



Red Hat JBoss Operations Network 3.3

Configuring JON Servers and Agents

For Red Hat JBoss Operations Network 3.3 and its patch releases

Red Hat JBoss Operations Network 3.3 Configuring JON Servers and Agents

For Red Hat JBoss Operations Network 3.3 and its patch releases

Jared Morgan
jmorgan@redhat.com

Zach Rhoads
zach@redhat.com

Ella Deon Ballard
dlackey@redhat.com

Legal Notice

Copyright © 2017 Red Hat.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

Read this guide to learn how to configure the Platform, Agents, and Storage Nodes associated with the product.

Table of Contents

CHAPTER 1. ABOUT JBOSS OPERATIONS NETWORK	4
1.1. ABOUT JBOSS ON AGENTS	4
1.2. ABOUT JBOSS ON SERVERS	4
1.3. ABOUT METRICS STORAGE NODES	5
CHAPTER 2. GENERAL MANAGEMENT	7
2.1. JBOSS ON FILE LOCATIONS	7
2.2. DEFAULT SERVER, AGENT, AND STORAGE NODE PORTS	11
2.3. THE RHQCTL CONTROL SCRIPT	12
2.4. STARTING THE JBOSS ON SERVER	15
2.5. STARTING THE JBOSS ON AGENT	19
2.6. STARTING THE STORAGE NODE	24
CHAPTER 3. TUNING JBOSS ON FOR BETTER PERFORMANCE	25
3.1. INVENTORY BASELINES	25
3.2. MONITORING AND ALERTING BASELINES	27
3.3. TUNING THE AGENT JVM MEMORY SIZE	28
3.4. CHANGING THE AGENT JVM HEALTH CHECK SETTINGS	28
3.5. TUNING THE SERVER JVM	29
3.6. SERVER TUNING FOR LARGE NUMBERS OF AGENTS	30
3.7. DATABASE AND DISK SPACE	32
CHAPTER 4. CONFIGURING SSL CONNECTIONS FOR SERVER-AGENT COMMUNICATION	33
4.1. SETTING UP ENCRYPTION	33
4.2. SETTING UP CLIENT AUTHENTICATION BETWEEN SERVERS AND AGENTS	36
4.3. TROUBLESHOOTING SSL PROBLEMS	42
CHAPTER 5. USING HIGH AVAILABILITY AND AGENT LOAD BALANCING	46
5.1. ABOUT AGENT-SERVER COMMUNICATION AND SERVER AVAILABILITY	46
5.2. CREATING AFFINITY GROUPS	51
5.3. PUTTING SERVERS IN MAINTENANCE MODE	55
5.4. REMOVING SERVERS FROM THE HIGH AVAILABILITY CLOUD	56
5.5. MANAGING PARTITION EVENTS	57
CHAPTER 6. CONFIGURING SERVERS	62
6.1. ENABLING DEBUG LOGGING FOR THE JBOSS ON SERVER	62
6.2. CHANGING THE JBOSS ON SERVER URL	64
6.3. EDITING JBOSS ON SERVER CONFIGURATION IN RHQ-SERVER.PROPERTIES	66
6.4. SYNCHRONIZING SERVER CONFIGURATION	83
CHAPTER 7. CONFIGURING AGENTS	91
7.1. REGISTERING AND RE-REGISTERING THE AGENT	91
7.2. REMOVING AN AGENT	95
7.3. WORKING WITH THE AGENT COMMAND PROMPT	97
7.4. INTERACTIONS WITH SYSTEM USERS FOR AGENTS AND RESOURCES	101
7.5. RUNNING THE AGENT AS A NON-ROOT USER	104
7.6. ENABLING DEBUG MODE FOR THE AGENT	106
7.7. CHANGING THE AGENT IP ADDRESS	108
7.8. MANAGING THE AGENT AS A RESOURCE	109
7.9. CONFIGURING THE AGENT QUIET TIME (TIMEOUT PERIOD)	110
7.10. CONFIGURING AGENT UPDATE SETTINGS	111
7.11. MANAGING THE AGENT'S PERSISTED CONFIGURATION	113
7.12. MANAGING THE AGENT JVM	117

7.13. INSTALLING MULTIPLE AGENTS WITH A SHARED DIRECTORY OR ACCOUNT	118
7.14. SETTING DISCOVERY SCAN INTERVALS	120
7.15. VIEWING THE SERVER FAILOVER LISTS FOR AGENTS	121
7.16. SETTING THE AGENT TO DETECT OR POLL THE SERVER	123
7.17. THROTTLING THE AGENT	125
7.18. SETTING GUARANTEED DELIVERY FOR COMMANDS	126
7.19. CONFIGURING AGENT COMMUNICATION	127
CHAPTER 8. MANAGING DATABASES ASSOCIATED WITH JBOSS ON	129
8.1. RUNNING SQL COMMANDS FROM JBOSS ON	129
8.2. CHANGING DATABASE PASSWORDS	129
8.3. EDITING THE JBOSS ON SERVER'S DATABASE CONFIGURATION	130
CHAPTER 9. DEPLOYING AND MANAGING STORAGE NODES	132
9.1. ABOUT HIGH-SPEED METRICS STORAGE	132
9.2. DEPLOYING AND UNDEPLOYING STORAGE NODES	134
9.3. VIEWING STORAGE NODE METRICS AND STATES	141
9.4. CHANGING CLUSTER PORTS	143
9.5. VIEWING STORAGE NODE ALERTS	144
9.6. ENABLING DEBUG MODE FOR THE STORAGE NODE	146
9.7. MANAGING THE STORAGE NODE HEAP	147
9.8. SCALING THE CASSANDRA STORAGE NODE FOR JBOSS ON SERVER	148
9.9. ADJUSTING THE CAPACITY OF THE STORAGE NODE TO RECEIVE METRICS (STORAGE NODE REQUEST THROTTLING)	150
9.10. BACKING UP AND RESTORING THE METRICS STORAGE DATABASE	153
APPENDIX A. DOCUMENT HISTORY	161

CHAPTER 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.

[Report a bug](#)

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.

[Report a bug](#)

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.

[Report a bug](#)

1.3. ABOUT METRICS STORAGE NODES

Collecting metrics and generating alerts can be resource-intensive. It results in near-constant write operations on the backend database. That creates a natural threshold for the number of metrics that can be collected (30,000 per minute) before encountering performance degradation.



NOTE

The natural threshold of 30,000 metrics per minute is based on internal performance testing. This threshold varies depending on system resources and overall load.

JBoss ON uses two databases to store its information. One is a central relational database (PostgreSQL or Oracle) which stores all configuration about the JBoss ON servers and agents, all resource inventory data, resource configuration, and other data. The other database is a distributed database (a cluster of storage nodes) which stores all numeric monitoring data — in other words, all collected metrics.

The metrics storage node can be installed on its own dedicated machine, which can significantly improve write performance to the database (and, therefore, improve monitoring performance):

- Dedicated CPU
- More available physical memory
- Faster disks
- More disk space

These are the same performance considerations as installing the relational database on a separate machine from the JBoss ON server. By using two databases, it is also possible to move write-intensive and resource-intensive metrics storage away from resource-intensive configuration data, such as drift snapshots and bundle configuration.

Additionally, the distributed database can be expanded, with multiple nodes in a cluster. This ability to add additional nodes according to load is a crucial management tool for administrators. Rather than encountering that hardware-driven limit of 30,000 metrics collected per minute, additional nodes can be added to improve performance.

The storage node cluster is created and managed by JBoss ON (which also minimizes the metrics storage node management overhead). A storage node is installed on a system, using the **rhqctl install --storage** command. The storage node always requires a companion agent.

[Report a bug](#)

CHAPTER 2. GENERAL MANAGEMENT

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

[Report a bug](#)

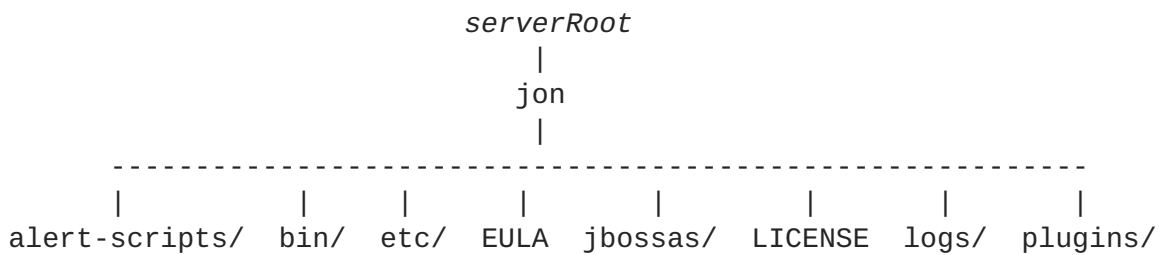
2.1. JBOSS ON FILE LOCATIONS

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.

[Report a bug](#)

2.1.1. 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.



The directories and files that are most commonly used to managed JBoss ON servers are listed in [Table 2.1, “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 2.1. Server Directories and Files

Configuration Area	Directory or File Location	Description
Configuration directory	<i>serverRoot</i> /bin/	Contains the server start scripts, PID files, and configuration file.
Start scripts	<i>serverRoot</i> /bin/rhqctl	The management script which is used to start and stop, install, and upgrade the server, storage nodes, and local agents.
Server configuration file	<i>serverRoot</i> /bin/rhq-server.properties	The configuration file for all server settings that are not stored in the JBoss ON database.
Storage node configuration file	<i>serverRoot</i> /bin/rhq-storage.properties	The configuration file for all storage node settings that are not stored in the JBoss ON database.

Configuration Area	Directory or File Location	Description
Password hash script	<i>serverRoot</i> /bin/rhq-encode-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	<i>serverRoot</i> /etc/RHQ-mib.txt	The JBoss ON MIB file to use for setting SNMP traps.
Log files	<i>serverRoot</i> /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	<i>serverRoot</i> /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	<i>serverRoot</i> /jbossas/	Contains all of the required JBoss AS client and server libraries. [a]
Server JAR files	<i>serverRoot</i> /jbossas/default/deploy/rhq.ear/	Contains all of the JAR files used by JBoss ON servers, web interface, and clients.
Server-side plug-ins directory	<i>serverRoot</i> /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	<i>serverRoot</i> /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	<i>serverRoot</i> /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	<i>serverRoot</i> /jbossas/default/deploy/rhq.ear/rhq-downloads/rhq-agent/	Contains the snapshot packages for the JBoss ON agent.
Web interface directory	<i>serverRoot</i> /jbossas/default/work/jboss.web/localhost/	Contains the directories that hold the files for rendering the web interface.

Configuration Area	Directory or File Location	Description
[a]	Most of the libraries and files in this directory don't relate directly to JBoss ON.	

[Report a bug](#)

2.1.2. 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.

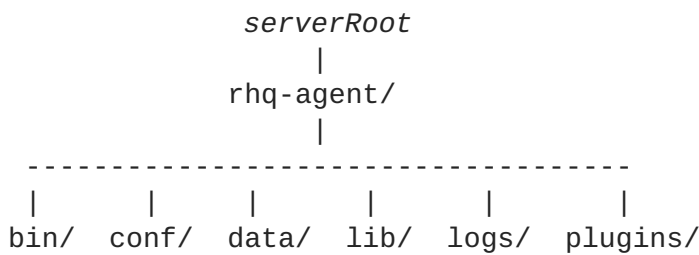


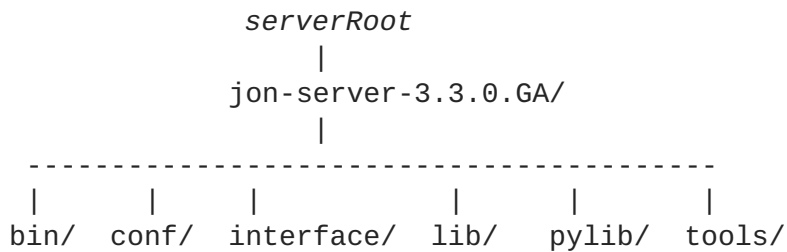
Table 2.2. Agent Directories and Files

Configuration Area	Directory or File Location	Description
Start scripts	<i>serverRoot</i> /rhq-agent/bin/	Contains the agent start scripts.
Configuration file	<i>serverRoot</i> /rhq-agent/conf/agent-configuration.xml	The configuration file for basic agent settings.
Library files	<i>serverRoot</i> /rhq-agent/lib/	Contains the libraries used by the agent to monitor resources.
Start scripts	<i>serverRoot</i> /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	<i>serverRoot</i> /rhq-agent/plugins/	Contains the plug-ins used by the agent for managing resources (like editing resource configuration).

[Report a bug](#)

2.1.3. Storage Node File Locations

The storage node is installed within the server's directory, **/opt/jon/jon-server-3.3.0.GA**.



Additionally, there is a separate directory which stores all of the metrics data and backup data.

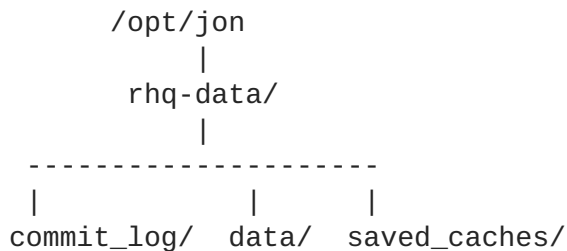


Table 2.3. Storage Directories and Files

Configuration Area	Directory or File Location	Description
Distributed database scripts	<i>serverRoot</i> /rhq-storage/bin/	Contains the scripts used to manage the distributed database nodes. These are invoked most frequently by the JBoss ON agent which manages the storage node, not the administrator.
PID file	<i>serverRoot</i> /rhq-storage/bin/cassandra.pid	The PID file for the node process.
Configuration file	<i>serverRoot</i> /rhq-storage/conf/rhq-storage-auth.conf	The configuration file for basic storage node settings.
Logging configuration file	<i>serverRoot</i> /rhq-storage/conf/log4j-server.properties	The configuration file for log4j for the storage node.
Library files	<ul style="list-style-type: none"> • <i>serverRoot</i>/rhq-agent/lib/ • <i>serverRoot</i>/rhq-agent/pylib/ • <i>serverRoot</i>/rhq-agent/tools/ 	Contains the libraries used by the storage node.
Data files	<i>serverRoot</i> /rhq-data/	The parent directory for the data, cache files, and backups for the distributed storage database nodes.

Configuration Area	Directory or File Location	Description
Data files	<i>serverRoot</i> /rhq-data/commit_log/	Contains binary commit logs which store write operations before they are written to the storage database.
Data files	<i>serverRoot</i> /rhq-data/saved_caches/	Contains database cache files for each database in the data/ subdirectories.
Data files	<i>serverRoot</i> /rhq-data/data/	Contains data files and snapshots for the node, organized by the keyspaces and the table. There are four keyspaces: rhq , system , system_auth , and system_traces .
Data files	<i>serverRoot</i> /rhq-data/data/rhq/	Contains data for all the metric tables. This is all of the new (raw) metrics and tables for each level of aggregation (one hour, six hour, and 24 hour). This also contains schema and index information for the version of metrics being collected.
Data files	<i>serverRoot</i> /rhq-data/data/system/	Contains information for the distributed database nodes.
Data files	<i>serverRoot</i> /rhq-data/data/system_auth/	Contains information for the credentials, users, and permissions configured for the distributed database.
Data files	<i>serverRoot</i> /rhq-data/data/system_traces/	Contains information on database sessions and events.

[Report a bug](#)

2.2. DEFAULT SERVER, AGENT, AND STORAGE NODE 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 several different 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.

Server to storage node communication

The JBoss ON server receives the monitoring data from an agent, and it then forwards that information to an available storage node. The server connects to the storage node over its client (CQL) port.

Storage node to storage node communication

For data integrity and availability, the monitoring data are regularly synchronized between the storage nodes. The nodes send data to one another over a communal cluster port, called a *gossip* 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 2.4, “Default Ports”](#). The port numbers can be changed for any of the JBoss ON services or different values can be used at installation.

Table 2.4. Default 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)
Server to storage node	9142
Storage node to storage node	7100

[Report a bug](#)

2.3. THE RHQCTL CONTROL SCRIPT

JBoss Operations Network has a control script which is used for basic lifecycle management for the server and storage nodes. It can open a server console, start and stop an instance, and install or upgrade components.

The **rhqctl** script has subcommands and options:

```
rhqctl [command] [[options]]
```

The option usually specifies what local instance to perform the task on, **--server**, **--agent**, or **--storage**.

Example 2.1. Installing Specific Services

The **install** command configures the JBoss ON server, storage node, and agent all at the same time.

While it is recommended that all three management services be run on the same system (and from the same parent directory), there may be some environments where it is beneficial to run the JBoss ON server on a separate machine from the storage node. In other cases, it may be required to install the different services at different times.

The **install** command has options for each service. If that option is used, the only that service is installed; the other services are excluded.

For example, this installs the server, storage node, and agent in three separate command invocations:

```
[root@server bin]# ./rhqctl install --storage --start
[root@server bin]# ./rhqctl install --server --start
[root@server bin]# ./rhqctl install --agent --start
```

After a server or a node is installed, the **rhqctl** command is used to start and stop the instance, so start/stop commands are the most common ones to use. Running a command with no options runs that command for all JBoss ON instances on the system.

```
[root@server bin]# ./rhqctl start
```

Using an option runs that command only for that instance.

```
[root@server bin]# ./rhqctl start --server --agent
```

Table 2.5. rhqctl Command

Command	Description
console	Starts a server or storage node in the foreground rather than as a service.
install	Installs a server, storage node, or agent instance.
restart	Restarts a JBoss ON service. If the type of service is not specified, then all local services are restarted.

Command	Description
start	Starts a JBoss ON service. If the type of service is not specified, then all local services are started.
status	Shows the status of a JBoss ON service. If the type of service is not specified, then the status of all local services is displayed.
stop	Stops a JBoss ON service. If the type of service is not specified, then all local services are stopped.
upgrade	Upgrades all JBoss ON services. This can be rerun to migrate historical metric data after a server migration.

Table 2.6. rhqctl Options

Option	Description	Commands That Allow It
--start	Starts all services.	<ul style="list-style-type: none"> • upgrade • install
--server	Runs the given command for the local server. If this is specified, then the command only is run for the server, and the other components are ignored (unless they are explicitly mentioned).	<ul style="list-style-type: none"> • console • restart • start • stop • status • install
--agent	Runs the given command for the agent. If this is specified, then the command only is run for the agent, and the other components are ignored (unless they are explicitly mentioned).	<ul style="list-style-type: none"> • restart • start • stop • status • install

Option	Description	Commands That Allow It
<code>--storage</code>	Runs the given command for the storage database node. If this is specified, then the command only is run for the storage database node, and the other components are ignored (unless they are explicitly mentioned).	<ul style="list-style-type: none"> • console • restart • start • stop • status • install
<code>--storage-data-root-dir <i>directory</i></code>	Changes the directory where the storage data are stored. By default, the storage node directory is <code>serverRoot/jon-server-3.3.0.GA/rhq-data/</code> .	<ul style="list-style-type: none"> • install
<code>--from-server-dir <i>directory</i></code>	Gives the directory path to the server to be upgraded.	<ul style="list-style-type: none"> • upgrade
<code>--from-agent-dir <i>directory</i></code>	Gives the directory path to the agent to be upgraded. The upgrade script assumes that the agent is installed in the same directory as the server, in a subdirectory named <code>rhq-agent/</code> . This option is required if the agent is in a different location.	<ul style="list-style-type: none"> • upgrade
<code>--run-data-migrator</code>	Migrates the data from an existing JBoss ON SQL database into a new monitoring storage database. The upgrade command creates the monitoring database. If this option is not used, any previous monitoring data are lost.	<ul style="list-style-type: none"> • upgrade

[Report a bug](#)

2.4. STARTING THE JBOSS ON SERVER

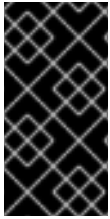
The JBoss ON server is actually a customized JBoss AS server, included in the JBoss ON installation, so starting the JBoss ON server means starting that JBoss instance.

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

[Report a bug](#)

2.4.1. Starting the JBoss ON Server (Basic)

The JBoss ON server process is started by running the `rhqctl` script in the `serverRoot/bin/` directory.



IMPORTANT

The JBoss ON server cannot start or restart *itself* through a resource operation in the command line or UI (unlike other EAP resources). The JBoss ON UI or CLI for one server can be used to start another JBoss ON server in the cloud, but it cannot be perform a start or restart operation reflexively.

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/rhqctl start --server
Trying to start the RHQ Server...
RHQ Server (pid 27547) is starting
```

The `rhqctl` script looks for specific environment variables during its execution, especially related to the JVM to use with the JBoss EAP 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_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.

```
[root@server ~]# ./rhqctl console --server
20:28:44,120 INFO [org.jboss.modules] JBoss Modules version 1.2.2.Final-
redhat-1
Starting RHQ Server in console...
JAVA_OPTS already set in environment; overriding default settings with
values: -
....
```

[Report a bug](#)

2.4.2. Running the JBoss ON Server as a Service

The JBoss ON server, storage node and agent can be managed and run as a SysV style init service on Red Hat Enterprise Linux using the following example:

1. Create a initialization script `/etc/init.d/jboss-on`:

```
#!/bin/sh
#
# Example Red Hat JBoss Operations Network init script
#
# chkconfig: - 95 20
```

```

# description: Red Hat JBoss Operations Network rhqctl services
# processname: standalone.sh
#
#
# This is an example script that can be used with a SysV style
# init service which starts a JBoss ON server, storage node, and
# agent using the rhqctl script provided by the JBoss ON server
# installation.

prog=`basename "$0"`

# If available, the following files will be sourced, in order, for
# configuration. This means that if the same value is defined in a
# later
# file, it will override the one specified in an earlier one:
#
# /etc/jboss-on.conf
# /etc/jboss-on/${prog}.conf
#
# The following environment variables can be defined:
#
# RHQ_HOME -- Defaults to /opt/jboss/jboss-on
# RHQ_SERVER_HOME -- Defaults to RHQ_HOME/jon-server-*
# RHQ_AGENT_HOME -- Defaults to RHQ_HOME/rhq-agent
# RHQ_JAVA_HOME -- Defaults to /usr
# RHQ_USER -- Defaults to jonadmin
#

# If available, source environment variables for the service
# Global defaults/settings
[ -r "/etc/jboss-on.conf" ] && . "/etc/jboss-on.conf"
# Settings specific to this service
[ -r "/etc/jboss-on/${prog}.conf" ] && . "/etc/jboss-
on/${prog}.conf"

# If not yet set, use some reasonable defaults:
[ -z "${RHQ_HOME}" ] && RHQ_HOME='/opt/jboss/jboss-on'
[ -z "${RHQ_SERVER_HOME}" ] && RHQ_SERVER_HOME=`eval echo
"${RHQ_HOME}/jon-server-*`"
[ -z "${RHQ_AGENT_HOME}" ] && RHQ_AGENT_HOME="${RHQ_HOME}"/rhq-
agent'
[ -z "${RHQ_JAVA_HOME}" ] && RHQ_JAVA_HOME='/usr'
[ -z "${RHQ_USER}" ] && RHQ_USER=jonadmin

# We have to export these variables so that they are available to
# child
# processes.
export RHQ_SERVER_HOME
export RHQ_AGENT_HOME
export RHQ_JAVA_HOME

# This is a convenient function that invokes rhqctl
rhqctl() {
  RHQCTL_SCRIPT="${RHQ_SERVER_HOME}/bin/rhqctl"
  [ ! -x "${RHQCTL_SCRIPT}" ] && {

```

```

    echo >&2 "${RHQCTL_SCRIPT}" was not found or is not executable.'
    exit 2
}
if [ -n "${RHQ_USER}" -a "$(whoami)" != "${RHQ_USER}" ]; then
    CMD="su -m ${RHQ_USER} -c '\${RHQCTL_SCRIPT}\\" $@"
else
    CMD="\${RHQCTL_SCRIPT}\" $@"
fi
eval ${CMD}
}

# We invoke rhqctl differently if start or stop is used so we need
this
# case statement to determine what should happen.
case "$1" in
start)
    # Because rhqctl is so chatty/verbose we redirect standard output
to
    # null so we do not see it during system start and stop.
    # Additionally, we invoke it in the background as it can block for
    # some time during start.
    rhqctl "$@" >/dev/null &
    ;;
stop)
    # Because rhqctl is so chatty/verbose we redirect standard output
to
    # null so we do not see it during system start and stop.
    rhqctl "$@" >/dev/null
    ;;
*)
    # Assuming we are not starting or stopping, we will display
complete
    # output.
    rhqctl "$@"
    ;;
esac

```

2. Change the permissions on the file so it can be executed:

```
sudo chmod +x /etc/init.d/jboss-on
```

3. Add the initialization script to chkconfig:

```
sudo chkconfig --add /etc/init.d/jboss-on
```

4. To indicate that you intend for the script to run system startup and stop during shutdown:

```
sudo chkconfig jboss-on on
```

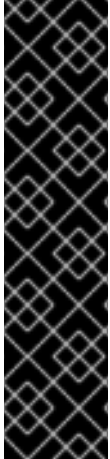
5. To override the default values for **RHQ_HOME**, **RHQ_SERVER_HOME**, **RHQ_AGENT_HOME**, **RHQ_JAVA_HOME** and **RHQ_USER**, create the service configuration file **/etc/jboss-on.conf** or **/etc/jboss-on/<SERVICE NAME>.conf** with your specific values:

```
RHQ_HOME=/usr/share/jboss-on
RHQ_USER=jbosson
```

[Report a bug](#)

2.5. 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 7.11](#), “Managing the Agent's Persisted Configuration”.

[Report a bug](#)

2.5.1. Starting the JBoss ON Agent on a Managed System

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 be started, and left to run in the background.)

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

RHQ 3.3.0-SNAPSHOT [cda7569] (Tue Apr 13 13:39:16 EDT 2017)
>
```

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 [Section 7.3.3](#), “Agent Prompt Commands”. 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.

[Report a bug](#)

2.5.2. Starting an Agent Installed on a Server Machine

If an agent is installed on the same system as a server or a storage node, then the agent is controlled by the same control script (`rhqctl`) that the server and storage node are. To start an agent, use the `--agent` option with the script.

```
[root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl start --agent
```

This starts the agent in the background, rather than opening the agent console.

By default, the agent init script runs as the local system account (*Default* or *.LocalSystem*). The system user account can be changed by editing the **rhq-agent-env.bat** script, as described in [Section 2.5.3, “Configuring the Agent as a Windows Service”](#). The agent on the server machine is still started and stopped using the **rhqctl** command.

[Report a bug](#)

2.5.3. Configuring the Agent as a Windows Service

1. Edit the **rhq-agent-env.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**, verify 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 7.3, “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`.

[Report a bug](#)

2.5.4. 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.

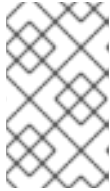
The agent configuration backing store is described in [Section 7.11, “Managing the Agent's Persisted Configuration”](#).

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_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
```

[Report a bug](#)

2.5.5. Starting an Agent with a Custom Command

When configuring the agent to run as a service in [Section 2.5.4, “Running the Agent as a Daemon or `init.d` Service”](#), the agent can be configured to start with a custom start command. This is used mainly to start the agent using `su` or `sudo`, allowing the agent to run as a different user.

The start command is defined with the other agent properties in the `rhq-agent-env.sh` file. There are two parts to the configuration: the start command itself and then a setting to enable a password prompt.

```
RHQ_AGENT_START_COMMAND="su -m test -c '${RHQ_AGENT_HOME}/bin/rhq-agent.sh'"
RHQ_AGENT_PASSWORD_PROMPT=true
```

Setting a start command overrides whatever command is passed in the command line to start the agent.

The password prompt may not be required; it depends on the `sudo` and agent user configuration. If it is required, then the password prompt should be enabled so that the user can enter the password or a password should be set in the `RHQ_AGENT_PASSWORD` parameter; otherwise, the start process will appear to hang.

If the defined start command is invalid, then the agent can fail to start. Along with the command formatting, the directory name can be affected; if there are spaces or special characters in the name, those must be escaped for the command to run.

[Report a bug](#)

2.5.6. 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.

[Report a bug](#)

2.6. STARTING THE STORAGE NODE

The JBoss ON storage node process is started by running the **rhqctl** script in the *serverRoot/bin/* directory for the JBoss ON server.

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

```
[root@server bin]# ./rhqctl start storage
22:03:01,362 INFO [org.jboss.modules] JBoss Modules version 1.2.2.Final-
redhat-1
INFO 22:03:03,138 Logging initialized
INFO 22:03:03,177 JVM vendor/version: OpenJDK 64-Bit Server VM/1.6.0_24
...
INFO 22:03:51,215 Node /10.16.46.34 state jump to normal
INFO 22:03:51,223 Startup completed! Now serving reads.
```

The **rhqctl** script looks for specific environment variables during its execution, especially related to the JVM for the storage node. A complete list of Java settings is given in the **/opt/jon/jon-server-3.3.0.GA/rhq-storage/conf/cassandra-jvm.properties** file.



NOTE

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

```
[root@server ~]# ./rhqctl console --server
20:28:44,120 INFO [org.jboss.modules] JBoss Modules version 1.2.2.Final-
redhat-1
Starting RHQ Storage in console...
JAVA_OPTS already set in environment; overriding default settings with
values: -
....
```

[Report a bug](#)

CHAPTER 3. TUNING JBOSS ON FOR BETTER PERFORMANCE

Every JBoss Operations Network deployment is unique, dependent on the environment which is being managed, the number and type of resources in the inventory, the configuration of individual resources, and the configuration of JBoss ON features such as metrics collection and alert definitions. These differences in environment translate into different performance for the JBoss ON server, agent, and backend database. This can make it difficult to define "universal" baselines or performance expectations.

Still (within the confines of each deployment), it is possible to identify some common behavior characteristics and potential tuning for the JBoss ON configuration which can improve server and agent performance.



NOTE

It is possible to tune a PostgreSQL or Oracle database to improve performance. That is outside the scope of the JBoss ON management. For database tuning information, see the appropriate product documentation.

[Report a bug](#)

3.1. INVENTORY BASELINES

The inventory baseline was determined based on how many resources could be imported into the inventory *at a single time*. For an agent, this parallels first installing the agent, when all resources are discovered and imported at the same time. For the server, this parallels an upgrade operation, when a large number of existing agents are added into its inventory.



IMPORTANT

These baselines are statements of generic JBoss ON deployments in a testing environment, and the information given reflects that environment. This is intended as a reference. It is not intended as a recommendation. It may not reflect a real JBoss ON deployment which has different hardware, resources, and other factors.

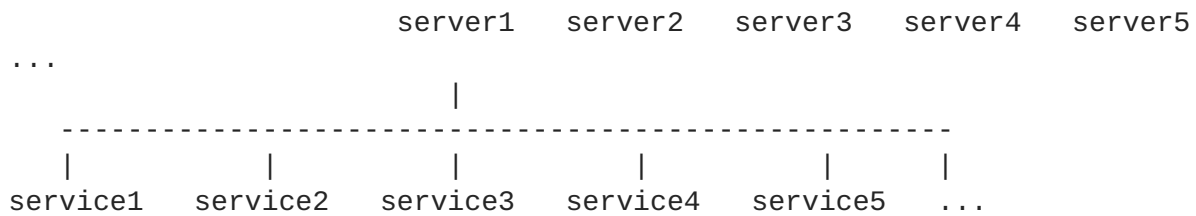
Server Baselines

A server could typically upgrade with 100 or more agents before encountering out of memory errors. If those errors are encountered, the server settings can be tuned as described in [Section 3.6, "Server Tuning for Large Numbers of Agents"](#) to increase the thread pool and concurrency limits to allow more requests to be processed.

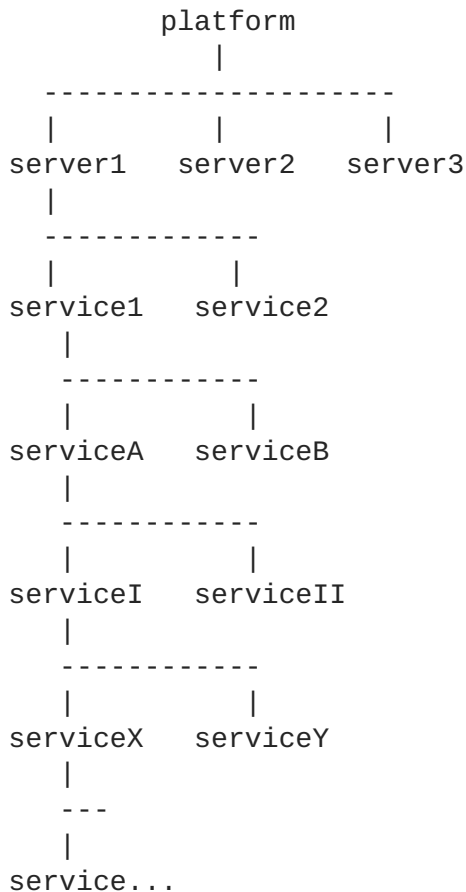
Agent Import Baselines

The resources on a system can be arranged in two slightly different ways. One way is a flat hierarchy, where there are fewer levels in the hierarchy and each level is very broad. Essentially, this is few parent resources with lots of child resources.





Alternatively, the hierarchy can be very deep, where parent resources have few direct children but there are levels of child services. This is especially common with EAP 6 and similar applications which have many subsystems and services.



The hierarchy of the resources does impact how the agent performs when importing large inventories because of how it recurses through the parent and child resources.

Table 3.1. Agent Import Performance

Resource Hierarchy	Tested Import Time	Notes
Deep hierarchy: 10 top-level servers, 10 mid-level servers, 750 child services (10x10x750) (75,000 total resources)	2 hours 46 minutes	Increasing the agent heap size to 2GB moderately decreased the import time, to 2 hours 34 minutes. It also reduced the risk of out-of-memory errors, which could occur when importing a large number of resources with the normal agent memory settings.

Resource Hierarchy	Tested Import Time	Notes
Flat hierarchy: 100 top-level servers, 750 child services (10x10x750) (75,000 total resources)	2 hours 14 minutes	

After garbage collection, the memory footprint during the import operation was an average of 665.3MB. After the import completed, the memory footprint was 586.2MB.

Agent and EAP 6 Resource Baselines

A single agent can manage 40 to 50 JBoss Enterprise Application Platform 6 instances.



NOTE

The number of EAP 6 resources which can be managed by a single agent is highly dependent on the hardware of the underlying system, the number of web contexts managed by each EAP instance, the size of the web contexts, and other factors.

[Report a bug](#)

3.2. MONITORING AND ALERTING BASELINES

Theoretically, there is no limit to the number of metrics that can be collected or the number of alerts that can be fired.

In reality, there are natural constraints within the IT environment that limit both monitoring and alert settings:

- Database performance, which is the primary factor in most environments
- Network bandwidth

There are no hard limits on JBoss ON's alerting and monitoring configuration since it depends on the number of resources, number of metrics, collection frequency, and the number of alerts.

As a rule of thumb, there are these performance thresholds:

- Up to 30,000 metrics can be collected per minute
- Up to 100,000 alerts can be fired per day (roughly 70 per minute)

Plan how to implement metrics collection and alerting. Prioritize resources and then the information required from those resources when enabling metrics schedules and setting collection frequencies. Then, based on those priorities, plan what alerts are required.

Clear monitoring and alerting strategies can help maintain performance while still gathering critical information.



NOTE

Additional storage nodes can be added to the storage cluster to extend the amount of metrics collected.

The storage nodes can be permanent, as part of the JBoss ON design, or can be created dynamically, using an alert condition on the existing nodes to trigger a new node deployment.

Creating additional nodes is covered in [Chapter 9, Deploying and Managing Storage Nodes](#).

[Report a bug](#)

3.3. TUNING 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 being 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. For example, if the agent is running as a service:

```
[root@server ~]# service rhq-agent-wrapper.sh stop
```

2. Open the **rhq-agent-env.sh** file to set the required environment variables that the agent will use.

```
[root@server ~]# vim agentRoot/rhq-agent/bin/rhq-agent-env.sh
```

3. Set the **RHQ_AGENT_JAVA_OPTS** value with the **-Xms** and **-Xmx** parameters for the minimum and maximum bounds of the heap size for the agent JVM. For example:

```
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 process to load the new configuration. For example, if the agent is running as a service:

```
[root@server ~]# service rhq-agent-wrapper.sh start
```

[Report a bug](#)

3.4. CHANGING THE AGENT JVM HEALTH CHECK SETTINGS

One of the more common issues to have with the agent is out of memory errors. Sizing the agent JVM heap is difficult. The agent cannot over-consume system resources or it can affect overall system

performance, yet it must have enough memory available to perform intensive tasks like metrics collection and drift monitoring.

The JVM health check periodically scans the JVM and checks its memory thresholds. If the available memory falls below a specified point, then the JVM process is restarted. This can help prevent the agent from running out of memory and shutting down in an unknown state.



NOTE

The JVM health check is beneficial, but it is not infallible. It is possible in some situations for the memory to be exhausted faster than the scan interval.

There are three parameters for the agent health check:

- The scan interval, **`rhq.agent.vm-health-check.interval-msecs`**
- The heap threshold, **`rhq.agent.vm-health-check.low-heap-mem-threshold`**
- The non-heap memory threshold, **`rhq.agent.vm-health-check.low-nonheap-mem-threshold`**

To change the health check settings:

1. Open the agent prompt. Use the **`-n`** option to open the prompt without

```
[root@server ~]# agentRoot/rhq-agent/bin/rhq-agent.sh -n
```

2. Send the **`setconfig`** with the name of the preference to edit and its new value.

```
> setconfig rhq.agent.vm-health-check.interval-msecs=3000
Set preference: rhq.agent.vm-health-check.interval-msecs=3000
> setconfig rhq.agent.vm-health-check.low-heap-mem-threshold=1.1
Set preference: rhq.agent.vm-health-check.low-heap-mem-threshold=1.1
> setconfig rhq.agent.vm-health-check.low-nonheap-mem-threshold=1.1
Set preference: rhq.agent.vm-health-check.low-nonheap-mem-threshold=1.1
```

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

```
[root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl start --agent
```

[Report a bug](#)

3.5. TUNING THE SERVER JVM

By default, the JBoss ON server runs with a modest allocated heap size and a configured thread limit that parallels the *nix system limit (1024). However, if the JBoss ON server is running on the same system as other resource-intensive applications such as PostgreSQL, then the JVM needs to be tuned to ensure that the JBoss ON server has adequate system resources. Otherwise, the server can encounter out of memory errors.

When tuning the server JVM, increase the heap size and increase the system thread limit to a high enough value that any memory errors are resolved while still leaving sufficient system resources for other applications.

1. As root, increase the user thread limit for the system.

```
[root@server ~]# ulimit -u 4096
```

2. Open the `rhq-server-env.sh` script to set the new JVM settings.

```
[root@server ~]# vim serverRoot/jon-server-3.3.0.GA/bin/rhq-server-env.sh
```

3. Change the Java options to increase the heap size, `-Xms` for the minimum heap and `-Xmx` for the maximum. For example:

```
RHQ_SERVER_JAVA_OPTS="-Xms512M -Xmx1024M -XX:MaxPermSize=128M -Djava.net.preferIPv4Stack=true -Dorg.jboss.resolver.warning=true"
```

4. Restart the server to load the new heap settings.

```
[root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl start --server
```

[Report a bug](#)

3.6. SERVER TUNING FOR LARGE NUMBERS OF AGENTS

When the JBoss ON server has a large number of agents in its inventory (depending on resource counts and configuration like monitoring schedules, this can be as few as 100 or over 1,000 agents), the default settings may not allow it to do an upgrade. The size of the data is enough that the server cannot load all of it cleanly.

While less likely, there could be similar performance issues during regular operation.

One of the most apparent symptoms is frequent timeouts for agent requests.

The problem is not memory related; rather, it is a threading problem. The number of agent requests overwhelms the JBoss ON server, which slows overall process.

There are three parts of the server configuration which can be adjusted to improve performance: increasing the storage node memory settings; increasing the EJB pool; and resetting the concurrency limits to increase the number of allowed agent connections.

1. Increase the default size of the storage node memory usage. This is only necessary for a large number of nodes, around 1,000 or more.

This setting is found in the storage node administration UI. To change the storage node's JVM heap size, enter a new value into the and click the Save button. The configuration change will be applied on disk and then the storage node be restarted so that the change takes effect.

Figure 3.1. Storage node configuration settings

Storage Node (10.16.23.170)

Storage Node Information

Endpoint Address : 10.16.23.170
 Associated Resource : RHQ Storage Node(10.16.23.170)
 Availability : ✔
 CQL Port : 9142
 JMX Port : 7299
 Operation Mode : NORMAL
 Cluster Status : NORMAL
 Installation Date : August 25, 2013 10:20:55 PM UTC-4
 Last Update Time : August 25, 2013 10:20:55 PM UTC-4
 Alerts : New Alerts (0)
 Note : Everything is ok

Chart	Metric	Minimum	Average	Maximum
	Heap Percent Used	3.5 %	33.5 %	72.4 %
	Disk Space Used by Storage Node	4.9 MB	6.1 MB	6.7 MB
	Total Disk Space Percent Used	21.0 %	21.0 %	21.0 %
	Data Disk Space Percent Used	0.0 %	0.0 %	0.0 %
	Free Disk To Data Size Ratio	1731.7	2261.6	2809.8
	Ownership	24.5 %	29.6 %	31.5 %
	Number of Tokens	256	256	256

Configuration

Storage Node Specific Settings

Property	Value	Description
Max Heap Size	512 MB	The maximum heap size. This value will be used with the <code>-Xmx</code> JVM option. If you are going to increase/decrease this value, then you should also increase/decrease the new generation proportionally. The value should be an integer with a suffix of M or G to indicate megabytes or gigabytes.
Heap New Size	128 MB	The size of the new generation portion of the heap. This value will be used with the <code>-Xmn</code> JVM option. The value should be an integer with a suffix of M or G to indicate megabytes or gigabytes.
Thread Stack Size	240	The thread stack size. This memory is allocated to each thread off heap. The value should be an integer that will be interpreted in kilobytes.
JMX Port	7299	The JMX port for the RHQ Storage Node

2. Increase the EJB pool.

- a. Open the server's `standalone-full.xml` profile.

```
[root@server ~]# vim /opt/jon/jon-server-3.3.0.GA/jbossas/standalone/configuration/standalone-full.xml
```

- b. Change the `strict-max-pool` key to increase the pool size. The default is 20. For example:

```
<strict-max-pool name="slsb-strict-max-pool" max-pool-size="2000" instance-acquisition-timeout="1" instance-acquisition-timeout-unit="MINUTES"/>
```



NOTE

If this option is chosen, and the `standalone-full.xml` file is modified, administrators will need to maintain it outside of the JON upgrade process. Patches and updates will revert the file to its default configuration.

3. Increase the concurrency limits to increase how many agents can communicate with the server simultaneously.

- a. Open the `rhq-server.properties` file.

```
[root@server ~]# vim serverRoot/jon-server-3.3.0.GA/bin/rhq-server.properties
```

- b. There is a block of communication-related parameters. Concurrency limits are set in the `concurrency-limit` parameters and the `rhq.communications.global-concurrency-limit` parameter. There are other communication limits for web UI connections and downloads. The different communication parameters are covered in [Section 6.3.3, "Setting Concurrency Limits"](#).

For example:

```
rhq.server.startup.web.max-connections=1000
rhq.server.agent-downloads-limit=45
rhq.server.client-downloads-limit=5
rhq.communications.global-concurrency-limit=200
rhq.server.concurrency-limit.inventory-report=25
rhq.server.concurrency-limit.availability-report=25
rhq.server.concurrency-limit.inventory-sync=25
rhq.server.concurrency-limit.content-report=25
rhq.server.concurrency-limit.content-download=25
rhq.server.concurrency-limit.measurement-report=25
rhq.server.concurrency-limit.measurement-schedule-request=25
rhq.server.concurrency-limit.configuration-update=25
```

- Restart the server to load the new settings.

```
[root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl restart -
-server
```



NOTE

The server will not use the new configuration until it is restarted.

[Report a bug](#)

3.7. DATABASE AND DISK SPACE

Certain JBoss ON features can have a significant impact on storage requirements. Anything that relates to storing content in the JBoss ON database — configuration drift snapshots, bundle versions, and content-backed resources like WARs — increases the storage requirements.

JBoss ON stores all versions of content. Therefore, the system which hosts the backend database (Oracle or PostgreSQL) must have enough disk space to store all versions of all content for any resources using drift monitoring, content updates, and bundles. Additionally, the database itself must have adequate tablespace for the content.

When calculating the required amount of space, estimate the size of every artifact (bundle, web application, monitored directory), and then the number of versions for each artifact. At a minimum, **have twice that amount of space available**; both PostgreSQL and Oracle require twice the database size to perform cleanup operations like vacuum, compression, and backup and recovery.

[Report a bug](#)

CHAPTER 4. 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.

[Report a bug](#)

4.1. 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.
 - a. Shut down the JBoss ON server.

```
serverRoot/jon-server-3.3.0.GA/bin/rhqctl.sh stop
```

- b. Open the `serverRoot/jon-server-3.3.0.GA/bin/rhq-server.properties` file for the JBoss ON server.
- c. 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
```

- d. 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
```

- e. 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=TLSv1,TLSv1.1,TLSv1.2
rhq.server.client.security.secure-socket-protocol=TLS
```

- f. Save the changes, and restart the JBoss ON server.

```
serverRoot/jon-server-3.3.0.GA/bin/rhqctl start
```

- g. Verify that the end point address and port number given in the configuration are actually the settings set for the server in JBoss ON.
 - i. Click the **Administration** tab in the top menu.
 - ii. In the **Topology** box on the left, select the **Servers** item.
 - iii. Check the port number in the **Secure Port** column.

Name ^	Mode	Endpoint Address	Nonsecure Por	Secure Port	Last Update Time	Agent Coun
localhost1	NORMAL	10.0.0.1	7080	7443	Oct 1, 2014 12:35:35 PM	6

- iv. If the value is wrong, click the name of the server to open the edit page.
- v. Click the **Edit** under the server information, and reset the end point address or port as necessary.

▼ **Details**

Name : localhost

Endpoint Address :

Nonsecure Port :

Secure Port :

Operation Mode : ▼

Affinity Group :

Installation Date : September 23, 2014 9:50:08 AM UTC-5

Last Update Time : October 1, 2014 12:37:35 PM UTC-5

Save

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
```

- a. Open the agent configuration file:

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

- b. Change the transport protocol to **sslsocket**.

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

- c. Set the appropriate server connection information so that it matches the configuration for the server. The following configuration would allow the agent to connect to a JBoss ON server on server.example.com port 55555 using sslsocket.

```
<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="" />
```

If using `sslservlet`, the configuration may look like:

```
<entry key="rhq.agent.server.transport" value="sslservlet" />
<entry key="rhq.agent.server.bind-port" value="7443" />
<entry key="rhq.agent.server.bind-address"
value="server.example.com" />
<entry key="rhq.agent.server.transport-params" value="/jboss-
remoting-servlet-invoker/ServerInvokerServlet" />
```

- d. For setting encryption alone, ensure 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" />
```

- e. 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" />
```

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

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

[Report a bug](#)

4.2. 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 achieve this, you must add another web connector to the JON Server's Tomcat configuration that provides a secure but un-authenticated https connector for browsers to use.

Once you add this new web connector, users can connect their browsers to that connector's port (e.g. `https://your-server-hostname:9443`) in order to access the JON GUI using **https** without requiring certificate authentication.

This leaves the original secure Tomcat connector free to accept certificate-authenticated agent requests.

You can add such a web connector using the JBoss EAP CLI located at **`JBOSS_HOME/bin/jboss-cli.[sh,bat]`**.

For example, to add a new web connector on port 9443, you use the JBoss CLI first to create a new socket binding for that port 9443, then to create a new web connector using that new socket binding:

Procedure 4.1. (Optional) Adding a new tomcat web connector

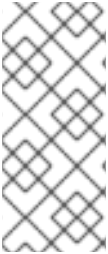
1.

```
jboss-cli.sh --controller=127.0.0.1:6999 --connect --
command='/socket-binding-group=standard-sockets/socket-
binding=httpsbrowser/:add(port=9443)'
```
2.

```
jboss-cli.sh --controller=127.0.0.1:6999 --connect --
command='/subsystem=web/connector=httpsbrowser/:add(socket-
binding=httpsbrowser,scheme=https,protocol=HTTP/1.1,secure=true,enab-
led=true)'
```
3.

```
jboss-cli.sh --controller=127.0.0.1:6999 --connect --
command='/subsystem=web/connector=httpsbrowser/ssl=configuration:add
(name=ssl,verify-client=false,key-
alias=RHQ,password=${VAULT::restricted::rhq.server.tomcat.security.k
eystore.password:5fb458952ebdaa86aa0b4e8d3eac5d13},certificate-key-
file=${jboss.server.config.dir}/rhq.keystore,certificate-
file=${jboss.server.config.dir}/rhq.keystore'
```
4. Restart the JBoss ON server

```
rhqctl restart --server
```



NOTE

The above procedure creates the SSL configuration assuming you want to use the self-signed certificate found in JON's keystore. It is recommended that you use your own certificate, especially if you want one that is not self-signed. To use your own keystore's information, modify in the above procedure (specifically, the key-alias, password, certificate-key-file and certificate-file values).

Procedure 4.2. Setting up client authentication between servers and agents

1. Enable encryption, as in [Section 4.1, "Setting up Encryption"](#), only ensure 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 RSA -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 RSA
- 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.
 - a. Export the self-signed certificate from each keystore:

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

- b. 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.

- c. 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 1 entries
```

```
server1, Feb 25, 2017, trustedCertEntry,
Certificate fingerprint (MD5):
24:D9:8A:50:BA:1B:26:08:DC:44:A8:2A:9E:8A:43:D9
```

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.
 - a. For the server, copy the keystore (**server1-keystore.dat**) into the **jon-server-version/jbossas/standalone/configuration/** directory of the JBoss AS server embedded in the JBoss Operations Network server. Rename **server1-keystore.dat** to **keystore.dat**.
 - b. For the server, copy the truststore into the **serverRoot/jon-server-3.3.0.GA/jbossas/standalone/configuration/** directory of the embedded JBoss AS server. Make sure this file is named **truststore.dat**.
 - c. 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-3.3.0.GA/bin/rhqctl.sh stop
```

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

```
vim serverRoot/jon-server-3.3.0.GA/bin/rhq-server.properties
```

8. Enable client authentication when using sslsocket 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`. When using sslservlet set `rhq.server.tomcat.security.client-auth-mode` parameter to `true` 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 when using sslsocket) 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=${jboss.server.c
onfig.dir}/keystore.dat
rhq.communications.connector.security.keystore.algorithm=SunX509
rhq.communications.connector.security.keystore.type=JKS
rhq.communications.connector.security.keystore.password=secret
rhq.communications.connector.security.keystore.key-password=secret
rhq.communications.connector.security.keystore.alias=server1
rhq.communications.connector.security.truststore.file=${jboss.server
.config.dir}/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=${jboss.server.config.dir}/
keystore.dat
rhq.server.client.security.keystore.algorithm=SunX509
rhq.server.client.security.keystore.type=JKS
rhq.server.client.security.keystore.password=secret
rhq.server.client.security.keystore.key-password=secret
rhq.server.client.security.keystore.alias=server1
rhq.server.client.security.truststore.file=${jboss.server.config.dir
}/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-3.3.0.GA/bin/rhqctl 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="need" />

<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="true" />
```



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
```

[Report a bug](#)

4.3. 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.

[Report a bug](#)

4.3.1. 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.

Figure 4.1. Server Hostname and Port Configuration

Name ^	Mode	Endpoint Address	Nonsecure Por	Secure Port	Last Update Time	Agent Coun
localhost1	NORMAL	10.0.0.1	7080	7443	Oct 1, 2014 12:35:35 PM	6

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 4.3.3, “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 4.2, “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`.

[Report a bug](#)

4.3.2. 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"
```

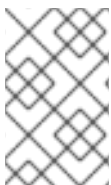
3. Restart the agent.

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

[Report a bug](#)

4.3.3. 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.



NOTE

The examples below show only minimal configuration and assume the use of the default keystore and truststore. Users should update the relevant keystore and truststore properties with appropriate values.

Example 4.1. Encryption Only: Server (sslservlet) and Agent (sslsocket)

Server Configuration	Agent Configuration
<pre>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</pre>	<pre><entry key="rhq.communications.connector.transport" value="sslsocket" /> <entry key="rhq.agent.server.transport" value="sslservlet" /> <entry key="rhq.agent.server.bind-port" value="7443" /></pre>

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 4.2. Encryption Only: Server (sslsocket) and Agent (sslsocket)

Server Configuration	Agent Configuration
<pre>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</pre>	<pre><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="" /></pre>

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

Example 4.3. Encryption and Client Authentication: Server (sslservlet) and Agent (sslsocket)

Server Configuration	Agent Configuration
<pre>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</pre>	<pre><entry key="rhq.communications.connector.transport" value="sslsocket" /> <entry key="rhq.agent.server.transport" value="sslservlet" /> <entry key="rhq.agent.server.bind-port" value="7443" /> <entry key="rhq.communications.connector.security.client-auth-mode" value="need" /> <entry key="rhq.agent.client.security.server-auth-mode-enabled" value="true" /></pre>

Example 4.4. Encryption and Client Authentication: Server (sslsocket) and Agent (sslsocket)

Server Configuration	Agent Configuration
<pre>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=need rhq.server.client.security.server-auth-mode-enabled=true</pre>	<pre><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="" /> <entry key="rhq.communications.connector.security.client-auth-mode" value="need" /> <entry key="rhq.agent.client.security.server-auth-mode-enabled" value="true" /></pre>

[Report a bug](#)

CHAPTER 5. USING HIGH AVAILABILITY AND AGENT LOAD BALANCING

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.

Having multiple JBoss ON servers in a cloud also allow agents to define a preference for which server they use for regular communications. This preference (*affinity*) is a way of load balancing agent-server communications for better overall performance.

[Report a bug](#)

5.1. ABOUT AGENT-SERVER COMMUNICATION AND SERVER AVAILABILITY

5.1.1. Agents and Server Communication

Part of planning whether to use high availability is understanding how agents and servers communicate with each other.

Agents and servers have two-way communication. Agents send current monitoring data, configuration settings, resources, and other current data to the server. The server sends configuration updates, alert definitions, drift definitions, and other settings to the agent.

When an agent is first installed, the agent configuration prompts for the hostname or IP address of a server to connect to. That is the registration server (which can be any server in the JBoss ON deployment). Then, as part of the agent registration, it receives a list of available JBoss ON servers. The first server in that list is the one that the agent attempts to communicate with most regularly, and it tries the other servers in the list in order (more on that in [Section 5.1.5, “Agents and Server Failover”](#)). That first server may be the registration server or it may be a different one; it does not matter.

While there are slight preferences in what server an agent connects to, there are no limits on what agents can connect to what servers and what servers can communicate with what agents. Any server can communicate with any agent at any given time, and vice versa.



NOTE

Because communication must be bidirectional, all servers must be accessible by all agents, and all agents must be accessible by all servers. Having resolvable hostnames and IP addresses for servers and agents — which are configured when the server or agent is installed — is critical.

However, servers never communicate *with each other*, so it is not necessary for servers to be able to resolve each other's hostnames or IP addresses.

[Report a bug](#)

5.1.2. Server Availability: Multiple Servers in a Single Cloud

In many deployments, there is a single JBoss ON server and all agents communicate with that server. However, there are a couple of environments where multiple servers are beneficial:

- There are problems processing the agent load, which can impact evaluating metrics, generating alerts or events, or reporting resource availability. This is not necessarily because of the number of agents; it could be related to network quality or other factors.
- You have a geographically distributed environment with multiple data centers or logical grouping of agents to servers.

Multiple JBoss ON servers in the same deployment are configured to use the same backend database. When a new JBoss ON server is added to the database, that server is automatically added to the *JBoss ON server high availability cloud*.

A high availability configuration does not necessarily imply a large number of JBoss ON agents.

Having multiple servers *does not* affect the ultimate load on the central database, so it does not have a huge effect on performance. The purpose of high availability is that the overall JBoss ON deployment requires responsiveness and availability, as well as fault tolerance, so multiple servers are required. This can be true even with relatively few agents.



IMPORTANT

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 backend database. Each JBoss ON server limits its concurrent database connections, but there is no restriction on the total number of connections across the cloud. Adding a second server can double the potential database connections, even if the number of agents remains the same. The increase in connections is linear as servers are added.

Basically, high availability is a way of providing natural failover and redundancy for the entire JBoss ON deployment. Because all servers use the same database backend, they all have access to the same agents and inventory, monitoring data, resource histories, and other information. This means that all of the JBoss ON servers are essentially identical.

JBoss ON servers can be added to and removed from the high availability cloud easily. Servers can also be temporarily removed by being put into maintenance mode.

There are some things to required when planning high availability:

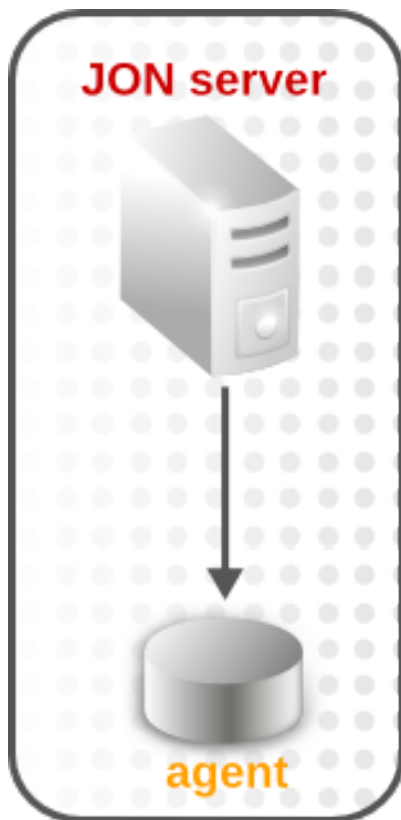
1. All servers must be running the same version of JBoss ON.
2. All servers must be uniquely named. This string is defined during server installation.
3. Each server must define a unique endpoint (hostname or IP address) that is resolvable by *all* JBoss ON agents running against the high availability server cloud.
4. *Optional.* Adjust the concurrency limits on the servers to prevent creating too much load on the database and damaging performance.

[Report a bug](#)

5.1.3. Agents and Server Partitions: Distributing Agent Load

When there is only a single server, the agent distribution is pretty simple: all agents communicate with that one server.

Figure 5.1. All on a Single Server



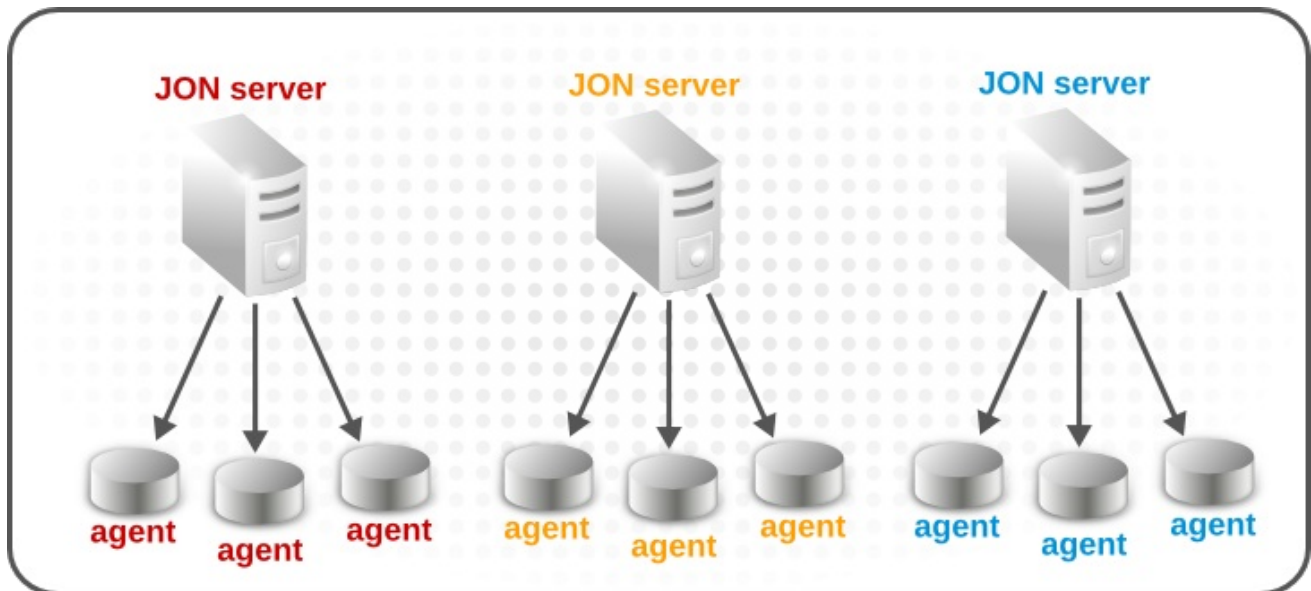
Once high availability is introduced, however, then agents have choices in what server to communicate with. All servers are on a list of available servers which is sent to the agent, but the agent always attempts to contact the first server in its list, its *primary server*.

The server list is slight different for each agent because the list is generated in a round-robin pattern. For example:

```
A1: S1, S2, S3  
A2: S2, S3, S1  
A3: S3, S1, S2
```

This creates a fairly even distribution of agents across the servers. The distribution is a *partition*.

Figure 5.2. Partitions: Agent Load Distributed Among Multiple Servers



As servers are added to the high availability cloud, the agent-server lists are updated. Changing the agent load distribution is called a *partition event*.

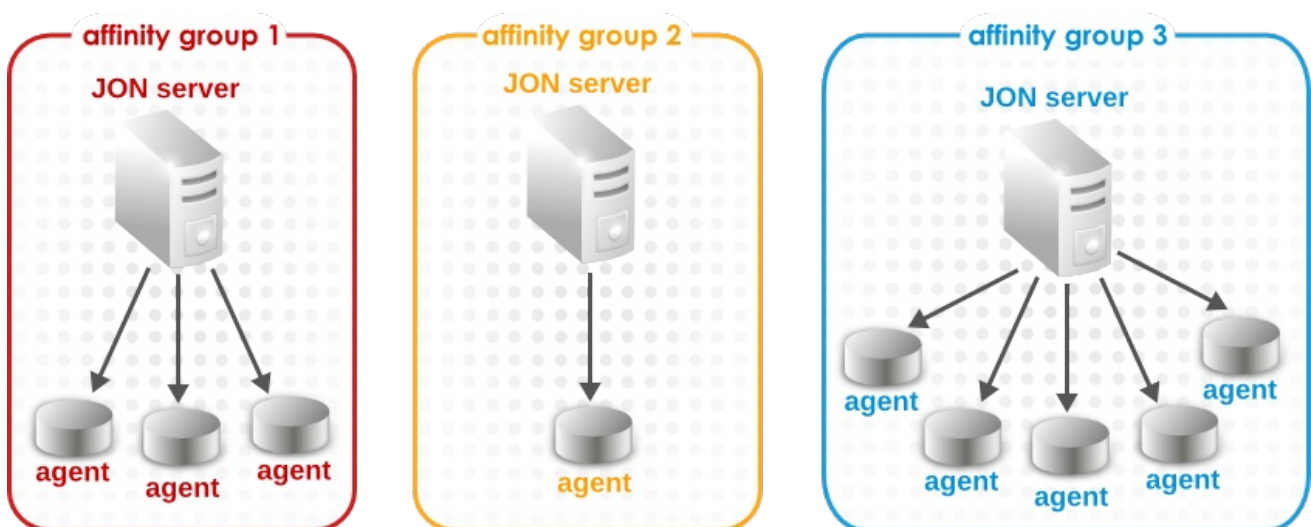
[Report a bug](#)

5.1.4. Agents and Preferred Servers: Affinity and Load Balancing

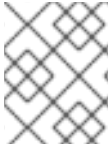
The natural distribution of the agent load is fairly random and creates an even distribution. However, in some network environments, a random distribution doesn't really make sense or provide the best efficiency. For example, servers and agents in the same region or facility can communicate faster than servers and agents that are much further away. In that case, an administrator would prefer an explicit, sensible server-agent relationship instead of a random one.

JBoss ON has the concept of server-agent *affinity*. An affinity is a defined preference for what server an agent communicates with. An affinity group is essentially a manual partition. It is a group of servers and agents, and the agents selectively communicate with servers in their affinity group first.

Figure 5.3. Affinity Preferences for Agent Load



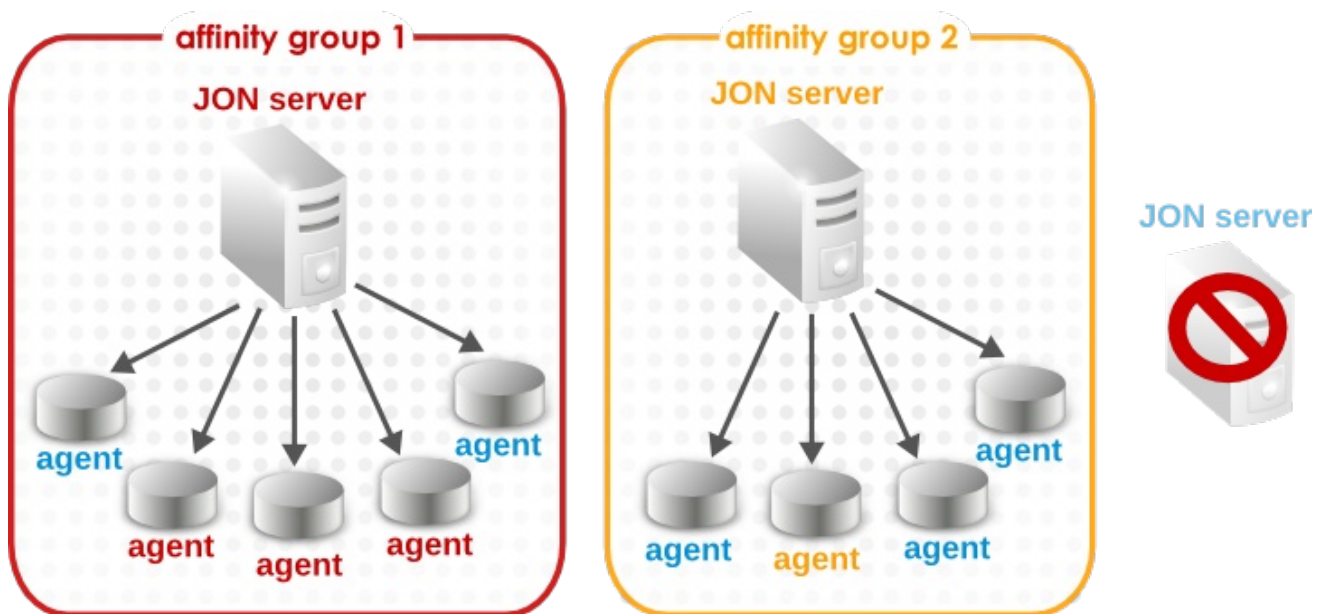
An affinity group creates a loose preference for an agent on what servers it communicates with — it does not create a hard rule or restrict what servers the agent can communicate with.

**NOTE**

An affinity group defines a *one-way preference*, from the agent to the server. Any server can contact any agent in the JBoss ON topology, regardless of any affinity preference.

When an agent's primary server is unavailable, the agent attempts to round-robin through the other servers in its affinity group. If none of those servers are available or there are no other servers in the affinity group, then it iterates through all of the servers in the JBoss ON high availability cloud, according to its failover list. That is true for all of the agents in the affinity group, so eventually any agents in one group would be evenly distributed among other JBoss ON servers.

Figure 5.4. Failover with Affinity



Affinity groups provide (at least) three potential advantages:

- **Physical or network 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 organization.** 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.

[Report a bug](#)

5.1.5. Agents and Server Failover

There is a central list of servers which is provided to each agent to identify what servers are available to that agent. This is the *failover list*. When a new server joins the cloud, it is added to the list and the list is updated to the agent.

Whatever server is first in the list for the agent is the server it most frequently communicates with — its *primary server*. If the agent cannot connect to that server, then it attempts to connect to the next server in the list, until it finds an available server.

The agent checks back periodically (every hour) to see when its primary server is back online and switches back to that server as soon as it is back.

For a regular distribution of agents, the agent runs through all available servers in a (relatively) random order, according to whatever failover list it was provided. If the agent belongs to an affinity group, it first tries all of the servers in that affinity group, and then moves on to servers outside the affinity group in whatever order is set in its failover list.

[Report a bug](#)

5.2. 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 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

Agents only 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, therefore 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.



NOTE

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

Figure 5.5. Listing Affinity Groups

■ Affinity Groups

Total Rows: 2 (selected: 0)

Name	Agent Count	Server Count
South Data Center	1	0
West Data Center	0	0

Procedure 5.1. Create an Affinity Group**NOTE**

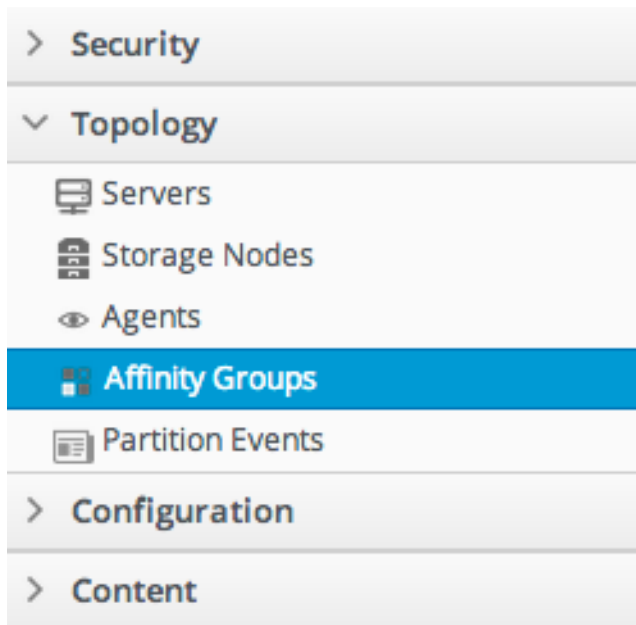
To edit an affinity group, click its name, then manage it the same as creating a new affinity group.

1. Click the **Administration**

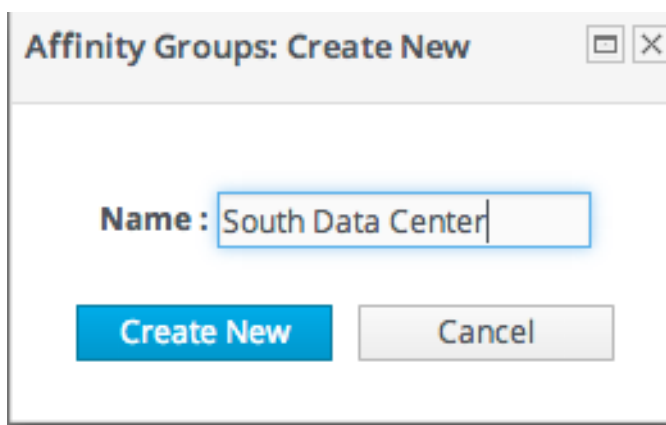


tab in the top menu.

2. Click .



3. Click the **Create New** button.
4. Enter a name for the new affinity group, and click **Create New**.



5. In the **Agent Members** group, click the checkboxes by the agent names to add to the group, and click **Update Membership**.

Affinity Groups

« Back to List

▼ Details

Name :

[Save](#)

▼ Agent Members

Total Rows: 0 (selected: 0)

Name ▲	Connected Server	Agent Bind Address	Agent Bind Por	Last Availability Repo	Last Availability Ping
No items to show					

[Update Membership...](#) [Refresh](#)

▼ Server Members

Total Rows: 0 (selected: 0)

Name ▲	Mode	Endpoint Address	Nonsecure Por	Secure Port	Last Update Time
No items to show					

[Update Membership...](#) [Refresh](#)

- In the **Server Members** group, click the checkboxes by the server names to add to the group, and click **Update Membership**.

Affinity Groups

[« Back to List](#)

▼ Details

Name :

[Save](#)

▼ Agent Members

Total Rows: 0 (selected: 0)

Name ▲	Connected Server	Agent Bind Address	Agent Bind Port	Last Availability Repo	Last Availability Ping
No items to show					

[Update Membership...](#) [Refresh](#)

▼ Server Members

Total Rows: 0 (selected: 0)

Name ▲	Mode	Endpoint Address	Nonsecure Port	Secure Port	Last Update Time
No items to show					

[Update Membership...](#) [Refresh](#)

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

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

[Report a bug](#)

5.3. 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.

Procedure 5.2. Set Maintenance Mode on a Server

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



2. Click .
3. Click the row to select the JBoss ON server to put into maintenance mode.
4. Click the **Set Maintenance** button.

 Servers

Total Rows: 1 (selected: 1)

Name ▾	Mode	Endpoint Address	Nonsecure Por	Secure Port	Last Update Time	Agent Coun ▾
localhost1	NORMAL	10.0.0.1	7080	7443	Oct 1, 2014 12:59:05 PM	6

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.

[Report a bug](#)

5.4. 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.

Procedure 5.3.

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

[Dashboard](#)[Inventory](#) ▾[Reports](#) ▾[Bundles](#)[Administration](#) ▾[Help](#)

2. Click .
3. Click to highlight the row of the JBoss ON server to remove, and click **Set Maintenance** to remove from the cloud.
4. When the screen refreshes, click to highlight the row of the JBoss ON server to remove, and click **Remove Selected**.

Servers

Total Rows: 1 (selected: 1)

Name ▾	Mode	Endpoint Address	Nonsecure Port	Secure Port	Last Update Time	Agent Count
localhost1	NORMAL	10.0.0.1	7080	7443	Oct 1, 2014 12:59:05 PM	6

[Set Normal](#)
[Set Maintenance](#)
[Remove Selected](#)
[Refresh](#)

[Report a bug](#)

5.5. MANAGING PARTITION EVENTS

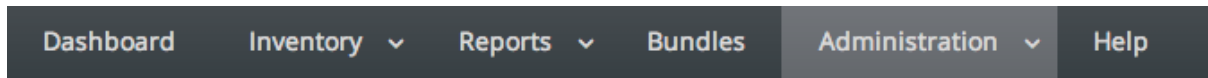
When an agent is assigned to be managed by a server, that is a *partition*. *Partition events* are almost like log messages that occur whenever a change in the partition configuration occurs.

[Report a bug](#)

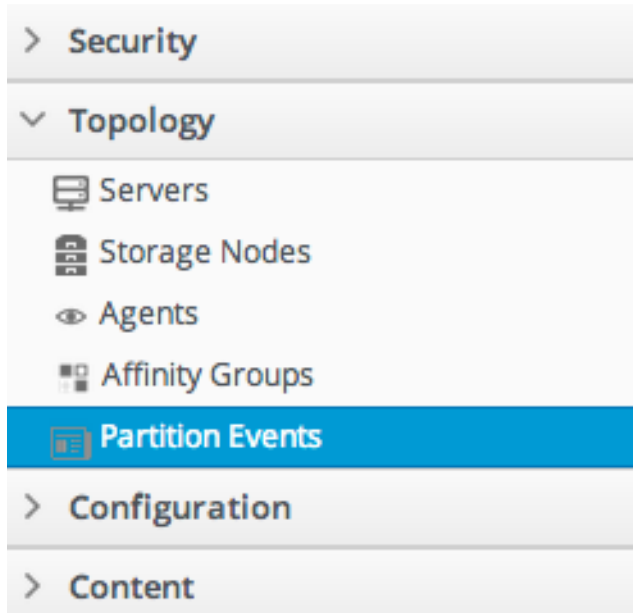
5.5.1. Viewing Partition Events

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 5.2, “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.

Partition Events

Partition Events

Execution Status Filter :		AUDIT, COMPLETED, IMMEDIATE, REQUESTED		Details Filter :		Total R	
Type Filter :		AGENT_REGISTRATION, AGENT_CONNECT, AGEN					
Execution Time	Type	Details	Initiated By	Execution Status			
Sep 23, 2014 9:50:56 AM	OPERATION_MODE_CHANGE	localhost1: INSTALLED -> DOWN	admin	AUDIT			
Sep 23, 2014 9:52:30 AM	OPERATION_MODE_CHANGE	localhost1: INSTALLED -> NORMAL	admin	AUDIT			
Sep 23, 2014 9:54:27 AM	OPERATION_MODE_CHANGE	localhost1: INSTALLED -> DOWN	admin	AUDIT			
Sep 23, 2014 9:55:51 AM	OPERATION_MODE_CHANGE	localhost1: INSTALLED -> NORMAL	admin	AUDIT			
Sep 23, 2014 9:56:52 AM	AGENT_REGISTRATION	localhost1 - localhost1	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 5.1, “Types of Partition Events”](#).

Table 5.1. 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.
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.

Partition Event	Description
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 5.2. 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 .

Field	Description
Execution Status	<p>Shows low for status descriptions. There are four different status settings:</p> <ul style="list-style-type: none"> • <i>Audit</i> shows that a change was made, but not an event that affects the partition topology. • <i>Immediate</i> shows that a partition change was made at the time of the event. • <i>Requested</i> 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. • <i>Completed</i> shows that a change has been completed.

[Report a bug](#)

5.5.2. 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

Figure 5.6. Removing Partition Events

Sep 23, 2014 2:38:01 PM	AGENT_CONNECT	localhost1 - localhost1	admin	AUDIT
Sep 23, 2014 12:37:48 PM	AGENT_REGISTRATION	localhost1 - localhost1	admin	IMMEDIATE
Sep 23, 2014 3:27:57 PM	AGENT_REGISTRATION	localhost1 - localhost1	admin	IMMEDIATE

[Report a bug](#)

CHAPTER 6. 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.

[Report a bug](#)

6.1. 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-3.3.0.GA/logs/server.log`.

[Report a bug](#)

6.1.1. Setting the Debug Environment Variable

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.

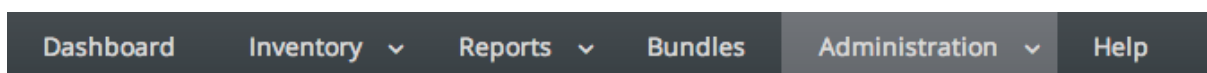
The quickest way to enable debug logging is to set the `RHQ_SERVER_DEBUG` environment variable to any value before starting the server.

[Report a bug](#)

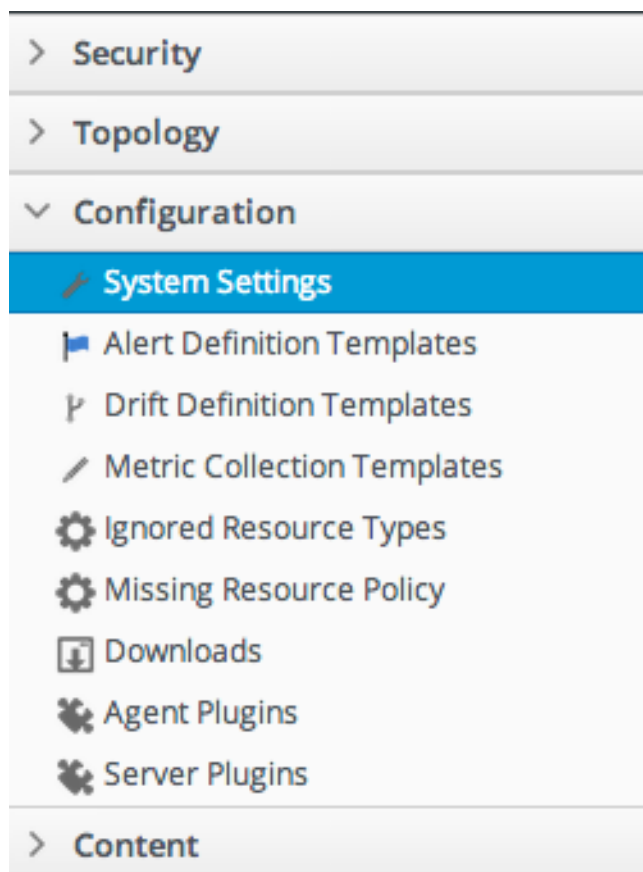
6.1.2. Dumping Current Server State to the Logs

Having a record of the current state of the server configuration can be useful for debugging and auditing. The current server details — such as its build number, database information, and measurement schedules — can be exported immediately to the server log.

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



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



3. In the main window, scroll to the bottom of the server configuration, and click the **Dump system info** button.

The screenshot shows the Administration tab of the JBoss ON console. The top navigation bar includes 'Inventory', 'Reports', 'Bundles', 'Administration', and 'Help'. The user 'rhqadmin' is logged in. Below the navigation bar, there are 'Favorites' and 'Message Center' buttons. The main content area is titled 'Server Details' and displays the following information:

- Name:** JBoss ON
- Version:** 3.1.0.ER3
- Build Number:** ca099bc:3a46aff
- Server Time Zone:** Eastern Standard Time
- Server Local Time:** May 20, 2012 10:46:08 PM EDT
- Server Installation Directory:** /usr/share/jboss-on-3.1.0.ER3/server
- Database Connection URL:** jdbc:postgresql://127.0.0.1:5432/rhq?loginTimeout=0&socketTimeout=0&prepareThreshold=5&unknownLength=2147483647&log
- Database Product Name:** PostgreSQL
- Database Product Version:** 8.4.9
- 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 21, 2012 8:00:00 AM EDT

Below the server details, there is a 'Jump to Section' dropdown menu and a list of configuration property sections: 'General Configuration Properties', 'Data Manager Configuration Properties', 'Automatic Baseline Configuration Properties', and 'LDAP Configuration Properties'. The 'LDAP Configuration Properties' section is expanded, showing a table with the following columns: 'Property', 'Unset?', 'Value', and 'Description'.

Property	Unset?	Value	Description
Enable LDAP	<input type="checkbox"/>	<input type="radio"/> Yes <input checked="" type="radio"/> No	Should LDAP be used to determine user identity?
Search Base	<input type="checkbox"/>	<input type="text" value="o=JBoss,c=US"/>	The base of the directory tree to search for usernames and passwords while authenticating users, e.g. ou=People,dc=redhat,dc=com

At the bottom of the configuration section, there are two buttons: 'Save' and 'Dump system info'. The 'Dump system info' button is circled in red.

- All of the current server settings and details are printed to the server log.

```

2012-05-14 19:44:28,587 INFO [SystemInfoManager] SystemInformation:
*****
CAM_LDAP_BIND_PW: [- non null -]
AlertDefinitionCount: [2]
CAM_LDAP_BASE_DN: [o=JBoss,c=US]
AVAILABILITY_PURGE: [31536000000]
CAM_JAAS_PROVIDER: [false]
BuildNumber: [ca099bc:3a46aff]
ServerCount: [27]
DATABASE_DRIVER_NAME: [PostgreSQL Native Driver]
RESOURCE_GENERIC_PROPERTIES_UPGRADE: [false]
... 8< ...

```

[Report a bug](#)

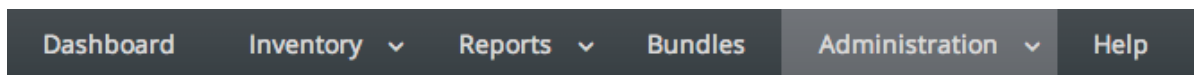
6.2. CHANGING THE JBOSS ON SERVER URL

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

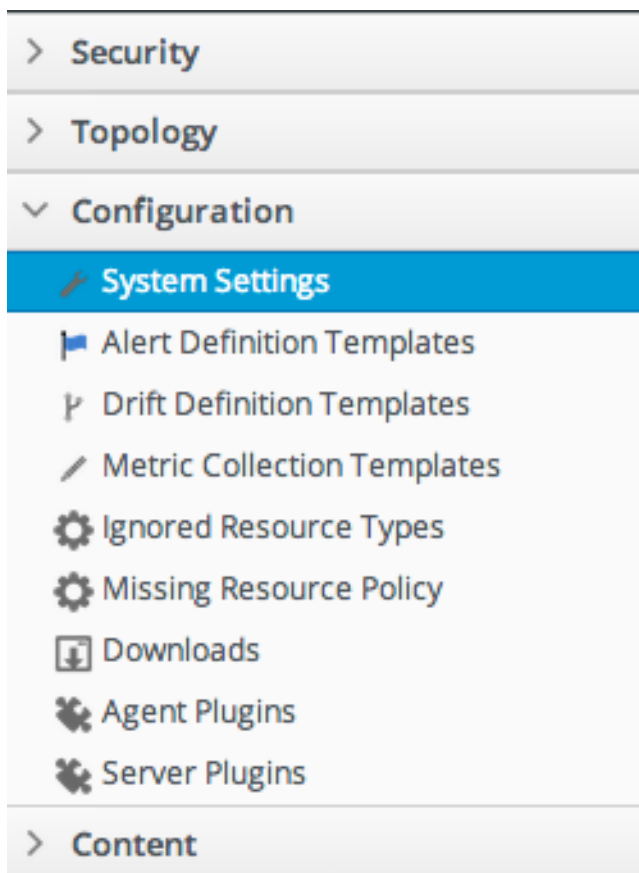
- Connecting to the GUI
- Details on alerts, contained in email notifications of alerts

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.

Server Details

Product Name : JBoss ON
 Version : 3.3.0
 Build Number : 0340db5:17b2d55
 Server Name : localhost1
 Server Time Zone : Eastern Standard Time
 Server Local Time : October 1, 2014 3:23:55 PM EDT
 Server Installation Directory : /home/hudson/jon-server-3.3.0
 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 : 8.4.11
 Database Driver Name : PostgreSQL Native Driver
 Database Driver Version : PostgreSQL 9.2 JDBC4 (build 1002)

Jump to Section

▼ **General Configuration Properties**

Property	Unset?	Value	Description
Agent Max Quiet Time Allowed		<input type="text" value="5"/>	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		<input type="text" value="http://10.0.0.1:7080/"/>	A URL to the server GUI, used mainly within alert email notifications.
Enable Agent Auto-Updates		<input checked="" type="radio"/> Yes <input type="radio"/> 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		<input type="radio"/> Yes <input checked="" type="radio"/> No	If enabled, the server will enter debug mode.
Enable Experimental Features		<input type="radio"/> Yes <input checked="" type="radio"/> No	If enabled, any experimental features that exist in the current product will be available.
Enable Login Without Roles		<input type="radio"/> Yes <input checked="" type="radio"/> No	If enabled, the coregui will block users that are not assigned to any role. This applies for both LDAP and normal users.
Default Remote SSH Password	✓	<input type="text"/>	The default SSH password to be used when logging into a remote machine for doing things like controlling or installing remote agents.
Default Remote SSH Username	✓	<input type="text"/>	The default SSH username to be used when logging into a remote machine for doing things like controlling or installing remote agents.

5. Click **Save**.

[Report a bug](#)

6.3. 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-3.3.0.GA/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.

[Report a bug](#)

6.3.1. Properties Set at Installation

The defining server properties — such as the server name, port, and the database to use — are set when the server instance is configured.

Changing the database properties is covered in [Chapter 8, Managing Databases Associated with JBoss ON](#).

title

Database Type

This sets the type or vendor of the database that is used by the JBoss ON server. This is either PostgreSQL or Oracle.

Database Connection URL

This gives the JDBC URL that the JBoss ON server uses when connecting to the database, such as *jdbc:postgresql://localhost:5432/rhq* or *jdbc:oracle:oci:@localhost:1521:orcl*.

Database JDBC Driver Class

This gives the fully qualified class name of the JDBC driver that the JBoss ON server uses to communicate with the database, such as *oracle.jdbc.driver.OracleDriver*.

Database XA DataSource Class

This gives the fully qualified class name of the JDBC driver that the JBoss ON server uses to communicate with the database, such as *org.postgresql.xa.PGXADatasource* or *oracle.jdbc.xa.client.OracleXADatasource*.

Database User Name

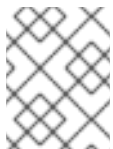
This gives the name of the user that the JBoss ON server uses when logging into the database. This user must already exist in the database; according to the basic installation instructions, this is a specially-created **rhqadmin** user (not related to the super user in JBoss ON).

Database Password

This gives 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 8.2, “Changing Database Passwords”](#).

Server Name

This sets a unique name for the JBoss ON server. The default is the system's hostname, but it can be any string, as long as it is unique within the JBoss ON server cloud.

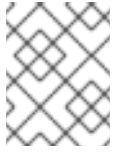


NOTE

Unlike other server properties, this is only managed through the JBoss ON UI, not its **rhq-server.properties** file.

Server Public Address

This gives the public IP address to use for the server. This is the address that must be recognized by all agents needing access to this server. By default, this is the value of the localhost's public IP address. The public IP address is used with the HTTP/HTTPS ports to provide a high availability endpoint for agents.



NOTE

Unlike other server properties, this is only managed through the JBoss ON UI, not its `rhq-server.properties` file.

HTTP Port

This sets the port that the JBoss ON GUI listens to for unsecured HTTP requests. This is the port number in the JBoss ON GUI URL, such as the 7080 in `http://localhost:7080`. This is also the unsecured port used as the endpoint in high availability.

Secure HTTPS Port

This sets the port that the JBoss ON GUI listens to for secured HTTPS requests. This is the port number used in the JBoss ON GUI URL, such as the 7443 in `https://localhost:7443`. This is also the secure port used as the endpoint in high availability.

Server Bind Address

This 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 URL, such as `server.example.com` in `http://server.example.com:7080`.

Email SMTP Hostname

This sets the hostname of the SMTP server used by the JBoss ON server. Emails are sent primarily for alert notifications.

Email From Address

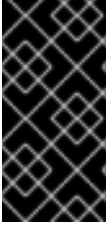
This sets the address to use for the *From* header of all emails sent by the JBoss ON server.

[Report a bug](#)

6.3.2. 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 user-configurable server connection parameters are described in the server property lists.

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 4.2, “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=true
rhq.server.tomcat.security.secure-socket-protocol=TLSv1, TLSv1.1, TLSv1.2
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 6.1. 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 6.1, “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.

rhq-server.properties General Server Connection Parameters

jboss.bind.address [\[1\]](#)[\[2\]](#)

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 URL, such as **server.example.com** in `http://server.example.com:7080`.

rhq.server.startup.web.http.port [\[1\]](#)[\[2\]](#)

Gives the port that the JBoss ON GUI listens to for unsecured HTTP requests. This is the port number in the JBoss ON GUI URL, such as the 7080 in `http://localhost:7080`. This is also the unsecured port used as the endpoint in high availability.

rhq.server.startup.web.https.port [\[1\]](#)[\[2\]](#)

Gives the port that the JBoss ON GUI listens to for secured HTTPS requests. This is the port number in the JBoss ON GUI URL, 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 [\[2\]](#)

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/standalone/configuration** directory. The JBoss ON server ships with a self-signed certificate in a default keystore file.

rhq.server.startup.keystore.password [\[2\]](#)

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 [\[2\]](#)

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.

rhq-server.properties SSL Server Connection Parameters

rhq.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.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.connector.security.keystore.algorithm (agent to server) ,
rhq.server.client.security.keystore.algorithm (server to agent)

rhq.connector.security.keystore.type (agent to server) ,
rhq.server.client.security.keystore.type (server to agent)

The file format of the keystore.para

rhq.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.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.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.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. See the *Incoming Client Authentication Mode*.

rhq.connector.security.truststore.algorithm (agent to server) ,
rhq.server.client.security.truststore.algorithm (server to agent)

rhq.connector.security.truststore.type (agent to server) ,
rhq.server.client.security.truststore.type (server to agent)

The file format of the truststore file.

rhq.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.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: **true** or **false**.

- **rhq.communications.connector.security.client-auth-mode:** none, want or need.
- **rhq.server.client.security.server-auth-mode-enabled:** true or false.

rhq-server.properties Transport Connection Server Parameters

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.

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 6.1, "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.

Table 6.1. Transport Parameters

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 is 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

Transport Parameter	Description	For Incoming Messages or for Outgoing Messages
maxPoolSize	The number of server threads for processing client requests. The default is 300.	Incoming
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 <code>CannotConnectException</code> will be thrown. The default is 30.	Outgoing
numberOfCallRetries	The number of retries for making the invocation. This is unrelated to <code>numberOfRetries</code> 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 <code>numberOfCallRetries</code> is (which defaults to 3). However, when $(\text{numberOfCallRetries} - 2)$ 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 <code>MarshalException</code> 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 is false by default.	Outgoing

[Report a bug](#)

6.3.3. 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 6.2, “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 connections. 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 `sslservlet` 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 6.2. 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 6.2. rhq-server.properties Parameters for Concurrency Limits

Parameter	Description
rhq.server.startup.web.max-connections	<p>Sets a limit on the number of web connections that can be concurrently created, including both connections to the GUI and connections by agents.</p> <div style="display: flex; align-items: flex-start;"> <div style="flex: 1;">  </div> <div style="flex: 2;"> <p>NOTE</p> <p>If agent requests are routed over web connections, make sure that the <i>rhq.communications.global-concurrency-limit</i> 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.</p> </div> </div> <p>The limit on web connections is the same for both HTTP and HTTPS (secure) requests, so the <i>total</i> 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.</p>

Parameter	Description
rhq.communications.global-concurrency-limit	<p>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.</p> <p>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.</p> <p>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.</p>
rhq.server.concurrency-limit.inventory-report	<p>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.</p>
rhq.server.concurrency-limit.availability-report	<p>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.</p>
rhq.server.concurrency-limit.inventory-sync	<p>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.</p>
rhq.server.concurrency-limit.content-report	<p>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.</p>
rhq.server.concurrency-limit.content-download	<p>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.</p>

Parameter	Description
<code>rhq.server.concurrency-limit.measurement-report</code>	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.
<code>rhq.server.concurrency-limit.measurement-schedule-request</code>	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.

[Report a bug](#)

6.3.4. 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/portal/admin/test/email.jsp>.

Table 6.3. rhq-server.properties Parameters for SMTP

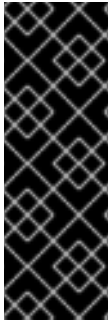
Parameter	Description
<code>rhq.server.email.smtp-host</code>	Sets the hostname of the SMTP server used by the JBoss ON server.
<code>rhq.server.email.smtp-port</code>	Sets the port of the SMTP server used by the JBoss ON server.
<code>rhq.server.email.from-address</code>	Sets the address to use for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

[Report a bug](#)

6.3.5. 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=true
rhq.autoinstall.database=auto
rhq.autoinstall.public-endpoint-address=
```



IMPORTANT

The autoinstaller options are only evaluated once, when the JBoss ON server is first installed. After that initial configuration, the autoinstaller is disabled. These properties are ignored once the server is set up and cannot be used to initiate a re-install of an existing instance.

To re-install the server, first delete the server installation directory, then unzip the original JBoss ON server archive and install the server as if it were new.

Table 6.4. rhq-server.properties Parameters for Silent Installation

Parameter	Description
<code>rhq.autoinstall.enabled</code>	Tells the installation process whether to load the configuration from the <code>rhq-server.properties</code> file (true) or from the web-based installer (false).
<code>rhq.autoinstall.database</code>	Tells the install process how to load or add database schema. There are three options: <ul style="list-style-type: none"> • 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.
<code>rhq.autoinstall.public-endpoint-address</code>	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.

[Report a bug](#)

6.3.6. Protecting sensitive information in the Server Configuration



IMPORTANT

By default all the passwords in the server configuration files are protected during upgrades and installations. This process is only needed when users want to protect additional properties.

JBoss ON supports encoding almost all properties in server configuration files, more specifically, the `{RHQ_SERVER_HOME}/jbossas/standalone/configuration/standalone-full.xml` and `{RHQ_SERVER_HOME}/bin/rhq-server.properties` files. JBoss ON provides the `rhq-encode-value.sh` and `rhq-encode-value.bat` scripts in the `{RHQ_SERVER_HOME}/bin/` directory to obfuscate properties.

[Report a bug](#)

6.3.6.1. standalone-full.xml

For `standalone-full.xml`, property values are encoded by a password vault mechanism and all passwords are protected by default at installation. This password vault mechanism is provided as a feature in JBoss Enterprise Application Platform 6. JBoss ON also provides this feature using a modified implementation of the same password vault mechanism for use within JBoss ON. For more information on the password vault mechanism, please see [Password Vaults for Sensitive Strings section of the JBoss Enterprise Application Platform 6 Administration and Configuration Guide](#).

[Report a bug](#)

6.3.6.2. rhq-server-properties

For `rhq-server.properties`, property values are encoded using the `RESTRICTED::` format and all password are protected by default at installation.

[Report a bug](#)

6.3.6.3. Using the rhq-encode-value Script for Encoding

Invoking the `rhq-encode-value` script will prompt users for the desired property and value to encode:

```
> ./rhq-encode-value.sh
Property rhq.autoinstall.server.admin.password [y/n]: n
Property rhq.server.database.password [y/n]: n
Property: rhq.protect.property
Value: 1234
*** !!! WARNING !!!
*** Both standalone-full.xml and rhq-server.properties need to be updated
if a property from rhq-server.properties is used in standalone-full.xml
*** !!! WARNING !!!
***
***
*** Encoded password for rhq-server.properties:
***   rhq.protect.property=RESTRICTED::-299a94df3b478ca8
***
*** Encoded password for standalone-full.xml with vault with password as
default value:
***   ${VAULT::restricted::rhq.protect.property::-299a94df3b478ca8}
***
*** Encoded password for standalone-full.xml with vault without default:
```

```

***      ${VAULT::restricted::rhq.protect.property:: }
***
*** Encoded password for agent-configuration.xml:
***      <entry key="rhq.protect.property" value="RESTRICTED::-
299a94df3b478ca8" />
***
*** Please consult the documentation for additional help.

```

After running the script, the values generated from **rhq-encode-value** should be copied and pasted into server configuration files.



IMPORTANT

Some properties present in **standalone-full.xml** are also present in **rhq-server.properties**. When protect a property present in both files, ensure that both locations are updated with the same encoded variant prior to restarting the server. Partial updates may render the server unusable.

[Report a bug](#)

6.4. 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. See the JBoss ON API Documentation on the [Customer Portal](#) for more information.

[Report a bug](#)

6.4.1. Exporting a Server's Configuration

1. Log into the JBoss ON CLI.

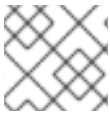
```
[root@server bin]# installDir/bin/rhq-cli.sh -u rhqadmin -p rhqadmin
```

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.

[Report a bug](#)

6.4.2. 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.

[Report a bug](#)

6.4.2.1. 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>

```

[Report a bug](#)

6.4.2.2. 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.entpris
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)
```

[Report a bug](#)

6.4.2.2.1. 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 synchronizers 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:values>
  </ci:list-property>
</default-configuration>
```

```

        <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: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`.

[Report a bug](#)

6.4.2.2.2. 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(defaultSettingsToImport + ', ' + 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)
```

[Report a bug](#)

6.4.2.3. 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 6.4.2.2, “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);
```

[Report a bug](#)

[1] 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.

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

CHAPTER 7. CONFIGURING AGENTS

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

[Report a bug](#)

7.1. 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*.

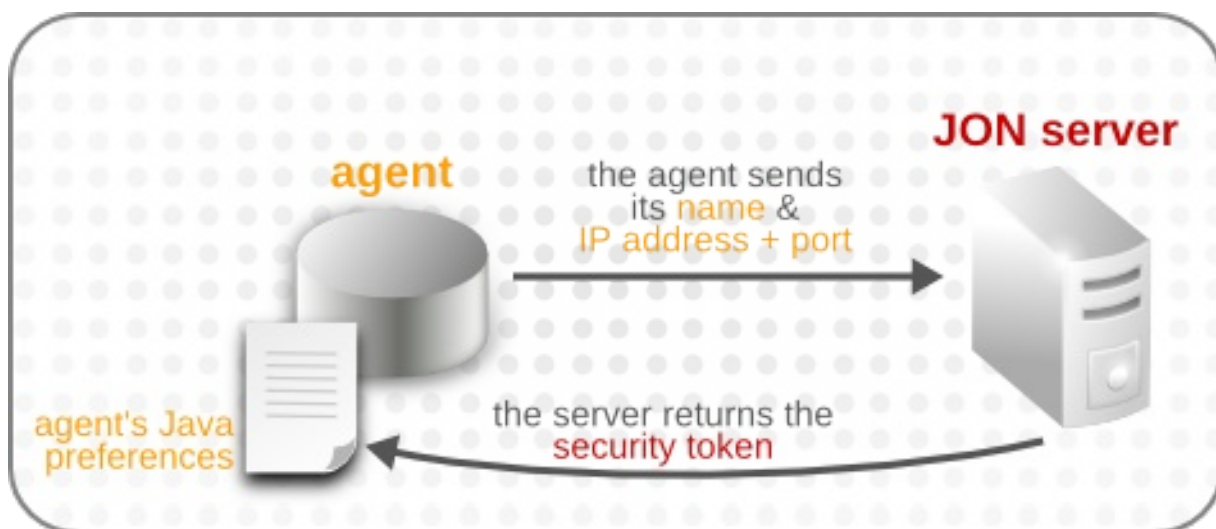
[Report a bug](#)

7.1.1. 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.

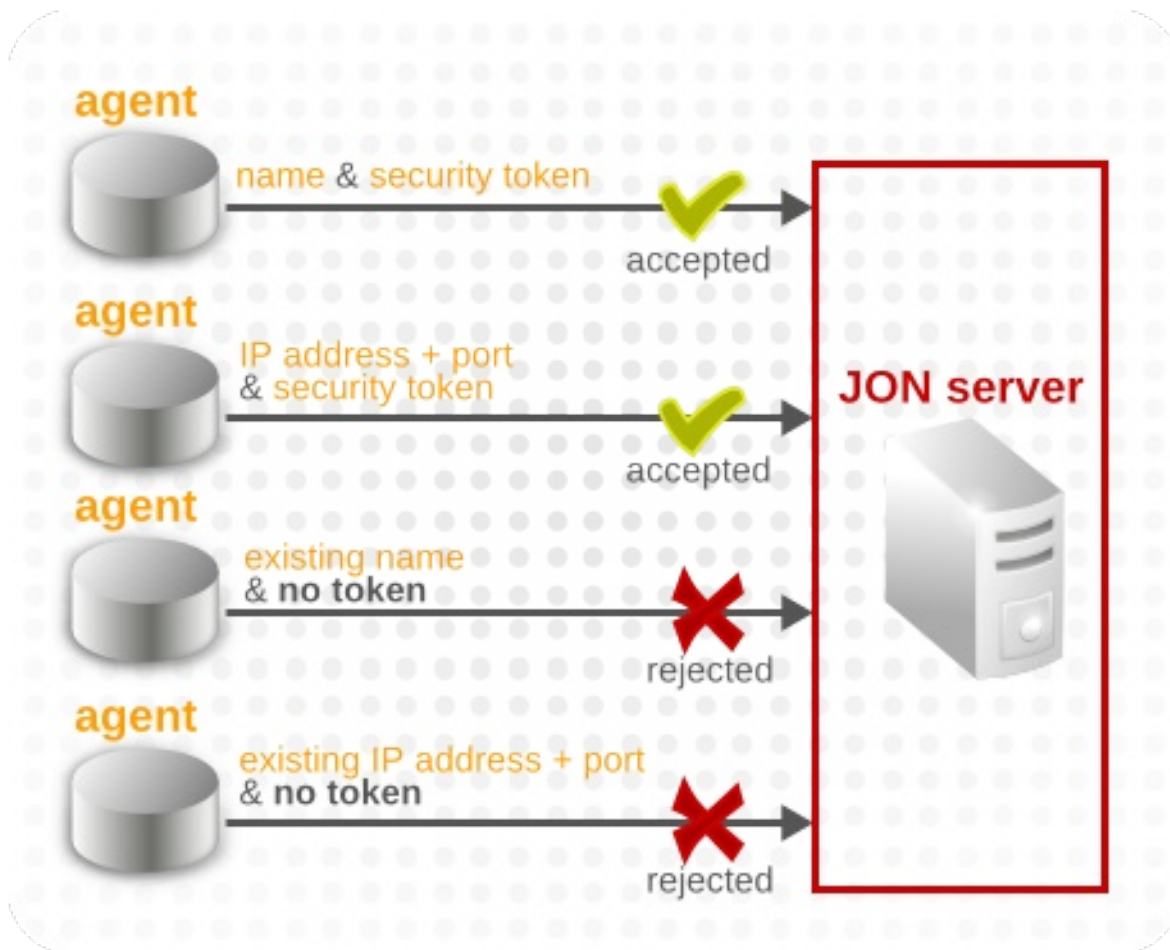
Figure 7.1. Agent Registration



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 7.2. 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 7.1.2, "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.

[Report a bug](#)

7.1.2. Re-installing a Lost Security Token

If a security token is accidentally deleted from the agent's configuration, then the agent can no longer communicate with the server. Any attempt fails with a *failure to authenticate* error.

A lost security token can be re-added to the agent's configuration manually.

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.

Agents

← Back to List

Details

Name : localhost1
 Agent Bind Address : 10.0.0.1
 Remote Endpoint : socket://10.0.0.1:16163/?rhq.communications.connector.rhqtype=agent&numAcceptThreads=1&maxPoolSize=303&clientMaxPoolSize=304&socketTim
 Agent Bind Port : 16163
 Token : ISxZBHeCrMIYPkNEAPkRyDQ9egr7bXwQ3wTrvuJ3hvQArZoVyHpwEh+LyAdDWksfLo=
 Last Availability Report : October 1, 2014 12:28:17 PM UTC-5
 Last Availability Ping : October 1, 2014 2:43:18 PM UTC-5
 Affinity Group : South Data Center
 Connected Server : localhost1

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
```

[Report a bug](#)

7.1.3. Reinstalling the Agent with a New Security Token

An agent can be re-installed and re-registered, with completely fresh configuration. There are three points of configuration for the agent: the agent's (local) persisted configuration, the agent inventory (and associated resource data), and the platform entry in the server inventory. Both the configuration on the

local machine and the agent and resource configuration on the JBoss ON server need to be cleared for the agent to be reinstalled successfully:

- The agent's persisted Java configuration should be purged.
- The agent's inventory should be purged, along with any resource history and configuration.
- The agent must be removed from the JBoss ON inventory. This can be done by removing the agent from the JBoss ON topology ([Section 7.2, "Removing an Agent"](#)) or by deleting the platform entry.
- The agent's original identifying information (name, IP address, and port) can be changed.

To reinstall the agent:

1. Make sure that the original agent instance is properly removed.
 - a. Stop the agent process.
 - b. Remove the platform entry from the JBoss ON server inventory.
2. Restart the agent with the `--fullcleanconfig` option. This registers the agent with a new security token and fresh configuration settings.

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



NOTE

If the agent was not removed from the JBoss ON inventory, then the re-installation fails with an error that the agent has an invalid security token.

[Report a bug](#)

7.1.4. Cleaning the Agent Configuration, with the Original Security Token

An alternative re-registration path cleans the agent configuration *except for* its security token. The agent uses that existing security token to register with the server, so it essentially refreshes its registration instead of re-registering.

In this case, almost all of the original agent configuration is preserved:

- The agent's persisted Java configuration is purged.
- The agent's inventory, along with any resource history and configuration, is saved.
- The agent (via the platform entry) remains in the JON inventory.
- The agent's name must remain the same (though the IP address or port number can be changed).

The main action, then, is that the agent configuration is refreshed, while the agent entry itself is preserved.

To clean the agent configuration, restart the agent with the `--cleanconfig` option. This registers the agent with fresh configuration settings (from the `conf/agent-configuration.xml` file) and reuses its previous security token.

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



NOTE

If the agent name is different, then this re-registration attempt fails because the existing security token cannot be validated with the given (new) agent name.

[Report a bug](#)

7.2. REMOVING AN AGENT

7.2.1. Removing an Agent from a Managed Platform

1. Stop the agent service.
2. Delete the agent from JBoss ON topology.
 - a. In the JBoss ON UI, click the **Administration** tab in the top menu.
 - b. In the **Topology** section in the left menu, select the **Agents** item.
 - c. Select the row for the agent to delete, in the list of installed agents.

The screenshot shows the JBoss ON UI interface. On the left is a navigation menu with sections: Security, Topology (containing Servers, Storage Nodes, Agents, Affinity Groups, Partition Events), Configuration (containing System Settings, Alert Definition Templates, Drift Definition Templates, Metric Collection Templates, Ignored Resource Types, Missing Resource Policy, Downloads, Agent Plugins, Server Plugins), and Content. The 'Agents' item under Topology is selected. The main area displays a table titled 'Agents' with 6 rows. The first row, 'localhost1', is selected. At the bottom of the table, there are buttons for 'New', 'Delete', 'Start', and 'Stop'. The 'Delete' button is circled in red.

Name	Connected Server	Agent Bind Address	Agent Bind Port	Last Availability Report	Last Availability Ping	Affinity Group
localhost1	localhost1	10.0.0.1	16163	Sep 23, 2014 11:34:07 AM	Sep 24, 2014 11:26:35 AM	
localhost2	localhost2	10.0.0.2	16163	Sep 23, 2014 3:17:26 PM	Oct 1, 2014 2:27:48 PM	
localhost3	localhost3	10.0.0.3	16163	Sep 23, 2014 3:30:24 PM	Oct 1, 2014 2:27:51 PM	
localhost4	localhost4	10.0.0.4	16163	Oct 1, 2014 12:28:17 PM	Oct 1, 2014 2:28:17 PM	South Data Center
localhost5	localhost5	10.0.0.5	16163	Sep 30, 2014 11:12:34 PM	Oct 1, 2014 2:27:42 PM	
localhost6	localhost6	10.0.0.6	16163	Oct 1, 2014 2:07:07 PM	Oct 1, 2014 2:27:57 PM	

- d. Click the **Delete** button at the bottom of the page.
 - e. Confirm that the agent should be deleted.
3. If the agent was configured as a system service, remove the associated files. On Windows, this can be done with the **rhq-agent.bat remove** command. On Linux, this means removing the **init.d/** file and updating **chkconfig**.
 4. Remove the agent installation directory.

[Report a bug](#)

7.2.2. Removing an Agent on a JBoss ON Server Machine

1. Stop the agent service.

```
[root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl.sh stop -  
-agent
```

2. Remove the agent.

```
[root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl remove --  
agent
```

3. Delete the agent from JBoss ON topology.
 - a. In the JBoss ON UI, click the **Administration** tab in the top menu.
 - b. In the **Topology** section in the left menu, select the **Agents** item.
 - c. Select the row for the agent to delete, in the list of installed agents.

The screenshot shows the 'Agents' management page. On the left is a navigation menu with categories like Security, Topology, Configuration, and Content. The main area displays a table of agents. The table has the following data:

Name	Connected Server	Agent Bind Address	Agent Bind Port	Last Availability Report	Last Availability Ping	Affinity Group
localhost1	localhost1	10.0.0.1	16163	Sep 23, 2014 11:34:07 AM	Sep 24, 2014 11:26:35 AM	
localhost2	localhost2	10.0.0.2	16163	Sep 23, 2014 3:17:26 PM	Oct 1, 2014 2:27:48 PM	
localhost3	localhost3	10.0.0.3	16163	Sep 23, 2014 3:30:24 PM	Oct 1, 2014 2:27:51 PM	
localhost4	localhost4	10.0.0.4	16163	Oct 1, 2014 12:28:17 PM	Oct 1, 2014 2:28:17 PM	South Data Center
localhost5	localhost5	10.0.0.5	16163	Sep 30, 2014 11:12:34 PM	Oct 1, 2014 2:27:42 PM	
localhost6	localhost6	10.0.0.6	16163	Oct 1, 2014 2:07:07 PM	Oct 1, 2014 2:27:57 PM	

At the bottom of the table, there are buttons for 'New', 'Delete', 'Start', 'Stop', and 'Refresh'. The 'Delete' button is highlighted with a red circle.

- d. Click the **Delete** button at the bottom of the page.
- e. Confirm that the agent should be deleted.

[Report a bug](#)

7.3. 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.

[Report a bug](#)

7.3.1. Opening the Agent Command Prompt

The agent command prompt opens when the agent start script is run.

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

[Report a bug](#)

7.3.2. 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 7.1, “Options for the rhq-agent.sh Script”](#).

Table 7.1. 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= <i>filename</i>	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 <i>name</i> [= <i>value</i>]		Overrides an agent configuration preference and sets a system property.
-e	--console= <i>type</i>	Specifies the implementation to use when reading console input. The three available values are <code>jline</code> , <code>sigar</code> , and <code>java</code> .
-g	--purgeplugins	Deletes all plug-ins, forcing the agent to re-download all of them
-h	--help	Opens the help message.
-i	--input= <i>filename</i>	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, <i>with the exception</i> 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.

Short Argument	Long Argument	Description
-o	--output= <i>filename</i>	Specifies a file to write all output from the script, excluding log messages (which are always written to the agent logs).
-p	--pref= <i>preferences_name</i>	Specifies the Java preference node to use for the agent configuration.
-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.
--		Stops the agent from processing options.

[Report a bug](#)

7.3.3. 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 7.2, “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 7.2. 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.
dumps pool	Shows the entries found in the command spool file.

Prompt Command	Description
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.
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.

Prompt Command	Description
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.

[Report a bug](#)

7.4. INTERACTIONS WITH SYSTEM USERS FOR AGENTS AND RESOURCES

The agent runs as a specific system user, and so do servers such as JBoss and Apache which are managed by JBoss ON. The general assumption with many of the agent management tasks, including discovery, is that the agent user is the same as the resource user. If the users are different, then that can have an impact on how resources can be discovered and managed.

The common types of servers which JBoss ON manages are:

- JBoss EAP servers
- PostgreSQL databases
- Tomcat servers
- Apache servers
- Generic JVMs

For some management operations initiated by the JBoss ON agent, the agent system user is never even involved. For example, the JBoss EAP plug-in connects to the EAP instance using authentication mechanisms managed by JBoss EAP itself, so no system ACLs or user permissions are required. As long as the user can access the JBoss EAP instance, everything works.

Cheat Sheet for Agent and Resource Users

PostgreSQL

No effect for monitoring and discovery.

The agent user must have read/write permissions to the PostgreSQL configuration file for configuration viewing and editing.

Apache

No effect for monitoring and discovery.

The agent user must have read/write permissions to the Apache configuration file for configuration viewing and editing.

Tomcat

Must use the same user or the agent can not be discovered.

JMX server or JVM

Different users are fine when using JMX remoting; cannot be discovered with different users and the attach API

JBoss AS/EAP

Different users are all right, but requires read permissions on run.jar and execute and search permission on all ancestor directories for run.jar

[Report a bug](#)

7.4.1. The Agent User

There is a general assumption that the agent runs as the same user as the managed resources, and this is the cleanest option for configuration.

When the JBoss ON agent is installed from the agent installer JAR file, the system user and group who own the agent installation files is the same user who installs the JAR. So, a special system user can be created or selected, and then the agent can be installed by that user.

[Report a bug](#)

7.4.2. Agent Users and Discovery

An agent discovers a resource by searching for certain common properties, such as PIDs and processes or start scripts.

It does not necessarily matter whether the agent has superior privileges as the resource user.

For most resources, the agent simply requires read access to that resource's configuration. For resources like Apache and Postgres, as long as the agent can read the resource configuration, the resources can be discovered.

For some other resources, the agent user has to have very specific permissions:

- For JBoss EAP resources, the agent must have read permissions to the **run.jar** file, plus execute and search permissions for every directory in the path to the **run.jar** file.
- Tomcat servers can only be discovered if the JBoss ON agent and the Tomcat server are running as the same user. Even if the agent is running as root, the Tomcat server cannot be discovered if it is running as a different user than the agent.
- If a JVM or JMX server is running with JMX remoting, then it can be discovered if the agent is running as a different user. However, if it is running with using the attach API, it has to be running as the same user as the agent for the resource to be discovered.

[Report a bug](#)

7.4.3. Users and Management Tasks

The system user which the agent runs as impacts several common agent tasks:

- Discovery
- Deploying applications
- Executing scripts
- Running start, stop, and restart operations
- Creating child resources through the JBoss ON UI
- Viewing and editing resource configuration

The key thing to determine is what tasks need to be performed and who needs to perform that operation, based on limits on the resource or the operating system for permissions or authorization.

For some actions — discovery, deploying applications, or creating child resources — setting system ACLs that grant the agent user permission are sufficient. This is covered in [Section 7.5, “Running the Agent as a Non-Root User”](#).

For running operations or executing scripts, it may be necessary to run the task as a user other than the agent user. This can be done using **sudo**.

Whatever method, the goal is to grant the JBoss ON user all of the required system permissions necessary to carry out the operations.

[Report a bug](#)

7.4.4. Using sudo with JBoss ON Operations

The time to use **sudo** is for long-running operations, such as starting a service or a process, or for scripts which are owned by a resource user. The user which executes the script should be the same as the resource user because that user already has the proper authorization and permissions.

The user can really be the same, or the JBoss ON user can be granted **sudo** rights to the given command.

When elevating the agent user's permissions, two things must be true:

- There can be no required interaction from the user, including no password prompts.
- It should be possible for the agent to pass variables to the script.

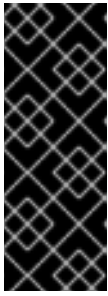
To set up **sudo** for resource scripts:

1. Grant the JBoss ON agent user **sudo** rights to the specific script or command. For example, to run a script as the **jbosadmin** user:

```
[root@server ~]# visudo
jbosson-agent    hostname=(jbosadmin)  NOPASSWD: /opt/jboss-
```

```
eap/jboss-as/bin/*myScript*.sh
```

Using the **NOPASSWD** option runs the command without prompting for a password.

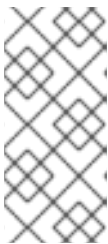


IMPORTANT

JBoss ON passes command-line arguments with the start script when it starts an EAP instance. This can be done either by including the full command-line script (including arguments) in the **sudoers** entry or by using the **sudo -u user** command in a wrapper script or a script prefix.

The second option has a simpler **sudoers** entry

2. Create or edit a wrapper script to use. Instead of invoking the resource's script directly, invoke the wrapper script which uses **sudo** to run the script.



NOTE

For the EAP start script, it is possible to set a script prefix in the connection settings, instead of creating a separate wrapper script:

```
/usr/bin/sudo -u jbosson-agent
```

For example, for a start script wrapper, **start-myScript.sh**:

```
#!/bin/sh
# start-myScript.sh
# Helper script to execute start-myConfig.sh as the user jbosson-agent
#
sudo -u jbosson-agent /opt/jboss-eap/jboss-as/bin/start-myConfig.sh
```

3. Create the start script, with any arguments or settings to pass with the **run.sh** script. For example, for **start-myConfig.sh**:

```
nohup ./run.sh -c MyConfig -b jonagent-host 2>&1> jboss-MyConfig.out &
```

[Report a bug](#)

7.5. 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 jbosson-agent
```

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:jbosson-agent:rw $PGDATA/postgresql.conf
```

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

```
# setfacl -m u:jbosson-agent:x $PGDATA
```

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

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



[Report a bug](#)

7.6. 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.

[Report a bug](#)

7.6.1. 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
```

[Report a bug](#)

7.6.2. 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 the [Apache log4j documentation](#).

[Report a bug](#)

7.6.3. Using the Agent debug Prompt Command

Debug logging can be enabled using the **debug** command in the agent command prompt ([Section 7.3.1](#), “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$ConditionObject.await
Nanos(AbstractQueuedSynchronizer.java:2081)
    at java.util.concurrent.DelayQueue.take(DelayQueue.java:193)
    at
java.util.concurrent.ScheduledThreadPoolExecutor$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)
    ...

```

[Report a bug](#)

7.7. 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
```

- Restart the agent process to load the new configuration.

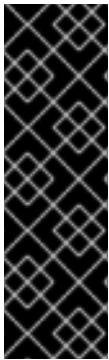
```
agentRoot/rhq-agent/bin/rhq-agent-wrapper.sh stop
```

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

[Report a bug](#)

7.8. 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 [Agent Child Resources](#).

Agent Child Resources

The agent itself



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.

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.

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.

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.

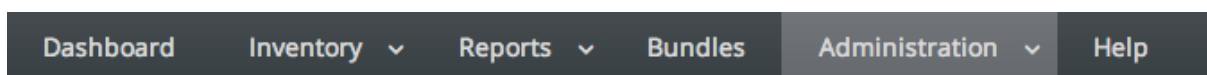
[Report a bug](#)

7.9. CONFIGURING THE AGENT QUIET TIME (TIMEOUT PERIOD)

The JBoss ON server waits a certain amount of time to hear from the agent before it considers that the agent is down. With the default settings, the agent sends a heartbeat to the server every minute. If the agent is quiet for five minutes, then the server reckons that the agent is down and marks the platform and all children as unavailable.

This setting, the agent *quiet time*, is a JBoss ON-wide setting. Every JBoss ON server in the cloud uses the same core settings.

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.

Server Details

Product Name : JBoss ON
 Version : 3.3.0
 Build Number : 0340db5:17b2d55
 Server Name : localhost1
 Server Time Zone : Eastern Standard Time
 Server Local Time : October 1, 2014 3:23:55 PM EDT
 Server Installation Directory : /home/hudson/jon-server-3.3.0
 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 : 8.4.11
 Database Driver Name : PostgreSQL Native Driver
 Database Driver Version : PostgreSQL 9.2 JDBC4 (build 1002)

Jump to Section

▼ **General Configuration Properties**

Property	Unset?	Value	Description
Agent Max Quiet Time Allowed		<input type="text" value="5"/>	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		<input type="text" value="http://10.0.0.1:7080/"/>	A URL to the server GUI, used mainly within alert email notifications.
Enable Agent Auto-Updates		<input checked="" type="radio"/> Yes <input type="radio"/> 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		<input type="radio"/> Yes <input checked="" type="radio"/> No	If enabled, the server will enter debug mode.
Enable Experimental Features		<input type="radio"/> Yes <input checked="" type="radio"/> No	If enabled, any experimental features that exist in the current product will be available.
Enable Login Without Roles		<input type="radio"/> Yes <input checked="" type="radio"/> No	If enabled, the coregui will block users that are not assigned to any role. This applies for both LDAP and normal users.
Default Remote SSH Password	✓	<input type="text"/>	The default SSH password to be used when logging into a remote machine for doing things like controlling or installing remote agents.
Default Remote SSH Username	✓	<input type="text"/>	The default SSH username to be used when logging into a remote machine for doing things like controlling or installing remote agents.

4. Change the **Agent Max Quiet Time Allowed** to the desired interval for the server to wait for the agent heartbeat before marking the agent as down.
5. Click the **Save** button. The changes are applied to all servers immediately.

[Report a bug](#)

7.10. CONFIGURING AGENT UPDATE SETTINGS

When an agent is installed from a JAR file, it has a configuration property that allows the agent to receive a version update automatically from the server. This means that every time the server is updated, all of the agents managed by that server will automatically be updated to the same version as the server. (This is beneficial because the server and the agent must be running as the same JBoss ON version.)

For a single agent, this is configured in the agent configuration file:

```
<entry key="rhq.agent.agent-update.enabled" value="true" />
```

A value of true means that the agent is allowed to receive updates from the server.



IMPORTANT

This value is set to false for all agents installed by RPM. **If an agent is installed from an RPM, the agent update setting must always be false.**

The agent update setting for a single agent can be reset by editing the configuration property.

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` command with the new value for `rhq.agent.agent-update.enabled` configuration preference.

```
> setconfig rhq.agent.agent-update.enabled=false
```

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
```

If all agents are installed using an RPM or if there is some environmental reason to prevent automatic upgrades for all agents, then automatic upgrades can be disabled in the JBoss ON server cloud. This means that no JBoss ON server will make updated packages available to the agents, regardless of the agent setting.

To change the agent update server setting:

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.

Server Details

Product Name : JBoss ON
 Version : 3.3.0
 Build Number : 0340db5:17b2d55
 Server Name : localhost1
 Server Time Zone : Eastern Standard Time
 Server Local Time : October 1, 2014 3:23:55 PM EDT
 Server Installation Directory : /home/hudson/jon-server-3.3.0
 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 : 8.4.11
 Database Driver Name : PostgreSQL Native Driver
 Database Driver Version : PostgreSQL 9.2 JDBC4 (build 1002)

Jump to Section

▼ **General Configuration Properties**

Property	Unset?	Value	Description
Agent Max Quiet Time Allowed		<input type="text" value="5"/>	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		<input type="text" value="http://10.0.0.1:7080/"/>	A URL to the server GUI, used mainly within alert email notifications.
Enable Agent Auto-Updates		<input checked="" type="radio"/> Yes <input type="radio"/> 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		<input type="radio"/> Yes <input checked="" type="radio"/> No	If enabled, the server will enter debug mode.
Enable Experimental Features		<input type="radio"/> Yes <input checked="" type="radio"/> No	If enabled, any experimental features that exist in the current product will be available.
Enable Login Without Roles		<input type="radio"/> Yes <input checked="" type="radio"/> No	If enabled, the coregui will block users that are not assigned to any role. This applies for both LDAP and normal users.
Default Remote SSH Password	✓	<input type="text"/>	The default SSH password to be used when logging into a remote machine for doing things like controlling or installing remote agents.
Default Remote SSH Username	✓	<input type="text"/>	The default SSH username to be used when logging into a remote machine for doing things like controlling or installing remote agents.

4. Set the **Enable Agent Auto-Updates** radio button to **No**. This prevents the server from sending new binaries to installed agents.
5. Click the **Save** button. The changes are applied to all servers immediately.

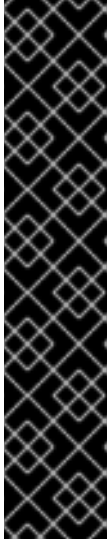
[Report a bug](#)

7.11. 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.

[Report a bug](#)

7.11.1. 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.

> RHQ Agent (RHQ Agent) ✔ ☆

Summary Inventory Alerts Monitoring Events Operations **Configuration**

Current History

Jump to Section

- > **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
- > **Communication Security** - Settings that secure the communications between the RHQ Agent and RHQ Server
- > **Auto Detection** - Configures auto-detection mechanisms that allow the RHQ Agent to find the RHQ Server and vice versa
- > **Client Sender** - Configures the client sender which is responsible for sending messages to the RHQ Server
- > **Startup** - Configures what the agent does while it is starting up
- > **Agent Auto-Update**
- > **Miscellaneous Settings** - Settings that usually do not need to change under normal circumstances

- 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
...
```

- 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.

[Report a bug](#)

7.11.2. 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.

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

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
```

[Report a bug](#)

7.11.3. Overriding Persisted Configuration Settings

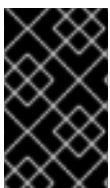
The settings in the Java backing store and in the **agent-configuration.xml** file for the agent can be overridden using the **-D** option, the configuration parameter name, and the new value when the agent is started.

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
```

[Report a bug](#)

7.11.4. Protecting sensitive information in the Agent Configuration



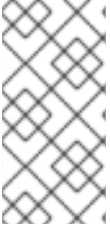
IMPORTANT

By default all the passwords in the **{RHQ_AGENT_HOME}/conf/agent-configuration.xml** file are protected during upgrades and installations. This process is only needed when users want to encode additional properties.

JBoss ON supports protecting almost all properties in the **{RHQ_AGENT_HOME}/conf/agent-configuration.xml** file. JBoss ON provides the **rhq-encode-value.sh** and **rhq-encode-value.bat** scripts in the **{RHQ_SERVER_HOME}/bin/** directory to obfuscate properties. For **agent-configuration.xml**, property values are encoded using the **RESTRICTED::** format and all password are protected by default at installation.

[Report a bug](#)

7.11.4.1. Using the rhq-encode-value Script for Encoding



NOTE

While the **rhq-encode-value** script can be used for encoding server and agent configuration files, JBoss ON only provides a **rhq-encode-value** script at the server level. Users who wish to encode values for the **agent-configuration.xml** file are required to invoke the **rhq-encode-value** script at the JBoss ON server level.

Invoking the **rhq-encode-value** script will prompt users for the desired property and value to encode:

```
> ./rhq-encode-value.sh
Property rhq.autoinstall.server.admin.password [y/n]: n
Property rhq.server.database.password [y/n]: n
Property: rhq.protect.property
Value: 1234
*** !!! WARNING !!!
*** Both standalone-full.xml and rhq-server.properties need to be updated
if a property from rhq-server.properties is used in standalone-full.xml
*** !!! WARNING !!!
***
***
*** Encoded password for rhq-server.properties:
***   rhq.protect.property=RESTRICTED::-299a94df3b478ca8
***
*** Encoded password for standalone-full.xml with vault with password as
default value:
***   ${VAULT::restricted::rhq.protect.property::-299a94df3b478ca8}
***
*** Encoded password for standalone-full.xml with vault without default:
***   ${VAULT::restricted::rhq.protect.property:: }
***
*** Encoded password for agent-configuration.xml:
***   <entry key="rhq.protect.property" value="RESTRICTED::-
299a94df3b478ca8" />
***
*** Please consult the documentation for additional help.
```

After running the script, the values generated from **rhq-encode-value** should be copied and pasted into **aagent-configuration.xml**.

[Report a bug](#)

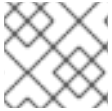
7.12. MANAGING THE AGENT JVM

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.

[Report a bug](#)

7.13. INSTALLING MULTIPLE AGENTS WITH A SHARED DIRECTORY OR ACCOUNT

Multiple agents, running on multiple systems, can share the same system user accounts. If the same user is used for a JBoss ON agent on different systems *and* those system users all use the same shared home directory, then they all share the same agent configuration location and preference node by default. Because of the way the agent uses Java preferences, this requires special agent configuration to prevent the agents from overwriting each other's preferences.

A similar situation can occur on Windows systems if the same domain user is used for the JBoss ON agent. In that case, the Java preferences are stored in a registry key which is used by the domain user and is loaded into the local user's profile. If there are multiple agents using the same domain user, then they will overwrite each other's registry keys.

All of the agent configuration, after setup, is stored in a Java preferences node, as described in [Section 7.11, "Managing the Agent's Persisted Configuration"](#). With the default configuration, the node name is **default**, and the node location is *agentUserHomeDir/.java/.userPrefs/rhq-agent/default*.

If multiple agents are installed using the same file share, then all of them attempt to use the same default node and location.

When multiple agents attempt to use the same Java preferences node, each new agent overwrites the previous agent's configuration as it is set up. This means that only the newest agent's configuration is saved, so only the newest agent can be started. Starting any of the previous agents fails because they cannot find their own configuration.

The preferences node is uniquely identified by two settings:

- Its name, which is defined as an agent configuration setting
- Its location, which is itself a Java option

To run multiple agents with the same home directory, the preferences node has to be uniquely identified for each agent. There are a couple of different ways to do that:

- Editing the agent configuration files directly
- Setting an explicit Java option

[Report a bug](#)

7.13.1. Editing the Configuration Files

When the agent is first set up, the name of the agent preferences node is set in the **agent - configuration.xml** file and is loaded from there. The node location is derived from the node name setting.

1. Edit the **agent - configuration.xml** file to use the new node name:

```
[rhquser@server ~]$ vim agentRoot/rhq-agent/conf/agent-configuration.xml

<node name="agent01-node">
```

2. Then, start the agent with the **--config** option to load the edited configuration file and the **--prefs** option to point to the specific node location:

```
[rhquser@server ~]$ agentRoot/rhq-agent/bin/rhq-agent.sh --prefs=agent01-node --config=agent-configuration.xml
```



IMPORTANT

If the custom Java preferences node is specified by editing the **agent - configuration.xml** file, then every time the agent restarts, the node location has to be passed to the agent using the **--prefs** option.

[Report a bug](#)

7.13.2. Setting a Java Option

Editing the **agent - configuration.xml** file only sets the node name; the node location still has to be passed every time the agent is started.

By setting a Java option in the **rhq-agent-env.sh** file, the Java preferences node information is set once and then persisted, so you can restart the agent as a service, without having to pass **--prefs** options or edit and reload the configuration.

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.

```
[rhquser@server ~]$ agentRoot/rhq-agent/bin/rhq-agent.sh -n
```

2. Use the **setconfig** command to set the **RHQ_AGENT_ADDITIONAL_JAVA_OPTS** value with the preference node. For example:

```
> setconfig RHQ_AGENT_ADDITIONAL_JAVA_OPTS="-Djava.util.prefs.userRoot=agentUserHomeDir/.java/.userPrefs/rhq-agent/agent01-node"
```

The preference node can be in the user preferences directory with a different name, such as **agent01-node**, or it can be in an entirely different location, such as **/etc/agent - preferences**, which is not a shared or filesystem-mounted location.

3. Restart the agent process to load the new configuration. For example, if the agent is running as a service:

```
[rhquser@server ~]$ service rhq-agent-wrapper.sh stop
[rhquser@server ~]$ service rhq-agent-wrapper.sh start
```



NOTE

It is also possible to stop the agent, edit the `rhq-agent-env.sh` file directly, and then restart the agent.



IMPORTANT

Do not set the Java option within the `rhq-agent` runtime directory because this file is overwritten during JBoss ON agent updates. The default location for the `rhq-agent` runtime is `$USERHOME/.java/.userPrefs/rhq-agent/default`.

[Report a bug](#)

7.14. 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 several different 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).
- The scan interval for low-level child services, set in the `rhq.agent.plugins.child-discovery.delay-secs`. The default is five (5) seconds.

Immediate children are discovered as soon as the parent resource is discovered. However, lower-level children are discovered on a subsequent discovery scan. This sets a delay between the initial import and the next discovery scan.

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

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. Use the `setconfig` command to reset the discovery scan intervals. The preference name is whatever the entry name is in the `agent-configuration.xml` file. For example:

```
> setconfig rhq.agent.plugins.server-discovery.period-secs=600
> setconfig rhq.agent.plugins.service-discovery.period-secs=1440
> setconfig rhq.agent.plugins.child-discovery.delay-secs=60
```

- Restart the agent process to load the new configuration. For example, if the agent is running as a service:

```
[root@server ~]# service rhq-agent-wrapper.sh stop
[root@server ~]# service rhq-agent-wrapper.sh start
```

[Report a bug](#)

7.15. 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 5.2, “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:

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

- The agent high availability page shows information about the agents, including items that are relevant for high availability:

The screenshot displays the 'Agents' page in the JBoss ON console. The left sidebar shows a navigation menu with categories like Security, Topology, Configuration, and Content. The 'Agents' section is selected under Topology. The main area shows a table with 6 rows of agent data. The table has columns for Name, Connected Server, Agent Bind Address, Agent Bind Port, Last Availability Report, Last Availability Ping, and Affinity Group. The first row is highlighted in blue, indicating it is selected. Below the table are buttons for 'New', 'Delete', 'Start', 'Stop', and 'Refresh'.

Name	Connected Server	Agent Bind Address	Agent Bind Port	Last Availability Report	Last Availability Ping	Affinity Group
localhost1	localhost1	10.0.0.1	16163	Sep 23, 2014 11:34:07 AM	Sep 24, 2014 11:26:35 AM	
localhost2	localhost2	10.0.0.2	16163	Sep 23, 2014 3:17:26 PM	Oct 1, 2014 2:27:48 PM	
localhost3	localhost3	10.0.0.3	16163	Sep 23, 2014 3:30:24 PM	Oct 1, 2014 2:27:51 PM	
localhost4	localhost4	10.0.0.4	16163	Oct 1, 2014 12:28:17 PM	Oct 1, 2014 2:28:17 PM	South Data Center
localhost5	localhost5	10.0.0.5	16163	Sep 30, 2014 11:12:34 PM	Oct 1, 2014 2:27:42 PM	
localhost6	localhost6	10.0.0.6	16163	Oct 1, 2014 2:07:07 PM	Oct 1, 2014 2:27:57 PM	

- 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.
- 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.

Agents

« Back to List

Details

Name : localhost1
 Agent Bind Address : 10.0.0.1
 Remote Endpoint : socket://10.0.0.1:16163/?rhq.communications.connector.rhqtype=agent&numAcceptThreads=1&maxPoolSize=303&clientMaxPoolSize=304&socketTim
 Agent Bind Port : 16163
 Token : ISxZBHeCrMIYPkNEAPkRyDQ9egr7bXwQ3wTrvuj3hvQArZoVyHpwEh+LyAdDWksfLo=
 Last Availability Report : October 1, 2014 12:28:17 PM UTC-5
 Last Availability Ping : October 1, 2014 2:43:18 PM UTC-5
 Affinity Group : South Data Center
 Connected Server : localhost1

Agent Failover List

Total Rows: 1 (selected: 0)

Order	Name	Mode	Endpoint Address	Nonsecure Por	Secure Port	Last Update Time	Affinity Group
0	localhost1	NORMAL	localhost1	7080	7443	Oct 1, 2014 2:43:35 PM	

[Report a bug](#)

7.16. 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.

[Report a bug](#)

7.16.1. 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
```

[Report a bug](#)

7.16.2. 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
```

[Report a bug](#)

7.17. 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 7.3, “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 than 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 7.3. 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.

Parameter	Description
rhq.agent.client.send-throttling	<p>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.</p> <p>This parameter sets both the number of commands and the quiet period, in the form <i>commands:timeout_milliseconds</i>. For example, 50 : 10000 sets a limit of 50 commands and a quiet period of 10000 milliseconds.</p>
rhq.agent.client.queue-throttling	<p>Limits the amount of commands that can be dequeued in a given amount of time; this is the <i>burst period</i>. 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.</p> <p>As with send throttling, this parameter sets both the number of commands and the quiet period, in the form <i>commands:timeout_milliseconds</i>. For example, 50 : 10000 sets a limit of 50 commands and a quiet period of 10000 milliseconds.</p> <p>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.</p> <p>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.</p>

[Report a bug](#)

7.18. 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 pool 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
```

[Report a bug](#)

7.19. 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&maxPool
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://docs.jboss.org/jbossremoting/2.5.4.SP4/guide/html/chapter-configuration.html>.

[Report a bug](#)

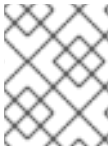
CHAPTER 8. MANAGING DATABASES ASSOCIATED WITH JBOSS ON

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

[Report a bug](#)

8.1. 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 **coregui/#Test/ServerAccess/SQL**. For example:

```
http://server.example.com:7080/coregui/#Test/ServerAccess/SQL
```

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.

[Report a bug](#)

8.2. 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 **rhq-encode-password.sh** script to encode the password.

```
serverRoot/bin/rhq-encode-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
```

[Report a bug](#)

8.3. 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 8.1. 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 8.1. rhq-server.properties Parameters for Database Configuration

Parameter	Description
<code>rhq.server.database.type-mapping</code>	Gives the type or vendor of the database that is used by the JBoss ON server. This is either PostgreSQL or Oracle10g (Oracle10g is used for Oracle database versions 10, 11, and 12).
<code>rhq.server.database.connection-url</code>	The JDBC URL that the JBoss ON server uses when connecting to the database, such as <code>jdbc:postgresql://localhost:5432/rhq</code> or <code>jdbc:oracle:oci:@localhost:1521:orcl</code> .

Parameter	Description
<code>rhq.server.database.driver-class</code>	The fully qualified class name of the JDBC driver that the JBoss ON server uses to communicate with the database, such as <i>oracle.jdbc.driver.OracleDriver</i> .
<code>rhq.server.database.xa-datasource-class</code>	The fully qualified class name of the JDBC driver that the JBoss ON server uses to communicate with the database, such as <i>org.postgresql.xa.PGXADatasource</i> or <i>oracle.jdbc.xa.client.OracleXADatasource</i> .
<code>rhq.server.database.user-name</code>	The name of the user that the JBoss ON server uses when logging into the database.
<code>rhq.server.database.password</code>	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 8.2, “Changing Database Passwords” .
<code>rhq.server.database.server-name</code>	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.
<code>rhq.server.database.port</code>	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.
<code>rhq.server.database.db-name</code>	The name of the database. This must match the name found in the connection URL. This is currently only used when connecting to PostgreSQL.
<code>rhq.server.quartz.driverDelegateClass</code>	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 .

[Report a bug](#)

CHAPTER 9. DEPLOYING AND MANAGING STORAGE NODES

Raw metric data and aggregated metric data are stored in a dedicated, scalable distributed database. Much like the JBoss ON server, there can be multiple storage nodes in a cluster (installed independently of the server). Nodes can be added and dropped from the cloud easily, allowing the metrics storage to be dynamically expanded according to the needs of the environment.

[Report a bug](#)

9.1. ABOUT HIGH-SPEED METRICS STORAGE

Collecting metrics and generating alerts can be resource-intensive. It results in near-constant write operations on the backend database. That creates a natural threshold for the number of metrics that can be collected (30,000 per minute) before encountering performance degradation.



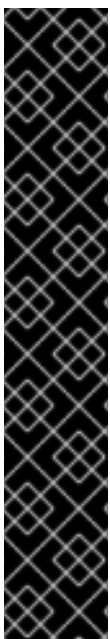
NOTE

The natural threshold of 30,000 metrics per minute is based on internal performance testing. This threshold varies depending on system resources and overall load.

JBoss ON uses two databases to store its information. One is a central relational database (PostgreSQL or Oracle) which stores all configuration about the JBoss ON servers and agents, all resource inventory data, resource configuration, and other data. The other database is a distributed database (a cluster of storage nodes) which stores all numeric monitoring data — in other words, all collected metrics.

The distributed database can be expanded, with multiple nodes in a cluster. This ability to add additional nodes according to load is a crucial management tool for administrators. Rather than encountering that hardware-driven limit of 30,000 metrics collected per minute, additional nodes can be added to improve performance.

At least one storage node must be created and managed by JBoss ON (which minimizes the metrics storage node management overhead since no external tools are required).



IMPORTANT

Regardless of the load or demand on a JBoss ON installation, the JBoss ON server will go into maintenance mode while:

- the storage node processes a backlog of storage requests (during times of peak demand),
- transfers data from memory to disk, or
- performs data aggregation or compaction.

In most instances, the system is designed to recover without interruption. However, as the demand or load on a system increases, the length of these outages will also increase. Once the length of these outages begins to interrupt operations, additional storage nodes should be installed to distribute the load. Additional storage nodes are also required when the storage node has insufficient storage space. For details on installing additional storage nodes, see [Section 9.2.4, “Installing Additional Nodes”](#).

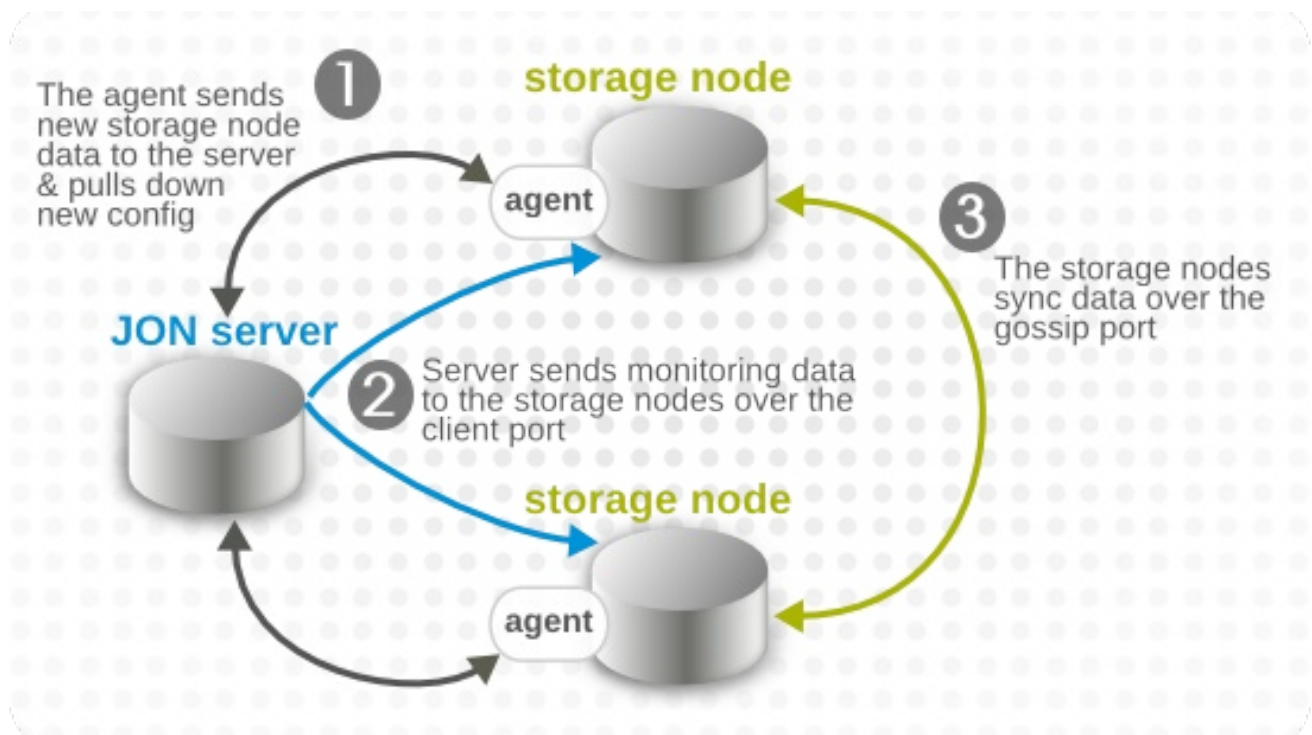
There are several paths of communication to handle metrics data, all working in parallel.

1. The agent sends the storage node configuration to the JBoss ON server. The JBoss ON server then sends that updated storage cluster information to every agent associated with a storage node.

Each companion agent then updates its storage cluster configuration, in the **rhq-storage-auth.conf**, with the hostname or IP address of the new node. (Likewise, when a node is removed, the server sends the information to each of the companion agents, and the agent removes the hostname or IP address from the list in the local storage node's **rhq-storage-auth.conf** file.)

2. The server receives monitoring data from all agents (not just those associated with a storage node), and sends that information to an available storage node to be stored.
3. The storage nodes replicate their monitoring data among each other for high availability and integrity.

Figure 9.1. Server, Agent, and Metrics Storage Node Communication



Node cluster communication requires three elements:

- The hostname or IP address of every storage node, stored in the **rhq-storage-auth.conf**
- A common port number for the JBoss ON server to use to communicate with the storage node (the *client port*)
- A common port number for the other storage nodes in the cluster to use to sync data between each other (the *gossip port*)

The metrics storage provides data availability and integrity by backing up the data in two ways:

- Replicating data between the storage nodes (over the gossip port)
- Taking local snapshots and backing up the data locally

[Report a bug](#)

9.2. DEPLOYING AND UNDEPLOYING STORAGE NODES

There can be multiple metrics storage nodes in the JBoss ON environment. Much like the JBoss ON servers can operate in a high availability cloud, multiple storage nodes are deployed in the same environment and communicate with each other to function in a cluster.

Nodes can be added and removed from the cluster by administrators or dynamically using JBoss ON scripts in response to changes in the environment.



NOTE

A companion agent is always installed with a storage node to enable server-node communication.

With the default cluster configuration, a node is deployed as soon as it is installed and configured in the JBoss ON server. These are actually two separate steps.

1. The bits are installed on a local system and the storage node is registered with the JBoss ON server.
2. The new node information is deployed to the cluster.

Whether to deploy nodes automatically or manually can be determined by the provisioning and monitoring requirements of the environment.



WARNING

Deploying a node lists that node's host in the cluster configuration and *any* allowed host can gain access to the data in the storage cluster.

Restrict access to the **rhq-storage-auth.conf** file so that the allowed hosts list cannot be altered to allow an attacker to gain access to the cluster and the stored data.

[Report a bug](#)

9.2.1. Storage Node Requirements

There are two critical requirements for deploying nodes:

- The hostnames or IP addresses of all storage nodes and the hostname and bind addresses of the JBoss ON server and agents must all be fully-resolvable in DNS. If the IP addresses and hostnames of the storage nodes, servers, or agents are not properly formatted in DNS, then all communication between the different JBoss ON components will fail.
- The firewall must allow communication over the two ports used by the storage nodes. By default, the ports are 9142 and 7100 for the server/client and gossip ports, respectively.

[Report a bug](#)

9.2.2. Installing the Storage Node with the Server

By default, the server installation script also installs the storage node and an agent:

```
[root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl install --start
```

No additional options are required.

[Report a bug](#)

9.2.3. Installing Storage Nodes Before Installing the Server

It is possible to create multiple storage nodes before installing a server, and then install the server with those pre-installed nodes. This is also useful if the storage database will be on a separate, dedicated machine.



WARNING

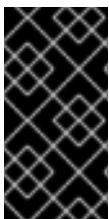
This is an advanced configuration. If the storage node or nodes within the cluster are not properly configured, then the cluster may not properly function.



WARNING

Deploying a node lists that node's host in the cluster configuration and *any* allowed host can gain access to the data in the storage cluster.

Restrict access to the **rhq-storage-auth.conf** file so that the allowed hosts list cannot be altered to allow an attacker to gain access to the cluster and the stored data.



IMPORTANT

Every storage node must use the same client (CQL) and gossip ports.

Additionally, the hostname and IP address of every storage node system must be fully resolvable in DNS or must be configured on each system's **hosts** file.

1. Determine the node and cluster configuration information to use.
 - Identify the hostname or IP address of each system which will host a node.
 - Define the two ports which the cluster uses for communication (9142 and 7100 by default).
2. *Before installing any storage node*, edit the storage properties file with all of the node and cluster information.

```
[root@server ~]# vim serverRoot/jon-server-3.3.0.GA/bin/rhq-storage.properties
```

For example, this configures three nodes, set in the ***rhq.storage.seeds*** parameter.

```
rhq.storage.cql-port=9142
rhq.storage.gossip-port=7100
rhq.storage.seeds=192.68.0.0, 192.68.0.1, 192.68.0.2
start=false
```

3. Install the storage node on each system, with its companion agent. This requires the IP address of the JBoss ON server, *even though the server is not yet installed*.

Do not start the storage node or the agent at this point. Do not use the `--start` option with the installation script.

```
[root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl install -
-storage --agent-preference="rhq.agent.server.bind-
address=192.68.0.2"
```

4. For each storage node, edit its local ***rhq-storage-auth.conf*** file. This lists the hostnames or IP addresses for all of the storage nodes in the cluster, one per line.

```
[root@server ~]# vim serverRoot/jon-server-3.3.0.GA/rhq-
storage/conf/rhq-storage-auth.conf

192.68.0.0
192.68.0.1
192.68.0.2
```

After the server is configured, the local agent will update the ***rhq-storage-auth.conf*** file with node hostnames or IP addresses as nodes are deployed and removed from the cluster.

5. Start each node.

```
[root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl start --
storage
```

6. Before installing the server, edit the ***rhq-server.properties*** file to include the connection information for the storage nodes.

Add each storage node in a comma-separated list to the ***rhq.storage.nodes*** parameter. Then, add the client and gossip port values.

```
[root@server ~]# vim serverRoot/jon-server-3.3.0.GA/bin/rhq-
server.properties

rhq.storage.nodes=192.68.0.0,192.68.0.1,192.68.0.2
rhq.storage.cql-port=9142
rhq.storage.gossip-port=7100
```

7. Install the server and an agent. Specifying the **`--server`** and **`--agent`** options only installs those two components; the storage database is excluded.

```
[root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl install -
-server --agent --start
```

If you are upgrading an existing JBoss ON agent, then run the upgrade script with the **--use-remote-storage-note** option, to load the storage database information from the properties file rather than installing a storage node.

```
[root@server]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl upgrade --
use-remote-storage-node=true
```

[Report a bug](#)

9.2.4. Installing Additional Nodes

When creating a new storage node, two components are always installed: the storage node and a companion agent. The only required information to set up the node is the IP address of the JBoss ON server. The agent registers with the server and then sends the storage node hostname or IP address. The server then distributes that information among the other agents and it gets propagated into the cluster.

If the cluster is using custom client and gossip ports, then edit the **rhq-storage.properties** file with the correct ports before running the installation script.

Run the installation script with the **--agent-preference** option to supply the server bind address. For example:

```
[root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl install --
storage --agent-preference="rhq.agent.server.bind-address=0.0.0.0"
```



NOTE

When installing on Linux, the **rhqctl** command must be run as root. On Windows, the command prompt must be opened with the option **Run as Administrator**.

The deployment operation can take several minutes to complete, as the new node information is propagated among the existing nodes. Until the deploy operation is complete, the node shows a status of *Joining*.

Figure 9.2. Joining the Cluster

Total Rows: 1 (selected: 0)						
Endpoint Address ▾	Alerts	Memory	Disk	Cluster Status	Installation Date	Resource
▶ 10.0.0.1	Unacknowledged Alerts (2)	49.1 %		JOINING	Sep 23, 2014 9:50:08 AM	Link to Resource



WARNING

Deploying a node lists that node's host in the cluster configuration and *any* allowed host can gain access to the data in the storage cluster.

Restrict access to the `rhq-storage-auth.conf` file so that the allowed hosts list cannot be altered to allow an attacker to gain access to the cluster and the stored data.

[Report a bug](#)

9.2.5. Setting Automatic Deployment for Nodes

When a node is created with the proper port and cluster settings, it can either join the cluster immediately or it can wait, essentially on standby, and then be deployed manually later. This is determined by a cluster-wide setting.

This can be useful if machines are being provisioned and taken offline repeatedly (such as in virtual environments) or if nodes should only be online during certain periods.

1. Click the **Administration** tab in the top navigation bar.
2. In the **Topology** area on the left, select the **Storage Nodes** item.

Endpoint Address	Alerts	Memory	Disk
10.0.0.1	Unacknowledged Alerts (0)	49.1 %	Sufficient

3. Open the **Cluster Settings** tab.
4. The **Automatic Deployment** option sets whether to deploy a storage node to the cluster as soon as it is installed in JBoss ON or to wait to deploy it manually.

The default is to deploy nodes automatically. This can be changed if the environment has other provisioning rules in place.

New Deployment Settings
Only applies to new installations.

Property	Value	Description
Automatic Deployment	<input checked="" type="radio"/> On <input type="radio"/> Off	If this is set, the newly installed Storage Nodes will be automatically deployed to the Storage Cluster. It only applies to new installations.

5. Click **Save** at the bottom of the page.

[Report a bug](#)

9.2.6. Deploying Nodes Manually

By default, when a new storage node is installed, it is automatically deployed to the cluster. However, if automatic deployment is disabled, then new nodes must be deployed manually after they are created.



WARNING

Deploying a node lists that node's host in the cluster configuration and *any* allowed host can gain access to the data in the storage cluster.

Restrict access to the `rhq-storage-auth.conf` file so that the allowed hosts list cannot be altered to allow an attacker to gain access to the cluster and the stored data.

1. Install a node using the `rhqctl install` command.
2. Click the **Administration** tab in the top navigation bar.
3. In the **Topology** area on the left, select the **Storage Nodes** item.

Endpoint Address	Alerts	Memory	Disk
10.0.0.1	Unacknowledged Alerts (0)	49.1 %	Sufficient

4. In the **Nodes** tab, select the row of the node to deploy.

Endpoint Address	Alerts	Memory	Disk	Cluster Status	Installation Date	Resource	Availability
10.0.0.1	Unacknowledged Alerts (0)	49.2 %	Sufficient	NORMAL	Sep 23, 2014 9:50:08 AM	Link to Resource	🟢

Operation:

5. Click the **Deploy** button.

A node can also be deployed by running a **Deploy** operation on the storage node resource.

NOTE

Storage nodes can be deployed using the JBoss ON CLI and scripts, as well. This can be useful when provisioning new systems in the infrastructure.

For example:

```
// deploy a storage node
nodes =
StorageNodeManager.findStorageNodesByCriteria(StorageNodeCriteria());
node = nodes.get(0);
StorageNodeManager.deployStorageNode(node);
```

This can be key in dynamically adding storage nodes as demands on the infrastructure increase. Nodes can be installed in advance and then hot-deployed and removed as necessary.

[Report a bug](#)

9.2.7. Removing Nodes

Undeploying a storage node removes it from the cluster and then removes the resource from the inventory and uninstalls the storage node bits from the machine.

1. Click the **Administration** tab in the top navigation bar.
2. In the **Topology** area on the left, select the **Storage Nodes** item.

Endpoint Address	Alerts	Memory	Disk
10.0.0.1	Unacknowledged Alerts (0)	49.1 %	Sufficient

3. In the **Nodes** tab, select the row of the node to remove. To select multiple rows, hold the **Ctrl** key and click the desired rows.

Endpoint Address	Alerts	Memory	Disk	Cluster Status	Installation Date	Resource	Availability
10.0.0.1	Unacknowledged Alerts (0)	49.2 %	Sufficient	NORMAL	Sep 23, 2014 9:50:08 AM	Link to Resource	🟢

Operation:

4. Click the **Undeploy Selected** button, and confirm the operation.

A node can also be removed by running an **Undeploy** operation on the storage node resource.



NOTE

Storage nodes can be removed using the JBoss ON CLI and scripts, as well. This can be useful when provisioning and removing systems from the infrastructure.

For example:

```
// undeploy a storage node
nodes =
StorageNodeManager.findStorageNodesByCriteria(StorageNodeCriteria());
node = nodes.get(0);
StorageNodeManager.undeployStorageNode(node);
```

[Report a bug](#)

9.3. VIEWING STORAGE NODE METRICS AND STATES

The storage node has a number of child resources for services related to memory, cache operations, data operations, logging, client requests, and others. Metrics which are critical for node performance, then, are spread across multiple child resources, especially the JVM memory service and the database management storage service.

The **Nodes** tab provides a summary of the most critical metrics collected across all node services, so it provides a good snapshot of the node performance. It also provides additional information of the node *within the cluster*, so it can indicate communication issues with a node that would not appear in service metrics.

To view the node metrics and states:

1. Click the **Administration** tab in the top navigation bar.
2. In the **Topology** area on the left, select the **Storage Nodes** item.

Endpoint Address	Alerts	Memory	Disk
▶ 10.0.0.1	Unacknowledged Alerts (0)	49.1 %	Sufficient

3. The **Nodes** tab shows the number of unacknowledged alerts for each node.

Nodes Cluster Settings Cluster Alerts (2)

Total Rows: 1 (selected: 1)

Endpoint Address	Alerts	Memory	Disk	Cluster Status	Installation Date	Resource	Availability
10.0.0.1	Unacknowledged Alerts (0)	49.2 %	Sufficient	NORMAL	Sep 23, 2014 9:50:08 AM	Link to Resource	

Metric	Minimum	Average	Maximum
Heap Used	118.8 MB	246.5 MB	360.8 MB
Heap Percent Used	23.8 %	49.4 %	72.3 %
Disk Space Used by Storage Node	290.0 MB	291.3 MB	292.9 MB
Total Disk Space Percent Used	67.0 %	67.0 %	67.0 %
Data Disk Space Percent Used	1.0 %	1.0 %	1.0 %
Free Disk To Data Size Ratio	31.9	32.6	33.1
Ownership	100.0 %	100.0 %	100.0 %
Number of Tokens	256	256	256

Clicking on the hostname or IP address of the node opens up a details page with all of the metrics and state data, the current configuration settings, and the ports on which the node is listening.

Figure 9.3. Storage Node Details

Nodes Cluster Settings Cluster Alerts (2)

« Back to List Storage Node (10.0.0.1)

Storage Node Information

Endpoint Address : 10.0.0.1

Associated Resource : RHO Storage Node(10.0.0.1)

Availability :

CQL Port : 9142
 JMX Port : 7299
 Operation Mode : NORMAL
 Cluster Status : NORMAL
 Installation Date : September 23, 2014 9:50:08 AM UTC-5
 Last Update Time : September 23, 2014 9:50:08 AM UTC-5
 Alerts : Unacknowledged Alerts (2)
 Note : Everything is ok.

Status

Chart	Metric	Minimum	Average	Maximum
	Heap Used	118.8 MB	247.7 MB	360.8 MB
	Heap Percent Used	23.8 %	49.6 %	72.3 %
	Disk Space Used by Sto	290.0 MB	291.4 MB	292.9 MB
	Total Disk Space Percer	67.0 %	67.0 %	67.0 %
	Data Disk Space Percen	1.0 %	1.0 %	1.0 %
	Free Disk To Data Size F	31.9	32.6	33.1
	Ownership	100.0 %	100.0 %	100.0 %
	Number of Tokens	256	256	256

Configuration

Storage Node Specific Settings

Property	Value	Description
Heap Size	512 MB	The maximum (and at the same time starting) heap size. This value will be used with the -Xmx and -Xms JVM options. If you are going to increase/decrease this value, then you should also increase/decrease the new generation proportionally. The value should be an integer with a suffix of M or G to indicate megabytes or gigabytes.
Heap New Size	128 MB	The size of the new generation portion of the heap. This value will be used with the -Xmn JVM option. The value should be an integer with a suffix of M or G to indicate megabytes or gigabytes.
Thread Stack Size	256	The thread stack size. This memory is allocated to each thread off heap. The value should be an integer that will be interpreted in kilobytes.
JMX Port	7299	The JMX port for the RHQ Storage Node

Save

[Report a bug](#)

9.4. CHANGING CLUSTER PORTS

The metrics storage node cluster relies on shared information to be able to identify and communicate with cluster nodes. The JBoss ON server uses those configured ports to communicate with the storage nodes and to manage the agent-storage configuration.

The cluster configuration requires that all members are communicating to the server on the same port (the *client port*) and to each other on the same port (the *gossip port*).

The port configuration must be changed in two places: in the JBoss ON configuration (which is used by servers and agents) and in each individual node.

1. Change the port settings used by the server.
 - a. Click the **Administration** tab in the top navigation bar.
 - b. In the **Topology** area on the left, select the **Storage Nodes** item.

Endpoint Address	Alerts	Memory	Disk
10.0.0.1	Unacknowledged Alerts (0)	49.1 %	Sufficient

- c. Open the **Cluster Settings** tab.
- d. Edit the port numbers.
 - The **CQL Port** sets the client port that the JBoss ON server uses to communicate with all storage nodes to send metrics data.
 - The **Gossip Port** sets the port that nodes use to contact each other to replicate metrics data.

Property	Value	Description
CQL Port	9142	Port on which the Storage Nodes listen for CQL client connections. On save this setting will not be propagated to existing Storage Nodes. Please review the documentation on how update the CQL port for all Storage Nodes. Warning: if this setting does not match the configured Storage Cluster CQL port, the server will not be able to communicate with the Storage Cluster and will go into maintenance mode.
Gossip Port	7100	The port used for internode communication in the Storage Cluster. On save this setting will not be propagated to existing Storage Nodes. Please review the documentation on how update the Gossip port for all Storage Nodes. Warning: if this setting does not match the configured Storage Cluster Gossip port, any new Storage Nodes will be able to communicate and be part of the existing Storage Cluster.

- e. Click **Save** at the bottom of the page.
2. Change the port settings for each node.
 - a. On the **Nodes** tab of the storage administration area, click **Link to Resource**. This opens the node's resource page.

Total Rows: 1 (selected: 0)							
Endpoint Addr	Alerts	Memory	Disk	Cluster Status	Installation Date	Resource	Availability
localhost1	Unacknowledged Alerts (0)	49.6 %	Sufficient	NORMAL	Sep 23, 2014 9:50:08 AM	Link to Resource	✔

- b. Open the **Configuration** tab for the storage node.
- c. Edit the port numbers.

The screenshot displays the configuration page for an RHQ Storage Node. The left-hand navigation pane shows a tree structure under 'localhost1', with 'RHQ Storage Node(localhost1)' expanded. The main panel is titled 'RHQ Storage Node(localhost1)' and has tabs for 'Summary', 'Inventory', 'Alerts', 'Monitoring', 'Operations', and 'Configuration'. The 'Configuration' tab is active, showing a 'Current' history view. Below a 'Jump to Section' field, there are two sections: 'Memory Settings' and 'Ports'. The 'Memory Settings' section contains a table with columns 'Property', 'Unset?', 'Value', and 'Description'. It lists 'Min Heap Size' (512M), 'Max Heap Size' (512M), 'Heap New Size' (128M), and 'Thread Stack Size' (256). The 'Ports' section also has a table with columns 'Property', 'Unset?', 'Value', and 'Description', listing 'CQL Port' (9142) and 'Gossip Port' (7100).

- The **CQL Port** sets the client port that the JBoss ON server uses to communicate with all storage nodes to send metrics data.
 - The **Gossip Port** sets the port that nodes use to contact each other to replicate metrics data.
- d. Click **Save** at the bottom of the page.
3. Restart each node.
 4. If the client (CQL) port was changed, then also restart each JBoss ON server.

[Report a bug](#)

9.5. VIEWING STORAGE NODE ALERTS

The storage nodes offer an elastic way of storing metrics information, which allows more metrics to be collected more frequently. Keeping the proper number of storage nodes for the specific infrastructure, number of resources, and number of metrics requires constant awareness of how the current nodes are performing, to show whether additional storage nodes are needed.

There are two primary indications that an additional metrics storage node should be deployed or that metrics collection schedules should be adjusted:

- The JVM heap hits its maximum threshold and causes performance degradation.

- The storage node begins using too much disk space on its system.

The heap size (along with other JVM parameters) are configurable per node and the disk space limits are relative to the system, but for large environments or intensive metrics collection, the storage node could still encounter hardware limits.

The storage node resource does not have configured metrics or alert definitions set on it directly, as with other resources. Rather, there are four alerts pre-defined for every storage node, to help determine when to add nodes through automatic monitoring:

- High heap usage, which can lead to out of memory errors and performance degradation. A dampening rule is in place to prevent alerts for momentary memory spikes.
- High disk usage, which can lead to problems with compaction and other routine operations.

Compaction operations are particularly important because compaction merges datafiles on the disk into a single disk file. This frees disk space and improves read performance. If this operations fails, then performance can degrade.

An alert is fired for high disk usage if any one of several conditions is met:

- The size of the storage node data exceeds 50% of total disk space.
- The *overall* amount of disk space used exceeds 75% of the total disk space (regardless of how much disk space the storage node is using).
- The ratio of free disk space to storage node data is less than 1.5. This is calculated by taking the amount of free disk space divided by the disk space used by the storage node. If there is 50MB of free space, and the storage node is using 35MB of disk, then the ratio is $50/35$ or 1.42. That is too low and would trigger an alert.

A dampening rule is in place to prevent alerts for momentary usage spikes.

- Snapshot failure, meaning a local routine backup operation has failed.
- Maintenance operation failure, meaning either a deploy or undeploy operation for a node failed. Any underlying causes, like an unavailable resource, can be addressed and then the operation can be re-run.

Each of the predefined alerts is set against child resources for the storage node.

Table 9.1. Storage Resources for Alerts

Alert	Parent Resource	Resource Type	Area to Address
High Heap Usage	Cassandra Server JVM	Memory Subsystem	Edit the heap sizes in the storage node JVM configuration
High Disk Usage	Database Management Services	Storage Service	Increase the disk space for the system hosting the node
Snapshot Failure	Database Management Services	Storage Service	

Alert	Parent Resource	Resource Type	Area to Address
Maintenance Operation Failure		Storage Node	Unavailable storage nodes in the cloud (which prevent updates)

To view the alerts for a storage cluster:

1. Click the **Administration** tab in the top navigation bar.
2. In the **Topology** area on the left, select the **Storage Nodes** item.

Endpoint Address	Alerts	Memory	Disk
10.0.0.1	Unacknowledged Alerts (0)	49.1 %	Sufficient

3. The **Nodes** tab shows the number of unacknowledged alerts for each node.

Endpoint Addr	Alerts	Memory	Disk	Cluster Status	Installation Date	Resource	Availability
localhost1	Unacknowledged Alerts (0)	49.6 %	Sufficient	NORMAL	Sep 23, 2014 9:50:08 AM	Link to Resource	✔

4. To view the list of alerts, open the **Cluster Alerts** tab.

Every alert is listed with a description of the condition which triggered it, the affected resource, and the time of the alert.

[Report a bug](#)

9.6. ENABLING DEBUG MODE FOR THE STORAGE NODE

1. On the storage node machine, stop the storage node.

```
[root@node ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl.sh stop -- storage
```

2. Open the storage node's **log4j-server.properties** file.

```
[root@server ~]# vim serverRoot/jon-server-3.3.0.GA/rhq-storage/conf/log4j-server.properties
```

3. Add or edit the lines for the log threshold property (**`log4j.appender.R.Threshold`**) and the apache property (**`log4j.logger.org.apache.cassandra`**) to enable debug logging.

```
log4j.appender.R.Threshold=DEBUG
log4j.logger.org.apache.cassandra=DEBUG
```

4. Restart the storage node.

```
[root@node ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl start --
storage
```

[Report a bug](#)

9.7. MANAGING THE STORAGE NODE HEAP

One of the most common issues to affect storage node performance is running out of heap. This can affect normal database operations, such as garbage collection, which can cause the storage node process to fail with out of memory errors.

By default, storage node JVM is set with a maximum heap size of 512MB. This may not be sufficient for large environments. The heap size can be reset in the node's configuration.

1. Click the **Administration** tab in the top navigation bar.
2. In the **Topology** area on the left, select the **Storage Nodes** item.

Endpoint Address	Alerts	Memory	Disk
10.0.0.1	Unacknowledged Alerts (0)	49.1 %	Sufficient

3. In the **Nodes** tab, click the hostname or IP address of the storage node to edit.
4. In the **Configuration** area, reset the **Max Heap Size** setting.


Nodes Cluster Settings Cluster Alerts (2)

« Back to List Storage Node (10.0.0.1)

Storage Node Information

Endpoint Address : 10.0.0.1

Associated Resource : [RHO Storage Node\(10.0.0.1\)](#)

Availability : 

CQL Port : 9142

JMX Port : 7299

Operation Mode : NORMAL

Cluster Status : NORMAL









Installation Date : September 23, 2014 9:50:08 AM UTC-5

Last Update Time : September 23, 2014 9:50:08 AM UTC-5

Alerts : [Unacknowledged Alerts \(2\)](#)

Note : Everything is ok.

Status

Chart	Metric	Minimum	Average	Maximum
	Heap Used	118.8 MB	247.7 MB	360.8 MB
	Heap Percent Used	23.8 %	49.6 %	72.3 %
	Disk Space Used by Sto	290.0 MB	291.4 MB	292.9 MB
	Total Disk Space Percer	67.0 %	67.0 %	67.0 %
	Data Disk Space Percen	1.0 %	1.0 %	1.0 %
	Free Disk To Data Size F	31.9	32.6	33.1
	Ownership	100.0 %	100.0 %	100.0 %
	Number of Tokens	256	256	256

Configuration

Storage Node Specific Settings

Property	Value	Description
Heap Size	<input type="text" value="512"/> MB	The maximum (and at the same time starting) heap size. This value will be used with the -Xmx and -Xms JVM options. If you are going to increase/decrease this value, then you should also increase/decrease the new generation proportionally. The value should be an integer with a suffix of M or G to indicate megabytes or gigabytes.
Heap New Size	<input type="text" value="128"/> MB	The size of the new generation portion of the heap. This value will be used with the -Xmn JVM option. The value should be an integer with a suffix of M or G to indicate megabytes or gigabytes.
Thread Stack Size	<input type="text" value="256"/>	The thread stack size. This memory is allocated to each thread off heap. The value should be an integer that will be interpreted in kilobytes.
JMX Port	<input type="text" value="7299"/>	The JMX port for the RHQ Storage Node

Save

- As recommended, adjust the **Heap New Size** setting in proportion to the heap size.
- Click the **Save** button at the bottom of the page.
- Restart the storage node. For example, on the storage node machine:

```
[root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl start --storage
```

[Report a bug](#)

9.8. SCALING THE CASSANDRA STORAGE NODE FOR JBOSS ON SERVER

IMPORTANT

Regardless of the load or demand on a JBoss ON installation, the JBoss ON server will go into maintenance mode while:

- the storage node processes a backlog of storage requests (during times of peak demand),
- transfers data from memory to disk, or
- performs data aggregation or compaction.

In most instances, the system is designed to recover without interruption. However, as the demand or load on a system increases, the length of these outages will also increase. Once the length of these outages begins to interrupt operations, additional storage nodes should be installed to distribute the load. Additional storage nodes are also required when the storage node has insufficient storage space. For details on installing additional storage nodes, see [Section 9.2.4, “Installing Additional Nodes”](#).

Scaling the Cassandra-based JBoss ON Storage Node helps improve performance and reliability, including improving the capacity of the storage node to respond to metric read and write requests. There are several ways the storage node can be scaled. These are:

- Increase the available memory in the JBoss ON Storage Node's JVM by increasing the **heap_max** and **heap_new** properties in ***RHQ_SERVER_HOME*/rhq-storage/conf/cassandra-jvm.properties**. For example:

```
heap_max=-Xmx1024M
heap_new=-Xmn384M
```

The heap size can also be changed in the JBoss ON UI, as described by [Section 9.7, “Managing the Storage Node Heap”](#).

IMPORTANT

Ensure the host machine has enough physical memory to handle the value set for maximum heap, otherwise the host machine uses swapping memory, slowing the processing speed of the storage node.

- Increase the length of time objects survive in the new generation space by increasing the **max_tenuring_threshold** property in ***RHQ_SERVER_HOME*/rhq-storage/conf/cassandra-jvm.properties**. For example:

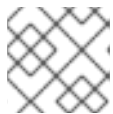
```
max_tenuring_threshold="-XX:MaxTenuringThreshold=4"
```

**WARNING**

If the max tenuring threshold of the JVM exceeds 15, the JVM garbage collection does not function correctly.

- Add additional JBoss ON Storage Nodes to the storage cluster to distribute processing across multiple machines and disks as described in [Section 9.2.4, “Installing Additional Nodes”](#).
- Relocate the **rhq-data** directory of the JBoss ON Storage Node to a dedicated disk to improve performance.
- Move the JBoss ON Storage Node's **rhq-data** and **commit_log** directories to dedicated physical disks.
- Increase the write timeout value by setting the **write_request_timeout_in_ms** property in **RHQ_SERVER_HOME/rhq-storage/conf/cassandra.yaml**. For example:

```
write_request_timeout_in_ms: 40000
```



NOTE

Increasing the timeout for write operations is not recommended.

[Report a bug](#)

9.9. ADJUSTING THE CAPACITY OF THE STORAGE NODE TO RECEIVE METRICS (STORAGE NODE REQUEST THROTTLING)

The JBoss ON storage node automatically throttles the request capacity (incoming metrics capacity), however in some situations manual tuning can be required. Manual tuning can be performed by adjusting the *Storage Client Settings* within the **rhq-server.properties** file or the JBoss ON GUI.

9.9.1. Storage Node Request Limit Settings (Storage Client Settings)

```
rhq.storage.request.limit.topology-delta=30000
```

The topology delta defines the request limit increase when additional storage nodes are added, expressed as requests per minute. For example, assuming the existing storage node has a limit of 30,000 requests per minute, adding a new storage node adds 30,000 requests per minute, increasing the limit to 60,000 requests per minute. For more details on adding additional storage nodes, see [Section 9.2.4, “Installing Additional Nodes”](#).

```
rhq.storage.request.limit.timeout-dampening=30000
```

The timeout dampener defines the period of time where successive timeouts are ignored, expressed in milliseconds (default 30,000ms). This timeout dampening property assumes that when a timeout occurs, there may be timeouts that immediately follow.

```
rhq.storage.request.limit.timeout-delta=0.2
```

(Deprecated from JBoss ON version 3.3.1) The timeout delta defines the proportional decrease (between 0 - 1, with a default of 0.2) in the request limit when a request timeout occurs. This assumes that request timeouts are caused by the hardware of the storage node having insufficient capacity to handle requests at the current request limit. For example, if the initial limit is 30,000

requests per minute and a request timeout occurs, the limit decreases by 20% to 24,000 requests per minute ($0.2 * 30,000$ requests per minute = 6,000 decrease in request limit, therefore the new request limit is 24,000 requests).

rhq.storage.request.limit.min=5000

(Deprecated from JBoss ON version 3.3.1) The **rhq.storage.request.limit.min** sets the minimum request limit. The storage request limit can not decrease below this value regardless of the number of timeouts.

rhq.storage.request.limit.warmup-period=3

(Introduced by the JBoss ON 3.3.1 update) The warm up period sets the increase in warm up time when storage request timeouts occur (expressed in minutes).

rhq.storage.request.limit.max-warmup-counter=10

(Introduced by the JBoss ON 3.3.1 update) The maximum warm up counter sets the maximum warm up time after multiple successive storage request timeouts. Maximum warm up time is a function of the warm up counter and the warm up period. For example, **warmup-period=3** multiplied by **max-warmup-counter=10** gives a maximum warm up time of 30 minutes.



USING STORAGE REQUEST LIMIT WARM UP SETTINGS

If a timeout occurs, the storage request limit is halved. The storage request limit returns to 100% of the configured storage request limit over the warm up time, initially equal to **rhq.storage.request.limit.warmup-period**. If subsequent timeouts occur during the warm up time (after accounting for the **rhq.storage.request.limit.timeout-dampening**), the warm up time increases by the warm up period up to a maximum warm up time. The maximum warm up time is given by: **rhq.storage.request.limit.max-warmup-counter** multiplied by **rhq.storage.request.limit.warmup-period** (the default maximum warm up time is 30 minutes).

9.9.2. Manually Change the Storage Node Request Limit Settings in `rhq-server.properties`

The storage node request limit settings can be changed manually in `<JBoss_ON_install_directory>/bin/rhq-server.properties` under the section *Storage Client Settings*.

The warm up settings for RHQ storage request limits and `<JBoss_ON_install_directory>/bin/rhq-server.properties.new` were added in JBoss ON version 3.3.1. To change the storage request limit warm up settings (**warmup-period** and **max-warmup-counter**) within the `<JBoss_ON_install_directory>/bin/rhq-server.properties` file, the settings need to be manually added as presented in [Section 9.9.1, “Storage Node Request Limit Settings \(Storage Client Settings\)”](#) or can be copied from `<JBoss_ON_install_directory>/bin/rhq-server.properties.new`.

9.9.3. Using the JBoss ON GUI to Adjust Storage Node Request Throttling

To change the Storage Node Request Throttling settings within the JBoss ON GUI:

1. Select the **Inventory** tab.
2. From the **Resources** menu, select **All Resources**.
3. Search for the **Storage Client Subsystem** and select the name to open.
4. From the **Storage Client Subsystem** page, select the **Configuration** tab.
5. Make your changes to the properties. Select the **Unset?** check box to use the default value, or clear the **Unset?** check box to use a custom value.
6. Click **Save** at the bottom of the page.

Figure 9.4. Example of Storage Node Request Throttling settings within the JBoss ON GUI.

The screenshot shows the JBoss ON GUI interface. The top navigation bar includes 'Dashboard', 'Inventory', 'Reports', 'Bundles', 'Administration', and 'Help'. The left sidebar shows a tree view of resources, with 'Storage Client Subsystem' selected under 'JBossAS7 Standalone Servers'. The main content area displays the configuration page for the 'Storage Client Subsystem (RHQ Server Storage Client Subsystem)'. The 'Configuration' tab is active, showing a table of properties.

Property	Unset?	Value	Description
Request Limit	<input type="checkbox"/>	30000.0	Sets throttling in terms of requests per second. Defaults to 30,000 if undefined. Note that this setting is automatically changed when various events such as request timeouts occur.
Request Limit Topology Change Delta	<input type="checkbox"/>	30000	The request limit will automatically change by this amount when there is a topology change event. Topology change events include nodes being added/removed and nodes going up/down. Defaults to 30,000 if undefined.
Warmup Period	<input type="checkbox"/>	3	The warmup period time during which the request limit is grown from 50% to 100% of the calculated value. The value is in minutes. Defaults to 3 minutes if undefined.
Warmup Counter Maximum	<input type="checkbox"/>	10	Sets the maximum throttling value for warmup periods. If a timeout occurs the request rate is halved with increased warmup period. This settings limits the maximum value of added warmup periods. The maximum warmup period is calculated: Warmup Period * Warmup Counter Maximum. Defaults to 10 if undefined (30 minutes maximum warmup with default values).

A 'Save' button is located at the bottom of the configuration area.

[Report a bug](#)

9.10. BACKING UP AND RESTORING THE METRICS STORAGE DATABASE

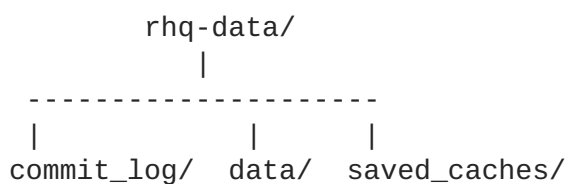
The storage nodes have a defined flow to manage and archive data. Scheduled *snapshots* back up all node data (both metrics and node configuration). This allows data to be restored if the cluster is ever corrupted.

[Report a bug](#)

9.10.1. About Storage Data Files

All of the database files for the node are stored in the **rhq-data/** directory, which is in the same root directory as the JBoss ON server (such as **/opt/jon/jon-server-3.3.0.GA**). This directory is divided into three subdirectories:

- A commit log for write data that has yet to be written to the database.
- A cache directory for archives.
- A data directory which contains both all monitoring data (aggregated and raw), snapshot archives of monitoring data, and node (system) data.



The **commit_log/** directory contains binary commit logs which store the writes before they are written to the database.

The **saved_caches/** directory contains archives for the major tables within the storage node, both the metrics tables and the storage node configuration.

The **data/** directory contains the most critical data. This directory is organized by keyspace. The keyspaces include the raw and aggregated metric data along with system related configuration, authentication, and schema details.

Some keyspace directories found under **data/** include a **snapshot/** sub-directory. The **snapshot/** directory represents a snapshot or backup at a specific point in time of the data found in the parent keyspace and its associated data files.

[Report a bug](#)

9.10.2. Storage Node Snapshots

JBoss ON provides the ability to take snapshots or backups of its monitoring and system data.

[Report a bug](#)

9.10.2.1. Taking an Individual Snapshot

An individual snapshot may be scheduled using the following steps:

1. Click on **Inventory** in the top navigation bar, then click on **Servers - Top Level Imports** in the **Resources** section in the left navigation bar.
2. Click on the desired storage node.
3. Click on the **Operations** tab, then click on the **New** button at the bottom.
4. Select **Take Snapshot** from the **Operation** drop down menu.
5. Fill in the desired snapshot and retention options.

Summary Inventory Alerts Monitoring **Operations** Configuration

Schedules History

Create New Operation Schedule

Operation *: **Take Snapshot** Takes a snapshot of all keyspaces. A snapshot first flushes all in-memory writes to disk and then creates a hard link of each SSTable file for the keyspace. Note that a column family can have multiple SSTables on disk. By default snapshots are stored in the `///snapshots` directory. On Linux/UNIX systems `cassandra_data_dir` defaults to `/var/lib/cassandra/data`

Parameters: *Enter parameters below...*

Property	Unset?	Value	Description
Snapshot Name	✓	<input type="text"/>	Snapshot name. If left empty current system time in milliseconds will be used as the snapshot name.
Retention Strategy		<input checked="" type="radio"/> Keep All <input type="radio"/> Keep Last N <input type="radio"/> Delete Older Than N days	Retention Strategy defines what happens to snapshots after being created. Default is Keep All - nothing is cleaned up, Deletion Strategy is ignored. Keep Last N - newest N snapshots remain, the rest of snapshots is cleaned up based on Deletion Strategy, N has to be specified using 'count' property. Delete Older Than N days - delete snapshots older than N days, rest of snapshots is cleaned up based on Deletion Strategy, N has to be specified using 'count' property
Count	✓	<input type="text"/>	Additional parameter to Retention Strategy, stands for N
Deletion Strategy		<input checked="" type="radio"/> Delete <input type="radio"/> Move	Deletion Strategy defines what happens to snapshots that don't match Retention Strategy. Default is Delete - snapshots will be deleted. Move - snapshots are moved to location specified by 'location' property.
Location	✓	<input type="text"/>	Additional parameter to Deletion Strategy, specifies location to move snapshots to, stands for L

Schedule Reset Cancel

6. *Optional.* A snapshot may be scheduled for a later time as well as repeated on a specified cycle. By default, snapshots are scheduled for "Now".

Summary Inventory Alerts Monitoring **Operations** Configuration

Schedules History

Create New Operation Schedule

Retention Strategy Keep All Keep Last N Delete Older Than N days
newest N snapshots remain, the rest of snapshots is cleaned up based on Deletion Strategy, N has to be specified using 'count' property. Delete Older Than N days - delete snapshots older than N days, rest of snapshots is cleaned up based on Deletion Strategy, N has to be specified using 'count' property

Count
Additional parameter to Retention Strategy, stands for N

Deletion Strategy Delete Move
Deletion Strategy defines what happens to snapshots that don't match Retention Strategy. Default is Delete - snapshots will be deleted. Move - snapshots are moved to location specified by 'location' property.

Location
Additional parameter to Deletion Strategy, specifies location to move snapshots to, stands for L

Schedule using : Calendar Cron Expression

Now Now & Repeat Later Later & Repeat

Run now and every * : For Until Indefinitely

Timeout : Notes :

- Click **Schedule** to schedule the snapshot. The user will then be taken to the operations history section, showing the status of the snapshot.

Summary Inventory Alerts Monitoring **Operations** Configuration

Schedules History

Operation Status : ✔ Success, 🔄 In Progress, ⚠ Canceled, ❌ Fail From : To : Total Rows: 1 (select)

Date Submitted	Operation	Requestor	Status
Sep 18, 2014 2:15:42 PM	Take Snapshot	rhqadmin	✔

Delete Force Delete Cancel Reschedule **New Schedule** Refresh

8. If snapshot was scheduled for later or set for repeating, it will also appear in the **Schedules** sub tab.

Total Rows: 1 (selected: 0)				
Schedule ID	Operation	Owner	Next Execution	Notes
10042	Take Snapshot	rhqadmin	Sep 25, 2014 2:19:03 PM	

[New](#)
[Delete](#)
[Refresh](#)

[Report a bug](#)

9.10.2.2. Scheduling Repeating Snapshots for the Storage Node Cluster

In addition to creating repeating snapshots at the individual storage node, repeating snapshots may also be scheduled at the storage node cluster level.

To schedule snapshots for the Storage Node Cluster:

1. Click on **Administration** in the top navigation bar, then click on **Storage Nodes** in the **Topology** section in the left navigation bar.
2. Click on the **Cluster Settings** tab and scroll down to the **Snapshot Management** section.

The screenshot shows the 'Cluster Settings' page for Storage Nodes. The left sidebar contains navigation menus for Security, Topology, Configuration, and Content. The main content area is divided into three sections:

- New Deployment Settings:** A table with columns 'Property', 'Value', and 'Description'. The 'Automatic Deployment' property is set to 'On'.
- Cluster Credentials:** A table with columns 'Property', 'Value', and 'Description'. The 'Username' is 'qefduuui' and the 'Password' is masked with '*****'. A 'Verify Password' field is also present.
- Snapshot Management:** A table with columns 'Property', 'Value', and 'Description'. The 'Enabled' property is set to 'On'. The 'Schedule' is '0 0 0 ? * SUN'. The 'Retention Strategy' is 'Keep All'. The 'Count' is '1'. The 'Deletion Strategy' is 'Delete'. The 'Location' field is empty.

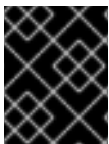
A 'Save' button is located at the bottom of the page.

3. Repeating snapshots may be enabled by setting the **Enabled** property to **On**.
4. Configure the desired schedule and retention strategy.
5. Click **Save**.



NOTE

Snapshot scheduled at the storage node cluster level will appear **Schedules** sub tab of each storage node in addition to any individually scheduled snapshots . If the snapshot schedule is updated at the storage node cluster level, this will also be reflected in the **Schedules** sub tab of each storage node. If a repeating snapshot scheduled at the storage node cluster level is disabled, it will be removed from the **Schedules** sub tab of each storage node and will not show up in **History** sub tab.

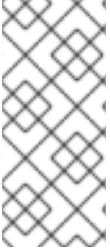


IMPORTANT

The snapshots are CPU- and disk-intensive. Performing snapshots during high-use times will result in performance degradation.

[Report a bug](#)

9.10.3. Restoring the Cluster

**NOTE**

The restore process restores an entire cluster to a previous state.

This is not intended to restore a single node. Because all data is replicated between all nodes, it is simpler and safer to remove a failed node and install a new one than it is to attempt to restore the node.

**IMPORTANT**

Every step must be performed on every node in the cluster.

1. Shut down every node in the storage cluster. Run the stop command on every storage machine:

```
[root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl.sh stop -
-storage
```

2. Remove the **commit_log/** directory for each node.

```
[root@server ~]# rm * /opt/jon/rhq-data/data/commit_log/*
```

3. Delete all files in the following directories, **except for the snapshot files**.

- /opt/jon/rhq-data/data/rhq/metrics_index/
- /opt/jon/rhq-data/data/rhq/one_hour_metrics/
- /opt/jon/rhq-data/data/rhq/raw_metrics/
- /opt/jon/rhq-data/data/rhq/schema_version/
- /opt/jon/rhq-data/data/rhq/six_hour_metrics/
- /opt/jon/rhq-data/data/rhq/twenty_four_hour_metrics/

```
[root@server ~]# cd /opt/jon/rhq-data/data/rhq/
[root@server rhq]# rm metrics_index/*. * one_hour_metrics/*. *
raw_metrics/*. * schema_version/*. * six_hour_metrics/*. *
twenty_four_hour_metrics/*. *
```

4. Copy all files from the snapshots directory into the **metrics_index** directory.

```
[root@server rhq]# cp metrics_index/snapshots/timestamp/*
/opt/jon/rhq-data/data/rhq/metrics_index
```

Repeat the command for the remaining directories:

- /opt/jon/rhq-data/data/rhq/metrics_index/
- /opt/jon/rhq-data/data/rhq/one_hour_metrics/
- /opt/jon/rhq-data/data/rhq/raw_metrics/
- /opt/jon/rhq-data/data/rhq/schema_version/

- /opt/jon/rhq-data/data/rhq/six_hour_metrics/
- /opt/jon/rhq-data/data/rhq/twenty_four_hour_metrics/

5. Restart each storage node.

```
[root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl start --  
storage
```

[Report a bug](#)

APPENDIX A. DOCUMENT HISTORY

Revision 3.3.7-3	Tue 11 Apr 2017	Scott Mumford
Rebuild for various bug fixes.		
Revision 3.3.7-2	Mon 05 Sep 2016	Scott Mumford
BZ-1370474: Fixed incorrect values in the configuration file shown in Procedure 4.1		
BZ-1369081: Corrected incorrect values in multiple rhq-server.properties entries		
BZ-1370498: Added missing configuration in Example SSL Configuration. Added note about updating keystore/truststore		
BZ-1370489: Clarified text in Enable client authentication		
Revision 3.3.2-13	Tue 23 Feb 2016	Scott Mumford
BZ-1323876: Updated "Setting up Client Authentication Between Servers and Agents" chapter.		
Revision 3.3.2-12	Tue 23 Feb 2016	Scott Mumford
BZ-1279557 and BZ-1279556: Updated listed secure-socket-protocols		
BZ-1214414: Updated commands using DSA algorithm to RSA.		
BZ-1187400: Clarified contradictory uses of SSL transport connector options in procedure.		
Revision 3.3.2-11	Thu 02 Jul 2015	Jared Morgan
Fixed HTML Anchors that were breaking links in the JBoss ON UI Help > Documentation menu.		
Removed preface matter.		
Revision 3.3.2-1	Tue 28 Apr 2015	Jared Morgan
Prepared for JBoss ON 3.3.2 Release		
Revision 3.3.1-1	Wed Feb 18 2015	Jared Morgan
Prepared for JBoss ON 3.3.1; Release		
Revision 3.3-32	Mon Nov 17 2014	Jared Morgan
See 3.3 Release Notes for Customer Reported Defects		
https://bugzilla.redhat.com/show_bug.cgi?id=1095805 - corrected an issue with JVM parameter for Section 3.5, "Tuning the Server JVM" .		
https://bugzilla.redhat.com/show_bug.cgi?id=1118072 - changed two steps in the Section 9.10.3, "Restoring the Cluster" procedure to reflect the additional tables required.		
https://bugzilla.redhat.com/show_bug.cgi?id=1128770 - added the -g --purgeplugins option as described in Section 7.3.2, "Agent Start Options" .		
https://bugzilla.redhat.com/show_bug.cgi?id=1148601 - Change from High Availability to Topology and converted to a menuchoice in Section 7.15, "Viewing the Server Failover Lists for Agents"		
https://bugzilla.redhat.com/show_bug.cgi?id=1148576 - Updated the procedure to reflect the new screenshots in Section 5.2, "Creating Affinity Groups" . Also did a copy-edit and XML tweaks on this section.		
https://bugzilla.redhat.com/show_bug.cgi?id=1148584 - Adjusted the procedure in Section 5.4, "Removing Servers from the High Availability Cloud" to reflect that you click a row to highlight it rather than a checkbox now.		
https://bugzilla.redhat.com/show_bug.cgi?id=1148581 - Converted orderedlist to a procedure, changed figures, and updated procedure text in Section 5.3, "Putting Servers in Maintenance Mode" .		
https://bugzilla.redhat.com/show_bug.cgi?id=1111311 - corrected an issue with the email.jsp path in the Section 6.3.4, "Configuring the SMTP Server for Email Notifications" section.		