transactions, application services, and backends in your Node.js application. Typically there is a one-to-one correspondence between a process and a Node.js application but sometimes an application consists of multiple processes.

This agent communicates with a Java proxy daemon that handles the communication between the Node.js agent and the AppDynamics Controller. The proxy reports the performance metrics to the Controller, where the data is stored, baselined, and analyzed. You can access this performance data interactively using the Browser UI or programmatically using the AppDynamics REST API.

The proxy component is automatically started when you start the Node.js app agent.



## Supported Environments and Versions for Node.js

- Supported Platform Matrix for the App Agent for Node.js
  - Node.js Versions
  - Operating Systems
  - Transaction Naming
  - HTTP Exit Points
  - Database Exit Points
  - Cache Exit Points

## Supported Platform Matrix for the App Agent for Node.js

### Node.js Versions

Supported Node.js Versions
0.8 +

### Operating Systems

Supported Operating System
Linux 32-bit
Linux 64-bit
Mac OSX v10.9.2

## Transaction Naming

Entry Type	Default Transaction Naming
Node.js Web	URI

## HTTP Exit Points

Supported HTTP Exit Points
Node.js HTTP client library

See <http://nodejs.org/api/http.html> for information about the Node.js HTTP client library.

## Database Exit Points

Supported Database Exit Points
MongoDB
MySQL
PGSQL
Riak

## Cache Exit Points

Supported Cache Exit Points
Memcached
Redis

## Install the App Agent for Node.js

- [Prerequisites for Agent Installation](#)
- [Instrumenting Node.js Applications](#)
  - [Installing the Agent](#)
  - [Modifying the Application Code](#)
- [Logs](#)
  - [Node.js Agent Log](#)
  - [Proxy Log](#)
- [Installing the Machine Agent on an Node.js Node](#)

## Prerequisites for Agent Installation

1. Download the tarball file for the App Agent for Node.js from the [AppDynamics beta download zone](#).

If you do not have access to the beta zone, please request it from your account manager and/or email the Node.js Product Manager: [omed@appdynamics.com](mailto:omed@appdynamics.com).

2. Be prepared to provide the following information:

- controller host and controller port: These are the host name or IP address and the port number of the AppDynamics controller that the agent connects to. On-premise customers establish these settings when they install the controller.
- AppDynamics application name: This is the name that you assign to the business application you will monitor with the App Agent for Node.js.
- AppDynamics tier name: This is the name that you assign to the tier you will monitor with the App Agent for Node.js.
- AppDynamics node name: This is the prefix to the dynamically-generated node name of the server that you will monitor with the App Agent for Node.js. On startup the agent assigns an index based on the startup of the node processes and appends this index to this prefix specified to create the node name that is displayed in the AppDynamics console.

3. You need permission to perform a package installation and to edit the application source code to complete the installation.

## Instrumenting Node.js Applications

There are two steps to instrument your Node.js applications:

- [Installing the Agent](#)
- [Modifying the Application Code](#)

### Installing the Agent

From the root directory of the Node.js application run the command:

```
npm install <nodejs_tarball_file>
```

### Modifying the Application Code

For every Node.js application that you are instrumenting, insert the following call in the application source code at the first line the main module (such as the server.js file), before any other require statements.

```
require("appdynamics").profile({
  controllerHostName: '<controller host name>',
  controllerPort: <controller port number>, // If SSL, be sure to
enable the next line
  controllerSslEnabled: true|false, // Optional - use if connecting
to controller via SSL
  accountName: '<AppDynamics account name>', // Required for a
controller running in multi-tenant mode.
  accountAccessKey: '<AppDynamics account key>', // Required for a
controller running in multi-tenant mode.
  applicationName: '<app_name>',
  tierName: '<tier_name>',
  nodeName: '<node_name>', // Prefix to the full node name.
  debug: true|false // Optional - defaults to false.
});
```

## Logs

There is an agent log and a proxy log for each application.

### Node.js Agent Log

If the agent is running in debug mode, the agent component logs to stdout/stderr. This log contains the transactions that the agent processes and sends to the proxy. This log is available in the same location to which stdout/stderr streams are directed from the monitored application.

If debug mode is not enabled, no agent log is generated.

You set debug mode in the require statement that instruments your Node.js application. See [Modifying the Application Code](#).

### Proxy Log

The proxy logs the transactions that it accepts from the agent and sends to the Controller. The proxy generates logs whether or not the agent is running in debug mode.

When the agent component launches the proxy, it displays in the agent log the directory path to which the proxy is logging.

## Installing the Machine Agent on an Node.js Node

If you install the Machine Agent on the machine hosting the instrumented Node.js node and you specify the tier and node name in the machine agent's controller-info.xml file, the App Agent for Node.js will fail to register.

To avoid this problem:

- Install the App Agent for Node.js before you install the Machine Agent
- Do not specify the tier and node in the machine agent controller-info.xml, where it is optional. The machine agent will pick up the tier and node from the app agent configuration.

## Uninstall the App Agent for Node.js

### To uninstall the App Agent for Node.js

1. From the application root directory of the application from which you want to uninstall the agent enter:

```
npm uninstall appdynamics
```

2. Remove the "require("appdynamics")" statement from your Node.js applications.

## Configure AppDynamics for Node.js

### Configure Transaction Detection for Node.js

- [Accessing Transaction Detection](#)
  - [To Access Business Transaction Detection Configuration](#)
- [Node.js Web Entry Point](#)
- [Business Transaction Naming and Identification](#)
  - [Dynamic Transaction Naming Based on the Request](#)
- [Optimize Business Transaction Detection](#)
  - [To exclude business transactions](#)
  - [To reverse business transaction exclusion](#)
- [Learn More](#)

### Accessing Transaction Detection

#### To Access Business Transaction Detection Configuration

1. From the left navigation pane select **Configure -> Instrumentation**.
2. Click the Transaction Detection tab if it is not already selected.
3. Click the Node.js - Transaction Detection tab.
4. Do one of the following:
  - To configure transaction detection at the application level, in the left panel select the application.  
If you select the application, you can optionally click the button to configure all tiers to use the application-level configuration.
  - To configure transaction detection at the tier level, in the left panel select the tier for which you want to configure transaction detection.  
You can choose the button to apply the application configuration to the selected tier or the button to create a custom configuration just for this tier.

### Node.js Web Entry Point



The entry point is where the business transaction begins. Typically an entry point is a method or operation or a URI.

To enable transaction monitoring check the Transaction Monitoring check box. You can disable monitoring at any time by clearing this check box.

For Node.js Web entry points to be automatically detected, check the Automatic Transaction Detection check box.

The screenshot shows the 'Node.js - Transaction Detection' configuration window. At the top, there are tabs for 'Java - Transaction Detection', '.NET - Transaction Detection', 'PHP - Transaction Detection', and 'Node.js - Transaction Detection' (which is selected). Below the tabs are two buttons: 'Copy' and 'Configure all Tiers to use this Configuration'. A section titled '▼ Entry Points' contains a table with columns 'Type', 'Transaction Monitoring', and 'Automatic Transaction Detection'. The 'Web' entry point is shown with 'Transaction Monitoring' checked and 'Automatic Transaction Detection' checked. A link 'Configure Naming' is visible next to the 'Automatic Transaction Detection' checkbox.

Type	Transaction Monitoring	Automatic Transaction Detection
Web	<input checked="" type="checkbox"/> Enabled	<input checked="" type="checkbox"/> Discover Transactions automatically for all web requests <a href="#">Configure Naming</a>

## Business Transaction Naming and Identification

The default convention for Node.js Web transactions is to use first two segments of the URI of the application to name the transaction. Click **Configure Naming** if you want to change the naming convention to the full URI or to use different segments of the URI to give the business transactions more meaningful names.

The screenshot shows the 'Node.js Web Transaction Naming Configuration' dialog. It asks 'What part of the URI should be used in the Transaction Name?'. There are two radio buttons: 'Use the full URI' (unselected) and 'Use a part of the URI (for example, if you have dynamic URIs)' (selected). Below the radio buttons, there is a dropdown menu set to 'Use the first' and a text box containing the number '2', followed by the text 'segments of the URI in Transaction Names'.

If the first two segments of all your transactions are similar, you might want to base the business transaction names on a different part of the URI:

The screenshot shows the 'Node.js Web Transaction Naming Configuration' dialog. It asks 'What part of the URI should be used in the Transaction Name?'. There are two radio buttons: 'Use the full URI' (unselected) and 'Use a part of the URI (for example, if you have dynamic URIs)' (selected). Below the radio buttons, there is a dropdown menu set to 'Use the last' and a text box containing the number '3', followed by the text 'segments of the URI in Transaction Names'. At the bottom, there is an unchecked checkbox labeled 'Name Transactions dynamically using part of the request'.

## Dynamic Transaction Naming Based on the Request

You can also configure dynamic transaction naming based on the details of the user request.

For example:

If the web context is **http://example.com/store/checkout**, the naming strategy would use the default first two segments to name these transactions as **/store/checkout**.

If the web context is **http://example.com/store/secure/internal/updateinventory**, the naming strategy would use last two segments to name these requests as **/internal/updateinventory**.

If the web context is **http://wines.com/store/orders.special**, the naming strategy would use the combination of the parameter value for "type" and the last two segments to name such requests as **/orders/process.special**. Similarly, depending on the web could, you can use the cookie, header, request method or any combinations of URI segments to configure useful business transaction names.

Node.JS Web Transaction Naming Configuration

What part of the URI should be used in the Transaction Name?

☐ Use the full URI

☒ Use a part of the URI (for example, if you have dynamic URIs)

Use the last

2

segments of the URI in Transaction Names

[What does this do?](#)

☒ Name Transactions dynamically using part of the request

☐ Use URI segment(s) in Transaction names

Segment Numbers
Enter a comma separated list of parameter numbers (e.g. 1,3,4)

☒ Use a parameter value in Transaction names

Parameter Name

type

☐ Use a header value in Transaction names

Header Name

☐ Use a cookie value in Transaction names

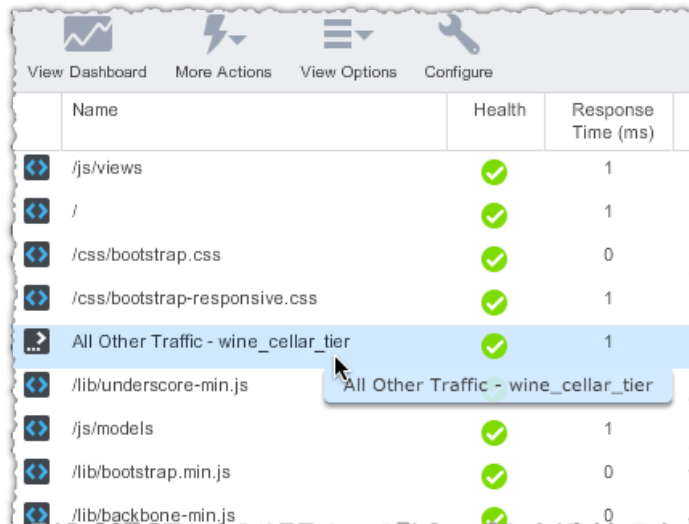
Cookie Name

☐ Use the request method (GET/POST/PUT) in Transaction names

## Optimize Business Transaction Detection

The agent detects all requests to an instrumented Node.js server as business transactions, including requests for static content such as images, CSS files, JavaScript files, and static HTML pages.

For this reason you may see business transactions automatically generated for static content in the business transaction list. You probably want to prevent these types of transactions from being detected, especially since there are default limits of 50 business transactions per agent and 200 per business application. After one of these limits is reached, all the excess traffic to the server is collected into a single business transaction called All Other Traffic. See [All Other Traffic Business Transaction](#) for more information about this. If necessary you can increase the business transaction limit by setting the the max-business-transactions property in <nodejs-agent-root>/proxy/conf/app-agent-config.xml.

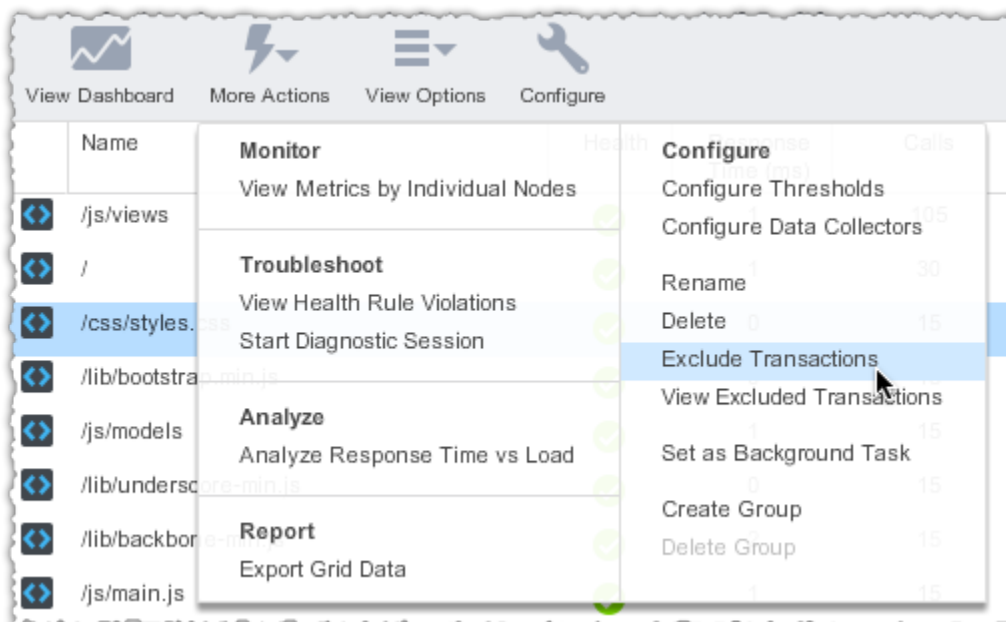


Name	Health	Response Time (ms)
/js/views	✓	1
/	✓	1
/css/bootstrap.css	✓	0
/css/bootstrap-responsive.css	✓	1
All Other Traffic - wine_cellar_tier	✓	1
/lib/underscore-min.js	✓	1
/js/models	✓	1
/lib/bootstrap.min.js	✓	0
/lib/backbone-min.js	✓	0

You can exclude unimportant business transactions in the business transactions list to stay under the limit.

To exclude business transactions

1. In the left navigation pane, click **Business Transactions**.
2. In the business transactions list, select the transaction(s) that you want to remove.
3. Click **More Actions**.
4. In the menu click **Exclude Transactions**.



Name	Monitor	Health	Configure	Calls
/js/views	View Metrics by Individual Nodes	✓	Configure Thresholds	105
/		✓	Configure Data Collectors	30
/css/styles.css		✓	Rename	15
/lib/bootstrap.min.js		✓	Delete	15
/js/models		✓	Exclude Transactions	15
/lib/underscore-min.js		✓	View Excluded Transactions	15
/lib/backbone-min.js		✓	Set as Background Task	15
/js/main.js		✓	Create Group	15
		✓	Delete Group	15

5. In the Exclude Business Transactions screen click **Exclude Business Transactions**.

To reverse business transaction exclusion

1. In the business transactions list, click **More Actions**.
2. In the menu click **View Excluded Transactions**.
3. In the Excluded Business Transactions window, do one of the following:

- To unexclude all the excluded transactions, click Unexclude All.
- To unexclude some of the excluded transactions, select the transaction(s) and click **Unexclude Selected**.

After a business transaction has been unexcluded, the agent starts monitoring it again and it is counted toward the business transaction limit.

## Learn More

- [Business Transaction Monitoring](#)

## Configure Error Detection for Node.js

- [Accessing Error Detection Configuration for the Node.js Agent](#)
  - [To Access Error Detection Configuration](#)
- [Ignoring Exceptions](#)
  - [To configure the Node.js agent to ignore an exception](#)

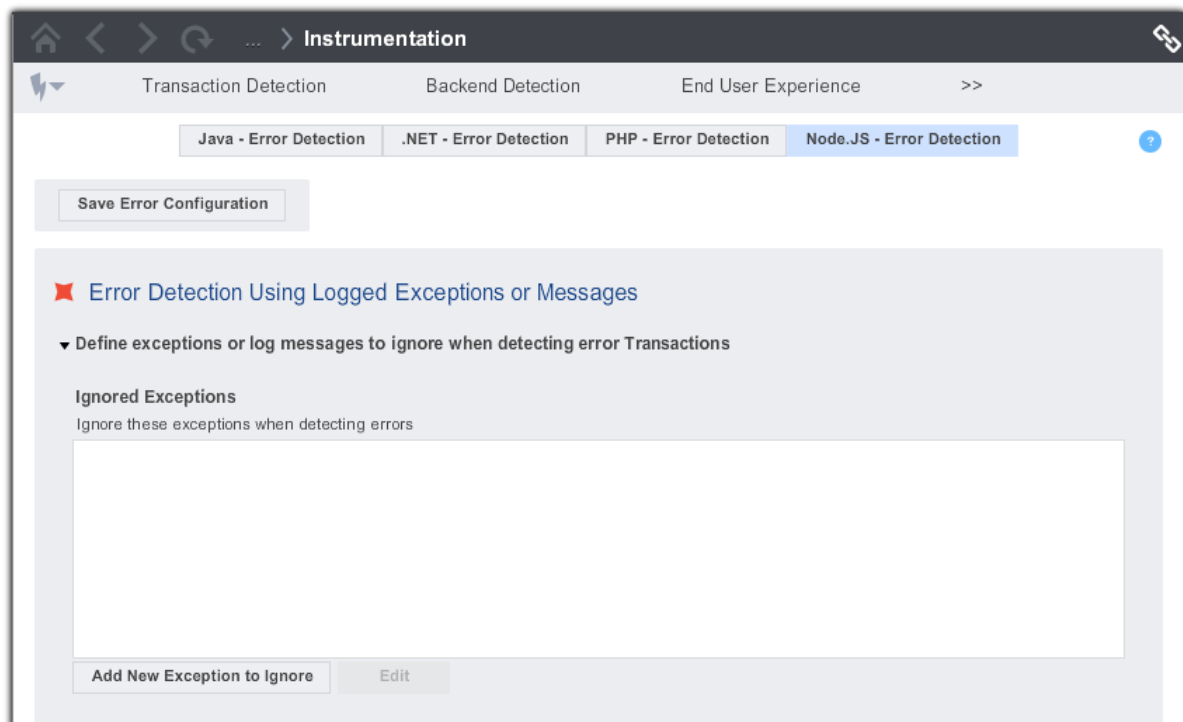
You can configure which exceptions and log messages the agent should ignore when reporting errors. Ignored errors are not included in the business transaction error count.

These configurations help you reduce the number of errors that the agent reports so you see just those that are most interested for monitoring and troubleshooting.

## Accessing Error Detection Configuration for the Node.js Agent

### To Access Error Detection Configuration

1. From the left navigation pane select **Configure -> Instrumentation**.
2. Click the Error Detection tab.
3. Click the Node.js - Error Detection tab.



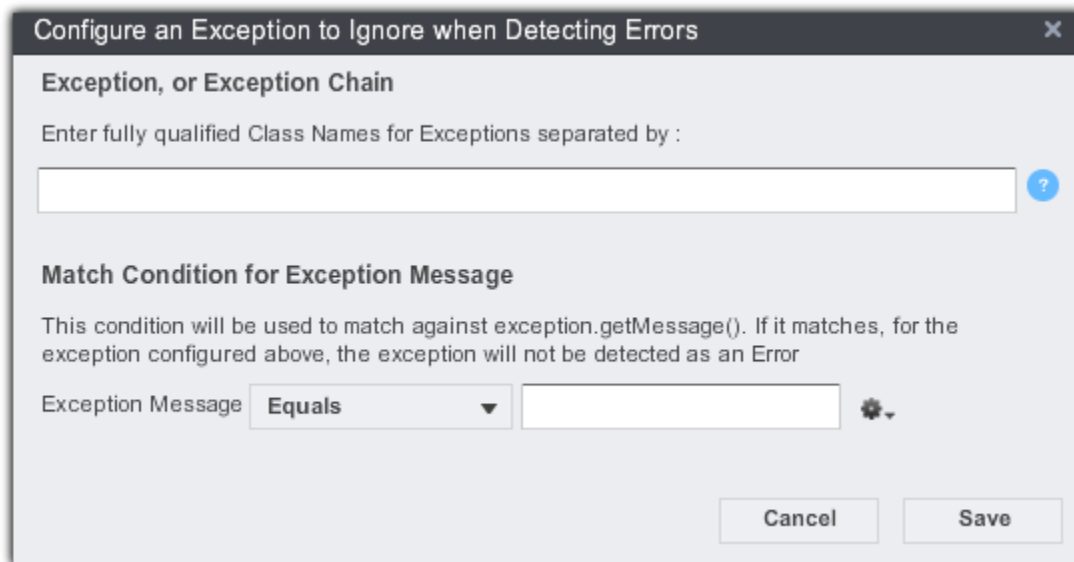
## Ignoring Exceptions

Exceptions that are ignored are listed in the Ignored Exceptions list.

To configure the Node.js agent to ignore an exception

Click **Add New Exceptions to Ignore** under the Ignored Exceptions list.

1. Enter the fully-qualified class names of exceptions to ignore, separated by colons.



Exception chains are supported. If you specify "A:B" as the exception chain, the agent matches any chain where B is in the chain of A, even if B is not an immediate cause of the exception.

2. Optionally, further qualify the exceptions to ignore by configuring a match condition for the exception message. Enter the string to match and the qualifier from the dropdown list (Equals, Contains, Starts with, Ends with, Matches Reg Ex). Click the gear icon to configure the NOT condition for the match.
3. Click **Save**.
4. Click **Save Error Configuration** in the top left corner of the window.

You can later edit or remove the ignored exception from the list by selecting it in the Ignored Exceptions list and clicking the Edit or Delete icon.

## Monitor Node.js Applications

A node in the AppDynamics model is a single Node.js process instrumented by an AppDynamics app agent.

The node dashboard for Node.js is like the node dashboards for the other app agent except:

- It includes a Process Snapshots tab for accessing the process snapshots for the node.

- It does not include a Memory tab for monitoring memory usage.

See [Node Dashboard](#) for general information about node dashboards.

The tier dashboard for Node.js is similar to the tier dashboards for the other app agents except that it includes a tab for accessing the process snapshots for the nodes contained by the tier.

See [Tier Dashboard](#) for general information about tier dashboards.

See [Monitor Node.js Processes](#) for information about process snapshots.

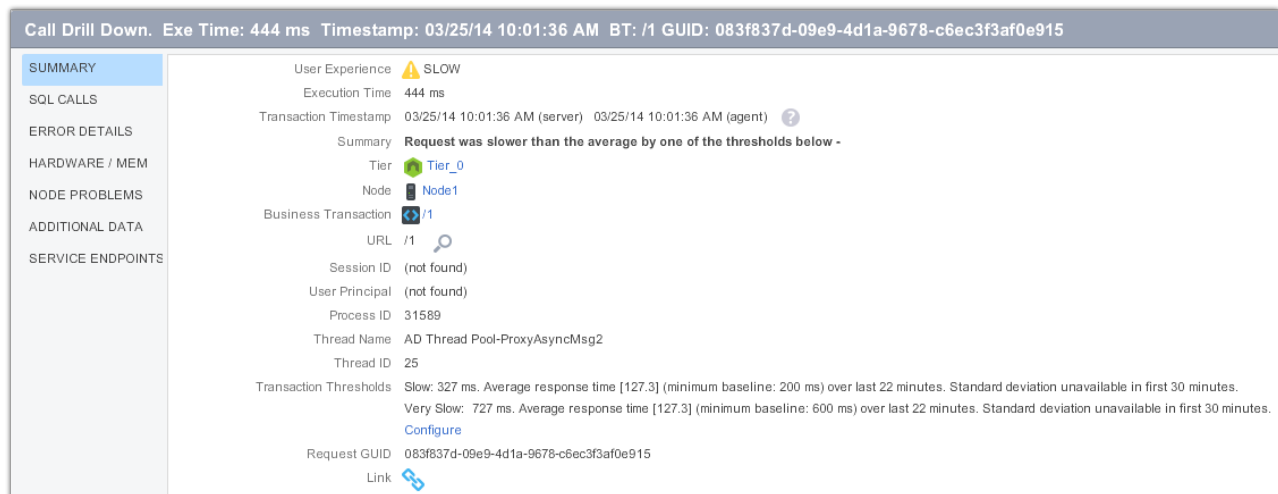
## Monitor Node.js Business Transactions

See [Business Transaction Monitoring](#) and [Transaction Snapshots](#) for general information about monitoring business transactions.

The beta App Agent for Node.js does not support:

- call graphs and hotspots in transaction snapshots
- data collectors
- distributed transactions
- correlation with end user monitoring (web and mobile) snapshots

This is a transaction snapshot captured by the App Agent for Node.js:



To view call graphs that reveal bottlenecks in your Node.js processes, use [process snapshots](#).

## Monitor Node.js Processes

- [Set Up Process Snapshot Collection](#)
- [Process Snapshots List](#)
  - [To filter the process snapshots list](#)
  - [More Actions Menu](#)
- [View Process Snapshots](#)

One reason for slow load times is inefficient code that uses a lot of CPU time. In a single-threaded model, such as Node.js, one slow process forces other processes to wait.

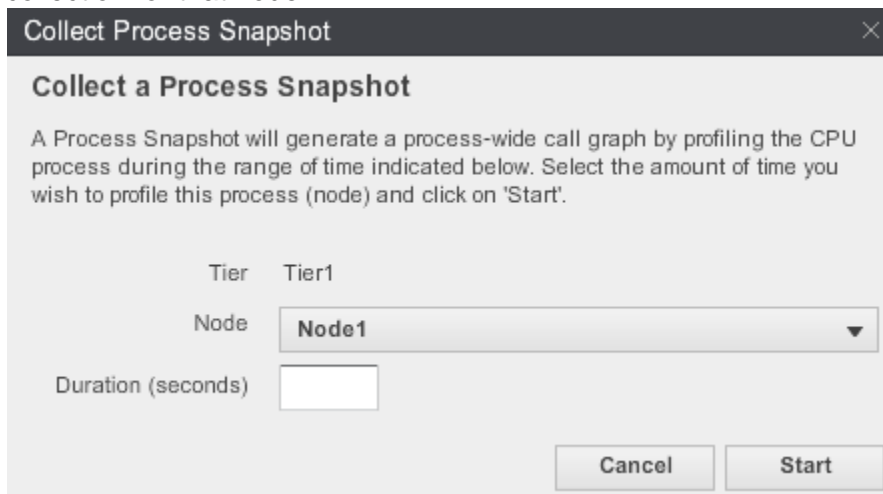
You can monitor Node.js processes using lists of process snapshots to identify which processes have high CPU times. From the list you can select and examine process snapshots of slow processes to identify exactly which functions in your code are blocking the CPU.

A process snapshot describes an instance of a CPU process on an instrumented node.js node. It generates a process-wide call graph for a CPU process over a configurable time range. Process snapshots are independent of any running business transactions. Process snapshots persist for 14 days, unless you archive them in which case they are available forever.

You can monitor process snapshots at the tier level or the node level.

## Set Up Process Snapshot Collection

1. Navigate to the dashboard for the tier or node for which you want to collect process snapshots.
2. Click the Process Snapshots tab.
3. Click **Collect Process Snapshots**.
4. If you are in the tier dashboard, select the node for which you want to collect snapshots from the Node dropdown list. If you are in the node dashboard, you can only set up snapshot collection for that node.



5. Enter how many seconds you want to collect process snapshots for this node. The maximum is 60 seconds.
6. Click **Start**.

The agent collects process snapshots for the configured duration.

## Process Snapshots List

To access the process snapshots list:

1. Navigate to the dashboard for the tier or node for which you want to view process snapshots.
2. Click the Process Snapshots tab.

For each process snapshot the list displays the time the process started, the process's execution time in milliseconds, and the tier and node in which the process executed.

Click the Exe Time column and then toggle the direction to sort the snapshots in descending order by execution time. The processes with the slowest CPU times will be at the top of the list. These are the snapshots you will want to examine.

	Time	Exe Time (▼)	Summary	Tier	Node
03/20/14 2:29:52 PM	60000	Triggered by snapshot request at:Thu Mar 20 14:28:	Tier_0	Node1	
03/20/14 3:24:57 PM	60000	Triggered by snapshot request at:Thu Mar 20 15:23	Tier_0	Node1	
03/20/14 4:13:58 PM	60000	Triggered by snapshot request at:Thu Mar 20 16:12:	Tier_0	Node1	
03/20/14 1:31:05 AM	24000	Triggered by snapshot request at:1395304214440	Tier_0	Node1	
03/20/14 3:42:32 PM	15000	Triggered by snapshot request at:Thu Mar 20 15:38	Tier_0	Node1	
03/20/14 3:42:11 PM	14000	Triggered by snapshot request at:Thu Mar 20 15:37	Tier_0	Node1	
03/20/14 1:28:54 AM	13000	Triggered by snapshot request at:1395304076677	Tier_0	Node1	
03/20/14 3:41:29 PM	12000	Triggered by snapshot request at:Thu Mar 20 15:37	Tier_0	Node1	
03/20/14 3:41:08 PM	11000	Triggered by snapshot request at:Thu Mar 20 15:37	Tier_0	Node1	
03/20/14 3:36:27 PM	10000	Triggered by snapshot request at:Thu Mar 20 15:34	Tier_0	Node1	
03/20/14 3:36:16 PM	9000	Triggered by snapshot request at:Thu Mar 20 15:34	Tier_0	Node1	
03/20/14 3:36:05 PM	8000	Triggered by snapshot request at:Thu Mar 20 15:34	Tier_0	Node1	
03/20/14 1:26:41 PM	7000	Triggered by snapshot request at:Thu Mar 20 13:22:	Tier_0	Node1	

#### To filter the process snapshots list

You can filter the process snapshot list to display only the snapshots that you are interested in. You can filter by execution time, whether the snapshot is archived, and the GUID of the request. If you access the list from the tier dashboard, you can also filter by node.



Hide Filters | Collect Process Snapshot | More

▼ Execution Time  
 >=  ms

▼ Nodes  
☐ Node1

▼ Archived  
☐ Return Only Archived Snapshots

▼ Advanced  
 Request GUID's

Clear Criteria Search

### More Actions Menu

Use the More Actions menu to select one or more process snapshots in the list and perform the following actions on them.

- **Archive** Use this option to archive the selected process snapshots. Normally snapshots are purged after two weeks. You can archive a snapshot beyond the normal snapshot lifespan to retain it for future analysis. You can view archived snapshots by checking the Archived filter in the Filters panel.
- **Delete items** Use this option to remove the snapshot from the list.
- **Copy a Link to this Snapshot to the clipboard** Use this option to send a link to the snapshot to someone.
- **Export Grid Data** Use this option to export the snapshot data to a file.

### View Process Snapshots

From the process snapshots list, double-click the process snapshot that you want to view.

A process snapshot contain the following tabs:

- **SUMMARY:** Displays the execution time of the process, timestamp of the snapshot, CPU time used, tier and node the process ran on, GUID of the request, etc..

You can click the link icon to copy a link of the snapshot URL.

Process Snapshot Drill Down. Exe Time: 10000 ms Timestamp: 03/25/14 9:43:16 AM GUID: f80aa563-55a2-4aa7-a7c6-c41618b6fd5

SUMMARY

CALL GRAPH

HOT SPOTS

ADDITIONAL DATA

Execution Time

CPU Time

Time

Summary

Tier

Node

Process ID

Request GUID

Link

10000 ms

0 ms (0 %)

03/25/14 9:43:16 AM (server) 03/25/14 9:43:16 AM (agent) ?

Triggered by snapshot request at: Tue Mar 25 09:42:51 PDT 2014

Tier\_0

Node1

31589

f80aa563-55a2-4aa7-a7c6-c41618b6fd5

Close

- **CALL GRAPH:** Shows the execution time of methods on the process's call stack. In the Time column you can identify which calls take the longest time to execute.

Process Snapshot Drill Down. Exe Time: 10000 ms. Timestamp: 03/28/14 9:32:03 AM. GUID: c1686944-a61c-439e-ac4d-02d1

SUMMARY

CALL GRAPH


HOT SPOTS

ADDITIONAL DATA

Execution Time: 10000 ms. Node Node1. Timestamp: 03/28/14 9:32:03 AM.

Set as Root Reset Root (?) Show Filters

Name	Time (ms)	
▼ Writeable::write - _stream_writable.js:160	0 ms (self)	0 %
▶ Socket::write - net.js:617	8 ms (total)	0.1 %
▼ (global)::(anonymous function) - http.js:1722	4 ms (self)	0 %
EventEmitter::emit - events.js:53	4 ms (self)	0 %
▼ (global)::onread - net.js:494	9 ms (self)	0.1 %
▼ Readable::push - _stream_readable.js:116	0 ms (self)	0 %
▼ (global)::readableAddChunk - _stream_readable.js:136	5 ms (self)	0.1 %
▼ (global)::emitReadable - _stream_readable.js:392	4 ms (self)	0 %
▼ (global)::emitReadable_ - _stream_readable.js:407	4 ms (self)	0 %
▼ EventEmitter::emit - events.js:53	5 ms (self)	0.1 %
▼ (global)::(anonymous function) - _stream_readable.js:733	0 ms (self)	0 %
▼ EventEmitter::emit - events.js:53	0 ms (self)	0 %
▼ (global)::(anonymous function) - lib/Connection.js:71	0 ms (self)	0 %
▶ Protocol::write - protocol/Protocol.js:36	8313 ms (total)	83.1 %
▼ Readable::read - _stream_readable.js:252	0 ms (self)	0 %
(global)::howMuchToRead - _stream_readable.js:214	4 ms (self)	0 %
▼ (global)::maybeReadMore - _stream_readable.js:418	4 ms (self)	0 %
(global)::apply -	4 ms (self)	0 %
▼ socket::ondata - http.js:1965	0 ms (self)	0 %
▼ (global)::parserOnHeadersComplete - http.js:68	0 ms (self)	0 %
▼ parser::onIncoming - http.js:2020	0 ms (self)	0 %
▼ EventEmitter::emit - events.js:53	0 ms (self)	0 %

Click the  to see more information about a call.

write:36

Name: Protocol:write - protocol/Protocol.js  
Type: JS  
Class: Protocol  
Method: write  
Line Number: 36

**Execution Time**

Self Time: 10 ms 0.1 %  
Total Time: 8313 ms 83.1 %


**Exit Calls**

No exit calls were made.

Close

The total processing time displayed for the root element in the call graph includes the CPU idle time.

Execution Time: 10000 ms. Node Node1. Timestamp: 03/28/14 9:32:03 AM.






Set as Root Reset Root (?) Show Filters 

Name	Time (ms)
(global)::(root) -	10000 ms (total) <span>100 %</span>

- **HOT SPOTS:** Sorts the calls by execution time, with the most expensive calls at the top. To see the invocation trace of a single call in the lower panel, select the call in the upper panel. Use the slider in the upper right corner to filter which calls to display as hot spots. For example, the following setting filters out all calls faster than 496 ms from the hot spots list.

Process Snapshot Drill Down. Exe Time: 10000 ms Timestamp: 03/28/14 9:32:03 AM GUID: c1686944-a61c-439e-ac4d-d2d3a49e4424

SUMMARY This screen displays all of the method calls in the call graph sorted by time Method Time (ms) 0 496 992 1.5k 2.0k  
CALL GRAPH don't filter by time

Name	Method Time (ms)	External Calls	Details
RowDataPacket:parse - packets/RowDataPacket.js:1	1984 ms (self) <span>19.8 %</span>		
Buffer:toString - buffer.js:392	1835 ms (self) <span>18.4 %</span>		
Parser:write - protocol/Parser.js:22	1248 ms (self) <span>12.5 %</span>		
RowDataPacket:_typeCast - packets/RowDataPac	770 ms (self) <span>7.7 %</span>		
(global)::Date -	501 ms (self) <span>5 %</span>		

**Invocation Trace**

(global.(root):0 (237ms self time, 10000 ms total time)  
(global.onread:494 (9ms self time, 8500 ms total time)  
Readabl.push:116 (0ms self time, 8343 ms total time)  
(global.readableAddChunk:136 (5ms self time, 8343 ms total time)  
(global.emitReadable:392 (4ms self time, 8330 ms total time)  
(global.emitReadable:\_407 (4ms self time, 8326 ms total time)  
EventEmitte.emit:53 (5ms self time, 8322 ms total time)  
(global.(anonymous function):733 (0ms self time, 8317 ms total time)  
EventEmitte.emit:53 (0ms self time, 8313 ms total time)

- **ADDITIONAL DATA :** Displays the unique id of the process.

## Monitor Node.js Backends

A backend is an entity in the AppDynamics model that the app agent does not instrument directly, but monitors traffic flows to it. The App Agent for Node.js monitors flows to database and cache exit points. See [Supported Environments and Versions for Node.js](#) for the current list of supported backends.

You cannot configure detection and naming for Node.js backends.

For general information about monitoring backends see:

- [Backend Monitoring](#)
- [Monitor Databases](#)
- [Monitor Remote Services](#)

## Troubleshoot Node.js Application Problems

### Troubleshoot Slow Response Times for Node.js

- [Slow and Stalled Transactions](#)
  - [To troubleshoot slow and stalled transactions](#)
- [High CPU Times](#)
  - [To troubleshoot slow processes](#)
- [Learn More](#)

When you click **Troubleshoot -> Slow Response Times**, the Slow Response Times window opens showing two tabs. You can drill down into transaction issues in the [Slow Transactions](#) tab and into database or remote services issues in the [Slowest DB & Remote Services](#) tab.

### Slow and Stalled Transactions

There are many reasons why a business transaction may be slow or stalled.

By default AppDynamics considers a slow transaction one that lasts longer than 3 times the standard deviation for the last two hours and a very slow transaction 4 times the baseline for the last two hours.

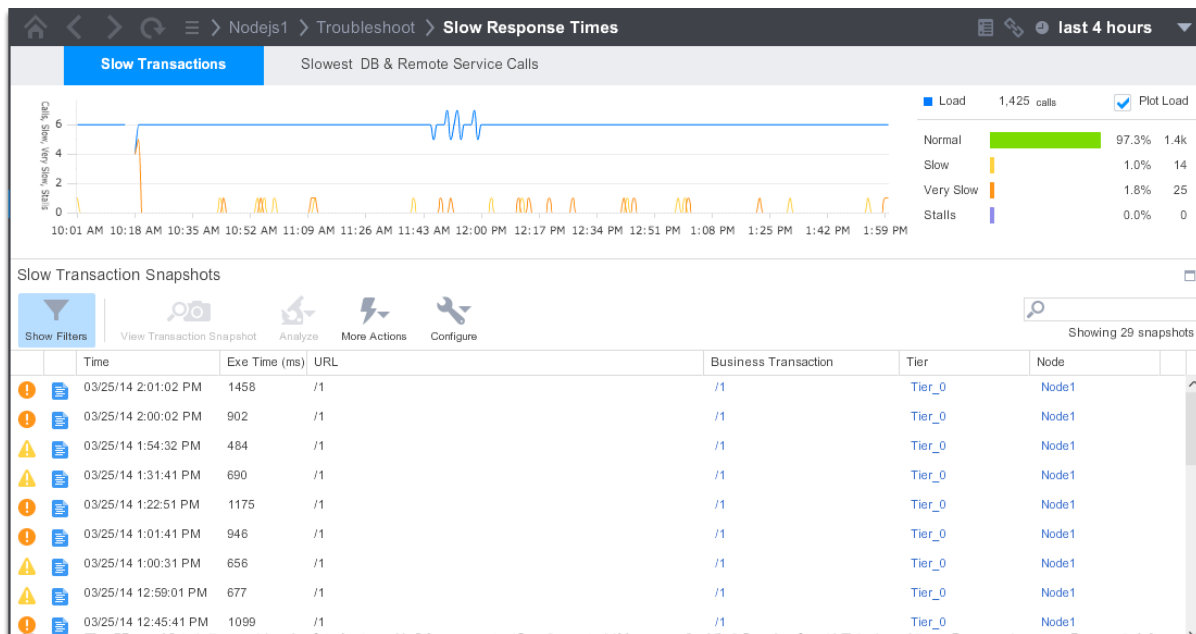
By default AppDynamics considers a transaction that lasts longer than 45 seconds (4500 milliseconds) to be stalled.

You can configure these thresholds to better match your environment. See [Thresholds](#) and [Configure Thresholds](#).

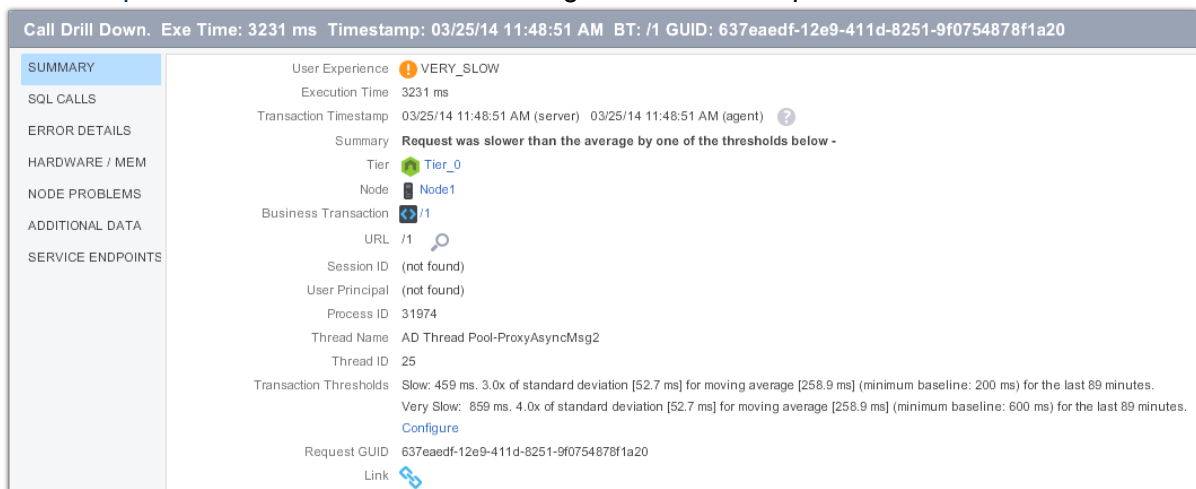
#### To troubleshoot slow and stalled transactions

1. Click **Troubleshoot -> Slow Response Times**.
2. Click the **Slow Transactions** tab if it is not selected.  
In the upper pane AppDynamics displays a graph of the slow, very slow, and stalled transactions for the time period specified in the Time Range drop-down menu. Click the Plot Load check box to see the load.

In the lower pane AppDynamics displays the transaction snapshots for slow, very slow, and stalled transactions.



- Click the Exe Time column to sort the transactions from slowest to fastest.
- To drill down, select a snapshot from the list and click **View Transaction Snapshot**. A transaction snapshot shows the details of an instance of a business transaction. See [Transaction Snapshots](#) for information about drilling down into a snapshot.




## High CPU Times

Your user experience may be slow because of processes that consume a lot of CPU time blocking other processes.

### To troubleshoot slow processes

- In the node dashboard click the Process Snapshots tab.
- Click Collect Process Snapshots.
- In the Collect a Process Snapshot window set the duration to between 1 and 60 seconds.

4. Click Start.  
In a few minutes you should see some process snapshots in the list.
5. Sort the list by toggling the Exe Time column in the process snapshots list so that the snapshots for the slowest processes are at the top of the list.
6. Double-click one of the slow process snapshots.
7. In the process snapshot, click the CALL GRAPH tab if it is not selected.
8. Examine the Time column in the call graph to identify which of the calls are slow.
9. Click the  at the end of the row for the slow call to see details.

## Learn More

- [Monitor Node.js Business Transactions](#)
- [Monitor Node.js Processes](#)
- [Transaction Snapshots](#)
- [Configure Thresholds](#)