Red Hat AMQ Broker 7.10

Managing AMQ Broker

For Use with AMQ Broker 7.10

Red Hat AMQ Broker 7.10 Managing AMQ Broker

For Use with AMQ Broker 7.10

Legal Notice

Abstract

This guide describes how to monitor, manage, and upgrade AMQ Broker.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message.

CHAPTER 1. OVERVIEW

AMQ Broker is a high-performance messaging implementation based on ActiveMQ Artemis. It has fast, journal-based message persistence and supports multiple languages, protocols, and platforms.

AMQ Broker provides multiple interfaces for managing and interacting with your broker instances, such as a management console, management APIs, and a command-line interface. In addition, you can monitor broker performance by collecting runtime metrics, configure brokers to proactively monitor for problems such as deadlock conditions, and interactively check the health of brokers and queues.

This guide provides detailed information about typical broker management tasks such as:

- Upgrading your broker instances

- Using the command-line interface and management API

- Checking the health of brokers and queues

- Collecting broker runtime metrics

- Proactively monitoring critical broker operations

1.1. SUPPORTED CONFIGURATIONS

Refer to the article "Red Hat AMQ 7 Supported Configurations " on the Red Hat Customer Portal for current information regarding AMQ Broker supported configurations.

1.2. DOCUMENT CONVENTIONS

This document uses the following conventions for the **sudo** command, file paths, and replaceable values.

The sudo command

In this document, **sudo** is used for any command that requires root privileges. You should always exercise caution when using **sudo**, as any changes can affect the entire system.

For more information about using **sudo**, see Managing sudo access.

About the use of file paths in this document

In this document, all file paths are valid for Linux, UNIX, and similar operating systems (for example, /**home**/**...**). If you are using Microsoft Windows, you should use the equivalent Microsoft Windows paths (for example, **C:\Users\...**).

Replaceable values

This document sometimes uses replaceable values that you must replace with values specific to your environment. Replaceable values are lowercase, enclosed by angle brackets (**< >**), and are styled using italics and **monospace** font. Multiple words are separated by underscores (**_**) .

For example, in the following command, replace ***<install_dir>*** with your own directory name.

```
$ <install_dir>/bin/artemis create mybroker
```

CHAPTER 2. UPGRADING YOUR BROKER

2.1. ABOUT UPGRADES

Red Hat releases new versions of AMQ Broker to the Customer Portal. Update your brokers to the newest version to ensure that you have the latest enhancements and fixes. In general, Red Hat releases a new version of AMQ Broker in one of three ways:

Major Release

A major upgrade or migration is required when an application is transitioned from one major release to the next, for example, from AMQ Broker 6 to AMQ Broker 7. This type of upgrade is not addressed in this guide.

Minor Release

AMQ Broker periodically provides minor releases, which are updates that include new features, as well as bug and security fixes. If you plan to upgrade from one AMQ Broker minor release to another, for example, from AMQ Broker 7.0 to AMQ Broker 7.1, code changes should not be required for applications that do not use private, unsupported, or tech preview components.

Micro Release

AMQ Broker also periodically provides micro releases that contain minor enhancements and fixes. Micro releases increment the minor release version by the last digit, for example from 7.0.1 to 7.0.2. A micro release should not require code changes, however, some releases may require configuration changes.

2.2. UPGRADING OLDER 7.X VERSIONS

2.2.1. Upgrading a broker instance from 7.0.x to 7.0.y

The procedure for upgrading AMQ Broker from one version of 7.0 to another is similar to the one for installation: you download an archive from the Customer Portal and then extract it.

The following subsections describe how to upgrade a 7.0.x broker for different operating systems.

- Upgrading from 7.0.x to 7.0.y on Linux

- Upgrading from 7.0.x to 7.0.y on Windows

2.2.1.1. Upgrading from 7.0.x to 7.0.y on Linux

The name of the archive that you download could differ from what is used in the following examples.

Prerequisites

- Before upgrading AMQ Broker, review the release notes for the target release.
  The release notes describe important enhancements, known issues, and changes to behavior in the target release.

  For more information, see the AMQ Broker 7.0 Release Notes .

Procedure

1. Download the desired archive from the Red Hat Customer Portal by following the instructions provided in Downloading the AMQ Broker archive .

2. Change the owner of the archive to the same user that owns the AMQ Broker installation to be upgraded.

```
sudo chown amq-broker:amq-broker jboss-amq-7.x.x.redhat-1.zip
```

3. Move the archive to the directory created during the original installation of AMQ Broker. In the following example, the directory **/opt/redhat** is used.

```
sudo mv jboss-amq-7.x.x.redhat-1.zip /opt/redhat
```

4. As the directory owner, extract the contents of the compressed archive. The archive is kept in a compressed format. In the following example, the user **amq-broker** extracts the archive by using the unzip command.

```
su - amq-broker
cd /opt/redhat
unzip jboss-amq-7.x.x.redhat-1.zip
```

5. Stop the broker if it is running.

```
<broker_instance_dir>/bin/artemis stop
```

6. Back up the instance directory of the broker by copying it to the home directory of the current user.

```
cp -r <broker_instance_dir> ~/
```

7. (Optional) Note the current version of the broker. After the broker stops, a line similar to the one below is displayed at the end of its log file, which can be found at **<broker_instance_dir>/log/artemis.log**.

```
INFO  [org.apache.activemq.artemis.core.server] AMQ221002: Apache ActiveMQ Artemis
Message Broker version 2.0.0.amq-700005-redhat-1 [4782d50d-47a2-11e7-a160-
9801a793ea45] stopped, uptime 28 minutes
```

8. Edit the **<broker_instance_dir>/etc/artemis.profile** configuration file to set the **ARTEMIS_HOME** property to the new directory created when the archive was extracted.

```
ARTEMIS_HOME='/opt/redhat/jboss-amq-7.x.x-redhat-1'
```

9. Start the upgraded broker.

```
<broker_instance_dir>/bin/artemis run
```

10. (Optional) Confirm that the broker is running and that the version has changed. After starting the broker, open the log file **<broker_instance_dir>/log/artemis.log** and find two lines similar to the ones below. Note the new version number that appears in the log after the broker is live.

```
INFO  [org.apache.activemq.artemis.core.server] AMQ221007: Server is now live
...
INFO  [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis
Message Broker version 2.1.0.amq-700005-redhat-1 [0.0.0.0, nodeID=4782d50d-47a2-11e7-
a160-9801a793ea45]
```

2.2.1.2. Upgrading from 7.0.x to 7.0.y on Windows

Prerequisites

- Before upgrading AMQ Broker, review the release notes for the target release.
  The release notes describe important enhancements, known issues, and changes to behavior in the target release.

  For more information, see the AMQ Broker 7.0 Release Notes .

Procedure

1. Download the desired archive from the Red Hat Customer Portal by following the instructions provided in Downloading the AMQ Broker archive .

2. Use a file manager to move the archive to the folder you created during the last installation of AMQ Broker.

3. Extract the contents of the archive. Right-click the .zip file and select **Extract All**.

4. Stop the broker if it is running by entering the following command.

   > *<broker_instance_dir>*\bin\artemis-service.exe stop

5. Back up the broker by using a file manager.

   a. Right-click the ***<broker_instance_dir>*** folder and select **Copy**.

   b. Right-click in the same window and select **Paste**.

6. (Optional) Note the current version of the broker. After the broker stops, a line similar to the one below is displayed at the end of its log file, which can be found at ***<broker_instance_dir>*\log\artemis.log**.

   > INFO [org.apache.activemq.artemis.core.server] AMQ221002: Apache ActiveMQ Artemis Message Broker version 2.0.0.amq-700005-redhat-1 [4782d50d-47a2-11e7-a160-9801a793ea45] stopped, uptime 28 minutes

7. Edit the ***<broker_instance_dir>*\etc\artemis.profile** configuration file to set the **ARTEMIS_HOME** property to the new directory created when the archive was extracted.

   > ARTEMIS_HOME=*<install_dir>*

8. Start the upgraded broker.

   > *<broker_instance_dir>*\bin\artemis-service.exe start

9. (Optional) Confirm that the broker is running and that the version has changed. After starting the broker, open the log file ***<broker_instance_dir>*\log\artemis.log** and find two lines similar to the ones below. Note the new version number that appears in the log after the broker is live.

   > INFO [org.apache.activemq.artemis.core.server] AMQ221007: Server is now live
   > …

> INFO [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis Message Broker version 2.1.0.amq-700005-redhat-1 [0.0.0.0, nodeID=4782d50d-47a2-11e7-a160-9801a793ea45]

2.2.2. Upgrading a broker instance from 7.0.x to 7.1.0

AMQ Broker 7.1.0 includes configuration files and settings that were not included with previous versions. Upgrading a broker instance from 7.0.x to 7.1.0 requires adding these new files and settings to your existing 7.0.x broker instances. The following subsections describe how to upgrade a 7.0.x broker instance to 7.1.0 for different operating systems.

> **IMPORTANT**
>
> Starting with AMQ Broker 7.1.0, you can access AMQ Management Console only from the local host by default. To learn about configuring remote access to the console, see Configuring local and remote access to AMQ Management Console .

- Upgrading from 7.0.x to 7.1.0 on Linux

- Upgrading from 7.0.x to 7.1.0 on Windows

2.2.2.1. Upgrading from 7.0.x to 7.1.0 on Linux

Before you can upgrade a 7.0.x broker, you need to install Red Hat AMQ Broker 7.1.0 and create a temporary broker instance. This will generate the 7.1.0 configuration files required to upgrade a 7.0.x broker.

Prerequisites

- Before upgrading AMQ Broker, review the release notes for the target release.
  The release notes describe important enhancements, known issues, and changes to behavior in the target release.

  For more information, see the AMQ Broker 7.1 Release Notes .

- Before upgrading your 7.0.x brokers, you must first install version 7.1.
  For steps on installing 7.1 on Linux, see Installing AMQ Broker .

Procedure

1. If it is running, stop the 7.0.x broker you want to upgrade:

   > $ *<broker_instance_dir>*/bin/artemis stop

2. Back up the instance directory of the broker by copying it to the home directory of the current user.

   > cp -r *<broker_instance_dir>* ~/

3. Open the file **artemis.profile** in the ***<broker_instance_dir>*/etc/** directory of the 7.0.x broker.

   a. Update the **ARTEMIS_HOME** property so that its value refers to the installation directory for AMQ Broker 7.1.0:

   > ▪

> ARTEMIS_HOME="*<7.1.0_install_dir>*"

b. On the line below the one you updated, add the property **ARTEMIS_INSTANCE_URI** and assign it a value that refers to the 7.0.x broker instance directory:

> ARTEMIS_INSTANCE_URI="file://*<7.0.x_broker_instance_dir>*"

c. Update the **JAVA_ARGS** property by adding the **jolokia.policyLocation** parameter and assigning it the following value:

> -Djolokia.policyLocation=${ARTEMIS_INSTANCE_URI}/etc/jolokia-access.xml

4. Create a 7.1.0 broker instance. The creation procedure generates the configuration files required to upgrade from 7.0.x to 7.1.0. In the following example, note that the instance is created in the directory **upgrade_tmp**:

> $ *<7.1.0_install_dir>*/bin/artemis create --allow-anonymous --user admin --password admin upgrade_tmp

5. Copy configuration files from the **etc** directory of the temporary 7.1.0 instance into the ***<broker_instance_dir>*/etc/** directory of the 7.0.x broker.

a. Copy the **management.xml** file:

> $ cp *<temporary_7.1.0_broker_instance_dir>*/etc/management.xml *<7.0_broker_instance_dir>*/etc/

b. Copy the **jolokia-access.xml** file:

> $ cp *<temporary_7.1.0_broker_instance_dir>*/etc/jolokia-access.xml *<7.0_broker_instance_dir>*/etc/

6. Open up the **bootstrap.xml** file in the ***<broker_instance_dir>*/etc/** directory of the 7.0.x broker.

a. Comment out or delete the following two lines:

> <app url="jolokia" war="jolokia.war"/>
> <app url="hawtio" war="hawtio-no-slf4j.war"/>

b. Add the following to replace the two lines removed in the previous step:

> <app url="console" war="console.war"/>

7. Start the broker that you upgraded:

> $ *<broker_instance_dir>*/bin/artemis run

Additional Resources

For more information about creating an instance of the broker, see Creating a broker instance .

2.2.2.2. Upgrading from 7.0.x to 7.1.0 on Windows

Before you can upgrade a 7.0.x broker, you need to install Red Hat AMQ Broker 7.1.0 and create a temporary broker instance. This will generate the 7.1.0 configuration files required to upgrade a 7.0.x broker.

Prerequisites

- Before upgrading AMQ Broker, review the release notes for the target release.
  The release notes describe important enhancements, known issues, and changes to behavior in the target release.

  For more information, see the AMQ Broker 7.1 Release Notes .

- Before upgrading your 7.0.x brokers, you must first install version 7.1.
  For steps on installing 7.1 on Windows, see Installing AMQ Broker .

Procedure

1. If it is running, stop the 7.0.x broker you want to upgrade:

   > *<broker_instance_dir>*\bin\artemis-service.exe stop

2. Back up the instance directory of the broker by using a file manager.

   a. Right-click the ***<broker_instance_dir>*** folder and select **Copy**.

   b. Right-click in the same window and select **Paste**.

3. Open the file **artemis.profile** in the ***<broker_instance_dir>*/etc/** directory of the 7.0.x broker.

   a. Update the **ARTEMIS_HOME** property so that its value refers to the installation directory for AMQ Broker 7.1.0:

   ARTEMIS_HOME="*<7.1.0_install_dir>*"

   b. On the line below the one you updated, add the property **ARTEMIS_INSTANCE_URI** and assign it a value that refers to the 7.0.x broker instance directory:

   ARTEMIS_INSTANCE_URI="file://*<7.0.x_broker_instance_dir>*"

   c. Update the **JAVA_ARGS** property by adding the **jolokia.policyLocation** parameter and assigning it the following value:

   -Djolokia.policyLocation=${ARTEMIS_INSTANCE_URI}/etc/jolokia-access.xml

4. Create a 7.1.0 broker instance. The creation procedure generates the configuration files required to upgrade from 7.0.x to 7.1.0. In the following example, note that the instance is created in the directory **upgrade_tmp**:

   > *<7.1.0_install_dir>*/bin/artemis create --allow-anonymous --user admin --password admin upgrade_tmp

5. Copy configuration files from the **etc** directory of the temporary 7.1.0 instance into the ***<broker_instance_dir>*/etc/** directory of the 7.0.x broker.

a. Copy the **management.xml** file:

```
> cp <temporary_7.1.0_broker_instance_dir>/etc/management.xml
<7.0_broker_instance_dir>/etc/
```

b. Copy the **jolokia-access.xml** file:

```
> cp <temporary_7.1.0_broker_instance_dir>/etc/jolokia-access.xml
<7.0_broker_instance_dir>/etc/
```

6. Open up the **bootstrap.xml** file in the *<broker_instance_dir>***/etc/** directory of the 7.0.x broker.

a. Comment out or delete the following two lines:

```
<app url="jolokia" war="jolokia.war"/>
<app url="hawtio" war="hawtio-no-slf4j.war"/>
```

b. Add the following to replace the two lines removed in the previous step:

```
<app url="console" war="console.war"/>
```

7. Start the broker that you upgraded:

```
> <broker_instance_dir>\bin\artemis-service.exe start
```

Additional Resources

For more information about creating an instance of the broker, see Creating a broker instance .

2.2.3. Upgrading a broker instance from 7.1.x to 7.2.0

AMQ Broker 7.2.0 includes configuration files and settings that were not included with 7.0.x versions. If you are running 7.0.x instances, you must first upgrade those broker instances from 7.0.x to 7.1.0 before upgrading to 7.2.0. The following subsections describe how to upgrade a 7.1.x broker instance to 7.2.0 for different operating systems.

> **IMPORTANT**
>
> Starting with AMQ Broker 7.1.0, you can access AMQ Management Console only from the local host by default. To learn about configuring remote access to the console, see Configuring local and remote access to AMQ Management Console .

- Upgrading from 7.1.x to 7.2.0 on Linux

- Upgrading from 7.1.x to 7.2.0 on Windows

2.2.3.1. Upgrading from 7.1.x to 7.2.0 on Linux

> **NOTE**
>
> The name of the archive that you download could differ from what is used in the following examples.

Procedure

1. Download the desired archive from the Red Hat Customer Portal by following the instructions provided in Downloading the AMQ Broker archive .

2. Change the owner of the archive to the same user that owns the AMQ Broker installation to be upgraded.

   ```
   sudo chown amq-broker:amq-broker amq-7.x.x.redhat-1.zip
   ```

3. Move the archive to the directory created during the original installation of AMQ Broker. In the following example, the directory **/opt/redhat** is used.

   ```
   sudo mv amq-7.x.x.redhat-1.zip /opt/redhat
   ```

4. As the directory owner, extract the contents of the compressed archive. In the following example, the user **amq-broker** extracts the archive by using the unzip command.

   ```
   su - amq-broker
   cd /opt/redhat
   unzip jboss-amq-7.x.x.redhat-1.zip
   ```

5. Stop the broker if it is running.

   ```
   <broker_instance_dir>/bin/artemis stop
   ```

6. Back up the instance directory of the broker by copying it to the home directory of the current user.

   ```
   cp -r <broker_instance_dir> ~/
   ```

7. (Optional) Note the current version of the broker. After the broker stops, a line similar to the one below is displayed at the end of its log file, which can be found at **<broker_instance_dir>/log/artemis.log**.

   ```
   INFO  [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis
   Message Broker version 2.5.0.amq-720001-redhat-1 [0.0.0.0, nodeID=554cce00-63d9-11e8-
   9808-54ee759954c4]
   ```

8. Edit the **<broker_instance_dir>/etc/artemis.profile** configuration file to set the **ARTEMIS_HOME** property to the new directory created when the archive was extracted.

   ```
   ARTEMIS_HOME='/opt/redhat/amq-7.x.x-redhat-1'
   ```

9. Start the upgraded broker.

   ```
   <broker_instance_dir>/bin/artemis run
   ```

10. (Optional) Confirm that the broker is running and that the version has changed. After starting the broker, open the log file **<broker_instance_dir>/log/artemis.log** and find two lines similar to the ones below. Note the new version number that appears in the log after the broker is live.

    ```
    INFO  [org.apache.activemq.artemis.core.server] AMQ221007: Server is now live
    ```

> ...
> INFO [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis Message Broker version 2.5.0.amq-720001-redhat-1 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-54ee759954c4]

Additional Resources

- For more information about creating an instance of the broker, see Creating a broker instance .

- You can now store a broker instance's configuration files and data in any custom directory, including locations outside of the broker instance's directory. In the ***<broker_instance_dir>*/etc/artemis.profile** file, update the **ARTEMIS_INSTANCE_ETC_URI** property by specifying the location of the custom directory after creating the broker instance. Previously, these configuration files and data could only be stored in the **etc/** and **data/** directories within the broker instance's directory.

2.2.3.2. Upgrading from 7.1.x to 7.2.0 on Windows

Procedure

1. Download the desired archive from the Red Hat Customer Portal by following the instructions provided in Downloading the AMQ Broker archive .

2. Use a file manager to move the archive to the folder you created during the last installation of AMQ Broker.

3. Extract the contents of the archive. Right-click the .zip file and select **Extract All**.

4. Stop the broker if it is running by entering the following command.

   > *<broker_instance_dir>*\bin\artemis-service.exe stop

5. Back up the broker by using a file manager.

   a. Right-click the ***<broker_instance_dir>*** folder and select **Copy**.

   b. Right-click in the same window and select **Paste**.

6. (Optional) Note the current version of the broker. After the broker stops, a line similar to the one below is displayed at the end of its log file, which can be found at ***<broker_instance_dir>*\log\artemis.log**.

   > INFO [org.apache.activemq.artemis.core.server] AMQ221002: Apache ActiveMQ Artemis Message Broker version 2.0.0.amq-700005-redhat-1 [4782d50d-47a2-11e7-a160-9801a793ea45] stopped, uptime 28 minutes

7. Edit the ***<broker_instance_dir>*\etc\artemis.profile.cmd** and ***<broker_instance_dir>*\bin\artemis-service.xml** configuration files to set the **ARTEMIS_HOME** property to the new directory created when the archive was extracted.

   > ARTEMIS_HOME=*<install_dir>*

8. Start the upgraded broker.

> *<broker_instance_dir>*\bin\artemis-service.exe start

9. (Optional) Confirm that the broker is running and that the version has changed. After starting the broker, open the log file **<broker_instance_dir>\log\artemis.log** and find two lines similar to the ones below. Note the new version number that appears in the log after the broker is live.

> INFO [org.apache.activemq.artemis.core.server] AMQ221007: Server is now live
>
> ...
>
> INFO [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis Message Broker version 2.5.0.amq-720001-redhat-1 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-54ee759954c4]

Additional Resources

- For more information about creating an instance of the broker, see Creating a broker instance .

- You can now store a broker instance's configuration files and data in any custom directory, including locations outside of the broker instance's directory. In the **<broker_instance_dir>\etc\artemis.profile** file, update the **ARTEMIS_INSTANCE_ETC_URI** property by specifying the location of the custom directory after creating the broker instance. Previously, these configuration files and data could only be stored in the **\etc** and **\data** directories within the broker instance's directory.

2.2.4. Upgrading a broker instance from 7.2.x to 7.3.0

The following subsections describe how to upgrade a 7.2.x broker instance to 7.3.0 for different operating systems.

2.2.4.1. Resolve exception due to deprecated dispatch console

Starting in version 7.3.0, AMQ Broker no longer ships with the Hawtio dispatch console plugin **dispatch-hawtio-console.war**. Previously, the dispatch console was used to manage AMQ Interconnect. However, AMQ Interconnect now uses its own, standalone web console. This change affects the upgrade procedures in the sections that follow.

If you take no further action before upgrading your broker instance to 7.3.0, the upgrade process produces an exception that looks like the following:

> 2019-04-11 18:00:41,334 WARN [org.eclipse.jetty.webapp.WebAppContext] Failed startup of context o.e.j.w.WebAppContext@1ef3efa8{/dispatch-hawtio-console,null,null}{/opt/amqbroker/amq-broker-7.3.0/web/dispatch-hawtio-console.war}: java.io.FileNotFoundException: /opt/amqbroker/amq-broker-7.3.0/web/dispatch-hawtio-console.war.

You can safely ignore the preceding exception without affecting the success of your upgrade.

However, if you would prefer not to see this exception during your upgrade, you must first remove a reference to the Hawtio dispatch console plugin in the **bootstrap.xml** file of your existing broker instance. The **bootstrap.xml** file is in the **{instance_directory}/etc/** directory of your broker instance. The following example shows some of the contents of the **bootstrap.xml** file for a AMQ Broker 7.2.4 instance:

> <broker xmlns="http://activemq.org/schema">
>
> ....
>
>   <!-- The web server is only bound to localhost by default -->

```
<web bind="http://localhost:8161" path="web">
    <app url="redhat-branding" war="redhat-branding.war"/>
    <app url="artemis-plugin" war="artemis-plugin.war"/>
    <app url="dispatch-hawtio-console" war="dispatch-hawtio-console.war"/>
    <app url="console" war="console.war"/>
  </web>
</broker>
```

To avoid an exception when upgrading AMQ Broker to version 7.3.0, delete the line **<app url="dispatch-hawtio-console" war="dispatch-hawtio-console.war"/>**, as shown in the preceding example. Then, save the modified bootstrap file and start the upgrade process, as described in the sections that follow.
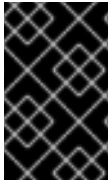
> **IMPORTANT**
>
> Starting with AMQ Broker 7.1.0, you can access AMQ Management Console only from the local host by default. To learn about configuring remote access to the console, see Configuring local and remote access to AMQ Management Console .

- Upgrading from 7.2.x to 7.3.0 on Linux

- Upgrading from 7.2.x to 7.3.0 on Windows

2.2.4.2. Upgrading from 7.2.x to 7.3.0 on Linux

> **NOTE**
>
> The name of the archive that you download could differ from what is used in the following examples.

Procedure

1. Download the desired archive from the Red Hat Customer Portal by following the instructions provided in Downloading the AMQ Broker archive .

2. Change the owner of the archive to the same user that owns the AMQ Broker installation to be upgraded.

   ```
   sudo chown amq-broker:amq-broker amq-7.x.x.redhat-1.zip
   ```

3. Move the archive to the directory created during the original installation of AMQ Broker. In the following example, the directory **/opt/redhat** is used.

   ```
   sudo mv amq-7.x.x.redhat-1.zip /opt/redhat
   ```

4. As the directory owner, extract the contents of the compressed archive. In the following example, the user **amq-broker** extracts the archive by using the unzip command.

   ```
   su - amq-broker
   cd /opt/redhat
   unzip jboss-amq-7.x.x.redhat-1.zip
   ```

5. Stop the broker if it is running.

```
<broker_instance_dir>/bin/artemis stop
```

6. Back up the instance directory of the broker by copying it to the home directory of the current user.

```
cp -r <broker_instance_dir> ~/
```

7. (Optional) Note the current version of the broker. After the broker stops, a line similar to the one below is displayed at the end of its log file, which can be found at **<broker_instance_dir>/log/artemis.log**.

```
INFO  [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis
Message Broker version 2.6.3.amq-720001-redhat-1 [0.0.0.0, nodeID=554cce00-63d9-11e8-
9808-54ee759954c4]
```

8. Edit the **<broker_instance_dir>/etc/artemis.profile** configuration file to set the **ARTEMIS_HOME** property to the new directory created when the archive was extracted.

```
ARTEMIS_HOME='/opt/redhat/amq-7.x.x-redhat-1'
```

9. Start the upgraded broker.

```
<broker_instance_dir>/bin/artemis run
```

10. (Optional) Confirm that the broker is running and that the version has changed. After starting the broker, open the log file **<broker_instance_dir>/log/artemis.log** and find two lines similar to the ones below. Note the new version number that appears in the log after the broker is live.

```
INFO  [org.apache.activemq.artemis.core.server] AMQ221007: Server is now live
...
INFO  [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis
Message Broker version 2.7.0.redhat-00054 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-
54ee759954c4]
```

Additional Resources

- For more information about creating an instance of the broker, see Creating a broker instance .

- You can now store a broker instance's configuration files and data in any custom directory, including locations outside of the broker instance's directory. In the **<broker_instance_dir>/etc/artemis.profile** file, update the **ARTEMIS_INSTANCE_ETC_URI** property by specifying the location of the custom directory after creating the broker instance. Previously, these configuration files and data could only be stored in the **etc/** and **data/** directories within the broker instance's directory.

2.2.4.3. Upgrading from 7.2.x to 7.3.0 on Windows

Procedure

1. Download the desired archive from the Red Hat Customer Portal by following the instructions provided in Downloading the AMQ Broker archive .

2. Use a file manager to move the archive to the folder you created during the last installation of AMQ Broker.

3. Extract the contents of the archive. Right-click the .zip file and select **Extract All**.

4. Stop the broker if it is running by entering the following command.

   <broker_instance_dir>\bin\artemis-service.exe stop

5. Back up the broker by using a file manager.

   a. Right-click the ***<broker_instance_dir>*** folder and select **Copy**.

   b. Right-click in the same window and select **Paste**.

6. (Optional) Note the current version of the broker. After the broker stops, a line similar to the one below is displayed at the end of its log file, which can be found at ***<broker_instance_dir>*\log\artemis.log**.

   INFO [org.apache.activemq.artemis.core.server] AMQ221002: Apache ActiveMQ Artemis Message Broker version 2.6.3.amq-720001-redhat-1 [4782d50d-47a2-11e7-a160-9801a793ea45] stopped, uptime 28 minutes

7. Edit the ***<broker_instance_dir>*\etc\artemis.profile.cmd** and ***<broker_instance_dir>*\bin\artemis-service.xml** configuration files to set the **ARTEMIS_HOME** property to the new directory created when the archive was extracted.

   ARTEMIS_HOME=*<install_dir>*

8. Edit the ***<broker_instance_dir>*\etc\artemis.profile.cmd** configuration file to set the JAVA_ARGS environment variable to reference the correct log manager version.

   JAVA_ARGS=*<install_dir>*\lib\jboss-logmanager-2.0.3.Final-redhat-1.jar

9. Edit the ***<broker_instance_dir>*\bin\artemis-service.xml** configuration file to set the bootstrap class path start argument to reference the correct log manager version.

   <startargument>Xbootclasspath/a:%ARTEMIS_HOME%\lib\jboss-logmanager-2.0.3.Final-redhat-1.jar</startargument>

10. Start the upgraded broker.

    <broker_instance_dir>\bin\artemis-service.exe start

11. (Optional) Confirm that the broker is running and that the version has changed. After starting the broker, open the log file ***<broker_instance_dir>*\log\artemis.log** and find two lines similar to the ones below. Note the new version number that appears in the log after the broker is live.

    INFO [org.apache.activemq.artemis.core.server] AMQ221007: Server is now live
    ...
    INFO [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis Message Broker version 2.7.0.redhat-00054 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-54ee759954c4]

Additional Resources

- For more information about creating an instance of the broker, see Creating a broker instance.

- You can now store a broker instance's configuration files and data in any custom directory, including locations outside of the broker instance's directory. In the **<broker_instance_dir>\etc\artemis.profile** file, update the **ARTEMIS_INSTANCE_ETC_URI** property by specifying the location of the custom directory after creating the broker instance. Previously, these configuration files and data could only be stored in the **\etc** and **\data** directories within the broker instance's directory.

2.2.5. Upgrading a broker instance from 7.3.0 to 7.4.0

The following subsections describe how to upgrade a 7.3.0 broker instance to 7.4.0 for different operating systems.
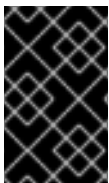
> **IMPORTANT**
>
> Starting with AMQ Broker 7.1.0, you can access AMQ Management Console only from the local host by default. To learn about configuring remote access to the console, see Configuring local and remote access to AMQ Management Console .

- Upgrading from 7.3.0 to 7.4.0 on Linux

- Upgrading from 7.3.0 to 7.4.0 on Windows

2.2.5.1. Upgrading from 7.3.0 to 7.4.0 on Linux

> **NOTE**
>
> The name of the archive that you download could differ from what is used in the following examples.

Procedure

1. Download the desired archive from the Red Hat Customer Portal. Follow the instructions provided in Downloading the AMQ Broker archive.

2. Change the owner of the archive to the same user that owns the AMQ Broker installation to be upgraded. The following example shows a user called **amq-broker**.

   sudo chown amq-broker:amq-broker amq-broker-7.x.x.redhat-1.zip

3. Move the archive to the directory created during the original installation of AMQ Broker. The following example uses **/opt/redhat**.

   sudo mv amq-broker-7.x.x.redhat-1.zip /opt/redhat

4. As the directory owner, extract the contents of the compressed archive. In the following example, the user **amq-broker** extracts the archive using the **unzip** command.

```
su - amq-broker
cd /opt/redhat
unzip amq-broker-7.x.x.redhat-1.zip
```

5. If the broker is running, stop it.

```
<broker_instance_dir>/bin/artemis stop
```

6. Back up the instance directory of the broker by copying it to the home directory of the current user.

```
cp -r <broker_instance_dir> ~/
```

7. (Optional) Note the current version of the broker. After the broker stops, you see a line similar to the one below at the end of the **<broker_instance_dir>/log/artemis.log** file.

```
INFO  [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis
Message Broker version 2.7.0.redhat-00054 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-
54ee759954c4]
```

8. Edit the **<broker_instance_dir>/etc/artemis.profile** configuration file.

   a. Set the **ARTEMIS_HOME** property to the new directory created when the archive was extracted.

   ```
   ARTEMIS_HOME='/opt/redhat/amq-broker-7.x.x-redhat-1'
   ```

   b. Edit the **JAVA_ARGS** property. Add the bootstrap class path argument, which references a dependent file for the log manager.

   ```
   -Xbootclasspath/a:$ARTEMIS_HOME/lib/wildfly-common-1.5.1.Final-redhat-00001.jar
   ```

9. Edit the **<broker_instance_dir>/etc/bootstrap.xml** configuration file. In the **<web>** configuration element, add a reference to the metrics plugin file for AMQ Broker.

```
<app url="metrics" war="metrics.war"/>
```

10. Start the upgraded broker.

```
<broker_instance_dir>/bin/artemis run
```

11. (Optional) Confirm that the broker is running and that the version has changed. After starting the broker, open the **<broker_instance_dir>/log/artemis.log** file. Find two lines similar to the ones below. Note the new version number that appears in the log when the broker is live.

```
INFO  [org.apache.activemq.artemis.core.server] AMQ221007: Server is now live
...
INFO  [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis
Message Broker version 2.9.0.redhat-00001 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-
54ee759954c4]
```

Additional Resources

- For more information about creating an instance of the broker, see Creating a broker instance .

- You can now store a broker instance's configuration files and data in any custom directory, including locations outside of the broker instance's directory. In the **<broker_instance_dir>/etc/artemis.profile** file, update the **ARTEMIS_INSTANCE_ETC_URI** property by specifying the location of the custom directory after creating the broker instance. Previously, these configuration files and data could only be stored in the **etc/** and **data/** directories within the broker instance's directory.

2.2.5.2. Upgrading from 7.3.0 to 7.4.0 on Windows

Procedure

1. Download the desired archive from the Red Hat Customer Portal. Follow the instructions provided in Downloading the AMQ Broker archive .

2. Use a file manager to move the archive to the folder you created during the last installation of AMQ Broker.

3. Extract the contents of the archive. Right-click the .zip file and select **Extract All**.

4. If the broker is running, stop it.

   > *<broker_instance_dir>*\bin\artemis-service.exe stop

5. Back up the broker using a file manager.

   a. Right-click the **<broker_instance_dir>** folder and select **Copy**.

   b. Right-click in the same window and select **Paste**.

6. (Optional) Note the current version of the broker. After the broker stops, you see a line similar to the one below at the end of the **<broker_instance_dir>\log\artemis.log** file.

   > INFO [org.apache.activemq.artemis.core.server] AMQ221002: Apache ActiveMQ Artemis
   > Message Broker version 2.7.0.redhat-00054 [4782d50d-47a2-11e7-a160-9801a793ea45]
   > stopped, uptime 28 minutes

7. Edit the **<broker_instance_dir>\etc\artemis.profile.cmd** and **<broker_instance_dir>\bin\artemis-service.xml** configuration files. Set the **ARTEMIS_HOME** property to the new directory created when the archive was extracted.

   > ARTEMIS_HOME=*<install_dir>*

8. Edit the **<broker_instance_dir>\etc\artemis.profile.cmd** configuration file. Set the JAVA_ARGS environment variable to reference the correct log manager version and dependent file.

   > JAVA_ARGS=-Xbootclasspath/%ARTEMIS_HOME%\lib\jboss-logmanager-2.1.10.Final-redhat-00001.jar;%ARTEMIS_HOME%\lib\wildfly-common-1.5.1.Final-redhat-00001.jar

9. Edit the **<broker_instance_dir>\bin\artemis-service.xml** configuration file. Set the bootstrap class path start argument to reference the correct log manager version and dependent file.

```
<startargument>-Xbootclasspath/a:%ARTEMIS_HOME%\lib\jboss-logmanager-2.1.10.Final-
redhat-00001.jar;%ARTEMIS_HOME%\lib\wildfly-common-1.5.1.Final-redhat-
00001.jar</startargument>
```

10. Edit the **<broker_instance_dir>\etc\bootstrap.xml** configuration file. In the **<web>** configuration element, add a reference to the metrics plugin file for AMQ Broker.

```
<app url="metrics" war="metrics.war"/>
```

11. Start the upgraded broker.

```
<broker_instance_dir>\bin\artemis-service.exe start
```

12. (Optional) Confirm that the broker is running and that the version has changed. After starting the broker, open the **<broker_instance_dir>\log\artemis.log** file. Find two lines similar to the ones below. Note the new version number that appears in the log when the broker is live.

```
INFO  [org.apache.activemq.artemis.core.server] AMQ221007: Server is now live
...
INFO  [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis
Message Broker version 2.9.0.redhat-00001 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-
54ee759954c4]
```

Additional Resources

- For more information about creating an instance of the broker, see Creating a broker instance .

- You can now store a broker instance's configuration files and data in any custom directory, including locations outside of the broker instance's directory. In the **<broker_instance_dir>\etc\artemis.profile** file, update the **ARTEMIS_INSTANCE_ETC_URI** property by specifying the location of the custom directory after creating the broker instance. Previously, these configuration files and data could only be stored in the **\etc** and **\data** directories within the broker instance's directory.

2.3. UPGRADING A BROKER INSTANCE FROM 7.4.0 TO 7.4.X

IMPORTANT

AMQ Broker 7.4 has been designated as a Long Term Support (LTS) release version. Bug fixes and security advisories will be made available for AMQ Broker 7.4 in a series of micro releases (7.4.1, 7.4.2, and so on) for a period of at least 12 months. This means that you will be able to get recent bug fixes and security advisories for AMQ Broker without having to upgrade to a new minor release. For more information, see Long Term Support for AMQ Broker.

IMPORTANT

Starting with AMQ Broker 7.1.0, you can access AMQ Management Console only from the local host by default. To learn about configuring remote access to the console, see Configuring local and remote access to AMQ Management Console .

The following subsections describe how to upgrade a 7.4.0 broker instance to 7.4.x for different operating systems.

- Upgrading from 7.4.0 to 7.4.x on Linux

- Upgrading from 7.4.0 to 7.4.x on Windows

2.3.1. Upgrading from 7.4.0 to 7.4.x on Linux

NOTE

The name of the archive that you download could differ from what is used in the following examples.

Procedure

1. Download the desired archive from the Red Hat Customer Portal. Follow the instructions provided in Downloading the AMQ Broker archive .

2. Change the owner of the archive to the same user that owns the AMQ Broker installation to be upgraded. The following example shows a user called **amq-broker**.

   ```
   sudo chown amq-broker:amq-broker amq-broker-7.4.x.redhat-1.zip
   ```

3. Move the archive to the directory created during the original installation of AMQ Broker. The following example uses /**opt**/**redhat**.

   ```
   sudo mv amq-broker-7.4.x.redhat-1.zip /opt/redhat
   ```

4. As the directory owner, extract the contents of the compressed archive. In the following example, the user **amq-broker** extracts the archive using the **unzip** command.

   ```
   su - amq-broker
   cd /opt/redhat
   unzip amq-broker-7.4.x.redhat-1.zip
   ```

5. If the broker is running, stop it.

   ```
   <broker_instance_dir>/bin/artemis stop
   ```

6. Back up the instance directory of the broker by copying it to the home directory of the current user.

   ```
   cp -r <broker_instance_dir> ~/
   ```

7. (Optional) Note the current version of the broker. After the broker stops, you see a line similar to the one below at the end of the **<broker_instance_dir>**/**log**/**artemis.log** file.

   ```
   INFO  [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis
   Message Broker version 2.7.0.redhat-00054 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-
   54ee759954c4]
   ```

8. Edit the *<broker_instance_dir>*/etc/artemis.profile configuration file. Set the ARTEMIS_HOME property to the new directory created when the archive was extracted.

   ARTEMIS_HOME='/opt/redhat/amq-broker-7.4.x-redhat-1'

9. Start the upgraded broker.

   <broker_instance_dir>/bin/artemis run

10. (Optional) Confirm that the broker is running and that the version has changed. After starting the broker, open the *<broker_instance_dir>*/log/artemis.log file. Find two lines similar to the ones below. Note the new version number that appears in the log when the broker is live.

    INFO [org.apache.activemq.artemis.core.server] AMQ221007: Server is now live
    ...
    INFO [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis Message Broker version 2.9.0.redhat-00001 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-54ee759954c4]

Additional Resources

- For more information about creating an instance of the broker, see Creating a broker instance.

- You can now store a broker instance's configuration files and data in any custom directory, including locations outside of the broker instance's directory. In the *<broker_instance_dir>*/etc/artemis.profile file, update the ARTEMIS_INSTANCE_ETC_URI property by specifying the location of the custom directory after creating the broker instance. Previously, these configuration files and data could only be stored in the etc/ and data/ directories within the broker instance's directory.

2.3.2. Upgrading from 7.4.0 to 7.4.x on Windows

Procedure

1. Download the desired archive from the Red Hat Customer Portal. Follow the instructions provided in Downloading the AMQ Broker archive.

2. Use a file manager to move the archive to the folder you created during the last installation of AMQ Broker.

3. Extract the contents of the archive. Right-click the .zip file and select **Extract All**.

4. If the broker is running, stop it.

   <broker_instance_dir>\bin\artemis-service.exe stop

5. Back up the broker using a file manager.

   a. Right-click the *<broker_instance_dir>* folder and select **Copy**.

   b. Right-click in the same window and select **Paste**.

6. (Optional) Note the current version of the broker. After the broker stops, you see a line similar to the one below at the end of the *<broker_instance_dir>*\log\artemis.log file.

> INFO [org.apache.activemq.artemis.core.server] AMQ221002: Apache ActiveMQ Artemis Message Broker version 2.7.0.redhat-00054 [4782d50d-47a2-11e7-a160-9801a793ea45] stopped, uptime 28 minutes

7. Edit the **<broker_instance_dir>\etc\artemis.profile.cmd** and **<broker_instance_dir>\bin\artemis-service.xml** configuration files. Set the **ARTEMIS_HOME** property to the new directory created when the archive was extracted.

   > ARTEMIS_HOME=*<install_dir>*

8. Start the upgraded broker.

   > *<broker_instance_dir>*\bin\artemis-service.exe start

9. (Optional) Confirm that the broker is running and that the version has changed. After starting the broker, open the **<broker_instance_dir>\log\artemis.log** file. Find two lines similar to the ones below. Note the new version number that appears in the log when the broker is live.

   > INFO [org.apache.activemq.artemis.core.server] AMQ221007: Server is now live
   >
   > ...
   >
   > INFO [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis Message Broker version 2.9.0.redhat-00001 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-54ee759954c4]

Additional Resources

- For more information about creating an instance of the broker, see Creating a broker instance .

- You can now store a broker instance's configuration files and data in any custom directory, including locations outside of the broker instance's directory. In the **<broker_instance_dir>\etc\artemis.profile** file, update the **ARTEMIS_INSTANCE_ETC_URI** property by specifying the location of the custom directory after creating the broker instance. Previously, these configuration files and data could only be stored in the **\etc** and **\data** directories within the broker instance's directory.

2.4. UPGRADING A BROKER INSTANCE FROM 7.4.X TO 7.5.0

The following subsections describe how to upgrade a 7.4.x broker instance to 7.5.0 for different operating systems.
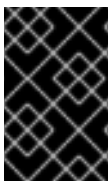
IMPORTANT

Starting with AMQ Broker 7.1.0, you can access AMQ Management Console only from the local host by default. To learn about configuring remote access to the console, see Configuring local and remote access to AMQ Management Console .

- Upgrading from 7.4.x to 7.5.0 on Linux

- Upgrading from 7.4.x to 7.5.0 on Windows

2.4.1. Upgrading from 7.4.x to 7.5.0 on Linux
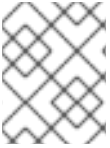
NOTE

The name of the archive that you download could differ from what is used in the following examples.

Procedure

1. Download the desired archive from the Red Hat Customer Portal. Follow the instructions provided in Downloading the AMQ Broker archive.

2. Change the owner of the archive to the same user that owns the AMQ Broker installation to be upgraded. The following example shows a user called **amq-broker**.

   ```
   sudo chown amq-broker:amq-broker amq-broker-7.5.0.redhat-1.zip
   ```

3. Move the archive to the directory created during the original installation of AMQ Broker. The following example uses **/opt/redhat**.

   ```
   sudo mv amq-broker-7.5.0.redhat-1.zip /opt/redhat
   ```

4. As the directory owner, extract the contents of the compressed archive. In the following example, the user **amq-broker** extracts the archive using the **unzip** command.

   ```
   su - amq-broker
   cd /opt/redhat
   unzip amq-broker-7.5.0.redhat-1.zip
   ```

5. If the broker is running, stop it.

   ```
   <broker_instance_dir>/bin/artemis stop
   ```

6. Back up the instance directory of the broker by copying it to the home directory of the current user.

   ```
   cp -r <broker_instance_dir> ~/
   ```

7. (Optional) Note the current version of the broker. After the broker stops, you see a line similar to the one below at the end of the **<broker_instance_dir>/log/artemis.log** file.

   ```
   INFO  [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis
   Message Broker version 2.7.0.redhat-00054 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-
   54ee759954c4]
   ```

8. Edit the **<broker_instance_dir>/etc/artemis.profile** configuration file.

   a. Set the **ARTEMIS_HOME** property to the new directory created when the archive was extracted.

      ```
      ARTEMIS_HOME='/opt/redhat/amq-broker-7.5.0-redhat-1'
      ```

   b. Edit the **JAVA_ARGS** property. Add the bootstrap class path argument, which references a dependent file for the log manager.

```
-Xbootclasspath/a:$ARTEMIS_HOME/lib/wildfly-common-1.5.2.Final-redhat-00001.jar
```

9. Start the upgraded broker.

   ```
   <broker_instance_dir>/bin/artemis run
   ```

10. (Optional) Confirm that the broker is running and that the version has changed. After starting the broker, open the **<broker_instance_dir>/log/artemis.log** file. Find two lines similar to the ones below. Note the new version number that appears in the log when the broker is live.

    ```
    INFO  [org.apache.activemq.artemis.core.server] AMQ221007: Server is now live
    ...
    INFO  [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis
    Message Broker version 2.9.0.redhat-00001 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-
    54ee759954c4]
    ```

Additional Resources

- For more information about creating an instance of the broker, see Creating a broker instance.

- You can now store a broker instance's configuration files and data in any custom directory, including locations outside of the broker instance's directory. In the **<broker_instance_dir>/etc/artemis.profile** file, update the **ARTEMIS_INSTANCE_ETC_URI** property by specifying the location of the custom directory after creating the broker instance. Previously, these configuration files and data could only be stored in the **etc/** and **data/** directories within the broker instance's directory.

2.4.2. Upgrading from 7.4.x to 7.5.0 on Windows

Procedure

1. Download the desired archive from the Red Hat Customer Portal. Follow the instructions provided in Downloading the AMQ Broker archive.

2. Use a file manager to move the archive to the folder you created during the last installation of AMQ Broker.

3. Extract the contents of the archive. Right-click the .zip file and select **Extract All**.

4. If the broker is running, stop it.

   ```
   <broker_instance_dir>\bin\artemis-service.exe stop
   ```

5. Back up the broker using a file manager.

   a. Right-click the **<broker_instance_dir>** folder and select **Copy**.

   b. Right-click in the same window and select **Paste**.

6. (Optional) Note the current version of the broker. After the broker stops, you see a line similar to the one below at the end of the **<broker_instance_dir>\log\artemis.log** file.

> INFO [org.apache.activemq.artemis.core.server] AMQ221002: Apache ActiveMQ Artemis Message Broker version 2.7.0.redhat-00054 [4782d50d-47a2-11e7-a160-9801a793ea45] stopped, uptime 28 minutes

7. Edit the **<broker_instance_dir>\etc\artemis.profile.cmd** and **<broker_instance_dir>\bin\artemis-service.xml** configuration files. Set the **ARTEMIS_HOME** property to the new directory created when the archive was extracted.

   > ARTEMIS_HOME=<install_dir>

8. Edit the **<broker_instance_dir>\etc\artemis.profile.cmd** configuration file. Set the **JAVA_ARGS** environment variable to reference the correct log manager version and dependent file.

   > JAVA_ARGS=-Xbootclasspath/%ARTEMIS_HOME%\lib\jboss-logmanager-2.1.10.Final-redhat-00001.jar;%ARTEMIS_HOME%\lib\wildfly-common-1.5.2.Final-redhat-00001.jar

9. Edit the **<broker_instance_dir>\bin\artemis-service.xml** configuration file. Set the bootstrap class path start argument to reference the correct log manager version and dependent file.

   > <startargument>-Xbootclasspath/a:%ARTEMIS_HOME%\lib\jboss-logmanager-2.1.10.Final-redhat-00001.jar;%ARTEMIS_HOME%\lib\wildfly-common-1.5.2.Final-redhat-00001.jar</startargument>

10. Start the upgraded broker.

    > <broker_instance_dir>\bin\artemis-service.exe start

11. (Optional) Confirm that the broker is running and that the version has changed. After starting the broker, open the **<broker_instance_dir>\log\artemis.log** file. Find two lines similar to the ones below. Note the new version number that appears in the log when the broker is live.

    > INFO [org.apache.activemq.artemis.core.server] AMQ221007: Server is now live
    > ...
    > INFO [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis Message Broker version 2.9.0.redhat-00001 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-54ee759954c4]

Additional Resources

- For more information about creating an instance of the broker, see Creating a broker instance .

- You can now store a broker instance's configuration files and data in any custom directory, including locations outside of the broker instance's directory. In the **<broker_instance_dir>\etc\artemis.profile** file, update the **ARTEMIS_INSTANCE_ETC_URI** property by specifying the location of the custom directory after creating the broker instance. Previously, these configuration files and data could only be stored in the **\etc** and **\data** directories within the broker instance's directory.

2.5. UPGRADING A BROKER INSTANCE FROM 7.5.0 TO 7.6.0

The following subsections describe how to upgrade a 7.5.0 broker instance to 7.6.0 for different operating systems.

> **IMPORTANT**
>
> Starting with AMQ Broker 7.1.0, you can access AMQ Management Console only from the local host by default. To learn about configuring remote access to the console, see Configuring local and remote access to AMQ Management Console .

- Upgrading from 7.5.0 to 7.6.0 on Linux

- Upgrading from 7.5.0 to 7.6.0 on Windows

2.5.1. Upgrading from 7.5.0 to 7.6.0 on Linux

> **NOTE**
>
> The name of the archive that you download could differ from what is used in the following examples.

Procedure

1. Download the desired archive from the Red Hat Customer Portal. Follow the instructions provided in Downloading the AMQ Broker archive .

2. Change the owner of the archive to the same user that owns the AMQ Broker installation to be upgraded. The following example shows a user called **amq-broker**.

   ```
   sudo chown amq-broker:amq-broker amq-broker-7.6.0.redhat-1.zip
   ```

3. Move the archive to the directory created during the original installation of AMQ Broker. The following example uses **/opt/redhat**.

   ```
   sudo mv amq-broker-7.6.0.redhat-1.zip /opt/redhat
   ```

4. As the directory owner, extract the contents of the compressed archive. In the following example, the user **amq-broker** extracts the archive using the **unzip** command.

   ```
   su - amq-broker
   cd /opt/redhat
   unzip amq-broker-7.6.0.redhat-1.zip
   ```

5. If the broker is running, stop it.

   ```
   <broker_instance_dir>/bin/artemis stop
   ```

6. Back up the instance directory of the broker by copying it to the home directory of the current user.

   ```
   cp -r <broker_instance_dir> ~/
   ```

7. (Optional) Note the current version of the broker. After the broker stops, you see a line similar to the one below at the end of the **<broker_instance_dir>/log/artemis.log** file.

> INFO [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis Message Broker version 2.9.0.redhat-00054 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-54ee759954c4]

8. Edit the ***<broker_instance_dir>*/etc/artemis.profile** configuration file.

    a. Set the **ARTEMIS_HOME** property to the new directory created when the archive was extracted.

    > ARTEMIS_HOME='/opt/redhat/amq-broker-7.6.0-redhat-1'

    b. Edit the **JAVA_ARGS** property. Add the bootstrap class path argument, which references a dependent file for the log manager.

    > -Xbootclasspath/a:$ARTEMIS_HOME/lib/wildfly-common-1.5.2.Final-redhat-00002.jar

9. Start the upgraded broker.

    > *<broker_instance_dir>*/bin/artemis run

10. (Optional) Confirm that the broker is running and that the version has changed. After starting the broker, open the ***<broker_instance_dir>*/log/artemis.log** file. Find two lines similar to the ones below. Note the new version number that appears in the log when the broker is live.

    > INFO [org.apache.activemq.artemis.core.server] AMQ221007: Server is now live
    > ...
    > INFO [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis Message Broker version 2.11.0.redhat-00001 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-54ee759954c4]

Additional Resources

- For more information about creating an instance of the broker, see Creating a broker instance.

- You can now store a broker instance's configuration files and data in any custom directory, including locations outside of the broker instance's directory. In the ***<broker_instance_dir>*/etc/artemis.profile** file, update the **ARTEMIS_INSTANCE_ETC_URI** property by specifying the location of the custom directory after creating the broker instance. Previously, these configuration files and data could only be stored in the **etc/** and **data/** directories within the broker instance's directory.

2.5.2. Upgrading from 7.5.0 to 7.6.0 on Windows

Procedure

1. Download the desired archive from the Red Hat Customer Portal. Follow the instructions provided in Downloading the AMQ Broker archive.

2. Use a file manager to move the archive to the folder you created during the last installation of AMQ Broker.

3. Extract the contents of the archive. Right-click the .zip file and select **Extract All**.

4. If the broker is running, stop it.

> *<broker_instance_dir>*\bin\artemis-service.exe stop

5. Back up the broker using a file manager.

   a. Right-click the ***<broker_instance_dir>*** folder and select **Copy**.

   b. Right-click in the same window and select **Paste**.

6. (Optional) Note the current version of the broker. After the broker stops, you see a line similar to the one below at the end of the ***<broker_instance_dir>*\log\artemis.log** file.

> INFO [org.apache.activemq.artemis.core.server] AMQ221002: Apache ActiveMQ Artemis Message Broker version 2.9.0.redhat-00054 [4782d50d-47a2-11e7-a160-9801a793ea45] stopped, uptime 28 minutes

7. Edit the ***<broker_instance_dir>*\etc\artemis.profile.cmd** and ***<broker_instance_dir>*\bin\artemis-service.xml** configuration files. Set the **ARTEMIS_HOME** property to the new directory created when the archive was extracted.

> ARTEMIS_HOME=*<install_dir>*

8. Edit the ***<broker_instance_dir>*\etc\artemis.profile.cmd** configuration file. Set the **JAVA_ARGS** environment variable to reference the correct log manager version and dependent file.

> JAVA_ARGS=-Xbootclasspath/%ARTEMIS_HOME%\lib\jboss-logmanager-2.1.10.Final-redhat-00001.jar;%ARTEMIS_HOME%\lib\wildfly-common-1.5.2.Final-redhat-00002.jar

9. Edit the ***<broker_instance_dir>*\bin\artemis-service.xml** configuration file. Set the bootstrap class path start argument to reference the correct log manager version and dependent file.

> <startargument>-Xbootclasspath/a:%ARTEMIS_HOME%\lib\jboss-logmanager-2.1.10.Final-redhat-00001.jar;%ARTEMIS_HOME%\lib\wildfly-common-1.5.2.Final-redhat-00002.jar</startargument>

10. Start the upgraded broker.

> *<broker_instance_dir>*\bin\artemis-service.exe start

11. (Optional) Confirm that the broker is running and that the version has changed. After starting the broker, open the ***<broker_instance_dir>*\log\artemis.log** file. Find two lines similar to the ones below. Note the new version number that appears in the log when the broker is live.

> INFO [org.apache.activemq.artemis.core.server] AMQ221007: Server is now live
>
> ...
> INFO [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis Message Broker version 2.11.0.redhat-00001 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-54ee759954c4]

Additional Resources

- For more information about creating an instance of the broker, see Creating a broker instance .

- You can now store a broker instance's configuration files and data in any custom directory, including locations outside of the broker instance's directory. In the **<broker_instance_dir>\etc\artemis.profile** file, update the **ARTEMIS_INSTANCE_ETC_URI** property by specifying the location of the custom directory after creating the broker instance. Previously, these configuration files and data could only be stored in the **\etc** and **\data** directories within the broker instance's directory.

2.6. UPGRADING A BROKER INSTANCE FROM 7.6.0 TO 7.7.0

The following subsections describe how to upgrade a 7.6.0 broker instance to 7.7.0 for different operating systems.

> **IMPORTANT**
>
> Starting with AMQ Broker 7.1.0, you can access AMQ Management Console only from the local host by default. To learn about configuring remote access to the console, see Configuring local and remote access to AMQ Management Console .

- Upgrading from 7.6.0 to 7.7.0 on Linux

- Upgrading from 7.6.0 to 7.7.0 on Windows

2.6.1. Upgrading from 7.6.0 to 7.7.0 on Linux

> **NOTE**
>
> The name of the archive that you download could differ from what is used in the following examples.

Procedure

1. Download the desired archive from the Red Hat Customer Portal. Follow the instructions provided in Downloading the AMQ Broker archive .

2. Change the owner of the archive to the same user that owns the AMQ Broker installation to be upgraded. The following example shows a user called **amq-broker**.

   ```
   sudo chown amq-broker:amq-broker amq-broker-7.7.0.redhat-1.zip
   ```

3. Move the archive to the directory created during the original installation of AMQ Broker. The following example uses **/opt/redhat**.

   ```
   sudo mv amq-broker-7.7.0.redhat-1.zip /opt/redhat
   ```

4. As the directory owner, extract the contents of the compressed archive. In the following example, the user **amq-broker** extracts the archive using the **unzip** command.

   ```
   su - amq-broker
   cd /opt/redhat
   unzip amq-broker-7.7.0.redhat-1.zip
   ```

5. If the broker is running, stop it.

   <broker_instance_dir>/bin/artemis stop

6. Back up the instance directory of the broker by copying it to the home directory of the current user.

   cp -r *<broker_instance_dir>* ~/

7. (Optional) Note the current version of the broker. After the broker stops, you see a line similar to the one below at the end of the *<broker_instance_dir>*/**log**/**artemis.log** file.

   INFO [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis Message Broker version 2.11.0.redhat-00001 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-54ee759954c4]

8. Edit the *<broker_instance_dir>*/**etc**/**artemis.profile** configuration file.

   a. Set the **ARTEMIS_HOME** property to the new directory created when the archive was extracted. For example:

      ARTEMIS_HOME='/opt/redhat/amq-broker-7.7.0-redhat-1'

   b. Locate the **JAVA_ARGS** property. Ensure that the bootstrap class path argument references the required version of a dependent file for the log manager, as shown below.

      -Xbootclasspath/a:$ARTEMIS_HOME/lib/wildfly-common-1.5.2.Final-redhat-00002.jar

9. Edit the *<broker_instance_dir>*/**etc**/**logging.properties** configuration file.

   a. On the list of additional loggers to be configured, include the **org.apache.activemq.audit.resource** resource logger that was added in AMQ Broker 7.7.0.

      loggers=org.eclipse.jetty,org.jboss.logging,org.apache.activemq.artemis.core.server,org.apache.activemq.artemis.utils,org.apache.activemq.artemis.journal,org.apache.activemq.artemis.jms.server,org.apache.activemq.artemis.integration.bootstrap,org.apache.activemq.audit.base,org.apache.activemq.audit.message,**org.apache.activemq.audit.resource**

   b. Before the **Console handler configuration** section, add a default configuration for the resource logger.

      ..

      logger.org.apache.activemq.audit.resource.level=ERROR
      logger.org.apache.activemq.audit.resource.handlers=AUDIT_FILE
      logger.org.apache.activemq.audit.resource.useParentHandlers=false

      # Console handler configuration
      ..

10. Start the upgraded broker.

> *<broker_instance_dir>*/bin/artemis run

11. (Optional) Confirm that the broker is running and that the version has changed. After starting the broker, open the ***<broker_instance_dir>*/log/artemis.log** file. Find two lines similar to the ones below. Note the new version number that appears in the log when the broker is live.

> INFO [org.apache.activemq.artemis.core.server] AMQ221007: Server is now live
>
> ...
>
> INFO [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis
> Mesq.audit.resource.handlers=AUDIT_FILE
> logger.org.apache.activemq.audit.resource.useParentHandlers=false
> sage Broker version 2.13.0.redhat-00003 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-54ee759954c4]

Additional Resources

- For more information about creating an instance of the broker, see Creating a broker instance .

- You can now store a broker instance's configuration files and data in any custom directory, including locations outside of the broker instance's directory. In the ***<broker_instance_dir>*/etc/artemis.profile** file, update the **ARTEMIS_INSTANCE_ETC_URI** property by specifying the location of the custom directory after creating the broker instance. Previously, these configuration files and data could only be stored in the **etc/** and **data/** directories within the broker instance's directory.

2.6.2. Upgrading from 7.6.0 to 7.7.0 on Windows

Procedure

1. Download the desired archive from the Red Hat Customer Portal. Follow the instructions provided in Downloading the AMQ Broker archive .

2. Use a file manager to move the archive to the folder you created during the last installation of AMQ Broker.

3. Extract the contents of the archive. Right-click the .zip file and select **Extract All**.

4. If the broker is running, stop it.

> *<broker_instance_dir>*\bin\artemis-service.exe stop

5. Back up the broker using a file manager.

   a. Right-click the ***<broker_instance_dir>*** folder and select **Copy**.

   b. Right-click in the same window and select **Paste**.

6. (Optional) Note the current version of the broker. After the broker stops, you see a line similar to the one below at the end of the ***<broker_instance_dir>*\log\artemis.log** file.

> INFO [org.apache.activemq.artemis.core.server] AMQ221002: Apache ActiveMQ Artemis
> Message Broker version 2.11.0.redhat-00001 [4782d50d-47a2-11e7-a160-9801a793ea45]
> stopped, uptime 28 minutes

7. Edit the ***<broker_instance_dir>*\etc\artemis.profile.cmd** and
***<broker_instance_dir>*\bin\artemis-service.xml** configuration files. Set the **ARTEMIS_HOME**
property to the new directory created when the archive was extracted.

> ARTEMIS_HOME=*<install_dir>*

8. Edit the ***<broker_instance_dir>*\etc\artemis.profile.cmd** configuration file. Ensure that the
JAVA_ARGS environment variable references the correct versions for the log manager and
dependent file, as shown below.

> JAVA_ARGS=-Xbootclasspath/%ARTEMIS_HOME%\lib\jboss-logmanager-2.1.10.Final-
> redhat-00001.jar;%ARTEMIS_HOME%\lib\wildfly-common-1.5.2.Final-redhat-00002.jar

9. Edit the ***<broker_instance_dir>*\bin\artemis-service.xml** configuration file. Ensure that the
bootstrap class path start argument references the correct versions for the log manager and
dependent file, as shown below.

> <startargument>-Xbootclasspath/a:%ARTEMIS_HOME%\lib\jboss-logmanager-2.1.10.Final-
> redhat-00001.jar;%ARTEMIS_HOME%\lib\wildfly-common-1.5.2.Final-redhat-
> 00002.jar</startargument>

10. Edit the ***<broker_instance_dir>*\etc\logging.properties** configuration file.

    a. On the list of additional loggers to be configured, include the
    org.apache.activemq.audit.resource resource logger that was added in AMQ Broker
    7.7.0.

    > loggers=org.eclipse.jetty,org.jboss.logging,org.apache.activemq.artemis.core.server,org.ap
    > ache.activemq.artemis.utils,org.apache.activemq.artemis.journal,org.apache.activemq.arte
    > mis.jms.server,org.apache.activemq.artemis.integration.bootstrap,org.apache.activemq.aud
    > it.base,org.apache.activemq.audit.message,**org.apache.activemq.audit.resource**

    b. Before the **Console handler configuration** section, add a default configuration for the
    resource logger.

    > ..
    >
    > logger.org.apache.activemq.audit.resource.level=ERROR
    > logger.org.apache.activemq.audit.resource.handlers=AUDIT_FILE
    > logger.org.apache.activemq.audit.resource.useParentHandlers=false
    >
    > # Console handler configuration
    > ..

11. Start the upgraded broker.

> *<broker_instance_dir>*\bin\artemis-service.exe start

12. (Optional) Confirm that the broker is running and that the version has changed. After starting
the broker, open the ***<broker_instance_dir>*\log\artemis.log** file. Find two lines similar to the
ones below. Note the new version number that appears in the log when the broker is live.

> INFO [org.apache.activemq.artemis.core.server] AMQ221007: Server is now live
> ...

> INFO [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis Message Broker version 2.13.0.redhat-00003 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-54ee759954c4]

Additional Resources

- For more information about creating an instance of the broker, see Creating a broker instance.

- You can now store a broker instance's configuration files and data in any custom directory, including locations outside of the broker instance's directory. In the ***<broker_instance_dir>*\etc\artemis.profile** file, update the **ARTEMIS_INSTANCE_ETC_URI** property by specifying the location of the custom directory after creating the broker instance. Previously, these configuration files and data could only be stored in the **\etc** and **\data** directories within the broker instance's directory.

2.7. UPGRADING A BROKER INSTANCE FROM 7.7.0 TO 7.8.0

The following subsections describe how to upgrade a 7.7.0 broker instance to 7.8.0 for different operating systems.

> **IMPORTANT**
>
> Starting with AMQ Broker 7.1.0, you can access AMQ Management Console only from the local host by default. To learn about configuring remote access to the console, see Configuring local and remote access to AMQ Management Console .

- Upgrading from 7.7.0 to 7.8.0 on Linux

- Upgrading from 7.7.0 to 7.8.0 on Windows

2.7.1. Upgrading from 7.7.0 to 7.8.0 on Linux

> **NOTE**
>
> The name of the archive that you download could differ from what is used in the following examples.

Procedure

1. Download the desired archive from the Red Hat Customer Portal. Follow the instructions provided in Downloading the AMQ Broker archive.

2. Change the owner of the archive to the same user that owns the AMQ Broker installation to be upgraded. The following example shows a user called **amq-broker**.

   > sudo chown amq-broker:amq-broker amq-broker-7.8.0.redhat-1.zip

3. Move the archive to the directory created during the original installation of AMQ Broker. The following example uses **/opt/redhat**.

   > sudo mv amq-broker-7.8.0.redhat-1.zip /opt/redhat

4. As the directory owner, extract the contents of the compressed archive. In the following example, the user **amq-broker** extracts the archive using the **unzip** command.

```
su - amq-broker
cd /opt/redhat
unzip amq-broker-7.8.0.redhat-1.zip
```

5. If the broker is running, stop it.

```
<broker_instance_dir>/bin/artemis stop
```

6. Back up the instance directory of the broker by copying it to the home directory of the current user.

```
cp -r <broker_instance_dir> ~/
```

7. (Optional) Note the current version of the broker. After the broker stops, you see a line similar to the one below at the end of the **<broker_instance_dir>/log/artemis.log** file.

```
INFO  [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis
Message Broker version 2.13.0.redhat-00003 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-
54ee759954c4]
```

8. Edit the **<broker_instance_dir>/etc/artemis.profile** configuration file.

   a. Set the **ARTEMIS_HOME** property to the new directory created when the archive was extracted. For example:

   ```
   ARTEMIS_HOME='/opt/redhat/amq-broker-7.8.0-redhat-1'
   ```

   b. Locate the **JAVA_ARGS** property. Ensure that the bootstrap class path argument references the required version of a dependent file for the log manager, as shown below.

   ```
   -Xbootclasspath/a:$ARTEMIS_HOME/lib/wildfly-common-1.5.2.Final-redhat-00002.jar
   ```

9. Edit the **<broker_instance_dir>/etc/bootstrap.xml** configuration file. In the **web** element, update the name of the **.war** file required by AMQ Management Console in 7.8.

```
<web bind="http://localhost:8161" path="web">
   ...
   <app url="console" war="hawtio.war"/>
   ...
</web>
```

10. Start the upgraded broker.

```
<broker_instance_dir>/bin/artemis run
```

11. (Optional) Confirm that the broker is running and that the version has changed. After starting the broker, open the **<broker_instance_dir>/log/artemis.log** file. Find two lines similar to the ones below. Note the new version number that appears in the log when the broker is live.

```
INFO  [org.apache.activemq.artemis.core.server] AMQ221007: Server is now live
```

```
...
INFO  [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis
Mesq.audit.resource.handlers=AUDIT_FILE
logger.org.apache.activemq.audit.resource.useParentHandlers=false
sage Broker version 2.16.0.redhat-00007 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-
54ee759954c4]
```

Additional Resources

- For more information about creating an instance of the broker, see Creating a broker instance .

- You can now store a broker instance's configuration files and data in any custom directory, including locations outside of the broker instance's directory. In the *<broker_instance_dir>*/**etc**/**artemis.profile** file, update the **ARTEMIS_INSTANCE_ETC_URI** property by specifying the location of the custom directory after creating the broker instance. Previously, these configuration files and data could only be stored in the **etc/** and **data/** directories within the broker instance's directory.

2.7.2. Upgrading from 7.7.0 to 7.8.0 on Windows

Procedure

1. Download the desired archive from the Red Hat Customer Portal. Follow the instructions provided in Downloading the AMQ Broker archive .

2. Use a file manager to move the archive to the folder you created during the last installation of AMQ Broker.

3. Extract the contents of the archive. Right-click the .zip file and select **Extract All**.

4. If the broker is running, stop it.

   ```
   <broker_instance_dir>\bin\artemis-service.exe stop
   ```

5. Back up the broker using a file manager.

   a. Right-click the *<broker_instance_dir>* folder and select **Copy**.

   b. Right-click in the same window and select **Paste**.

6. (Optional) Note the current version of the broker. After the broker stops, you see a line similar to the one below at the end of the *<broker_instance_dir>***log****artemis.log** file.

   ```
   INFO  [org.apache.activemq.artemis.core.server] AMQ221002: Apache ActiveMQ Artemis
   Message Broker version 2.13.0.redhat-00003 [4782d50d-47a2-11e7-a160-9801a793ea45]
   stopped, uptime 28 minutes
   ```

7. Edit the *<broker_instance_dir>***etc****artemis.profile.cmd** and *<broker_instance_dir>***bin****artemis-service.xml** configuration files. Set the **ARTEMIS_HOME** property to the new directory created when the archive was extracted.

   ```
   ARTEMIS_HOME=<install_dir>
   ```

8. Edit the **<broker_instance_dir>\etc\artemis.profile.cmd** configuration file. Ensure that the **JAVA_ARGS** environment variable references the correct versions for the log manager and dependent file, as shown below.

JAVA_ARGS=-Xbootclasspath/%ARTEMIS_HOME%\lib\jboss-logmanager-2.1.10.Final-redhat-00001.jar;%ARTEMIS_HOME%\lib\wildfly-common-1.5.2.Final-redhat-00002.jar

9. Edit the **<broker_instance_dir>\bin\artemis-service.xml** configuration file. Ensure that the bootstrap class path start argument references the correct versions for the log manager and dependent file, as shown below.

<startargument>-Xbootclasspath/a:%ARTEMIS_HOME%\lib\jboss-logmanager-2.1.10.Final-redhat-00001.jar;%ARTEMIS_HOME%\lib\wildfly-common-1.5.2.Final-redhat-00002.jar</startargument>

10. Edit the **<broker_instance_dir>\etc\bootstrap.xml** configuration file. In the **web** element, update the name of the **.war** file required by AMQ Management Console in 7.8.

```
<web bind="http://localhost:8161" path="web">
    ...
    <app url="console" war="hawtio.war"/>
    ...
</web>
```

11. Start the upgraded broker.

<broker_instance_dir>\bin\artemis-service.exe start

12. (Optional) Confirm that the broker is running and that the version has changed. After starting the broker, open the **<broker_instance_dir>\log\artemis.log** file. Find two lines similar to the ones below. Note the new version number that appears in the log when the broker is live.

INFO [org.apache.activemq.artemis.core.server] AMQ221007: Server is now live

...

INFO [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis Message Broker version 2.16.0.redhat-00007 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-54ee759954c4]

Additional Resources

- For more information about creating an instance of the broker, see Creating a broker instance.

- You can now store a broker instance's configuration files and data in any custom directory, including locations outside of the broker instance's directory. In the **<broker_instance_dir>\etc\artemis.profile** file, update the **ARTEMIS_INSTANCE_ETC_URI** property by specifying the location of the custom directory after creating the broker instance. Previously, these configuration files and data could only be stored in the **\etc** and **\data** directories within the broker instance's directory.

2.8. UPGRADING A BROKER INSTANCE FROM 7.8.X TO 7.9.X

The following subsections describe how to upgrade a 7.8.x broker instance to 7.9.x for different operating systems.

> **IMPORTANT**
>
> Starting with AMQ Broker 7.1.0, you can access AMQ Management Console only from the local host by default. To learn about configuring remote access to the console, see Configuring local and remote access to AMQ Management Console .

- Upgrading from 7.8.x to 7.9.x on Linux

- Upgrading from 7.8.x to 7.9.x on Windows

2.8.1. Upgrading from 7.8.x to 7.9.x on Linux

> **NOTE**
>
> The name of the archive that you download could differ from what is used in the following examples.

Procedure

1. Download the desired archive from the Red Hat Customer Portal. Follow the instructions provided in Downloading the AMQ Broker archive .

2. Change the owner of the archive to the same user that owns the AMQ Broker installation to be upgraded. The following example shows a user called **amq-broker**.

   ```
   sudo chown amq-broker:amq-broker amq-broker-7.x.x-bin.zip
   ```

3. Move the archive to the directory created during the original installation of AMQ Broker. The following example uses **/opt/redhat**.

   ```
   sudo mv amq-broker-7.x.x-bin.zip /opt/redhat
   ```

4. As the directory owner, extract the contents of the compressed archive. In the following example, the user **amq-broker** extracts the archive using the **unzip** command.

   ```
   su - amq-broker
   cd /opt/redhat
   unzip amq-broker-7.x.x-bin.zip
   ```

5. If the broker is running, stop it.

   ```
   <broker_instance_dir>/bin/artemis stop
   ```

6. Back up the instance directory of the broker by copying it to the home directory of the current user.

   ```
   cp -r <broker_instance_dir> ~/
   ```

7. (Optional) Note the current version of the broker. After the broker stops, you see a line similar to the one below at the end of the **<broker_instance_dir>/log/artemis.log** file.

> INFO [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis Message Broker version 2.13.0.redhat-00003 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-54ee759954c4]

8. Edit the ***<broker_instance_dir>*/etc/artemis.profile** configuration file.

    a. Set the **ARTEMIS_HOME** property to the new directory created when the archive was extracted. For example:

    > ARTEMIS_HOME='/opt/redhat/amq-broker-7.x.x-bin'

    b. Locate the **JAVA_ARGS** property. Ensure that the bootstrap class path argument references the required version of a dependent file for the log manager, as shown below.

    > -Xbootclasspath/a:$ARTEMIS_HOME/lib/wildfly-common-1.5.2.Final-redhat-00002.jar

9. Edit the ***<broker_instance_dir>*/etc/bootstrap.xml** configuration file. In the **web** element, update the name of the **.war** file required by AMQ Management Console in 7.9.

    ```
    <web bind="http://localhost:8161" path="web">
        ...
        <app url="console" war="hawtio.war"/>
        ...
    </web>
    ```

10. Start the upgraded broker.

    > *<broker_instance_dir>*/bin/artemis run

11. (Optional) Confirm that the broker is running and that the version has changed. After starting the broker, open the ***<broker_instance_dir>*/log/artemis.log** file. Find two lines similar to the ones below. Note the new version number that appears in the log when the broker is live.

    > INFO [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis Mes
    > INFO [org.apache.activemq.artemis.core.server] AMQ221007: Server is now live
    > ...
    > sage Broker version 2.18.0.redhat-00010 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-54ee759954c4]

Additional Resources

- For more information about creating an instance of the broker, see Creating a broker instance .

- You can now store a broker instance's configuration files and data in any custom directory, including locations outside of the broker instance's directory. In the ***<broker_instance_dir>*/etc/artemis.profile** file, update the **ARTEMIS_INSTANCE_ETC_URI** property by specifying the location of the custom directory after creating the broker instance. Previously, these configuration files and data could only be stored in the **etc/** and **data/** directories within the broker instance's directory.

2.8.2. Upgrading from 7.8.x to 7.9.x on Windows

Procedure

1. Download the desired archive from the Red Hat Customer Portal. Follow the instructions provided in Downloading the AMQ Broker archive.

2. Use a file manager to move the archive to the folder you created during the last installation of AMQ Broker.

3. Extract the contents of the archive. Right-click the .zip file and select **Extract All**.

4. If the broker is running, stop it.

   > *<broker_instance_dir>*\bin\artemis-service.exe stop

5. Back up the broker using a file manager.

   a. Right-click the ***<broker_instance_dir>*** folder amd select **Copy**.

   b. Right-click in the same window and select **Paste**.

6. (Optional) Note the current version of the broker. After the broker stops, you see a line similar to the one below at the end of the ***<broker_instance_dir>*\log\artemis.log** file.

   > INFO [org.apache.activemq.artemis.core.server] AMQ221002: Apache ActiveMQ Artemis Message Broker version 2.13.0.redhat-00003 [4782d50d-47a2-11e7-a160-9801a793ea45] stopped, uptime 28 minutes

7. Edit the ***<broker_instance_dir>*\etc\artemis.profile.cmd** and ***<broker_instance_dir>*\bin\artemis-service.xml** configuration files. Set the **ARTEMIS_HOME** property to the new directory created when the archive was extracted.

   > ARTEMIS_HOME=*<install_dir>*

8. Edit the ***<broker_instance_dir>*\etc\artemis.profile.cmd** configuration file. Ensure that the **JAVA_ARGS** environment variable references the correct versions for the log manager and dependent file, as shown below.

   > JAVA_ARGS=-Xbootclasspath/%ARTEMIS_HOME%\lib\jboss-logmanager-2.1.10.Final-redhat-00001.jar;%ARTEMIS_HOME%\lib\wildfly-common-1.5.2.Final-redhat-00002.jar

9. Edit the ***<broker_instance_dir>*\bin\artemis-service.xml** configuration file. Ensure that the bootstrap class path start argument references the correct versions for the log manager and dependent file, as shown below.

   > <startargument>-Xbootclasspath/a:%ARTEMIS_HOME%\lib\jboss-logmanager-2.1.10.Final-redhat-00001.jar;%ARTEMIS_HOME%\lib\wildfly-common-1.5.2.Final-redhat-00002.jar</startargument>

10. Edit the ***<broker_instance_dir>*\etc\bootstrap.xml** configuration file. In the **web** element, update the name of the **.war** file required by AMQ Management Console in 7.9.

    > <web bind="http://localhost:8161" path="web">
    >   ...
    >   <app url="console" war="hawtio.war"/>

> ...
> </web>

11. Start the upgraded broker.

    > *<broker_instance_dir>*\bin\artemis-service.exe start

12. (Optional) Confirm that the broker is running and that the version has changed. After starting the broker, open the *<broker_instance_dir>*\log\artemis.log file. Find two lines similar to the ones below. Note the new version number that appears in the log when the broker is live.

    > INFO [org.apache.activemq.artemis.core.server] AMQ221007: Server is now live
    >
    > ...
    > INFO [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis Message Broker version 2.18.0.redhat-00010 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-54ee759954c4]

Additional Resources

- For more information about creating an instance of the broker, see Creating a broker instance .

- You can now store a broker instance's configuration files and data in any custom directory, including locations outside of the broker instance's directory. In the *<broker_instance_dir>*\etc\artemis.profile file, update the **ARTEMIS_INSTANCE_ETC_URI** property by specifying the location of the custom directory after creating the broker instance. Previously, these configuration files and data could only be stored in the **\etc** and **\data** directories within the broker instance's directory.

2.9. UPGRADING A BROKER INSTANCE FROM 7.9.X TO 7.10.X

The following subsections describe how to upgrade a 7.9.x broker instance to 7.10.x for different operating systems.

> **IMPORTANT**
>
> Starting with AMQ Broker 7.1.0, you can access AMQ Management Console only from the local host by default. To learn about configuring remote access to the console, see Configuring local and remote access to AMQ Management Console .

- Upgrading from 7.9.x to 7.10.x on Linux

- Upgrading from 7.9.x to 7.10.x on Windows

2.9.1. Upgrading from 7.9.x to 7.10.x on Linux

> **NOTE**
>
> The name of the archive that you download could differ from what is used in the following examples.

Prerequisites

- At a minimum, AMQ Broker 7.10 requires Java version 11 to run. Ensure that each AMQ Broker host is running Java version 11 or higher. For more information on supported configurations, see Red Hat AMQ Broker 7 Supported Configurations .

- If AMQ Broker 7.9 is configured to persist message data in a database, the data type of the HOLDER_EXPIRATION_TIME column is **timestamp** in the node manager database table. In AMQ Broker 7.10, the data type of the column changed to **number**. Before you upgrade to AMQ Broker 7.10, you must drop the node manager table, that is, remove it from the database. After you drop the table, it is recreated with the new schema when you restart the upgraded broker. In a shared store high availability (HA) configuration, the node manager table is shared between brokers. Therefore, you must ensure that all brokers that share the table are stopped before you drop the table. The following example drops a node manager table called **NODE_MANAGER_TABLE**:

```
DROP TABLE NODE_MANAGER_TABLE
```

Procedure

1. Download the desired archive from the Red Hat Customer Portal. Follow the instructions provided in Downloading the AMQ Broker archive .

2. Change the owner of the archive to the same user that owns the AMQ Broker installation to be upgraded. The following example shows a user called **amq-broker**.

   ```
   sudo chown amq-broker:amq-broker amq-broker-7.x.x-bin.zip
   ```

3. Move the archive to the directory created during the original installation of AMQ Broker. The following example uses **/opt/redhat**.

   ```
   sudo mv amq-broker-7.x.x-bin.zip /opt/redhat
   ```

4. As the directory owner, extract the contents of the compressed archive. In the following example, the user **amq-broker** extracts the archive using the **unzip** command.

   ```
   su - amq-broker
   cd /opt/redhat
   unzip amq-broker-7.x.x-bin.zip
   ```

5. If the broker is running, stop it.

   ```
   <broker_instance_dir>/bin/artemis stop
   ```

6. Back up the instance directory of the broker by copying it to the home directory of the current user.

   ```
   cp -r <broker_instance_dir> ~/
   ```

7. (Optional) Note the current version of the broker. After the broker stops, you see a line similar to the one below at the end of the **<broker_instance_dir>/log/artemis.log** file.

   ```
   INFO  [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis Message Broker version 2.18.0.redhat-00010 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-54ee759954c4]
   ```

8. Edit the *<broker_instance_dir>*/**etc**/**artemis.profile** configuration file.

   a. Set the **ARTEMIS_HOME** property to the new directory created when the archive was extracted. For example:

   ```
   ARTEMIS_HOME='/opt/redhat/amq-broker-7.x.x-bin'
   ```

9. Edit the *<broker_instance_dir>*/**etc**/**bootstrap.xml** configuration file.
   In the **web** element, update the name of the .war file required by AMQ Management Console in 7.10.

   ```
   <web path="web">
      <binding uri="https://localhost:8161"
      ...
      <app url="console" war="hawtio.war"/>
      ...
   </web>
   ```

   In the **broker xmlns** element, change the schema value from **"http://activemq.org/schema"** to **"http://activemq.apache.org/schema"**.

   ```
   <broker xmlns="http://activemq.apache.org/schema">
   ```

10. Edit the *<broker_instance_dir>*/**etc**/**management.xml** file.
    In the **management-context xmlns** element, change the schema value from **"http://activemq.org/schema"** to **"http://activemq.apache.org/schema"**.

    ```
    <management-context xmlns="http://activemq.apache.org/schema">
    ```

11. Start the upgraded broker.

    ```
    <broker_instance_dir>/bin/artemis run
    ```

12. (Optional) Confirm that the broker is running and that the version has changed. After starting the broker, open the *<broker_instance_dir>*/**log**/**artemis.log** file. Find two lines similar to the ones below. Note the new version number that appears in the log when the broker is live.

    ```
    INFO [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis Mes
    INFO [org.apache.activemq.artemis.core.server] AMQ221007: Server is now live
    ...
    sage Broker version 2.21.0.redhat-00025 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-54ee759954c4]
    ```

Additional Resources

- For more information about creating an instance of the broker, see Creating a broker instance.

- You can now store a broker instance's configuration files and data in any custom directory, including locations outside of the broker instance's directory. In the *<broker_instance_dir>*/**etc**/**artemis.profile** file, update the **ARTEMIS_INSTANCE_ETC_URI**

property by specifying the location of the custom directory after creating the broker instance. Previously, these configuration files and data could only be stored in the **etc/** and **data/** directories within the broker instance's directory.

2.9.2. Upgrading from 7.9.x to 7.10.x on Windows

Prerequisites

- At a minimum, AMQ Broker 7.10 requires Java version 11 to run. Ensure that each AMQ Broker host is running Java version 11 or higher. For more information on supported configurations, see Red Hat AMQ Broker 7 Supported Configurations .

- If AMQ Broker 7.9 is configured to persist message data in a database, the data type of the HOLDER_EXPIRATION_TIME column is **timestamp** in the node manager database table. In AMQ Broker 7.10, the data type of the column changed to **number**. Before you upgrade to AMQ Broker 7.10, you must drop the node manager table, that is, remove it from the database. After you drop the table, it is recreated with the new schema when you restart the upgraded broker. In a shared store high availability (HA) configuration, the node manager table is shared between brokers. Therefore, you must ensure that all brokers that share the table are stopped before you drop the table. The following example drops a node manager table called **NODE_MANAGER_TABLE**:

```
DROP TABLE NODE_MANAGER_TABLE
```

Procedure

1. Download the desired archive from the Red Hat Customer Portal. Follow the instructions provided in Downloading the AMQ Broker archive .

2. Use a file manager to move the archive to the folder you created during the last installation of AMQ Broker.

3. Extract the contents of the archive. Right-click the .zip file and select **Extract All**.

4. If the broker is running, stop it.

   ```
   <broker_instance_dir>\bin\artemis-service.exe stop
   ```

5. Back up the broker using a file manager.

   a. Right-click the ***<broker_instance_dir>*** folder and select **Copy**.

   b. Right-click in the same window and select **Paste**.

6. (Optional) Note the current version of the broker. After the broker stops, you see a line similar to the one below at the end of the ***<broker_instance_dir>*****\log\artemis.log** file.

   ```
   INFO  [org.apache.activemq.artemis.core.server] AMQ221002: Apache ActiveMQ Artemis
   Message Broker version 2.18.0.redhat-00010[4782d50d-47a2-11e7-a160-9801a793ea45]
   stopped, uptime 28 minutes
   ```

7. Edit the ***<broker_instance_dir>*****\etc\artemis.profile.cmd** and ***<broker_instance_dir>*****\bin\artemis-service.xml** configuration files. Set the **ARTEMIS_HOME** property to the new directory created when the archive was extracted.

> ARTEMIS_HOME=*<install_dir>*

8. Edit the ***<broker_instance_dir>*\etc\artemis.profile.cmd** configuration file. Ensure that the **JAVA_ARGS** environment variable references the correct versions for the log manager and dependent file, as shown below.

> JAVA_ARGS=-Xbootclasspath/%ARTEMIS_HOME%\lib\jboss-logmanager-2.1.10.Final-redhat-00001.jar;%ARTEMIS_HOME%\lib\wildfly-common-1.5.2.Final-redhat-00002.jar

9. Edit the ***<broker_instance_dir>*\bin\artemis-service.xml** configuration file. Ensure that the bootstrap class path start argument references the correct versions for the log manager and dependent file, as shown below.

> <startargument>-Xbootclasspath/a:%ARTEMIS_HOME%\lib\jboss-logmanager-2.1.10.Final-redhat-00001.jar;%ARTEMIS_HOME%\lib\wildfly-common-1.5.2.Final-redhat-00002.jar</startargument>

10. Edit the ***<broker_instance_dir>*\etc\bootstrap.xml** configuration file.
    In the **web** element, update the name of the **.war** file required by AMQ Management Console in 7.10.

> ```
> <web path="web">
>    <binding uri="https://localhost:8161"
>    ...
>    <app url="console" war="hawtio.war"/>
>    ...
> </web>
> ```

In the **broker xmlns** element, change the schema value from **"http://activemq.org/schema"** to **"http://activemq.apache.org/schema"**.

> <broker xmlns="http://activemq.apache.org/schema">

11. Edit the ***<broker_instance_dir>*/etc/management.xml** file.
    In the **management-context xmlns** element, change the schema value from **"http://activemq.org/schema"** to **"http://activemq.apache.org/schema"**.

> <management-context xmlns="http://activemq.apache.org/schema">

12. Start the upgraded broker.

> *<broker_instance_dir>*\bin\artemis-service.exe start

13. (Optional) Confirm that the broker is running and that the version has changed. After starting the broker, open the ***<broker_instance_dir>*\log\artemis.log** file. Find two lines similar to the ones below. Note the new version number that appears in the log when the broker is live.

> INFO [org.apache.activemq.artemis.core.server] AMQ221007: Server is now live
> ...
> INFO [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis Message Broker version 2.21.0.redhat-00025 [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-54ee759954c4]

Additional Resources

- For more information about creating an instance of the broker, see Creating a broker instance.

- You can now store a broker instance's configuration files and data in any custom directory, including locations outside of the broker instance's directory. In the **<broker_instance_dir>\etc\artemis.profile** file, update the **ARTEMIS_INSTANCE_ETC_URI** property by specifying the location of the custom directory after creating the broker instance. Previously, these configuration files and data could only be stored in the **\etc** and **\data** directories within the broker instance's directory.

CHAPTER 3. USING THE COMMAND LINE INTERFACE

The command line interface (CLI) allows interaction with the message broker by use of an interactive terminal. Manage broker actions, configure messages, and enter useful commands by using the CLI.

The command line interface (CLI) allows users and roles to be added to files, by using an interactive process.

3.1. STARTING BROKER INSTANCES

A broker instance is a directory containing all the configuration and runtime data, such as logs and data files. The runtime data is associated with a unique broker process.

You can start a broker in the foreground by using the **artemis** script, as a Linux service, or as a Windows service.

3.1.1. Starting the broker instance

After the broker instance is created, you use the **artemis run** command to start it.

Procedure

1. Switch to the user account you created during installation.

   ```
   $ su - amq-broker
   ```

2. Use the **artemis run** command to start the broker instance.

   ```
   $ /var/opt/amq-broker/mybroker/bin/artemis run


          __  __  ____  ____      _
     /\  | V |/ __ \ | _ \       | |
    / \ | \ /| | | | | |_) |_ __ ___ | | _____ _ __
   / /\ \ | |V| | | | | |  _ <| '__/ _ \| |/ / _ ' __|
  / ____ \| | | | |__| | | |_) | | | (_) |   < __/ |
 /_/    \_\_| |_|\___\_\ |____/|_| \___/|_|\_\___|_|

   Red Hat JBoss AMQ 7.2.1.GA

   10:53:43,959 INFO  [org.apache.activemq.artemis.integration.bootstrap] AMQ101000:
   Starting ActiveMQ Artemis Server
   10:53:44,076 INFO  [org.apache.activemq.artemis.core.server] AMQ221000: live Message
   Broker is starting with configuration Broker Configuration
   (clustered=false,journalDirectory=./data/journal,bindingsDirectory=./data/bindings,largeMessage
   sDirectory=./data/large-messages,pagingDirectory=./data/paging)
   10:53:44,099 INFO  [org.apache.activemq.artemis.core.server] AMQ221012: Using AIO
   Journal
   ...
   ```

   The broker starts and displays log output with the following information:

   - The location of the transaction logs and cluster configuration.

   - The type of journal being used for message persistence (AIO in this case).

- The URI(s) that can accept client connections.
  By default, port 61616 can accept connections from any of the supported protocols (CORE, MQTT, AMQP, STOMP, HORNETQ, and OPENWIRE). There are separate, individual ports for each protocol as well.

- The web console is available at http://localhost:8161.

- The Jolokia service (JMX over REST) is available at http://localhost:8161/jolokia.

3.1.2. Starting a broker as a Linux service

If the broker is installed on Linux, you can run it as a service.

Procedure

1. Create a new **amq-broker.service** file in the **/etc/systemd/system/** directory.

2. Copy the following text into the file.
   Modify the path and user fields according to the information provided during the broker instance creation. In the example below, the user **amq-broker** starts the broker service installed under the **/var/opt/amq-broker/mybroker/** directory.

   ```
   [Unit]
   Description=AMQ Broker
   After=syslog.target network.target

   [Service]
   ExecStart=/var/opt/amq-broker/mybroker/bin/artemis run
   Restart=on-failure
   User=amq-broker
   Group=amq-broker

   # A workaround for Java signal handling
   SuccessExitStatus=143

   [Install]
   WantedBy=multi-user.target
   ```

3. Open a terminal.

4. Enable the broker service using the following command:

   ```
   sudo systemctl enable amq-broker
   ```

5. Run the broker service using the following command:

   ```
   sudo systemctl start amq-broker
   ```

3.1.3. Starting a broker as a Windows service

If the broker is installed on Windows, you can run it as a service.

Procedure

1. Open a command prompt to enter the commands

2. Install the broker as a service with the following command:

   > *<broker_instance_dir>*\bin\artemis-service.exe install

3. Start the service by using the following command:

   > *<broker_instance_dir>*\bin\artemis-service.exe start

4. (Optional) Uninstall the service:

   > *<broker_instance_dir>*\bin\artemis-service.exe uninstall

3.2. STOPPING BROKER INSTANCES

Stop the broker instance manually or configure the broker to shutdown gracefully.

3.2.1. Stopping the broker instance

After creating the standalone broker and producing and consuming test messages, you can stop the broker instance.

This procedure manually stops the broker, which forcefully closes all client connections. In a production environment, you should configure the broker to stop gracefully so that client connections can be closed properly.

Procedure

- Use the **artemis stop** command to stop the broker instance:

  > $ /var/opt/amq-broker/mybroker/bin/artemis stop
  > 2018-12-03 14:37:30,630 INFO [org.apache.activemq.artemis.core.server] AMQ221002:
  > Apache ActiveMQ Artemis Message Broker version 2.6.1.amq-720004-redhat-1 [b6c244ef-
  > f1cb-11e8-a2d7-0800271b03bd] stopped, uptime 35 minutes
  > Server stopped!

3.2.2. Stopping a broker instance gracefully

A manual shutdown forcefully disconnects all clients after a **stop** command is entered. As an alternative, configure the broker to shut down gracefully by using the **graceful-shutdown-enabled** configuration element.

When **graceful-shutdown-enabled** is set to **true**, no new client connections are allowed after a **stop** command is entered. However, existing connections are allowed to close on the client-side before the shutdown process is started. The default value for **graceful-shutdown-enabled** is **false**.

Use the **graceful-shutdown-timeout** configuration element to set a length of time, in milliseconds, for clients to disconnect before connections are forcefully closed from the broker side. After all connections are closed, the shutdown process is started. One advantage of using **graceful-shutdown-timeout** is that it prevents client connections from delaying a shutdown. The default value for **graceful-shutdown-timeout** is **-1**, meaning the broker waits indefinitely for clients to disconnect.

The following procedure demonstrates how to configure a graceful shutdown that uses a timeout.

Procedure

1. Open the configuration file **<broker_instance_dir>\etc\broker.xml**.

2. Add the **graceful-shutdown-enabled** configuration element and set the value to **true**.

```
<configuration>
    <core>
        ...
        <graceful-shutdown-enabled>
            true
        </graceful-shutdown-enabled>
        ...
    </core>
</configuration>
```

3. Add the **graceful-shutdown-timeout** configuration element and set a value for the timeout in milliseconds. In the following example, client connections are forcefully closed 30 seconds (**30000** milliseconds) after the **stop** command is issued.

```
<configuration>
    <core>
        ...
        <graceful-shutdown-enabled>
            true
        </graceful-shutdown-enabled>
        <graceful-shutdown-timeout>
            30000
        </graceful-shutdown-timeout>
        ...
    </core>
</configuration>
```

3.3. AUDITING MESSAGES BY INTERCEPTING PACKETS

Intercept packets entering or exiting the broker, to audit packets or filter messages. Interceptors change the packets that they intercept. This makes interceptors powerful, but also potentially dangerous.

Develop interceptors to meet your business requirements. Interceptors are protocol specific and must implement the appropriate interface.

Interceptors must implement the **intercept()** method, which returns a boolean value. If the value is **true**, the message packet continues onward. If **false**, the process is aborted, no other interceptors are called, and the message packet is not processed further.

3.3.1. Creating interceptors

Interceptors can change the packets they intercept. You can create your own incoming and outgoing interceptors. All interceptors are protocol specific and are called for any packet entering or exiting the server respectively. This allows you to create interceptors to meet business requirements such as auditing packets.

Interceptors and their dependencies must be placed in the Java classpath of the broker. You can use the *<broker_instance_dir>*/**lib** directory because it is part of the classpath by default.

The following examples demonstrate how to create an interceptor that checks the size of each packet passed to it.
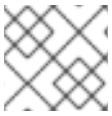
> **NOTE**
>
> The examples implement a specific interface for each protocol.

Procedure

1. Implement the appropriate interface and override its **intercept()** method.

   a. If you are using the AMQP protocol, implement the **org.apache.activemq.artemis.protocol.amqp.broker.AmqpInterceptor** interface.

   ```java
   package com.example;

   import org.apache.activemq.artemis.protocol.amqp.broker.AMQPMessage;
   import org.apache.activemq.artemis.protocol.amqp.broker.AmqpInterceptor;
   import org.apache.activemq.artemis.spi.core.protocol.RemotingConnection;

   public class MyInterceptor implements AmqpInterceptor
   {
     private final int ACCEPTABLE_SIZE = 1024;

     @Override
     public boolean intercept(final AMQPMessage message, RemotingConnection connection)
     {
       int size = message.getEncodeSize();
       if (size <= ACCEPTABLE_SIZE) {
         System.out.println("This AMQPMessage has an acceptable size.");
         return true;
       }
       return false;
     }
   }
   ```

   b. If you are using Core Protocol, your interceptor must implement the **org.apache.artemis.activemq.api.core.Interceptor** interface.

   ```java
   package com.example;

   import org.apache.artemis.activemq.api.core.Interceptor;
   import org.apache.activemq.artemis.core.protocol.core.Packet;
   import org.apache.activemq.artemis.spi.core.protocol.RemotingConnection;

   public class MyInterceptor implements Interceptor
   {
     private final int ACCEPTABLE_SIZE = 1024;

     @Override
     boolean intercept(Packet packet, RemotingConnection connection)
   ```

```
  throws ActiveMQException
  {
    int size = packet.getPacketSize();
    if (size <= ACCEPTABLE_SIZE) {
      System.out.println("This Packet has an acceptable size.");
      return true;
    }
    return false;
  }
}
```

c. If you are using the MQTT protocol, implement the
org.apache.activemq.artemis.core.protocol.mqtt.MQTTInterceptor interface.

```
package com.example;

import org.apache.activemq.artemis.core.protocol.mqtt.MQTTInterceptor;
import io.netty.handler.codec.mqtt.MqttMessage;
import org.apache.activemq.artemis.spi.core.protocol.RemotingConnection;

public class MyInterceptor implements Interceptor
{
  private final int ACCEPTABLE_SIZE = 1024;

  @Override
  boolean intercept(MqttMessage mqttMessage, RemotingConnection connection)
  throws ActiveMQException
  {
    byte[] msg = (mqttMessage.toString()).getBytes();
    int size = msg.length;
    if (size <= ACCEPTABLE_SIZE) {
      System.out.println("This MqttMessage has an acceptable size.");
      return true;
    }
    return false;
  }
}
```

d. If you are using the STOMP protocol, implement the
org.apache.activemq.artemis.core.protocol.stomp.StompFrameInterceptor interface.

```
package com.example;

import org.apache.activemq.artemis.core.protocol.stomp.StompFrameInterceptor;
import org.apache.activemq.artemis.core.protocol.stomp.StompFrame;
import org.apache.activemq.artemis.spi.core.protocol.RemotingConnection;

public class MyInterceptor implements Interceptor
{
  private final int ACCEPTABLE_SIZE = 1024;

  @Override
  boolean intercept(StompFrame stompFrame, RemotingConnection connection)
  throws ActiveMQException
  {
```

```
      int size = stompFrame.getEncodedSize();
      if (size <= ACCEPTABLE_SIZE) {
        System.out.println("This StompFrame has an acceptable size.");
        return true;
      }
      return false;
    }
  }
```

3.3.2. Configuring the broker to use interceptors

Prerequisites

- Create an interceptor class and add it (and its dependencies) to the Java classpath of the broker. You can use the ***<broker_instance_dir>*/lib** directory since it is part of the classpath by default.

Procedure

1. Open ***<broker_instance_dir>*/etc/broker.xml**

2. Configure the broker to use an interceptor by adding configuration to ***<broker_instance_dir>*/etc/broker.xml**

    a. If the interceptor is intended for incoming messages, add its **class-name** to the list of **remoting-incoming-interceptors**.

    ```
    <configuration>
     <core>
       ...
        <remoting-incoming-interceptors>
           <class-name>org.example.MyIncomingInterceptor</class-name>
        </remoting-incoming-interceptors>
       ...
     </core>
    </configuration>
    ```

    b. If the interceptor is intended for outgoing messages, add its **class-name** to the list of **remoting-outgoing-interceptors**.

    ```
    <configuration>
     <core>
       ...
        <remoting-outgoing-interceptors>
           <class-name>org.example.MyOutgoingInterceptor</class-name>
        </remoting-outgoing-interceptors>
     </core>
    </configuration>
    ```

3.3.3. Interceptors on the client side

Clients can use interceptors to intercept packets either sent by the client to the server or by the server to the client. If the broker-side interceptor returns a **false** value, then no other interceptors are called and the client does not process the packet further. This process happens transparently, unless an

outgoing packet is sent in a **blocking** fashion. In this case, an **ActiveMQException** is thrown to the caller. The **ActiveMQException** thrown contains the name of the interceptor that returned the **false** value.

On the server, the client interceptor classes and their dependencies must be added to the Java classpath of the client, to be properly instantiated and invoked.

3.4. CHECKING THE HEALTH OF BROKERS AND QUEUES

AMQ Broker includes a command-line utility that enables you to perform various health checks on brokers and queues in your broker topology.

The following example shows how to the use the utility to run health checks.

Procedure

1. See the list of checks that you can run for a particular broker (that is, *node*) in your broker topology.

   ```
   $ <broker_instance_dir>/bin/artemis help check node
   ```

   You see output that describes the set of options that you can use with the **artemis check node** command.

   ```
   NAME
           artemis check node - Check a node

   SYNOPSIS
           artemis check node [--backup] [--clientID <clientID>]
                   [--diskUsage <diskUsage>] [--fail-at-end] [--live]
                   [--memoryUsage <memoryUsage>] [--name <name>] [--password <password>]
                   [--peers <peers>] [--protocol <protocol>] [--silent]
                   [--timeout <timeout>] [--up] [--url <brokerURL>] [--user <user>]
                   [--verbose]

   OPTIONS
           --backup
               Check that the node has a backup

           --clientID <clientID>
               ClientID to be associated with connection

           --diskUsage <diskUsage>
               Disk usage percentage to check or -1 to use the max-disk-usage

           --fail-at-end
               If a particular module check fails, continue the rest of the checks

           --live
               Check that the node has a live

           --memoryUsage <memoryUsage>
               Memory usage percentage to check

           --name <name>
   ```

Name of the target to check

--password <password>
Password used to connect

--peers <peers>
Number of peers to check

--protocol <protocol>
Protocol used. Valid values are amqp or core. Default=core.

--silent
It will disable all the inputs, and it would make a best guess for any required input

--timeout <timeout>
Time to wait for the check execution, in milliseconds

--up
Check that the node is started, it is executed by default if there are no other checks

--url <brokerURL>
URL towards the broker. (default: tcp://localhost:61616)

--user <user>
User used to connect

--verbose
Adds more information on the execution

2. For example, check that the disk usage of the local broker is below the maximum disk usage configured for the broker.

```
$ <broker_instance_dir>/bin/artemis check node --url tcp://localhost:61616 --diskUsage -1

Connection brokerURL = tcp://localhost:61616
Running NodeCheck
Checking that the disk usage is less then the max-disk-usage ... success
Checks run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.065 sec - NodeCheck
```

In the preceding example, specifying a value of **-1** for the **--diskUsage** option means that the utility checks disk usage against the **maximum** disk usage configured for the broker. The maximum disk usage of a broker is configured using the **max-disk-usage** parameter in the **broker.xml** configuration file. The value specified for **max-disk-usage** represents the percentage of available physical disk space that the broker is allowed to consume.

3. See the list of checks that you can run for a particular queue in your broker topology.

```
$ <broker_instance_dir>/bin/artemis help check queue
```

You see output that describes the set of options that you can use with the **artemis check queue** command.

```
NAME
        artemis check queue - Check a queue
```

SYNOPSIS
    artemis check queue [--browse <browse>] [--clientID <clientID>]
        [--consume <consume>] [--fail-at-end] [--name <name>]
        [--password <password>] [--produce <produce>] [--protocol <protocol>]
        [--silent] [--timeout <timeout>] [--up] [--url <brokerURL>]
        [--user <user>] [--verbose]

OPTIONS
    --browse <browse>
        Number of the messages to browse or -1 to check that the queue is
        browsable

    --clientID <clientID>
        ClientID to be associated with connection

    --consume <consume>
        Number of the messages to consume or -1 to check that the queue is consumable

    --fail-at-end
        If a particular module check fails, continue the rest of the checks

    --name <name>
        Name of the target to check

    --password <password>
        Password used to connect

    --produce <produce>
        Number of the messages to produce

    --protocol <protocol>
        Protocol used. Valid values are amqp or core. Default=core.

    --silent
        It will disable all the inputs, and it would make a best guess for any required input

    --timeout <timeout>
        Time to wait for the check execution, in milliseconds

    --up
        Check that the queue exists and is not paused, it is executed by default if there are no
other checks

    --url <brokerURL>
        URL towards the broker. (default: tcp://localhost:61616)

    --user <user>
        User used to connect

    --verbose
        Adds more information on the execution

4. The utility can execute multiple options with a single command. For example, to check
   production, browsing, and consumption of 1000 messages on the default **helloworld** queue on
   the local broker, use the following command:

```
$ <broker_instance_dir>/bin/artemis check queue --name helloworld --produce 1000 --
browse 1000 --consume 1000

Connection brokerURL = tcp://localhost:61616
Running QueueCheck
Checking that a producer can send 1000 messages to the queue helloworld ... success
Checking that a consumer can browse 1000 messages from the queue helloworld ... success
Checking that a consumer can consume 1000 messages from the queue helloworld ...
success
Checks run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.882 sec - QueueCheck
```

In the preceding example, observe that you did not specify a broker URL when running the queue check. If you do not explicitly specify a URL, the utility uses a default value of **tcp://localhost:61616**.

3.5. COMMAND LINE TOOLS

AMQ Broker includes a set of command line interface (CLI) tools, so you can manage your messaging journal. The table below lists the name for each tool and its corresponding description.

Tool	Description
address	Addresses tool groups (create/delete/update/show) (example **./artemis address create**).
browser	Browses messages on an instance.
consumer	Consumes messages on an instance.
data	Prints reports about journal records and compacts the data.
decode	Imports the internal journal format from encode.
encode	Shows an internal format of the journal encoded to String.
exp	Exports the message data using a special and independent XML format.
help	Displays help information.
imp	Imports the journal to a running broker using the output provided by **exp**.
kill	Kills a broker instance started with --allow-kill.
mask	Masks a password and prints it out.
perf-journal	Calculates the journal-buffer timeout you should use with the current data folder.
queue	Queues tool groups (create/delete/update/stat) (example **./artemis queue create**).

Tool	Description
run	Runs the broker instance.
stop	Stops the broker instance.
user	Default file-based user managament (add/rm/list/reset) (example **./artemis user list**)

For a full list of commands available for each tool, use the **help** parameter followed by the tool's name. For instance, in the example below, the CLI output lists all the commands available to the **data** tool after the user enters the command **./artemis help data**.

```
$ ./artemis help data

NAME
      artemis data - data tools group
      (print|imp|exp|encode|decode|compact) (example ./artemis data print)

SYNOPSIS
      artemis data
      artemis data compact [--broker <brokerConfig>] [--verbose]
            [--paging <paging>] [--journal <journal>]
            [--large-messages <largeMessges>] [--bindings <binding>]
      artemis data decode [--broker <brokerConfig>] [--suffix <suffix>]
            [--verbose] [--paging <paging>] [--prefix <prefix>] [--file-size <size>]
            [--directory <directory>] --input <input> [--journal <journal>]
            [--large-messages <largeMessges>] [--bindings <binding>]
      artemis data encode [--directory <directory>] [--broker <brokerConfig>]
            [--suffix <suffix>] [--verbose] [--paging <paging>] [--prefix <prefix>]
            [--file-size <size>] [--journal <journal>]
            [--large-messages <largeMessges>] [--bindings <binding>]
      artemis data exp [--broker <brokerConfig>] [--verbose]
            [--paging <paging>] [--journal <journal>]
            [--large-messages <largeMessges>] [--bindings <binding>]
      artemis data imp [--host <host>] [--verbose] [--port <port>]
            [--password <password>] [--transaction] --input <input> [--user <user>]
      artemis data print [--broker <brokerConfig>] [--verbose]
            [--paging <paging>] [--journal <journal>]
            [--large-messages <largeMessges>] [--bindings <binding>]

COMMANDS
      With no arguments, Display help information

      print
          Print data records information (WARNING: don't use while a
          production server is running)


      ...
```

You can use the **help** parameter for more information on how to execute each of the commands. For example, the CLI lists more information about the **data print** command after the user enters the **./artemis help data print**.

```
$ ./artemis help data print

NAME
      artemis data print - Print data records information (WARNING: don't use
      while a production server is running)

SYNOPSIS
      artemis data print [--bindings <binding>] [--journal <journal>]
            [--paging <paging>]

OPTIONS
      --bindings <binding>
         The folder used for bindings (default ../data/bindings)

      --journal <journal>
         The folder used for messages journal (default ../data/journal)

      --paging <paging>
         The folder used for paging (default ../data/paging)
```

CHAPTER 4. USING AMQ MANAGEMENT CONSOLE

AMQ Management Console is a web console included in the AMQ Broker installation that enables you to use a web browser to manage AMQ Broker.

AMQ Management Console is based on hawtio.

4.1. OVERVIEW

AMQ Broker is a full-featured, message-oriented middleware broker. It offers specialized queueing behaviors, message persistence, and manageability. It supports multiple protocols and client languages, freeing you to use many of your application assets.

AMQ Broker's key features allow you to:

- monitor your AMQ brokers and clients

  - view the topology

  - view network health at a glance

- manage AMQ brokers using:

  - AMQ Management Console

  - Command-line Interface (CLI)

  - Management API

The supported web browsers for AMQ Management Console are Firefox and Chrome. For more information on supported browser versions, see AMQ 7 Supported Configurations .

4.2. CONFIGURING LOCAL AND REMOTE ACCESS TO AMQ MANAGEMENT CONSOLE

The procedure in this section shows how to configure local and remote access to AMQ Management Console.

Remote access to the console can take one of two forms:

- Within a console session on a local broker, you use the **Connect** tab to connect to another, remote broker

- From a remote host, you connect to the console for the local broker, using an externally-reachable IP address for the local broker

Prerequisites

- You must upgrade to at least AMQ Broker 7.1.0. As part of this upgrade, an access-management configuration file named **jolokia-access.xml** is added to the broker instance. For more information about upgrading, see Upgrading a Broker instance from 7.0.x to 7.1.0 .

Procedure

1. Open the ***<broker_instance_dir>*/etc/bootstrap.xml** file.

2. Within the **web** element, observe that the web port is bound only to **localhost** by default.

```
<web path="web">
  <binding uri="http://localhost:8161">
    <app url="redhat-branding" war="redhat-branding.war"/>
    <app url="artemis-plugin" war="artemis-plugin.war"/>
    <app url="dispatch-hawtio-console" war="dispatch-hawtio-console.war"/>
    <app url="console" war="console.war"/>
  </binding>
</web>
```

3. To enable connection to the console for the local broker from a remote host, change the web port binding to a network-reachable interface. For example:

```
<web path="web">
  <binding uri="http://0.0.0.0:8161">
```

In the preceding example, by specifying **0.0.0.0**, you bind the web port to **all** interfaces on the local broker.

4. Save the **bootstrap.xml** file.

5. Open the *<broker_instance_dir>*/**etc/jolokia-access.xml** file.

6. Within the **<cors>** (that is, *Cross-Origin Resource Sharing*) element, add an **allow-origin** entry for each HTTP origin request header that you want to allow to access the console. For example:

```
<cors>
  <allow-origin>*://localhost*</allow-origin>
  <allow-origin>*://192.168.0.49*</allow-origin>
  <allow-origin>*://192.168.0.51*</allow-origin>
  <!-- Check for the proper origin on the server side, too -->
  <strict-checking/>
</cors>
```

In the preceding configuration, you specify that the following connections are allowed:

- Connection from the local host (that is, the host machine for your local broker instance) to the console.

  - The first asterisk (*****) wildcard character allows either the **http** or **https** scheme to be specified in the connection request, based on whether you have configured the console for secure connections.

  - The second asterisk wildcard character allows any port on the host machine to be used for the connection.

- Connection from a remote host to the console for the local broker, using the externally-reachable IP address of the local broker. In this case, the externally-reachable IP address of the local broker is **192.168.0.49**.

- Connection from within a console session opened on another, remote broker to the local broker. In this case, the IP address of the remote broker is **192.168.0.51**.

7. Save the **jolokia-access.xml** file.

8. Open the **<broker_instance_dir>/etc/artemis.profile** file.

9. To enable the **Connect** tab in the console, set the value of the **Dhawtio.disableProxy** argument to **false**.

```
-Dhawtio.disableProxy=false
```

> **IMPORTANT**
>
> It is recommended that you enable remote connections from the console (that is, set the value of the **Dhawtio.disableProxy** argument to **false**) **only** if the console is exposed to a secure network.

10. Add a new argument, **Dhawtio.proxyWhitelist**, to the **JAVA_ARGS** list of Java system arguments. As a comma-separated list, specify IP addresses for any remote brokers that you want to connect to from the local broker (that is, by using the **Connect** tab within a console session running on the local broker). For example:

```
-Dhawtio.proxyWhitelist=192.168.0.51
```

Based on the preceding configuration, you can use the **Connect** tab within a console session on the local broker to connect to another, remote broker with an IP address of **192.168.0.51**.

11. Save the **aretmis.profile** file.

Additional resources

- To learn how to access the console, see Section 4.3, "Accessing AMQ Management Console" .

- For more information about:

    - Cross-Origin Resource Sharing, see W3C Recommendations.

    - Jolokia security, see Jolokia Protocols.

    - Securing connections to the console, see Section 4.4.3, "Securing network access to AMQ Management Console".

4.3. ACCESSING AMQ MANAGEMENT CONSOLE

The procedure in this section shows how to:

- Open AMQ Management Console from the local broker

- Connect to other brokers from within a console session on the local broker

- Open a console instance for the local broker from a remote host using the externally-reachable IP address of the local broker

Prerequisites

- You must have already configured local and remote access to the console. For more information, see Section 4.2, "Configuring local and remote access to AMQ Management Console".

Procedure

1. In your web browser, navigate to the console address for the local broker.
   The console address is **http://*<host:port>*/console/login**. If you are using the default address,
   navigate to http://localhost:8161/console/login. Otherwise, use the values of host and port that
   are defined for the **bind** attribute of the **web** element in the
   ***<broker_instance_dir>*/etc/bootstrap.xml** configuration file.

Figure 4.1. Console login page
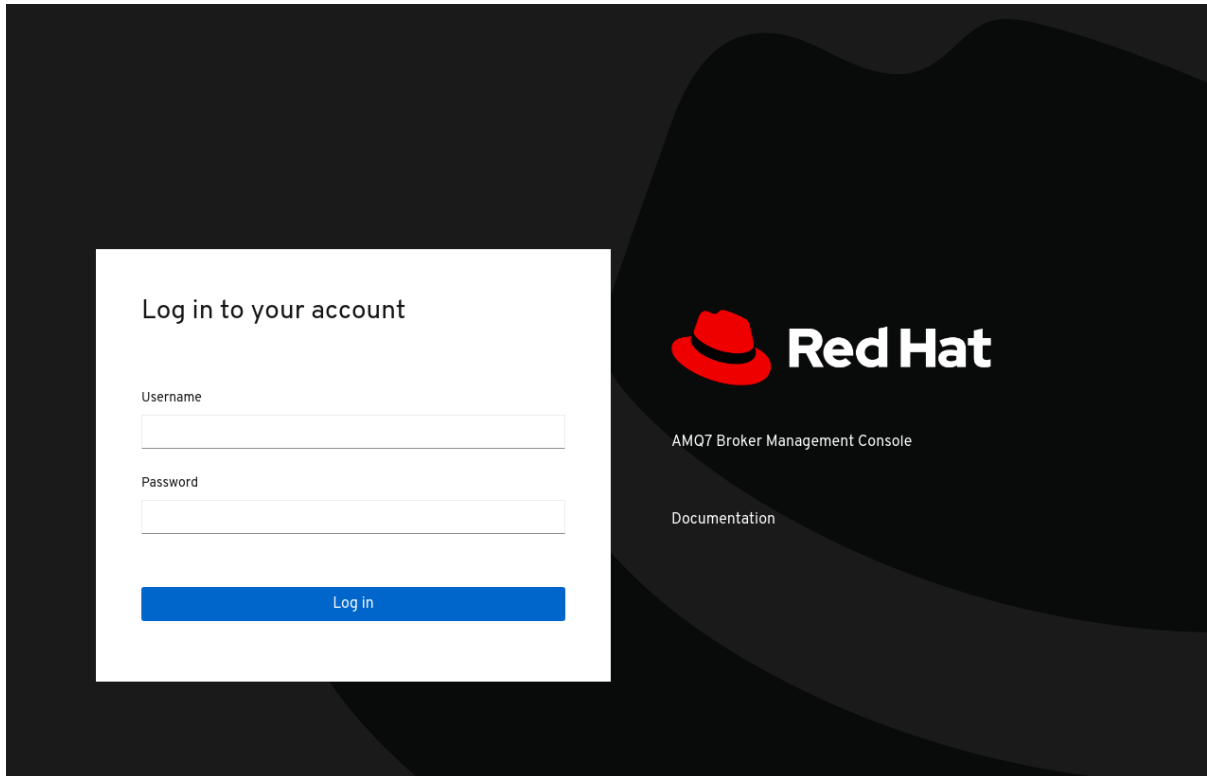
2. Log in to AMQ Management Console using the default user name and password that you
   created when you created the broker.

3. To connect to another, remote broker from the console session of the local broker:

   a. In the left menu, click the **Connect** tab.

   b. In the main pane, on the **Remote** tab, click the **Add connection** button.

   c. In the **Add Connection** dialog box, specify the following details:

      Name

         Name for the remote connection, for example, **my_other_broker**.

      Scheme

         Protocol to use for the remote connection. Select **http** for a non-secured connection, or
         https for a secured connection.

      Host

         IP address of a remote broker. You must have already configured console access for this
         remote broker.

      Port

Port on the local broker to use for the remote connection. Specify the port value that is defined for the **bind** attribute of the **web** element in the ***<broker_instance_dir>*/etc/bootstrap.xml** configuration file. The default value is **8161**.

Path

Path to use for console access. Specify **console/jolokia**.

d. To test the connection, click the **Test Connection** button.
If the connection test is successful, click the **Add** button. If the connection test fails, review and modify the connection details as needed. Test the connection again.

e. On the **Remote** page, for a connection that you have added, click the **Connect** button.
A new web browser tab opens for the console instance on the remote broker.

f. In the **Log In** dialog box, enter the user name and password for the remote broker. Click **Log In**.
The console instance for the remote broker opens.

4. To connect to the console for the local broker from a remote host, specify the Jolokia endpoint for the local broker in a web browser. This endpoint includes the externally-reachable IP address that you specified for the local broker when configuring remote console access. For example:

> http://192.168.0.49/console/jolokia

4.4. CONFIGURING AMQ MANAGEMENT CONSOLE

Configure user access and request access to resources on the broker.

4.4.1. Securing AMQ Management Console using Red Hat Single Sign-On

Prerequisites

- Red Hat Single Sign-On 7.4

Procedure

1. Configure Red Hat Single Sign-On:

    a. Navigate to the realm in Red Hat Single Sign-On that you want to use for securing AMQ Management Console. Each realm in Red Hat Single Sign-On includes a client named **Broker**. This client is not related to AMQ.

    b. Create a new client in Red Hat Single Sign-On, for example **artemis-console**.

    c. Navigate to the client settings page and set:

        - **Valid Redirect URIs** to the AMQ Management Console URL followed by *****, for example:

            > https://broker.example.com:8161/console/*

        - **Web Origins** to the same value as **Valid Redirect URIs**. Red Hat Single Sign-On allows you enter **+**, indicating that allowed CORS origins includes the value for **Valid Redirect URIs**.

    d. Create a role for the client, for example **guest**.

e. Make sure all users who require access to AMQ Management Console are assigned the above role, for example, using Red Hat Single Sign-On groups.

2. Configure the AMQ Broker instance:

   a. Add the following to your ***<broker-instance-dir>*/instances/broker0/etc/login.config** file to configure AMQ Management Console to use Red Hat Single Sign-On:

   ```
   console {
       org.keycloak.adapters.jaas.BearerTokenLoginModule required
           keycloak-config-file="${artemis.instance}/etc/keycloak-bearer-token.json"
           role-principal-
   class=org.apache.activemq.artemis.spi.core.security.jaas.RolePrincipal
         ;
   };
   ```

   Adding this configuration sets up a JAAS principal and a requirement for a bearer token from Red Hat Single Sign-On. The connection to Red Hat Single Sign-On is defined in the **keycloak-bearer-token.json** file, as described in the next step.

   b. Create a file ***<broker-instance-dir>*/etc/keycloak-bearer-token.json** with the following contents to specify the connection to Red Hat Single Sign-On used for the bearer token exchange:

   ```
   {
     "realm": "<realm-name>",
     "resource": "<client-name>",
     "auth-server-url": "<RHSSO-URL>/auth",
     "principal-attribute": "preferred_username",
     "use-resource-role-mappings": true,
     "ssl-required": "external",
     "confidential-port": 0
   }
   ```

   <realm-name>

   the name of the realm in Red Hat Single Sign-On

   <client-name>

   the name of the client in Red Hat Single Sign-On

   <RHSSO-URL>

   the URL of Red Hat Single Sign-On

   c. Create a file ***<broker-instance-dir>*/etc/keycloak-js-token.json** with the following contents to specify the Red Hat Single Sign-On authentication endpoint:

   ```
   {
     "realm": "<realm-name>",
     "clientId": "<client-name>",
     "url": "<RHSSO-URL>/auth"
   }
   ```

   d. Configure the security settings by editing the ***<broker-instance-dir>*/etc/broker.xml** file. For example, to allow users with the **amq** role consume messages and allow users with the **guest** role send messages, add the following:

```
<security-setting match="Info">
    <permission roles="amq" type="createDurableQueue"/>
    <permission roles="amq" type="deleteDurableQueue"/>
    <permission roles="amq" type="createNonDurableQueue"/>
    <permission roles="amq" type="deleteNonDurableQueue"/>
    <permission roles="guest" type="send"/>
    <permission roles="amq" type="consume"/>
</security-setting>
```

3. Run the AMQ Broker instance and validate AMQ Management Console configuration.

4.4.2. Setting up user access to AMQ Management Console

You can access AMQ Management Console using the broker login credentials. The following table provides information about different methods to add additional broker users to access AMQ Management Console:

Authentication Method	Description
Guest authentication	Enables anonymous access. In this configuration, any user who connects without credentials or with the wrong credentials will be authenticated automatically and assigned a specific user and role. For more information, see Configuring guest access in *Configuring AMQ Broker*.
Basic user and password authentication	For each user, you must define a username and password and assign a security role. Users can only log into AMQ Management Console using these credentials. For more information, see Configuring basic user and password authentication in *Configuring AMQ Broker*.
LDAP authentication	Users are authenticated and authorized by checking the credentials against user data stored in a central X.500 directory server. For more information, see Configuring LDAP to authenticate clients in *Configuring AMQ Broker*.

4.4.3. Securing network access to AMQ Management Console

To secure AMQ Management Console when the console is being accessed over a WAN or the internet, use SSL to specify that network access uses **https** instead of **http**.

Prerequisites

The following should be located in the ***<broker_instance_dir>*/etc/** directory:

- Java key store

- Java trust store (needed only if you require client authentication)

Procedure

1. Open the **<broker_instance_dir>/etc/bootstrap.xml** file.

2. In the **<web>** element, add the following attributes:

```
<web path="web">
    <binding uri="https://0.0.0.0:8161" keyStorePath="<path_to_keystore>"
keyStorePassword="<password>"
        clientAuth="<true/false>" trustStorePath="<path_to_truststore>" trustStorePassword="
<password>">
    </binding>
</web>
```

bind

For secure connections to the console, change the URI scheme to **https**.

keyStorePath

Path of the keystore file. For example:

```
keyStorePath="<broker_instance_dir>/etc/keystore.jks"
```

keyStorePassword

Key store password. This password can be encrypted.

clientAuth

Specifies whether client authentication is required. The default value is **false**.

trustStorePath

Path of the trust store file. You need to define this attribute only if **clientAuth** is set to **true**.

trustStorePassword

Trust store password. This password can be encrypted.

Additional resources

- For more information about encrypting passwords in broker configuration files, including
  bootstrap.xml, see Encrypting Passwords in Configuration Files .

4.4.4. Configuring AMQ Management Console to use certificate-based authentication

You can configure AMQ Management Console to authenticate users by using certificates instead of passwords.

Procedure

1. Obtain certificates for the broker and clients from a trusted certificate authority or generate self-signed certificates. If you want to generate self-signed certificates, complete the following steps:

   a. Generate a self-signed certificate for the broker.

   ```
   $ keytool -storetype pkcs12 -keystore broker-keystore.p12 -storepass securepass -
   keypass securepass -alias client -genkey -keyalg "RSA" -keysize 2048 -dname
   "CN=ActiveMQ Broker, OU=Artemis, O=ActiveMQ, L=AMQ, S=AMQ, C=AMQ" -ext
   bc=ca:false -ext eku=cA
   ```

b. Export the certificate from the broker keystore, so that it can be shared with clients.

```
$ keytool -storetype pkcs12 -keystore broker-keystore.p12 -storepass securepass -alias
client -exportcert -rfc > broker.crt
```

c. On the client, import the broker certificate into the client truststore.

```
$ keytool -storetype pkcs12 -keystore client-truststore.p12 -storepass securepass -
keypass securepass -importcert -alias client-ca -file broker.crt -noprompt
```

d. On the client, generate a self-signed certificate for the client.

```
$ keytool -storetype pkcs12 -keystore client-keystore.p12 -storepass securepass -
keypass securepass -alias client -genkey -keyalg "RSA" -keysize 2048 -dname
"CN=ActiveMQ Client, OU=Artemis, O=ActiveMQ, L=AMQ, S=AMQ, C=AMQ" -ext
bc=ca:false -ext eku=cA
```

e. Export the client certificate from the client keystore to a file so that it can be added to the broker truststore.

```
$ keytool -storetype pkcs12 -keystore client-keystore.p12 -storepass securepass -alias
client -exportcert -rfc > client.crt
```

f. Import the client certificate into the broker truststore.

```
$ keytool -storetype pkcs12 -keystore client-truststore.p12 -storepass securepass -
keypass securepass -importcert -alias client-ca -file client.crt -noprompt
```

> **NOTE**
>
> On the broker machine, ensure that the keystore and truststore files are in a location that is accessible to the broker.

2. In the **<broker_instance_dir>/etc/bootstrap.xml** file, update the web configuration to enable the HTTPS protocol and client authentication for the broker console. For example:

```
...
<web path="web">
    <binding uri="https://localhost:8161" keyStorePath="${artemis.instance}/etc/server-
keystore.p12" keyStorePassword="password"
    clientAuth="true" trustStorePath="${artemis.instance}/etc/client-truststore.p12"
trustStorePassword="password">
    ...
    </binding>
</web>
...
```

binding uri

Specify the https protocol to enable SSL and add a host name and port.

keystorePath

The path to the keystore where the broker certificate is installed.

keystorePassword

The password of the keystore where the broker certificate is installed.

ClientAuth

Set to true to configure the broker to require that each client presents a certificate when a client tries to connect to the broker console.

trustStorePath

If clients are using self-signed certificates, specify the path to the truststore where client certificates are installed.

trustStorePassword

If clients are using self-signed certificates, specify the password of the truststore where client certificates are installed .

NOTE. You need to configure the **trustStorePath** and **trustStorePassword** properties only if clients are using self-signed certificates.

3. Obtain the Subject *Distinguished Names* (DNs) from each client certificate so you can create a mapping between each client certificate and a broker user.

   a. Export each client certificate from the client's keystore file into a temporary file. For example:

      ```
      keytool -export -file <file_name> -alias broker-localhost -keystore broker.ks -storepass <password>
      ```

   b. Print the contents of the exported certificate:

      ```
      keytool -printcert -file <file_name>
      ```

      The output is similar to that shown below:

      ```
      Owner: CN=AMQ Client, OU=Artemis, O=AMQ, L=AMQ, ST=AMQ, C=AMQ
      Issuer: CN=AMQ Client, OU=Artemis, O=AMQ, L=AMQ, ST=AMQ, C=AMQ
      Serial number: 51461f5d
      Valid from: Sun Apr 17 12:20:14 IST 2022 until: Sat Jul 16 12:20:14 IST 2022
      Certificate fingerprints:
        SHA1: EC:94:13:16:04:93:57:4F:FD:CA:AD:D8:32:68:A4:13:CC:EA:7A:67
        SHA256:
      85:7F:D5:4A:69:80:3B:5B:86:27:99:A7:97:B8:E4:E8:7D:6F:D1:53:08:D8:7A:BA:A7:0A:7A:
      96:F3:6B:98:81
      ```

      The **Owner** entry is the Subject DN. The format used to enter the Subject DN depends on your platform. The string above could also be represented as;

      ```
      Owner: `CN=localhost,\ OU=broker,\ O=Unknown,\ L=Unknown,\ ST=Unknown,\ C=Unknown`
      ```

4. Enable certificate-based authentication for the broker's console.

   a. Open the ***<broker_instance_dir>*/etc/login.config** configuration file. Add the certificate login module and reference the user and roles properties files. For example:

      ```
      activemq {
      ```

```
        org.apache.activemq.artemis.spi.core.security.jaas.TextFileCertificateLoginModule
            debug=true
            org.apache.activemq.jaas.textfiledn.user="artemis-users.properties"
            org.apache.activemq.jaas.textfiledn.role="artemis-roles.properties";
    };
```

org.apache.activemq.artemis.spi.core.security.jaas.TextFileCertificateLoginModule

The implementation class.

org.apache.activemq.jaas.textfiledn.user

Specifies the location of the user properties file relative to the directory that contains the login configuration file.

org.apache.activemq.jaas.textfiledn.role

Specifies the properties file that maps users to defined roles for the login module implementation.

> **NOTE**
>
> If you change the default name of the certificate login module configuration in the ***<broker_instance_dir>*/etc/login.config** file, you must update the value of the **-dhawtio.realm** argument in the **<broker_instance_dir>/etc/artemis.profile** file to match the new name. The default name is **activemq**.

b. Open the **<broker_instance_dir>/etc/artemis-users.properties** file. Create a mapping between client certificates and broker users by adding the Subject DNS that you obtained from each client certificate to a broker user. For example:

```
        user1=CN=user1,O=Progress,C=US
        user2=CN=user2,O=Progress,C=US
```

In this example, the user1 broker user is mapped to the client certificate that has a Subject Distinguished Name of CN=user1,O=Progress,C=US Subject DN. After you create a mapping between a client certificate and a broker user, the broker can authenticate the user by using the certificate.

c. Open the **<broker_instance_dir>/etc/artemis-roles.properties** file. Grant users permission to log in to the console by adding them to the role that is specified for the HAWTIO_ROLE variable in the **<broker_instance_dir>/etc/artemis.profile** file. The default value of the HAWTIO_ROLE variable is **amq**. For example:

```
        amq=user1, user2
```

5. Configure the following recommended security properties for the HTTPS protocol.

a. Open the **<broker_instance_dir>/etc/artemis.profile** file.

b. Set the **hawtio.http.strictTransportSecurity** property to allow only HTTPS requests to the AMQ Management Console and to convert any HTTP requests to HTTPS. For example:

```
        hawtio.http.strictTransportSecurity = max-age=31536000; includeSubDomains; preload
```

c. Set the **hawtio.http.publicKeyPins** property to instruct the web browser to associate a specific cryptographic public key with the AMQ Management Console to decrease the risk of "man-in-the-middle" attacks using forged certificates. For example:

```
hawtio.http.publicKeyPins = pin-sha256="..."; max-age=5184000; includeSubDomains
```

4.5. MANAGING BROKERS USING AMQ MANAGEMENT CONSOLE

You can use AMQ Management Console to view information about a running broker and manage the following resources:

- Incoming network connections (acceptors)

- Addresses

- Queues

## 4.5.1. Viewing details about the broker

To see how the broker is configured, in the left menu, click **Artemis**. In the folder tree, the local broker is selected by default.

In the main pane, the following tabs are available:

Status

Displays information about the current status of the broker, such as uptime and cluster information. Also displays the amount of address memory that the broker is currently using. The graph shows this value as a proportion of the **global-max-size** configuration parameter.

Figure 4.2. Status tab
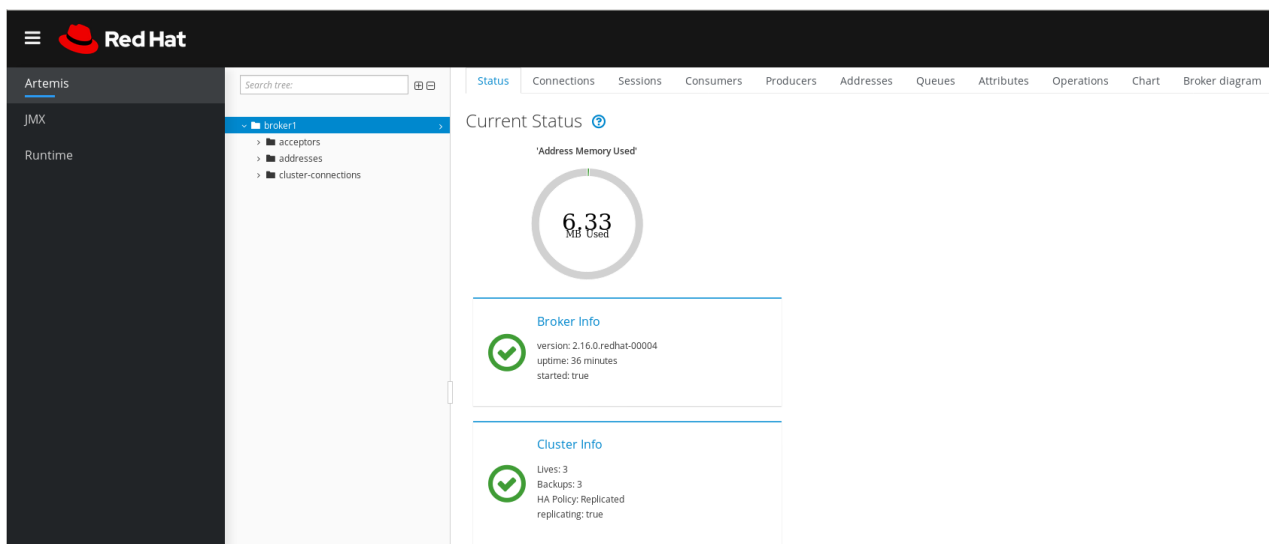
Connections

Displays information about broker connections, including client, cluster, and bridge connections.

Sessions

Displays information about all sessions currently open on the broker.

Consumers

Displays information about all consumers currently open on the broker.

Producers

Displays information about producers currently open on the broker.

Addresses

Displays information about addresses on the broker. This includes internal addresses, such as store-and-forward addresses.

Queues

Displays information about queues on the broker. This includes internal queues, such as store-and-forward queues.

Attributes

Displays detailed information about attributes configured on the broker.

Operations

Displays JMX operations that you can execute on the broker from the console. When you click an operation, a dialog box opens that enables you to specify parameter values for the operation.

Chart

Displays real-time data for attributes configured on the broker. You can edit the chart to specify the attributes that are included in the chart.

Broker diagram

Displays a diagram of the cluster topology. This includes all brokers in the cluster and any addresses and queues on the local broker.

4.5.2. Viewing the broker diagram

You can view a diagram of all AMQ Broker resources in your topology, including brokers (live and backup brokers), producers and consumers, addresses, and queues.
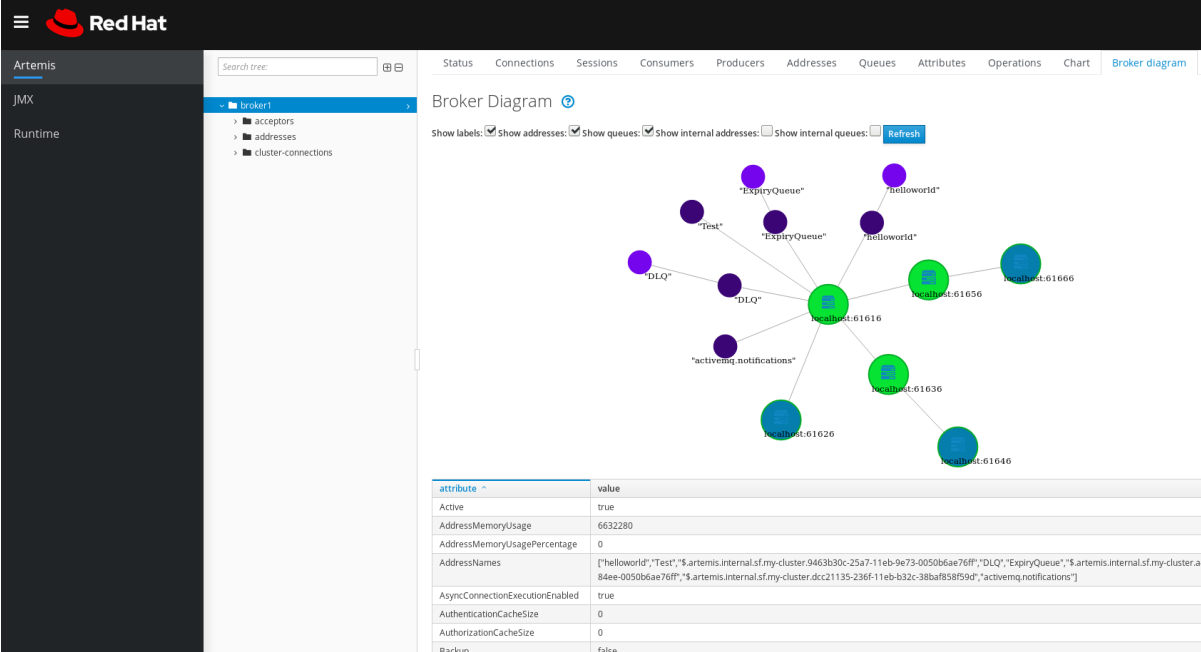
Procedure

1. In the left menu, click **Artemis**.

2. In the main pane, click the **Broker diagram** tab.
   The console displays a diagram of the cluster topology. This includes all brokers in the cluster and any addresses and queues on the local broker, as shown in the figure.

Figure 4.3. Broker diagram tab
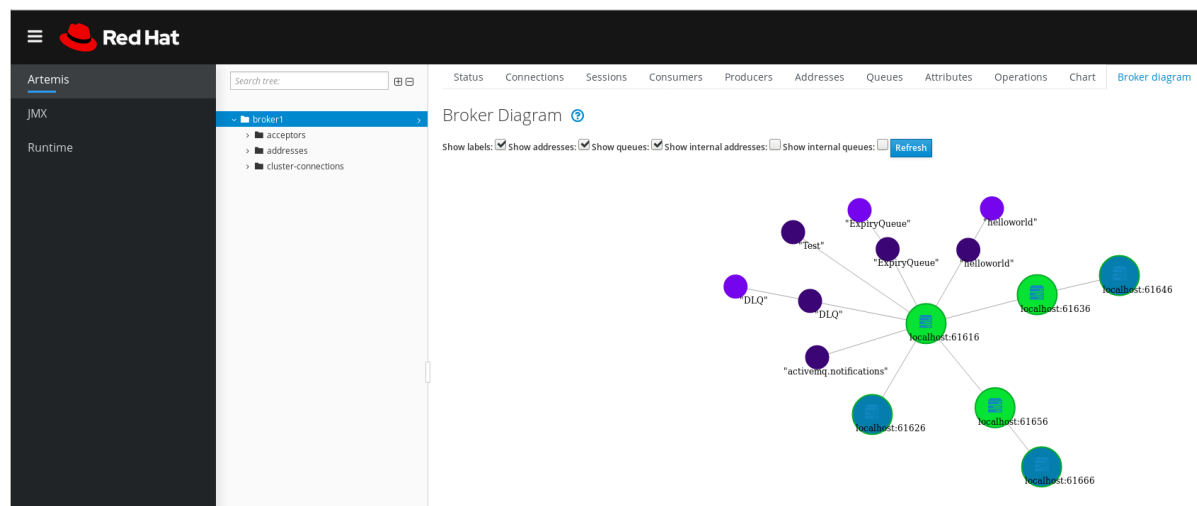
3. To change what items are displayed on the diagram, use the check boxes at the top of the diagram. Click **Refresh**.

4. To show attributes for the local broker or an address or queue that is connected to it, click that node in the diagram. For example, the following figure shows a diagram that also includes attributes for the local broker.

Figure 4.4. Broker diagram tab, including attributes

4.5.3. Viewing acceptors

You can view details about the acceptors configured for the broker.

Procedure

1. In the left menu, click **Artemis**.

2. In the folder tree, click **acceptors**.

3. To view details about how an acceptor is configured, click the acceptor.
   The console shows the corresponding attributes on the **Attributes** tab, as shown in the figure.

Figure 4.5. AMQP acceptor attributes

4. To see complete details for an attribute, click the attribute. An additional window opens to show the details.

4.5.4. Managing addresses and queues

An address represents a messaging endpoint. Within the configuration, a typical address is given a unique name.

A queue is associated with an address. There can be multiple queues per address. Once an incoming message is matched to an address, the message is sent on to one or more of its queues, depending on the routing type configured. Queues can be configured to be automatically created and deleted.

4.5.4.1. Creating addresses

A typical address is given a unique name, zero or more queues, and a routing type.

A routing type determines how messages are sent to the queues associated with an address. Addresses can be configured with two different routing types.

If you want your messages routed to...	Use this routing type...
A single queue within the matching address, in a point-to-point manner.	Anycast
Every queue within the matching address, in a publish-subscribe manner.	Multicast

You can create and configure addresses and queues, and then delete them when they are no longer in use.

Procedure

1. In the left menu, click **Artemis**.

2. In the folder tree, click **addresses**.

3. In the main pane, click the **Create address** tab.
   A page appears for you to create an address, as shown in the figure.

   Figure 4.6. Create Address page
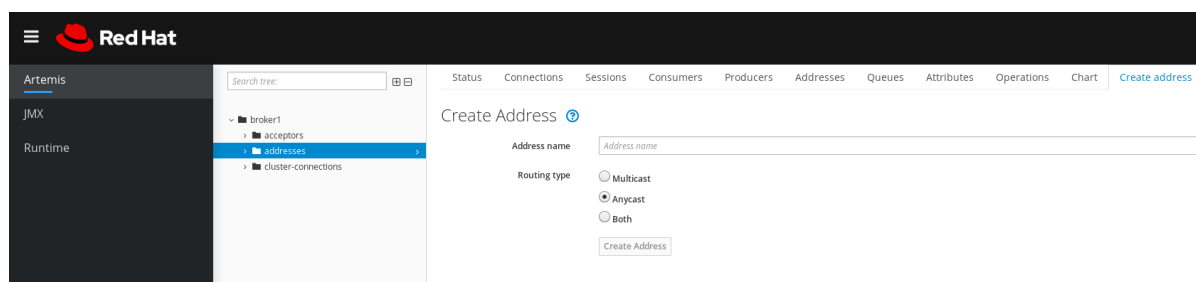
4. Complete the following fields:

   Address name

   The routing name of the address.

   Routing type

   Select one of the following options:

   - **Multicast**: Messages sent to the address will be distributed to all subscribers in a publish-subscribe manner.

- **Anycast**: Messages sent to this address will be distributed to only one subscriber in a point-to-point manner.

- **Both**: Enables you to define more than one routing type per address. This typically results in an anti-pattern and is not recommended.

> **NOTE**
>
> If an address does use both routing types, and the client does not show a preference for either one, the broker defaults to the **anycast** routing type. The one exception is when the client uses the MQTT protocol. In that case, the default routing type is **multicast**.

5. Click **Create Address**.

4.5.4.2. Sending messages to an address

The following procedure shows how to use the console to send a message to an address.

Procedure

1. In the left menu, click **Artemis**.

2. In the folder tree, select an address.

3. On the navigation bar in the main pane, click **More → Send message**.
   A page appears for you to create a message, as shown in the figure.

   Figure 4.7. Send Message page
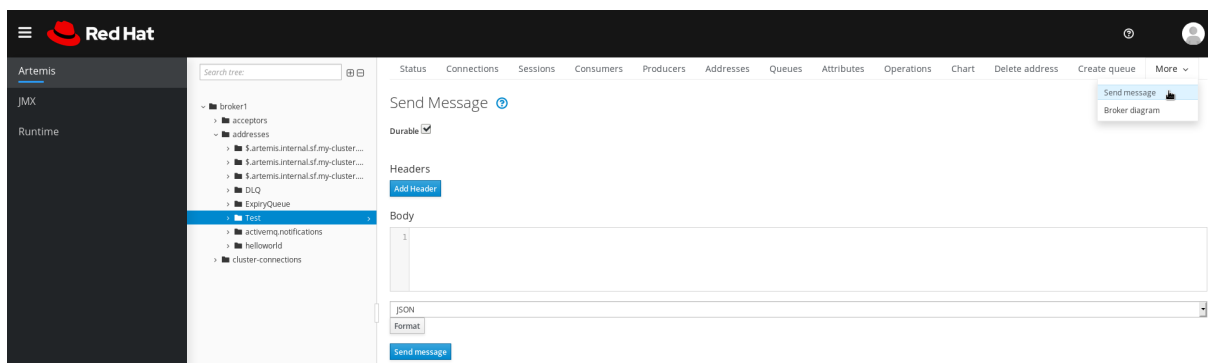
4. If necessary, click the **Add Header** button to add message header information.

5. Enter the message body.

6. In the **Format** drop-down menu, select an option for the format of the message body, and then click **Format**. The message body is formatted in a human-readable style for the format you selected.

7. Click **Send message**.
   The message is sent.

8. To send additional messages, change any of the information you entered, and then click **Send message**.

4.5.4.3. Creating queues

Queues provide a channel between a producer and a consumer.

Prerequisites

- The address to which you want to bind the queue must exist. To learn how to use the console to create an address, see Section 4.5.4.1, "Creating addresses".

Procedure

1. In the left menu, click **Artemis**.

2. In the folder tree, select the address to which you want to bind the queue.

3. In the main pane, click the **Create queue** tab.
   A page appears for you to create a queue, as shown in the figure.

   Figure 4.8. Create Queue page
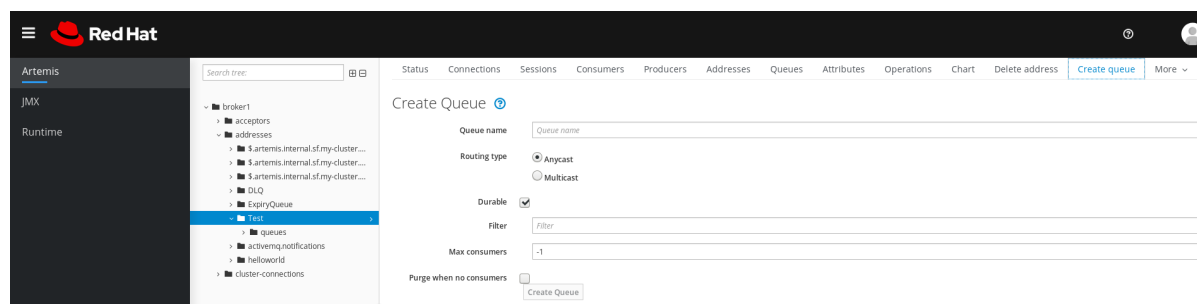
4. Complete the following fields:

   Queue name

   A unique name for the queue.

   Routing type

   Select one of the following options:

   - **Multicast**: Messages sent to the parent address will be distributed to all queues bound to the address.

   - **Anycast**: Only one queue bound to the parent address will receive a copy of the message. Messages will be distributed evenly among all of the queues bound to the address.

   Durable

   If you select this option, the queue and its messages will be persistent.

   Filter

   The username to be used when connecting to the broker.

   Max Consumers

   The maximum number of consumers that can access the queue at a given time.

   Purge when no consumers

   If selected, the queue will be purged when no consumers are connected.

5. Click **Create Queue**.

4.5.4.4. Checking the status of a queue

Charts provide a real-time view of the status of a queue on a broker.

Procedure

1. In the left menu, click **Artemis**.

2. In the folder tree, navigate to a queue.

3. In the main pane, click the **Chart** tab.
   The console displays a chart that shows real-time data for all of the queue attributes.

Figure 4.9. Chart tab for a queue
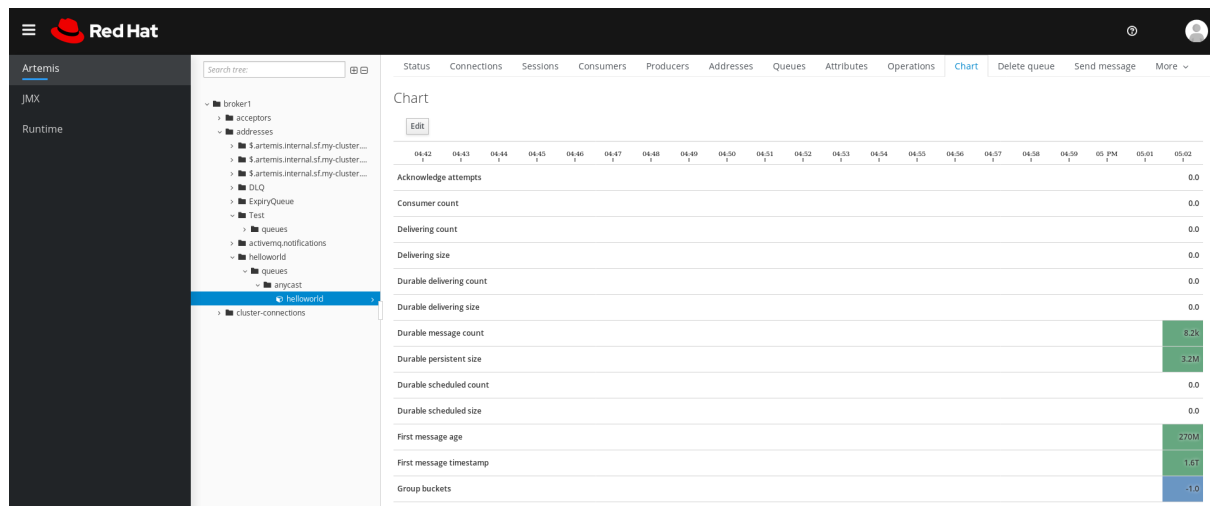
> **NOTE**
>
> To view a chart for multiple queues on an address, select the **anycast** or
> **multicast** folder that contains the queues.

4. If necessary, select different criteria for the chart:

   a. In the main pane, click **Edit**.

   b. On the **Attributes** list, select one or more attributes that you want to include in the chart. To
      select multiple attributes, press and hold the **Ctrl** key and select each attribute.

   c. Click the **View Chart** button. The chart is updated based on the attributes that you
      selected.

4.5.4.5. Browsing queues

Browsing a queue displays all of the messages in the queue. You can also filter and sort the list to find
specific messages.

Procedure

1. In the left menu, click **Artemis**.

2. In the folder tree, navigate to a queue.
   Queues are located within the addresses to which they are bound.

3. On the navigation bar in the main pane, click **More → Browse queue**.
   The messages in the queue are displayed. By default, the first 200 messages are displayed.

   Figure 4.10. Browse Queue page
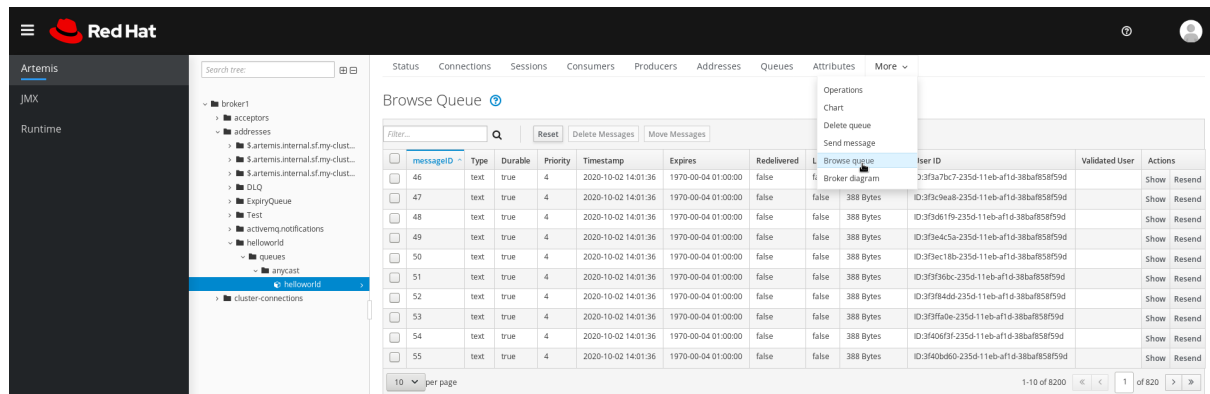
4. To browse for a specific message or group of messages, do one of the following:

To...	Do this...
Filter the list of messages	In the **Filter...** text field, enter filter criteria. Click the search (that is, magnifying glass) icon.
Sort the list of messages	In the list of messages, click a column header. To sort the messages in descending order, click the header a second time.

5. To view the content of a message, click the **Show** button.
   You can view the message header, properties, and body.

4.5.4.6. Sending messages to a queue

After creating a queue, you can send a message to it. The following procedure outlines the steps required to send a message to an existing queue.

Procedure

1. In the left menu, click **Artemis**.

2. In the folder tree, navigate to a queue.

3. In the main pane, click the **Send message** tab.
   A page appears for you to compose the message.

Figure 4.11. Send Message page for a queue
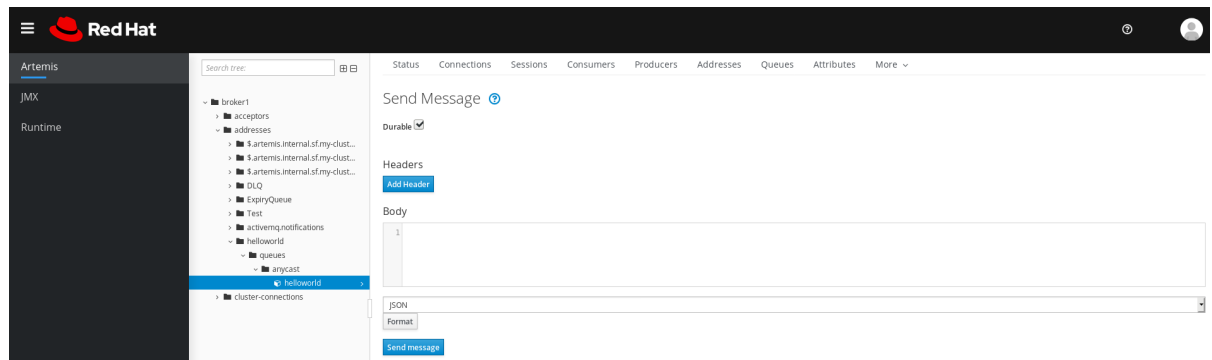
4. If necessary, click the **Add Header** button to add message header information.

5. Enter the message body.

6. In the **Format** drop-down menu, select an option for the format of the message body, and then click **Format**. The message body is formatted in a human-readable style for the format you selected.

7. Click **Send message**. The message is sent.

8. To send additional messages, change any of the information you entered, and click **Send message**.

4.5.4.7. Resending messages to a queue

You can resend previously sent messages.

Procedure

1. Browse for the message you want to resend .

2. Click the check box next to the message that you want to resend.

3. Click the **Resend** button. The message is displayed.

4. Update the message header and body as needed, and then click **Send message**.

4.5.4.8. Moving messages to a different queue

You can move one or more messages in a queue to a different queue.

Procedure

1. Browse for the messages you want to move .

2. Click the check box next to each message that you want to move.

3. In the navigation bar, click **Move Messages**.
   A confirmation dialog box appears.

4. From the drop-down menu, select the name of the queue to which you want to move the messages. Click **Move**.

4.5.4.9. Deleting messages or queues

You can delete a queue or purge all of the messages from a queue.

Procedure

1. Browse for the queue you want to delete or purge .

2. Do one of the following:

To...	Do this...
Delete a message from the queue	1. Click the check box next to each message that you want to delete. 2. Click the **Delete** button.
Purge all messages from the queue	1. On the navigation bar in the main pane, click **Delete queue**. 2. Click the **Purge Queue** button.
Delete the queue	1. On the navigation bar in the main pane, click **Delete queue**. 2. Click the **Delete Queue** button.

CHAPTER 5. MONITORING BROKER RUNTIME METRICS

When you install AMQ Broker, a Prometheus metrics plugin is included in your installation. Prometheus is software built for monitoring large, scalable systems and storing historical runtime data over an extended time period. You must modify the broker configuration to enable the plugin. When enabled, the plugin collects runtime metrics for the broker and exports these to Prometheus format. You can then use Prometheus to review the metrics. You might also use a graphical tool such as Grafana to configure more advanced visualizations of the data.

> **NOTE**
>
> The Prometheus metrics plugin enables you to collect and export broker metrics in Prometheus **format**. However, Red Hat **does not** provide support for installation or configuration of Prometheus itself, nor of visualization tools such as Grafana. If you require support with installing, configuring, or running Prometheus or Grafana, visit the product websites for resources such as community support and documentation.

In addition to the broker metrics collected by the Prometheus plugin, you can modify the broker configuration to capture standard sets of metrics relating to the host Java Virtual Machine (JVM) for the broker. Specifically, you can capture JVM metrics for Garbage Collection (GC), memory, and threads.

The sections that follow describe:

- The metrics that the Prometheus plugin exports

- How to enable the Prometheus plugin

- How to configure the broker to collect JVM metrics

5.1. METRICS OVERVIEW

To monitor the health and performance of your broker instances, you can use the Prometheus plugin for AMQ Broker to monitor and store broker runtime metrics. The AMQ Broker Prometheus plugin exports the broker runtime metrics to Prometheus format, enabling you to use Prometheus itself to visualize and run queries on the data.

You can also use a graphical tool, such as Grafana, to configure more advanced visualizations and dashboards for the metrics that the Prometheus plugin collects.

The metrics that the plugin exports to Prometheus format are described below.

Broker metrics

artemis_address_memory_usage

Number of bytes used by all addresses on this broker for in-memory messages.

artemis_address_memory_usage_percentage

Memory used by all the addresses on this broker as a percentage of the **global-max-size** parameter.

artemis_connection_count

Number of clients connected to this broker.

artemis_total_connection_count

Number of clients that have connected to this broker since it was started.

Address metrics

artemis_routed_message_count

Number of messages routed to one or more queue bindings.

artemis_unrouted_message_count

Number of messages *not* routed to any queue bindings.

Queue metrics

artemis_consumer_count

Number of clients consuming messages from a given queue.

artemis_delivering_durable_message_count

Number of durable messages that a given queue is currently delivering to consumers.

artemis_delivering_durable_persistent_size

Persistent size of durable messages that a given queue is currently delivering to consumers.

artemis_delivering_message_count

Number of messages that a given queue is currently delivering to consumers.

artemis_delivering_persistent_size

Persistent size of messages that a given queue is currently delivering to consumers.

artemis_durable_message_count

Number of durable messages currently in a given queue. This includes scheduled, paged, and in-delivery messages.

artemis_durable_persistent_size

Persistent size of durable messages currently in a given queue. This includes scheduled, paged, and in-delivery messages.

artemis_messages_acknowledged

Number of messages acknowledged from a given queue since the queue was created.

artemis_messages_added

Number of messages added to a given queue since the queue was created.

artemis_message_count

Number of messages currently in a given queue. This includes scheduled, paged, and in-delivery messages.

artemis_messages_killed

Number of messages removed from a given queue since the queue was created. The broker kills a message when the message exceeds the configured maximum number of delivery attempts.

artemis_messages_expired

Number of messages expired from a given queue since the queue was created.

artemis_persistent_size

Persistent size of all messages (both durable and non-durable) currently in a given queue. This includes scheduled, paged, and in-delivery messages.

artemis_scheduled_durable_message_count

Number of durable, scheduled messages in a given queue.

artemis_scheduled_durable_persistent_size

Persistent size of durable, scheduled messages in a given queue.

artemis_scheduled_message_count

Number of scheduled messages in a given queue.

artemis_scheduled_persistent_size

Persistent size of scheduled messages in a given queue.

For higher-level broker metrics that are not listed above, you can calculate these by aggregating lower-level metrics. For example, to calculate total message count, you can aggregate the **artemis_message_count** metrics from all queues in your broker deployment.

For an on-premise deployment of AMQ Broker, metrics for the Java Virtual Machine (JVM) hosting the broker are also exported to Prometheus format. This does not apply to a deployment of AMQ Broker on OpenShift Container Platform.

5.2. ENABLING THE PROMETHEUS METRICS PLUGIN FOR AMQ BROKER

When you install AMQ Broker, a Prometheus metrics plugin is included in your installation. Although the plugin is already configured for use, you need to enable the plugin in your broker configuration. When enabled, the plugin collects runtime metrics for the broker and exports these to Prometheus format.

The following procedure shows how to enable the Prometheus plugin for AMQ Broker.

Procedure

1. Copy the Prometheus metrics plugin **.jar** file from your AMQ Broker 7.10 extracted archive to the **lib** directory of your broker instance.

   ```
   $ cp amq-broker-7.10.0/lib/artemis-prometheus-metrics-plugin-1.0.0.CR1-redhat-00010.jar
   <broker_instance_dir>/lib
   ```

2. Open the **<broker_instance_dir>/etc/broker.xml** configuration file.

3. Enable the Prometheus plugin in the broker configuration. Add a **<metrics>** element with a **<plugin>** sub-element, configured as shown below.

   ```
   <metrics>
       <plugin class-
   name="org.apache.activemq.artemis.core.server.metrics.plugins.ArtemisPrometheusMetricsPlu
   gin"/>
   </metrics>
   ```

4. Save the **broker.xml** configuration file. The metrics plugin starts to gather broker runtime metrics in Prometheus format.

5.3. CONFIGURING THE BROKER TO COLLECT JVM METRICS

The following procedure shows how to configure the broker to collect Java Virtual Machine (JVM) metrics for Garbage Collection (GC), memory, and threads.

Prerequisites

- You have previously enabled the Prometheus metrics plugin in your broker configuration. For more information, see Section 5.2, "Enabling the Prometheus metrics plugin for AMQ Broker" .

Procedure

1. Open the ***<broker_instance_dir>*/etc/broker.xml** configuration file.

2. In the **<metrics>** element that you added to the configuration when enabling the Prometheus metrics plugin, specify whether the broker collects JVM metrics for Garbage Collection (GC), memory, and threads. For example:

```
<metrics>
    <jvm-gc>true</jvm-gc>
    <jvm-memory>true</jvm-memory>
    <jvm-threads>true</jvm-threads>
    <plugin class-
name="org.apache.activemq.artemis.core.server.metrics.plugins.ArtemisPrometheusMetricsPlu
gin"/>
</metrics>
```

> **NOTE**
>
> If you do not explicitly add the **jvm-memory** parameter to your configuration and specify a value, the broker uses a default value of **true**. This means that the broker exports JVM memory metrics by default. The default values of the **jvm-gc** and **jvm-threads** parameters are **false**.

3. Save the **broker.xml** configuration file. The broker starts to gather the JVM metrics that you have enabled. These metrics are also exported to Prometheus format.

5.4. DISABLING METRICS COLLECTION FOR SPECIFIC ADDRESSES

When you configure a metrics plugin for AMQ Broker (for example, the Prometheus metrics plugin), metrics collection is enabled by default. However, within the **address-setting** configuration element of a specific address or *set* of addresses, you can explicitly disable metrics collection.

The following procedure shows how disable metrics collection for a specific address or *set* of addresses.

Procedure

1. Open the ***<broker_instance_dir>*/etc/broker.xml** configuration file.

2. In the **address-setting** element of a matching address or *set* of addresses, add the **enable-metrics** parameter and set the value of the parameter to **false**. For example, the following configuration disables metrics collection for an address called **orders**.

```
<configuration>
 <core>
  ...
  <address-settings>
    <address-setting match="orders">
       ...
       <enable-metrics>false</enable-metrics>
       ...
    </address-setting>
  </address-settings>
```

```
    ...
   </core>
</configuration>
```

5.5. ACCESSING BROKER RUNTIME DATA USING PROMETHEUS

Prerequisites

- To query and visualize the broker runtime data collected by the Prometheus plugin, you need to install Prometheus. For more information, see Installing Prometheus in the Prometheus documentation.

Procedure

1. From your Prometheus installation directory, open the **prometheus.yml** configuration file.

2. In the **static_configs** section of the configuration file, change the **targets** element to **localhost:8161**. This location is where the broker runs its web server. By default, **/metrics** is appended to this host name, forming the full path to the metrics stored on the broker web server.

3. To view the broker runtime metrics collected by the Prometheus plugin, open **localhost:8161/metrics** in a web browser.
   On the resulting web page, you see the current values of the metrics collected by the plugin, based on the queues and addresses that you have configured on the broker. If you have more than one running broker instance in your JVM, you see metrics for each broker.

4. From your Prometheus installation directory, run Prometheus.

   ```
   $ ./prometheus
   ```

   When Prometheus starts, the shell output includes the following line:

   ```
   component=web, msg="Start listening for connections" address=0.0.0.0:9090
   ```

   The preceding line indicates that Prometheus is listening for HTTP traffic on port 9090.

5. To access the Prometheus web console, open **127.0.0.1:9090** in a web browser.

6. In the Prometheus web console, you can use the **Expression** field to create a query on your broker data. The queries you create are based on the Prometheus query language, PromQL. Broker metrics that are available to insert in your query are in the **Insert metric** drop-down list. As a simple example, suppose you want to query the message count on the DLQ queue, over time. In this case, select **artemis_message_count** from the metrics drop-down list. Complete your query by specifying the DLQ queue name and address. This example query is shown below.

   ```
   artemis_message_count{address="DLQ", queue="DLQ"}
   ```

   For more advanced visualizations, you can use regular expressions to create complex queries that overlay several metrics, for example. Or, you can perform mathematical operations on a number of metrics, such as aggregating them. For more information about creating Prometheus queries, see Querying Prometheus in the Prometheus documentation.

CHAPTER 6. USING THE MANAGEMENT API

AMQ Broker has an extensive management API, which you can use to modify a broker's configuration, create new resources (for example, addresses and queues), inspect these resources (for example, how many messages are currently held in a queue), and interact with them (for example, to remove messages from a queue).

In addition, clients can use the management API to manage the broker and subscribe to management notifications.

6.1. METHODS FOR MANAGING AMQ BROKER USING THE MANAGEMENT API

There are two ways to use the management API to manage the broker:

- Using JMX — JMX is the standard way to manage Java applications

- Using the JMS API — management operations are sent to the broker using JMS messages and the AMQ JMS client

Although there are two different ways to manage the broker, each API supports the same functionality. If it is possible to manage a resource using JMX it is also possible to achieve the same result by using JMS messages and the AMQ JMS client.

This choice depends on your particular requirements, application settings, and environment. Regardless of the way you invoke management operations, the management API is the same.

For each managed resource, there exists a Java interface describing what can be invoked for this type of resource. The broker exposes its managed resources in the **org.apache.activemq.artemis.api.core.management** package. The way to invoke management operations depends on whether JMX messages or JMS messages and the AMQ JMS client are used.
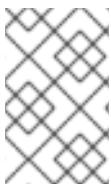
> **NOTE**
>
> Some management operations require a **filter** parameter to choose which messages are affected by the operation. Passing **null** or an empty string means that the management operation will be performed on *all messages*.

6.2. MANAGING AMQ BROKER USING JMX

You can use Java Management Extensions (JMX) to manage a broker. The management API is exposed by the broker using MBeans interfaces. The broker registers its resources with the domain **org.apache.activemq**.

For example, the **ObjectName** to manage a queue named **exampleQueue** is:

```
org.apache.activemq.artemis:broker="__BROKER_NAME__",component=addresses,address="exampleQueue",subcomponent=queues,routingtype="anycast",queue="exampleQueue"
```

The MBean is:

```
org.apache.activemq.artemis.api.management.QueueControl
```

The MBean's **ObjectName** is built using the helper class
org.apache.activemq.artemis.api.core.management.ObjectNameBuilder. You can also use jconsole to find the **ObjectName** of the MBeans you want to manage.

Managing the broker using JMX is identical to management of any Java applications using JMX. It can be done by reflection or by creating proxies of the MBeans.

6.2.1. Configuring JMX management

By default, JMX is enabled to manage the broker. You can enable or disable JMX management by setting the **jmx-management-enabled** property in the **broker.xml** configuration file.

Procedure

1. Open the ***<broker_instance_dir>*/etc/broker.xml** configuration file.

2. Set **<jmx-management-enabled>**.

   > <jmx-management-enabled>true</jmx-management-enabled>

   If JMX is enabled, the broker can be managed locally using **jconsole**.
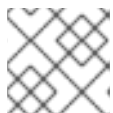
   > **NOTE**
   >
   > Remote connections to JMX are not enabled by default for security reasons.

3. If you want to manage multiple brokers from the same **MBeanServer**, configure the JMX domain for each of the brokers.
   By default, the broker uses the JMX domain **org.apache.activemq.artemis**.

   > <jmx-domain>my.org.apache.activemq</jmx-domain>

   > **NOTE**
   >
   > If you are using AMQ Broker on a Windows system, system properties must be set in **artemis**, or **artemis.cmd**. A shell script is located under ***<install_dir>*/bin**.

Additional resources

- For more information on configuring the broker for remote management, see Oracle's Java Management Guide.

6.2.2. Configuring JMX management access

By default, remote JMX access to a broker is disabled for security reasons. However, AMQ Broker has a JMX agent that allows remote access to JMX MBeans. You enable JMX access by configuring a connector element in the broker **management.xml** configuration file.

NOTE

While it is also possible to enable JMX access using the `com.sun.management.jmxremote` JVM system property, that method is not supported and is not secure. Modifying that JVM system property can bypass RBAC on the broker. To minimize security risks, consider limited access to localhost.

IMPORTANT

Exposing the JMX agent of a broker for remote management has security implications.

To secure your configuration as described in this procedure:

- Use SSL for all connections.

- Explicitly define the connector host, that is, the host and port to expose the agent on.

- Explicitly define the port that the RMI (Remote Method Invocation) registry binds to.

Prerequisites

- A working broker instance

- The Java **jconsole** utility

Procedure

1. Open the ***<broker-instance-dir>*/etc/management.xml** configuration file.

2. Define a connector for the JMX agent. The connector-port setting establishes an RMI registry that clients such as jconsole query for the JMX connector server. For example, to allow remote access on port 1099:

   ```
   <connector connector-port="1099"/>
   ```

3. Verify the connection to the JMX agent using **jconsole**:

   ```
   service:jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi
   ```

4. Define additional properties on the connector, as described below.

   connector-host

   The broker server host to expose the agent on. To prevent remote access, set **connector-host** to **127.0.0.1** (localhost).

   rmi-registry-port

   The port that the JMX RMI connector server binds to. If not set, the port is always random. Set this property to avoid problems with remote JMX connections tunnelled through a firewall.

   jmx-realm

   JMX realm to use for authentication. The default value is **activemq** to match the JAAS configuration.

object-name

Object name to expose the remote connector on. The default value is **connector:name=rmi**.

secured

Specify whether the connector is secured using SSL. The default value is **false**. Set the value to **true** to ensure secure communication.

key-store-path

Location of the keystore. Required if you have set **secured="true"**.

key-store-password

Keystore password. Required if you have set **secured="true"**. The password can be encrypted.

key-store-provider

Keystore provider. Required if you have set **secured="true"**. The default value is **JKS**.

trust-store-path

Location of the truststore. Required if you have set **secured="true"**.

trust-store-password

Truststore password. Required if you have set **secured="true"**. The password can be encrypted.

trust-store-provider

Truststore provider. Required if you have set **secured="true"**. The default value is **JKS**

password-codec

The fully qualified class name of the password codec to use. See the password masking documentation, linked below, for more details on how this works.

5. Set an appropriate value for the endpoint serialization using **jdk.serialFilter** as described in the Java Platform documentation.

Additional resources

- For more information about encrypted passwords in configuration files, see Encrypting Passwords in Configuration Files.

6.2.3. MBeanServer configuration

When the broker runs in standalone mode, it uses the Java Virtual Machine's **Platform MBeanServer** to register its MBeans. By default, Jolokia is also deployed to allow access to the MBean server using REST.

6.2.4. How JMX is exposed with Jolokia

By default, AMQ Broker ships with the Jolokia HTTP agent deployed as a web application. Jolokia is a remote JMX over HTTP bridge that exposes MBeans.

> **NOTE**
>
> To use Jolokia, the user must belong to the role defined by the **hawtio.role** system property in the **<broker_instance_dir>/etc/artemis.profile** configuration file. By default, this role is **amq**.

Example 6.1. Using Jolokia to query the broker's version

This example uses a Jolokia REST URL to find the version of a broker. The **Origin** flag should specify the domain name or DNS host name for the broker server. In addition, the value you specify for **Origin** must correspond to an entry for **<allow-origin>** in your Jolokia Cross-Origin Resource Sharing (CORS) specification.

```
$ curl
http://admin:admin@localhost:8161/console/jolokia/read/org.apache.activemq.artemis:broker=\"0.
0.0.0\"/Version -H "Origin: mydomain.com"
{"request":
{"mbean":"org.apache.activemq.artemis:broker=\"0.0.0.0\"","attribute":"Version","type":"read"},"val
ue":"2.4.0.amq-710002-redhat-1","timestamp":1527105236,"status":200}
```

Additional resources

- For more information on using a JMX-HTTP bridge, see the Jolokia documentation.

- For more information on assigning a user to a role, see Adding Users.

- For more information on specifying Jolokia Cross-Origin Resource Sharing (CORS), see section 4.1.5 of Security.

6.2.5. Subscribing to JMX management notifications

If JMX is enabled in your environment, you can subscribe to management notifications.

Procedure

- Subscribe to **ObjectName org.apache.activemq.artemis:broker="*<broker-name>*"**.

Additional resources

- For more information about management notifications, see Section 6.5, "Management notifications".

6.3. MANAGING AMQ BROKER USING THE JMS API

The Java Message Service (JMS) API allows you to create, send, receive, and read messages. You can use JMS and the AMQ JMS client to manage brokers.

6.3.1. Configuring broker management using JMS messages and the AMQ JMS Client

To use JMS to manage a broker, you must first configure the broker's management address with the **manage** permission.

Procedure

1. Open the *<broker_instance_dir>*/**etc**/**broker.xml** configuration file.

2. Add the **<management-address>** element, and specify a management address.

By default, the management address is **queue.activemq.management**. You only need to specify a different address if you do not want to use the default.

> <management-address>my.management.address</management-address>

3. Provide the management address with the **manage** user permission type.
   This permission type enables the management address to receive and handle management messages.

> <security-setting-match="queue.activemq.management">
>   <permission-type="manage" roles="admin"/>
> </security-setting>

6.3.2. Managing brokers using the JMS API and AMQ JMS Client

To invoke management operations using JMS messages, the AMQ JMS client must instantiate the special management queue.

Procedure

1. Create a **QueueRequestor** to send messages to the management address and receive replies.

2. Create a **Message**.

3. Use the helper class **org.apache.activemq.artemis.api.jms.management.JMSManagementHelper** to fill the message with the management properties.

4. Send the message using the **QueueRequestor**.

5. Use the helper class **org.apache.activemq.artemis.api.jms.management.JMSManagementHelper** to retrieve the operation result from the management reply.

Example 6.2. Viewing the number of messages in a queue

This example shows how to use the JMS API to view the number of messages in the JMS queue **exampleQueue**:

```
Queue managementQueue = ActiveMQJMSClient.createQueue("activemq.management");

QueueSession session = ...
QueueRequestor requestor = new QueueRequestor(session, managementQueue);
connection.start();
Message message = session.createMessage();
JMSManagementHelper.putAttribute(message, "queue.exampleQueue", "messageCount");
Message reply = requestor.request(message);
int count = (Integer)JMSManagementHelper.getResult(reply);
System.out.println("There are " + count + " messages in exampleQueue");
```

6.4. MANAGEMENT OPERATIONS

Whether you are using JMX or JMS messages to manage AMQ Broker, you can use the same API management operations. Using the management API, you can manage brokers, addresses, and queues.

6.4.1. Broker management operations

You can use the management API to manage your brokers.

Listing, creating, deploying, and destroying queues

A list of deployed queues can be retrieved using the **getQueueNames()** method.
Queues can be created or destroyed using the management operations **createQueue()**, **deployQueue()**, or **destroyQueue()** on the **ActiveMQServerControl** (with the **ObjectName org.apache.activemq.artemis:broker="***BROKER_NAME***"** or the resource name **server**).

createQueue will fail if the queue already exists while **deployQueue** will do nothing.

Pausing and resuming queues

The **QueueControl** can pause and resume the underlying queue. When a queue is paused, it will receive messages but will not deliver them. When it is resumed, it will begin delivering the queued messages, if any.

Listing and closing remote connections

Retrieve a client's remote addresses by using **listRemoteAddresses()**. It is also possible to close the connections associated with a remote address using the **closeConnectionsForAddress()** method. Alternatively, list connection IDs using **listConnectionIDs()** and list all the sessions for a given connection ID using **listSessions()**.

Managing transactions

In case of a broker crash, when the broker restarts, some transactions might require manual intervention. Use the the following methods to help resolve issues you encounter.
List the transactions which are in the prepared states (the transactions are represented as opaque Base64 Strings) using the **listPreparedTransactions()** method lists.

Commit or rollback a given prepared transaction using **commitPreparedTransaction()** or **rollbackPreparedTransaction()** to resolve heuristic transactions.

List heuristically completed transactions using the **listHeuristicCommittedTransactions()** and **listHeuristicRolledBackTransactions** methods.

Enabling and resetting message counters

Enable and disable message counters using the **enableMessageCounters()** or **disableMessageCounters()** method.
Reset message counters by using the **resetAllMessageCounters()** and **resetAllMessageCounterHistories()** methods.

Retrieving broker configuration and attributes

The **ActiveMQServerControl** exposes the broker's configuration through all its attributes (for example, **getVersion()** method to retrieve the broker's version, and so on).

Listing, creating, and destroying Core Bridge and diverts

List deployed Core Bridge and diverts using the **getBridgeNames()** and **getDivertNames()** methods respectively.

Create or destroy using bridges and diverts using **createBridge()** and **destroyBridge()** or **createDivert()** and **destroyDivert()** on the **ActiveMQServerControl** (with the **ObjectName org.apache.activemq.artemis:broker="***BROKER_NAME***"** or the resource name **server**).

Stopping the broker and forcing failover to occur with any currently attached clients

Use the **forceFailover()** on the **ActiveMQServerControl** (with the **ObjectName org.apache.activemq.artemis:broker="***BROKER_NAME***"** or the resource name **server**)
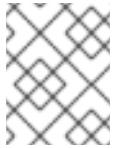
> **NOTE**
>
> Because this method actually stops the broker, you will likely receive an error. The exact error depends on the management service you used to call the method.

6.4.2. Address management operations

You can use the management API to manage addresses.

Manage addresses using the **AddressControl** class with **ObjectName org.apache.activemq.artemis:broker="***<broker-name>***", component=addresses,address="***<address-name>***"** or the resource name **address.***<address-name>*.

Modify roles and permissions for an address using the **addRole()** or **removeRole()** methods. You can list all the roles associated with the queue with the **getRoles()** method.

6.4.3. Queue management operations

You can use the management API to manage queues.

The core management API deals with queues. The **QueueControl** class defines the queue management operations (with the **ObjectName,org.apache.activemq.artemis:broker="***<broker-name>***",component=addresses,address="***<bound-address>***",subcomponent=queues,routing-type="***<routing-type>***",queue="***<queue-name>***"** or the resource name **queue.***<queue-name>*).

Most of the management operations on queues take either a single message ID (for example, to remove a single message) or a filter (for example, to expire all messages with a given property).

Expiring, sending to a dead letter address, and moving messages

Expire messages from a queue using the **expireMessages()** method. If an expiry address is defined, messages are sent to this address, otherwise they are discarded. You can define the expiry address for an address or *set* of addresses (and hence the queues bound to those addresses) in the **address-settings** element of the **broker.xml** configuration file. For an example, see the "Default message address settings" section in Understanding the default broker configuration .
Send messages to a dead letter address using the **sendMessagesToDeadLetterAddress()** method. This method returns the number of messages sent to the dead letter address. If a dead letter address is defined, messages are sent to this address, otherwise they are removed from the queue and discarded. You can define the dead letter address for an address or *set* of addresses (and hence the queues bound to those addresses) in the **address-settings** element of the **broker.xml** configuration file. For an example, see the "Default message address settings" section in Understanding the default broker configuration.

Move messages from one queue to another using the **moveMessages()** method.

Listing and removing messages

List messages from a queue using the **listMessages()** method. It will return an array of **Map**, one **Map** for each message.

Remove messages from a queue using the **removeMessages()** method, which returns a **boolean** for the single message ID variant or the number of removed messages for the filter variant. This method takes a **filter** argument to remove only filtered messages. Setting the filter to an empty string will in effect remove all messages.

Counting messages

The number of messages in a queue is returned by the **getMessageCount()** method. Alternatively, the **countMessages()** will return the number of messages in the queue which match a given filter.

Changing message priority

The message priority can be changed by using the **changeMessagesPriority()** method which returns a **boolean** for the single message ID variant or the number of updated messages for the filter variant.

Message counters

Message counters can be listed for a queue with the **listMessageCounter()** and **listMessageCounterHistory()** methods (see Section 6.6, "Using message counters"). The message counters can also be reset for a single queue using the **resetMessageCounter()** method.

Retrieving the queue attributes

The **QueueControl** exposes queue settings through its attributes (for example, **getFilter()** to retrieve the queue's filter if it was created with one, **isDurable()** to know whether the queue is durable, and so on).

Pausing and resuming queues

The **QueueControl** can pause and resume the underlying queue. When a queue is paused, it will receive messages but will not deliver them. When it is resumed, it will begin delivering the queued messages, if any.

6.4.4. Remote resource management operations

You can use the management API to start and stop a broker's remote resources (acceptors, diverts, bridges, and so on) so that the broker can be taken offline for a given period of time without stopping completely.

Acceptors

Start or stop an acceptor using the **start()** or. **stop()** method on the **AcceptorControl** class (with the **ObjectName org.apache.activemq.artemis:broker="*<broker-name>*",component=acceptors,name="*<acceptor-name>*"** or the resource name **acceptor.*<address-name>***). Acceptor parameters can be retrieved using the **AcceptorControl** attributes. See Network Connections: Acceptors and Connectors for more information about Acceptors.

Diverts

Start or stop a divert using the **start()** or **stop()** method on the **DivertControl** class (with the **ObjectName org.apache.activemq.artemis:broker="*<broker-name>*",component=diverts,name="*<divert-name>*"** or the resource name **divert.*<divert-name>***). Divert parameters can be retrieved using the **DivertControl** attributes.

Bridges

Start or stop a bridge using the **start()** (resp. **stop()**) method on the **BridgeControl** class (with the **ObjectName org.apache.activemq.artemis:broker="*<broker-name>*",component=bridge,name="*<bridge-name>*"** or the resource name **bridge.*<bridge-*

name>). Bridge parameters can be retrieved using the **BridgeControl** attributes.

Broadcast groups

Start or stop a broadcast group using the **start()** or **stop()** method on the **BroadcastGroupControl** class (with the **ObjectName org.apache.activemq.artemis:broker="*<broker-name>*",component=broadcast-group,name="*<broadcast-group-name>*"** or the resource name **broadcastgroup.*<broadcast-group-name>***). Broadcast group parameters can be retrieved using the **BroadcastGroupControl** attributes. See Broker discovery methods for more information.

Discovery groups

Start or stop a discovery group using the **start()** or **stop()** method on the **DiscoveryGroupControl** class (with the **ObjectName org.apache.activemq.artemis:broker="*<broker-name>*",component=discovery-group,name="*<discovery-group-name>*"** or the resource name **discovery.*<discovery-group-name>***). Discovery groups parameters can be retrieved using the **DiscoveryGroupControl** attributes. See Broker discovery methods for more information.

Cluster connections

Start or stop a cluster connection using the **start()** or **stop()** method on the **ClusterConnectionControl** class (with the **ObjectName org.apache.activemq.artemis:broker="*<broker-name>*",component=cluster-connection,name="*<cluster-connection-name>*"** or the resource name **clusterconnection.*<cluster-connection-name>***). Cluster connection parameters can be retrieved using the **ClusterConnectionControl** attributes. See Creating a broker cluster for more information.

6.5. MANAGEMENT NOTIFICATIONS

Below is a list of all the different kinds of notifications as well as which headers are on the messages. Every notification has a **_AMQ_NotifType** (value noted in parentheses) and **_AMQ_NotifTimestamp** header. The time stamp is the unformatted result of a call to **java.lang.System.currentTimeMillis()**.

Notification type	Headers
BINDING_ADDED (0)	**_AMQ_Binding_Type**
	_AMQ_Address
	_AMQ_ClusterName
	_AMQ_RoutingName
	_AMQ_Binding_ID
	_AMQ_Distance
	_AMQ_FilterString

Notification type	Headers
BINDING_REMOVED (1)	**_AMQ_Address** **_AMQ_ClusterName** **_AMQ_RoutingName** **_AMQ_Binding_ID** **_AMQ_Distance** **_AMQ_FilterString**
CONSUMER_CREATED (2)	**_AMQ_Address** **_AMQ_ClusterName** **_AMQ_RoutingName** **_AMQ_Distance** **_AMQ_ConsumerCount** **_AMQ_User** **_AMQ_RemoteAddress** **_AMQ_SessionName** **_AMQ_FilterString**
CONSUMER_CLOSED (3)	**_AMQ_Address** **_AMQ_ClusterName** **_AMQ_RoutingName** **_AMQ_Distance** **_AMQ_ConsumerCount** **_AMQ_User** **_AMQ_RemoteAddress** **_AMQ_SessionName** **_AMQ_FilterString**
SECURITY_AUTHENTICATION_VIOLATION (6)	**_AMQ_User**

Notification type	Headers
SECURITY_PERMISSION_VIOLATION (7)	**_AMQ_Address** **_AMQ_CheckType** **_AMQ_User**
DISCOVERY_GROUP_STARTED (8)	**name**
DISCOVERY_GROUP_STOPPED (9)	**name**
BROADCAST_GROUP_STARTED (10)	**name**
BROADCAST_GROUP_STOPPED (11)	**name**
BRIDGE_STARTED (12)	**name**
BRIDGE_STOPPED (13)	**name**
CLUSTER_CONNECTION_STARTED (14)	**name**
CLUSTER_CONNECTION_STOPPED (15)	**name**
ACCEPTOR_STARTED (16)	**factory** **id**
ACCEPTOR_STOPPED (17)	**factory** **id**
PROPOSAL (18)	**_JBM_ProposalGroupId** **_JBM_ProposalValue** **_AMQ_Binding_Type** **_AMQ_Address** **_AMQ_Distance**

Notification type	Headers
PROPOSAL_RESPONSE (19)	**_JBM_ProposalGroupId** **_JBM_ProposalValue** **_JBM_ProposalAltValue** **_AMQ_Binding_Type** **_AMQ_Address** **_AMQ_Distance**
CONSUMER_SLOW (21)	**_AMQ_Address** **_AMQ_ConsumerCount** **_AMQ_RemoteAddress** **_AMQ_ConnectionName** **_AMQ_ConsumerName** **_AMQ_SessionName**

6.6. USING MESSAGE COUNTERS

You use message counters to obtain information about queues over time. This helps you to identify trends that would otherwise be difficult to see.

For example, you could use message counters to determine how a particular queue is being used over time. You could also attempt to obtain this information by using the management API to query the number of messages in the queue at regular intervals, but this would not show how the queue is actually being used. The number of messages in a queue can remain constant because no clients are sending or receiving messages on it, or because the number of messages sent to the queue is equal to the number of messages consumed from it. In both of these cases, the number of messages in the queue remains the same even though it is being used in very different ways.

6.6.1. Types of message counters

Message counters provide additional information about queues on a broker.

count

The total number of messages added to the queue since the broker was started.

countDelta

The number of messages added to the queue since the last message counter update.

lastAckTimestamp

The time stamp of the last time a message from the queue was acknowledged.

lastAddTimestamp

The time stamp of the last time a message was added to the queue.

messageCount

The current number of messages in the queue.

messageCountDelta

The overall number of messages added/removed from the queue since the last message counter update. For example, if **messageCountDelta** is **-10**, then 10 messages overall have been removed from the queue.

udpateTimestamp

The time stamp of the last message counter update.

> **NOTE**
>
> You can combine message counters to determine other meaningful data as well. For example, to know specifically how many messages were consumed from the queue since the last update, you would subtract the **messageCountDelta** from **countDelta**.

6.6.2. Enabling message counters

Message counters can have a small impact on the broker's memory; therefore, they are disabled by default. To use message counters, you must first enable them.

Procedure

1. Open the **<broker_instance_dir>/etc/broker.xml** configuration file.

2. Enable message counters.

   ```
   <message-counter-enabled>true</message-counter-enabled>
   ```

3. Set the message counter history and sampling period.

   ```
   <message-counter-max-day-history>7</message-counter-max-day-history>
   <message-counter-sample-period>60000</message-counter-sample-period>
   ```

   message-counter-max-day-history

   The number of days the broker should store queue metrics. The default is 10 days.

   message-counter-sample-period

   How often (in milliseconds) the broker should sample its queues to collect metrics. The default is 10000 milliseconds.

6.6.3. Retrieving message counters

You can use the management API to retrieve message counters.

Prerequisites

- Message counters must be enabled on the broker.
  For more information, see Section 6.6.2, "Enabling message counters" .

Procedure

- Use the management API to retrieve message counters.

```
// Retrieve a connection to the broker's MBeanServer.
MBeanServerConnection mbsc = ...
JMSQueueControlMBean queueControl =
(JMSQueueControl)MBeanServerInvocationHandler.newProxyInstance(mbsc,
   on,
   JMSQueueControl.class,
   false);

// Message counters are retrieved as a JSON string.
String counters = queueControl.listMessageCounter();

// Use the MessageCounterInfo helper class to manipulate message counters more easily.
MessageCounterInfo messageCounter = MessageCounterInfo.fromJSON(counters);
System.out.format("%s message(s) in the queue (since last sample: %s)\n",
messageCounter.getMessageCount(),
messageCounter.getMessageCountDelta());
```

Additional resources

- For more information about message counters, see Section 6.4.3, "Queue management operations".

CHAPTER 7. MONITORING BROKERS FOR PROBLEMS

AMQ Broker includes an internal tool called the *Critical Analyzer* that actively monitors running brokers for problems such as deadlock conditions. In a production environment, a problem such as a deadlock condition can be caused by IO errors, a defective disk, memory shortage, or excess CPU usage caused by other processes.

The Critical Analyzer periodically measures the response time for critical operations such as queue delivery (that is, adding of messages to a queue on the broker) and journal operations. If the response time of a checked operation exceeds a configurable timeout value, the broker is considered unstable. In this case, you can configure the Critical Analyzer to simply log a message or take action to protect the broker, such as shutting down the broker or stopping the virtual machine (VM) that is running the broker.

7.1. CONFIGURING THE CRITICAL ANALYZER

The following procedure shows how to configure the Critical Analyzer to monitor the broker for problems.

Procedure

1. Open the ***<broker_instance_dir>*/etc/broker.xml** configuration file.
   The default configuration for the Critical Analyzer is shown below.

   ```
   <critical-analyzer>true</critical-analyzer>
   <critical-analyzer-timeout>120000</critical-analyzer-timeout>
   <critical-analyzer-check-period>60000</critical-analyzer-check-period>
   <critical-analyzer-policy>HALT</critical-analyzer-policy>
   ```

2. Specify parameter values, as described below.

   critical-analyzer

   Specifies whether to enable or disable the Critical Analyzer tool. The default value is **true**, which means that the tool is enabled.

   critical-analyzer-timeout

   Timeout, in milliseconds, for the checks run by the Critical Analyzer. If the time taken by one of the checked operations exceeds this value, the broker is considered unstable.

   critical-analyzer-check-period

   Time period, in milliseconds, between consecutive checks by the Critical Analyzer for each operation.

   critical-analyzer-policy

   If the broker fails a check and is considered unstable, this parameter specifies whether the broker logs a message (**LOG**), stops the virtual machine (VM) hosting the broker (**HALT**), or shuts down the broker (**SHUTDOWN**).

   Based on the policy option that you have configured, if the response time for a critical operation exceeds the configured timeout value, you see output that resembles one of the following:

   critical-analyzer-policy=LOG

   ```
   [Artemis Critical Analyzer] 18:11:52,145 WARN [org.apache.activemq.artemis.core.server]
   AMQ224081: The component
   ```

> org.apache.activemq.artemis.tests.integration.critical.CriticalSimpleTest$2@5af97850 is not responsive

critical-analyzer-policy=HALT

> [Artemis Critical Analyzer] 18:10:00,831 ERROR [org.apache.activemq.artemis.core.server] AMQ224079: The process for the virtual machine will be killed, as component org.apache.activemq.artemis.tests.integration.critical.CriticalSimpleTest$2@5af97850 is not responsive

critical-analyzer-policy=SHUTDOWN

> [Artemis Critical Analyzer] 18:07:53,475 ERROR [org.apache.activemq.artemis.core.server] AMQ224080: The server process will now be stopped, as component org.apache.activemq.artemis.tests.integration.critical.CriticalSimpleTest$2@5af97850 is not responsive

You also see a thread dump on the broker that resembles the following:

> [Artemis Critical Analyzer] 18:10:00,836 WARN [org.apache.activemq.artemis.core.server] AMQ222199: Thread dump: AMQ119001: Generating thread dump
> *
> ===
> ==== AMQ119002: Thread Thread[Thread-1 (ActiveMQ-scheduled-threads),5,main] name = Thread-1 (ActiveMQ-scheduled-threads) id = 19 group = java.lang.ThreadGroup[name=main,maxpri=10] sun.misc.Unsafe.park(Native Method)
> java.util.concurrent.locks.LockSupport.park(LockSupport.java:175)
> java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.await(AbstractQueuedSynchronizer.java:2039)
> java.util.concurrent.ScheduledThreadPoolExecutor$DelayedWorkQueue.take(ScheduledThreadPoolExecutor.java:1088)
> java.util.concurrent.ScheduledThreadPoolExecutor$DelayedWorkQueue.take(ScheduledThreadPoolExecutor.java:809)
> java.util.concurrent.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:1067)
> java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1127)
> java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
> java.lang.Thread.run(Thread.java:745)
> ===
> ==== ....
> ===
> ==== AMQ119003: End Thread dump *

Revised on 2024-02-21 09:49:44 UTC