JBoss Operations Network 3.0

Configuring JBoss ON Servers and Agents

customizing server and agent configuration and providing high availability performance

Edition 3.0.1

Last Updated: 2017-09-19

JBoss Operations Network 3.0 Configuring JBoss ON Servers and Agents

customizing server and agent configuration and providing high availability performance
Edition 3.0.1

Ella Deon Lackey
dlackey@redhat.com

Legal Notice

Abstract

Both servers and agents in JBoss ON can be configured to improve performance within your specific environment. High availability, affinity, and failover settings all improve performance large JBoss ON deployments. Other settings related to tasks like discovery and monitoring can be tuned to provide better performance and quality within your specific environment. This guide provides information to understand and edit JBoss ON server and agent configuration.

Table of Contents

1. ABOUT JBOSS OPERATIONS NETWORK

The primary use for JBoss ON is to give administrators a single point of access to view their systems. Functionally, that means that JBoss ON provides a means to develop and monitor a system's *inventory*. Every *managed resource* – from platforms to applications to services – is contained and organized in the inventory, no matter how complex the IT environment is.

JBoss ON centralizes all of its operations in an installed *server*. The JBoss ON server communicates with locally installed JBoss ON *agents*, which interact directly with the platform and services to carry out local tasks such as monitoring. The types of resources that can be managed by JBoss ON and the operations that can be carried out are determined by the server and agent plug-ins which are loaded in JBoss ON.

The relationships between servers, agents, plug-ins, and resources are what define JBoss ON.

1.1. About JBoss ON Agents

JBoss ON agents are deployed on every machine that JBoss ON manages. The agent is an intermediary between the resource itself and the central JBoss ON server.

The agents receive updates like configuration changes, updated packages, new settings for alerts, and operations from the JBoss ON server and then it carries out those tasks on the resource. The agent also collects information from the resource which it forwards to the server. This allows the server to process alerts, metrics, and availability information for the resource.

Because the agent is independent of the server, it can continue with its monitoring tasks and gather information about the resource even if the server is down or the resource loses network connectivity.

Each resource is arranged in a hierarchy, showing relationships between platforms, servers, and services. Only one agent is required per machine; once the platform is managed as a resource, all or a subset of installed applications or services can be added as resources, all using the same local agent.

1.2. About JBoss ON Servers

JBoss ON is built around a central server. The server performs two vital functions:

- Stores the configuration for both resources and resource groups.
- Organizes and responds to the data collect by the agents.

The JBoss ON server is the central location for administrators to manage an operating environment. The server is used to set baseline configuration and provision applications, to define alerts and notifications, and to initiate operations. As agents send information back to the server from the resource, then the server can also perform monitoring tasks (by providing metrics and reporting) and can also respond to events by sending alerts or launching operations.

The data used by the JBoss ON server is stored in a backend SQL database. These data include:

- The inventory of resources
- The configured groups
- Monitoring data
- Configuration data
- Content available to resources
- User and access control information

The JBoss ON server hosts the graphical user interface which is used to interact with JBoss ON.

One very important aspect of JBoss ON servers is this: they only communicate with the backend database and the JBoss ON agent. As long as JBoss ON servers use the same backend database, they are automatically included in a server cloud that allows for failover and scalability, without additional configuration in the servers or the database. JBoss ON agents can be configured to use a list of preferred JBoss ON servers, which naturally distributes the agent load among the servers and provides agent-server failover without detailed configuration.

2. GENERAL MANAGEMENT

This section covers the configuration, files, and options for the JBoss Operations Network server and agents.

2.1. JBoss ON File Locations

This section covers the common files and directories by JBoss ON servers and agents. A basic reference for these files can make managing and troubleshooting JBoss ON easier.

2.1.1. JBoss ON Server File Locations

All JBoss Operations Network servers are installed in a single, user-defined server root directory. In the documentation and examples, this is called *serverRoot*. The directory layout within that server root directory are the same for every server.

```
                           serverRoot
                               |
                              jon
                               |
      ----------------------------------------------------------
      |          |     |      |       |        |       |       |
alert-scripts/  bin/  etc/  EULA  jbossas/  LICENSE  logs/  plugins/
```

The directories and files that are most commonly used to managed JBoss ON servers are listed in Table 1, “JBoss ON Server Directories and Files”. The server root varies for each installation and each platform, but the layout of the JBoss ON subdirectories is the same for every platform.

Table 1. JBoss ON Server Directories and Files

Configuration Area	Directory or File Location	Description
Configuration directory	*serverRoot*/bin/	Contains the server start scripts, PID files, and configuration file.
Start scripts	*serverRoot*/bin/rhq-server{.sh|.bat}	The script to start, stop, and check the status of the server.
Configuration file	*serverRoot*/bin/rhq-server.properties	The configuration file for all server settings that are not stored in the JBoss ON database.
Password hash script	*serverRoot*/bin/generate-db-password{.sh|.bat}	For a migrated server, it generates an encoded form of the database password to use in the **`rhq-server.properties`** file.
SNMP files	*serverRoot*/etc/RHQ-mib.txt	The JBoss ON MIB file to use for setting SNMP traps.
Log files	*serverRoot*/logs/	The JBoss ON server log files are automatically created in this directory. The current log is named **`rhq-server-log4j.log`**. Older log files are named **`rhq-server-log4j.log`**.#, and the higher the number, the older the log file.
Custom plug-in deployment directory	*serverRoot*/plugins/	The directory where custom plug-in files can be dropped for them to be automatically detected and polled by the JBoss ON server.
JBoss AS directory	*serverRoot*/jbossas/	Contains all of the required JBoss AS client and server libraries.[a]
Server JAR files	*serverRoot*/jbossas/default/deploy/rhq.ear/	Contains all of the JAR files used by JBoss ON servers, web interface, and clients.

Configuration Area	Directory or File Location	Description
Server-side plug-ins directory	*serverRoot*/jbossas/default/deploy/rhq.ear/rhq-serverplugins/	Contains all of the JAR files for the default JBoss ON server-side plug-ins.
Agent plug-ins directory	*serverRoot*/jbossas/default/deploy/rhq.ear/rhq-downloads/rhq-plugins/	Contains all of the JAR files for the default JBoss ON agent plug-ins.
Server-side plug-ins directory	*serverRoot*/jbossas/default/deploy/rhq.ear/rhq-serverplugins/	Contains all of the JAR files for the default JBoss ON server-side plug-ins.
Agent package directory	*serverRoot*/jbossas/default/deploy/rhq.ear/rhq-downloads/rhq-agent/	Contains the snapshot packages for the JBoss ON agent.
Web interface directory	*serverRoot*/jbossas/default/work/jboss.web/localhost/	Contains the directories that hold the files for rendering the web interface.

[a] Most of the libraries and files in this directory don't relate directly to JBoss ON.

2.1.2. JBoss ON Agent File Locations

Like the server, the JBoss ON agent is installed in a single, user-defined root directory. All of the agent files and directories are under the **rhq-agent/** directory in that root directory.

```
                   serverRoot
                        |
                  rhq-agent/
                        |
          ----------------------------------
          |      |     |      |      |       |
        bin/  conf/  data/  lib/  logs/  plugins/
```

Table 2. JBoss ON Agent Directories and Files

Configuration Area	Directory or File Location	Description
Start scripts	*serverRoot*/rhq-agent/bin/	Contains the agent start scripts.
Configuration file	*serverRoot*/rhq-agent/conf/agent-configuration.xml	The configuration file for basic agent settings.

Configuration Area	Directory or File Location	Description
Library files	*serverRoot*/rhq-agent/lib/	Contains the libraries used by the agent to monitor resources.
Start scripts	*serverRoot*/rhq-agent/logs/	The JBoss ON agent log files are automatically created in this directory. The current log is named **`agent.log`**. Older log files are named **`agent.log`**.#, and the higher the number, the older the log file.
Plug-ins directory	*serverRoot*/rhq-agent/plugins/	Contains the plug-ins used by the agent for managing resources (like editing resource configuration).

2.2. Default Server and Agent Ports

As with other servers and services, JBoss ON servers and agents communicate with each other by connecting over system ports. JBoss ON uses ports for three types of connections:

- Server to database communication

  The server has to be able to connect to its database. The database port number depends on both the type of database and the specific configuration for the database.

- Server to agent communication

  The server connects to an agent over a single port configured for the agent. This port is used for both standard and SSL communications between the server and agent.

- Agent to server communication

  An agent can talk to multiple JBoss ON servers, even if they use the same port (since each server is on a different host.) The agent will use either a standard port or an SSL port to connect to the JBoss ON server, depending on the connection (transport) method that is configured. The agent will only attempt to use a single port.

NOTE

Servers do not talk to one another directly, so there are no ports for server-to-server links.

The default port numbers for JBoss ON connections are listed in Table 3, “Default JBoss ON Ports”. The port numbers can be changed for any of the JBoss ON services or different values can be used at installation.

Table 3. Default JBoss ON Ports

Connection Type	Port Number
Server to agent	16163
Agent to server (standard)	7080
Agent to server (secure)	7443
Server to database	5432 (default PostgreSQL port)

2.3. Starting the JBoss ON Server

The JBoss ON server is actually a customized JBoss AS server, included in the JBoss ON installation, so starting the JBoss ON server means starting that JBoss instance.

The JBoss ON server can be started manually or can be configured to start and run as a system service.

2.3.1. Starting the JBoss ON Server (Basic)

The JBoss ON server process is started by running a scripting in the *serverRoot*`/bin/` directory. There is an **`.sh`** script for Linux and Unix systems and a **`.bat`** script for Windows systems.

The simplest way to start the server is simply to run the script with the **`start`** command. This starts the server process and then exits from the script.

```
serverRoot/bin/rhq-server.{sh|bat} start
Trying to start the RHQ Server...
RHQ Server (pid 27547) is starting
```

The **`rhq-server.{sh|bat}`** script looks for specific environment variables during its execution, especially related to the JVM to use with the JBoss AS server instance. A complete list of environment variables is given in the script itself; defaults based on the installation information are used, so most environment variables don't need to be reset.

NOTE

The **`RHQ_SERVER_JAVA_HOME`** environment variable must be set on Red Hat Enterprise Linux systems for the server to start. This can be set to a general value like **`/usr/`**.

The server can also be started in console mode, which prints detailed information about the server process to the terminal and leaves the script open as long as the server is running.

```
serverRoot/bin/rhq-server.{sh|bat} console

Starting RHQ Server in console...
=========================================================================
  JBoss Bootstrap Environment

  JBOSS_HOME: serverRoot/jon-server-3.0.0.GA1/jbossas

  JAVA: /usr/bin/java

  JAVA_OPTS: -Dprogram.name=run.sh -Dapp.name=rhq-server -Xms1024M -
Xmx1024M -XX:PermSize=256M -XX:MaxPermSize=256M -
Djava.net.preferIPv4Stack=true -Djboss.server.log.dir=serverRoot/jon-
server-3.0.0.GA1/logs -Djava.awt.headless=true -
Djboss.platform.mbeanserver -Dsun.lang.ClassLoader.allowArraySyntax=true -
Djava.util.logging.config.file=serverRoot/jon-server-
3.0.0.GA1/jbossas/server/default/conf/logging.properties  -
Djava.net.preferIPv4Stack=true

  CLASSPATH: serverRoot/jon-server-3.0.0.GA1/jbossas/bin/run.jar

=========================================================================

15:51:45,955 INFO  [Server] Starting JBoss (MX MicroKernel)...
15:51:45,956 INFO  [Server] Release ID: JBoss [Trinity] 4.2.3.GA (build:
SVNTag=JBoss_4_2_3_GA date=200807181417)
15:51:45,957 INFO  [Server] Home Dir: serverRoot/jon-server-
3.0.0.GA1/jbossas
15:51:45,957 INFO  [Server] Home URL: file:serverRoot/jon-server-
3.0.0.GA1/jbossas/
15:51:45,957 INFO  [Server] Patch URL: null
15:51:45,958 INFO  [Server] Server Name: default
15:51:45,958 INFO  [Server] Server Home Dir: serverRoot/jon-server-
3.0.0.GA1/jbossas/server/default
15:51:45,958 INFO  [Server] Server Home URL: file:serverRoot/jon-server-
3.0.0.GA1/jbossas/server/default/
15:51:45,958 INFO  [Server] Server Log Dir: serverRoot/jon-server-
3.0.0.GA1/logs
15:51:45,958 INFO  [Server] Server Temp Dir: serverRoot/jon-server-
3.0.0.GA1/jbossas/server/default/tmp
15:51:45,958 INFO  [Server] Root Deployment Filename: jboss-service.xml
15:51:46,183 INFO  [ServerInfo] Java version: 1.6.0_15,Sun Microsystems
Inc.
15:51:46,183 INFO  [ServerInfo] Java VM: Java HotSpot(TM) Server VM 14.1-
b02,Sun Microsystems Inc.
15:51:46,184 INFO  [ServerInfo] OS-System: Linux 2.6.18-164.15.1.el5,i386
15:51:46,377 INFO  [Server] Core system initialized
....
```

2.3.2. Running the JBoss ON Server as a Service

The JBoss ON server can be configured to run as a service, managed with systems tools, on both Red Hat Enterprise Linux and Windows.

2.3.2.1. Configuring the JBoss ON Server as a Service on Red Hat Enterprise Linux

The **rhq-server.sh** script can be managed by the**init** process so that the server starts automatically when the system boots. This also allows the server process to be managed by services like **service** and **chkconfig**.

1. Copy the **rhq-server.sh** script into the **/etc/init.d/** directory.

   ```
   cp serverRoot/bin/rhq-server.sh /etc/init.d/
   ```

2. Edit the **/etc/init.d/rhq-server.sh** script to set the **RHQ_SERVER_HOME** variable to the JBoss ON server install directory and the **RHQ_SERVER_JAVA_HOME** variable to the appropriate directory for the JVM. For example:

   ```
   RHQ_SERVER_HOME=serverRoot/jon-server-3.0.0.GA1/
   ```

   ```
   RHQ_SERVER_JAVA_HOME=/usr/
   ```

3. Edit the **/etc/init.d/rhq-server.sh** script, and add the following lines to the top of the file, directly under **#!/bin/sh**.

   ```
   #!/bin/sh
   #chkconfig: 2345 95 20
   #description: JBoss Operations Network Server
   #processname: run.sh
   ```

   The last two numbers in the **#chkconfig: 2345 95 20** line specify the start and stop priority, respectively, for the JBoss ON server.

4. Add the service to the **chkconfig** service management command, and verify that it was added properly.

   ```
   chkconfig --add rhq-server.sh
   chkconfig rhq-server.sh --list
   ```

5. Set the **rhq-server.sh** service to run at run level 5.

   ```
   chkconfig --level 5 rhq-server.sh on
   ```

Once the **init** scripts and **chkconfig** files are updated, then the JBoss ON server can be started and stopped using the **service**command. The status of the process can also be checked.

```
service rhq-server.sh {start|stop|status}
```

2.3.2.2. Configuring JBoss ON as a Windows Service

The **rhq-server.bat** script has an installation option that installs the script as a Windows service. Once installed, the JBoss ON server can be started, stopped, and managed through Windows tools (Add and Remove Programs and Services) or through the **rhq-server.bat** script.

1. Set the environment variable to run the Windows service as.

   Every Windows service has to run as some system user. There are two environment variables in the **rhq-server.bat** script that set the user to use:

   - **RHQ_SERVER_RUN_AS** sets any Windows user to be the JBoss ON server user. The username given here must be in the standard Windows format, *DOMAIN\user*, such as **EXAMPLEDOMAIN\jsmith**.

- **RHQ_SERVER_RUN_AS_ME** sets the server to run as whoever the current user is. This overrides the **RHQ_SERVER_RUN_AS**, if both as set.

If neither environment variable is set, then the JBoss ON server runs as the system account.

2. Run the **rhq-server.bat** script with the **install** option to set up the service. This prompts for the password of whatever user account is used for the JBoss ON service.

```
serverRoot\bin\rhq-server.bat install
```

After the service is set up, the JBoss ON server can be started or stopped using Windows administrative tools or by using any of the options in Table 4, “rhq-server.bat Options” with the script.

Table 4. rhq-server.bat Options

Option	Description
start	Starts the server service.
stop	Stops the server service.
status	Prints the current status (running or stopped) of the service.
remove	Removes, or uninstalls, the JBoss ON server service.

The JBoss ON server Windows service can be modified by changing or adding parameters in the service wrapper configuration file, *serverRoot***bin\wrapper\rhq-server-wrapper.conf**. Table 5, “Common Wrapper Properties” lists some of the wrapper properties that are most commonly edited.

NOTE

Before editing the wrapper file, check out the list of properties in the Java Service Wrapper documentation at http://wrapper.tanukisoftware.org/doc/english/properties.html.

Table 5. Common Wrapper Properties

Parameter	Description

Parameter	Description
wrapper.app.parameter.#	Passes command-line options to the server (the JBoss AS container). Each individual option and its value must be given its own wrapper configuration property and must be placed in numerical order. **IMPORTANT** Do not change any of the five default properties, **`wrapper.app.parameter.1`**. The number for new properties must begin at 5.
wrapper.java.additional.#	Passes additional options to the virtual machine, such as **`-Xmx`** or **`-D`**. Increments the parameters upward numerically. **IMPORTANT** Do not edit the **`wrapper.java.additiona.1`** property unless you want to point to your own log configuration file. Any other properties can be added, removed, or modified. For example: `wrapper.java.additional.5=-XX:+DisableExplicitGC`
wrapper.ntservice.starttype	Sets the start type, either automatically when the system boots (**`AUTO_START`**) or manually (**`DEMAND_START`**).

Alternatively, the wrapper service can be configured by creating a wrapper include file, in the *serverRoot*`\bin\wrapper\rhq-server-wrapper.inc`. An include file augments the service wrapper configuration file and is the recommended way to add more Java VM.

2.4. Starting the JBoss ON Agent

The JBoss ON agent can be started manually or can be configured to start and run as a system service.

IMPORTANT

The agent's configuration is determined by what user is running the agent. If the agent is run as one user and then later run as another user, the agent will have a different configuration that second time because it will use a different backing store for its configuration settings.

This means that if one user is used to configure the agent when it is installed, that same user must be used to run the agent subsequently, or the agent will apparently lose its configuration and need to be reconfigured under the new user.

The agent configuration backing store is described in Section 6.8, “Managing the Agent's Persisted Configuration”.

2.4.1. Starting the JBoss ON Agent (Basic)

The agent is started and runs using a script in the agent's **bin/** directory. Unlike the server start script, which starts the server process and then exits the script, the agent script remains open, with a prompt to accept further input if necessary. (Usually, the script can simply be started and left to run in the background.)

```
/opt/rhq-agent/bin/rhq-agent.sh

RHQ 3.0.0-SNAPSHOT [cda7569] (Tue Apr 13 13:39:16 EDT 2011)
>
```

Most of the time, the JBoss ON agent can run without any additional options or settings. All of the available options for the **rhq-agent.sh** script are listed in Table 13, “Options for the rhq-agent.sh Script”. Additional configuration options can be set by editing the **rhq-agent-env.sh** script file.

NOTE

If there are any errors when starting the JBoss ON agent, run the agent start script with the **--cleanconfig** to wipe the previous agent configuration and start fresh.

2.4.2. Running the Agent as a Windows Service

IMPORTANT

The agent does not prompt for the configuration when it is started as a service. The agent must either be pre-configured or have already been started once and the configuration entered. Both options are described in the *Installation Guide*.

1. Edit the **rhq-agent-wrapper.bat** script and set the environment variable to define the system user as whom the init script will run. There are two options:

    - **RHQ_AGENT_RUN_AS** explicitly sets the user account name. This must match the format of a Windows user account name, *DOMAIN\username*.

- **RHQ_AGENT_RUN_AS_ME** forces the agent to run as whoever the current user is; this uses the format *. \ %USERNAME %*. If both environment variables are defined, this variable overrides **RHQ_AGENT_RUN_AS**.

NOTE

Before setting **RHQ_AGENT_RUN_AS_ME** or **RHQ_AGENT_RUN_AS**, make sure that the given user actually has permission to start services. If necessary, assign the user the appropriate rights. Assigning rights is covered in the Windows documentation.

If neither variable is set, the agent init script runs as the System user.

Other available environment variables are listed and defined in the comments in the **rhq-agent-wrapper.bat** script.

2. Run the **rhq-agent-wrapper.bat** script to install the init script as a service. Use the **install** command to install the init script.

3. When prompted, fill in the password for the system user as whom the service will run.

The agent service starts automatically when the Windows system boots. The service can be started or stopped through the Windows Services Administrative Tools.

The agent service can also be started and stopped through the **rhq-agent-wrapper.bat** script using the **start** and **stop** commands. The **status** command shows whether the agent init script is installed as a service and whether it is running. The **remove** command removes the agent init script as a service.

The JBoss ON agent Windows scripts use the Java Wrapper Service to control the service. A configuration file, *agentRoot***bin\wrapper\rhq-agent-wrapper.conf**, contains the service configuration properties. These are standard wrapper service properties; more information is available at http://wrapper.tanukisoftware.org/doc/english/properties.html.

There are some common properties to edit to custom the service:

- ***wrapper.app.parameter.#*** set command-line options to pass to the agent. These are the same options listed in Section 6.2, “Working with the Agent Command Prompt”. Each option requires its own configuration property. Properties must be placed in numeric order and the first two properties (***wrapper.app.parameter.1*** and ***wrapper.app.parameter.2***) are reserved. Start with ***wrapper.app.parameter.3***.

- ***wrapper.java.additional.#*** set additional JVM options that are passed directly to the VM, corresponding to the **-D** and **-X** options. These also must be incremented numerically. ***wrapper.java.additional.1*** always specifies the log configuration file.

- ***wrapper.ntservice.starttype*** sets when to start the service. The default is **AUTO_START**, which starts the service when the system boots. To start the service manually, the value is **DEMAND_START**.

2.4.3. Running the Agent as a Daemon or init.d Service

IMPORTANT

The agent does not prompt for the configuration when it is started as a service. The agent must either be pre-configured or have already been started once and the configuration entered. Both options are described in the *Installation Guide*.

The agent's configuration is determined by what user is running the agent. If the agent is run as one user and then later run as another user, the agent will have a different configuration that second time because it will use a different backing store for its configuration settings.

This means that if one user is used to configure the agent when it is installed, that same user must be used to run the agent subsequently, or the agent will apparently lose its configuration and need to be reconfigured under the new user.

The agent configuration backing store is described in Section 6.8, “Managing the Agent's Persisted Configuration”.

Once the agent is configured (or pre-configured), the agent can be started in two ways. The **rhq-agent.sh** script starts the agent and opens the command console. The**rhq-agent-wrapper.sh** script simply starts the agent daemon and exits. Both methods can have additional environment variables configured through the **rhq-agent-env.sh** script file.

The daemon can be started and run as a system service. On Red Hat Enterprise Linux, this is done by configuring **/etc/init.d** and then installing it using**chkconfig**. For Solaris and other Unix systems, this is done by configuring **/etc/init.d** and then using other system tools to set up the service.

1. Make sure the agent is fully set up.

2. Open the **rhq-agent-env.sh** file.

3. Uncomment and configure the required environment variables for the agent's **bin** directory, the JDK directory, and the PID directory (which must be writable by the agent user).

   ```
   RHQ_AGENT_HOME=agentRoot/rhq-agent/bin/
   export RHQ_AGENT_JAVA_HOME=/usr
   PIDFILEDIR=/var/run
   ```

   ### NOTE

   When setting the **PIDFILEDIR** on Red Hat Enterprise Linux, edit the ***pidfile*** setting in the**rhq-agent-wrapper.sh** script file. The wrapper script value is used by **chkconfig**.

4. Set any of the optional environment variables.

   - **RHQ_AGENT_DEBUG** enables debug logging.

   - **RHQ_AGENT_JAVA_EXE_FILE_PATH** specifies a Java executable.

   - **RHQ_AGENT_JAVA_OPTS** passes settings to the agent JVM.

   - **RHQ_AGENT_ADDITIONAL_JAVA_OPTS** passes additional Java options to the JVM.

5. Log into the system as root.

   IMPORTANT

   The rest of this procedure describes how to configure the agent init script as a service on Red Hat Enterprise Linux. For other Unix systems, follow a similar procedure that corresponds to the specific platform.

6. Make sure the wrapper script is executable.

   ```
   [root@server rhq-agent]# chmod a+x agentRoot/rhq-agent/bin/rhq-agent-wrapper.sh
   ```

7. Symlink the **rhq-agent-wrapper.sh** file to **/etc/init.d/**. For example:

   ```
   # ln -s agentRoot/rhq-agent/bin/rhq-agent-wrapper.sh
   /etc/init.d/rhq-agent-wrapper.sh
   ```

   IMPORTANT

   On Solaris, symlinking the agent script file requires invoking ***readlink*** in **rhq-agent-wrapper.sh**. ***readlink*** is not supplied by default in some Solaris installations. Solaris users must download ***readlink*** from a source such as Sunfreeware.

8. Register **rhq-agent-wrapper.sh** with **chkconfig**.

   ```
   # /sbin/chkconfig --add rhq-agent-wrapper.sh
   ```

9. Enable the agent service to run at boot time and have it stop gracefully at when the system shuts down.

   ```
   # /sbin/chkconfig rhq-agent-wrapper.sh on
   ```

If the agent service should not be started when the system boots, turn the script off in **chkconfig**:

```
# /sbin/chkconfig rhq-agent-wrapper.sh off
```

2.4.4. Restarting the Agent and the JVM

The agent can be restarted without taking down the agent JVM process. It is also possible to restart both the agent and its JVM.

The agent is managed through a plug-in container managed by the JBoss ON server. The container loads and manages the lifecycle of all agents. Restarting the plug-in container restarts the agent and all its components without destroying the JVM.

1. Select the **Resources** menu in the top navigation bar, and select the**Servers** menu item.
2. Click the agent resource in the list.
3. Click the **Operations** tab.
4. Select and launch the **Restart** task.

Alternatively, both the agent and its JVM can be restarted (this can be useful if, for instance, the launcher script or the JVM options have been edited).

1. Select the **Resources** menu in the top navigation bar, and select the**Servers** menu item.
2. Click the agent resource in the list.
3. Navigate to the launcher script child resource beneath the agent.
4. Click the **Operations** tab for the launcher script resource.
5. Select and launch the **Restart** task.

3. CONFIGURING SSL CONNECTIONS FOR SERVER-AGENT COMMUNICATION

By default, the JBoss ON server and JBoss ON agents talk to each other *in the clear*, meaning all communications traffic is unencrypted and no authentication is performed on either end.

Running servers in the clear, particularly since JBoss ON can perform configuration changes on some types of resources, can have security considerations for your network. JBoss ON should only be run without encryption or authentication if JBoss ON is being tested or if all JBoss ON servers and agents are deployed on a fully secured network, with access limited by a firewall or VPN and restricted to trusted personnel.

JBoss ON uses SSL/TLS to secure connections between agents and servers in two separate ways:

- *Encryption* specially encodes the data sent between agents and servers during a session.
- *Authentication* uses SSL server and client certificates to verify the identity of an agent before it connects to a server, and vice versa.

NOTE

There is a basic authentication mechanism employed by the server in which it assigns security tokens to its agents which are used to identify and "authenticate" registered agents. This token mechanism should not, however, be considered a strong authentication scheme for the purposes of protecting your JBoss ON network from infiltration.

Setting up encryption is very simple; it only requires enabling the proper transport mechanism between servers and agents. This prevents an attacker from intercepting communications or data between a legitimate JBoss ON server and a legitimate JBoss ON agent, by sniffing data or setting up a man-in-the-middle attack.

Authentication adds another layer of protection by preventing an attacker from installing a "rogue" JBoss ON agent and letting it register itself on the JBoss ON system, so that the rogue agent has access to the network. Although setting up authentication is more complicated than using encryption alone, it is worth the effort to implement for the additional protection, especially if there are vulnerabilities in the network setup.

3.1. Setting up Encryption

All that need to be done to set up encryption is to enable the SSL transport connectors in the JBoss ON server and agent configuration files. There are two transport options for SSL, **sslservlet** and **sslsocket**.

The JBoss ON server has a default certificate that it can use for encryption and the agent can generate a self-signed certificate, so it's not necessary to generate or install additional SSL certificates for encryption alone.

1. First, enable SSL encryption on the JBoss ON server.

   1. Shut down the JBoss ON server.

      ```
      serverRoot/jon-server-3.0.0.GA1/bin/rhq-server.sh stop
      ```

   2. Open the ***serverRoot*/jon-server-3.0.0.GA1/bin/rhq-server.properties** file for the JBoss ON server.

   3. Edit the **rhq.communications.connector.*** settings to use SSL. To use the **sslsocket** transport method, which is recommended for authentication, update the **rhq.communications.connector.transport** method, set the port number to use for the socket, and remove the servlet specified in the transport parameters setting.

      ```
      rhq.communications.connector.transport=sslsocket
      rhq.communications.connector.bind-address=
      rhq.communications.connector.bind-port=55555
      rhq.communications.connector.transport-params=
      ```

      To use the **sslservlet** transport method, all that's necessary is to change the **rhq.communications.connector.transport** method.

      ```
      rhq.communications.connector.transport=sslservlet
      rhq.communications.connector.bind-address=
      rhq.communications.connector.bind-port=
      rhq.communications.connector.transport-params=/jboss-remoting-
      servlet-invoker/ServerInvokerServlet
      ```

   4. For setting encryption alone, make sure that certificate-based authentication is disabled:

      ```
      rhq.server.tomcat.security.client-auth-mode=false
      rhq.server.client.security.server-auth-mode-enabled=false
      ```

5. Optionally, define the secure protocol to use. The default is TLS (which is usually fine), but you can set it to SSL.

   ```
   rhq.server.tomcat.security.secure-socket-protocol=TLS
   rhq.server.client.security.secure-socket-protocol=TLS
   ```

6. Save the changes, and restart the JBoss ON server.

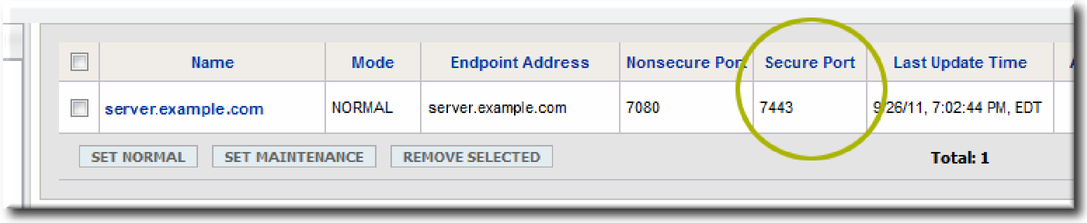
   ```
   serverRoot/jon-server-3.0.0.GA1/bin/rhq-server.sh start
   ```

7. Verify that the end point address and port number given in the configuration are actually the settings set for the server in JBoss ON.

   1. Click the **Administration** tab in the top menu.

   2. In the **Topology** box on the left, select the**Servers** item.

   3. Check the port number in the **Secure Port** column.

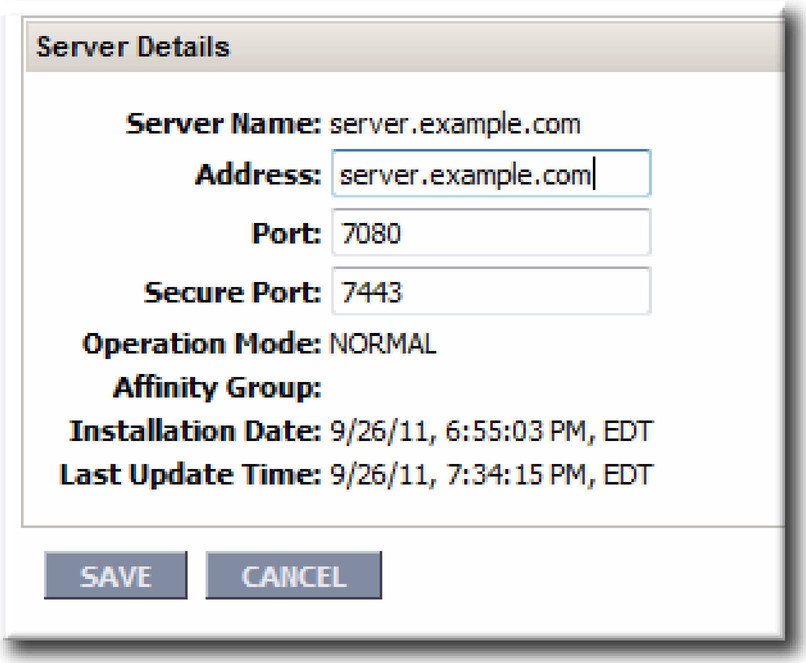


□	Name	Mode	Endpoint Address	Nonsecure Por	Secure Port	Last Update Time
□	server.example.com	NORMAL	server.example.com	7080	7443	9/26/11, 7:02:44 PM, EDT

Total: 1

   4. If the value is wrong, click the name of the server to open the edit page.

   5. Click the **Edit** under the server information, and reset the end point address or port as necessary.



Server Details

Server Name:	server.example.com
Address:	server.example.com
Port:	7080
Secure Port:	7443
Operation Mode:	NORMAL
Affinity Group:	
Installation Date:	9/26/11, 6:55:03 PM, EDT
Last Update Time:	9/26/11, 7:34:15 PM, EDT

2. Then, enable SSL encryption in the agent.

NOTE

This shows how to edit the agent configuration by editing the agent configuration file. The agent configuration can also be edited by going through the advanced setup mode in the agent start script:

```
agentRoot/rhq-agent/bin/rhq-agent.sh --cleanconfig --
setup --advanced
```

1. Open the agent configuration file:

   ```
   vim agentRoot/rhq-agent/conf/agent-configuration.xml
   ```

2. Change the transport protocol to **`sslsocket`**.

   ```
   <entry key="rhq.communications.connector.transport"
   value="sslsocket" />
   ```

3. Set the server connection information so that it matches the configuration for the server. The bind address for the server is commented out by default, and the other parameters are set to the JBoss ON server defaults, including using **`sslservlet`** for the server's transport protocol.

   ```
   <entry key="rhq.agent.server.transport"        value="sslsocket"
   />
   <entry key="rhq.agent.server.bind-port"        value="55555" />
   <entry key="rhq.agent.server.bind-address"
   value="server.example.com" />
   <entry key="rhq.agent.server.transport-params" value="" />
   <entry key="rhq.communications.connector.transport"
   value="sslservlet" />
   ```

4. For setting encryption alone, make sure that certificate-based authentication is disabled. These parameters can be left commented out or can be explicitly set to turn off authentication.

   ```
   <entry key="rhq.communications.connector.security.client-auth-
   mode"       value="none" />
   <entry key="rhq.agent.client.security.server-auth-mode-enabled"
   value="false" />
   ```

5. Optionally, define additional protocol settings for the agent. This is necessary if the server is configured to use transport protocols other than TLS.

   ```
   <entry key="rhq.communications.connector.security.secure-socket-
   protocol" value="TLS" />
   <entry key="rhq.agent.client.security.secure-socket-protocol"
   value="TLS" />
   ```

6. Exit the agent and restart it, using the **`--cleanconfig`** option to load the new configuration.

```
agentRoot/rhq-agent/bin/rhq-agent.sh --cleanconfig
```

3.2. Setting up Client Authentication Between Servers and Agents

*Authentication*is the process of verifying something's identity. With certificate-based authentication, an entity has to obtain a certificate file from a trusted source and, when initiating an SSL connection, that certificate is used to identify that entity. This ensures that the only parties involved in an SSL connection are who they say they are.

To set up certificate-based authentication for JBoss ON, several steps need to be taken. Encryption has to be enabled, certificates have to be issued and stored for the JBoss ON server and agents, and the servers and agents have to be configured to reject messages from untrusted clients.

SSL authentication for JBoss ON is *bi-directional*. The agents are configured to authenticate to the server, and then the server is configured to authentication to the agents.

> **NOTE**
>
> It is possible to configure one-way authentication, where only the server or only the agents have to authenticate. The best security is with bi-directional authentication, which is the configuration given here.

There are two transport methods in JBoss ON that allow SSL connections, **`sslservlet`** and **`sslsocket`**.

The procedure below uses **`sslsocket`**, which allows the default given port to be used for GUI connections while a special port is used for server-agent SSL connections.

Using **`sslservlet`** leverages the embedded Tomcat server, but this requires GUI users to authenticate to the server as well as enabling certificate-based authentication for agents. To allow GUI users to authenticate using their usernames and passwords, set up SSL more or less as outlined below (with some difference in the configuration file settings) and edit the JBoss ON server's Tomcat configuration file (***serverRoot*/jon-server-3.0.0.GA1/jbossas/server/default/deploy/jboss-web.deployer/server.xml** to uncomment the **`<Connector>`** section which says *Provides a secure but un-authenticated https connector for browsers to use.* and set the port for them to use.

1. Enable encryption, as in Section 3.1, “Setting up Encryption”, only make sure that client authentication is *not* disabled.

2. SSL socket connections will occur over a user-defined port. If necessary, open the firewall or VPN to allow access to that port.

3. Generate SSL certificates for each JBoss ON server and agent. For example:

   ```
   keytool -genkey -dname "CN=server1.example.com"  -keystore server1-
   keystore.dat -validity 3650 -alias server1 -keyalg DSA -storetype
   JKS -keypass secret -storepass secret
   ```

   This creates a self-signed certificate with the following characteristics:

   - A common name (CN) value that is the same as the server hostname, **`server1.example.com`**. The **`-dname`** value must be the same as the hostname because during the initial steps of the SSL connection (the SSL handshake), the client will verify that the same identity which was issued the certificate is the same as the one presenting it. Meaning, it will match the hostname in the CN against the hostname of the server or agent presenting the certificate.

- A keystore file called **server1-keystore.dat**
- A validity period of 3650 days
- An alias of **server1**
- A key algorithm of DSA
- Stored in the JKS format in the keystore
- Key and storage passwords of **secret**

Your organization may have a method already for generating or obtaining certificates. This example uses **keytool**; other utilities, like **certutil**, can be used as well. The **keytool** documentation is available through the Oracle-Sun site at http://java.sun.com/javase/6/docs/technotes/tools/windows/keytool.html.

4. Put each self-signed certificate in a single truststore file.

   1. Export the self-signed certificate from each keystore:

      ```
      keytool -export -keystore server1-keystore.dat -alias server1 -
      storetype JKS -storepass secret -file server1-cert
      ```

   2. Import every certificate into a single truststore file:

      ```
      keytool -import -keystore truststore.dat -alias server1 -
      storetype JKS -file server1-cert -noprompt -keypass secret -
      storepass secret
      ```

      -alias is the name to give to the imported certificate in the truststore. For convenience, this is the same as the alias of the original keystore file.

      IMPORTANT

      Import *every* exported server and agent certificate into the same truststore file.

   3. Verify that all the certificates were successfully imported by using the **keytool** to list the certificates:

      ```
      keytool -list -keystore truststore.dat -storepass secret -
      storetype JKS

      Keystore type: JKS
      Keystore provider: SUN

      Your keystore contains 2 entries

      server2, Feb 25, 2011, trustedCertEntry,
      ```

```
Certificate fingerprint (MD5):
24:D9:8A:50:BA:1B:26:08:DC:44:A8:2A:9E:8A:43:D9
server, Feb 25, 2011, trustedCertEntry,
Certificate fingerprint (MD5):
91:F8:78:15:21:E8:0C:73:EC:B6:3B:1D:5A:EC:2B:01
```

5. Distribute both the keystore and the truststore files to all the JBoss ON and server and agent machines. Be sure to distribute the keystores only to the machines which match the hostname in the CN of the certificate; putting the keystore on the wrong machine will cause SSL connections to fail.

    1. For the server, copy the keystore into the ***serverRoot*/jon-server-3.0.0.GA1/jbossas/server/default/conf/** directory of the JBoss AS server embedded in the JBoss Operations Network server. Make sure this file is named **keystore.dat**.

    2. For the server, copy the truststore into the ***serverRoot*/jon-server-3.0.0.GA1/jbossas/server/default/conf/** directory of the embedded JBoss AS server. Make sure this file is named **truststore.dat**.

    3. For the agent, copy the keystore into the ***agentRoot*/rhq-agent/conf** directory. Any certificate file in the ***agentRoot*/rhq-agent/conf** directory is retained even after an automatic update.

6. Shut down the JBoss ON server.

    ```
    serverRoot/jon-server-3.0.0.GA1/bin/rhq-server.sh stop
    ```

7. Open the **rhq-server.properties** file for the JBoss ON server.

    ```
    vim serverRoot/jon-server-3.0.0.GA1/bin/rhq-server.properties
    ```

8. Enable client authentication by setting the **rhq.communications.connector.security.client-auth-mode** parameter to **need** and the **rhq.server.client.security.server-auth-mode-enabled** parameter to **true**.

    Set the information about the keystore and truststore files.

    All of the configuration for incoming messages (agent-to-server communications) is set in **rhq.communications.connector.security.*** parameters. The configuration for outgoing messages is set in **rhq.server.client.security.*** parameters.

    ```
    # Server-side SSL Security Configuration (for incoming messages from
    agents)
    # These are used when secure transports other than sslservlet are
    used
    rhq.communications.connector.security.secure-socket-protocol=TLS
    rhq.communications.connector.security.keystore.file=${jboss.server.h
    ome.dir}/conf/keystore.dat
    rhq.communications.connector.security.keystore.algorithm=SunX509
    rhq.communications.connector.security.keystore.type=JKS
    rhq.communications.connector.security.keystore.password=secret
    rhq.communications.connector.security.keystore.key-password=secret
    rhq.communications.connector.security.keystore.alias=server1
    ```

```
rhq.communications.connector.security.truststore.file=${jboss.server
.home.dir}/conf/truststore.dat
rhq.communications.connector.security.truststore.algorithm=SunX509
rhq.communications.connector.security.truststore.type=JKS
rhq.communications.connector.security.truststore.password=secret
rhq.communications.connector.security.client-auth-mode=need

...

# Client-side SSL Security Configuration (for outgoing messages to
agents)
rhq.server.client.security.secure-socket-protocol=TLS
rhq.server.client.security.keystore.file=${jboss.server.home.dir}/co
nf/keystore.dat
rhq.server.client.security.keystore.algorithm=SunX509
rhq.server.client.security.keystore.type=JKS
rhq.server.client.security.keystore.password=secret
rhq.server.client.security.keystore.key-password=secret
rhq.server.client.security.keystore.alias=myhost
rhq.server.client.security.truststore.file=${jboss.server.home.dir}/
conf/truststore.dat
rhq.server.client.security.truststore.algorithm=SunX509
rhq.server.client.security.truststore.type=JKS
rhq.server.client.security.truststore.password=secret
rhq.server.client.security.server-auth-mode-enabled=true
```

9. Save the file and restart the server.

```
serverRoot/jon-server-3.0.0.GA1/bin/rhq-server.sh start
```

10. In the agent configuration file, uncomment the lines related to secure connections. These parameters begin with **rhq.communications.connector.security.*** and **rhq.agent.client.security.*** for agent-to-server communications and server-to-agent connections, respectively.

    Fill in the appropriate values.

```
<entry key="rhq.communications.connector.security.secure-socket-
protocol" value="TLS" />
<entry key="rhq.communications.connector.security.keystore.file"
value="conf/keystore.dat" />
<entry
key="rhq.communications.connector.security.keystore.algorithm"
value="SunX509" />
<entry key="rhq.communications.connector.security.keystore.type"
value="JKS" />
<entry key="rhq.communications.connector.security.keystore.password"
value="rhqpwd" />
<entry key="rhq.communications.connector.security.keystore.key-
password"  value="rhqpwd" />
<entry key="rhq.communications.connector.security.keystore.alias"
value="rhq" />
<entry key="rhq.communications.connector.security.truststore.file"
value="conf/truststore.dat" />
<entry
```

```
key="rhq.communications.connector.security.truststore.algorithm"
value="SunX509" />
<entry key="rhq.communications.connector.security.truststore.type"
value="JKS" />
<entry
key="rhq.communications.connector.security.truststore.password"
value="" />
<entry key="rhq.communications.connector.security.client-auth-mode"
value="none" />

<entry key="rhq.agent.client.security.secure-socket-protocol"
value="TLS" />
<entry key="rhq.agent.client.security.keystore.file"
value="conf/keystore.dat" />
<entry key="rhq.agent.client.security.keystore.algorithm"
value="SunX509" />
<entry key="rhq.agent.client.security.keystore.type"
value="JKS" />
<entry key="rhq.agent.client.security.keystore.password"
value="rhqpwd" />
<entry key="rhq.agent.client.security.keystore.key-password"
value="rhqpwd" />
<entry key="rhq.agent.client.security.keystore.alias"
value="rhq" />
<entry key="rhq.agent.client.security.truststore.file"
value="conf/truststore.dat" />
<entry key="rhq.agent.client.security.truststore.algorithm"
value="SunX509" />
<entry key="rhq.agent.client.security.truststore.type"
value="JKS" />
<entry key="rhq.agent.client.security.truststore.password"
value="" />
<entry key="rhq.agent.client.security.server-auth-mode-enabled"
value="false" />
```

NOTE

This shows how to edit the agent configuration by editing the agent configuration file. The agent configuration can also be edited by going through the advanced setup mode in the agent start script:

```
agentRoot/rhq-agent/bin/rhq-agent.sh --cleanconfig --
setup --advanced
```

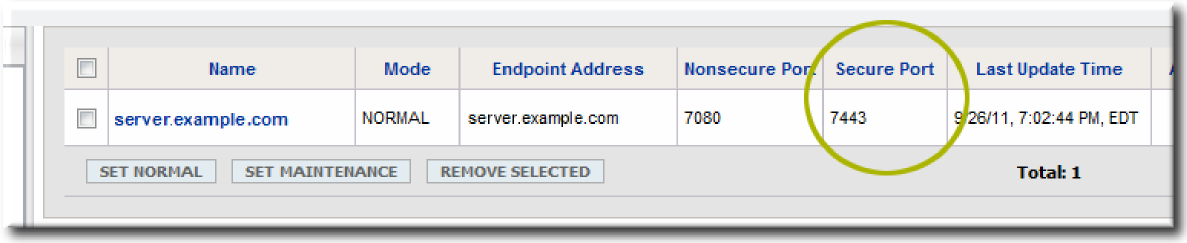
3.3. Troubleshooting SSL Problems

The most common symptom of an SSL connection problem is that the agent will hang when it starts up because it is unable to establish a connection to the JBoss ON server. There are several different areas to check when an SSL problem occurs.

3.3.1. Common SSL Connection Issues

An SSL problem is simply a connection problem, which indicates that there is a problem with the agent or server configuration. There are some general areas to check to make sure that the configuration is all right:

- Make sure that both the agent and the server hostnames are resolvable to the hostnames in their server certificates.
- Make sure that port number given for the server's secure port is actually the port number configured for the server. Check the **`Administration > High Availability > Servers`** page and verify that the public endpoint address and port are correct. Edit the server definition in the UI so they are the same as the SSL configuration.

□	Name	Mode	Endpoint Address	Nonsecure Por	Secure Port	Last Update Time	A
□	server.example.com	NORMAL	server.example.com	7080	7443	9/26/11, 7:02:44 PM, EDT	

  Figure 1. Server Hostname and Port Configuration

  If these values do not match the same values configured for the SSL connection, the agent will not be able to talk to the server.

- Make sure that both the agent and the server hostnames are resolvable to the hostnames in their server certificates.
- Make sure that every certificate that is used for agent-server communication is stored in the requisite keystores with the proper aliases.
- Check that the password is properly set to access the keystore.
- Make sure that the communication is set to use TLS.
- Validate the server and agent configuration, especially the assigned transport (socket or servlet) options. There are examples of configuration in Section 3.3.3, “Example SSL Configuration”.
- If client authentication is required and the server is using the **`sslservlet`** transport option, make sure that every user who connects to the JBoss ON UI has an installed user certificate so that they can connect to the server UI using client authentication. As with the agent certificate, the user certificates must be stored in the server's keystore, Section 3.2, “Setting up Client Authentication Between Servers and Agents”.

  If users are unable to connect using client authentication, then change the server to use **`sslsocket`** instead of **`sslservlet`**.

3.3.2. Enabling SSL Debugging

Enabling verbose logging in the agent can return more details SSL communication messages in the agent log, which can help diagnose connection problems.

1. Open the agent environment variable file. This defines some settings for the JVM which the agent runs in, including debug log settings.

```
vim agentRoot/rhq-agent/bin/rhq-agent-env.sh
```

2. Add a **`RHQ_AGENT_ADDITIONAL_JAVA_OPTS`** line to set a debug environment variable.

```
RHQ_AGENT_ADDITIONAL_JAVA_OPTS="-Djavax.net.debug=all"
```

3. Restart the agent.

```
agentRoot/rhq-agent/bin/rhq-agent.sh
```

3.3.3. Example SSL Configuration

These examples show what correct configuration looks like in both the server and the agent configuration files for the different encryption and authentication configuration scenarios.

Example 1. Encryption Only: Server (sslservlet) and Agent (sslsocket)

Server Configuration	Agent Configuration

rhq.communications.connector.transport=sslservlet
rhq.communications.connector.bind-address=
rhq.communications.connector.bind-port=
rhq.communications.connector.transport-params=/jboss-remoting-servlet-invoker/ServerInvokerServlet
rhq.server.tomcat.security.client-auth-mode=false
rhq.server.client.security.server-auth-mode-enabled=false
	<entry key="rhq.agent.server.transport" value="sslservlet" />
	<entry key="rhq.agent.server.bind-port" value="7443" />

The agent configuration defines the *server's* connection information, so it can be either **`sslservlet`** or **`sslsocket`**. The agent can only receive incoming messages over **`sslsocket`**.

Example 2. Encryption Only: Server (sslsocket) and Agent (sslsocket)

Server Configuration	Agent Configuration

rhq.communications.connector.transport=sslsocket
rhq.communications.connector.bind-address=
rhq.communications.connector.bind-port=7800
rhq.communications.connector.transport-params=
rhq.server.tomcat.security.client-auth-mode=false
rhq.server.client.security.server-auth-mode-enabled=false
	<entry key="rhq.agent.server.transport" value="sslsocket" />
	<entry key="rhq.agent.server.bind-port" value="7800" />
	<entry key="rhq.agent.server.transport-params" value="" />

Because the agent configuration defines the *server*'s connection information, it must match the configuration in the server's **`rhq-server.properties`** file.

Example 3. Encryption and Client Authentication: Server (sslservlet) and Agent (sslsocket)

Server Configuration	Agent Configuration

rhq.communications.connector.transport=sslservlet
rhq.communications.connector.bind-address=
rhq.communications.connector.bind-port=
rhq.communications.connector.transport-params=/jboss-remoting-servlet-invoker/ServerInvokerServlet
rhq.server.tomcat.security.client-auth-mode=true
rhq.server.client.security.server-auth-mode-enabled=true
	<entry key="rhq.agent.server.transport" value="sslservlet" />
	        <entry key="rhq.agent.server.bind-port" value="7443" />

Example 4. Encryption and Client Authentication: Server (sslsocket) and Agent (sslsocket)

Server Configuration	Agent Configuration

rhq.communications.connector.transport=sslsocket
rhq.communications.connector.bind-address=
rhq.communications.connector.bind-port=55555
rhq.communications.connector.transport-params=
 	
rhq.communications.connector.security.client-auth-mode=true
rhq.server.client.security.server-auth-mode-enabled=true
	<entry key="rhq.agent.server.transport" value="sslsocket" />
	<entry key="rhq.agent.server.bind-port" value="55555" />
	<entry key="rhq.agent.server.transport-params" value="" />

4. CONFIGURING HIGH AVAILABILITY

High availability with JBoss ON servers means that all JBoss ON servers which use the same, central database interact together in a cloud. This allows seamless failover between servers when a server has to be taken offline for maintenance, and it provides a natural method for load balancing agent and resource operations.

Agents are assigned, or *partitioned*, among servers in the high availability cloud. Agents aren't strictly assigned to a server to be managed. Agents are handled by servers first based on a preference, or *affinity*, for a server because they belong to to the same affinity group. After affinity, or if an affinity group isn't configured, then agents are managed by the server in a round-robin style for availability.

IMPORTANT

Only agents have an affinity preference in high availability. This means that agents have a preference in which server they attempt to contact. JBoss ON uses two-way communication, however, so servers also contact agents. Servers — regardless of the partition or the agent affinity configuration — can contact any agent in JBoss ON even if the server is not in that agent's affinity group or if the server does not manage the agent.

4.1. Prepping High Availability

High availability means that JBoss ON servers have natural and transparent failover and redundancy: if one server fails, both agents and UI access can be seamless switched to another server. A multi-server high availability configuration provides fault tolerance and improved scalability.

When multiple JBoss ON servers are installed, they use the same backend database to store and retrieve data. This approach distributes the load over multiple servers and provides agent failover.

If an agent's server becomes unavailable, the agent to switch to a different server. Through the use of *affinity groups*, agents switch to a preferred alternative server.

4.1.1. Deciding When to Use High Availability

In many circumstances, it may be satisfactory to run a single-server configuration. However, some environments require a multi-server approach because of the demands on the system:

- Agent report processing is not meeting requirements, for collecting metrics, generating alerts or events, or reporting resource availability.
- You have a geographically distributed environment with multiple data centers or logical grouping of agents to servers.
- The agent load is too high for the server, meaning the server is having trouble processing the agent load. The optimal load per server will depend on the server, rather than the number of agents the server is managing.

There are three things to note about high availability server-agent connections:

- The JBoss ON server defined in the agent's initial configuration is used only for the agent registration. It is not necessarily the one that the agent is told to connect to.
- High availability servers communicate solely through the database. Therefore, server endpoints do not need to be visible to each other. No direct server to server communication is ever made.
- Only agents have an affinity preference in high availability. This means that agents have a preference in which server they attempt to contact. JBoss ON uses two-way communication, however, so servers also contact agents. Servers — regardless of the partition or the agent affinity configuration — can contact any agent in JBoss ON even if the server is not in that agent's affinity group or if the server does not manage the agent.

4.1.2. High Availability Infrastructure Impact

In general, JBoss ON servers can be added or removed from the high availability cloud at any time. A single-server environment can be turned into a multi-server environment by installing a new JBoss ON server on a second machine and configuring it to use.

There are two things to consider when implementing a high availability cloud:

- The potential demands on the shared backend database.
- The IP addresses or names of the servers.

4.1.2.1. Database Impact

Although JBoss ON servers can be added to the high availability server cloud with relative ease, it should be done cautiously due to the potential impact on the back-end database. Each JBoss ON server limits its concurrent database connections, but there is no restriction on the cloud itself. Adding a second server can double the potential database connections, even if the number of agents remains the same. The increase is linear as servers are added.

Each JBoss ON server instance has built-in mechanisms for limiting the load it will put on the database. Each JBoss ON server can use less connections than the maximum limit, but the limits guarantee that they will each never use more than the maximum allowed connections to the database.

NOTE

A high availability configuration does not necessarily imply a large number of JBoss ON agents. It may be the case that a relatively small JBoss ON implementation may be in place, with only a few JBoss ON agents. Those agents, however, may need high availability, and, therefore, failover servers are required. In this case, the backend database will still have a high number of potential connections but, in reality, will not reach that limit.

4.1.2.2. Server and Agent Endpoints

In a multi-server high availability configuration, it is possible for any agent to try to connect to any server. It is *critical* that every JBoss ON agent be able to resolve the endpoint address set for every server in the high availability server cloud. When defining the server in the installer, it is important that the endpoint address be public to the degree that the agent population can reach the server via the defined address and port

An agent connecting to a server must provide an endpoint reachable by the server to allow for the necessary two-way communication.

4.1.2.3. Summary of Server Requirements

1. All high availability servers must be running the same version of JBoss ON.

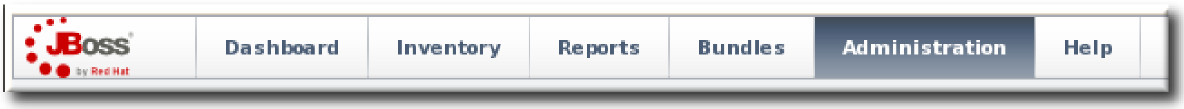
2. All high availability servers must be uniquely named. This string is defined during server installation.

3. Each high availability server must define a unique endpoint that is resolvable by *all* JBoss ON agents running against the high availability server cloud. This address/port is defined during installation. It can be an alias as long as that alias is resolvable by all JBoss ON agents. Any given JBoss ON agent may be running against any JBoss ON server at a given time.

4.2. Putting Servers in Maintenance Mode

Putting a JBoss ON server in maintenance mode temporarily removes it from the high availability cloud so it no longer processes agent operations. This is useful when the server is offline for upgrades or because of a service interruption.

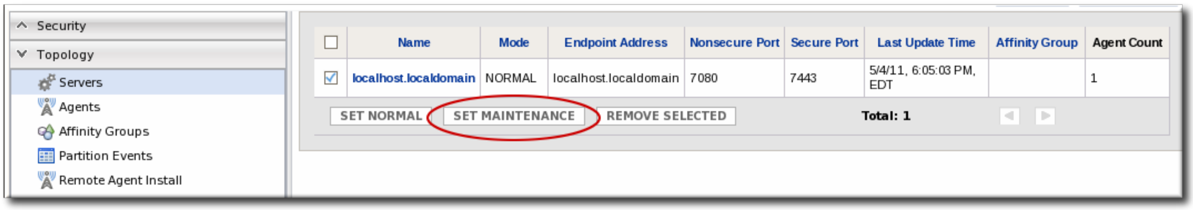
1. Click the **Administration** tab in the top menu.



2. In the **Topology** menu table, select the**Servers** item.

3. Select the check box next to the name of the JBoss ON server to put into maintenance mode.

4. Click the **SET MAINTENANCE** button.



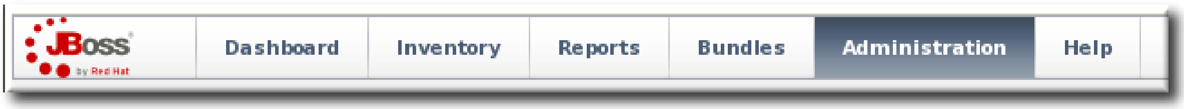
☐	Name	Mode	Endpoint Address	Nonsecure Port	Secure Port	Last Update Time	Affinity Group	Agent Count
☑	localhost.localdomain	NORMAL	localhost.localdomain	7080	7443	5/4/11, 6:05:03 PM, EDT		1

The JBoss ON server can be added back to the high availability cloud by clicking **SET NORMAL** button. Agents migrate back to the server incrementally.

4.3. Removing Servers from the High Availability Cloud

A JBoss ON server that is in maintenance mode can be permanently removed from the high availability cloud.

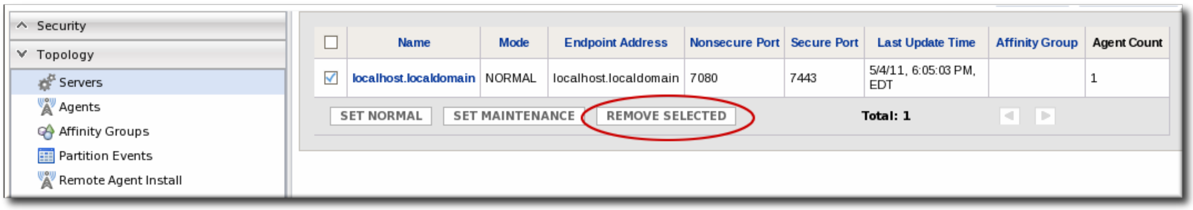
1. Click the **Administration** tab in the top menu.



2. In the **Topology** menu table, select the**Servers** item.

3. Select the check box next to the name of the JBoss ON server to remove from the cloud, and click **SET MAINTENANCE**.

4. When the screen refreshes, select the check box next to the name of the JBoss ON server again, and click the **REMOVE SELECTED** button.



☐	Name	Mode	Endpoint Address	Nonsecure Port	Secure Port	Last Update Time	Affinity Group	Agent Count
☑	localhost.localdomain	NORMAL	localhost.localdomain	7080	7443	5/4/11, 6:05:03 PM, EDT		1

4.4. Defining Affinity for Agents

By default, agent load is distributed evenly amongst the servers in the cloud. Balance can change in failover situations, but agent load is evenly distributed when all agents and all servers are running.

4.4.1. About Affinity

Affinity groups link agents to servers so that agents in an affinity group first try connecting to a server in the same affinity group before connecting to a server outside the affinity group.

Affinity groups provide three distinct advantages:

- Physical efficiency. Generally, if certain agent-server connections clearly run more efficiently than others, then defining affinity to prefer those connections makes sense. This could include servers and agents co-located in the same data center, geographic grouping, or network topology.

- Logical Efficiency. There may be organizational reasons, apart from operating efficiency, to group specific agents and servers together, such as administrative responsibilities or business unit assignments.
- Warm backup. It may be the case that certain machines should not be assigned agent load unless specifically needed for failover purposes. In this case, all agents should be assigned affinity to a subset of available servers, leaving some servers without any associated agents in normal operation.

Section 4.1, “Prepping High Availability” covers the common considerations and restrictions when planning availability for servers and this also applies to planning affinity for agents.

Affinity assignments can be added or removed at any time, but it is useful to consider your initial approach, even if it confirms that affinity assignments are unnecessary.

IMPORTANT

Only agents have an affinity preference in high availability. This means that agents have a preference in which server they attempt to contact. JBoss ON uses two-way communication, however, so servers also contact agents. Servers — regardless of the partition or the agent affinity configuration — can contact any agent in JBoss ON even if the server is not in that agent's affinity group or if the server does not manage the agent.

NOTE

Since high availability environments typically involve many agents, it may be useful to perform pre-configured agent installations to avoid having to answer initial setup questions interactively.

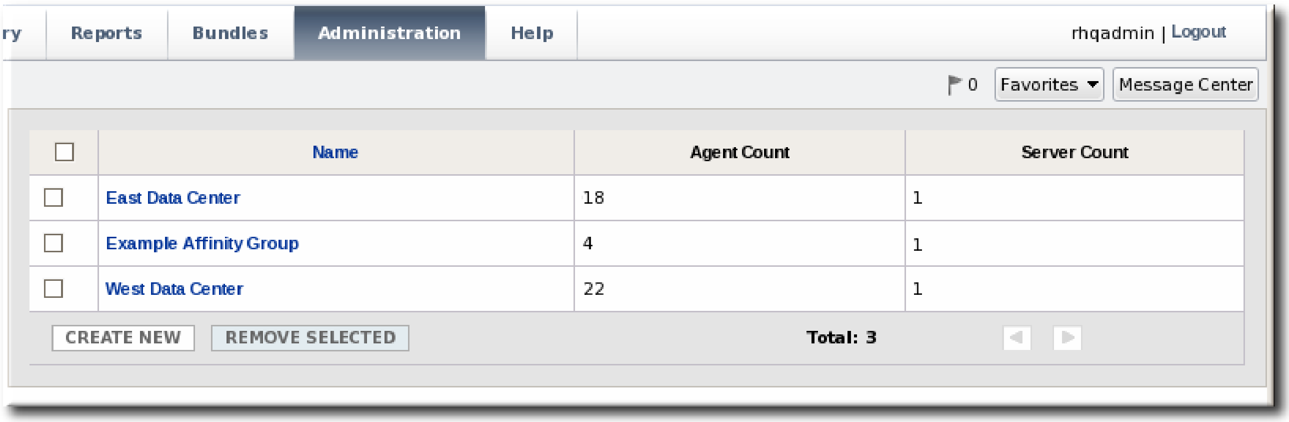
4.4.2. Creating Affinity Groups

An *affinity group* sets a preference for which JBoss ON servers manage which JBoss ON agents. An affinity group only sets a preference or hint for which server will manage the agent, not an absolute requirement. All agents are still managed within the JBoss ON server cloud, so any JBoss ON server can, theoretically, manage any JBoss ON agent based on the current load and performance.

IMPORTANT

Only agents have an affinity preference in high availability. This means that agents have a preference in which server they attempt to contact. JBoss ON uses two-way communication, however, so servers also contact agents. Servers — regardless of the partition or the agent affinity configuration — can contact any agent in JBoss ON even if the server is not in that agent's affinity group or if the server does not manage the agent.

The affinity groups page shows the number of agents and servers assigned to each affinity group.



□	Name	Agent Count	Server Count
□	East Data Center	18	1
□	Example Affinity Group	4	1
□	West Data Center	22	1

Figure 2. Listing Affinity Groups

NOTE

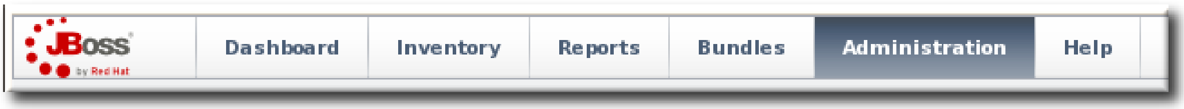
An agent and a server can only belong to a single affinity group.

To create a new affinity group:

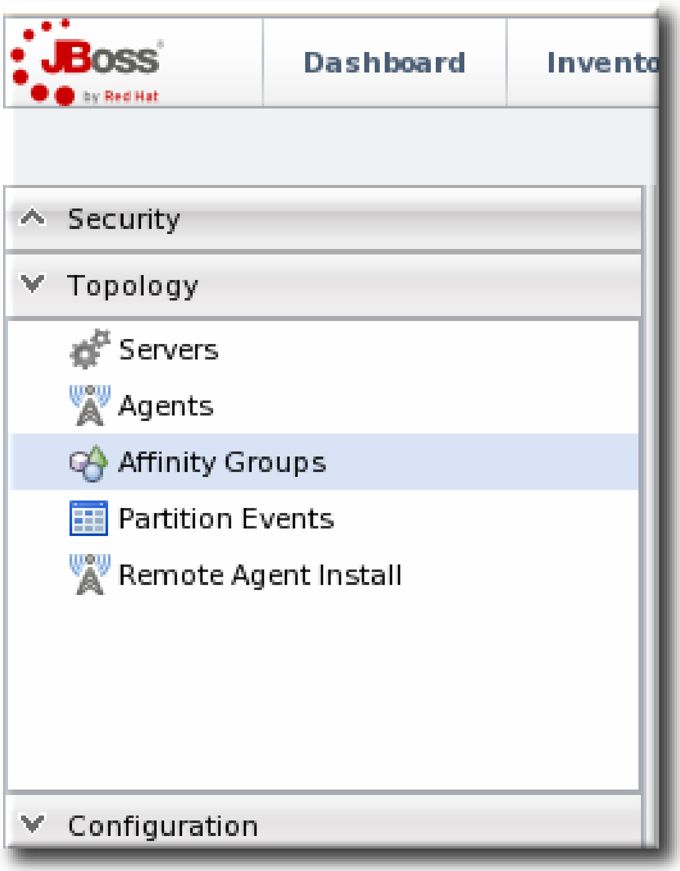
NOTE

To edit an affinity group, click its name, then manage it the same as creating a new affinity group.

1. Click the **`Administration`** tab in the top menu.

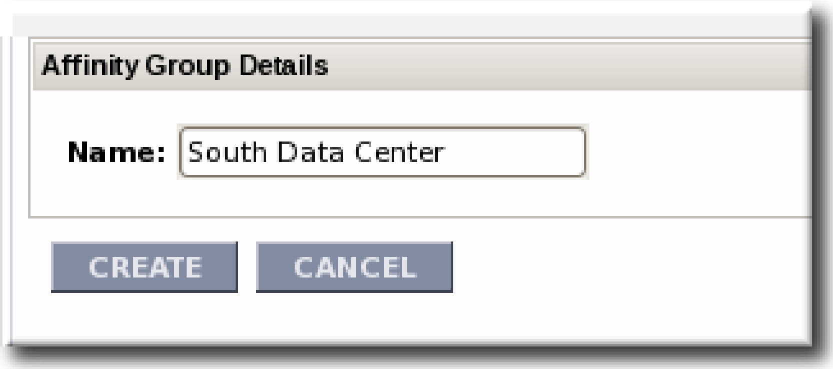


2. In the **Topology** menu table on the left, select the**Affinity Groups** item.



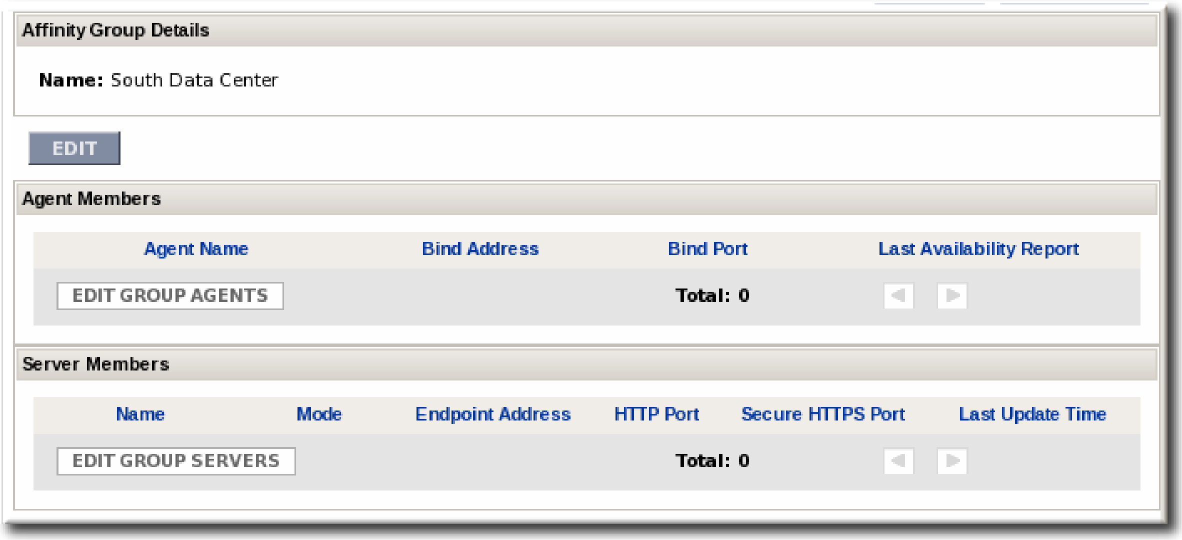
3. Click the **CREATE NEW** button.

4. Enter a name for the new affinity group, and click **OK**.


Affinity Group Details

Name: South Data Center

5. In the new affinity group's details page, click the **EDIT GROUP AGENTS** button.

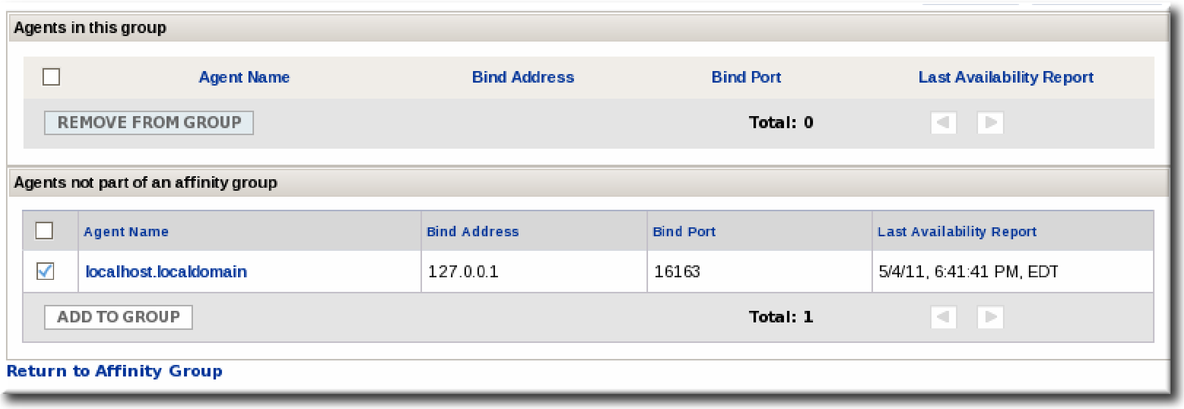


Name: South Data Center

Agent Name	Bind Address	Bind Port	Last Availability Report
		Total: 0	

Name	Mode	Endpoint Address	HTTP Port	Secure HTTPS Port	Last Update Time
			Total: 0		

6. In the lower section, **`Agents not part of an affinity group`**, click the checkboxes by the agent names to add to the group, and click **`ADD TO GROUP`**.



□	Agent Name	Bind Address	Bind Port	Last Availability Report
			Total: 0	

□	Agent Name	Bind Address	Bind Port	Last Availability Report
☑	localhost.localdomain	127.0.0.1	16163	5/4/11, 6:41:41 PM, EDT
			Total: 1	

7. Click the **`Return to Affinity Group Link`**.

8. As with the agents, click the **`EDIT GROUP SERVERS`** button to open the server lists and look at the list in the lower section of servers which do not currently belong to the affinity group. Click the checkboxes by the server names to add to the group, and click **`ADD TO GROUP`**.

Once both servers and agents have been added to the affinity group, the group is fully configured.

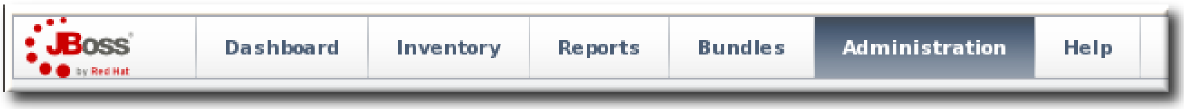
4.5. Managing Partition Events

When an agent is assigned to be managed by a server, that is a *partition*. *Partition events* are almost like log messages that occur whenever a change in the partition configuration occurs.

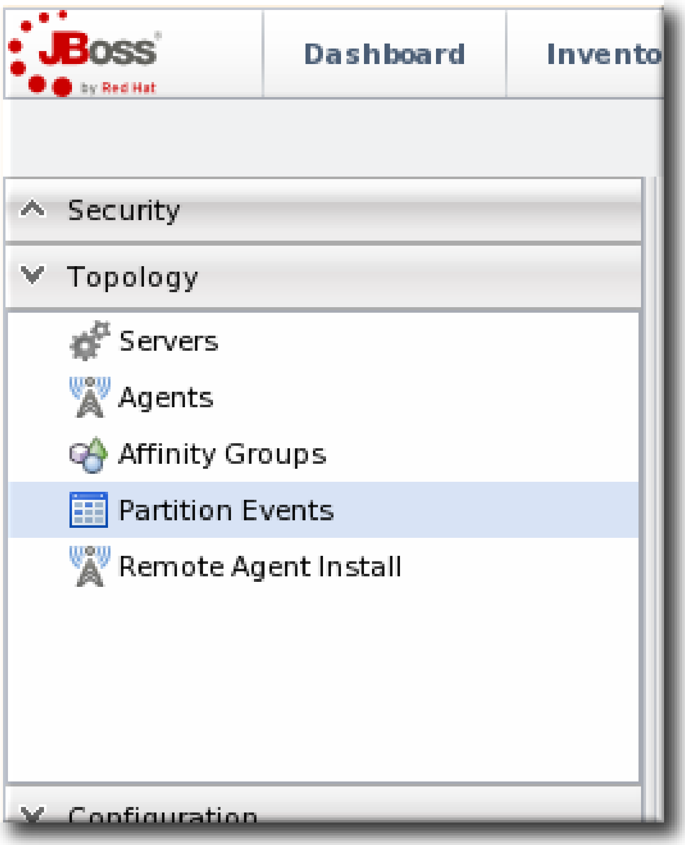
4.5.1. Viewing Partition Events

The partition events log is accessed in the high availability configuration.

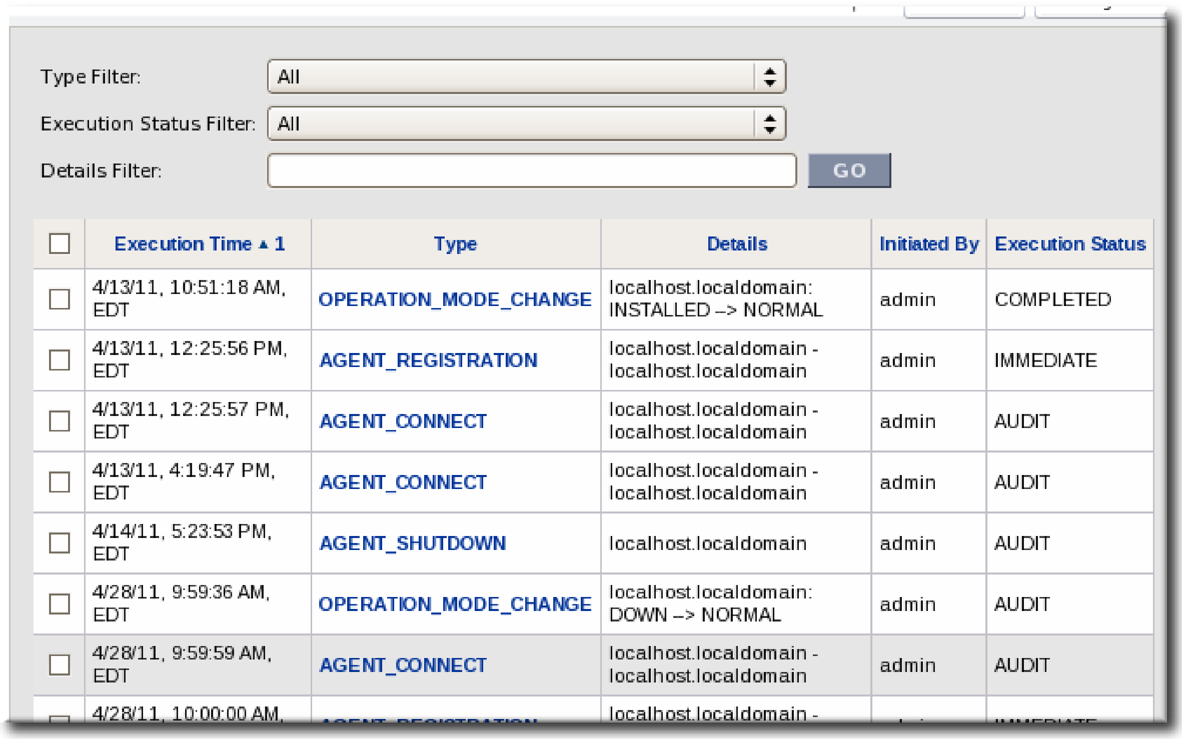
1. Click the **`Administration`** tab in the top menu.



2. In the **Topology** menu table on the left, select the**Partition Events** item.



3. The partition events page lists all of the events that have been recorded. (Table 7, “Partition Events Entries” describes the different fields.) Click the type name of any partition event opens up that record with more information about how the partition assignments were affected.



Type Filter: All

Execution Status Filter: All

Details Filter:

□	Execution Time ▲ 1	Type	Details	Initiated By	Execution Status
□	4/13/11, 10:51:18 AM, EDT	OPERATION_MODE_CHANGE	localhost.localdomain: INSTALLED --> NORMAL	admin	COMPLETED
□	4/13/11, 12:25:56 PM, EDT	AGENT_REGISTRATION	localhost.localdomain - localhost.localdomain	admin	IMMEDIATE
□	4/13/11, 12:25:57 PM, EDT	AGENT_CONNECT	localhost.localdomain - localhost.localdomain	admin	AUDIT
□	4/13/11, 4:19:47 PM, EDT	AGENT_CONNECT	localhost.localdomain - localhost.localdomain	admin	AUDIT
□	4/14/11, 5:23:53 PM, EDT	AGENT_SHUTDOWN	localhost.localdomain	admin	AUDIT
□	4/28/11, 9:59:36 AM, EDT	OPERATION_MODE_CHANGE	localhost.localdomain: DOWN --> NORMAL	admin	AUDIT
□	4/28/11, 9:59:59 AM, EDT	AGENT_CONNECT	localhost.localdomain - localhost.localdomain	admin	AUDIT
□	4/28/11, 10:00:00 AM,	AGENT_REGISTRATION	localhost.localdomain -	admin	IMMEDIATE

The partition events log can be filtered to display entries which match the type, status, or details in the event record.

There are basically four categories of partition events that are recorded.

- Affinity group changes
- Agent events
- Server events
- Partition changes

All of the recorded events are listed in Table 6, “Types of Partition Events”.

Table 6. Types of Partition Events

Partition Event	Description
Affinity Group Changes	
AFFINITY_GROUP_CHANGE	Registers a change in the agent or server assignments for an affinity group or that a group was added.
AFFINITY_GROUP_DELETE	Registers an affinity group was deleted from the JBoss ON configuration.
AGENT_AFFINITY_GROUP_ASSIGN	Registers that an agent was added to an affinity group.

Partition Event	Description
AGENT_AFFINITY_GROUP_REMOVE	Registers that an agent was removed from an affinity group.
SERVER_AFFINITY_GROUP_ASSIGN	Registers that a server was added to an affinity group.
SERVER_AFFINITY_GROUP_REMOVE	Registers that a server was removed from an affinity group.
Agent Events	
AGENT_CONNECT	Shows that a JBoss ON agent was started.
AGENT_SHUTDOWN	Shows that a JBoss ON agent was stopped.
AGENT_LEAVE	Shows that a JBoss ON agent was permanently removed from the JBoss ON configuration.
AGENT_REGISTRATION	Shows that a new JBoss ON agent was added to the JBoss ON configuration.
Server Events	
SERVER_DELETION	Shows that a JBoss ON server was permanently removed from the JBoss ON configuration.
SERVER_COMPUTE_POWER_CHANGE	
OPERATION_MODE_CHANGE	Shows that a server went stopped, was started, or was newly installed. The type also shows how the mode transitioned (such as **`server.example.com: DOWN --> NORMAL`**).
Partition Changes	
SYSTEM_INITIATED_PARTITION	Shows that JBoss ON initiated a repartition of the servers.
ADMIN_INITIATED_PARTITION	Shows that a JBoss ON user initiated a repartition of the servers.

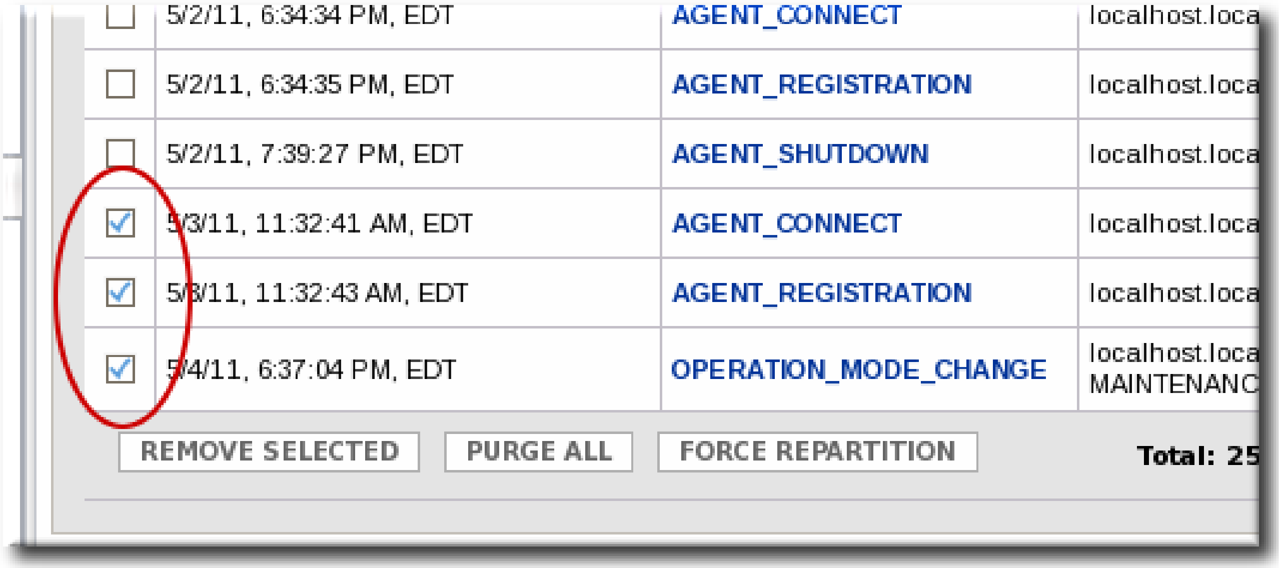
Table 7. Partition Events Entries

Field	Description
Execution Time	The time of the partition event.
Type	Shows the partition event type. This indicates what happened in the system affecting agent partition.
Details	Gives detailed information about the partition event; the type of information given varies based on the partition event type.
Initiated By	Shows the name of the user who initiated the action generating the partition event. Events initiated by the JBoss ON server itself show they were initiated by **`admin`**.
Execution Status	Shows low for status descriptions. There are four different status settings: • *Audit* shows that a change was made, but not an event that affects the partition topology. • *Immediate* shows that a partition change was made at the time of the event. • *Requested* shows that a partition change was requested and deferred until the next execution of the JBoss ON server cloud job (usually once a minute). Repartition requests usually have a requested status. • *Completed* shows that a change has been completed.

4.5.2. Removing Partition Events

There are two ways to remove partition event records:

- By selecting individual records and click **`REMOVE SELECTED`**
- By clicking the **`PURGE ALL`** to remove all events



□	5/2/11, 6:34:34 PM, EDT	AGENT_CONNECT	localhost.loca
□	5/2/11, 6:34:35 PM, EDT	AGENT_REGISTRATION	localhost.loca
□	5/2/11, 7:39:27 PM, EDT	AGENT_SHUTDOWN	localhost.loca
☑	5/3/11, 11:32:41 AM, EDT	AGENT_CONNECT	localhost.loca
☑	5/3/11, 11:32:43 AM, EDT	AGENT_REGISTRATION	localhost.loca
☑	5/4/11, 6:37:04 PM, EDT	OPERATION_MODE_CHANGE	localhost.loca MAINTENANC

Figure 3. Removing Partition Events

5. CONFIGURING SERVERS

The JBoss ON configuration is edited in one of two areas, depending on the configuration setting:

- In the JBoss ON GUI

  NOTE

  Settings that can be edited in the JBoss ON UI *must* be edited in the JBoss ON UI.

- In the **rhq-server.properties** configuration file

Additional configuration is stored in the database backend used by the JBoss ON server.

5.1. Enabling Debug Logging for the JBoss ON Server

Debug mode records debugging messages caused or encountered by the start scripts to the server logs.

Debug messages are in the log file, ***serverRoot*/jon-server-3.0.0.GA1/logs/rhq-server-log4j.log**.

In some cases, you will want debug messages from the JBoss ON server launcher scripts. To do this, you need to set the environment variable **RHQ_SERVER_DEBUG** to any value. After setting this variable when you start the launcher, scripts will output debug messages.

5.1.1. Using an Environment Variable

The quickest way to enable debug logging is to set the **RHQ_SERVER_DEBUG** environment variable to any value before starting the server.

5.1.2. Setting log4j Priorities

log4j categories support *priorities* for logging levels. This means that different areas of the agent can be configured for different log levels.

NOTE

Do not set the **RHQ_SERVER_DEBUG** environment variable if you are setting priorities in the **rhq-server-log4j.xml** file. The environment variable overrides this **rhq-server-log4j.xml** configuration.

WARNING

Do not modify anything else in the **jbossas** directory. This could adversely affect the performance of the JBoss ON server.

To enable debug logging for a category, change the priority value to **DEBUG**:

1. Open the **jboss-log4j.xml** file:

```
# vim serverRoot/jon-server-
3.0.0.GA1/jbossas/server/default/conf/jboss-log4j.log
```

2. Uncomment the **org.rhq** category to set the priority for all JBoss ON server subsystems to DEBUG.

```
    <category name="org.rhq">
       <priority value="DEBUG"/>
    </category>
```

   Alternatively, set DEBUG priorities for individual subsystems in the server. Uncomment the specific categories and set the ***priority*** element for the category to DEBUG. Many categories are defined for JBoss ON server subsystems, including database upgrades, global concurrency settings, inventory reports, authorization, alerting, and the UI. There are also categories for third-party subsystems like JBoss/Remoting and Hibernate. For example:

```
    ...
         <!--
    <category name="org.rhq.enterprise.server.alert">
      <priority value="DEBUG"/>
    </category>
    -->

    <!--
    <category name="org.rhq.enterprise.server.authz">
      <priority value="DEBUG"/>
    </category>
    -->

    <!--
```

```
<category name="org.rhq.enterprise.server.event">
  <priority value="DEBUG"/>
</category>
-->

<!--
<category name="org.rhq.enterprise.server.measurement">
  <priority value="DEBUG"/>
</category>
-->

...
```

NOTE

By default, the console window will not display debug messages. This is because the log4j **CONSOLE** appender has a threshold at**INFO**. For debug messages to appear in the UI, change the **CONSOLE** appender's threshold setting to **DEBUG**.

```
   <appender name="CONSOLE"
class="org.apache.log4j.ConsoleAppender">
      <errorHandler
class="org.jboss.logging.util.OnlyOnceErrorHandler"/>
      <param name="Target" value="System.out"/>
    <param name="Threshold" value="DEBUG"/>
    ....
```

3. Restart the server to load the new configuration.

```
serverRoot/jon-server-3.0.0.GA1/bin/rhq-server.sh stop
```

The **log4j** file format is described more in theApache log4j documentation.

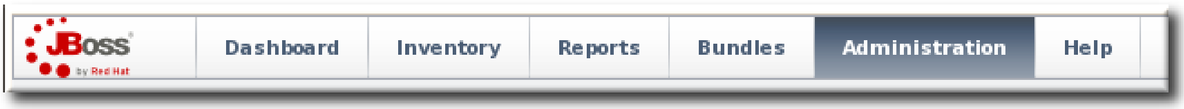
5.2. Changing the JBoss ON Server URL

The server URL is the URL used to identify and connect to the server in two ways:

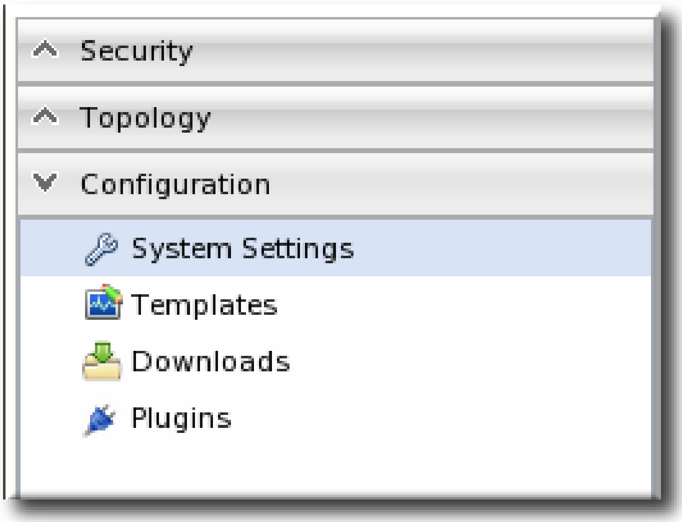
- Connecting to the GUI
- Details on alerts, contained in email notifications of alerts

The server URL does not need to be changed unless the JBoss ON connects to the Internet through a proxy server.

1. Click the **Administration** tab in the top menu.

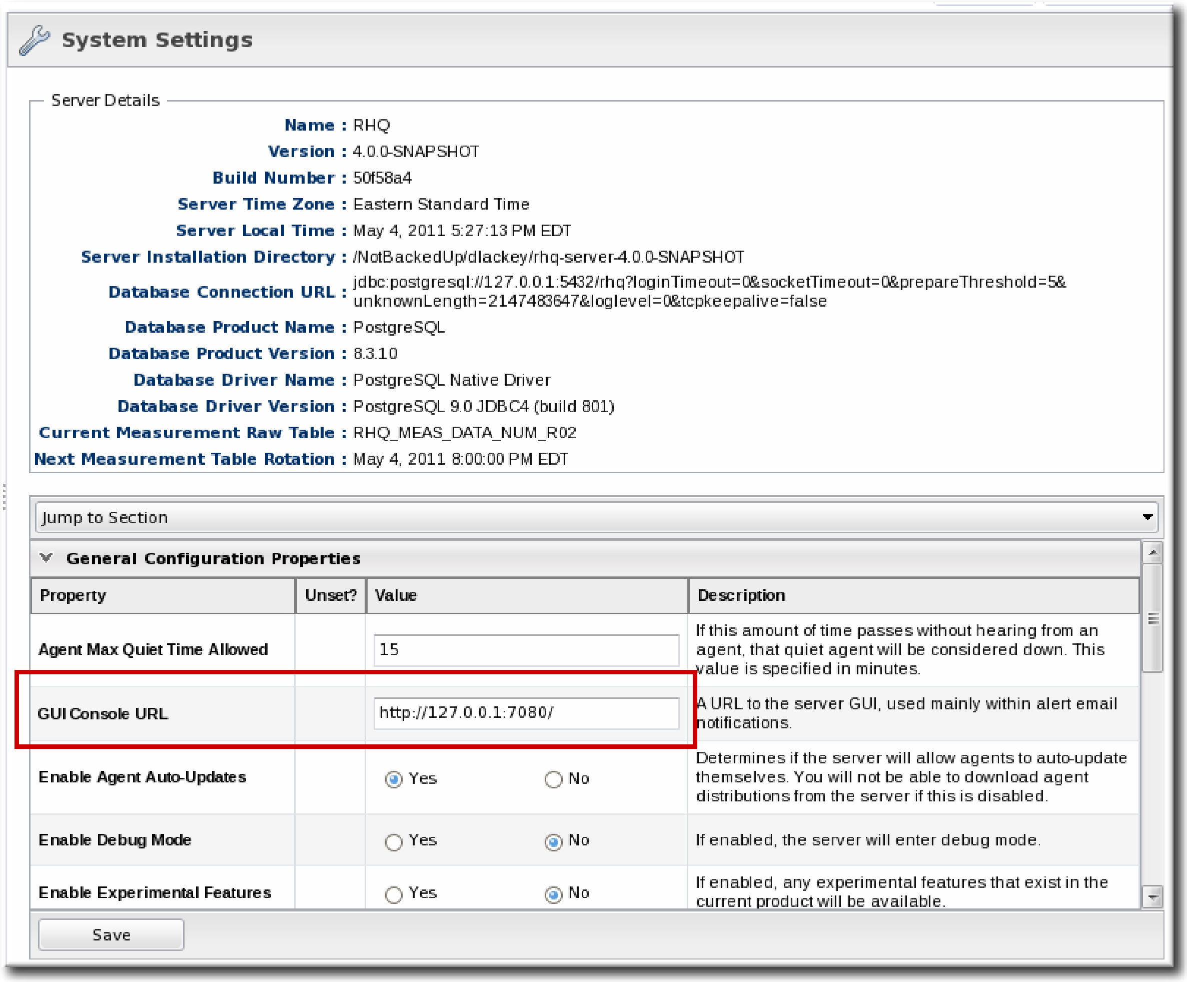


2. In the **Configuration** menu table on the left, select the**System Settings** item.



3. Scroll to the **`JON General Configuration Properties`** section in the main work area.

4. Change the hostname or IP address in the **`GUI Console URL`** field.


System Settings

Server Details

Name :	RHQ
Version :	4.0.0-SNAPSHOT
Build Number :	50f58a4
Server Time Zone :	Eastern Standard Time
Server Local Time :	May 4, 2011 5:27:13 PM EDT
Server Installation Directory :	/NotBackedUp/dlackey/rhq-server-4.0.0-SNAPSHOT
Database Connection URL :	jdbc:postgresql://127.0.0.1:5432/rhq?loginTimeout=0&socketTimeout=0&prepareThreshold=5&unknownLength=2147483647&loglevel=0&tcpkeepalive=false
Database Product Name :	PostgreSQL
Database Product Version :	8.3.10
Database Driver Name :	PostgreSQL Native Driver
Database Driver Version :	PostgreSQL 9.0 JDBC4 (build 801)
Current Measurement Raw Table :	RHQ_MEAS_DATA_NUM_R02
Next Measurement Table Rotation :	May 4, 2011 8:00:00 PM EDT

General Configuration Properties

Property	Unset?	Value	Description
Agent Max Quiet Time Allowed		15	If this amount of time passes without hearing from an agent, that quiet agent will be considered down. This value is specified in minutes.
GUI Console URL		http://127.0.0.1:7080/	A URL to the server GUI, used mainly within alert email notifications.
Enable Agent Auto-Updates		● Yes ○ No	Determines if the server will allow agents to auto-update themselves. You will not be able to download agent distributions from the server if this is disabled.
Enable Debug Mode		○ Yes ● No	If enabled, the server will enter debug mode.
Enable Experimental Features		○ Yes ● No	If enabled, any experimental features that exist in the current product will be available.

5. Click **Save**.

5.3. Editing JBoss ON Server Configuration in rhq-server.properties

JBoss ON server configuration properties are stored either in the **`rhq-server.properties`** configuration file in the ***`serverRoot`*`/jon-server-3.0.0.GA1/bin`** directory or in the JBoss

ON database. The configuration file contains most of the basic information about the JBoss ON server, such as the TCP/IP ports it listens on and its hostname or IP address.

The JBoss ON server configuration is loaded from the **`rhq-server.properties`** file when the server starts. The initial configuration is defined by the installer when the JBoss ON program is set up.

NOTE

Because the configuration properties are loaded from **`rhq-server.properties`** when the JBoss ON server starts up, you have to restart the JBoss ON server after making any changes to that configuration file so the new settings are loaded.

5.3.1. Configuring Communication Settings

JBoss ON servers are configured to communicate to agents by defining and identifying ways that the server and agent can connect, as well as how other clients (like users accessing the JBoss ON GUI) can connect to the server. These communication endpoints include identifying the server hostname or IP address, ports that the server listens on for different types of messages, and protocols used to access the server. All of the server connection parameters are described in Table 8, “rhq-server.properties Parameters for Server Connections”.

Connections, or transport methods, for the server are implemented through servlets (HTTP and HTTPS) or sockets (HTTPS). The **`servlet`** (HTTP) and **`sslservlet`** (HTTPS) transports route traffic through the Tomcat server embedded in the JBoss ON server.

IMPORTANT

If the server is using the transport **`servlet`** or **`sslservlet`**, the agent communication is over the Tomcat connector. This means ***`rhq.communications.connector.bind-port`*** is ignored and the Tomcat connector port is used to send messages from agent to server. By default, the Tomcat connector port is 7080 (servlet) or 7443 (sslservlet).

NOTE

Servlet-based transports leverage the Tomcat connector infrastructure to handle both agent and GUI requests. Using servlets, however, limits agents and GUI clients to use the same connection methods; for certificate-based SSL connections, servlets require users to authenticate to the GUI using a stored browser certificate. For SSL, then, it may be preferable to use socket connections, which allow different authentication methods for agent and GUI sessions.

See Section 3.2, “Setting up Client Authentication Between Servers and Agents” for setting up SSL sockets.

The general configuration settings set the port numbers that users can used to access the server.

```
# General Properties
rhq.server.startup.web.http.port=7080
rhq.server.startup.web.https.port=7443
```

Additional connection settings can be added to configure SSL connections for inbound connections to the server (messages from the agent to the server) and outbound connections (messages from the server to the agent). For example:

```
rhq.server.tomcat.security.client-auth-mode=want
rhq.server.tomcat.security.secure-socket-protocol=TLS
rhq.server.tomcat.security.algorithm=SunX509
rhq.server.tomcat.security.keystore.alias=RHQ
rhq.server.tomcat.security.keystore.file=conf/rhq.keystore
rhq.server.tomcat.security.keystore.password=RHQManagement
rhq.server.tomcat.security.keystore.type=JKS
rhq.server.tomcat.security.truststore.file=conf/rhq.truststore
rhq.server.tomcat.security.truststore.password=RHQManagement
rhq.server.tomcat.security.truststore.type=JKS
```

The third part of JBoss ON server communications provides more control over information the connection endpoints for JBoss ON servers and agents to use to talk with each other. These are *transport* parameters for the server. Both the JBoss ON agent and port use the same remoting framework, using invocation strings that are similar to URLs. These connection strings have the format:

```
protocol://server:port/transportParm1=value1&transportParam2=value2
```

IMPORTANT

For most communications, the JBoss ON server must use either servlet or sslservlet protocols. The only instance where socket can be used is for passing transport parameters. Otherwise, socket and sslsocket are not supported.

The transport configuration first sets up connectors (called *endpoints*) that the servers and agents jointly define and use for communications. This means that the same information must be in both the server and agent configuration files. Each aspect of the remoting URL is built using separate server configuration parameters.

The standard server configuration has four parts, for the transport method, server IP address, agent port, and any parameters to append to the URL. For example:

```
rhq.communications.connector.transport=servlet
rhq.communications.connector.bind-address=192.168.1.2
rhq.communications.connector.bind-port=16163
rhq.communications.connector.transport-params=/jboss-remoting-servlet-
invoker/ServerInvokerServlet
```

That standard configuration is merged to create this URL:

```
servlet://192.168.1.2:16163/jboss-remoting-servlet-
invoker/ServerInvokerServlet
```

A corresponding entry, with the same endpoint definition, is also listed in the agent configuration so that it knows how to send communications to the server, such as sending registration and availability reports.

```
RHQ Server IP Address=192.168.1.2
RHQ Server Port=16163
RHQ Server Transport Protocol=servlet
RHQ Server Transport Parameters=/jboss-remoting-servlet-
invoker/ServerInvokerServlet
```

> **Example 5. Basic Server-Agent Transport Example**
>
> A server with an IP address of **`192.168.0.10`** will connect to agents over the standard agent port of 16163. The server configuration has the following configuration to define the server address (***`rhq.communications.connector.bind-address`***), the agent communications port (**`rhq.communications.connector.bind-port`**), and the connection protocol (***`rhq.communications.connector.transport`***):
>
> ```
> rhq.communications.connector.transport=servlet
> rhq.communications.connector.bind-address=192.168.0.10
> rhq.communications.connector.bind-port=16163
> rhq.communications.connector.transport-
> params=enableTcpNoDelay=true&backlog=200
> ```
>
> The connection URL, then, is:
>
> ```
> servlet://192.169.0.10:16163/enableTcpNoDelay=true&backlog=200
> ```
>
> The JBoss ON agent configuration will contain corresponding entries which match the server configuration:
>
> ```
> RHQ Server IP Address=192.168.0.10
> RHQ Server Port=16163
> RHQ Server Transport Protocol=socket
> RHQ Server Transport Parameters=enableTcpNoDelay=true&backlog=200
> ```

Transport parameters can pass relevant information about both incoming and outgoing messages (called server and client messages, respectively, because of how the JBoss ON server handles the messages). These transport parameters are strung together with ampersands (&), as with a standard web URL parameters.

Both server and client transport parameters are passed in the same URL; the JBoss ON server only processes whatever parameters are relevant for the current operation. In Example 5, “Basic Server-Agent Transport Example”, for instance, the configuration sets two transport parameters, **`enableTcpNoDelay`** (client) and **`backlog`** (server). When the JBoss ON server is receiving messages — when it function as a communications server — it uses the **`backlog`** parameter and ignore **`enableTcpNoDelay`** because **`enableTcpNoDelay`** is only for outgoing (client) messages.

Table 8. rhq-server.properties Parameters for Server Connections

Parameter	Description
General Connection Parameters	
jboss.bind.address[a][b]	Gives the IP address for the JBoss ON GUI console, among other services, to bind to. This is the host in the JBoss ON GUI URLs; e.g. the myhost in http://myhost:7080.
rhq.server.startup.web.http.port[a][b]	Gives the port that the JBoss ON GUI listens to for unsecured HTTP requests. This is the port number in the JBoss ON GUI URLs, such as the 7080 in *http://localhost:7080*. This is also the unsecure port used as the endpoint in high availability.
rhq.server.startup.web.https.port[a][b]	Gives the port that the JBoss ON GUI listens to for secured HTTPS requests. This is the port number in the JBoss ON GUI URLs, such as the 7443 in *https://localhost:7443*. This is also the secure port used as the endpoint in high availability.
rhq.server.startup.keystore.filename[b]	The JBoss ON GUI can accept browser requests over HTTPS. If you want to authenticate the JBoss ON GUI to remote browsers, you need to put an SSL certificate in a keystore file. This setting points to the location of the keystore file. Note that this file name must be a relative file path relative to the ***<JBoss ON server Install Dir>*/jbossas/server/default/conf** directory. The JBoss ON server ships with a self-signed certificate in a default keystore file.
rhq.server.startup.keystore.password[b]	The password of the keystore file. This is so the JBoss ON GUI can access the keystore file in order to be able to serve the certificate to browser clients.
rhq.server.startup.keystore.sslprotocol[b]	The protocol that browser clients should use to communicate with the JBoss ON GUI.
rhq.server.maintenance-mode-at-start	Sets whether to start the server in maintenance mode (true) or whether to start the server in whatever mode it was in when it shut down (false). The default is false.

Parameter	Description
- rhq.server.startup.webservice.port[a][b] - rhq.server.startup.namingservice.port[a][b] - rhq.server.startup.namingservice.rmiport[a][b] - rhq.server.startup.jrmpinvoker.rmiport[a][b] - rhq.server.startup.pooledinvoker.rmiport[a][b] - rhq.server.startup.ajp.port[a][b] - rhq.server.startup.unifiedinvoker.port[a][b] - rhq.server.startup.aspectdeployer.bind-port[a][b]	Ports used by internal services.
SSL Connection Parameters	
- rhq.communications.connector.security.secure-socket-protocol (agent to server) - rhq.server.client.security.secure-socket-protocol (server to agent)	The secure protocol that agents must use when communicating with this JBoss ON server.
- rhq.communications.connector.security.keystore.file (agent to server) - rhq.server.client.security.keystore.file (server to agent)	The keystore file that contains a certificate that authenticates the JBoss ON server to the agents.
- rhq.communications.connector.security.keystore.algorithm (agent to server) - rhq.server.client.security.keystore.algorithm (server to agent)	

Parameter	Description
- rhq.communications.connector.security.keystore.type (agent to server) - rhq.server.client.security.keystore.type (server to agent)	The file format of the keystore.
- rhq.communications.connector.security.keystore.password (agent to server) - rhq.server.client.security.keystore.password (server to agent)	The password that is used to gain access to the keystore file.
- rhq.communications.connector.security.keystore.key-password (agent to server) - rhq.server.client.security.keystore.key-password (server to agent)	The password that is used to gain access to the key inside the keystore.
- rhq.communications.connector.security.keystore.alias (agent to server) - rhq.server.client.security.keystore.alias (server to agent)	The alias that identifies the JBoss ON server's key within its keystore.
- rhq.communications.connector.security.truststore.file (agent to server) - rhq.server.client.security.truststore.file (server to agent)	The truststore file that contains certificates that this JBoss ON server trusts. If you need the JBoss ON server to authenticate JBoss ON agents, you must set this; otherwise it is not needed. This truststore contains certificates for all JBoss ON agents that need to communicate with this JBoss ON server. Refer to the *Incoming Client Authentication Mode*.
- rhq.communications.connector.security.truststore.algorithm (agent to server) - rhq.server.client.security.truststore.algorithm (server to agent)	

Parameter	Description
- rhq.communications.connector.security.truststore.type (agent to server) - rhq.server.client.security.truststore.type (server to agent)	The file format of the truststore file.
- rhq.communications.connector.security.truststore.password (agent to server) - rhq.server.client.security.truststore.password (server to agent)	The password that is used to gain access to the truststore file.
- rhq.communications.connector.security.client-auth-mode (agent to server) - rhq.server.client.security.server-auth-mode-enabled (server to agent)	Indicates if the JBoss ON server must authenticate the JBoss ON agents that are sending it messages. If the server is using secure connections, but does not have trusted certificates for all of the JBoss ON agents in a truststore, set this to none. The valid values are none, want, or need.
Transport Connection Parameters	

Parameter	Description
rhq.communications.connector.transport	Defines how the JBoss ON agents need to transport messages to the JBoss ON server. The allowed values are either servlet or sslservlet. The agent requests go through the JBoss ON server web application layer (i.e. the secure Tomcat Connector). With sslservlet, not only do agent requests route through the web application layer, but they are also secured through the secure Tomcat Connector. The keystore used for incoming agent message authentication is the same as that configured in ***`rhq.communications.connector.security.keystore.file`***. **NOTE** This transport setting does *not* restrict agents from only going over that particular connection method. By default, the JBoss ON server always deploys the communications connector that allows for both servlet and sslservlet traffic. This setting tells the agent to decide what transport is used when it sends messages to the server. If the server has its transport set to servlet, but the agent is configured to talk to the server via sslservlet, the messages the agent sends will be via sslservlet.
rhq.communications.connector.bind-address	This is the address that is placed in the server's JBoss/Remoting locator URL. This defines the endpoint that the JBoss ON server will bind its connector to. It also represents the public endpoint address that all agents can use to connect to the server.

Parameter	Description
rhq.communications.connector.bind-port	Defines the endpoint that the JBoss ON server binds to, as well as the public address that all agents can use to connect to the server. This is hidden from view in the installer, although it still appears in the **`rhq-server.properties`** file. This value can be blank; the server sets this to either the HTTP or HTTPS port, depending on the transport configured for the server.
rhq.communications.connector.transport-params	Defines additional transport parameters the JBoss ON server will set on its connector that will accept incoming messages from the JBoss ON agents. All of the possible transport parameters are listed in Table 9, “Transport Parameters”.
rhq.communications.multicast-detector.enabled	If true, the JBoss ON server will attempt to auto-detect JBoss ON agents coming online and going offline using multicast detection. Your network must support multicast traffic for this to work.
rhq.communications.multicast-detector.bind-address	The address that the multicast detector directly binds to. This is not used, or needed, if you have not enabled multicast detection.
rhq.communications.multicast-detector.multicast-address	The address that the multicast detector will broadcast messages to. This is not used, or needed, if you have not enabled multicast detection.
rhq.communications.multicast-detector.port	The port that the multicast detector will broadcast messages to. This is not used, or needed, if you have not enabled multicast detection.

[a] These settings configure specific IP addresses and ports for the JBoss ON server instance. If there are firewall issues the require different settings, then these parameters can be changed.

[b] The JBoss ON server has to be restarted for any changes to this value to take effect.

Table 9. Transport Parameters

Transport Parameter	Description	For Incoming Messages or for Outgoing Messages

Transport Parameter	Description	For Incoming Messages or for Outgoing Messages
serverBindAddress	The address on which the server socket binds to listen for requests. The default is an empty value which indicates the server socket should be bound to the host provided by the InvokerLocator URL (the host).	Incoming
serverBindPort	The port to listen for requests on.	Incoming
timeout	The socket timeout value. The default on the server side is 60000 (one minute). If the timeout parameter is set, its value will also be passed to the client-side (see below).	Incoming
backlog	The preferred number of unaccepted incoming connections allowed at a given time. The actual number may be greater than the specified backlog. When the queue is full, further connection requests are rejected. Must be a positive value greater than 0. If the value passed if equal or less than 0, then the default value will be assumed. The default value is 200.	Incoming
numAcceptThreads	The number of threads that exist for accepting client connections. The default is 1.	Incoming
maxPoolSize	The number of server threads for processing client requests. The default is 300.	Incoming

Transport Parameter	Description	For Incoming Messages or for Outgoing Messages
socket.check_connection	Indicates if the invoker should try to check the connection before re-using it by sending a single byte ping from the client to the server and then back from the server. This configuration needs to be set on both the client and server to work. The default value is false.	Incoming
clientConnectAddress	The IP address or hostname the client will use to connect to the server-side socket. This would be needed in the case that the client will be going through a router that forwards requests made externally to a different IP address or hostname internally. If no clientConnectAddress or serverBindAddress is specified, the local host's address is used.	Outgoing
clientConnectPort	The port the client will use to connect to the server-side socket. This would be needed in the case that the client will be going through a router that forwards requests made externally to a different port internally.	Outgoing
timeout	The socket timeout value. The default on the client side is 1800000 (or 30 minutes).	Outgoing
enableTcpNoDelay	Indicates if the client socket should have TCP_NODELAY turned on or off. TCP_NODELAY is for a specific purpose; to disable the Nagle buffering algorithm. It should only be set for applications that send frequent small bursts of information without getting an immediate response. The default is false.	Outgoing

Transport Parameter	Description	For Incoming Messages or for Outgoing Messages
clientMaxPoolSize	The client-side maximum number of active socket connections. This basically equates to the maximum number of concurrent client calls that can be made from the socket client invoker. The default is 50.	Outgoing
numberOfRetries	The number of retries to get a socket from the pool. This basically equates to the number of seconds the client will wait to get a client socket connection from the pool before timing out. If the max retries is reached, a CannotConnectException will be thrown. The default is 30.	Outgoing
numberOfCallRetries	The number of retries for making the invocation. This is unrelated to numberOfRetries in that when this comes into play is after it has already received a client socket connection from the pool. However, it is possible that the socket connection timed out while waiting within the pool. Since a connection check is not done by default, the connection is thrown away and an attempt to get a new one will be made. This will happen for however many numberOfCallRetries is (which defaults to 3). However, when (numberOfCallsRetries - 2) is reached, the entire connection pool is flushed under the assumption that all connections in the pool have timed out and are invalid and will start over by creating a new connection. If this still fails, a MarshalException is thrown.	Outgoing

Transport Parameter	Description	For Incoming Messages or for Outgoing Messages
socket.check_connection	Indicates if the invoker should try to check the connection before re-using it by sending a single byte ping from the client to the server and then back from the server. This configuration needs to be set on both client and server to work. This if false by default.	Outgoing

5.3.2. Setting Concurrency Limits

JBoss ON can handle large numbers of agents, potentially hundreds. The JBoss ON server can possibly be flooded with messages if many agents attempt to communicate with the server simultaneously. This can happen if the JBoss ON server is restarted after being down for a period of time; when JBoss ON agents detect that the JBoss ON server has come back, they all immediately attempt to send it a backlog of messages.

The JBoss ON server can have a configurable limit on the number of concurrent messages that can be processed at one time, to mitigate any risk of flooding the server. Any messages that come in past that limit are dropped and the agent is asked to send them later.

All of the concurrency-related parameters are listed in Table 10, “rhq-server.properties Parameters for Concurrency Limits”.

Concurrency limits not only limit the number of agent connections, but also the number of connections to the GUI and other web connections to the server. There are three primary parameters that control the concurrency limits:

- A global limit on the total number of incoming messages to the server (***rhq.communications.global-concurrency-limit***).

  This is the total number of allowed agent connections. There are other concurrency limits for specific message types which can help tune performance for content downloads, inventory synchronization, and other resource-intensive or recurring agent operations. Those concurrency limits apply only to those specific message types, and those limits are evaluated independently of each other. The global concurrency limit is the total cap for *all* agent conenctions. This is the effective concurrency limit, even if the sum of the other concurrency limits is higher.

- A limit on the total number of concurrent web connections allowed (***rhq.server.startup.web.max-connections***).

  This counts any client connection which connects to the JBoss ON server over an HTTP or HTTPS connection. This includes web GUI connections, of course, but it also includes all agent connections which use the (default) **servlet** or **ssslservlet** transports.

  The limit on web connections is the same for both non-secured HTTP requests and HTTPS requests, but the limit is additive so HTTP and HTTPS connections count against different pools. The *total* maximum connections allowed is actually twice whatever the ***rhq.server.startup.web.max-connections*** value is. For example, if the setting is 300, then 300 HTTP requests are allowed and 300 HTTPS requests are allowed, for total of 600 concurrent web connections.

- Limits on the number of downloads from agents (***rhq.server.agent-downloads-limit***) and from other clients (***rhq.server.client-downloads-limit***).

Example 6. Concurrency Limits

```
rhq.server.startup.web.max-connections=200
rhq.server.agent-downloads-limit=45
rhq.server.client-downloads-limit=5
rhq.communications.global-concurrency-limit=30
```

Table 10. rhq-server.properties Parameters for Concurrency Limits

Parameter	Description
rhq.server.startup.web.max-connections	Sets a limit on the number of web connections that can be concurrently created, including both connections to the GUI and connections by agents. **NOTE** If agent requests are routed over web connections, make sure that the ***rhq.communications.global-concurrency-limit*** value is slightly lower than the web connections limit. Otherwise, GUI users could be blocked from accessing the JBoss ON UI whenever there is a high agent load. The limit on web connections is the same for both HTTP and HTTPS (secure) requests, so the *total* max connections allowed is actually twice what this setting is. For example, if the max web connections is set to 300, then 300 HTTP requests will be allowed and 300 HTTPS requests will be allowed, for a total of 600 concurrent web connections.

Parameter	Description
rhq.communications.global-concurrency-limit	Sets the total number of agent messages that come into the server. This only affected incoming agent messages, not GUI requests. If this global concurrency limit is set to 300, no more than 300 total agent messages can be processed at any one time, regardless of what kinds of messages are coming in. Even if the sum of the other concurrency limits are higher than this global limit, they are capped at this global limit since there can never be more messages processed than the global limit. This value should be slightly lower than the number of allowed web connections so that web connections to the GUI are not blocked when there is a high agent load.
rhq.server.concurrency-limit.inventory-report	Inventory reports are sent from the agent when the agent starts up, and periodically thereafter. Inventory reports can be large, depending on the number of resources on the agent machine.
rhq.server.concurrency-limit.availability-report	Availability reports are regularly sent from the agent, typically every 60 seconds. Availability reports are usually very small, but occur in large numbers due to the high frequency of their transmission.
rhq.server.concurrency-limit.inventory-sync	Inventory synchronizations occur when the agent needs to synchronize its inventory with that of the server. Agents typically synchronize at startup. Traffic that flows as part of inventory synchronizations is usually large, depending upon the number of resources managed by the agent.
rhq.server.concurrency-limit.content-report	Content reports are similar to inventory reports except they contain information about discovered content (i.e., installed packages of software). These reports can be large depending on the number of installed software the agent has discovered and is managing.
rhq.server.concurrency-limit.content-download	Content downloads occur when a resource on an agent needs to ask for the content of a package version, usually for the purpose of installing the package.

Parameter	Description
rhq.server.concurrency-limit.measurement-report	Measurement reports are periodically sent to the server whenever the agent completes measurement collections. The number and size of measurement reports can vary, depending on the number and frequency of measurements scheduled to be collected. The greater the number of schedule measurements the agent needs to collect, the more frequently measurement reports are sent, and the larger the reports will be.
rhq.server.concurrency-limit.measurement-schedule-request	Similar to inventory synchronization, measurement schedule requests are sent to the agent asking the server for an up-to-date set of measurement schedules that have to be collected.

5.3.3. Configuring the SMTP Server for Email Notifications

Each JBoss ON server talks to a specific SMTP server. The SMTP server is defined in the **rhq-server.properties** file. The default configuration points to the local JBoss ON server hosts.

```
# Email
rhq.server.email.smtp-host=localhost
rhq.server.email.smtp-port=25
rhq.server.email.from-address=rhqadmin@localhost
```

These settings can be edited to use a different SMTP server or email account.

NOTE

To confirm that the SMTP settings are correct and the server can send emails successfully, go to the test email page at **http://***server***/admin/test/email.jsp**.

Table 11. rhq-server.properties Parameters for SMTP

Parameter	Description
rhq.server.email.smtp-host	Sets the hostname of the SMTP server used by the JBoss ON server.
rhq.server.email.smtp-port	Sets the port of the SMTP server used by the JBoss ON server.
rhq.server.email.from-address	Sets the address to use for the From header of all emails sent by the JBoss ON server.

5.3.4. Installing a Server Silently

Some options in the **`rhq-server.properties`** file tell the installation process to load the server configuration from the file rather than the from the web-based installer.

```
# Auto-Install Pre-Configuration Settings
rhq.autoinstall.enabled=false
rhq.autoinstall.database=auto
rhq.autoinstall.public-endpoint-address=
```

These settings are only used for installation.

Table 12. rhq-server.properties Parameters for Silent Installation

Parameter	Description
rhq.autoinstall.enabled	Tells the installation process whether to load the configuration from the **`rhq-server.properties`** file (**`true`**) or from the web-based installer (**`false`**).
rhq.autoinstall.database	Tells the install process how to load or add database schema. There are three options: • **`auto`** creates a new schema for new installation or upgrades existing schema without overwriting the data. • **`overwrite`** overwrites the database and creates a new, empty schema. • **`skip`** skips the entire database process so no database is created or updated.
rhq.autoinstall.public-endpoint-address	Sets the IP address or hostname to use for the server. If no value is given, then the server detects and sets its own value when it starts.

5.4. Synchronizing Server Configuration

Even in different environments, JBoss ON servers can share a lot of the same configuration. For example, different JBoss ON servers may manage a development environment, staging environment, and production environment, yet on all three, the servers use similar metric templates and configuration settings.

To simplify managing separate but similar environments, JBoss ON can export the configuration for a server and then import that configuration into another server.

Any user with permissions to manage settings can export the server configuration. There are two categories of data:

- System settings, which include how long alerts, events, and monitoring metrics are stored; the baseline calculation schedule; and the LDAP server configuration.

- Metric collection settings for each resource types.

The information is exported to dumped to a gzipped XML file, which can be easily edited before being imported into another server.

NOTE

Syncing server configuration is only necessary when servers use different backend databases. Servers which share a database (in the high availability cloud) already share their configuration.

Import and export operations are only done through the JBoss ON CLI. This API is available with the other JBoss ON documentation. Running the CLI is covered more inRunning JBoss ON Command-Line Scripts.

5.4.1. Exporting a Server's Configuration

1. Log into the JBoss ON CLI.

   ```
   [root@server bin]#  installDir/bin/rhq-cli.sh -u rhqadmin -p
   rhqadmin
   ```

2. Export the data to a database object:

   ```
   rhqadmin@localhost:7080$ var ex =
   SynchronizationManager.exportAllSubsystems();
   ```

3. Convert that object into an export file. The file extension should be **`.xml.gz`** because the export format is a GZIP'ed XML file.

   ```
   rhqadmin@localhost:7080$ saveBytesToFile(ex.exportFile,
   'export.xml.gz');
   ```

NOTE

The user must have the manage settings permission to export the server data.

5.4.2. Importing a Server's Configuration

Server configuration is exported into an XML file. Administrators can edit this file to control what kind of information is imported into the other JBoss ON servers, so there is a lot of adaptability in the import process. When the file is imported, it first runs through a series of validation tests to make sure that the configuration data can actually be imported into the server. Then, two classes or *synchronizers*, one for system settings and one for metric templates, are used to import the data.

The import process can be changed by administrators, so there are several common import scenarios:

- The configuration data are imported directly into the server, using all of the default settings.

- The XML file can be edited so that the configuration values are adapted to the target JBoss ON servers.
- The synchronizer behavior is changed, which changes what data elements are imported.

5.4.2.1. Editing the XML Import File

All of the data are dumped to a single XML file, which contains the system settings, metric settings for each resource type, and some processing instructions.

The configuration entries all defined in two large **`<entities>`** elements.

Metric templates list each metric separately in individual **`<entity>`** elements, with the metric itself identified by its name, resource type, and plug-in as arguments for the element. The entity ID identifies the template in the JBoss ON database, but is ignored during import because the IDs do not need to match between servers.

```
<entities id="org.rhq.enterprise.server.sync.MetricTemplateSynchronizer">
 <entity>
  <data>
   <metricTemplate
    enabled="false"
    defaultInterval="300000"
    perMinute="false"
    metricName="trap_count"
    resourceTypePlugin="snmptrapd"
    resourceTypeName="SnmpTrapd"
    referencedEntityId="10001">
   </metricTemplate>
  </data>
 </entity>
 .....
```

System settings, on the other hand, are all defined in a single **`<entity>`** element, and each configuration parameter is given as a key on the entry. Not all of these keys are imported into the target server; the keys which are imported depend on the synchronizer configuration.

```
<entities id="org.rhq.enterprise.server.sync.SystemSettingsSynchronizer">
 <entity>
  <data>
   <systemSettings referencedEntityId="0">
    <entry key="CAM_BASE_URL">http://10.16.65.121:7080/</entry>
    <entry key="CAM_DATA_PURGE_6H">2678400000</entry>
    <entry key="CAM_LDAP_BIND_DN"></entry>
    .....
   </systemSettings>
  </data>
 </entity>
</entities>
```

5.4.2.2. Changing the Synchronizer Configuration

JBoss ON uses *synchronizers* to set what elements — like what metric schedules — are imported into the JBoss ON server and how to apply them to the server. The synchronizer has a default template which applies configuration changes programmatically to every import operation. There is also synchronizer configuration in the exported XML file, which are applied to that specific import operation.

NOTE

Custom settings in the XML file override the programmatic template settings. Programmatic settings passed with the CLI commands override the settings in the XML file.

To print the configuration for a specific synchronizer, specify the synchronizer name in **`SynchronizationManager.getImportConfigurationDefinition()`**. For example:

```
rhqadmin@localhost:7080$ var configDef =
SynchronizationManager.getImportConfigurationDefinition('org.rhq.enterpris
e.server.sync.SystemSettingsSynchronizer')
```

To print all of the configuration for both synchronizers:

```
rhqadmin@localhost:7080$ var configDefs =
SynchronizationManager.importConfigurationDefinitionOfAllSynchronizers
rhqadmin@localhost:7080$ configDef = configDefs.get(0)

rhqadmin@localhost:7080$
pretty.print(configDef.configurationDefinition.defaultTemplate.configurati
on)
```

5.4.2.2.1. Changing the Synchronizer Settings in the XML File

The simplest way to customize the synchronizer configuration is to change the configuration in the exported XML file. The settings and metrics synhcronizers use XML elements that are very similar to the resource plug-in configuration. The root element for a synchronizer is **`<default-configuration>`**, and the configuration settings are listed as properties within that element.

The settings synchronizer has the simplest configuration. It has a single **`<ci:simple-property>`** element, and the list of settings to import is given in the***value=*** flag on the **`<ci:simple-property>`** element.

```
<default-configuration>
    <ci:simple-property value="AGENT_MAX_QUIET_TIME_ALLOWED,
ENABLE_AGENT_AUTO_UPDATE, ENABLE_DEBUG_MODE, ENABLE_EXPERIMENTAL_FEATURES,
CAM_DATA_PURGE_1H, CAM_DATA_PURGE_6H, CAM_DATA_PURGE_1D,
CAM_DATA_MAINTENANCE, DATA_REINDEX_NIGHTLY, RT_DATA_PURGE, ALERT_PURGE,
EVENT_PURGE,
TRAIT_PURGE, AVAILABILITY_PURGE, CAM_BASELINE_FREQUENCY,
CAM_BASELINE_DATASET" type="string" name="propertiesToImport">
        <c:description>The names of the properties that should be
imported. Note that these are the INTERNAL names as used in the RHQ
database</c:description>
    </ci:simple-property>
</default-configuration>
```

NOTE

The values for the settings are the names used in the JBoss ON database for the server settings.

The metrics schedules settings are much more complex because the potential metrics schedules are different for each resource. A metric schedule can be defined in any of three ways (or a combination):

- A simple list, which has a **<ci:list-property>** list members defined by a property (**<ci:simple-property>**) and a list of values

```
<default-configuration>
    <ci:list-property name="my-list">
        <c:simple-property name="element" type="string"/>
        <ci:values>
           <ci:simple-value value="a"/>
           <ci:simple-value value="b"/>
           <ci:simple-value value="c"/>
        </ci:values>
    </ci:list-property>
</default-configuration>
```

- A map of values, which is very similar to a simple list in that it uses a list of properties (**<ci:simple-property>**) and a corresponding list of values (**<ci:simple-value>**), except that each value corresponds to a single, specified property based on the name

```
<default-configuration>
    <ci:map-property name="my-map">
        <c:simple-property name="prop1" type="integer"/>
        <c:simple-property name="prop2" type="string"/>
        <c:simple-property name="prop3" type"boolean"/>
        <ci:values>
            <ci:simple-value property-name="prop1" value="1"/>
            <ci:simple-value property-name="prop2" value="abc"/>
            <ci:simple-value property-name="prop3" value="true"/>
        </ci:values>
    </ci:map-property>
</default-configuration>
```

- A table, which is a list of maps. Each set of maps specifies one table in the row.

```
<default-configuration>
    <ci:list-property name="table">
        <c:map-property name="row">
            <c:simple-property name="column1" type="integer"/>
            <c:simple-property name="column2" type="boolean"/>
            <c:simple-property name="column3" type="string"/>
        </c:map-property>
        <ci:values>
            <ci:map-value>
               <ci:simple-value property-name="column1" value="1"/>
               <ci:simple-value property-name="column2"
```

```
value="true"/>
               <ci:simple-value property-name="column3" value="a"/>
            </ci:map-value>
            <ci:map-value>
               <ci:simple-value property-name="column1" value="2"/>
               <ci:simple-value property-name="column2"
value="true"/>
               <ci:simple-value property-name="column3" value="b"/>
            </ci:map-value>
            <ci:map-value>
               <ci:simple-value property-name="column1" value="3"/>
               <ci:simple-value property-name="column2"
value="false"/>
               <ci:simple-value property-name="column3" value="c"/>
            </ci:map-value>
        </ci:values>
    </ci:list-property>
</default-configuration>
```

For example, this uses a map to import only the metric schedule for the free memory metric for a JBoss AS 5 server:

```
<default-configuration>
    <ci:simple-property value="false" type="boolean"
name="updateAllSchedules" />
    <ci:list-property name="metricUpdateOverrides">
        <c:map-property summary="false" required="true" readOnly="false"
name="metricUpdateOverride">
            <c:simple-property type="string" summary="false"
required="true" readOnly="false" name="metricName" />
            <c:simple-property type="string" summary="false"
required="true" readOnly="false" name="resourceTypeName" />
            <c:simple-property type="string" summary="false"
required="true" readOnly="false" name="resourceTypePlugin" />
            <c:simple-property type="boolean" summary="false"
required="true" readOnly="false" name="updateSchedules" />
        </c:map-property>
        <ci:values>
           <ci:map-value>
               <ci:simple-value name="metricName"
value="MCBean|ServerInfo|*|freeMemory"/>
               <ci:simple-value name="resourceTypeName" value="JBoss AS
Server"/>
               <ci:simple-value name="resourceTypePlugin"
value="JBossAS5"/>
               <ci:simple-value name="updateSchedules" value="true"/>
           </ci:map-value>
        </ci:values>
    </ci:list-property>
</default-configuration>
```

To update all metrics schedules, set the **`<ci:simple-property>`** element to **`name="updateAllSchedules"`**.

To update a single metric schedule, then set the property element's name to **`metricUpdateOverride`** and set the **`updateSchedules`** property value to **`true`**.

5.4.2.2.2. Changing the Synchronizer Settings Programmatically

To change the configuration, create a new instance of the default and use the **`setValue`** configuration object to add or remove keys from the list. For the settings synchronizer, this lists the key name to import:

```
configurationObject.getSimple('propertiesToImport').setValue(defaultSettin
gsToImport + ', keyName')
```

For metrics schedules, it lists the metric schedule per resource type, based on a properties list or a properties map:

```
var update = new PropertyMap('metricUpdateOverrides')
update.put(new PropertySimple('propertyName', 'resourcePluginName'))
```

1. Get the default definition.

   ```
   rhqadmin@localhost:7080$ var
   systemSettingsImportConfigurationDefinition =
   SynchronizationManager.getImportConfigurationDefinition('org.rhq.ent
   erprise.server.sync.SystemSettingsSynchronizer')
   ```

2. Create a new configuration instance.

   ```
   rhqadmin@localhost:7080$ var configurationObject =
   systemSettingsImportConfigurationDefinition.configurationDefinition.
   defaultTemplate.createConfiguration()

   rhqadmin@localhost:7080$ var systemSettingsImportConfiguration = new
   ImportConfiguration(systemSettingsImportConfigurationDefinition.sync
   hronizerClassName, configurationObject)
   ```

3. Change the settings in the new instance.

   For example, for the server settings synchronizer:

   ```
   rhqadmin@localhost:7080$ var defaultSettingsToImport =
   configurationObject.getSimple('propertiesToImport').stringValue

   rhqadmin@localhost:7080$
   configurationObject.getSimple('propertiesToImport').setValue(default
   SettingsToImport + ', CAM_BASE_URL')
   ```

   For the metrics template synchronizer:

   ```
   configurationObject.getSimple('updateAllSchedules').setBooleanValue(
   true)
   var updateList = new PropertyList('metricUpdateOverrides')
   var update = new PropertyMap('metricUpdateOverride')
   update.put(new PropertySimple('metricName',
   'MCBean|ServerInfo|*|freeMemory'))
   update.put(new PropertySimple('resourceTypeName', 'JBossAS Server'))
   update.put(new PropertySimple('resourceTypePlugin', 'JBossAS5'))
   update.put(new PropertySimple('updateSchedules', 'true'))
   ```

```
updateList.add(update)

configurationObject.put(updateList)
```

5.4.2.3. Importing the Configuration

1. Log into the JBoss ON CLI.

   ```
   [root@server bin]#  installDir/bin/rhq-cli.sh -u rhqadmin -p
   rhqadmin
   ```

2. Import the XML file containing the configuration:

   ```
   rhqadmin@localhost:7080$ var data = getFileBytes('export.xml.gz');
   rhqadmin@localhost:7080$
   SynchronizationManager.importAllSubsystems(ex, null);
   ```

   The *null* parameter means that the import process uses the default settings in the XML file or, if the defaults are missing from the XML, that it uses the settings defined on the target server. If alternate settings were constructed in Section 5.4.2.2, “Changing the Synchronizer Configuration”, then they can be specified programmatically instead. For example:

   ```
   rhqadmin@localhost:7080$ var configsToImport = new
   java.util.ArrayList()
   rhqadmin@localhost:7080$
   configsToImport.add(systemSettingsImportConfiguration);
   rhqadmin@localhost:7080$
   configsToImport.add(metricTemplatesImportConfiguration);
   rhqadmin@localhost:7080$
   SynchronizationManager.importAllSubsystems(ex, configToImport);
   ```

6. CONFIGURING AGENTS

The agent can be configured and managed through the agent prompt, which is opened through the **rhq-agent.sh** script.

6.1. Registering and Re-registering the Agent

When an agent registers with the JBoss ON server, the agent name is used as a unique resource key to identify the agent. In addition, the server generates a random string which it sends to the agent to use as a registration token or *security token*.

6.1.1. About the Security Token and Agent Registration

When the JBoss ON agent starts up, it registers with the JBoss ON server and sends the server its information. The JBoss ON server creates an entry based on the given agent name, IP address, and port number.

The JBoss ON server also creates a randomly-generated string, a *security token*, which is also associated with the agent name and with the IP address and port number pair.

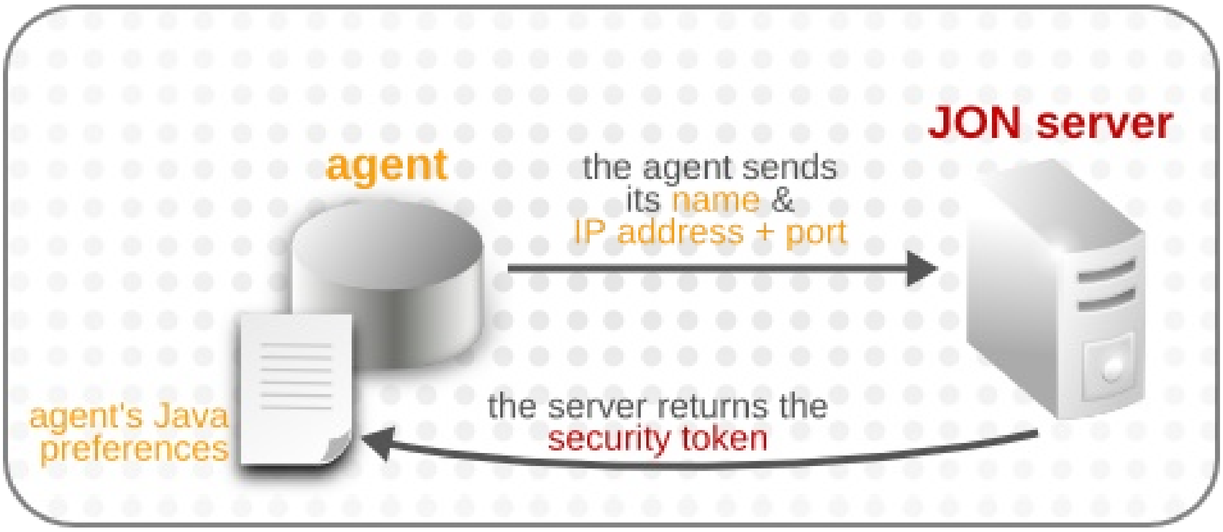


Figure 4. Agent Registration

The agent sends its security token to the server when it restarts as a form of pseudo-authentication. The JBoss ON server uses the unique resource key (the agent's name) and its security token as a way to verify the agent identity.

The JBoss ON server associates the agent name and its security token every time the agent starts up and registers with the server. If the agent-supplied information does not match the information that the JBoss ON server has for that agent, then it rejects the agent's connection attempt.

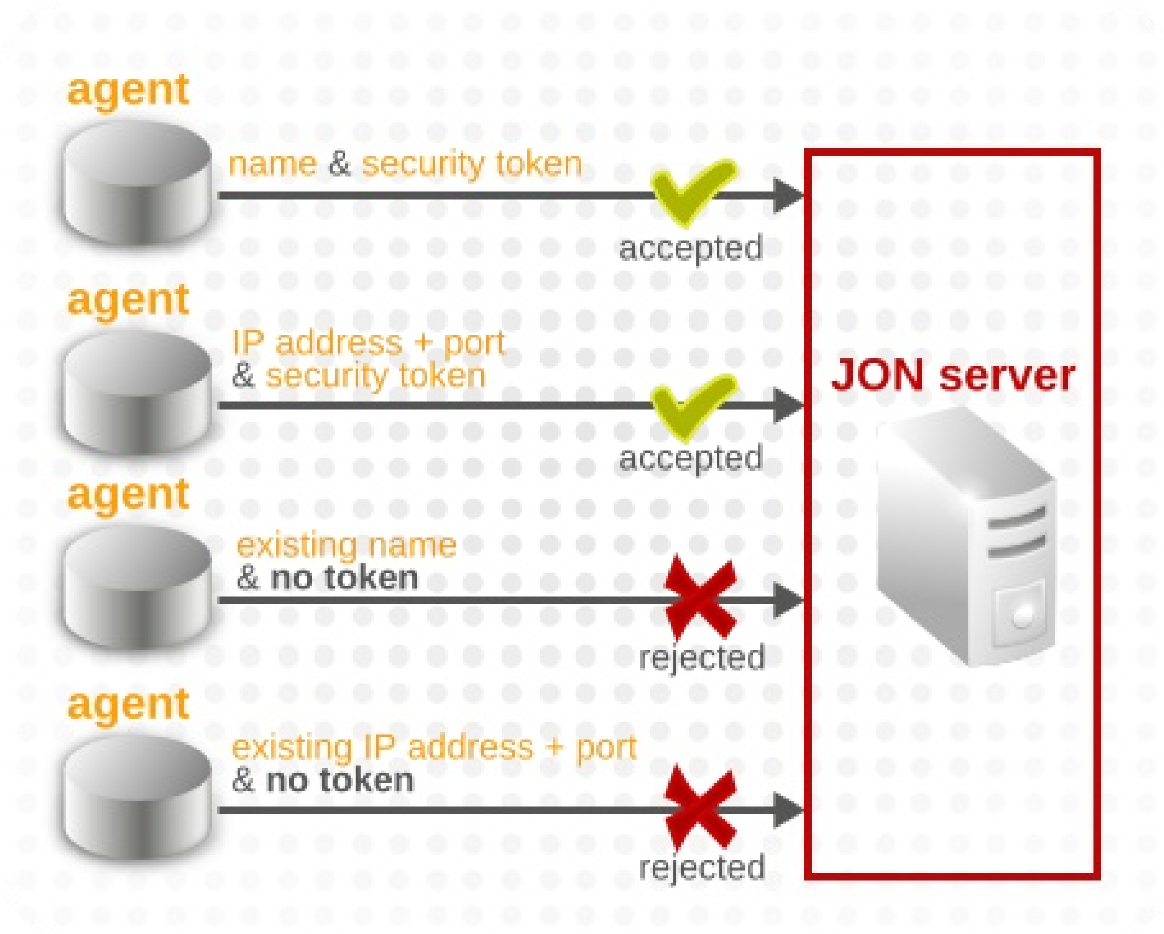


Figure 5. Different Agent Connection Attempts

That means that there are a few rules about when the JBoss ON server will accept changes to the agent's registration information:

- An agent cannot register with an existing agent name *without* the corresponding security token.

  To register an agent with an existing agent name, you must first install the corresponding security token, as described in Section 6.1.2, “Re-installing a Lost Security Token”.

- An agent cannot register with an existing IP address/port combination *without* having the corresponding security token *and* using the original agent name.

  This essentially means that you cannot rename an agent. If an agent is registered with an existing IP address/port combination, then both the original security token and the original name must also be used. This re-establishes the original identity of the agent and prevents one agent from effectively stealing the identity of another agent.

- An agent *can* register with an existing name and a new IP address/port combination if it has the security token which corresponds to that agent name.

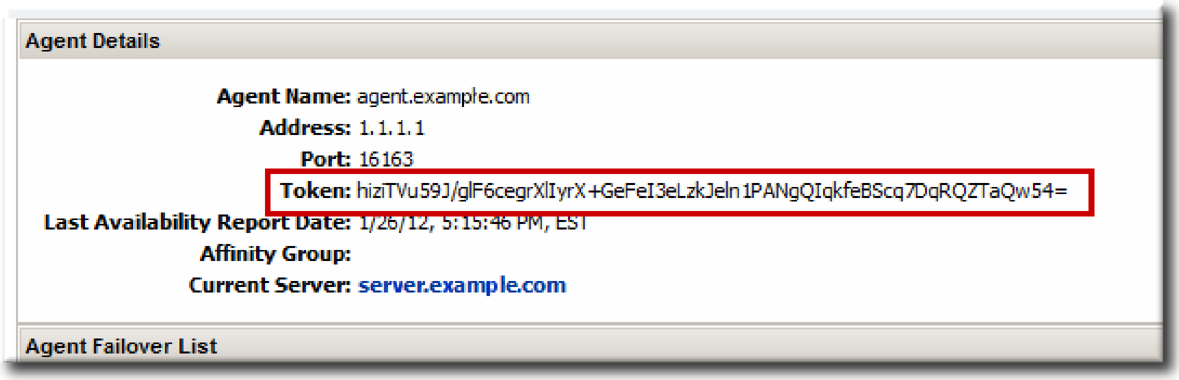
  While the agent name cannot be changed during re-registration, the agent IP address, the agent port, or both can be changed. This is a common and useful scenario in cloud, virtual, or DHCP environments where an existing agent needs to re-register with a new IP address or port.

NOTE

The security token is stored in the agent's Java preferences. This security token persists even if the agent is restarted, is uninstalled, or has its configuration wiped with **`--cleanconfig`**. This allows the agent to re-register easily.

6.1.2. Re-installing a Lost Security Token

1. Stop the agent.
2. Log into the web UI as a user with manage security permissions.
3. Click the **`Administration`** tab and select the **`Agents`** link under the **`Topology`** section on the left.
4. Select the agent from the list, and click its name to open its details page.



Agent Details

Agent Name: agent.example.com
Address: 1.1.1.1
Port: 16163
Token: hiziTVu59J/glF6cegrXlIyrX+GeFeI3eLzkJeln1PANgQIqkfeBScq7DqRQZTaQw54=
Last Availability Report Date: 1/26/12, 5:15:46 PM, EST
Affinity Group:
Current Server: server.example.com

Agent Failover List

5. Copy the security token.

6. Restart the agent, and use the **-D** option to set the ***rhq.agent.security-token*** property to the security token.

   ```
   agentRoot/rhq-agent/bin/rhq-agent.sh -Drhq.agent.security-
   token=abcd1234
   ```

6.2. Working with the Agent Command Prompt

When the agent is started in a terminal, then (along with starting the agent process) the script starts the agent command prompt. The agent prompt can be used to managed the agent by checking configuration, executing some tasks, or editing the agent setup.

6.2.1. Opening the Agent Command Prompt

The agent command prompt opens when the agent start script is run. If the agent is already running, then the command prompt can be opened by itself, without attempting to start the agent, by using the **--nostart** option.

```
$ rhq-agent.sh --nostart
```

6.2.2. Agent Start Options

Some agent management can be performed by passing options with the **rhq-agent.sh** start script; these mainly relate to passing persistent configuration options to the server by loading external preferences through input files or passed parameters. These options are listed in Table 13, “Options for the rhq-agent.sh Script”.

Table 13. Options for the rhq-agent.sh Script

Short Argument	Long Argument	Description
-a	--advanced	Runs the agent script in setup mode, rather than basic start mode.
-c	--config=*filename*	Specifies an agent configuration preferences file on filesystem or classpath.
-d	--daemon	Runs the agent in daemon mode, which means it will not read additional commands from stdin.
-D*name*[=*value*]		Overrides an agent configuration preference and sets a system property.

Short Argument	Long Argument	Description
-e	--console=*type*	Specifies the implementation to use when reading console input. The three available values are jline, sigar, and java.
-h	--help	Opens the help message.
-i	--input=*filename*	Specifies a script file to use for input.
-l	--cleanconfig	Clears out any existing configuration and data files so the agent starts with blank configuration, *with the exception* of the agent security token, which is preserved.
-L	--fullcleanconfig	Clears out any existing configuration and data files so the agent starts with a totally clean slate, including purging the security token.
-n	--nostart	Runs the agent script without starting the agent process.
-o	--output=*filename*	Specifies a file to write all output from the script, excluding log messages (which are always written to the agent logs).
-p	--pref=*preferences_name*	Specifies the agent preferences name used to identify what configuration to use.
-s	--setup	Forces the agent to ask setup questions.
-t	--nonative	Forces the agent to disable the native system.
-u	--purgedata	Purges persistent inventory and other data files.

Short Argument	Long Argument	Description
--		Stops the agent from processing options.

6.2.3. Agent Prompt Commands

The agent processes prompt commands that are passed to it, either interactively through the agent prompt or from an input file that can be passed when the start script is launched. Agent prompt commands (listed in Table 14, “Agent Prompt Commands”) can be used to manage resource (by checking availability, running discovery, or checking monitoring information) or to manage the agent itself (such as registering with a server, loading plug-ins, or viewing or reloading configuration settings).

Table 14. Agent Prompt Commands

Prompt Command	Description
avail	Provides availability of inventoried resources.
config	Manages the agent configuration.
debug	Provides features to help debug the agent.
discovery	Asks a plug-in to run a server scan discovery.
download	Downloads a file from the JBoss ON server.
dumpspool	Shows the entries found in the command spool file.
exit	Shuts down the agent's communications services and kills the agent.
failover	Shows or updates the high availability server failover list.
gc	Helps free up memory by invoking the garbage collector.
getconfig	Displays one, several or all agent configuration preferences.
help	Shows help for a given command.
identify	Asks to identify a remote server.
inventory	Provides information about the current inventory of resources.

Prompt Command	Description
log	Configures some settings for the log messages.
metrics	Shows the agent metrics.
native	Accesses native system information.
pc	Starts and stops the plug-in container and all deployed plug-ins.
ping	Pings the JBoss ON server.
piql	Executes a PIQL query to search for running processes.
plugins	Updates the agent plug-ins with the latest versions from the server.
quit	Exits the agent prompt (without stopping the agent).
register	Registers this agent with the JBoss ON server.
schedules	Retrieves measurement schedule information for the specified resource.
sender	Controls the command sender to start or stop sending commands.
setconfig	Sets an agent configuration preference.
setup	Sets up the agent configuration by asking a series of questions.
shutdown	Shuts down all communications services without killing the agent.
sleep	Puts the agent prompt to sleep for a given amount of seconds.
start	Starts the agent comm services so it can accept remote requests.
timer	Times how long it takes to execute another prompt command.
update	Provides agent update functionality.
version	Shows the agent version information.

6.3. Running the Agent as a Non-Root User

To access some resource information, the agent must have root access to the resource itself. However, for security, many administrators do not want to run the agent process as root.

On Red Hat Enterprise Linux, it is possible to grant access to the agent to specific resources while running the agent as a non-root user. This is done by setting local *access control rules* to the local directories or files for the resource.

> **NOTE**
>
> This example sets ACLs for a PostgreSQL database; the directories and files to specify in the **setfacl** command will vary depending on the resource type.

1. Log into the system as root.

2. Make sure that the acl package is installed on the system.

   ```
   # rpm -q acl
   acl-2.2.39-6.el5
   ```

   The **acl** option must be applied to the filesystem. This can be done by editing the **/etc/fstab** file or using **tune2fs**. For example:

   ```
   # vim /etc/fstab

   LABEL=/           /            ext3    defaults,acl    1 1
   ...
   ```

   Then re-mount the filesystem.

   ```
   # mount -o remount /
   ```

3. Optionally, create a system user to use for the agent.

   ```
   useradd jonagent
   ```

4. For PostgreSQL, the agent needs to be able to access the **postgresql.conf** file. Open the PostgreSQL directory:

   ```
   # cd /var/lib/pgsql
   ```

5. Grant read and write access to the **postgresql.conf** file to the agent user. For example:

   ```
   # setfacl -m u:jonagent:rw $PGDATA/postgresql.conf
   ```

6. Then, grant access to the **data/** directory to the agent user. For example:

   ```
   # setfacl -m u:jonagent:x $PGDATA
   ```

7. Check that the new ACLs were added properly using the **getfacl** command:

   ```
   # getfacl .
   # file: .
   # owner: postgres
   # group: postgres
   user::rwx
   user:jonagent:--x
   group::---
   mask::--x
   other::---
   ```

6.4. Enabling Debug Mode for the Agent

The JBoss ON agent, like the JBoss ON server, uses **log4j** for its logging. To troubleshoot agent performance or server-agent communication, enable debug logging for the agent, which enables the **log4j** debug log.

The log files are in the ***agentRoot*/rhq-agent/logs** directory.

6.4.1. Using an Environment Variable

The quickest way to enable debug logging is to set the **RHQ_AGENT_DEBUG** environment variable to any value before starting the agent. When you start the agent, both the launcher scripts and the agent itself will output debug messages.

If the JBoss ON agent is running on Microsoft Windows using the service wrapper, set **RHQ_AGENT_DEBUG** and then install the service:

```
rhq-agent-wrapper.bat install
```

6.4.2. Setting log4j Priorities

log4j categories support *priorities* for logging levels. This means that different areas of the agent can be configured for different log levels.

NOTE

Do not set the **RHQ_AGENT_DEBUG** environment variable if you are setting priorities in the **log4j.xml** file. The environment variable overrides this **log4j.xml** configuration.

To enable debug logging for a category, change the priority value to **DEBUG**:

1. Open the agent **log4j** file:

   ```
   # vim agentRoot/rhq-agent/conf/log4j.xml
   ```

2. Reset the ***priority*** element for the category. By default, the agent configuration has logging for both incoming and outgoing server-agent communication and for the base **org.rhq** class. Optionally, logging can be enabled for plug-in class loaders and JBoss remoting communication.

```
   <!-- ================ -->
   <!-- Limit categories -->
   <!-- ================ -->

   <!-- RHQ -->
   <category name="org.rhq">
      <priority value="INFO"/>
   </category>

   <!-- RHQ outgoing command tracing  - set to TRACE to trace 
commands sent by the agent -->
   <category 
name="org.rhq.enterprise.communications.command.client.OutgoingComma
ndTrace">
      <priority value="NONE"/>
      <appender-ref ref="COMMANDTRACE"/>
   </category>
   ...
```

3. Restart the agent to load the new configuration.

The **log4j** file format is described more in theApache log4j documentation.

6.4.3. Using the Agent debug Prompt Command

Debug logging can be enabled using the **debug** command in the agent command prompt (Section 6.2.1, “Opening the Agent Command Prompt”).

Using the **--enable** option enables the **log4j** debug log.

```
> debug --enable
log4j:WARN No appenders could be found for logger
(org.rhq.core.pc.measurement.MeasurementCollectorRunner).
log4j:WARN Please initialize the log4j system properly.
Switched to log file [log4j-debug.xml]. Root log level is [DEBUG]
started>
```

To enable debug logging specifically for server-agent communication layers, set the **--comm** option to true.

```
> debug --comm=true
Agent-server communications tracing has been enabled.
You may set the following, additional configuration settings
to collect more detailed trace data. You can set these
using the setconfig prompt command. Please refer to the
documentation for more information on these settings. The
values you see here are the current settings:
   rhq.trace-command-config=true
   rhq.trace-command-response-results=256
   rhq.trace-command-size-threshold=99999
   rhq.trace-command-response-size-threshold=99999
```

The **debug** command can also be used to check all of the agent threads, to the server and to the system management handlers, using the **--threaddump** option. This prints the information for each thread, whether the thread is running or any errors that the agent is encountering, per thread. For example:

```
> debug --threaddump
"DestroyJavaVM" Id=47 RUNNABLE


"RHQ Agent Prompt Input Thread" Id=46 RUNNABLE


"EventManager.sender-2" Id=49 TIMED_WAITING on
java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject@17d7
c01
        at sun.misc.Unsafe.park(Native Method)
        -  waiting on
java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject@17d7
c01
        at
java.util.concurrent.locks.LockSupport.parkNanos(LockSupport.java:226)
        at
java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.awai
tNanos(AbstractQueuedSynchronizer.java:2081)
        at java.util.concurrent.DelayQueue.take(DelayQueue.java:193)
        at
java.util.concurrent.ScheduledThreadPoolExecutor$DelayedWorkQueue.take(Sch
eduledThreadPoolExecutor.java:688)
        at
java.util.concurrent.ScheduledThreadPoolExecutor$DelayedWorkQueue.take(Sch
eduledThreadPoolExecutor.java:681)
        at
java.util.concurrent.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:10
43)
        at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:
1103)
 ...
```

6.5. Changing the Agent IP Address

The agent IP address is set in the ***rhq.communications.connector.bind-address*** configuration preference. This is the IP address the agent binds to when it starts its server socket, meaning this is the site that the agent uses to listen for incoming messages from the server.

NOTE

Do not attempt to edit the **agent-configuration.xml** file. The agent does not use this file once the initial setup is complete, so any changes to this file aren't loaded automatically by the agent.

1. Open the agent prompt. For example, if the agent process is already running, the prompt can be opened by re-running the **rhq-agent.sh** script with the **-n** option.

   ```
   agentRoot/rhq-agent/bin/rhq-agent.sh -n
   ```

2. Send the **`setconfig`** with the ***`rhq.communications.connector.bind-address`*** configuration preference and new value.

   ```
   > setconfig rhq.communications.connector.bind-address=1.2.3.4
   ```

3. Restart the agent process to load the new configuration.

   ```
   agentRoot/rhq-agent/bin/rhq-agent-wrapper.sh stop

   agentRoot/rhq-agent/bin/rhq-agent.sh
   ```

6.6. Managing the Agent as a Resource

The agent can be added as a resource to the JBoss ON inventory, so its behavior and metrics can be monitored to ensure that it is working properly and it can have alerts and operations launched, as with any other resource.

IMPORTANT

If the agent is shut down, the JBoss ON GUI cannot be used to restart it because there is no active agent available to issue the start command. To restart the agent, use the restart operation on the agent's child resource of the launcher script, rather than the agent resource itself.

The shutdown operation kills the agent process if it is running as a daemon. If the agent is running as a command prompt, the shutdown operation stops the agent but not the JVM, so that prompt commands can still be run through the agent command prompt.

When the agent is imported into the inventory, several child resources are automatically added as well. These are listed in Table 15, “Agent Child Resources”.

Table 15. Agent Child Resources

Child Resource	Description

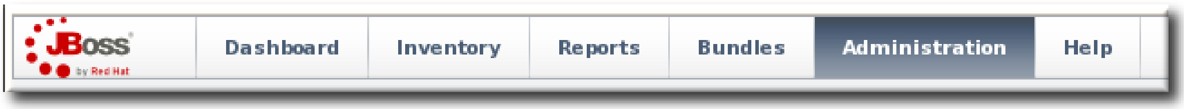
Child Resource	Description
The agent itself	Provides monitoring, configuration, and control functionality for the agent and its internal components. These configuration settings correspond to the preferences defined in the **`agent-configuration.xml`** file and are persisted on the agent machine as Java preferences. **IMPORTANT** The operations for the agent resource normally do not affect the agent process directory. These do not provide control over the JVM settings or process or the JRE options. Controlling the JVM is done through the agent child resources, not the agent resource.
Agent measurement subsystem	Provides data on the measurement collection and reporting components in the agent.
Agent JVM	Provides fine-grained monitoring and management of the JVM that is running the agent and all its plugins, which includes the classloader, threading and memory management subsystems, among others. This is a child server.
Agent environment setup script	Configured environment variables that server set when the agent launcher script is started.
Agent plug-in container	Provides a view into the embedded plug-in container and gives management data related directly to the plug-in container. The plug-in container runs within the agent and handles the deployment of all management plug-ins and infrastructure necessary to run those plug-ins.

Child Resource	Description
Java service wrapper launcher (Windows)	Controls the Java service wrapper. This is a third-party library that installs and runs the agent as a Windows service. There is one primary configuration file for the Java service wrapper, the read-only **`rhq-agent-wrapper.conf`** file. This defines the base set of configuration settings necessary for the agent to start and operate properly. Two additional groups of configuration settings can customize the agent's environment. The Environment group defines environment variables that are used by the main configuration in addition to the environment variables defined by the common Environment Setup Script. The Includes group defines any of the wrapper configuration settings. These groups should almost never be edited except to configure debugging or to pass new JVM options to the agent JVM.
Agent launcher script (UNIX)	Controls the agent. If the agent is running as a background daemon process that was spawned by the launcher script, the launcher script stops or restarts it. There is no additional configuration. The launcher script is configured by the Environment Setup Script.

6.7. Configuring Agent Update Settings

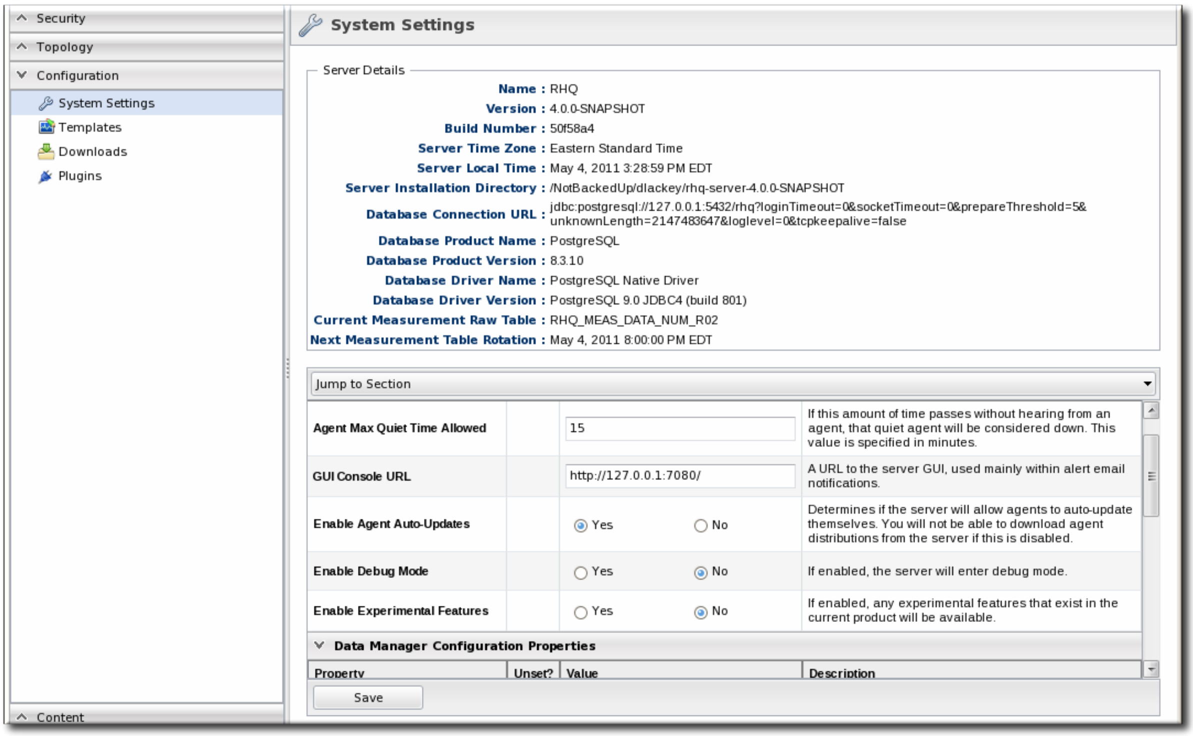
The JBoss ON server configuration defines two settings for how the server sends updates to the JBoss ON agents. One setting sets how long the server waits to hear from the agent before it considers that the agent is down. The second enables automatic updates for agent configuration.

1. Click the **`Administration`** tab in the top menu.



2. In the **Configuration** menu table on the left, select the**System Settings** item.

3. Scroll to the **`JON General Configuration Properties`** section in the main work area.



System Settings

Server Details

Name :	RHQ
Version :	4.0.0-SNAPSHOT
Build Number :	50f58a4
Server Time Zone :	Eastern Standard Time
Server Local Time :	May 4, 2011 3:28:59 PM EDT
Server Installation Directory :	/NotBackedUp/dlackey/rhq-server-4.0.0-SNAPSHOT
Database Connection URL :	jdbc:postgresql://127.0.0.1:5432/rhq?loginTimeout=0&socketTimeout=0&prepareThreshold=5&unknownLength=2147483647&loglevel=0&tcpkeepalive=false
Database Product Name :	PostgreSQL
Database Product Version :	8.3.10
Database Driver Name :	PostgreSQL Native Driver
Database Driver Version :	PostgreSQL 9.0 JDBC4 (build 801)
Current Measurement Raw Table :	RHQ_MEAS_DATA_NUM_R02
Next Measurement Table Rotation :	May 4, 2011 8:00:00 PM EDT

Jump to Section

Agent Max Quiet Time Allowed		15	If this amount of time passes without hearing from an agent, that quiet agent will be considered down. This value is specified in minutes.
GUI Console URL		http://127.0.0.1:7080/	A URL to the server GUI, used mainly within alert email notifications.
Enable Agent Auto-Updates		● Yes ○ No	Determines if the server will allow agents to auto-update themselves. You will not be able to download agent distributions from the server if this is disabled.
Enable Debug Mode		○ Yes ● No	If enabled, the server will enter debug mode.
Enable Experimental Features		○ Yes ● No	If enabled, any experimental features that exist in the current product will be available.

Data Manager Configuration Properties

Property	Unset?	Value	Description

4. Change the agent settings:

   - **`Agent Max Quiet Time Allowed`** sets the longest interval that the server waits for agent availability reports before marking the agent as down.

   - **`Enable Agent Auto-Updates`** sets whether the agent is allowed to check, itself, for updates to configuration or new agent JAR files.

6.8. Managing the Agent's Persisted Configuration

The agent uses Java preferences in the Java platform to store its configuration. Java preferences in general are described in the Java documentation at http://download.oracle.com/javase/1.5.0/docs/guide/preferences/index.html. JBoss ON stores user preferences in the backing store's root node.

The location of the backing store depends on the system:

- On Windows, the backing store is located in the Windows registry.

- On Linux and Unix systems, the backing store is in the agent user's home directory, in **`~/.java`**.

IMPORTANT

The agent's configuration is determined by what user is running the agent. If the agent is run as one user and then later run as another user, the agent will have a different configuration that second time because it will use a different backing store for its configuration settings.

For example, if the agent is configured by a system user named **`jsmith`**, its persisted configuration is in**`~jsmith/.java`**. If the agent is then configured to run as a background service as the root user, the agent looks for its configuration in **`~root/.java`**, and it finds different configuration settings.

This means that if one user is used to configure the agent when it is installed, that same user must be used to run the agent subsequently, or the agent will apparently lose its configuration and need to be reconfigured under the new user.

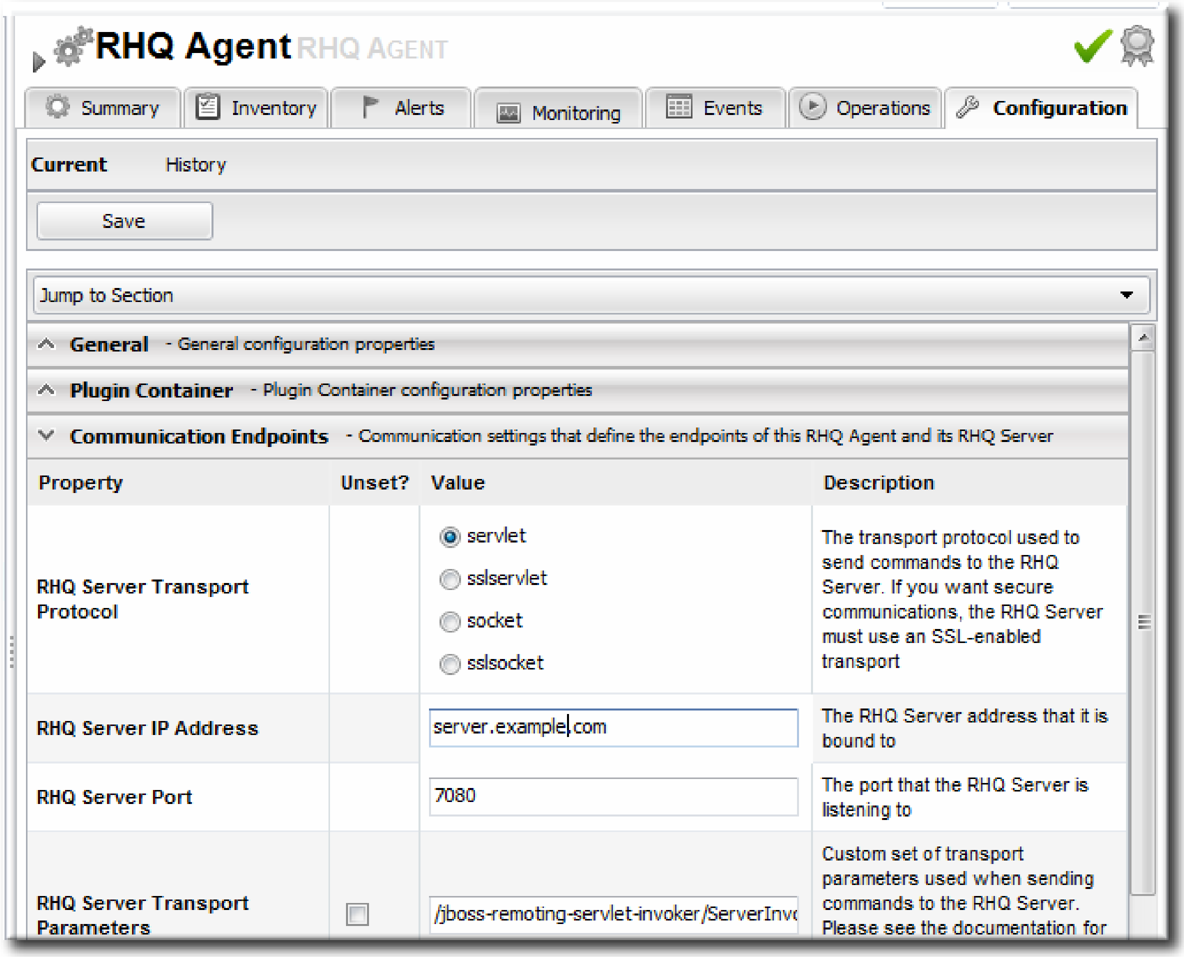
The agent gets the configuration that it uses to run from its backing store. It only reads configuration settings from the **`agent-configuration.xml`** file when the agent needs to initialize its backing store, either at its first configuration or if the agent was started with **`--cleanconfig`** and fresh configuration settings should be loaded.

6.8.1. Viewing the Persisted Configuration

Agent configuration is comprised of Java *preferences*, which are persisted for each JBoss ON user. The way that the configuration is persisted depends on the operating system; Windows stores the configuration in the registry, for example, while Unix keeps it in the user's home directory.

The agent configuration is loaded when it is first set up and then persisted in the database, with the exception of a few parameters which can be set and loaded through the **`rhq-agent-env.sh`** file. The agent's persisted configuration can be viewed in several different ways:

1. If the agent is in the JBoss ON inventory, then its complete configuration settings are visible through the **`Configuration`** tab, with collapsible tables that display each configuration area.



General - General configuration properties

Plugin Container - Plugin Container configuration properties

Communication Endpoints - Communication settings that define the endpoints of this RHQ Agent and its RHQ Server

Property	Unset?	Value	Description
RHQ Server Transport Protocol		● servlet ○ sslservlet ○ socket ○ sslsocket	The transport protocol used to send commands to the RHQ Server. If you want secure communications, the RHQ Server must use an SSL-enabled transport
RHQ Server IP Address		server.example.com	The RHQ Server address that it is bound to
RHQ Server Port		7080	The port that the RHQ Server is listening to
RHQ Server Transport Parameters	□	/jboss-remoting-servlet-invoker/ServerInvc	Custom set of transport parameters used when sending commands to the RHQ Server. Please see the documentation for

2. The configuration can also be returned through the **getconfig** or **config** prompt commands for the agent. These commands can be run through a terminal, if the agent is running through a command prompt, or through the **Execute Command Prompt** operation in the JBoss ON UI for the agent resource.

```
> getconfig
rhq.agent.agent-update.enabled=true
rhq.agent.client.command-preprocessors=org.rhq.enterprise.agent.
SecurityTokenCommandPreprocessor: org.rhq.enterprise.agent.
ExternalizableStrategyCommandPreprocessor
rhq.agent.client.command-spool-file.compressed=true
rhq.agent.client.command-spool-file.name=command-spool.dat
rhq.agent.client.command-spool-file.params=10000000:75
rhq.agent.client.command-timeout-msecs=600000
rhq.agent.client.max-concurrent=5
rhq.agent.client.max-retries=10
rhq.agent.client.queue-size=50000
rhq.agent.client.queue-throttling=200:2000
rhq.agent.client.retry-interval-msecs=15000
rhq.agent.client.send-throttling=100:1000
rhq.agent.client.server-polling-interval-msecs=60000
rhq.agent.configuration-schema-version=5
rhq.agent.configuration-setup-flag=true
rhq.agent.data-directory=data
rhq.agent.disable-native-system=false
rhq.agent.name=localhost.localdomain
rhq.agent.plugins.directory=plugins
...
```

3. The agent configuration is persisted in Java preferences, so any tool which examines Java preferences can be used to view the persisted configuration.

> **WARNING**
>
> Do not attempt to change the values of the preferences using third-party tools. Setting an agent preference to a bad value can completely disable the agent.

6.8.2. Changing Preferences in the Persisted Configuration (Agent Preferences)

The agent's configuration is initially read from **agent-configuration.xml** and overlaid with the values entered at the setup prompts at start up. After the agent is initially configured, the agent persists that configuration and never refers to the **agent-configuration.xml** again, unless the configuration is purged and reloaded. Most configuration changes are made to the **rhq-agent-env.sh** file, which is loaded every time the agent starts.

It is possible to change the persisted configuration (without editing the configuration files) using the **setconfig** command at the agent prompt.

1. Open the agent prompt. For example, if the agent process is already running, the prompt can be opened by re-running the **rhq-agent.sh** script with the **-n** option.

   ```
   agentRoot/rhq-agent/bin/rhq-agent.sh -n
   ```

2. Send the **setconfig** with the name of the preference to edit and its new value. The preference name is whatever the entry name is in the **agent-configuration.xml** file. For example:

   ```
   > setconfig rhq.agent.client.max-concurrent=20
   ```

3. Restart the agent process to load the new configuration.

   ```
   agentRoot/rhq-agent/bin/rhq-agent-wrapper.sh stop

   agentRoot/rhq-agent/bin/rhq-agent.sh
   ```

6.8.3. Overriding Persisted Configuration Settings

The settings in the Java backing store and in the **agent-configuration.xml** file for the agent can be overridden using the **-D** option, the configuration parameter name, and the new value when the agent is started.

For example, to set a temporary value for how long the agent waits at startup to detect the JBoss ON server (**rhq.agent.wait-for-server-at-startup-msecs**), pass this argument with the start command:

```
agentRoot/rhq-agent/bin/rhq-agent.sh -Drhq.agent.wait-for-server-at-
startup-msecs=90000
```

6.9. Managing the Agent JVM

6.9.1. Setting Options for the Agent JVM

The agent runs in a Java Virtual Machine, and aspects of its behavior can be defined in the **rhq-agent-env.sh** file and passed to the JVM.

There are two arguments that set JVM options:

- **RHQ_AGENT_JAVA_OPTS** resets the any of the default JVM settings.
- **RHQ_AGENT_ADDITIONAL_JAVA_OPTS** adds JVM settings without changing any of the default settings.

For more information on JVM settings, see http://java.sun.com/javase/technologies/hotspot/vmoptions.jsp and other Sun JVM documentation.

NOTE

Restart the agent after making changes to the JVM settings to load the new settings.

6.9.2. Setting the Agent JVM Memory Size

When an agent manages a large number of resources, it can begin running out of memory with the default settings of its JVM. This can cause errors like *memory has crossed the threshold and is low* to be recorded in the agent log, and the agent is automatically rebooted. This is usually caused by the agent's heap size begin set too low, but it can also be related to a low perm gen size.

To change the agent's memory settings, use the **RHQ_AGENT_JAVA_OPTS** in the **rhq-agent-env.sh** file to set the appropriate JVM settings.

1. Stop the agent.
2. Open the **rhq-agent-env.sh** file.

   ```
   vim agentRoot/rhq-agent/bin/rhq-agent-env.sh
   ```

3. Uncomment the **RHQ_AGENT_JAVA_OPTS** line, and set the **-Xms** and **-Xmx** parameters to set the minimum and maximum bounds of the heap size for the agent JVM.

   ```
   RHQ_AGENT_JAVA_OPTS="-Xms1024m -Xmx1024m -XX:PermSize=256M -
   XX:MaxPermSize=256M -Djava.net.preferIPv4Stack=true"
   ```

4. Optionally, use **-XX:PermSize** and **-XX:MaxPermSize** to set the perm gen size.
5. Restart the agent.

```
agentRoot/rhq-agent/bin/rhq-agent.sh
```

6.10. Setting Discovery Scan Intervals

The agent scans a platform routinely to look for new servers or services to add to the discovery queue and, subsequently, to inventory. There are two parameters which set scan intervals:

- The scan interval for servers, set in the ***rhq.agent.plugins.server-discovery.period-secs***. The default is 900 seconds (15 minutes).
- The scan interval for services, set in the ***rhq.agent.plugins.service-discovery.period-secs***. The default is 86400 seconds (24 hours).

These are set in the **agent-configuration.xml** file, so the configuration must be cleanly reloaded before the changes take effect.

1. Stop the agent.
2. Open the **agent-configuration.xml** file.

   ```
   vim agentRoot/rhq-agent/conf/agent-configuration.xml
   ```

3. Uncomment the scan interval entry keys and set the new values.

   ```
   <entry key="rhq.agent.plugins.server-discovery.period-secs"
   value="600"/>

   <entry key="rhq.agent.plugins.service-discovery.period-secs"
   value="1440"/>
   ```

4. Start the agent with the **--cleanconfig** option to reload the configuration from the **agent-configuration.xml** file.

   ```
   agentRoot/rhq-agent/bin/rhq-agent.sh --cleanconfig
   ```

6.11. Viewing the Server Failover Lists for Agents

JBoss ON agents are automatically included in high availability in order to assign them to servers for management. Agent-server preferences are assigned through affinity groups (Section 4.4.2, “Creating Affinity Groups”). The agent high availability settings show its affinity groups, the server currently managing it, and any servers available for failover.

The first server that an agent contacts is defined in its **agent-configuration.xml** file, and that is the server that the agent sends its initial registration request. After registration, the agent joins the high availability cloud, and it sends its updates — monitoring information, resource changes — to any server in the cloud. At registration, the agent gets its first affinity group assignment. If its primary server is different than its registration server, then the agent switches communication over to the primary server.

The high availability server cloud helps define the relationships between servers and agents once the agent is running normally.

The group of servers that an agent sends updates to can be loosely restricted by defining an affinity group. The affinity group creates a list of servers that the agent prefers to access. This list is ordered; the first server entry is the primary server that the agent connects to. If that primary server is unavailable, then the agent cycles through the other servers in the list in order. This allows the agent to connect to defined servers in the high availability cloud gracefully and automatically, without interrupting JBoss ON performance.

If the agent cannot connect to any server in the failover list, then the agent temporarily stops communication and spools its messages. After a period of time, it will run through the failover list again, beginning with its primary server.

An agent always try to ensure that it is connected to its primary server. Once an hour, by default, it checks its connection to verify that the server it is using is its primary server. If it is not, then the agent tries to reconnect to its primary server.

The actual failover list for an agent is generated by the server and edited in the affinity group configuration for the server. Any changes to the affinity group, like new servers or agents, changed server priority, or new group assignments, are sent to the agent hourly when the agent polls the server for configuration changes.

To view the agent's failover list from the agent command prompt:

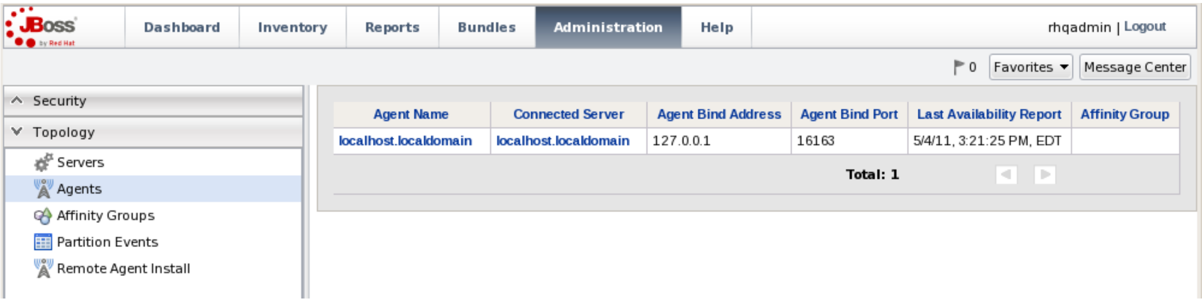
```
> failover --list localhost.localdomain:7080/7443
server2.example.com:7080/7443 1.2.34.56:7080/7443
```

To view the failover list from the UI:

1. Click the **`Administration`** tab in the top menu.

2. In the **High Availability** menu, select the **Agents** item.

3. The agent high availability page shows information about the agents, including three things that are relevant for high availability:

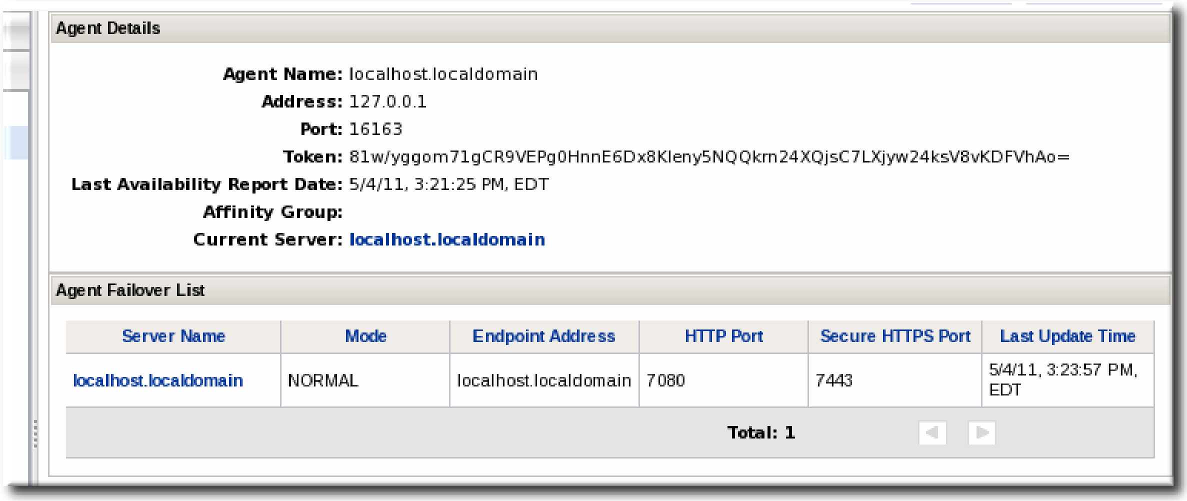


Agent Name	Connected Server	Agent Bind Address	Agent Bind Port	Last Availability Report	Affinity Group
localhost.localdomain	localhost.localdomain	127.0.0.1	16163	5/4/11, 3:21:25 PM, EDT	

Total: 1

   - The JBoss ON server that the agent is currently connected to (or the one it was most recently connected to).
   - The time that the last agent availability report was sent to the server.
   - The affinity group that the agent is assigned to.

4. Click the name of the agent. This opens the agent's server failover list. The first server listed is the primary server for the agent; all other servers are available in the high availability cloud. The connected server is usually also the primary server, unless the primary is offline.



Agent Details

Agent Name: localhost.localdomain
Address: 127.0.0.1
Port: 16163
Token: 81w/yggom71gCR9VEPg0HnnE6Dx8Kleny5NQQkrn24XQjsC7LXjyw24ksV8vKDFVhAo=
Last Availability Report Date: 5/4/11, 3:21:25 PM, EDT
Affinity Group:
Current Server: localhost.localdomain

Agent Failover List

Server Name	Mode	Endpoint Address	HTTP Port	Secure HTTPS Port	Last Update Time
localhost.localdomain	NORMAL	localhost.localdomain	7080	7443	5/4/11, 3:23:57 PM, EDT

Total: 1

6.12. Setting the Agent to Detect or Poll the Server

The agent has to stay in contact with a JBoss ON server. This can either be done by using multicast detection to monitor when its primary JBoss ON server comes online or goes offline or by simply polling the JBoss ON server at intervals to see if the server is online.

These polling methods aren't exclusive; they can both be set, so that the agent can use whatever method is convenient or available to poll the server.

Polling the server allows the agent to stop sending commands and data to the server if the server goes offline and then to resume automatically when the server is back online. If server polling is not enabled on the agent, then the agent always assumes that the server is online and sends its information to the server. If the server goes down, then the agent records repeated *connection refused* errors, which (if the server is down for a long time) can make the agent log grow very large.

6.12.1. Settings for Polling the JBoss ON Server

The simplest configuration is to set a polling interval for the agent. With this method, the agent simply pings the server at the predefined interval.

1. Open the agent prompt. For example, if the agent process is already running, the prompt can be opened by re-running the **rhq-agent.sh** script with the **-n** option.

   ```
   agentRoot/rhq-agent/bin/rhq-agent.sh -n
   ```

2. Send the **setconfig** with the ***rhq.agent.client.server-polling-interval-msecs*** setting and a value (in milliseconds). Setting this value to zero (0) or a negative number disables server polling.

   ```
   > setconfig rhq.agent.client.server-polling-interval-msecs=500
   ```

3. Restart the agent process to load the new configuration.

   ```
   agentRoot/rhq-agent/bin/rhq-agent-wrapper.sh stop
   
   agentRoot/rhq-agent/bin/rhq-agent.sh
   ```

6.12.2. Setting up Multicast Detection

Multicast detection uses JBoss's Remoting framework, which allows the agent to detect whenever a server comes on or goes off line within a few seconds. Using the Remoting framework requires support for multicast traffic; otherwise, the agent cannot detect the server. This has more configuration parameters than simple polling:

- Setting to enable both server detection and multicast traffic (***`rhq.agent.server-auto-detection`*** and ***`rhq.communications.multicast-detector.enabled`***, respectively).

- A wait interval between server communications (***`rhq.communications.multicast-detector.default-time-delay`***); if the server is silent longer than that interval, then the server is considered offline.

- Await, or heartbeat, interval for the agent's own messages (***`rhq.communications.multicast-detector.heartbeat-time-delay`***). This value must be shorter than the JBoss ON server's heartbeat interval (***`rhq.communications.multicast-detector.default-time-delay`***), or it results in unnecessary messages and network traffic.

To enable multicast detection:

1. Open the agent prompt. For example, if the agent process is already running, the prompt can be opened by re-running the **`rhq-agent.sh`** script with the **`-n`** option.

   ```
   agentRoot/rhq-agent/bin/rhq-agent.sh -n
   ```

2. Send the **`setconfig`** with the multicast settings. The time-delay values are in milliseconds.

   ```
   > setconfig rhq.agent.server-auto-detection=true
   > setconfig rhq.communications.multicast-detector.enabled=true
   > setconfig rhq.communications.multicast-detector.default-time-
   delay=75000
   > setconfig rhq.communications.multicast-detector.heartbeat-time-
   delay=60000
   ```

3. Restart the agent process to load the new configuration.

   ```
   agentRoot/rhq-agent/bin/rhq-agent-wrapper.sh stop

   agentRoot/rhq-agent/bin/rhq-agent.sh
   ```

6.13. Throttling the Agent

Some agent settings control how many resources the agent can access and how many tasks it can perform at one time. Throttling the agent has a twofold purpose: it limits how many resources on its host it can monopolize (which can improve performance on the host machine) and it keeps the agent from flooding the server with data and overloading or monopolizing the server.

Several different settings (listed in Table 16, “Agent Parameters for Throttling Agent Operations”) can be used to throttle different aspects of the agent performance. These settings operate independently of each other, but they can be more effective when the settings are made after considering the other values. For example, the queue size should be set larger then the command timeout period, unless the max-concurrent setting is increased. Changing one of these values has a different effect than adjusting all of these values.

Table 16. Agent Parameters for Throttling Agent Operations

Parameter	Description
rhq.agent.client.queue-size	Sets the maximum number of commands the agent can queue up for sending to the JBoss ON server. The larger the number, the more memory the agent can use, and setting this to zero (0) means the queue size is unlimited. Setting this to 0 could allow the agent to queue up more commands than the machine has memory for, if the server goes offline for a long time.
rhq.agent.client.max-concurrent	Sets the maximum number of messages the agent can send to the server at any one time. A larger number allows the agent to process its queue more quickly, but this can also set the agent to use more CPU cycles.
rhq.agent.client.command-timeout-msecs	Sets the amount of time the agent waits for a reply from the JBoss ON server for an agent command before it aborts the command. A long interval can give the server the time it needs to complete some commands, but it also means that other messages are queued up waiting to be processed.
rhq.agent.client.retry-interval-msecs	Sets the time that the agent waits before retrying a command. Only commands with the guaranteed delivery tag are retried.
rhq.agent.client.send-throttling	Sets a limit on the number of commands than an agent can send before it enters a quiet period, when the agent suspends sending commands. This setting only affects commands which can be throttled, which are commands that are sent to the server frequently and in large numbers, such as metric collection. Send-throttling prevents messages storms to the server. This parameter sets both the number of commands and the quiet period, in the form *commands:timeout_milliseconds*. For example, **`50:10000`** sets a limit of 50 commands and a quiet period of 10000 milliseconds.

Parameter	Description
rhq.agent.client.queue-throttling	Limits the amount of commands that can be dequeued in a given amount of time; this is the *burst period*. If more commands are attempted to be dequeued during the burst period than allowed, those dequeue requests are blocked until the next burst period begins. As with send throttling, this parameter sets both the number of commands and the quiet period, in the form *commands:timeout_milliseconds*. For example, **`50:10000`** sets a limit of 50 commands and a quiet period of 10000 milliseconds. Queue throttling prevents the agent from spinning the CPU by trying to process and send commands as fast as possible. Queue throttling is one way to reduce the amount of CPU required by the agent. When setting the queue throttling value, be sure to set the queue size to a large enough value that the agent has room to queue up the additional commands.

6.14. Setting Guaranteed Delivery for Commands

Many commands, like pings between the agent and server, are not critical to JBoss ON functions. These are *volatile commands*. Volatile commands are sent once; if they fail, the failure is logged, the agent drops the command, and the next command is processed.

Critical commands, however, must be sent to the JBoss ON server and successfully processed. The agent must guarantee that these commands are delivered. These are *guaranteed commands*. The agent guarantees, as far as possible, that these commands reach the server (although outside events, such as a JVM crashing, can keep the commands from being sent). Guaranteed commands persist in a *command spool file* even if the agent shuts down, so that the next time the agent starts, it can be loaded and queued to be delivered to the server.

There are four parameters that are related to guaranteed delivery:

- A time interval that sets how frequently the agent should try to resend a failed command (***`rhq.agent.client.retry-interval-msecs`***)
- A filename for the spool file (***`rhq.agent.client.command-spool-file.name`***)
- A setting that configures the spool file (***`rhq.agent.client.command-spool-file.params`***). This settings has the format*max_file_size:purge_percentage*. The file size is defined in bytes; once the file hits that file size, then a purge operation trims the file down to whatever the percentage is. So, if the file is set to be 100 KB (100000) and the purge percentage is 90, then the file is trimmed back to 90 KB after a purge operation. The purge operation first tries to compress unused space, and then begins purging commands, starting with the oldest.
- An optional setting that allows the spool file to be compressed

(***rhq.agent.client.command-spool-file.compressed***). Compressing the spool file can reduce its size 30-40%, but in some corner cases, it can adversely affect agent performance (such as when the agent shuts down before all of the guaranteed commands have been sent).

Guaranteed delivery is configured by default, allowing both the agent to resend critical commands and to compress spool file.

```
rhq.agent.client.command-spool-file.compressed=true
rhq.agent.client.command-spool-file.name=command-spool.dat
rhq.agent.client.command-spool-file.params=10000000:75
rhq.agent.client.retry-interval-msecs=15000
```

To change any of the guaranteed delivery settings:

1. Open the agent prompt. For example, if the agent process is already running, the prompt can be opened by re-running the **rhq-agent.sh** script with the **-n** option.

   ```
   agentRoot/rhq-agent/bin/rhq-agent.sh -n
   ```

2. Send the **setconfig** with the new guaranteed delivery settings.

   ```
   > setconfig rhq.agent.client.command-spool-file.compressed=true
   rhq.agent.client.command-spool-file.name=my-spool.dat
   rhq.agent.client.command-spool-file.params=25000000:67
   rhq.agent.client.retry-interval-msecs=25000
   ```

3. Restart the agent process to load the new configuration.

   ```
   agentRoot/rhq-agent/bin/rhq-agent-wrapper.sh stop

   agentRoot/rhq-agent/bin/rhq-agent.sh
   ```

6.15. Configuring Agent Communication

Both the JBoss ON agent and server use the same underlying communications services. The types of connections used for agent-server communication are defined through agent preferences and can be edited by changing those preferences. The agent uses two settings for communications:

- A parameter which defines the protocol that the agent uses to talk to the server (***rhq.agent.server.transport***) and any additional transport parameters (***rhq.agent.server.transport-params***)

- A parameter which defines the protocol that the agent expects for incoming communications from the server (***rhq.communications.connector.transport***) and then any optional transport parameters (***rhq.communications.connector.transport-params***)

Both JBoss ON servers and agents use communications layers that are build on the JBoss Remoting framework. Agents support four different transport types:

- servlet (only for agent to server communications)

- sslservlet
- socket (only for server to agent communications)
- sslsocket

NOTE

Unlike JBoss ON servers, JBoss ON agents do not host a servlet container. This means that servlets cannot be used for server-to-agent communications; these connections use sockets. Only agent-to-server connections use servlets.

The behavior of connections between agents and servers can be controlled by setting transport parameters. The connections between agents and servers are defined by strings which look, roughly, like URLs, with this basic format:

```
protocol://hostname:port/?param1=value&param2=value
```

For example:

```
socket://server.example.com:16163/?
serverBindAddress=127.0.0.1&serverBindPort=16163&numAcceptThreads=3&maxPoo
lSize=303&clientMaxPoolSize=304&socketTimeout=60000&enableTcpNoDelay=true&
backlog=200
```

Both servers and agents have a ***rhq.communications.connector.transport-params*** configuration settings which allows transport parameters to be set. These parameters are appended to the end of the URL and can configure both server-side and client-side behavior. For example, the **backlog** parameter is used by JBoss ON servers; with this URL, the server sets its backlog value to 200, but this setting is ignored by agents since they are clients. Likewise, the **enableTcpNoDelay** parameter is used by agents when they connect to servers, but is ignored by the servers themselves.

For more information on all available transport parameters, see the JBoss Remoting documentation at http://labs.jboss.com/portal/jbossremoting/docs/guide/ch05.html.

7. MANAGING DATABASES ASSOCIATED WITH JBOSS ON

There are several basic tasks that can be done to manage the Oracle or PostgreSQL databases that are used by the JBoss ON server.

7.1. Running SQL Commands from JBoss ON

SQL commands can be run through the JBoss ON web UI on any database that the JBoss ON server is using for its data.

NOTE

The database management page is not accessible through the normal JBoss ON GUI. Administrators must manually navigate to the admin area of the JBoss ON UI.

NOTE

Whatever JBoss ON user you are logged in as must have adequate user rights *on the database* to execute the SQL commands.

1. Open the administrative page, with the location **admin/test/sql.jsp**. For example:

   ```
   http://server.example.com:7080/admin/test/sql.jsp
   ```

2. Enter the SQL commands, as appropriate for the JBoss ON Oracle or PostgreSQL database instance. If there are multiple commands, make sure the **Continue if statements fail?** checkbox is selected. That way, even if one command fails, the other commands will be submitted. Otherwise, the series will be terminated at the first failure.

3. Click the **Execute SQL** button.

7.2. Changing Database Passwords

The JBoss ON server connects to its database instance as a database user. It does this automatically, using the credentials given in its **rhq-server.properties** file. The database password is encoded automatically by the installer before it is stored in the **rhq-server.properties** file, to provide some extra protection against unauthorized access to the database password.

It's possible that the password for that database user account is changed. This change always occurs at the database, not in JBoss ON, so the password in the **rhq-server.properties** file has to be manually encoded and updated for JBoss ON to continue to function.

1. Change the password for the JBoss ON user (**rhqadmin** by default) in the database.

2. Use the **generate-db-password.sh** script to encode the password.

   ```
   serverRoot/bin/generate-db-password.sh mypassword
   Encoded password: 1d31b70b3650168f79edee9e04977e34
   ```

   JBoss ON stores its database password in an encoded form in the **rhq-server-properties** file. Therefore, the new database has to be properly encoded before it's added to the **rhq-server-properties** file so that the server reads it correctly.

3. Edit the ***rhq.server.database.password*** value in the **rhq-server.properties** file so that it has the new encoded password value.

   ```
   vim serverRoot/bin/rhq-server.properties

   rhq.server.database.password=1d31b70b3650168f79edee9e04977e34
   ```

7.3. Editing the JBoss ON Server's Database Configuration

The JBoss ON server is always connected to a backend database to store most of its data, such as agents and resources in its inventory and plug-in configuration. The parameters for connecting with the database are defined in **rhq-server.properties**.

-

Example 7. Default Configuration for a PostgreSQL Database

```
# Database
rhq.server.database.connection-url=jdbc:postgresql://127.0.0.1:5432/rhq
rhq.server.database.driver-class=org.postgresql.Driver
rhq.server.database.xa-datasource-class=org.postgresql.xa.PGXADataSource
rhq.server.database.user-name=rhqadmin
rhq.server.database.password=1eeb2f255e832171df8592078de921bc
rhq.server.database.type-mapping=PostgreSQL
rhq.server.database.server-name=127.0.0.1
rhq.server.database.port=5432
rhq.server.database.db-name=rhq
hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

Table 17. rhq-server.properties Parameters for Database Configuration

Parameter	Description
rhq.server.database.type-mapping	Gives the type or vendor of the database that is used by the JBoss ON server. This is either PostgreSQL or Oracle.
rhq.server.database.connection-url	The JDBC URL that the JBoss ON server uses when connecting to the database. An example is *jdbc:postgresql://localhost:5432/rhq* or *jdbc:oracle:oci:@localhost:1521:orcl*.
rhq.server.database.driver-class	The fully qualified class name of the JDBC driver that the JBoss ON server uses to communicate with the database. An example is *oracle.jdbc.driver.OracleDriver*.
rhq.server.database.xa-datasource-class	The fully qualified class name of the JDBC driver that the JBoss ON server uses to communicate with the database. Examples of this are *org.postgresql.xa.PGXADataSource* or *oracle.jdbc.xa.client.OracleXADatasource*.
rhq.server.database.user-name	The name of the user that the JBoss ON server uses when logging into the database
rhq.server.database.password	The password of the database user that is used by the JBoss ON server when logging into the database. This password is stored in a hash in the **rhq-server.properties** file. When the password is changed in the database, then the password must be manually hashed and copied into the **rhq-server.properties** file. This is described in Section 7.2, “Changing Database Passwords”.

Parameter	Description
rhq.server.database.server-name	The server name where the database is found. This must match the server in the connection URL. This is currently only used when connecting to PostgreSQL.
rhq.server.database.port	The port on which the database is listening. This must match the port in the connection URL. This is currently only used when connecting to PostgreSQL.
rhq.server.database.db-name	The name of the database. This must match the name found in the connection URL. This is currently only used when connecting to PostgreSQL.
rhq.server.quartz.driverDelegateClass	The Quartz driver used for connections between the server and the database. The value of this is set by the installer and depends on the type of database used to store the JBoss ON information. For PostgreSQL, this is **org.quartz.impl.jdbcjobstore.PostgreSQLDelegate**, and for Oracle, this is **org.quartz.impl.jdbcjobstore.oracle.OracleDelegate**.

8. DOCUMENT INFORMATION

This guide is part of the overall set of guides for users and administrators of JBoss ON. Our goal is clarity, completeness, and ease of use.

8.1. Giving Feedback

If there is any error in this *Basic Admin Guide* or there is any way to improve the documentation, please let us know. Bugs can be filed against the documentation for the community-based RHQ Project in Bugzilla, http://bugzilla.redhat.com/bugzilla. Make the bug report as specific as possible, so we can be more effective in correcting any issues:

1. Select the **Other** products group.

2. Select **RHQ Project** from the list.

3. Set the component to **Documentation**.

4. Set the version number to 3.0.

5. For errors, give the page number (for the PDF) or URL (for the HTML), and give a succinct description of the problem, such as incorrect procedure or typo.

   For enhancements, put in what information needs to be added and why.

6. Give a clear title for the bug. For example, **`"Incorrect command example for setup script options"`** is better than **`"Bad example"`**.

We appreciate receiving any feedback — requests for new sections, corrections, improvements, enhancements, even new ways of delivering the documentation or new styles of docs. You are welcome to contact Red Hat Content Services directly at docs@redhat.com.

8.2. Document History

Revision	Date	Author
Revision 3.0.1-5	**2013-10-31**	**Rüdiger Landmann**
Rebuild with publican 4.0.0		
Revision 3.0.1-1	**April 2, 2012**	**Ella Deon Lackey**
Typo in the security token parameter name.		
Revision 3.0.1-0	**March 18, 2012**	**Ella Deon Lackey**
Updates for JBoss Operations Network 3.0.1.		
Revision 3.0-0	**December 7, 2011**	**Ella Deon Lackey**
Initial release of JBoss ON 3.0.		

INDEX