vRealize Hyperic Configuration Guide

vRealize Hyperic 5.8.4

vmware®

You can find the most up-to-date technical documentation on the VMware website at:

https://docs.vmware.com/

If you have comments about this documentation, submit your feedback to

docfeedback@vmware.com

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Contents

About vRealize Hyperic Configuration Guide

The *vRealize Hyperic Configuration Guide* provides information about configuring and running the vRealize Hyperic agent and server.

Configuring and Running a vRealize Hyperic Agent

1

There are various options that you can use to run a vRealize Hyperic agent. You can customize certain parameters to suit your environment requirements.

- Using a Command Line to Run the Agent Launcher

  You can use the command line to initiate the vRealize Hyperic agent launcher and agent lifestyle commands.

- Running the Agent Launcher from the vRealize Hyperic User Interface

  In vRealize Hyperic, you can issue selected commands to a running vRealize Hyperic agent.

- Run the vRealize Hyperic Agent Without a Java Service Wrapper

  If you run a If you run an Hyperic agent on a system that does not support the Java Service Wrapper, or for other reasons prefer not to use the wrapper, you can start the agent without the wrapper.

- Configuring Agent Logging

  You can configure the name, location, and logging level for vRealize Hyperic agent logs. You can also redirect system messages to the agent log, and configure the debug log level for an agent subsystem.

- Configuring Plug-in Loading

  At startup, a vRealize Hyperic agent loads all the plug-ins in the `AgentHome/bundles/agent-x.y.z-nnnn/pdk/plugins` directory. You can reduce the agent's memory footprint by configuring it to load only the plug-ins that you use.

- Configuring an Agent to Enable a Resource Plug-in

  You can configure a vRealize Hyperic agent to enable a specific plug-in to perform one or more of its management functions.

- Managing the vRealize Hyperic Agent

  You can monitor your vRealize Hyperic agents and tune them to your requirements. You can view the status of all the agents, view the metrics of a specific agent and reduce the memory footprint of an agent.

- Agent Properties

  Multiple properties are support in the `agent.properties` file for a vRealize Hyperic agent. Not all supported properties are included by default in the `agent.properties` file.

- Configuring the Agent Java Service Wrapper

  The configuration file for the vRealize Hyperic agent;s Java service wrapper is located in `AgentHome/bundles/BundleHome/conf/wrapper.conf`.

- Tailoring the Agent Wrapper Configuration

  The Java options that are supplied to the vRealize Hyperic agent at startup are configured in the `Java Additional Parameters` section of the `wrapper.conf` file.

- Automated Deployment of Multiple vRealize Hyperic Agents

  You can deploy multiple vRealize Hyperic agents simultaneously, using vRealize Infrastructure Navigator. You configure the agent properties prior to deployment.

Using a Command Line to Run the Agent Launcher

You can use the command line to initiate the vRealize Hyperic agent launcher and agent lifestyle commands.

- Run the Agent Launcher from a Linux Command Line

  You initiate the agent launcher and agent life cycle commands with the `hq-agent.sh` script in the `AgentHome/bin` directory.

- Run the Agent Launcher from a Windows Command Line

  You initiate the agent launcher and agent life cycle commands with the `hq-agent.bat` script in the `AgentHome/bin` directory.

Run the Agent Launcher from a Linux Command Line

You initiate the agent launcher and agent life cycle commands with the `hq-agent.sh` script in the `AgentHome/bin` directory.

Procedure

1. Open a command shell or terminal window.
2. Type the required command using the format **`sh hq-agent.sh command`** where `command` is one of the following.

Option	Description
start	Starts the agent as a daemon process.
stop	Stops the agent's JVM process.
restart	Stops and then starts the agent's JVM process.
status	Queries the status of the agent's JVM process..
dump	Runs a thread dump for the agent process, and writes the results to the `agent.log` file in `AgentHome/log` .

Option	Description
ping	Pings the agent process.
setup	Causes the vRealize Hyperic agent to prompt you for the agent-server connection properties, enabling you to change values that were provided at first agent startup.

Run the Agent Launcher from a Windows Command Line

You initiate the agent launcher and agent life cycle commands with the `hq-agent.bat` script in the `AgentHome/bin` directory.

Procedure

1. Open a terminal window.
2. Type the required command using the format **`hq-agent.bat command`** where `command` is one of the following.

Option	Description
start	Starts the agent as an NT service.
stop	Stops the agent as an NT service.
restart	Stops and then starts the agent's JVM process.
install	Installs the agent NT service
remove	Removes the agent's service from the NT service table
query	Queries the current status of the agent NT service (status)
ping	Pings the agent process.
setup	Prompts for setup configuration for the agent process.

Running the Agent Launcher from the vRealize Hyperic User Interface

In vRealize Hyperic, you can issue selected commands to a running vRealize Hyperic agent.

Agent control commands are available on the **Views** tab for a vRealize Hyperic agent or a group of agents in inventory.

- Restart an Agent from the Hyperic User Interface

  You can use the **restart** action to invoke the restart command in the Java Service Wrapper of the vRealize Hyperic agent.

- Ping an Agent from the vRealize Hyperic User Interface

  You can ping an agent or group of agents to check connectivity.

- Upgrade an Agent from the vRealize Hyperic User Interface

  You can upgrade agents directly from the vRealize Hyperic user interface.

- Push a Resource Plug-in to an Agent from the vRealize Hyperic User Interface

  The **push plugin** action sends new and changed resource plug-ins to the target agent or agents.

Restart an Agent from the Hyperic User Interface

You can use the **restart** action to invoke the restart command in the Java Service Wrapper of the vRealize Hyperic agent.

The **restart** action shuts down the JVM process in which the agent runs, waits for the process to terminate cleanly, and generates a new JVM process for the agent. During the restart process, the agent's metric collection and resource control functionality is interrupted.

The **restart** action occurs asynchronously. To verify that the restart succeeded go to the page for the agent in the vRealize Hyperic user interface and check its availability. Alternatively, you could configure an alert for the agent that is triggered when the agent's availability changes.

Procedure

1. On the **Views** tab for a vRealize Hyperic agent or group of agents, select **restart** from the drop-down menu.
2. Click **Execute**.

The agent restarts.

Ping an Agent from the vRealize Hyperic User Interface

You can ping an agent or group of agents to check connectivity.

Procedure

1. On the **Views** tab for a vRealize Hyperic agent or group of agents, select **ping** from the drop-down menu.
2. Click **Execute**.

Example:

What to do next

Upgrade an Agent from the vRealize Hyperic User Interface

You can upgrade agents directly from the vRealize Hyperic user interface.

Procedure

1. On the **Views** tab for a vRealize Hyperic agent or group of agents, select **upgrade** from the drop-down menu.
2. Select the relevant agent bundle and click **Execute**.

1 The agent bundle is transferred from the vRealize Hyperic server to the target agent or agents.

2 The agent expands the bundle locally.

3 The agent updates the local bundle property.

4 The server restarts the agent.

The configuration properties in the agent's `/conf/agent.properties` file are preserved.

Push a Resource Plug-in to an Agent from the vRealize Hyperic User Interface

The **push plugin** action sends new and changed resource plug-ins to the target agent or agents.

Procedure

1 On the **Views** tab for a vRealize Hyperic agent or group of agents, select **push plugin** from the drop-down menu.

2 Select the required plugin and click **Execute**.

1 The plug-in is transferred from the vRealize Hyperic server to the target agent or agents.

2 The server restarts the agent.

Run the vRealize Hyperic Agent Without a Java Service Wrapper

If you run a If you run an Hyperic agent on a system that does not support the Java Service Wrapper, or for other reasons prefer not to use the wrapper, you can start the agent without the wrapper.

Procedure

- From the `AgentHome/bundles/agent-x.y.z/bin` directory, run the `hq-agent-nowrapper.sh` agent start script using `nohup`.

  ```
  nohup AgentHome/bundles/agent-x.y.z/bin/hq-agent-nowrapper.sh
  ```

Configuring Agent Logging

You can configure the name, location, and logging level for vRealize Hyperic agent logs. You can also redirect system messages to the agent log, and configure the debug log level for an agent subsystem.

- Agent Log Files

  The vRealize Hyperic agent log files are stored in the `AgentHome/log` directory.

- Configuring the Agent Log Name or Location

  Use these properties to change the name or location of the agent log file.

- Configuring the Agent Logging Level

  Use this property to control the severity level of messages that the vRealize Hyperic agent writes to the agent log file.

- Redirecting System Messages to the Agent Log

  You can use these properties to redirect system-generated messages to the vRealize Hyperic agent log file.

- Configuring the Debug Level for an Agent Subsystem

  For troubleshooting purposes, you can increase the logging level for an individual agent subsystem.

Agent Log Files

The vRealize Hyperic agent log files are stored in the `AgentHome/log` directory.

Agent log files include the following:

- `agent.log`
- `agent.startup.log`
- `wrapper.log`

The Java service wrapper-based agent launcher writes messages to the `wrapper.log` file.

Configuring the Agent Log Name or Location

Use these properties to change the name or location of the agent log file.

agent.logDir

You can add this property to the `agent.properties` file to specify the directory where the vRealize Hyperic agent will write its log file. If you do not specify a fully qualified path, `agent.logDir` is evaluated relative to the agent installation directory.

This property does not exist in the `agent.properties` file unless you explicitly add it. The default behavior is equivalent to the `agent.logDir=log` setting, resulting in the agent log file being written to the `AgentHome/log` directory.

To change the location for the agent log file, add `agent.logDir` to the `agent.properties` file and enter a path relative to the agent installation directory, or a fully qualified path.

The name of the agent log file is configured with the `agent.logFile` property.

agent.logFile

This property specifies the path and name of the agent log file.

In the `agent.properties` file, the default setting for the `agent.LogFile` property is made up of a variable and a string, `agent.logFile=${agent.logDir}\`*`agent.logDir`*.

- *agent.logDir* is a variable that supplies the value of an identically named agent property. By default, the value of *agent.logDir* is `log`, interpreted relative to the agent installation directory.
- `agent.log` is the name for the agent log file.

By default, the agent log file is named `agent.log` and is written to the `AgentHome/log` directory.

To configure the agent to log to a different directory, you must explicitly add the `agent.logDir` property to the `agent.properties` file.

Configuring the Agent Logging Level

Use this property to control the severity level of messages that the vRealize Hyperic agent writes to the agent log file.

agent.logLevel

This property specifies the level of detail of the messages that the vRealize Hyperic agent writes to the log file. Available values are `INFO` and `DEBUG`. The default value is `INFO`.

Setting the `agent.logLevel` property value to `DEBUG` level is not advised. This level of logging across all subsystems imposes overhead, and can also cause the log file to roll over so frequently that log messages of interest are lost. It is preferable to configure debug level logging only at the subsystem level.

Redirecting System Messages to the Agent Log

You can use these properties to redirect system-generated messages to the vRealize Hyperic agent log file.

agent.logLevel.SystemErr

This property redirects `System.err` to `agent.log`. Commenting out this setting causes `System.err` to be directed to `agent.log.startup`.

The default value is `ERROR`.

agent.logLevel.SystemOut

This property redirects `System.out` to `agent.log`. Commenting out this setting causes `System.out` to be directed to `agent.log.startup`.

The default value is `INFO`.

Configuring the Debug Level for an Agent Subsystem

For troubleshooting purposes, you can increase the logging level for an individual agent subsystem.

To increase the logging level for an individual agent subsystem, uncomment the appropriate line in the section of the `agent.properties` file that is labelled `Agent Subsystems: Uncomment individual subsystems to see debug messages`.

Agent log4j Properties

This is the `log4j` properties in the `agent.properties` file.

```
log4j.rootLogger=${agent.logLevel}, R

log4j.appender.R.File=${agent.logFile}
log4j.appender.R.MaxBackupIndex=1
log4j.appender.R.MaxFileSize=5000KB
log4j.appender.R.layout.ConversionPattern=%d{dd-MM-yyyy HH:mm:ss,SSS z} %-5p [%t] [%c{1}@%L] %m%n
log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R=org.apache.log4j.RollingFileAppender

##
## Disable overly verbose logging
##
log4j.logger.org.apache.http=ERROR
log4j.logger.org.springframework.web.client.RestTemplate=ERROR
log4j.logger.org.hyperic.hq.measurement.agent.server.SenderThread=INFO
log4j.logger.org.hyperic.hq.agent.server.AgentDListProvider=INFO
log4j.logger.org.hyperic.hq.agent.server.MeasurementSchedule=INFO
log4j.logger.org.hyperic.util.units=INFO
log4j.logger.org.hyperic.hq.product.pluginxml=INFO

# Only log errors from naming context
log4j.category.org.jnp.interfaces.NamingContext=ERROR
log4j.category.org.apache.axis=ERROR

#Agent Subsystems: Uncomment individual subsystems to see debug messages.
#-------------------------------------------------------------------------
#log4j.logger.org.hyperic.hq.autoinventory=DEBUG
#log4j.logger.org.hyperic.hq.livedata=DEBUG
#log4j.logger.org.hyperic.hq.measurement=DEBUG
#log4j.logger.org.hyperic.hq.control=DEBUG

#Agent Plugin Implementations
#log4j.logger.org.hyperic.hq.product=DEBUG

#Server Communication
#log4j.logger.org.hyperic.hq.bizapp.client.AgentCallbackClient=DEBUG

#Server Realtime commands dispatcher
#log4j.logger.org.hyperic.hq.agent.server.CommandDispatcher=DEBUG

#Agent Configuration parser
#log4j.logger.org.hyperic.hq.agent.AgentConfig=DEBUG

#Agent plugins loader
#log4j.logger.org.hyperic.util.PluginLoader=DEBUG

#Agent Metrics Scheduler (Scheduling tasks definitions & executions)
#log4j.logger.org.hyperic.hq.agent.server.session.AgentSynchronizer.SchedulerThread=DEBUG

#Agent Plugin Managers
```

```
#log4j.logger.org.hyperic.hq.product.MeasurementPluginManager=DEBUG
#log4j.logger.org.hyperic.hq.product.AutoinventoryPluginManager=DEBUG
#log4j.logger.org.hyperic.hq.product.ConfigTrackPluginManager=DEBUG
#log4j.logger.org.hyperic.hq.product.LogTrackPluginManager=DEBUG
#log4j.logger.org.hyperic.hq.product.LiveDataPluginManager=DEBUG
#log4j.logger.org.hyperic.hq.product.ControlPluginManager=DEBUG
```

Configuring Plug-in Loading

At startup, a vRealize Hyperic agent loads all the plug-ins in the `AgentHome/bundles/agent-x.y.z-nnnn/pdk/plugins` directory. You can reduce the agent's memory footprint by configuring it to load only the plug-ins that you use.

You can either specify a list of plug-ins to exclude, or configure a list of plug-ins to load.

plugins.exclude

Use this property to specify the plug-ins that the vRealize Hyperic agent must not load at startup. This property is useful for reducing an agent's memory footprint.

You supply a comma-separated list of plugins to exclude. For example, `plugins.exclude=jboss,apache,mysql`.

plugins.include

Use this property to specify the plug-ins that the vRealize Hyperic agent must load at startup. This property is useful for reducing an agent's memory footprint.

You supply a comma-separated list of plugins to include. For example, `plugins.include=weblogic,apache`.

Configuring an Agent to Enable a Resource Plug-in

You can configure a vRealize Hyperic agent to enable a specific plug-in to perform one or more of its management functions.

- Configuring Agent Account Privileges under Solaris 10

  To auto-discover certain products under Solaris 10, the vRealize Hyperic agent must run as `root`, or you must grant additional permissions to the account where the agent runs.
- Configuring the Agent HTTP Request Header

  If you monitor a remote HTTP server, it is useful to configure the HTTP request header for agent HTTP requests.
- Configuring the Agent to Monitor JBoss

  You can specify the location of the JBoss root directory so that the vRealize Hyperic agent can monitor JBoss.

- Configuring the Data to Log Windows Events

  When log tracking is enabled for a Windows resource, you can use the `platform.log_track.eventfmt` agent property to customize the content of events that the vRealize Hyperic agent logs for Windows events.

Configuring Agent Account Privileges under Solaris 10

To auto-discover certain products under Solaris 10, the vRealize Hyperic agent must run as `root`, or you must grant additional permissions to the account where the agent runs.

Under Solaris 10's Least Privilege Model (LPM), default privileges are minimal. The vRealize Hyperic agent must be able to read `./proc/$pid/` files on the platform.

Problems with auto-discovery on Solaris 10 might be the result of insufficient privileges. Depending on your account privilege implementation you might need to grant the `proc_zone` privilege to the agent account.

For example, you could add the following line to `/etc/user_attr`, to grant the **proc_owner** privilege to the vRealize Hyperic user and to deny the **proc_session** privilege:

```
hq::::type=normal;defaultpriv=basic,proc_owner,!proc_session
```

After changing account privileges, you need to re-login. Your approach tp enabling agent access to `/proc/$pid/` files is dependent on your company's LPM implementation and best practices.

Configuring the Agent HTTP Request Header

If you monitor a remote HTTP server, it is useful to configure the HTTP request header for agent HTTP requests.

http.useragent

The `http.useragent` property defines the value for the `User-Agent` request header in HTTP requests issued by the vRealize Hyperic agent.

By default, the `User-Agent` in agent requests includes the vRealize Hyperic agent version, and so changes when the agent is upgraded. Therefore, if a target HTTP server is configured to block requests with an unknown `User-Agent`, agent requests fail following an agent upgrade.

You can use `http.useragent` to define a `User-Agent` value that is consistent across upgrades.

The `agent.properties` file does not contain this property by default. You must add it to the file.

The default is `Hyperic-HQ-Agent/Version` For example, `Hyperic-HQ-Agent/4.1.2-EE`.

Configuring the Agent to Monitor JBoss

You can specify the location of the JBoss root directory so that the vRealize Hyperic agent can monitor JBoss.

jboss.installpath

To enable the agent to monitor JBoss, specify the location of the JBoss root directory. The default location is `/usr/local/jboss–4.0.0`.

Configuring the Data to Log Windows Events

When log tracking is enabled for a Windows resource, you can use the `platform.log_track.eventfmt` agent property to customize the content of events that the vRealize Hyperic agent logs for Windows events.

platform.log_track.eventfmt

This property specifies the content and format of the Windows event attributes that a vRealize Hyperic agent includes when logging a Windows event as an event in `Hyperic. agent.properties` does not contain the `platform.log_track.eventfmt` property, you must add it if before you can customize the data logged for Windows events.

By default, when Windows log tracking is enabled, an entry in the format `[Timestamp] Log Message (EventLogName):EventLogName:EventAttributes` is logged for events that match the criteria you specified on the resource's Configuration Properties page.

Attribute	Description
`Timestamp`	The time at which the event occurred.
`Log Message`	A text string.
`EventLogName`	The Windows event log type, `System`, `Security`, or `Application`.
`EventAttributes`	A colon-delimited string comprising the Windows event Source and Message attributes.

The following example is for a Windows event that was written to the Windows System event log at 6:06 AM on 04/19/2010. The Windows event Source and Message attributes, are `Print` and `Printer HP LaserJet 6P was paused.`, respectively.

```
04/19/2010 06:06 AM Log Message (SYSTEM): SYSTEM: Print: Printer HP LaserJet 6P was paused.
```

Configuration

You can use the following parameters to configure the Windows event attributes that the agent writes for a Windows event. Each parameter maps to a Windows event attribute of the same name.

Parameter	Description
%user%	The name of the user on whose behalf the event occurred.
%computer%	The name of the computer on which the event occurred.
%source%	The software that logged the Windows event.
%event%	A number identifying the particular event type.
%message%	The event message.
%category%	An application-specific value used for grouping events.

For example, if you set the following properties, `platform.log_track.eventfmt=%user%@%computer% %source%:%event%:%message%`, the vRealize Hyperic agent will write the following data when logging a Windows event.

```
04/19/2010 06:06 AM Log Message (SYSTEM): SYSTEM: HP_Admistrator@Office Print:7:Printer HP LaserJet 6P was paused
```

The entry is for as for a Windows event that was written to the Windows System event log at 6:06 AM on 04/19/2010. The software associated with the event was running as HP_Administrator on the Office host. The Windows event's Source, Event, and Message attributes, are `Print`, `7`, and `Printer HP LaserJet 6P was paused.`, respectively.

Managing the vRealize Hyperic Agent

You can monitor your vRealize Hyperic agents and tune them to your requirements. You can view the status of all the agents, view the metrics of a specific agent and reduce the memory footprint of an agent.

- Viewing the Status of All Agents

  You can view the health status of your vRealize Hyperic agents, including the number of platforms an agent monitors, the number of resource metrics that an agent collects, and the number of licenses consumed by an agent.
- Viewing the Metrics for an Agent

  A vRealize Hyperic agent monitors itself. You can tailor the metric collection settings for an agent, use agent metrics to troubleshoot problems, and base alerts on agent metrics or events.
- View Agent Indicator Charts

  The Indicators page for an agent charts the agent's indicator metrics.
- View Agent Metric Data

  The Metric Data page for an agent displays all of the metrics collected for the agent.
- vRealize Hyperic Agent Metrics

  This table lists the metrics that can be collected for a vRealize Hyperic agent.
- Reducing the Agent Memory Footprint

  There are various options you can employ to reduce the amount of memory an agent uses.

Viewing the Status of All Agents

You can view the health status of your vRealize Hyperic agents, including the number of platforms an agent monitors, the number of resource metrics that an agent collects, and the number of licenses consumed by an agent.

You view the status of all agent that are registered with the vRealize Hyperic server on the **Agents** tab of the HQ Health page.

Health Data for an Agent

The data that you can view is described in the following table.

Table 1-1. Agent Health Data

Field	Description	Notes
FQDN	Fully-qualified domain name of the platform where the agent runs.	
Address	IP address of the platform where the agent runs.	
Port	Port where the agent listens for communication with the vRealize Hyperic server.	If you configure unidirectional agent - server communications, the agent initiates all communications with the vRealize Hyperic server.
Version	Agent version number.	Although an agent might work successfully with an vRealize Hyperic server of a later version, it is strongly recommended that you run the same version of the agent and server.
Build #	Agent build number.	
Bundle Version	Agent bundle version.	
Creation Time	The date/time that the vRealize Hyperic agent was first started up.	
# Platforms	Number of platforms the agent manages.	Typically, an agent manages one platform - the platform where it runs. Exceptions include: ■ If the agent manages a vSphere vCenter instance, the number of platforms shown is the number of virtual machines the vCenter server manages. ■ If the agent manages remote network devices or network host platform types.
# Metrics	The number of resource metrics the agent collects. This is the total number of metrics that are configured for collection across all resources the agent monitors.	If one agent bears an inordinate metric load, you might be able to distribute it more evenly.

Table 1-1. Agent Health Data (Continued)

Field	Description	Notes
Time Offset (ms)	The difference in system clock time between the agent and the vRealize Hyperic server.	A time offset can cause incorrect availability reporting.
License Count	The number of platform licenses consumed by the agent.	Typically, a single agent consumes a single license. If an agent manages a vCenter vSphere instance, it consumes a license for the platform that hosts vCenter, a license for each vSphere vHost administered by the vCenter instance, and, if an agent is installed in each virtual machine, license for each vSphere virtual machine on each vHost.

Viewing the Metrics for an Agent

A vRealize Hyperic agent monitors itself. You can tailor the metric collection settings for an agent, use agent metrics to troubleshoot problems, and base alerts on agent metrics or events.

The metrics that an agent reports for itself are:

- Availability
- JVM Free Memory
- JVM Total Memory
- Number of Metrics Collected Per Minute
- Number of Metrics Sent to the Server Per Minute
- Server Offset
- Total Time Spend Fetching Metrics per Minute

View Agent Indicator Charts

The Indicators page for an agent charts the agent's indicator metrics.

By default, the indicator metrics include

- JVM Free Memory
- JVM Total Memory
- Number of Metrics Collected Per Minute

Procedure

1. Click **Resources > Browse**.
2. Click **Servers**.

3 Select **HQ Agent** from the **Server Type** menu.

View Agent Metric Data

The Metric Data page for an agent displays all of the metrics collected for the agent.

Procedure

1 Click **Resources > Browse**.

2 Click **Servers**.

3 Select **HQ Agent** from the **Server Type** menu.

vRealize Hyperic Agent Metrics

This table lists the metrics that can be collected for a vRealize Hyperic agent.

Table 1-2. Available Metrics for a vRealize Hyperic Agent .

Category	Metric	Notes
Availability		
	Availability	Collected by default.
	Start Time	
	Up Time	
Throughput		
	Number of Active Threads	
	Number of Metrics Collected	
	Number of Metrics Collected per Minute	By default, this is an indicator metric
	Number of Metrics that Failed to be Collected	
	Number of Metrics that Failed to be Collected per Minute	
	Number of Requests Served	
	Number of Requests Served per Minute	
	Number of Scheduled Metrics	
Performance		
	Maximum Time Spent Fetching a Metric	
	Maximum Time Spent Processing a Request	
	Minimum Time Spent Fetching a Metric	
	Minimum Time Spent Processing a Request	
	Number of Connection Failures	

Table 1-2. Available Metrics for a vRealize Hyperic Agent . (Continued)

Category	Metric	Notes
	Number of Connection Failures per Minute	
	Number of Metric Batches Sent to Server	
	Number of Metric Batches Sent to Server per Minute	
	Number of Metrics Sent to Server	
	Number of Metrics Sent to Server per Minute	Collected by default.
	Server Offset	Collected by default.
	Total Time Spent Fetching Metrics	
	Total Time Spent Fetching Metrics per Minute	Collected by default. High value can indicate overloaded agent or problem with scheduling thread.
	Total Time Spent Processing Requests	
	Total Time Spent Processing Requests per Minute	
	Total Time Spent Sending Metrics to Server	
	Total Time Spent Sending Metrics to Server per Minute	
Utilization		
	Cpu Total Time	
	Cpu Total Time per Minute	
	JVM Free Memory	By default, this is an indicator metric.
	JVM Total Memory	By default, this is an indicator metric.
	Open File Descriptors	
	Resident Memory Used	Resident Memory" is the amount of memory the Hyperic Agent occupies in memory
	Time Spent in System Mode	
	Time Spent in System Mode per Minute	
	Time Spent in User Mode	
	Time Spent in User Mode per Minute	
	Total Memory Used	

Reducing the Agent Memory Footprint

There are various options you can employ to reduce the amount of memory an agent uses.

Limit Plug-in Loading

The best way to reduce an agent's footprint is to configure it to load only the plug-ins for the resource types you want to monitor. See Configuring Plug-in Loading.

Reduce Java Heap

To reduce the Java heap size that an agent allocates to itself on startup, add the `agent.javaOpts` property to the agent's `agent.properties` file. This property does not exist in the default `agent.properties` file.

You can reduce the heap from 128m to 64m.

Delete JavaDocs File

In an environment in which every MB is critical, you can delete the agent's javadocs folder, `agent-4.x.x/bundles/agent-4.x.x-yyyy/pdk/javadoc;`. Note that this action only reduces the agent footprint by (approximately) 70 MB.

Agent Properties

Multiple properties are support in the `agent.properties` file for a vRealize Hyperic agent. Not all supported properties are included by default in the `agent.properties` file.

You must add any properties that you want to use that are not included in the default `agent.properties` file.

Following is a list of the available properties.

agent.eventReportBatchSize Property

This property specifies the maximum number of events that a vRealize Hyperic agent sends per contact with the server.

Default

By default, the `agent.properties` file does not include this property.

The default behavior of the agent is to send a maximum of 100 events per contact with the server.

agent.keystore.alias Property

This property configures the name of the user-managed keystore for the agent for agents configured for unidirectional communication with the vRealize Hyperic server.

Example: Defining the Name of a Keystore

Given this user-managed keystore for a unidirectional agent

```
hq self-signed cert), Jul 27, 2011, trustedCertEntry,
Certificate fingerprint (MD5): 98:FF:B8:3D:25:74:23:68:6A:CB:0B:9C:20:88:74:CE
hq-agent, Jul 27, 2011, PrivateKeyEntry,
Certificate fingerprint (MD5): 03:09:C4:BC:20:9E:9A:32:DC:B2:E8:29:C0:3C:FE:38
```

you define the name of the keystore like this

```
agent.keystore.alias=hq-agent
```

If the value of this property does not match the keystore name, agent-server communication fails.

Default

The default behavior of the agent is to look for the `hq` keystore.

For unidirectional agents with user-managed keystores, you must define the keystore name using this property.

agent.keystore.password Property

This property configures the password for a vRealize Hyperic agent's SSL keystore.

Define the location of the keystore using the agent.keystore.path Property property.

By default, the first time you start the vRealize Hyperic agent following installation, if `agent.keystore.password` is uncommented and has a plain text value, the agent automatically encrypts the property value. You can encrypt this property value yourself, prior to starting the agent.

It is good practice to specify the same password for the agent keystore as for the agent private key.

Default

By default, the `agent.properties` file does not include this property.

agent.keystore.path Property

This property configures the location of a vRealize Hyperic agent's SSL keystore.

Specify the full path to the keystore. Define the password for the keystore using the `agent.keystore.password` property. See agent.keystore.password Property.

Specifying the Keystore Path on Windows

On Windows platforms, specifiy the path to the keystore in this format.

```
C:/Documents and Settings/Desktop/keystore
```

Default

`AgentHome/data/keystore`.

agent.listenIp Property

The IP address to which the agent binds at startup.

Default

The default value allows the agent to listen on all IP addresses on the agent host. This behavior is equivalent to setting the property to an asterisk.

agent.logDir Property

You can add this property to the `agent.properties` file to specify the directory where the vRealize Hyperic agent writes its log file. If you do not specify a fully qualified path, `agent.logDir` is evaluated relative to the agent installation directory.

To change the location for the agent log file, enter a path relative to the agent installation directory, or a fully qualified path.

Note that the name of the agent log file is configured with the `agent.logFile` property.

Default

By default, the `agent.properties` file does not include this property.

The default behavior is `agent.logDir=log`, resulting in the agent log file being written to the `AgentHome/log` directory.

agent.logFile Property

The path and name of the agent log file.

Default

In the `agent.properties` file, the default setting for the `agent.LogFile` property is made up of a variable and a string

```
agent.logFile=${agent.logDir}\agent.log
```

where

- *agent.logDir* is a variable that supplies the value of an identically named agent property. By default, the value of *agent.logDir* is `log`, interpreted relative to the agent installation directory.
- `agent.log` is the name for the agent log file.

By default, the agent log file is named `agent.log`, and is written to the `AgentHome/log` directory.

agent.logLevel Property

The level of detail of the messages the Agent writes to the log file.

Allowable values are `INFO` and `DEBUG`.

Default

`INFO`

agent.logLevel.SystemErr Property

Redirects `System.err` to the `agent.log` file.

Commenting out this setting causes `System.err` to be directed to `agent.log.startup`.

Default

`ERROR`

agent.logLevel.SystemOut Property

Redirects `System.out` to the `agent.log` file.

Commenting out this setting causes `System.out` to be directed to `agent.log.startup`.

Default

`INFO`

agent.maxBatchSize Property

The maximum number of metrics that the agent will send per contact with the server.

Default

The default behavior of the agent is to send a maximum of 500 per contact with the server.

By default, the `agent.properties` file does not include this property.

agent.proxyHost Property

The host name or IP address of the proxy server that the vRealize Hyperic agent must connect to first when establishing a connection to the vRealize Hyperic server.

This property is supported for agents configured for unidirectional communication.

Use this property in conjunction with `agent.proxyPort` and `agent.setup.unidirectional`.

Default

None

agent.proxyPort Property

The port number of the proxy server that the vRealize Hyperic agent must connect to first when establishing a connection to the vRealize Hyperic server.

This property is supported for agents configured for unidirectional communication.

Use this property in conjunction with `agent.proxyPort` and `agent.setup.unidirectional`.

Default

None

agent.setup.acceptUnverifiedCertificate Property

This property controls whether or not a vRealize Hyperic agent issues a warning when the vRealize Hyperic server presents an SSL certificate that is not in the agent's keystore and is either self-signed or signed by a different certificate authoridy than the one that signed the agent's SSL certificate.

When the default is used, the agent issues the warning

```
The authenticity of host 'localhost' can't be established.
Are you sure you want to continue connecting? [default=no]:
```

If you respond **`yes`**, the agent imports the server's certificate and will continue to trust the certificate from this point on.

Default

`agent.setup.acceptUnverifiedCertificate=false`

agent.setup.camIP Property

Use this property to define the IP address of the vRealize Hyperic server for the agent. The vRealize Hyperic agent reads this value only in the event that it cannot find connection configuration in its data directory.

You can specify this and other `agent.setup.*` properties to reduce the user interaction required to configure an agent to communicate with the server.

The value can be provided as an IP address or a fully qualified domain name. To identify an server on the same host as the server, set the value to 127.0.0.1.

If there is a firewall between the agent and server, specify the address of the firewall, and configure the firewall to forward traffic on port 7080, or 7443 if you use the SSL port, to the vRealize Hyperic Server.

Default

Commented out, `localhost`.

agent.setup.camLogin Property

At first startup after installation, use this property to define the vRealize Hyperic agent username to use when the agent is registering itself with the server.

The permission required on the server for this initialization is `Create`, for Platforms.

Login from the agent to the server is only required during the initial configuration of the agent.

The agent reads this value only in the event that it cannot find connection configuration in its data directory.

You can specify this and other `agent.setup.*` properties to reduce the user interaction required to configure an agent to communicate with the server.

Default

Commented our `hqadmin`.

agent.setup.camPort Property

At first startup after installation, use this property to define the vRealize Hyperic agent server port to use for non-secure communications with the server.

The agent reads this value only in the event that it cannot find connection configuration in its data directory.

You can specify this and other `agent.setup.*` properties to reduce the user interaction required to configure an agent to communicate with the server.

Default

Commented out `7080`.

agent.setup.camPword Property

Use this property to define the password that the vRealize Hyperic agent uses when connecting to the vRealize Hyperic server, so that the agent does not prompt a user to supply the password interactively at first startup.

. (The password for the user is that specified by `agent.setup.camLogin`.

The agent reads this value only in the event that it cannot find connection configuration in its data directory.

You can specify this and other `agent.setup.*` properties to reduce the user interaction required to configure an agent to communicate with the server.

The first time you start the vRealize Hyperic agent after installation, if `agent.keystore.password` is uncommented and has a plain text value, the agent automatically encrypts the property value. You can encrypt these property values prior to starting the agent.

Default

Commented our `hqadmin`.

agent.setup.camSSLPort Property

At first startup after installation, use this property to define the vRealize Hyperic agent server port to use for SSL communications with the server.

The agent reads this value only in the event that it cannot find connection configuration in its data directory.

You can specify this and other `agent.setup.*` properties to reduce the user interaction required to configure an agent to communicate with the server.

Default

Commented out `7443`.

agent.setup.agentIP Property

Specifies the IP address that the vRealize Hyperic server uses to contact the vRealize Hyperic agent.

This If the agent is on the same host as the server, a value of `127.0.0.1` is valid.

If there is a firewall between the server and agent, specify the IP address of the firewall, and configure the firewall to forward traffic intended for the agent to the agent's listen address, which can be configured with `agent.listenIP`.

The agent reads this value only in the event that it cannot find connection configuration in its data directory.

You can specify this and other `agent.setup.*` properties to reduce the user interaction required to configure an agent to communicate with the server.

Default

Commented out `default`.

If you use the agent.setup.* properties to supply an agent's configuration at first startup, then uncomment this property, leaving the value `default`, the vRealize Hyperic server contacts the agent using the IP address that SIGAR detects on the agent host.

agent.setup.agentPort Property

This property specifies the port (on the IP address configured with `agent.setup.agentIP`) on the vRealize Hyperic agent on which the vRealize Hyperic server communicates with the agent.

If there is a firewall between the agent and the server, set agent.setup.agentPort to the appropriate port on the firewall, and configure the firewall to forward traffic intended for the agent to the agent listen port.

The agent reads this value only in the event that it cannot find connection configuration in its data directory.

You can specify this and other `agent.setup.*` properties to reduce the user interaction required to configure an agent to communicate with the server.

Default

Commented out `default`.

If you use the agent.setup.* properties to supply an agent's configuration at first startup, then uncomment this property, leaving the value `default`, the vRealize Hyperic server contacts the agent on port 2144, unless SIGAR detects it is not available, in which case another default is selected.

agent.setup.resetupToken Property

Use this property to configure a vRealize Hyperic agent to create a new token to use for authentication with the server at startup. Regenerating a token is useful if the Agent cannot connect to the server because the token has been deleted or corrupted.

The agent reads this value only in the event that it cannot find connection configuration in its data directory.

Regardless of the value of this property, an agent generates a token the first time it is started after installation.

Default

Commented out `no`.

agent.setup.unidirectional Property

Enables unidirectional communications between the vRealize Hyperic agent and vRealize Hyperic server.

If you configure an agent for unidirectional communication, all communication with the server is initiated by the agent.

For a unidirectional agent with a user-managed keystore, you must configure the keystore name in the `agent.properties` file.

Default

Commented out `no`.

agent.startupTimeOut Property

The number of seconds that the agent startup script waits before determining that the agent has not started up successfully. If the agent is determined to not be listening for requests within this period, an error is logged, and the startup script times out.

Default

By default, the `agent.properties` file does not include this property.

The default behavior of the agent is to timeout after 300 seconds.

autoinventory.defaultScan.interval.millis Property

Specifies how frequently the agent performs a default autoinventory scan.

The default scan detects servers and platform services, typically using the process table or the Windows registry. Default scans are less resource-intensive than runtime scans.

Default

The agent performs the default scan at startup and every 15 minutes thereafter.

Commented out `86,400,000` milliseconds, or one day.

autoinventory.runtimeScan.interval.millis Property

Specifies how frequently the agent performs a runtime scan.

A runtime scan may use more resource-intensive methods to detect services than a default scan. For example, a runtime scan might involve issuing an SQL query or looking up an MBean.

Default

`86,400,000` milliseconds, or one day.

http.useragent Property

Defines the value for the user-agent request header in HTTP requests issued by the vRealize Hyperic agent.

You can use `http.useragent` to define a user-agent value that is consistent across upgrades.

By default, the `agent.properties` file does not include this property.

Default

By default, the user-agent in agent requests includes the vRealize Hyperic Agent version, and so changes when the agent is upgraded. If a target HTTP server is configured to block requests with an unknown user-agent, agent requests fail after an agent upgrade.

`Hyperic-HQ-Agent/Version`, for example, `Hyperic-HQ-Agent/4.1.2-EE`.

log4j Properties

The `log4j` properties for vRealize Hyperic are described here.

```
log4j.rootLogger=${agent.logLevel}, R

log4j.appender.R.File=${agent.logFile}
log4j.appender.R.MaxBackupIndex=1
log4j.appender.R.MaxFileSize=5000KB
log4j.appender.R.layout.ConversionPattern=%d{dd-MM-yyyy HH:mm:ss,SSS z} %-5p [%t] [%c{1}@%L] %m%n
log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R=org.apache.log4j.RollingFileAppender

##
## Disable overly verbose logging
##
log4j.logger.org.apache.http=ERROR
log4j.logger.org.springframework.web.client.RestTemplate=ERROR
log4j.logger.org.hyperic.hq.measurement.agent.server.SenderThread=INFO
log4j.logger.org.hyperic.hq.agent.server.AgentDListProvider=INFO
log4j.logger.org.hyperic.hq.agent.server.MeasurementSchedule=INFO
log4j.logger.org.hyperic.util.units=INFO
log4j.logger.org.hyperic.hq.product.pluginxml=INFO

# Only log errors from naming context
log4j.category.org.jnp.interfaces.NamingContext=ERROR
log4j.category.org.apache.axis=ERROR

#Agent Subsystems: Uncomment individual subsystems to see debug messages.
#-------------------------------------------------------------------------
#log4j.logger.org.hyperic.hq.autoinventory=DEBUG
#log4j.logger.org.hyperic.hq.livedata=DEBUG
#log4j.logger.org.hyperic.hq.measurement=DEBUG
#log4j.logger.org.hyperic.hq.control=DEBUG

#Agent Plugin Implementations
```

```
#log4j.logger.org.hyperic.hq.product=DEBUG

#Server Communication
#log4j.logger.org.hyperic.hq.bizapp.client.AgentCallbackClient=DEBUG

#Server Realtime commands dispatcher
#log4j.logger.org.hyperic.hq.agent.server.CommandDispatcher=DEBUG

#Agent Configuration parser
#log4j.logger.org.hyperic.hq.agent.AgentConfig=DEBUG

#Agent plugins loader
#log4j.logger.org.hyperic.util.PluginLoader=DEBUG

#Agent Metrics Scheduler (Scheduling tasks definitions & executions)
#log4j.logger.org.hyperic.hq.agent.server.session.AgentSynchronizer.SchedulerThread=DEBUG

#Agent Plugin Managers
#log4j.logger.org.hyperic.hq.product.MeasurementPluginManager=DEBUG
#log4j.logger.org.hyperic.hq.product.AutoinventoryPluginManager=DEBUG
#log4j.logger.org.hyperic.hq.product.ConfigTrackPluginManager=DEBUG
#log4j.logger.org.hyperic.hq.product.LogTrackPluginManager=DEBUG
#log4j.logger.org.hyperic.hq.product.LiveDataPluginManager=DEBUG
#log4j.logger.org.hyperic.hq.product.ControlPluginManager=DEBUG
```

jboss.installpath

Specifies the location of the JBoss root directory to enable the agent to monitor JBoss.

Default

`/usr/local/jboss-4.0.0`

platform.log_track.eventfmt Property

Specifies the content and format of the Windows event attributes that a vRealize Hyperic agent includes when logging a Windows event as an event in vRealize Hyperic.

By default, the `agent.properties` file does not include this property.

Default

When Windows log tracking is enabled, an entry in the form `[Timestamp] Log Message (EventLogName):EventLogName:EventAttributes` is logged for events that match the criteria you specified on the resource's Configuration Properties page.

Attribute	Description
`Timestamp`	When the event occurred
`Log Message`	A text string

Attribute	Description
`EventLogName`	The Windows event log type `System`, `Security`, or `Application`
`EventAttributes`	A colon delimited string made of the Windows event Source and Message attributes

For example, the log entry: `04/19/2010 06:06 AM Log Message (SYSTEM): SYSTEM: Print: Printer HP LaserJet 6P was paused.` is for a Windows event written to the Windows System event log at 6:06 AM on 04/19/2010. The Windows event Source and Message attributes, are "`Print`" and "`Printer HP LaserJet 6P was paused.`", respectively.

Configuration

Use the following parameters to configure the Windows event attributes that the agent writes for a Windows event. Each parameter maps to Windows event attribute of the same name.

Parameter	Description
`%user%`	The name of the user on whose behalf the event occurred.
`%computer%`	The name of the computer on which the event occurred.
`%source%`	The software that logged the Windows event.
`%event%`	A number identifying the particular event type.
`%message%`	The event message.
`%category%`	An application-specific value used for grouping events.

For example, with the property setting `platform.log_track.eventfmt=%user%@%computer% %source%:%event%:%message%`, the vRealize Hyperic agent writes the following data when logging the Windows event `04/19/2010 06:06 AM Log Message (SYSTEM): SYSTEM: HP_Admistrator@Office Print: 7:Printer HP LaserJet 6P was paused.`. This entry is for a Windows event written to the Windows system event log at 6:06 AM on 04/19/2010. The software associated with the event was running as "HP_Administrator" on the host "Office". The Windows event's Source, Event, and Message attributes, are "Print", "7", and "Printer HP LaserJet 6P was paused.", respectively.

plugins.exclude Property

Specifies plug-ins that the vRealize Hyperic agent does not load at startup. This is useful for reducing an agent's memory footprint.

Usage

Supply a comma-separated list of plug-ins to exclude. For example,

```
plugins.exclude=jboss,apache,mysql
```

plugins.include Property

Specifies plug-ins that the vRealize Hyperic agent loads at startup. This is useful for reducing the agent's memory footprint.

Usage

Supply a comma-separated list of plug-ins to include. For example,

```
plugins.include=weblogic,apache
```

postgresql.database.name.format Property

This property specifies the format of the name that the PostgreSQL plug-in assigns to auto-discovered `PostgreSQL Database` and `vPostgreSQL Database` database types.

By default, the name of a PostgreSQL or vPostegreSQL database is `Database` *`DatabaseName`*, where *DatabaseName* is the auto-discovered name of the database.

To use a different naming convention, define `postgresql.database.name.format`. The variable data you use must be available from the PostgreSQL plug-in.

Use the following syntax to specify the default table name assigned by the plug-in,

```
Database ${db}
```

where

`postgresql.db` is the auto-discovered name of the `PostgreSQL` or `vPostgreSQL` database.

Default

By default, the `agent.properties` file does not include this property.

postgresql.index.name.format Property

This property specifies the format of the name that the PostgreSQL plug-in assigns to auto-discovered `PostgreSQL Index` and `vPostgreSQL Index` index types.

By default, the name of a PostgreSQL or vPostegreSQL index is `Index` *`DatabaseName.Schema.Index`*, comprising the following variables

Variable	Description
`DatabaseName`	The auto-discovered name of the database.
`Schema`	The auto-discovered schema for the database.
`Index`	The auto-discovered name of the index.

To use a different naming convention, define `postgresql.index.name.format`. The variable data you use must be available from the PostgreSQL plug-in.

Use the following syntax to specify the default index name assigned by the plug-in,

```
Index ${db}.${schema}.${index}
```

where

Attribute	Description
db	Identifies the platform that hosts the PostgreSQL or vPostgreSQL server.
schema	Identifies the schema associated with the table.
index	The index name in PostgreSQL.

Default

By default, the *agent.properties* file does not include this property.

postgresql.server.name.format Property

This property specifies the format of the name that the PostgreSQL plug-in assigns to auto-discovered PostgreSQL and vPostgreSQL server types.

By default, the name of a PostgreSQL or vPostegreSQL server is *Host:Port*, comprising the following variables

Variable	Description
Host	The FQDN of the platform that hosts the server.
Port	The PostgreSQL listen port.

To use a different naming convention, define postgresql.server.name.format. The variable data you use must be available from the PostgreSQL plug-in.

Use the following syntax to specify the default server name assigned by the plug-in,

```
${postgresql.host}:${postgresql.port}
```

where

Attribute	Description
postgresql.host	Identifies the FQDN of the hosting platform.
postgresql.port	Identifies the database listen port.

Default

By default, the *agent.properties* file does not include this property.

postgresql.table.name.format Property

This property specifies the format of the name that the PostgreSQL plug-in assigns to auto-discovered `PostgreSQL Table` and `vPostgreSQL Table` table types.

By default, the name of a PostgreSQL or vPostegreSQL table is `Table` *`DatabaseName.Schema.Table`*, comprising the following variables

Variable	Description
`DatabaseName`	The auto-discovered name of the database.
`Schema`	The auto-discovered schema for the database.
`Table`	The auto-discovered name of the table.

To use a different naming convention, define `postgresql.table.name.format`. The variable data you use must be available from the PostgreSQL plug-in.

Use the following syntax to specify the default table name assigned by the plug-in,

```
Table ${db}.${schema}.${table}
```

where

Attribute	Description
`db`	Identifies the platform that hosts the PostgreSQL or vPostgreSQL server.
`schema`	Identifies the schema associated with the table.
`table`	The table name in PostgreSQL.

Default

By default, the `agent.properties` file does not include this property.

scheduleThread.cancelTimeout Property

The maximum time, in milliseconds, the ScheduleThread will allow a metric collection process to run before attempting to interrupt it.

When the timeout is exceeded, collection of the metric is interrupted, if it is in a `wait()`, `sleep()` or non-blocking `read()` state.

Usage

```
scheduleThread.cancelTimeout=5000
```

Default

`5000` milliseconds.

scheduleThread.fetchLogTimeout Property

The property controls when a warning message is issued for a long-running metric collection process.

If a metric collection process exceeds the value of this property, measured in milliseconds, the agent writes a warning message to the `agent.log` file.

Usage

`scheduleThread.fetchLogTimeout=2000`

Default

`2000` milliseconds.

scheduleThread.poolsize Property

This property allows a plug-in to use multiple threads for metric collection. The property can increase metric throughput for plug-ins known to be thread safe.

Usage

Specify the plug-in by name and the number of threads to allocate for metric collection

```
scheduleThread.poolsize.PluginName=2
```

where *`PluginName`* is the name of the plug-in to which you are allocating threads. For example,

```
scheduleThread.poolsize.vsphere=2
```

Default

1

scheduleThread.queuesize Property

Use this property to limit the metric collection queue size (the number of metrics) for a plug-in.

Usage

Specify the plug-in by name and the maximum metric queue length number:

```
scheduleThread.queuesize.PluginName=15000
```

where *`PluginName`* is the name of the plug-in on which you are imposing a metric limit.

For example,

```
scheduleThread.queuesize.vsphere=15000
```

Default

`1000`

sigar.mirror.procnet Property

`mirror /proc/net/tcp` on Linux.

Default

`true`

snmpTrapReceiver.listenAddress Property

Specifies the port on which the vRealize Hyperic agent listens for SNMP traps

By default, the `agent.properties` file does not include this property.

Typically SNMP uses the UDP port 162 for trap messages. This port is in the privileged range, so an agent listening for trap messages on it must run as `root` (or as an administrative user on Windows).

You can run the agent in the context of a non-administrative user, by configuring the agent to listen for trap messages on an unprivileged port.

Usage

Specify an IP address (or `0.0.0.0` to specify all interfaces on the platform) and the port for UDP communications in the format

```
snmpTrapReceiver.listenAddress=udp:IP_address/port
```

To enable the vRealize Hyperic agent to receive SNMP traps on an unprivileged port, specify port 1024 or higher. The following setting allows the agent to receive traps on any interface on the platform, on UDP port 1620.

```
snmpTrapReceiver.listenAddress=udp:0.0.0.0/1620
```

weblogic.auth.method Property

Enables a vRealize Hyperic agent to communicate with a WebLogic administration server using two-way SSL.

Usage

By default, the `agent.properties` file does not include this property.

Add the following line to the `agent.properties` file to specify that the agent uses two-way SSL for communications with the administration server.

```
weblogic.auth.method=ssl2ways
```

Default

none

weblogic.discovery.new Property

This property controls how WebLogic Server Administration Servers and Managed Servers are discovered. Define this property only if recommended by vRealize Hyperic Support.

By default, the `agent.properties` file does not include this property.

weblogic.installpath Property

Enables the vRealize Hyperic agent to monitor WebLogic 8.1.

Usage

Specify the location `server/lib/weblogic.jar`.

Default

`/usr/local/bea/weblogic-8.1`

weblogic.ssl2ways.cert Property

Enables a vRealize Hyperic agent to communicate with a WebLogic administration server using two-way SSL.

Usage

By default, the `agent.properties` file does not include this property.

Add `weblogic.ssl2ways.cert` to the `agent.properties` file and set its value to the location of the client certificate that the vRealize Hyperic agent presents to the administration server `weblogic.ssl2ways.cert=Client2Cert.pem` where *`Client2Cert.pem`* is the path to the client certificate that the vRealize Hyperic agent presents to the administration server that it manages.

Default

none

weblogic.ssl2ways.key Property

Enables a vRealize Hyperic agent to communicate with a WebLogic administration server using two-way SSL.

Usage

By default, the `agent.properties` file does not include this property.

Add `weblogic.ssl2ways.key` to the `agent.properties` file and set its value to the location of client's private key `weblogic.ssl2ways.key=clientKey.pem` where *`clientkey.pem`* is the path to the private key that the vRealize Hyperic agent presents to the administration server that the agent manages.

Default

none

weblogic.ssl2ways.key.pass Property

Enables a vRealize Hyperic agent to communicate with a WebLogic administration server using two-way SSL.

Usage

By default, the `agent.properties` file does not include this property.

Add `weblogic.ssl2ways.key.pass` to the `agent.properties` file and set its value to the pass phrase for the client private key `weblogic.ssl2ways.key.pass=ClientKey` where *`ClientKey`* is the pass phrase for the client private key.

Default

none

websphere.installpath Property

This property enables the agent to monitor WebSphere.

Usage

Specify the location of the WebSphere JAR files.

Default

`/opt/WebSphere/AppServer`

websphere.useext Property

This property is required to enable management of WebSphere 6.0 and 6.1.

Do not define the `websphere.useext` property to monitor WebSphere 7.

Usage

By default, the `agent.properties` file does not include this property.

For a vRealize Hyperic agent that manages WebSphere 6.0 or 6.1, add `websphere.useext` to the `agent.properties` file.

Configuring the Agent Java Service Wrapper

The configuration file for the vRealize Hyperic agent;s Java service wrapper is located in `AgentHome/bundles/BundleHome/conf/wrapper.conf`.

Following is a list of the Java Service Wrapper Properties for the vRealize Hyperic agent:

```
#********************************************************************
# Java Service Wrapper Properties for the vRealize
                Hyperic Agent
#********************************************************************

# default JAVA_HOME in case it is not already set
set.default.HQ_JAVA_HOME=../../jre

# Java Application
wrapper.java.command=%HQ_JAVA_HOME%/bin/java

# Java Main class.  This class must implement the WrapperListener interface
#  or guarantee that the WrapperManager class is initialized.  Helper
#  classes are provided to do this for you.  See the Integration section
#  of the documentation for details.
wrapper.java.mainclass=org.tanukisoftware.wrapper.WrapperStartStopApp

# Java Classpath (include wrapper.jar)  Add class path elements as
#  needed starting from 1
wrapper.java.classpath.1=../../bundles/%HQ_AGENT_BUNDLE%/lib/*.jar
wrapper.java.classpath.2=../../bundles/%HQ_AGENT_BUNDLE%/pdk/lib/*.jar
wrapper.java.classpath.3=../../wrapper/lib/*.jar
wrapper.java.classpath.4=../../bundles/%HQ_AGENT_BUNDLE%/lib
wrapper.java.classpath.5=../../bundles/%HQ_AGENT_BUNDLE%/pdk/lib/jdbc/*.jar
wrapper.java.classpath.6=../../bundles/%HQ_AGENT_BUNDLE%/pdk/lib/mx4j/*.jar

# Java Library Path (location of Wrapper.DLL or libwrapper.so)
wrapper.java.library.path.1=%LD_LIBRARY_PATH%
wrapper.java.library.path.2=../../wrapper/lib

# Java Additional Parameters
wrapper.java.additional.1=-Djava.security.auth.login.config=../../bundles/%HQ_AGENT_BUNDLE%/jaas.config
wrapper.java.additional.2=-Xmx128m
```

```
wrapper.java.additional.3=-Djava.net.preferIPv4Stack=true
wrapper.java.additional.4=-Dagent.install.home=../..
wrapper.java.additional.5=-Dagent.bundle.home=../../bundles/%HQ_AGENT_BUNDLE%
wrapper.java.additional.6=-Dsun.net.inetaddr.ttl=60

# Initial Java Heap Size (in MB)
#wrapper.java.initmemory=3

# Maximum Java Heap Size (in MB)
#wrapper.java.maxmemory=64

# Application parameters.  Add parameters as needed starting from 1
#wrapper.app.parameter.1=
wrapper.app.parameter.1=org.hyperic.hq.bizapp.agent.client.AgentClient
wrapper.app.parameter.2=1
wrapper.app.parameter.3=start

# The start parameters are followed by the name of the class whose main
# method is to be called to stop the application.  The stop class name
# is followed by a flag that controls whether the Wrapper should
# wait for all non daemon threads to complete before exiting the JVM.
# The flag is followed by the number of parameters to be passed to the
# stop class's main method.  Finally comes the actual parameters.
wrapper.app.parameter.4=org.hyperic.hq.bizapp.agent.client.AgentClient
wrapper.app.parameter.5=true
wrapper.app.parameter.6=2
wrapper.app.parameter.7=die
wrapper.app.parameter.8=30

#*****************************************************************
# Wrapper Logging Properties
#*****************************************************************
# Format of output for the console.  (See docs for formats)
wrapper.console.format=PM

# Log Level for console output.  (See docs for log levels)
wrapper.console.loglevel=INFO

# Log file to use for wrapper output logging.
wrapper.logfile=../../log/wrapper.log

# Format of output for the log file.  (See docs for formats)
wrapper.logfile.format=LPTM

# Log Level for log file output.  (See docs for log levels)
wrapper.logfile.loglevel=INFO

# Maximum size that the log file will be allowed to grow to before
#  the log is rolled. Size is specified in bytes.  The default value
#  of 0, disables log rolling.  May abbreviate with the 'k' (kb) or
#  'm' (mb) suffix.  For example: 10m = 10 megabytes.
wrapper.logfile.maxsize=0

# Maximum number of rolled log files which will be allowed before old
#  files are deleted.  The default value of 0 implies no limit.
```

```
wrapper.logfile.maxfiles=0

# Log Level for sys/event log output.  (See docs for log levels)
wrapper.syslog.loglevel=NONE

#********************************************************************
# Wrapper Windows Properties
#********************************************************************
# Title to use when running as a console
wrapper.console.title=Hyperic HQ Agent

#********************************************************************
# Wrapper Windows NT/2000/XP Service Properties
#********************************************************************
# WARNING - Do not modify any of these properties when an application
#  using this configuration file has been installed as a service.
#  Uninstall the service before modifying this section.  The
#  service can then be reinstalled.

# Name of the service
wrapper.ntservice.name=Hyperic HQ Agent

# Display name of the service
wrapper.ntservice.displayname=Hyperic HQ Agent

# Description of the service
wrapper.ntservice.description=Agent for Hyperic HQ

# Service dependencies.  Add dependencies as needed starting from 1
wrapper.ntservice.dependency.1=

# Mode in which the service is installed.  AUTO_START or DEMAND_START
wrapper.ntservice.starttype=AUTO_START

# Allow the service to interact with the desktop.
wrapper.ntservice.interactive=false

# restart the JVM for all exit codes except the exit code 0
wrapper.on_exit.default=RESTART
wrapper.on_exit.0=SHUTDOWN

# limit the number of JVM restarts
wrapper.max_failed_invocations=5
# if running for over 60 sec assume it was successfully started
wrapper.successful_invocation_time=60
```

Tailoring the Agent Wrapper Configuration

The Java options that are supplied to the vRealize Hyperic agent at startup are configured in the `Java Additional Parameters` section of the `wrapper.conf` file.

You can edit the lines in the `Java Additional Parameters` section, to specify the Java options to apply at startup.

For example, to set the heap size to 256M, you would modify the wrapper.java.additional.2 line, as shown below.

```
Java Additional Parameters
wrapper.java.additional.1=-Djava.security.auth.login.config=../../bundles/%HQ_AGENT_BUNDLE%/jaas.config

wrapper.java.additional.2=-Xmx256m
wrapper.java.additional.3=-Djava.net.preferIPv4Stack=true
wrapper.java.additional.4=-Dagent.install.home=../..
wrapper.java.additional.5=-Dagent.bundle.home=../../bundles/%HQ_AGENT_BUNDLE%

wrapper.java.additional.6=-Dsun.net.inetaddr.ttl=60
```

Automated Deployment of Multiple vRealize Hyperic Agents

You can deploy multiple vRealize Hyperic agents simultaneously, using vRealize Infrastructure Navigator. You configure the agent properties prior to deployment.

For more information, see *vRealize Infrastructure Navigator Installation and Configuration Guide*.

Configuring and Running the vRealize Hyperic Server

2

There are a number of steps that are required to get up and running with the vRealize Hyperic server, and in configuring it for your environment.

- Starting the vRealize Hyperic Server

  The method that you use to start the vRealize Hyperic server depends on how it was installed and how it will run.

- Configuring Metric Baselining and Alert Processing Behavior

  You can configure global control of alert processing, including enabling and disabling alert definitions and notifications, and you can specify an hierarchical method for triggering alerts. You can also specify settings that control the metrics baselining process that influences when alerts are triggered.

- Scaling and Tuning vRealize Hyperic Performance

  You can tune vRealize Hyperic server for large deployments. The sizing profile that you select affects the server properties.

- Integrating vRealize Hyperic Server with Other Systems

  vRealize Hyperic server can be integrated with Kerberos, LDAP, and SMTP Server, and can be configured to send SNMP traps.

- Managing the vRealize Hyperic Database

  You can configure one or more vRealize Hyperic databases, including implementing optional properties.

- Clustering vRealize Hyperic Servers for Failover

  To avoid interruption of vRealize Hyperic server operation in the case of failure, you can configure a cluster of vRealize Hyperic servers.

- vRealize Hyperic Server Properties

  The configuration settings that vRealize Hyperic server requires to start up are included in the `hq-server.conf` file. For example, `hq-server.conf` has properties that tell the server how to connect to the database and where to listen for agent and web application communications.

- Tuning the vRealize Hyperic vApp

  You can tune the vRealize Hyperic vApp for large deployments. A large deployment is defined as one in which the vRealize Hyperic server manages more than 250 platforms.

Starting the vRealize Hyperic Server

The method that you use to start the vRealize Hyperic server depends on how it was installed and how it will run.

- Start the Server on a Unix-Based Platform

  You run a script to start the server.

- Start a Server That Was Installed Using an RPM Package

  If you installed the vRealize Hyperic server from the VMware `yum` repository to a RHEL virtual machine, the vRealize Hyperic server is configured to start automatically each time the virtual machine starts up.

- Start the Server to Run as a Windows Service

  The first time that you start the server following installation, use this command to start it as a Windows service.

Start the Server on a Unix-Based Platform

You run a script to start the server.

Prerequisites

Verify that you did not install vRealize Hyperic server from an RPM package.

Procedure

- At the command line, run `ServerHome/bin/hq-server.sh start`.

  The script displays some startup information on `stdout`, then detaches and runs in the background.

After the server is started, information is written to the `server.log` and `bootstrap.log` files in the `ServerHome/logs` directory.

Start a Server That Was Installed Using an RPM Package

If you installed the vRealize Hyperic server from the VMware `yum` repository to a RHEL virtual machine, the vRealize Hyperic server is configured to start automatically each time the virtual machine starts up.

If you installed the vRealize Hyperic server from a downloaded RPM, follow this procedure to start the vRealize Hyperic server as a daemon.

Procedure

1. Log in to the vRealize Hyperic server host as `root`.
2. Open a terminal window and run the `/etc/init.d/hyperic-hq-server start` .

Start the Server to Run as a Windows Service

The first time that you start the server following installation, use this command to start it as a Windows service.

In subsequent startups, use the Windows Service control panel to start or stop the server.

Procedure

- Run *Server Installation directory*\bin\hq-server.bat install.

Configuring Metric Baselining and Alert Processing Behavior

You can configure global control of alert processing, including enabling and disabling alert definitions and notifications, and you can specify an hierarchical method for triggering alerts. You can also specify settings that control the metrics baselining process that influences when alerts are triggered.

- Configuring Global Alert Properties

  You can use global alert settings to specify immediate global control of alert processing.
- Configure Alert Notification Throttling

  You can use notification throttling to limit the number of alert email actions (notifications sent by email for a triggered alert) that vRealize Hyperic issues in a 15 second interval.
- Alert Notification Email Properties

  Before you can use email notifications when alerts are triggered, specific properties must be defined.
- Metric Baselining Properties

  You set the properties for automatic baseline configuration to control the baselining process and the accuracy of the baseline.

Configuring Global Alert Properties

You can use global alert settings to specify immediate global control of alert processing.

There are three categories of control that you have over alerts.

Alerts — You can enable or disable all alert definitions for all resources immediately. Disabling stops any alerts from being triggered.

	Notifications that have been defined in escalations that are currently in progress are completed.
Alert Notifications	You can enable or disable alert notifications for all resources immediately. Disabling stops all notifications, including those for alerts with escalations currently in progress.
Hierarchical Alerting	The purpose of hierarchical alerting is to avoid triggering alerts for every resource affected by a single root cause. This setting controls whether alerts are evaluated using the hierarchical alerting method. When hierarchical alerting is enabled, before an alert is triggered for a resource, vRealize Hyperic considers the availability and alert status of the resource's parent. You can extend the effect of hierarchical alerting by configuring the relationship between a network device or virtual host and the platforms that depend on it, using the Network and Host Dependency Manager.

Configure Alert Notification Throttling

You can use notification throttling to limit the number of alert email actions (notifications sent by email for a triggered alert) that vRealize Hyperic issues in a 15 second interval.

When the specified threshold is reached, vRealize Hyperic stops sending email alert notifications and instead sends a summary of alert activity every ten minutes to the recipients that you specify.

After starting to throttle, vRealize Hyperic re-evaluates the notification volume for triggered alerts every 10 minutes. When it determines that the per-interval volume of individual notifications that triggered alerts would generate is less than the specified threshold, vRealize Hyperic resumes sending individual notifications.

Procedure

1. Go to **Administration > Server Settings > Notification Throttling Configuration Properties** and click **Notification Throttling ON**.
2. In the **Threshold** text box, type the maximum number of notifications to send in a 15 second interval.
3. Type one or more email addresses for notification recipients in the **Notification Email(s)** text box.
4. Restart vRealize Hyperic.

Alert Notification Email Properties

Before you can use email notifications when alerts are triggered, specific properties must be defined.

Property	Description

Base URL	The `address:port` on which the vRealize Hyperic server listens for web application requests. The initial value of `Base URL` is the Web application listen port that was configured when the vRealize Hyperic server was installed. For example,
	`http://Ms-MacBook-Pro-15.local:7080`
	The base URL forms the prefix of the URL to which vRealize Hyperic appends the remainder of the URL, which points to the Alert Detail page for the triggered alert. For example,
	`http://Ms-MacBook-Pro-15.local:7080/alerts/Alerts.do?mode=viewAlert&eid=5:10611&a=16431`

From Email Address	The email address listed as the sender of the alert emails. For example, `hq@demo2.vmware.com`

Metric Baselining Properties

You set the properties for automatic baseline configuration to control the baselining process and the accuracy of the baseline.

Server Setting	Default	Description
Baseline Frequency	3 days	The frequency with which vRealize Hyperic calculates a baseline for each metric.
Baseline Dataset	7 days	The time range of metric data used in calculating the baseline.
Baseline Minimum Data Points	40	The minimum number of data points used in calculating a baseline.
Track Out-of-Bounds Metrics	off	Controls whether or not vRealize Hyperic tracks out of box metrics.

Scaling and Tuning vRealize Hyperic Performance

You can tune vRealize Hyperic server for large deployments. The sizing profile that you select affects the server properties.

You can run the vRealize Hyperic installer to change the current sizing profile that is applied to the server.

Sizing Considerations

The number of platforms that the vRealize Hyperic server can manage depends on the hardware it runs on, the number of vRealize Hyperic agents reporting to the server, the volume of metrics that are collected, and the size of the vRealize Hyperic database.

Typically, a minimal system configuration supports 25 or more vRealize Hyperic agents. On a high performance platform, a correctly configured vRealize Hyperic server can support up to 2,000 agents.

There are a variety of vRealize Hyperic server properties that govern the system resources available to the server. You determine the properties' values based on the number of platforms under management.

- Server Configuration Settings for Scaling

  You can use the values in the table to determine the server requirements for scaling your environment.

- Java Heap and Garbage Collection

  Heap size startup options are set in the `server.java.opts` property.

- vRealize Hyperic Server Caches

  vRealize Hyperic server uses Ehcache for in-memory caching. Effective cache management is necessary for server stability and performance.

- Monitoring vRealize Hyperic Caches

  You can monitor vRealize Hyperic caches on the **Cache** tab of the HQ Health page.

- Interpreting Cache Statistics

  The values that indicate a well-tuned cache vary by the nature of the caches, and a host of deployment-specific factors. Key things to check for include whether the cache limit has been reached and the hits:misses ratio.

- Configuring Caches

  You can modify the size of a vRealize Hyperic cache by editing an element in the `ehcache.xml` file. Generally, only cache sizes need to be changed.

Server Configuration Settings for Scaling

You can use the values in the table to determine the server requirements for scaling your environment.

By default, at installation the values for the small environment are specified.

Table 2-1. Configuration Settings for Scaling

Property	Small (Fewer than 50 platforms)	Medium (50-250 platforms)	Large (Greater than 250 platforms)
`server.jms.highmemory`	350	1400	2400
`server.jms.maxmemory`	400	1600	3600
`server.database-minpoolsize`	5	20	50
`server.database-maxpoolsize`	100	200	400

Table 2-1. Configuration Settings for Scaling (Continued)

Property	Small (Fewer than 50 platforms)	Medium (50-250 platforms)	Large (Greater than 250 platforms)
server.java.opts	-Djava.awt.headless=true -XX:MaxPermSize=192m -Xmx512m -Xms512m -XX:+HeapDumpOnOutOfMemoryError -XX:+UseConcMarkSweepGC	-Djava.awt.headless=true -XX:MaxPermSize=192m -Xmx4g -Xms4g -XX:+HeapDumpOnOutOfMemoryError -XX:+UseConcMarkSweepGC	-Djava.awt.headless=true -XX:MaxPermSize=192m -Xmn4g -Xmx8g -Xms8g -XX:+HeapDumpOnOutOfMemoryError -XX:+UseConcMarkSweepGC -XX:SurvivorRatio=12 -XX:+UseCompressedOops
tomcat.maxthreads	500	2000	4000
tomcat.minsparethreads	50	100	200
org.hyperic.lather.maxConns(in ServerHome\hq-engine\hq-server\webapps\ROOT \WEB-INF\web.xml)	475	1900	3800

Java Heap and Garbage Collection

Heap size startup options are set in the `server.java.opts` property.

The amount that you can increase the heap size depends on the volume of RAM on the vRealize Hyperic server host.

Given sufficient RAM, you could use these settings`server.java.opts=–Djava.awt.headless=true –XX:MaxPermSize=192m –Xmx4096m –Xms4096m –XX:+UseConcMarkSweepGC –XX:+UseCompressedOops`.

If you are running the vRealize Hyperic server on a 64-bit system with 4GB (4096 MB) or less memory, vRealize Hyperic recommends that you use 32-bit JVM. If you use a 64-bit JVM, ensure that you set the `–XX:+UseCompressedOops` property in `server.java.opts`, with the `oops` option set.

vRealize Hyperic Server Caches

vRealize Hyperic server uses Ehcache for in-memory caching. Effective cache management is necessary for server stability and performance.

Caching policies that define the cache size (maximum number of objects to cache) for each type are defined in `server-n.n.n-EE\hq-engine\hq-server\webapps\ROOT\WEB-INF\classes\ehcache.xml`. The cache size for a type depends on the on how often it likely to be is updated.

Given a fixed amount of memory, cache sizing in vRealize Hyperic attempts to allocate cache according to these guidelines:

- Relatively static types - Caches for types that are not frequently updated. For example, resource, platform, server, and measurement resized to keep objects in memory for the lifetime of the server. An very low miss rate preferable.

  The default cache sizes (the maximum number elements in cache) configured in `ehcache.xml` for inventory types are:

  - Platforms - 2,000
  - Servers ---- 50,000
  - Services - 100,000

    This sizing should be adequate for medium to large deployments.

Dynamic types - Caches for types that are frequently updated - for example, Alert and Galert - and therefore get stale sooner, are configured such that objects age more quickly. A high hit/miss ratio is optimal for dynamic types, in larger environments, in the order of 2:1 or 4:1.

Monitoring vRealize Hyperic Caches

You can monitor vRealize Hyperic caches on the **Cache** tab of the HQ Health page.

You can also view size, hits and misses by running the `ehCache Diagnostics` query on the **Diagnostics** tab.

This data is also written periodically written to `server.log`.

The following information is available for each cache.

Size	The number of objects currently in the cache.
Hits	The number of times a requested object was available in the cache since the last vRealize Hyperic server restart.
Misses	The number of times a requested object was not available in the cache since the last vRealize Hyperic server restart.

Limit	The maximum number of objects the cache can contain.
Total Memory Usage	The amount of memory (in KB) that is currently consumed by all objects in the cache.

Interpreting Cache Statistics

The values that indicate a well-tuned cache vary by the nature of the caches, and a host of deployment-specific factors. Key things to check for include whether the cache limit has been reached and the hits:misses ratio.

The table below lists statistics for several Hyperic caches and, in the "Comments" column, a possible interpretation of the data.

Table 2-2. Cache Statistics Interpretation

Cache	Size	Hits	Misses	Limit	Comments
`Agent.findByAgentToken605`	605	26010977	605	5000	This cache appears healthy. It contains relatively static objects. The cache has not filled up, and the number of misses is equal to the number of hits, so misses occurred only on the first request of each object.
`org.hyperic.hq.events.server.session.Alert`	70526	48049	71274	100000	This cache appears healthy. It contains a type that is likely to become stale relatively quickly, so aging out is appropriate. Although there are more misses than hits, the low number of objects in memory, compared to the cache limit, indicates a low level of server activity since the last restart.
`org.hyperic.hq.events.server.session.AlertDefinition`	66287	44385	66340	100000	This cache appears healthy. It contains a relatively static type, so it is appropriate that the objects do not age out. The cache is not filled up, and the number of misses is very close to the number of hits, indicating most misses occurred on the first request of the object.
`Measurement.findByTemplateForInstance`	10000	6766	25772	10000	This cache appears less healthy. It has reached its maximum size, and the hit ratio is around 20-25%. Ideally, the number of misses should peak at about the maximum size of the cache. Increasing the cache limit would probably improve Hyperic performance. Note that the generalization that misses should peak around the limit of the cache does not apply to the `UpdateTimestampsCache` and the `PermissionCache` caches, which contain types that are invalidated frequently.

Configuring Caches

You can modify the size of a vRealize Hyperic cache by editing an element in the `ehcache.xml` file. Generally, only cache sizes need to be changed.

The `ehcache.xml` file can be found under `server-n.n.n-EE\hq-engine\hq-server\webapps\ROOT\WEB-INF\classes\`.

Each cache is defined with an entry like

```
<cache name="DerivedMeasurement.findByTemplateForInstance"
    maxElementsInMemory="10000"
    eternal="true"
    timeToIdleSeconds="0"
    timeToLiveSeconds="0"
    memoryStoreEvictionPolicy="LRU"/>
```

You might need to iterate the cache size until you find the optimal setting.

Caches that you Cannot Change

There are two caches that you cannot configure:

- `org.hibernate.cache.UpdateTimestampsCache`, which is managed by Hibernate
- `AvailabilityCache`, which is managed by the vRealize Hyperic server

Integrating vRealize Hyperic Server with Other Systems

vRealize Hyperic server can be integrated with Kerberos, LDAP, and SMTP Server, and can be configured to send SNMP traps.

- Configure Kerberos Properties

  You can configure the vRealize Hyperic server to use Kerberos authentication.
- Configure LDAP Authentication

  You can configure the vRealize Hyperic server to use LDAP authentication for new users, and to assign user roles based on LDAP group membership.
- Configure vRealize Hyperic Server for SMTP Server

  vRealize Hyperic sends emails using the SMTP server specified during vRealize Hyperic server installation. To use a remote SMTP server, you configure the vRealize Hyperic server with the remote host connection information, and set up authentication in `hq-server.conf`.
- EnablevRealize Hyperic to Send SNMP Traps

  You can configure vRealize Hyperic to send SNMP traps to an SNMP management system. You can use SNMP notifications in alert definitions, as alert actions and escalation steps.

Configure Kerberos Properties

You can configure the vRealize Hyperic server to use Kerberos authentication.

Procedure

- Go to **Admin > HQ Server Settings**and specify values for the following properties.

Property	Description
`Realm`	Identifies the Kerberos realm.
`KDC`	Identifies the Kerberos kdc.
`Debug`	Enables debug logging.

Configure LDAP Authentication

You can configure the vRealize Hyperic server to use LDAP authentication for new users, and to assign user roles based on LDAP group membership.

Procedure

1. On the **Admin** tab, click **HQ Server Settings**.
2. In the LDAP Configuration Properties section, enter appropriate values for the following properties.

Property	Description
`Use LDAP Authentication`	Select the checkbox to enable LDAP authentication.
`URL`	Enter the location of your LDAP or Active Directory server. If other than the standard LDAP port is used, specify it the URL. Add the port to the end of the URL, after a colon (:) character. For example, `ldap://YourLDAPHost:44389`. If your LDAP directory requires SSL, specify the SSL port in the URL.
`SSL`	Select the checkbox if your LDAP directory requires SSL connections.
`Username`	Supply an LDAP username with sufficient privileges to view the sections of the directory that contain the information for LDAP users who will access vRealize Hyperic. (This property is not necessary if the LDAP directory allows anonymous searching. This is not something that is common in secure environments.)
`Password`	Supply the password for the LDAP user specified in `Username`.
`Search Base`	(Mandatory) The `Search Base` property, sometimes referred to as the suffix, defines the location in the LDAP directory from which the LDAP user search begins. Supply the full path to the branch for example, `ou=people,dc=example,dc=com` Consult your LDAP administrator if necessary.
`Search Filter`	Optionally, enter a filter to limit the LDAP user search to a subset of the object identified by the `Search Base` property. For example, `(!(location=SFO*))`.
`Login Property`	(Mandatory) Specify the LDAP property (for an LDAP user) that vRealize Hyperic will use as the username for the user's vRealize Hyperic account. The default value is `cn`. Depending on your LDAP environment, a different property, for example, `uid`, might be appropriate.

Property	Description
Group Search Base	For vRealize Hyperic to automatically assign vRealize Hyperic roles to new users, supply a value for this property. The property defines the location in the LDAP directory from which the LDAP group search begins.
Search Subtree	If you have configured the `Group Search Base` property, select the checkbox to enable search of the entire subtree of the object identified by `Group Search Base`.
Group Search Filter	If you have configured the `Group Search Base` property, enter a filter to limit the LDAP group search to a subset of the objects found in the group search. The default value `Member={0}`, results in filtering by the full distinguished name of a user. To filter by user login name, set`Member={1}`.

3 Click **OK**.

Configure vRealize Hyperic Server for SMTP Server

vRealize Hyperic sends emails using the SMTP server specified during vRealize Hyperic server installation. To use a remote SMTP server, you configure the vRealize Hyperic server with the remote host connection information, and set up authentication in `hq-server.conf`.

On many Unix and Linux machines, the default - `localhost` is satisfactory. In this case, no additional configuration is required.

Procedure

1 Open *`HQ Server directory`*`/conf/hq-server.conf` and navigate to the Email Settings section.

2 Add the following mail properties to override the default settings of vRealize Hyperic.

The properties that you define depend on whether you require plain text or SSL communication.

a (Optional) To configure plain text communication, add the mail properties below to the file.

The values in the following example is equivalent to the vRealize Hyperic default. Replace the values to something appropriate for you environment.

```
# Change to the SMTP gateway server
# maps to mail.smtp.host,
  server.mail.host=localhost
# Change to SMTP port
  mail.smtp.port=25
# SMTP properties
  mail.smtp.auth=false
  mail.smtp.socketFactory.class=javax.net.SocketFactory
  mail.smtp.socketFactory.fallback=false
  mail.smtp.socketFactory.port=25
  mail.smtp.starttls.enable=false
mail.smtp.connectiontimeout=20000
mail.smtp.timeout=20000
```

b (Optional) To configure SSL communication, add the mail properties below to the file.

```
server.mail.host=SmtpServerHost
mail.user=SmtpUser
mail.password=SmtpPassword
mail.smtp.port=587
mail.smtp.auth=true
mail.smtp.socketFactory.class=javax.net.ssl.SSLSocketFactory
mail.smtp.socketFactory.fallback=false
mail.smtp.socketFactory.port=465
mail.smtp.starttls.enable=true
mail.smtp.connectiontimeout=20000
mail.smtp.timeout=20000
```

3 (Optional) If you are using SSL/TLS, add the SMTP Server's TLS certificate to the JRE keystore.

a Obtain a copy of the public certificate for the SMTP server's TLS configuration (not the private key) on the vRealize Hyperic server.

b With vRealize Hyperic installation owner credentials, execute the following command in the server installation directory.

The certificate import example assumes the use of a JRE that is bundled with the vRealize Hyperic server. When using a non-bundled JRE, use that JRE's keytool and cacerts file.

```
jre/bin/keytool –keystore jre/lib/security/cacerts –import –storepass changeit –
file /path/to/smtp_server_tls.cert
```

.

What to do next

Restart vRealize Hyperic server to implement your changes.

Enable vRealize Hyperic to Send SNMP Traps

You can configure vRealize Hyperic to send SNMP traps to an SNMP management system. You can use SNMP notifications in alert definitions, as alert actions and escalation steps.

Procedure

1. On the **Admin** tab, click **HQ Server Settings**.
2. Select the version that matches your SNMP protocol from the **SNMP Protocol Version** menu.
3. In the SNMP Server Configuration Properties section, enter values that are appropriate to the SNMP protocol that you are using for the following properties.

Table 2-3. vRealize Hyperic Server for SNMP v1

Configuration Option	Description	Allowable Values
`SNMP Trap OID`	The OID of the notification to be sent. Supplies the value of `snmpTrapOID.0` - the second varbind in a trap or inform that Hyperic Server generates. (The first varbind is `SysUpTime.0 .`)	
`Default Notification Mechanism`	Your selection governs the notification type that will appear as the default notification type option in the "Notification Mechanism" pulldown list that is presented in configuration dialogs when user configures an SNMP notification as an alert action, or as a step in an escalation	For v1 of the SNMP protocol, choose V1 Trap. This is the only trap type you can generate for SNMP v1.
`Enterprise OID`	Enterprise OID.	
`Community`	The community name to be sent with the trap.	
`Generic ID`	Single digit identifier of the trap type.	0 - coldStart 1 - warmStart 2 - linkDown 3 - linkUp 4 - authenticationFailure 5 - egpNeighborLoss 6 - enterpriseSpecific
`Specific ID`	The specific trap code for an enterprise-specific trap (when `Generic ID` is set to to 6).	
`Agent Address`	Address of the managed object that generates the trap	

Table 2-4. vRealize Hyperic Server for SNMP v2c

Configuration Option	Description	Allowable Values
SNMP Trap OID	The OID of the notification to be sent. Supplies the value of snmpTrapOID.0 - the second varbind in a trap or inform that Hyperic Server generates. (The first varbind is SysUpTime.0 .)	
Default Notification Mechanism	Specifies the default notification type that will appear in configuration dialogs when an authorized user configures an SNMP notification as an alert action, or as a step in an escalation. This choice simply defines the default option - the user configuring an alert action or escalation can choose a different message type.	■ V1 Trap ■ V2cTrap ■ Inform
Community	The community name to be sent with the trap.	

Table 2-5. vRealize Hyperic Server for SNMP v3

Configuration Option	Description	Allowable Values
SNMP Trap OID	The OID of the notification to be sent. Supplies the value of snmpTrapOID.0 - the second varbind in a trap or inform that Hyperic Server generates. (The first varbind is SysUpTime.0 .)	
Default Notification Mechanism	Specifies the default notification type that will appear in configuration dialogs when an authorized user configures an SNMP notification as an alert action, or as a step in an escalation. This choice simply defines the default option - the user configuring an alert action or escalation can choose a different message type.	■ V1 Trap ■ V2cTrap ■ Inform
Security Name	The username vRealize Hyperic's SNMP agent should use when sending notifications to the NMS.	Required
Local Engine ID	ID of vRealize Hyperic's SNMP agent; this value appears automatically, and is not user-configurable.	
Auth Protocol	The SNMP authentication protocol vRealize Hyperic server should use for communications with the NMS.	■ none ■ MD5 ■ SHA

Table 2-5. vRealize Hyperic Server for SNMP v3 (Continued)

Configuration Option	Description	Allowable Values
Generic ID	Single digit identifier of the trap type.	0 - coldStart 1 - warmStart 2 - linkDown 3 - linkUp 4 - authenticationFailure 5 - egpNeighborLoss 6 - enterpriseSpecific
Auth Passphrase	The SNMP authorization passphrase configured for use when communication with the NMS.	
Privacy Protocol	The SNMP Privacy Protocol vRealize Hyperic server should use for communication with the NMS	
Context Engine ID	The EngineID of the NMS. This, along with Context Name, identifies the SNMP context for accessing management data	Required for v1 and v2c traps. Do not supply for Inform.
Context Name	The name of the SNMP context that provides access to management information on the NMS. A context is identified by the Context Name and Context Engine ID.	

When you have completed the configuration, the **SNMP Trap** notification tab is available when you define or edit an alert definition.

Managing the vRealize Hyperic Database

You can configure one or more vRealize Hyperic databases, including implementing optional properties.

- Building a Metric Data Warehouse

  The retention strategy that vRealize Hyperic uses for measurement data is it to store the minimum amount of data required to pinpoint when changes in performance or availability occur.

- Metric Replication Strategy Overview

  A secondary database instance is configured to store detailed measurement data replicated from the primary vRealize Hyperic database.

- vRealize Hyperic Database Backup and Recovery

  The vRealize Hyperic database contains most of the data necessary to recreate your vRealize Hyperic server environment after a failure, or to move the database to a different host.

- Back up the Built-In PostgreSQL Database

  If you use the built-in vRealize Hyperic database, you must back it up.

- Additional vRealize Hyperic Files to Back Up

  In addition to backing up the vRealize Hyperic database, you might want to back up other files in your server directory.

- Configuring vRealize Hyperic Server Data Compression and Purge Behavior

  vRealize Hyperic server stores monitoring results using a tiered model to minimize the volume of data stored, while still providing sufficient data granularity. Periodically, the vRealize Hyperic server removes detailed metric data from the database and archives it.

- Monitoring the vRealize Hyperic Database

  vRealize Hyperic administrators can view real-time vRealize Hyperic server and database health and load data by clicking **HQ Health** on the Administration page in the vRealize Hyperic user interface.

- vRealize Hyperic Database Table Schemas

  The key vRealize Hyperic database tables include information about resources, metric collection, and measurements.

- Key Resource and Measurement Structures in the vRealize Hyperic Database Tables

  Here is a list of the key resources and measurement table structure for the vRealize Hyperic database.

- MySQL Maintenance Examples

  Here are examples of regular maintenance for MySQL

Building a Metric Data Warehouse

The retention strategy that vRealize Hyperic uses for measurement data is it to store the minimum amount of data required to pinpoint when changes in performance or availability occur.

By default, detailed measurement data is stored for two days, after which the data is compressed and archived as hourly averages with highs and lows. You can configure vRealize Hyperic to keep detailed measurement data for up to a maximum of seven days.

To support requirements for trend analysis over a longer time frame, vRealize Hyperic provides the `MetricDataReplicator` class, which you can use to replicate uncompressed measurement data in a secondary database.

Metric Replication Strategy Overview

A secondary database instance is configured to store detailed measurement data replicated from the primary vRealize Hyperic database.

The secondary database contains one table, EAM_MEASUREMENT_DATA.

The secondary database has a database link to the primary vRealize Hyperic database, and five views that point to the primary vRealize Hyperic database for resource inventory data. The resource inventory data does not physically reside on the secondary database. The database link to the main database allows views on the secondary database to access inventory data in the primary Hyperic database.

The views that are required on the secondary database are

- EAM_PLATFORM
- EAM_SERVER
- EAM_SERVICE
- EAM_RESOURCE
- EAM_MEASUREMENT_TEMPL
- EAM_MEASUREMENT

For more information about these database tables, see vRealize Hyperic Database Table Schemas.

vRealize Hyperic Database Backup and Recovery

The vRealize Hyperic database contains most of the data necessary to recreate your vRealize Hyperic server environment after a failure, or to move the database to a different host.

In addition to historical metrics, the database contains configuration settings, such as vRealize Hyperic agent connection information, collection intervals, portlet configurations, groups, roles, and users. Some server configuration data, such as database connection information, the mail server for alerts, and Java arguments used at server startup, is stored in external files.

Like any other database, your vRealize Hyperic database should be backed up on a regular basis, so that you can restore the data in the event of a failure that corrupts or destroys the database. It is also good practice to backup the database prior to upgrading vRealize Hyperic, your database server, or other software that resides on the server machine.

You should define vRealize Hyperic backup procedures and incorporate them into your overall backup processes. Your local requirements and practices will dictate backup frequency, timing, naming conventions, and retention policies. A daily backup is sufficient for most environments.

Shut Down vRealize Hyperic Server if Backup Makes the Database Unavailable

If your database backup process makes the vRealize Hyperic database unavailable, shut down the vRealize Hyperic server before running the backup.

Back up the Built-In PostgreSQL Database

If you use the built-in vRealize Hyperic database, you must back it up.

Always use this method to back up the built-in database. Do not simply copy the contents of the database's data directory.

Procedure

1 Run the following command to back up the database.

```
pg_dump hqdb | gzip > hqdb-MM.DD.YY.dump.gz
```

2 Copy the dump file to your backup location.

Additional vRealize Hyperic Files to Back Up

In addition to backing up the vRealize Hyperic database, you might want to back up other files in your server directory.

You can back up these additional files without having to first stop the vRealize Hyperic server.

```
conf/
bin/hq-server.sh
hqdb/data/postgresql.conf
```

The contents of these files are stable. Changes are infrequent after your vRealize Hyperic server is installed and configured. Back the files up following installation, and after making changes to the sever configuration.

Configuring vRealize Hyperic Server Data Compression and Purge Behavior

vRealize Hyperic server stores monitoring results using a tiered model to minimize the volume of data stored, while still providing sufficient data granularity. Periodically, the vRealize Hyperic server removes detailed metric data from the database and archives it.

. Alerts and events older than a specified age are removed from the database, and not archived.

The server performs the following periodic data management functions.

Compresses and archives measurement data.	vRealize Hyperic server stores detailed metric data (all data points reported) in the vRealize Hyperic database for a configurable period (up to 7 days) of time, after which the metrics are eligible for compression and archival. On a (configurable) periodic basis, the server removes the aged individual metric data points from the database, and archives the metric data in compressed form: hourly metric averages, highs, and lows. vRealize Hyperic server retains the archived metric data for 2 years.
Purges alert data	vRealize Hyperic server retains fired alert data for a configurable period, after which the alerts are deleted.
Purges event data	vRealize Hyperic server retains event data for a configurable period, after which the events are deleted.
Rebuilds metric table indexes	During normal vRealize Hyperic operation, the metric data tables in the vRealize Hyperic database contain a lot of frequently changing data. The vRealize Hyperic server rebuilds the metric table indexes on a (configurable) periodic basis to avoid performance problems that heavily fragmented indexes can cause.

Monitoring the vRealize Hyperic Database

vRealize Hyperic administrators can view real-time vRealize Hyperic server and database health and load data by clicking **HQ Health** on the Administration page in the vRealize Hyperic user interface.

The information on the HQ Health page is useful to vRealize Hyperic internals experts. vRealize Hyperic support engineers can use the HQ Health data and diagnostics to diagnose and troubleshoot vRealize Hyperic server and database issues.

vRealize Hyperic Database Table Schemas

The key vRealize Hyperic database tables include information about resources, metric collection, and measurements.

EAM_RESOURCE Table

The `EAM_RESOURCE` table contains information about the types in the vRealize Hyperic inventory model and instances of those types in the database. This table has a row for every managed resource in the vRealize Hyperic database, including

- Operating system platforms, and the servers and services that run on them.
- Virtual or network host platforms, and the servers and services that run on them.
- Groups and applications.
- Roles and users.
- Escalations.

Tables for Inventory Resources

The following tables have information about resource instances of a particular inventory type:

EAM_PLATFORM	Contains a row for each platform in inventory.
EAM_SERVER	Contains a row for each server in inventory.
EAM_SERVICE	Contains a row for each service in inventory.
EAM_RESOURCE_GROUP	Contains a row for each group in inventory.

Tables for Platform, Server, and Service Types

The following tables have information about resource types for an inventory type.

EAM_PLATFORM_TYPE	Contains a row for every platform type that vRealize Hyperic can manage.
EAM_SERVER_TYPE	Contains a row for every server type that vRealize Hyperic can manage.
EAM_SERVICE_TYPE	Contains a row for every service type that Hyperic can manage.

Tables for Measurement Information

The following tables have information about the measurements that vRealize Hyperic can collect.

These tables do not store metric values. Metric data is stored in the EAM_MEASUREMENT_DATA_1H, EAM_MEASUREMENTt_DATA_6H, and EAM_MEASUREMENT_DATA_1D tables.

EAM_ MEASUREMENT_TEMPL	Contains a row for a every metric available for every inventory resource type with its metric template and default metric collection settings.
EAM_MEASUREMENT	Contains a row for every metric available for every resource in inventory, with metric collection configuration information: whether collection is enabled and the collection interval for enabled metrics.

Key Resource and Measurement Structures in the vRealize Hyperic Database Tables

Here is a list of the key resources and measurement table structure for the vRealize Hyperic database.

EAM_PLATFORM

The table contains a row for each platform in inventory.

Table 2-6. EAM_PLATFORM Table Fields

Field	Type	Description
ID	int4	An ID for the platform, unique among platforms.
VERSION_COL	int8	Version of the row. Increments when the row is modified. increments with any change to configuration of this row.
FQDN	varchar(200)	Fully qualified domain name of the platform.
CERTDN	varchar(200)	SSL Certificate for the agent which is monitoring this platform.
CID	int4	Not used.
DESCRIPTION	varchar(256)	Description of platform.
CTIME	int8	Creation time of platform.
MTIME	int8	Last modification time of the platform.
MODIFIED_BY	varchar(100)	Last modification user.
LOCATION	varchar(100)	String entered by user, optionally.
COMMENT_TEXT	varchar(256)	String entered by user, optionally.
CPU_COUNT	int4	Number of CPUs on this platform.
PLATFORM_TYPE_ID	int4	ID for the platform type. Points to EAM_PLATFORM_TYPE table.

Table 2-6. EAM_PLATFORM Table Fields (Continued)

Field	Type	Description
CONFIG_RESPONSE_ID	int4	Link to configuration string in plug-in XML file.
AGENT_ID	Int4	A unique identifier to the agent which is monitoring this platform.
RESOURCE_ID	int4	Uniquely identifies the resource, unique across platforms, servers, services. Points to the EAM_RESOURCE table.

EAM_PLATFORM_TYPE

The table contains a row table for each vRealize Hyperic-supported platform type.

EAM_SERVER

The table contains a row for each server in the vRealize Hyperic inventory.

Table 2-7. EAM_SERVER Table Fields

Field	Type	Description
ID	int4	A unique identifier of the server.
VERSION_COL	int8	A column which increments with any change to configuration of this row.
CID	int4	
DESCRIPTION	varchar(300)	Description of server.
CTIME	int8	Creation time of server.
MTIME	int8	Last modification time of the server.
MODIFIED_BY	varchar(100)	Last modification user.
LOCATION	varchar(100)	
PLATFORM_ID	int4	The Unique ID of the platform on which this server is installed.
AUTOINVENTORYIDENTIFIER	varchar(250)	A unique ID describing this server via the plug-in XML.
RUNTIMEAUTODISCOVERY	bool	Is runtime auto discovery enabled on this server?
WASAUTODISCOVERED	bool	Was this server auto discovered?
SERVICESAUTOMANAGED	bool	Not used
AUTODISCOVERY_ZOMBIE	bool	Were there deletions on the client side for this server?
INSTALLPATH	varchar(200)	Install path of this server on the platform.
SERVER_TYPE_ID	int4	Unique ID of the server type that describes this server.

Table 2-7. EAM_SERVER Table Fields (Continued)

Field	Type	Description
CONFIG_RESPONSE_ID	int4	Link to configuration string in plug-in XML file.
RESOURCE_ID	int4	Uniquely identifies the resource, unique across platforms, servers, services. Points to the EAM_ RESOURCE table.

EAM_SERVICE

The table contains a row for for each service in the vRealize Hyperic inventory.

Table 2-8. EAM_SERVICE Table Fields

Field	Type	Description
ID	int4	An ID for the service, unique among services.
VERSION_COL	int8	A column which increments with any change to configuration of this row.
CID	int4	
DESCRIPTION	varchar(200)	Description of service.
CTIME	int8	Creation time of service.
MTIME	int8	Last modification time of the service.
MODIFIED_BY	varchar(100)	Last modification user.
LOCATION	varchar(100)	Not used.
AUTODISCOVERY_ZOMBIE	bool	Were there deletions on the client side for this service?
SERVICE_RT	bool	Is response time enabled for this service?
ENDUSER_RT	bool	Is end user response time enabled for this service?
PARENT_SERVICE_ID	int4	Unique ID into the parent service for this service.
SERVER_ID	int4	Were there deletions on the client side for this server?
AUTOINVENTORYIDENTIFIER	varchar(500)	A unique ID describing this server via the plug-in XML.
SERVICE_TYPE_ID	int4	Unique ID of service type for this service.
CONFIG_RESPONSE_ID	int4	Link to configuration string in plug-in XML file.
RESOURCE_ID	int4	Uniquely identifies the resource, unique across platforms, servers, services. Points to the EAM_ RESOURCE table.

EAM_RESOURCE

This table contains a row for each type in the vRealize Hyperic inventory, and a row for each instance of each type in the vRealize Hyperic database, including:

- Basic inventory types: platforms, servers, and services
- Configurable inventory types: groups and applications
- Users and roles
- Escalations

Table 2-9. EAM_RESOURCE Table Fields

Field	Type	Description
ID	int4	Uniquely identifies a type or an instance of a type.
VERSION_COL	int8	Increments with any change to configuration of this row.
RESOURCE_TYPE_ID	int4	Identifies a type in the vRealize Hyperic inventory model.
INSTANCE_ID	int4	Uniquely identifies a type or an instance of a particular type in the inventory model. For a type, corresponds to the ID column in one of the following tables: `EAM_PLATFORM_TYPE`, `EAM_SERVER_TYPE&`, `EAM_SERVICE_TYPE`, `EAM_APPLICATION_TYPE`, or `EAM_RESOURCE_TYPE.` For an instance of a type, corresponds to the ID column in one of the following tables: `EAM_PLATFORM`, `EAM_SERVER`, `EAM_SERVICE, EAM_RESOURCE_GROUP`, `EAM_APPLICATION`, `EAM_ROLE,` `EAM_SUBJECT. EAM_ESCALATION`
SUBJECT_ID	int4	Identifies the vRealize Hyperic user who owns the resource
PROTO_ID	int4	For a type, value is zero. For an instance of a type, contains the value of the ID column for the type in this table.
NAME	varchar(500)	Display name for a resource, for example, "My-Office-MacBook-Pro-15.local JBoss 4.2 default ServiceManager Stateless Session EJB".
SORT_NAME	varchar(500)	Same as the `NAME` column but all in upper case, for example, "MY-OFFICE-MACBOOK-PRO-15.LOCAL JBOSS 4.2 DEFAULT SERVICEMANAGER STATELESS SESSION EJB".

Table 2-9. EAM_RESOURCE Table Fields (Continued)

Field	Type	Description
FSYSTEM	boolean	
MTIME	int8	Last modification time of the resource.

EAM_MEASUREMENT

Each row contains information about a measurement for a resource under management.

EAM_MEASUREMENT_TEMPL

Contains a row for a every measurement that vRealize Hyperic can collect, for every resource type it can manage, with information about the default metric collection settings.

Table 2-10. EAM_MEASUREMENT_TEMPL Table Fields

Field	Type	Description
ID	int4	A unique identifier of a measurement template for a metric for a resource.
VERSION_COL	int8	A column which increments with any change to configuration of this row.
NAME	varchar(100)	Name of this measurement template.
ALIAS	varchar(100)	String that describes the alias portion of XML file.
UNITS	varchar(50)	Units of this measurement.
COLLECTION_TYPE	int4	Static/dynamic data.
DEFAULT_ON	bool	Does this measurement collect by default?
DEFAULT_INTERVAL	int8	The default collection interval of this metric.
DESIGNATE	bool	Is this metric on the indicator page by default?
TEMPLATE	varchar(2048)	Template string from plug-in XML.
PLUGIN	varchar(250)	Name of the plug-in which houses this measurement template.
CTIME	int8	Creation time of server.
MTIME	int8	Last modification time of the server.
MONITORABLE_TYPE_ID	int4	Key into the monitorable type data.
CATEGORY_ID	int4	Key into the category ID table.

MySQL Maintenance Examples

Here are examples of regular maintenance for MySQL

Table 2-11. Simple MySQL Backup Script

Field	Type	Description
ID	int4	Unique ID for a metric that can be collected for a resource. Points to actually measurements in EAM_MEASUREMENT_DATA_* tables.
VERSION_COL	int8	Indicates version of the row, increments upon each change to the row.
INSTANCE_ID	int4	The resource type the measurement is for. Uniquely identifies a resource type of a given inventory level - platform, server, service. For example, the ID 10001 uniquely identifies the platform type "MacOSX".
TEMPLATE_ID	int4	ID of a template that points to the EAM_MEASUREMENT_TEMPL table.
MTIME	int8	Time modified.
ENABLED	boolean	Is this metric enabled?
COLL_INTERVAL	int8	How often this metric is collected.
DSN	varchar(2048)	A string which describes the measurement from the plugin-xml text.
RESOURCE_ID	int4	Uniquely identifies the resource for which the metric is associated, unique across platforms, servers, services. Points to the EAM_RESOURCE table.

```
#!/bin/sh

START=`date '+%A %Y/%m/%d %H:%M:%S'`
DAY=`date +%A`
MYSQLADMIN="/usr/bin/mysqladmin"
MYSQLDUMP="/usr/bin/mysqldump"
USER="root"
PASSWORD="mysql"
DBNAME="hqdb"
DEST="/home/mysql/dumps/$DBNAME-$DAY.sql.gz"
flushCmd="$MYSQLADMIN -u $USER -p$PASSWORD flush-logs"
dumpCmd="$MYSQLDUMP -u $USER -p$PASSWORD --quick --single-transaction $DBNAME"
gzip="gzip"
echo "Starting backup: $START"
echo "$flushCmd && $dumpCmd | $gzip > $DEST"
$flushCmd && $dumpCmd | $gzip > $DEST
END=`date '+%A %Y/%m/%d %H:%M:%S'`
echo "Backup completed: $END"
```

Simple Log Rollover Scheme

This can be used with error files, log files, and so on.

```
cp /path/to/mysql/log/mysqld.err /path/to/mysql/log/mysqld-`date '+%w'`.err ;
cp /dev/null /path/to/mysql/log/mysqld.err
```

Sample Unix Cron Entries

Be aware that empty lines in the code with fail in cron.

```
#
#       Field 1: (0-59) minute
#       Field 2: (0-23) hour
#       Field 3: (1-31) day  of  the month
#       Field 4: (1-12) month of the year
#       Field 5: (0-6)  day of the week - 1=Monday
# ------------------------------------------------------------------------
#
0 2 * * * backup.sh
0 1 * * * cp /path/to/mysql/log/mysqld.err /path/to/mysql/log/mysqld-`date '+%w'`.err ;
cp /dev/null /path/to/mysql/log/mysqld.err
```

Clustering vRealize Hyperic Servers for Failover

To avoid interruption of vRealize Hyperic server operation in the case of failure, you can configure a cluster of vRealize Hyperic servers.

The failover configuration uses:

- EHCache's distributed caching for replicating changes throughout the cluster.
- The `nodeStatus.hqu` plug-in for monitoring the availability of nodes.
- A hardware load balancer for managing failover when an node becomes unavailable. The load balancer checks the status of each node every 10 seconds, by issuing an HTTP request to the node's `nodeStatus.hqu` plug-in. The check returns a response of `master=true` with a return code of `200` for the primary node. The check returns `master=false` with a return code of `404` inside the body of the response for other nodes in the cluster.

A vRealize Hyperic server cluster contains multiple nodes. Two nodes are generally sufficient. One vRealize Hyperic server, automatically selected by vRealize Hyperic, serves as the primary node. The other node or nodes serve as hot backups. The hot backups do not share the workload with the primary node.

A failover configuration exists that is transparent to users and vRealize Hyperic administrators. This configuration starts a new session that requires you to log in again. It is not apparent that the active vRealize Hyperic server instance is clustered, or which node is currently active.

Requirements for a Failover Deployment

The are several factors that are required to be available to create a failover deployment.

Verify that the following conditions are met.

- A hardware-based load balancer. is available.
- Only one vRealize Hyperic server in an vRealize Hyperic server cluster should receive agent communications at a time. The load balancer should not direct agent connections to an vRealize Hyperic server instance that serves as the secondary node.
- Database Considerations: All nodes in the vRealize Hyperic cluster must share the same database. You cannot use vRealize Hyperic's internal PostgreSQL database in a failover configuration. You must use an external PostgreSQL database.
- The database password, and the encryption key used to encrypt the database password on each vRealize Hyperic server instance, must be identical. Supply the same database password and encryption key when installing each of the server instances to be clustered.

Configure a Server Cluster

Several tasks are required to create and configure a server cluster that you can use as a failover.

This process assumes that you do not already have a vRealize Hyperic installation.

Procedure

1 Install the First vRealize Hyperic Server Instance

You install the vRealize Hyperic server using the process described in the *vRealize Hyperic Installation Guide*.

2 Install Additional vRealize Hyperic Server Nodes

You create additional vRealize Hyperic server nodes in a similar way to the first server installation, but change some parameters during the installation process.

3 Configure the Cluster Name and Communications Properties

You must configure the cluster-related properties on each of the vRealize Hyperic servers in a cluster.

4 Configure the Load Balancer

Configure the load balancer, according to the vendor or supplier instructions. Procedures vary, but at a minimum you will identify the vRealize Hyperic server nodes in the cluster and the failover behavior.

5 Configure Agents to Communicate with the vRealize Hyperic Server Cluster

The vRealize Hyperic agents in your environment communicate with the vRealize Hyperic server cluster through the load balancer. When you start up a newly installed agent, either supply the load balancer listen address and port interactively, or specify the connection information in the `agent.properties` file.

6 Start the Nodes

After you have completed the installation and configuration of all the servers in the cluster, and the load balancer, you must start all the server nodes.

Install the First vRealize Hyperic Server Instance

You install the vRealize Hyperic server using the process described in the *vRealize Hyperic Installation Guide*.

Take note of the encryption key that you specify during the installation process.

Procedure

- Run the vRealize Hyperic installer, selecting the external PostgreSQL database option.

  Clustering requires the use of an external vRealize Hyperic database.

  The installer creates the vRealize Hyperic database schema.

Install Additional vRealize Hyperic Server Nodes

You create additional vRealize Hyperic server nodes in a similar way to the first server installation, but change some parameters during the installation process.

Perform this procedure for each additional node.

Procedure

1. Run the full vRealize Hyperic installer and select the external PostgreSQL database option.
2. When the installer prompts for the location of the vRealize Hyperic database, specify the location of the database that was created for the first server instance.
3. When the installer asks if you want to upgrade, overwrite, or exit the process, select **upgrade.**
4. When the installer prompts you to supply the database password and the encryption key to use to encrypt the database password, enter the same password and encryption key supplied when you installed the first server instance.

Configure the Cluster Name and Communications Properties

You must configure the cluster-related properties on each of the vRealize Hyperic servers in a cluster.

Perform the following process for each server in the cluster.

Procedure

1. In the `Cluster Settings` section of the `conf/hq-server.conf` file, specify the following values.

Property	Value
`ha.partition`	The name of the cluster. This value must be identical for each node in the cluster.
`ha.node.address`	The multicast listen address. Specify the IP address or hostname on which the node listens for multicast traffic. This value is unique to each node in the cluster.

2 (Optional) Use the following properties to control communication behaviors for the nodes in the cluster.

Option	Description
`ha.node.mcast_addr` and `ha.node.mcast_port`	The address and port for sending multicast messages to other nodes. The `ha.node.mcast_addr` value must be the same on each node.
`ha.node.cacheListener.port` and `ha.node.cacheProvider.port`	The ports used for discovering and synchronizing with cache peers.

Configure the Load Balancer

Configure the load balancer, according to the vendor or supplier instructions. Procedures vary, but at a minimum you will identify the vRealize Hyperic server nodes in the cluster and the failover behavior.

Procedure

1 Identify the vRealize Hyperic server nodes in the cluster.

2 Configure the load balancer to check the `nodeStatus.hqu` URL every 10 seconds.

For example, in a 2-node cluster, if the IP addresses of the nodes are 10.0.0.1 and 10.0.0.2, configure the load balancer to check these URLs every 10 seconds.

```
http://hqadmin:hqadmin@10.0.0.1:7080/hqu/health/status/nodeStatus.hqu
```

```
http://hqadmin:hqadmin@10.0.0.2:7080/hqu/health/status/nodeStatus.hqu
```

3 Configure the load balancer to direct all traffic to the node whose status is `master=true`.

Configure Agents to Communicate with the vRealize Hyperic Server Cluster

The vRealize Hyperic agents in your environment communicate with the vRealize Hyperic server cluster through the load balancer. When you start up a newly installed agent, either supply the load balancer listen address and port interactively, or specify the connection information in the `agent.properties` file.

Procedure

- For existing agents, run `hq-agent.sh setup`, to force the setup dialog.

Start the Nodes

After you have completed the installation and configuration of all the servers in the cluster, and the load balancer, you must start all the server nodes.

Procedure

- Start all of the server nodes.

Troubleshooting a Failover Configuration

There are several common sources of problems that might prevent your failover configuration from running.

Note the following potential causes of failure of the failover configuration.

- Multicast blocking: The cluster detection and cache peer detection relies on multicast. Verify that your router is not blocking multicast packets, otherwise the vRealize Hyperic cluster will fail to initialize correctly. It is also common for virtualization technologies like VMware and Xen to not enable multicast by default.
- Do not register agents using the loopback address. If you install a vRealize Hyperic agent on the same machine as a vRealize Hyperic server node, when you specify the IP address the server must use to contact the agent, do not specify loopback address (127.0.0.1).
- If alerts that were currently being triggered or were being escalated were lost, a failover to another cluster node might have occurred in the middle of the alerts being triggered or escalated. The alert state can be lost.

vRealize Hyperic Server Properties

The configuration settings that vRealize Hyperic server requires to start up are included in the `hq-server.conf` file. For example, `hq-server.conf` has properties that tell the server how to connect to the database and where to listen for agent and web application communications.

When you install vRealize Hyperic server, the selections you make, such as port selections, use of plain text, or SSL communications, correspond to properties in the `hq-server.conf` file. The configuration settings that you supply during installation are saved in `ServerHome/conf/hq-server.conf`.

In addition to the properties that reflect installation choices, `hq-server.conf` contains properties with default values that you can modify after installation, based on the your environment and the size of your vRealize Hyperic deployment. For example, there are properties that set defaults for database and JMS configuration options.

Every time vRealize Hyperic server starts, it reads the values of the properties in `hq-server.conf`.

The vRealize Hyperic server supports some properties that do not appear in `hq-server.conf` unless you add them explicitly.

After you change values of properties in `hq-server.conf` or add new properties to the file, you must restart the server for the new settings to take effect

Configuration Settings in the Database

Some of the configuration data that governs the vRealize Hyperic server behavior is stored in the vRealize Hyperic server database. For example, the data that the vRealize Hyperic server needs to contact a vRealize Hyperic agent is stored in the vRealize Hyperic database.

accept.unverified.certificates Property

This property controls whether or a warning is issued when a vRealize Hyperic agent presents an SSL certificate that is not in the server's keystore and is either self-signed or signed by a different certificate authority than the one that signed the server's SSL certificate.

if `accept.unverified.certificates=false`, as it is by default, the following warning is issued:

```
The server to agent communication channel is using a self-signed certificate and could not be verified
Are you sure you want to continue connecting? [default=no]: yes
```

If you respond **yes**, the server imports the agent's certificate, and trusts it in the future.

Note Do NOT set accept.unverified.certificates=true unless ALL agents reporting to the vRealize Hyperic server have been upgraded to vRealize HypericHyperic 4.6, or later.

If `accept.unverified.certificates` is "true", the server automatically accepts and imports the certificate presented by a vRealize Hyperic agent, and does not issue a warning if an agent presents a certificate that the server does not trust.

Default

`agent.setup.acceptUnverifiedCertificate=false`

server.caf.brokerAddress Property

The address upon which the vRealize Hyperic server's internal RabbitMQ node (an Advanced Message Queuing Protocol (AMQP) Broker) listens for TCP/IP requests from Common Agent Framework (CAF) agents.

Default

`localhost`

server.caf.clientId Property

Common Agent Framework (CAF) UUID.

server.database-driver Property

The JDBC driver to use.

This property is rarely modified.

Default

None. The value is set as a result of the database selected during vRealize Hyperic server installation.

server.java.opts Property

Options to pass to Java at vRealize Hyperic server startup.

For information about the effect of this and other server properties on Hyperic performance, see Scaling and Tuning vRealize Hyperic Performance.

Default

The value of the property can vary according to the installation profile for the vRealize Hyperic server.

- Small

```
server.java.opts=Djava.awt.headless=true –XX:MaxPermSize=192m –Xmx512m –Xms512m –XX:
+HeapDumpOnOutOfMemoryError –XX:+UseConcMarkSweepGC
```

- Medium

```
Djava.awt.headless=true –XX:MaxPermSize=192m –Xmx4g –Xms4g –XX:+HeapDumpOnOutOfMemoryError –XX:
+UseConcMarkSweepGC
```

- large

```
Djava.awt.headless=true –XX:MaxPermSize=192m –Xmx8g –Xms8g –XX:+HeapDumpOnOutOfMemoryError –
XX:SurvivorRatio=12 –XX:+UseConcMarkSweepGC –XX:+UseCompressedOops –Xmn4g
```

Setting the vRealize Hyperic Server Time Zone

You can set the time zone for the JVM in which vRealize Hyperic server runs by adding `Duser.timezone=Area/Location` to `server.java.opts`, where

- `Area` is a continent or ocean, for example, `America}`
- `Location` is a city, with an underscore (_) for embedded spaces, for example `New_York`.

server.quartzDelegate Property

The PostgreSQL drive class used by the vRealize Hyperic server scheduler service.

Default

`org.quartz.impl.jdbcjobstore.PostgreSQLDelegate`

server.database-url Property

The JDBC URL to connect to.

Default

None. The value is set as a result of the database selected during HQ Server installation.

If you select...	The default database URL is...
The vRealize Hypericbuilt-in database PostgreSQL	`postgresql://127.0.0.1:9432/hqdb?protocolVersion=2`
An external PostgreSQL	`jdbc:postgresql://localhost:5432/HQ?protocolVersion=2`

server.connection-validation-sql Property

The SQL query to run in order to validate a connection from the pool.

Default

`server.connection-validation-sql=select 1`

server.database-password Property

The database user's password.

Default

None

server.database-user Property

The user name to use for connecting to the database.

Default

`hqadmin`

server.encryption-key Property

The key for decrypting the vRealize Hyperic database user password.

The key must be at least eight characters long, and can contain letters and numbers.

Default

None. The vRealize Hyperic installer prompts for `server.encryption-key` during the vRealize Hyperic server installation.

server.webapp.port Property

The HTTP listen port for the vRealize Hyperic web-based GUI.

Default

7080

server.webapp.secure.port Property

The HTTPS listen port for the vRealize Hyperic web-based GUI.

Default

7443

server.keystore.password Property

This property configures the password for the vRealize Hyperic server's SSL keystore. The location of the keystore is defined by the `server.keystore.path` property.

The vRealize Hyperic installer (in `–full` mode), prompts for the values of `server.keystore.path` and `server.keystore.password` and stores the responses in `hq–server.conf`.

The vRealize Hyperic server's keystore password and private key password must be the same, otherwise the vRealize Hyperic server's internal Tomcat-based server will be unable to start. Follow the same convention as for a vRealize Hyperic agent keystore — set the password for the agent keystore to be the same as the agent private key

Default

The initial value of `server.keystore.password` is set, based on the response to the installation dialog when you run the vRealize Hyperic installer in -full mode, depending on how you respond to the prompt `Would you like us to use a user managed java keystore?`

- If the response is `yes`, the installer prompts for the path and password for your keystore, and saves the values supplied in `server.keystore.path` and `server.keystore.password` (this property), respectively.
- If the response is `no`, the installer sets the value of `server.keystore.path`to `server.keystore.path=ServerHome/conf/hyperic.keystore`, which is the default location for the self-signed certificate that the vRealize Hyperic server generates at first startup, and sets `server.keystore.password` to the default password, `hyperic`.

server.keystore.path Property

This property configures the location of the vRealize Hyperic server's SSL keystore.

Supply the full path to the keystore. The password for the keystore is defined by the `server.keystore.password` property.

The vRealize Hyperic installer (in `–full` mode), prompts for the values of `server.keystore.path` and `server.keystore.password` and stores the responses in `hq–server.conf`.

Specifying the Keystore Path on Windows

On Windows platforms, specify the path to the keystore using Unix-style syntax. To use specify a full Windows path:

- Replace back slashes with forward slashes.
- Put a forward slash at the beginning of the path (before the drive letter.)
- If the path contains spaces, put a backslash before each space in the path For example, to specify this Windows path using Unix syntax `C:\Documents and Settings\Desktop\keystore` and change it to `/C:/Documents\ and\ Settings/Desktop/keystore`.

Default

The initial value of `server.keystore.path` is set, based on the response to the installation dialog when you run the vRealize Hyperic installer in `–full` mode, depending on how you respond to the following prompt:

`Would you like us to use a user managed java keystore?`

- If the response is `yes`, the installer prompts for the path and password for your keystore, and saves the values supplied in `server.keystore.path` (this property) and `server.keystore.password`, respectively.
- If the response is `no`, the installer sets the value of `server.keystore.path` to `server.keystore.path=ServerHome/conf/hyperic.keystore`, which is the default location for the self-signed certificate that the vRealize Hyperic server generates at first startup, and sets `server.keystore.password` to the default password.

server.pluginsync.enable Property

The `server.pluginsync.enable` property enables or disables vRealize Hyperic's server-agent plug-in synchronization (SAPS) feature.

Default

`true`

server.hibernate.dialect Property

The database-specific dialect class used by Hibernate in vRealize Hyperic.

Default

`org.hyperic.hibernate.dialect.PostgreSQLDialect`

server.jms.jmxport Property

This property specifies the port on which the vRealize Hyperic server the Java virtual machine listens for JMX requests, if JMX is enabled.

By default, JMX is disabled. The property that controls whether or not JMX is enabled is `server.jms.usejmx`.

The primary reason to enable JMX is to enable monitoring of the vRealize Hyperic server's internal ActiveMQ server. With the port closed, a vRealize Hyperic agent on the same platform as the vRealize Hyperic server will discover ActiveMQ, but cannot obtain ActiveMQ metrics.

Default

1099

server.jms.usejmx Property

This property controls whether the JMX port on the vRealize Hyperic server Java virtual machine is open or closed.

The primary reason to open the port is to enable monitoring of the vRealize Hyperic server's internal ActiveMQ Server. With the port closed, a vRealize Hyperic agent on the same platform as the vRealize Hyperic server will discover ActiveMQ, but cannot obtain ActiveMQ metrics.

When JMX is enabled, the property that defines the JMX port is `server.jms.jmxport`.

Default

`false`

server.database-blockingtimeout Property

Maximum time in milliseconds to wait for a connection from the pool.

Default

10000

server.database-minpoolsize Property

The minimum number of database connections to keep in the pool.

For information about the effect of this property on vRealize Hyperic performance, see Scaling and Tuning vRealize Hyperic Performance.

Default

The default value of the property depends on the installation sizing profile for the server. A sizing profile is selected during the installation process, and may be updated by running the vRealize Hyperic server installer with the `–updateScale` option.

- small - 5
- medium - 20
- large - 50

server.database Property

The type of database that the vRealize Hyperic server will use.

The only valid value is PostgreSQL.

Default

PostgreSQL

tomcat.minsparethreads Property

The minimum number of unused request processing threads that must be available invRealize Hyperic server's internal tc server's thread pool.

Default

The value of the property varies by the installation profile for the vRealize Hyperic server.

- small - 50
- medium - 100
- large - 200

tomcat.maxthreads Property

The maximum size of vRealize Hyperic server's internal tc server's thread pool.

Default

The value of the property varies by the installation profile for the vRealize Hyperic server.

- small - 500
- medium - 2000
- large - 4000

server.jms.maxmemory Property

Configures the JMS broker memory limit.

If the broker memory limit is reached, the broker will block the `send()` call until some messages are consumed and space becomes available on the broker.

The recommended setting for `server.jms.maxmemory` is 90% of the Java heap size. Erratic alert behavior or missed alerts might indicate the settings are too low.

Default

The default value of the property depends on the installation sizing profile for the server. A sizing profile is selected during the installation process, and may be updated by running the vRealize Hyperic server installer with the `–updateScale` option.

- small - 400
- medium - 1600
- large - 3600

server.jms.highmemory Property

The high memory mark for the JMS queue.

Default

The default value of the property depends on the installation sizing profile for the server. A sizing profile is selected during the installation process, and may be updated by running the vRealize Hyperic server installer with the `–updateScale` option.

- small - 350
- medium - 1400
- large - 2400

server.database-maxpoolsize Property

The maximum number of database connections to keep in the pool.

The value must be set lower than the total number of connections allowed to the backend database.

Default

The default value of the property depends on the installation sizing profile for the server. A sizing profile is selected during the installation process, and may be updated by running the vRealize Hyperic server installer with the `–updateScale` option.

- small - 100
- medium - 200

- large - 400

server.mail.host Property

The IP or hostname of the SMTP server that the vRealize Hyperic server will use for sending alerts and other vRealize Hyperic-related emails. Most UNIX platforms have a local SMTP server, in which case `localhost` or 127.0.0.1 can be used here.

Default

127.0.0.1

vfabric.licenseServer.url Property

This property only applies to vFabric Hyperic servers acquired as a part of vFabric Suite; it is ignored if you obtainedvRealize Hyperic as a stand-alone component.

Use `vfabric.licenseServer.url` to specify the URL of the VMware Licensing Server that administers your vFabric Suite license.

By default, `ServerHome/conf` does not contain `vfabric.licenseServer.url`. To define the location of the VMware License Server, you must add `vfabric.licenseServer.url` to `ServerHome/conf/hq-server.conf`.

vcloud.license.key Property

If you have a vCloud license, use this property to specify the key in `ServerHome/conf/hq-server.conf`.

Tuning the vRealize Hyperic vApp

You can tune the vRealize Hyperic vApp for large deployments. A large deployment is defined as one in which the vRealize Hyperic server manages more than 250 platforms.

Operating System Settings

On the vRealize Hyperic server platform, add the following parameters to `/etc/security/limits.conf`.

```
hyperic user    soft     nofile     8192
hyperic user    hard     nofile    16384
```

On the vRealize Hyperic database platform, add the following parameters to `/etc/security/limits.conf`. The *custom user* variable is usually `hqadmin`.

```
custom user    soft     nofile     8192
custom user    hard     nofile    16384
```

Restart the vApps after saving the changes to `/etc/security/limits.conf`.

On the vRealize Hyperic server platform and on the vRealize Hyperic database platform, add the following parameters to `/etc/sysctl.conf`.

```
net.ipv4.neigh.default.gc_thresh1 = 1024
net.ipv4.neigh.default.gc_thresh2 = 4096
net.ipv4.neigh.default.gc_thresh3 = 8192

net.core.rmem_max=33554432
net.core.wmem_max=33554432
net.ipv4.tcp_rmem=4096 87380 16777216
net.ipv4.tcp_wmem=4096 65536 16777216
net.core.netdev_max_backlog=50000
```

After saving the changes to `/etc/sysctl.conf`, reload the file with the command `root@localhost# sysctl –p`.

vRealize Hyperic Server Settings

Increase the virtual machine memory to 12GB.

Add the following to `hq–server.conf`.

```
server.java.opts=–Djava.awt.headless=true –XX:MaxPermSize=192m –Xmx8g –Xms8g –XX:
+HeapDumpOnOutOfMemoryError –XX:SurvivorRatio=12 –XX:+UseConcMarkSweepGC –XX:+UseCompressedOops –Xmn4g
tomcat.maxthreads=3000
server.database–maxpoolsize=400
```

vRealize Hyperic Database Settings

Increase the virtual machine memory to 16GB.

Edit `/opt/vmware/vpostgres/9.1/data/postgresql.conf` as follows.

```
shared_buffers = 8GB
effective_cache_size = 2GB
max_connections = 410
```