Build a Monitoring Extension Using Scripts

On this page:

Related pages:

- Extensions and Custom Metrics
- Build a Monitoring Extension Using Java
- Standalone Machine Agent HTTP Listener
- Configure Health Rules
- Alert and Respond
- Remediation Actions

Watch the video:

Standalone Machine Agent - Report Metrics from Extensions

You can write a monitoring extension script (also known as a custom monitor or hardware monitor) to add custom metrics to the metric set that AppDynamics already collects and reports to the Controller. Your script reports the custom metrics every minute to the Standalone Machine Agent. The Standalone Machine Agent passes these metrics to the Controller.

This topic describes the steps for adding custom metrics using a shell script and includes an example.

Review Existing Extensions

Before creating your own extension, review the extensions that have been created and shared among members of the AppDynamics community. New extensions are added continuously. It is possible that someone has already created exactly what you need or something close enough that you can download it and use it after making a few simple modifications.

The extensions are described and their source is available for free download at: http://www.appdynamics.com/community/exchange/

Process to Create a Monitoring Extension Using a Script

The general steps to create a monitoring extension using a script are the following:

1. Create your script. See Create the script file.
2. Create a monitor.xml configuration file. See Create the monitor.xml file.
3. Create a subdirectory (<your_extension_dir>) in <machine_agent_home>/monitors. See Create a directory under the Standalone Machine Agent monitors directory.
4. Copy your script file and the monitor.xml file into the new subdirectory.
5. Restart the Standalone Machine Agent.

Agent Configuration Requirements
Confirm that you have correctly configured the machine agent in the controller-info.xml file and on the agent start command on the command line. For information on configuring required and optional agent properties, see Standalone Machine Agent Configuration Properties.

Defining Your Metrics

Metric names must be unique within the same metric path but need not be unique for the entire metric hierarchy. It is a good idea to use short metric names so that the whole name is visible when displayed in the Metric Browser. Prepend the metric path to the metric name when you upload the metrics to the Controller.

Metric Processing Qualifiers

The Controller has various qualifiers for how it processes a metric with regard to aggregation, time rollup and tier rollup. There are three types of metric qualifiers:

1. Aggregation qualifier
2. Time roll-up qualifier
3. Cluster roll-up qualifier

In the script, specify the metric qualifiers after the name-value pair for the metric. A typical metric entry in the script file has the following structure:

```
name=<metric name>,value=<long value>,aggregator=<aggregator type>,
time-rollup=<time-rollup strategy>, cluster-rollup=<cluster-rollup
strategy>
```

Aggregation Qualifier

The **aggregator** qualifier specifies how the Standalone Machine Agent aggregates the values reported during a one-minute period. Specify the aggregation qualifier as aggregator="aggregator type" This value is an enumerated type. If no value is reported during that minute, no data is reported to the controller, and an UNCHANGED notice appears in the Machine Agent log for that metric.

Valid values are:

Aggregator Type	Description
AVERAGE	Default. Average of all reported values in that minute.
SUM	Sum of all reported values in the minute, causes the metric to behave like a counter.
OBSERVATION	Last reported value in the minute.

Time Roll Up Qualifier

The **time-rollup** qualifier specifies how the Controller rolls up the values when it converts from one-minute granularity tables to 10-minute granularity and 60-minute granularity tables over time. The value is an enumerated type. Valid values are:

Roll up Strategy	Description
AVERAGE	Average of all one-minute values when adding it to the 10-minute granularity table; average of all 10-minute values when adding it to the 60-minute granularity table.

SUM	Sum of all one-minute values when adding it to the 10-minute granularity table; sum of all 10-minute values when adding it to the 60-minute granularity table.
CURRENT	Last reported one-minute value in that 10-minute interval; last reported ten-minute value in that 60-minute interval.

Cluster Rollup Qualifier

The **cluster-rollup** qualifier specifies how the Controller aggregates metric values in a tier (a cluster of nodes). The value is an enumerated type. Valid values are:

Roll up Strategy	Description
INDIVIDUAL	Aggregates the metric value by averaging the metric values across each node in the tier.
COLLECTIVE	Aggregates the metric value by adding up the metric values for all the nodes in the tier.

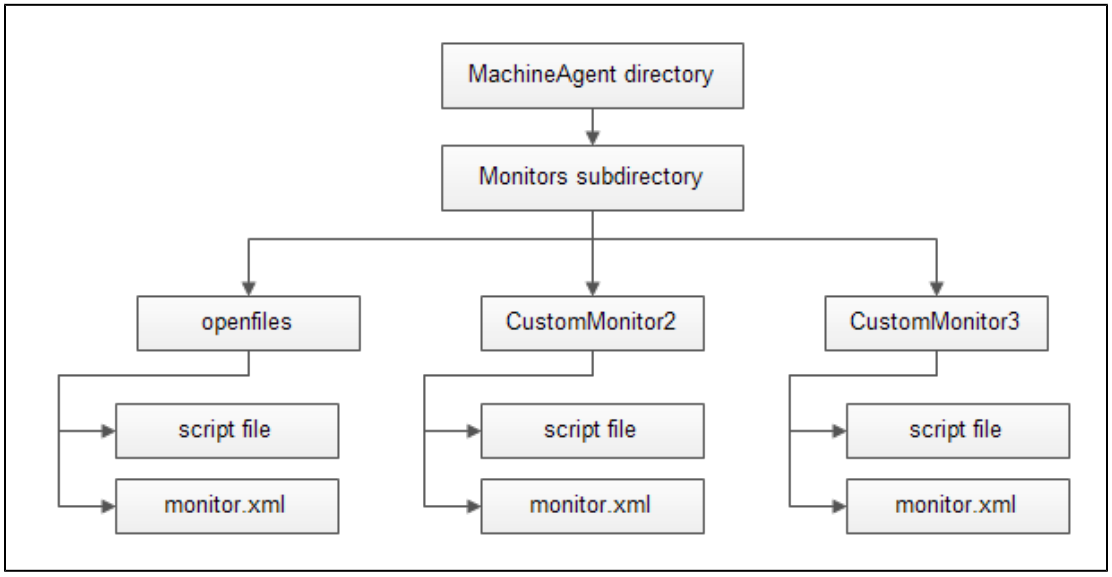
For example, if a tier has two nodes, Node A and Node B, and Node A has 3 errors per minute and Node B has 7 errors per minute, the INDIVIDUAL qualifier reports a value of 5 errors per minute and and COLLECTIVE qualifier reports 10 errors per minute. INDIVIDUAL is appropriate for metrics such as % CPU Busy where you want the value for each node. COLLECTIVE is appropriate for metrics such as Number of Calls where you want a value for the entire tier.

Steps To Add a Monitoring Extension Script

Step 1. Create a subdirectory under the Standalone Machine Agent monitors directory

The <*machine_agent_home*>/monitors directory is the repository for all monitoring extensions. For each new extension, create a subdirectory under the /monitors directory. The user running the agent requires read, write, and execute permissions to this subdirectory.

For example to create a monitoring extension that monitors open files in the JVM, create a subdirectory named "openfiles" under the <machine agent home/monitors> directory. The structure looks like this:



Step 2. Create the script file

A script writes data to STDOUT. The Standalone Machine Agent parses STDOUT and sends information to the Controller every

minute. Use the following instructions to create the script file.

1. Specify a name-value pair for the metrics.
   Each metric has a name-value pair that is converted to a java 'long' value. A typical metric entry in the script file has the following structure:

   ```
   name=<metric name>,value=<long value>,aggregator=<aggregator
   type>, time-rollup=<time-rollup strategy>,
   cluster-rollup=<cluster-rollup strategy>
   ```

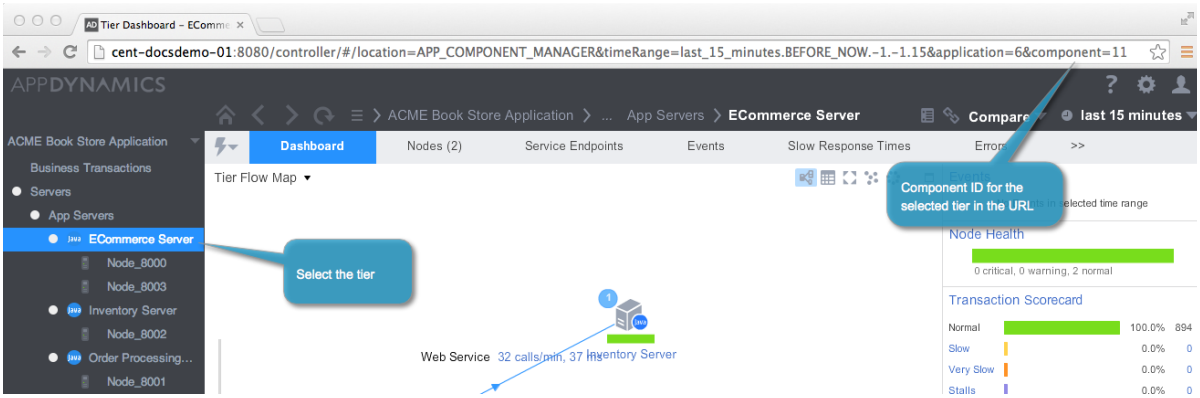
   Use the following format:

	Format
Standard Form	Hardware Resources| Instrument Name=Instrument Value
Fully Qualified Form	Hardware Resources| <metric name>,value=<long value>

2. Define the category of the metric, for example:
   a. Infrastructure (for the default hardware metrics, see Server Monitoring)
   b. JVM (for the default metrics, see Monitor JVMs)
   c. Custom Metrics

   Custom metrics must have the path prefixes:
   a. Custom Metrics
   b. Server|Component:<id>

   Metrics with the Custom Metrics prefix are common across all tiers in your application. Metrics with the Server|Component:<id> prefix appear only under the specified tier.
   To discover the component ID of a tier, select the tier from the Controller UI application tree. The component ID appears in the URL of the browser.



   > The machine agent has to be associated with the target/destination for the metrics. If you try to publish metrics to a Tier that is not associated with the Machine agent, the metrics can not be reported.

   The "|" character separates the branches in the metric hierarchy, telling the Controller where the metric should appear in the metric tree:

   ```
   Custom Metrics|Hardware Resources|Disks|Total Disk Usage %
   Custom Metrics|Hardware Resources|Disks|Disk 1|Current Disk
   Usage %
   ```

   You can insert a custom metric alongside an existing type of metric. For example, the following declaration causes the custom metric named pool usage to appear alongside the JMX metrics:

a. Server|Component:18|JMX|Pool|First|pool usage

The metric can then be used in health rules as would other types of JMX metrics.

To monitor multiple metrics with the same script file, have the script write a different line for each one to STDOUT, such as the following:

```
name=Custom Metrics|Hardware Resources|Disks|Total Disk Usage %,
value=23
name=Custom Metrics|Hardware Resources|Disks|Disk 1|Current Disk
Usage %, value=56
```

Step 3. Copy the script file to the subdirectory created in Step 1

Ensure that the agent process has execute permissions not only for the script file but also for the contents of the file.

Step 4. Create the monitor.xml file

For each custom monitoring extension script create a monitor.xml file. The monitor.xml file executes the script file created in Step 2. You can edit the following sample file to create your file.

```
<monitor>
    <name>HardwareMonitor</name>
       <type>managed</type>
        <description>Monitors system resources - CPU, Memory, Network
I/O, and Disk I/O.</description>
        <monitor-configuration>     </monitor-configuration>
        <monitor-run-task>
           <!--  Edit execution-style as needed. -->
           <execution-style>continuous</execution-style>
           <name>Run</name>
           <type>executable</type>
           <task-arguments></task-arguments>
            <executable-task>
                <type>file</type>
                <!--  Use only one file element per os-type. -->
                 <file os-type="linux">linux-stat.sh</file>
                 <file os-type="mac">macos-stat.sh</file>
                 <file os-type="windows">windows-stat.bat</file>
                 <file os-type="solaris">solaris-stat.sh</file>
                 <file os-type="sunos">solaris-stat.sh</file>
                 <file os-type="aix">aix-stat.sh</file>
           </executable-task>
         </monitor-run-task>
</monitor>
```

The os-type attribute is optional for the executable-task file element when only one os-type is specified. One monitor.xml file executes one script per os-type.

1. Select the execution style from one of the following:

Execution Style	Description	Example

continuous	Choose "continuous", if you want data collection averaged over time, for example, average CPU usage over a minute.
	For the monitor to be declared as 'continuous', the script should also run in an infinite loop. This ensures that the script keeps running until the Standalone Machine Agent process is terminated.
		`while [ 1 ]; do`
		... the actual script goes here ...
		`sleep 60`
		`done`

periodic	Choose periodic to report data from system performance counters periodically. The periodic task runs every minute by default and the data is aggregated.
	
	To specify a different frequency, use the execution-frequency-in-seconds element. The execution frequency must be less than 60. For periodic execution style, you can also specify the timeout setting as shown in the example.
		`<monitor-run-task>`
		
		`...`
		
		`<execution-style>periodic</execution-style>`
		`<execution-frequency-in-seconds>30</execution-frequency-i`
		`<execution-timeout-in-secs>30</execution-timeout-in-secs>`
		
		`...`
		
		`</monitor-run-task>`

2. Add the name of your script file to the `<file>` element in the monitor.xml file. Be sure to use the correct `os-type` attribute. The `os-type` value should match the value returned from calling `System.getProperty("os.name")`. See Unknown macro: 'link-window' (link opens in a new window).

```
<file os-type="your-os-type">{script file name}</file>
```

You can use either the relative or absolute path of the script.

Step 5. Copy the monitor.xml file to the subdirectory created in Step 1

Step 6. Restart the Standalone Machine Agent

> **Required Agent Properties**
> Ensure that you have correctly configured the agent in the controller-info.xml file and on the agent start command on the command line. For information on configuring required and optional agent properties, see Database Agent Configuration Properties.

After restarting the Standalone Machine Agent, you should see following message in your log file:

```
Executing script [<script_name>] on the console to make sure your
changes work with the machine agent.
```

Step 7. Verify execution of the monitoring extension script

To verify the execution of extension, wait for at least one minute and check the metric data in the Metric Browser.

You can now create alerts based on any of these metrics.

Example: Create a monitoring extension for open files

This section provides instructions to create a custom monitor for monitoring all the open files for JVMs.

1. Create a new directory in the custom monitor repository.
2. Create the script file.

   This is a sample script. Modify this script for the specific process name (for example: Author, Publish, and so on).

```
lookfor="<process name 1>"
pid=`ps aux | grep "$lookfor" | grep -v grep | tr -s " " | cut
-f2 -d' '`
count1=`lsof -p $pid | wc -l | xargs`

lookfor="<process name 2>"
pid=`ps aux | grep "$lookfor" | grep -v grep | tr -s " " | cut
-f2 -d' '`
count2=`lsof -p $pid | wc -l | xargs`

echo "name=JVM|Files|<process name 1>,value="$count1
echo "name=JVM|Files|<process name 2>,value="$count2
```

3. Create the monitor.xml and point this XML file to the script file created in step 2.

```
<monitor>
    <name>MyMonitors</name>
    <type>managed</type>
    <description>Monitor open file count </description>
    <monitor-configuration>
    </monitor-configuration>
    <monitor-run-task>
        <execution-style>continuous</execution-style>
        <name>Run</name>
        <type>executable</type>
        <task-arguments>
        </task-arguments>
        <executable-task>
            <type>file</type>
            <file>openfilecount.sh</file>
        </executable-task>
    </monitor-run-task>
</monitor>
```