# Red Hat AMQ 7.5

# Managing AMQ Broker

For Use with AMQ Broker 7.5

# Red Hat AMQ 7.5 Managing AMQ Broker

For Use with AMQ Broker 7.5

## Legal Notice

## Abstract

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

# Table of Contents

# CHAPTER 1. UPGRADING YOUR BROKER

## 1.1. 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. For instructions on how to upgrade from previous releases of AMQ Broker, see Migrating to Red Hat AMQ 7 .

**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.

## 1.2. 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

### 1.2.1. 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. Restart the broker by entering the following command:

```
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]

## 1.2.2. 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 file contents into the directory by right–clicking on the zip file and choosing **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 on 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=*NEW_INSTALL_DIR*

8. Restart the broker entering the following command:

   > *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]

## 1.3. 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 the AMQ Console only from the local host by default. You must modify the configuration in ***BROKER_INSTANCE_DIR*/etc/jolokia-access.xml** to enable remote access. For more information, see  Securing AMQ Console and AMQ Broker Connections.

- Upgrading from 7.0.x to 7.1.0 on Linux

- Upgrading from 7.0.x to 7.1.0 on Windows

### 1.3.1. 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 .

## 1.3.2. 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 on 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 .

## 1.4. 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 the AMQ Console only from the local host by default. You must modify the configuration in **BROKER_INSTANCE_DIR/etc/jolokia-access.xml** to enable remote access. For more information, see Securing AMQ Console and AMQ Broker Connections.

- Upgrading from 7.1.x to 7.2.0 on Linux

- Upgrading from 7.1.x to 7.2.0 on Windows

## 1.4.1. 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. Restart the broker by entering the following command:

> *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.

## 1.4.2. 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 file contents into the directory by right–clicking on the zip file and choosing **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 on 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=*NEW_INSTALL_DIR*

8. Restart the broker entering the following command:

> *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.

## 1.5. 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.

### 1.5.1. 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 the AMQ Console only from the local host by default. You must modify the configuration in **BROKER_INSTANCE_DIR/etc/jolokia-access.xml** to enable remote access. For more information, see Securing AMQ Console and AMQ Broker Connections.

- Upgrading from 7.2.x to 7.3.0 on Linux

- Upgrading from 7.2.x to 7.3.0 on Windows

## 1.5.2. 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. Restart the broker by entering the following command:

```
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.

## 1.5.3. 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 file contents into the directory by right-clicking on the zip file and choosing **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 on 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=*NEW_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=*NEW_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 argument to reference the correct log manager version.

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

10. Restart the broker entering the following command:

> *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.

## 1.6. 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 the AMQ Console only from the local host by default. You must modify the configuration in **BROKER_INSTANCE_DIR/etc/jolokia-access.xml** to enable remote access. For more information, see Securing AMQ Console and AMQ Broker Connections.

- Upgrading from 7.3.0 to 7.4.0 on Linux

- Upgrading from 7.3.0 to 7.4.0 on Windows

### 1.6.1. 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. Restart the 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.

## 1.6.2. 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 file contents into the installation directory. 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. Select **Copy**.

   b. Right click in the same window. 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=NEW_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 argument to reference the correct log manager version and dependent file.

   ```
   <argument>-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</argument>
   ```

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. Restart the 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.

## 1.7. 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.
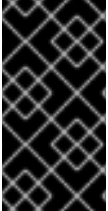
IMPORTANT

Starting with AMQ Broker 7.1.0, you can access the AMQ Console only from the local host by default. You must modify the configuration in **BROKER_INSTANCE_DIR/etc/jolokia-access.xml** to enable remote access. For more information, see Securing AMQ Console and AMQ Broker Connections.

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

## 1.7.1. 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. Restart the 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.

## 1.7.2. 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 file contents into the installation directory. 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. Select **Copy**.

   b. Right click in the same window. 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=*NEW_INSTALL_DIR*

8. Restart the 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.

## 1.8. 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.
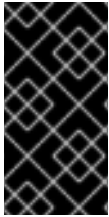
> **IMPORTANT**
>
> Starting with AMQ Broker 7.1.0, you can access the AMQ Console only from the local host by default. You must modify the configuration in **BROKER_INSTANCE_DIR/etc/jolokia-access.xml** to enable remote access. For more information, see Securing AMQ Console and AMQ Broker Connections.

- Upgrading from 7.4.x to 7.5.0 on Linux

- Upgrading from 7.4.x to 7.5.0 on Windows

### 1.8.1. 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. Restart the 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.

## 1.8.2. 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 file contents into the installation directory. 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. Select **Copy**.

   b. Right click in the same window. 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=*NEW_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 argument to reference the correct log manager version and dependent file.

   > <argument>-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</argument>

10. Restart the 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.

# CHAPTER 2. USING AMQ CONSOLE

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

AMQ Console is based on hawtio.

## 2.1. 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 Console

  - Command-line Interface (CLI)

  - Management API

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

## 2.2. ACCESSING AMQ CONSOLE

After installing AMQ Console, you can log in and connect to the brokers in your environment. Access AMQ Console through a single broker instance, regardless of how many brokers are installed in your environment.
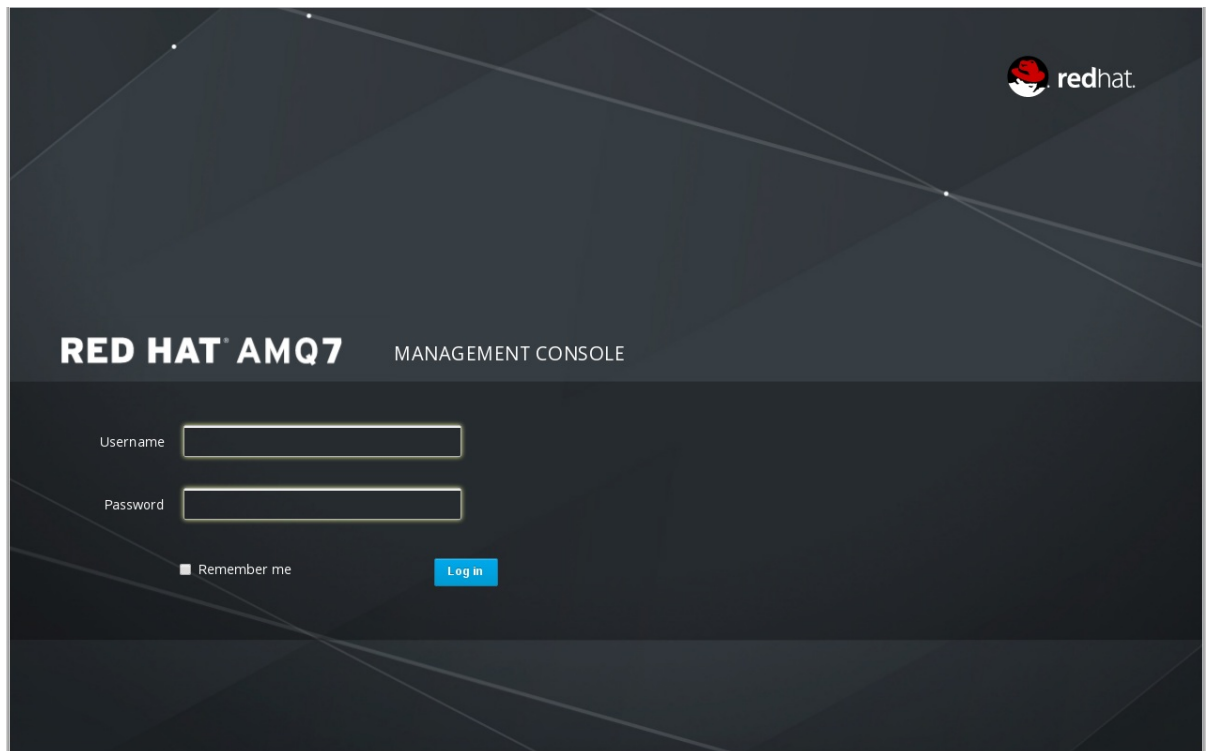
**Procedure**

1. Start the AMQ broker instances that you want to manage in AMQ Console.

2. Navigate to the web console address for the broker instance that you started.

> **NOTE**
>
> The web console address is **http://*HOST*:*PORT*/console/login**. If you are using the default address, navigate to http://localhost:8161/console/login.

3. Log in to AMQ Console using the default username and password that you created when you created the broker instance.

**Additional resources**

For more information on getting started with the broker, see Starting the broker in *Getting Started with AMQ Broker*.

## 2.3. CONFIGURING AMQ CONSOLE

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

### 2.3.1. Setting up user access to AMQ Console

You can access AMQ Console using the broker login credentials. The following table provides information about different methods to add additional broker users to access AMQ 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. <br><br> For more information, see Enabling 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 Console using these credentials. <br><br> For more information, see Enabling Password Authentication in *Configuring AMQ Broker*. |

| Authentication Method | Description |
| --- | --- |
| LDAP Authentication | Users are authenticated and authorized by checking the credentials against user data stored in a central X.500 directory server.<br><br>For more information, see Adding Certificate-Based Authentication in *Configuring AMQ Broker.* |

## 2.3.2. Securing AMQ Console and AMQ Broker connections

To allow AMQ Console to access resources on the broker, specify the permitted origin URLs that can access it by editing the **allow-origin** parameters in the access management configuration file on the broker.

### Prerequisites

- You must first upgrade to AMQ Broker 7.1.0, during which the 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 ***&lt;broker-instance-dir&gt;*/etc/jolokia-access.xml** file in a text editor.

2. Within the **&lt;cors&gt;** section, edit the **allow-origin** settings to add each URL that you want to allow to access AMQ Console. For example:

   ```
   <cors>
      <!-- allow access to web console from localhost -->
      <allow-origin>https://localhost:8161/*</allow-origin>
      <!-- Check for the proper origin on the server side, too -->
      <strict-checking/>
   </cors>
   ```

3. Save the file.

### Additional resources

- For more information on Cross-Origin Resource Sharing, see W3C Recommendations.

- For more information on security commands, see Jolokia Protocols.

## 2.3.3. Securing network access to AMQ Console

To secure AMQ Console when it 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 ***&lt;broker-instance-dir&gt;*/etc/**:

- Java KeyStore (.jks)

- Java TrustStore (only if you want to require client authentication)

**Procedure**

1. Open the ***<broker-instance-dir>*/etc/bootstrap.xml** file.

2. In the **<web>** element, add the following attributes:

   ```
   <web bind="https://localhost:8161"
       path="web"
       keyStorePath="<path_to_KeyStore>"
       keyStorePassword="<password>"
       clientAuth="<true/false>"
       trustStorePath="<path_to_TrustStore>"
       trustStorePassword="<password>">
       ...
   </web>
   ```

| Attribute | Description |
|---|---|
| **bind** | Change the URI scheme to **https**. |
| **keyStorePath** | The path of the KeyStore file. For example: <br><br> keyStorePath="${artemis.instance}/etc/keystore.jks" |
| **keyStorePassword** | The KeyStore's password. |
| **clientAuth** | Specifies whether client authentication is required. The default is **false**, but you can change it to**true** to enable authentication. |
| **trustStorePath** | The path of the TrustStore file. This attribute is only needed if **clientAuth** is **true**. |
| **trustStorePassword** | The TrustStore's password. |

## 2.4. MONITORING YOUR AMQ BROKER DEPLOYMENT

You can use the AMQ Console dashboard page to monitor the status of AMQ Broker. You can also create your own dashboards to display the real-time charts, diagrams, and metrics most important to you.

### 2.4.1. Viewing a dashboard

Dashboards provide you with real-time data about your AMQ Broker environment.

**Procedure**

1. In AMQ Console, click the **Dashboard** tab.
   The **Monitor** dashboard appears, displaying real-time data about *hawtio*.
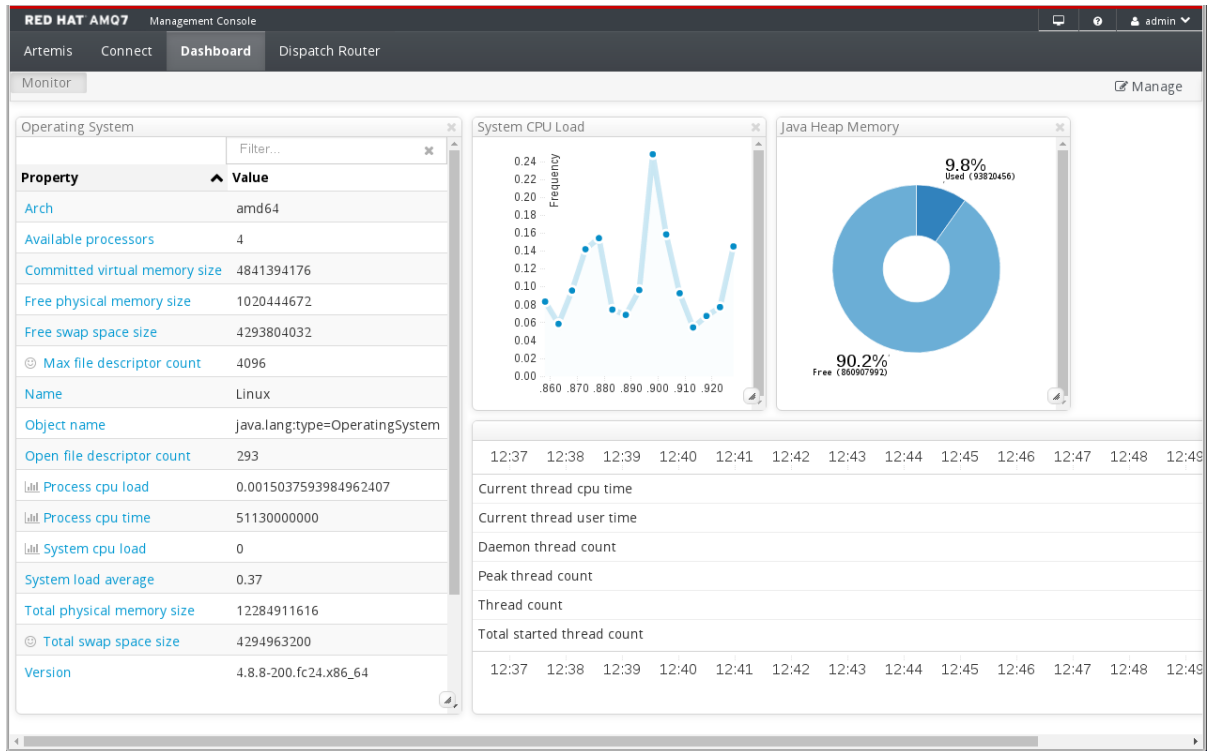
2. To switch to a different dashboard, click the dashboard tab.

## 2.4.2. Creating a new dashboard

Dashboards contain widgets, each of which can display a chart, diagram, or metrics. You can create as many dashboards as needed.

**Procedure**

1. In AMQ Console, click the **Dashboard** tab.

2. On the navigation bar, click **Manage**.

> **NOTE**
>
> The **Manage** page appears displaying a list of existing dashboards.



3. Do one of the following:

| To... | Do this... |
|---|---|
| Create a new, blank dashboard | Click **Create**. |

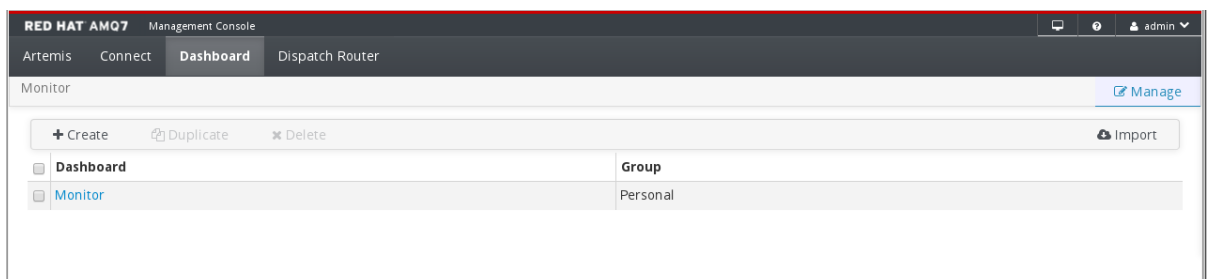| To... | Do this... |
|-------|-----------|
| Create a dashboard similar to an existing dashboard | 1. Click the checkbox next to an existing dashboard.<br><br>2. Click **Duplicate**. |

4. To change the name of the dashboard:

    a. Hover over the dashboard name and click the pencil icon (   ).

    b. Enter a new name for the dashboard and then click the checkmark icon (   ).

### 2.4.3. Creating AMQ Broker dashboards

You can create new dashboards to display real-time data for AMQ Broker.

**Procedure**

1. Create a new dashboard .

2. Add AMQ Broker data to the dashboard .

3. Change the dashboard layout as needed.

### 2.4.4. Adding AMQ Broker data to the AMQ Console dashboard

Add any of the available queue and topic charts to a dashboard.

**Procedure**

1. Click the **Artemis** tab.

2. On the navigation bar, click the **add** icon (   ).
   The **Dashboard** tab appears, displaying a list of available dashboards.

3. Select the dashboard (or dashboards) which you want the chart to appear, and then click **Add View To Dashboard**. The chart is added to the dashboards you selected.

### 2.4.5. Changing the layout of a dashboard

Dashboards contain widgets, which display metrics, diagrams, and charts. You can change the way these widgets are displayed on a dashboard.

| To... | Do this... |
|-------|-----------|
| Move or rearrange widgets | Click and drag a widget to a new position on the dashboard. |

| To... | Do this... |
|---|---|
| Change the title of a widget | 1. Hover over the widget's title bar and click the edit icon ( ✏️ ). <br><br> 2. Enter a name for the widget and then click the checkmark icon ( ✅ ). |
| Resize a widget | In the bottom-right corner of the widget, click and drag the resize icon ( ◢ ). |
| Remove a widget from the dashboard | In the widget's title bar, click the close icon ( ✖ ). |

## 2.5. MANAGING AMQ BROKER

You can use AMQ Console to view important information about AMQ Broker brokers and manage the following resources:

- Incoming network connections (acceptors)

- Addresses

- Queues

### 2.5.1. Viewing details about the broker

View configuration properties and their values to see how the broker is configured.

**Procedure**

- On the **Artemis** tab, in the folder tree, select a broker.
  A list of configuration properties are displayed for the broker.

  **Connections**

  Displays information about the client connections.

  **Sessions**

  Displays information about the client sessions.

  **Consumers**

  Displays information about the client consumers.

  **Producers**

  Displays information about the session producers.

  **Addresses**

  Displays information about the addresses.

  **Queues**

  Displays information about the queues.

  **Diagrams**

Displays diagram of all AMQ Broker resources in your topology, including brokers (masters and slaves), producers and consumers, addresses, and queues.

**Attributes**
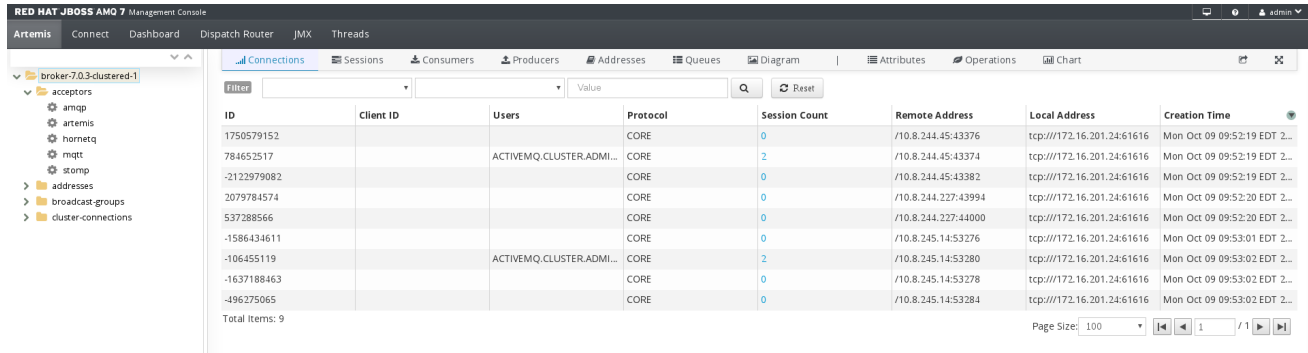
Displays information about the configured attributes.

**Operations**

Displays information about the operations that can be executed on the server.

**Chart**

Displays real-time data for the selected attributes.



## 2.5.2. Viewing the broker diagram

You can view a diagram of all AMQ Broker resources in your topology, including brokers (masters and slaves), producers and consumers, addresses, and queues.

**Procedure**

1. On the **Artemis** tab, click **Diagram**.
   This example shows three brokers with 10 queues.
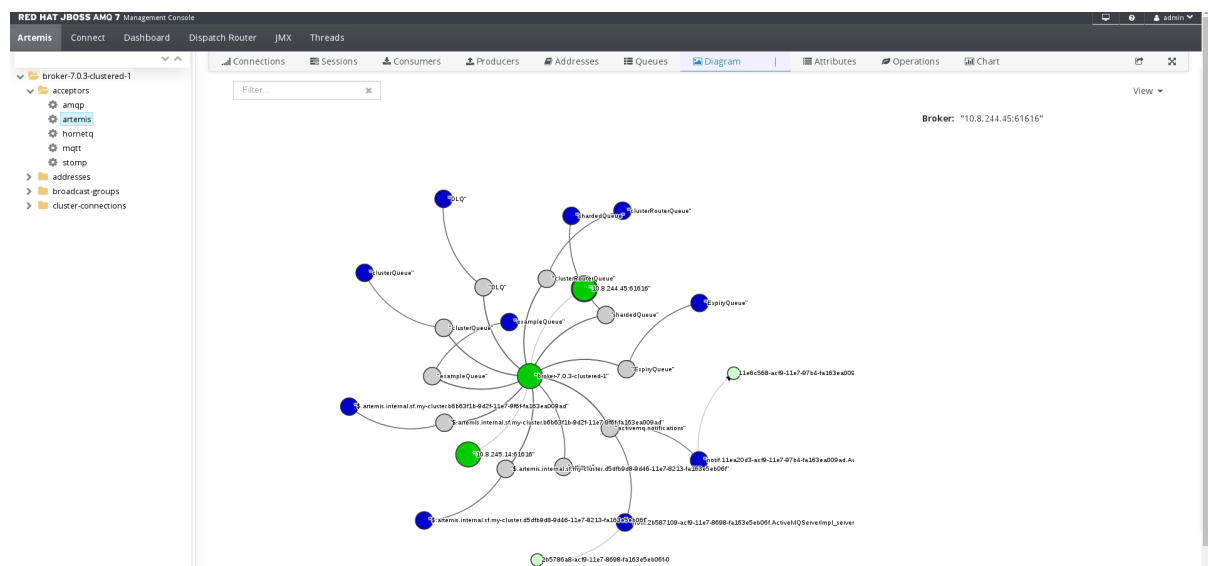


2. To change what objects are displayed on the diagram, click the **View** drop-down and select the items that you want to be displayed.

## 2.5.3. Viewing acceptors

You can view details about the acceptors configured for the broker.

**Procedure**

1. On the **Artemis** tab, in the folder tree, expand the **acceptors** folder.

2. Click an acceptor to view details about how it is configured.
   This example shows the configuration properties for the **amqp** acceptor, which is the default acceptor provided for the AMQP protocol:
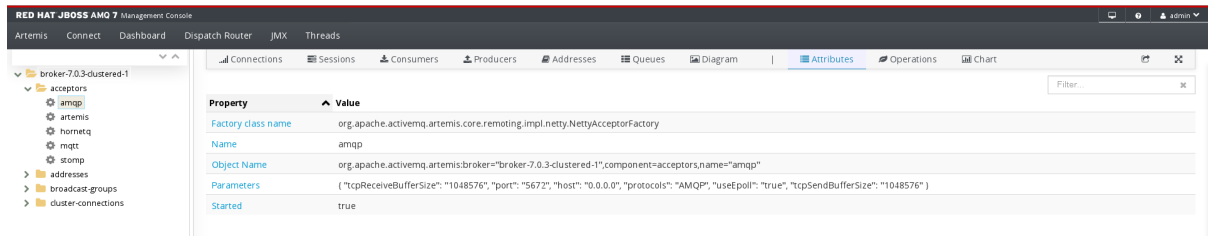


## 2.5.4. Managing addresses and queues

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.

### 2.5.4.1. Creating addresses

A typical address is given a unique name, 0 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 folder tree, select a broker.

2. On the navigation bar, click ⌄ drop-down icon, and then click **Create**.
   A page appears for creating an address.
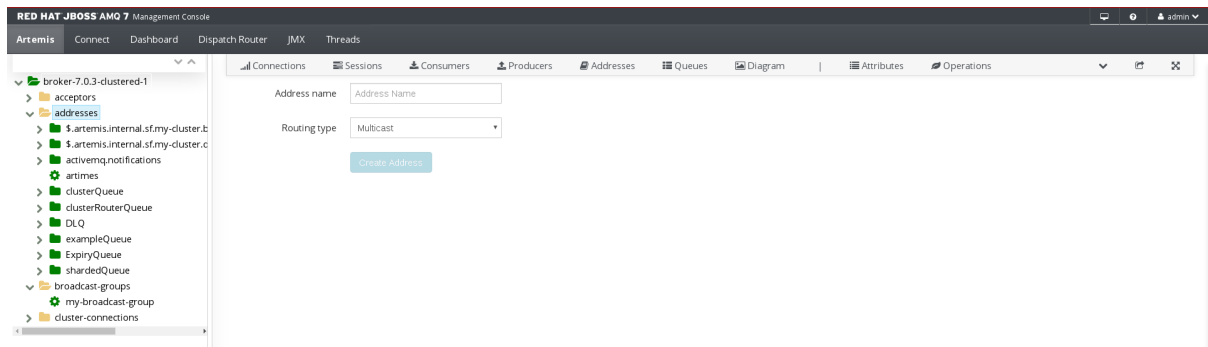
3. Complete the following fields:

   **Address name**

   The routing name of the address.

   **Routing type**

   Select one of the following options:

   **Multicast**

   Messages sent to this 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 therefore not recommended.

   NOTE

   If an address does use both routing types, however, and the client does not show a preference for either one, the broker typically 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**.

4. Click **Create Address**.

### 2.5.4.2. Sending messages to an address

The following procedure outlines the steps required to send a message to an address.

Procedure

1. In the folder tree, select an address.

2. On the navigation bar, click ⌄ drop-down icon, and then click **Send**.
   A page appears for you to compose the message.

3. If necessary, click the **Header** button to add message header information.

4. Enter the message body.

5. In the **Payload format** drop-down, select an option for the format of the message body, and then click **Auto format**. The message body is formatted in a human-readable style for the format you selected.

6. Click **Send message**. The message is sent.

7. To send additional messages, change any of the information you entered, and then click **Send message**.

### 2.5.4.3. 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.

**Procedure**

1. In the folder tree, select the address to which you want to bind the queue.

2. On the navigation bar, click  drop-down icon, and then click **Create**.
   A page appears for you to create the queue.
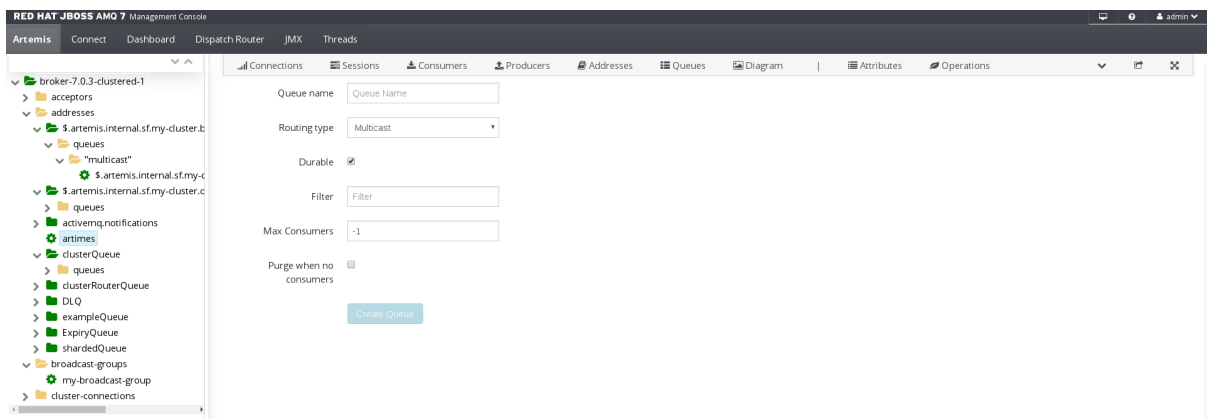


3. Complete the following fields:

**Queue name**

> A unique name for the queue.

**Routing type**

> Select one of the following options:

**Multicast**

> Messages sent to this 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.

4. Click **Create Queue**.

The queue is created. You can access it in the folder tree under the address to which it is bound. Queues for an address are organized into a **Queues** folder. Within the **Queues** folder, queues are further organized by routing type (**MULTICAST** and **ANYCAST**).

In this example, the **clusterQueue** queue is located within the **clusterQueue** address:
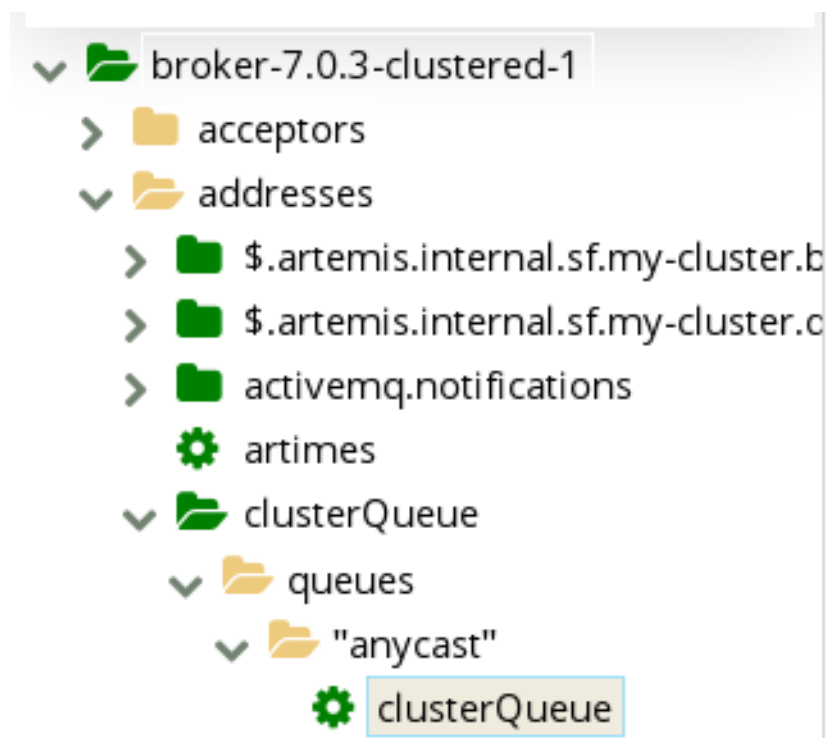


### 2.5.4.4. Checking the status of a queue

Charts provide a real-time view of the status of a queue on a broker.

**Procedure**

1. In the folder tree, navigate to a queue.
   To view a chart for multiple queues for an address, select the **ANYCAST** or **MULTICAST** folder that contains the queues.

2. On the navigation bar, click ⌄ drop-down icon, and then click **Chart**.
   A chart is displayed showing real-time data for all of the queue's attributes.
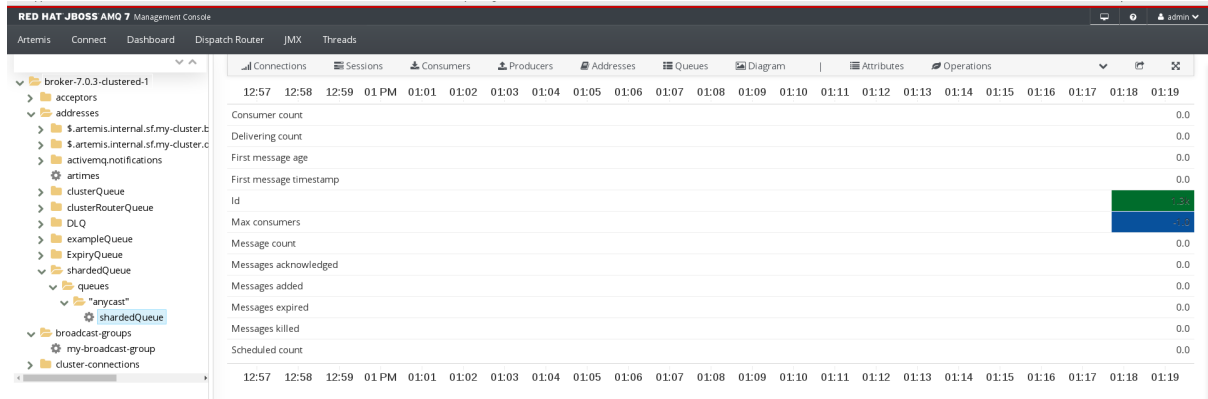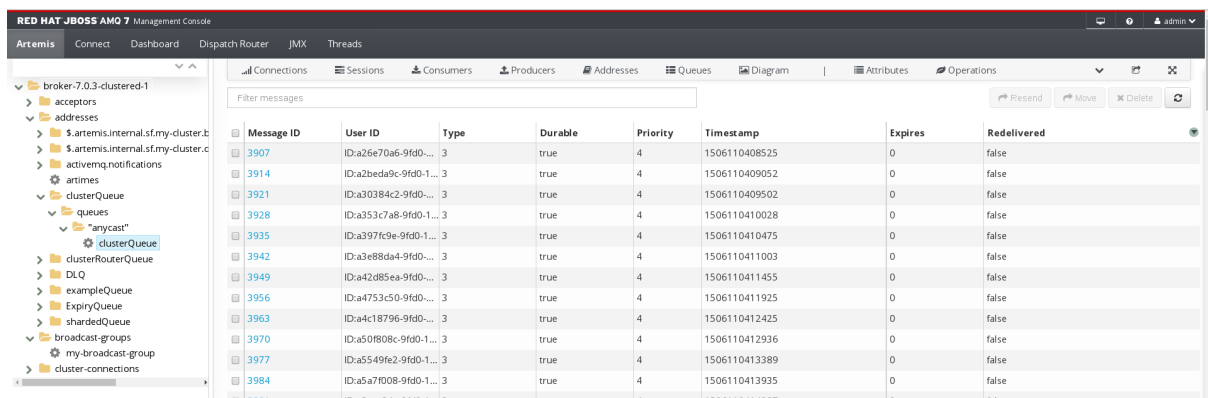


3. If necessary, select different criteria for the chart:

   a. On the navigation bar, click ⌄ drop-down icon, and then click **Edit Chart**.

   b. In 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 displayed based on the criteria you selected.

## 2.5.4.5. 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 folder tree, navigate to a queue.
   Queues are located within the address to which they are bound.

2. On the navigation bar, click ⌄ drop-down icon, and then click **Browse**.
   The messages in the queue are displayed. By default, the first 200 messages are displayed.

3. 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 messages** text field, enter a filter criteria and then press **Enter**. |
| 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. |

4. To view the content of a message, click the message ID.
   You can view the message header, properties, and body.

### 2.5.4.6. 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 folder tree, select the queue to which you want to send the message.

2. On the navigation bar, click ⌄ drop-down icon, and then click **Send**.
   A page appears for you to compose the message.



3. If necessary, click the **Header** button to add message header information.

4. Enter the message body.

5. In the **Payload format** drop-down, select an option for the format of the message body, and then click **Auto format**. The message body is formatted in a human-readable style for the format you selected.

6. Click **Send message**. The message is sent.

7. To send additional messages, change any of the information you entered, and click **Send message**.

### 2.5.4.7. Resending messages to a queue

You can resend previously sent messages.

**Procedure**

1. Browse for the message you want to resend .

2. Click the checkbox 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**.

### 2.5.4.8. 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 checkbox next to each message that you want to move.

3. Click the **Move** button.
   A confirmation dialog box appears.



4. Enter the name of the queue to which you want to move the messages, and then click **Move**.

### 2.5.4.9. Deleting 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... |
| --- | --- |

| To... | Do this... |
| --- | --- |
| Delete a message from the queue | 1. Click the checkbox next to each message you want to delete.<br><br>2. Click the **Delete** button. |
| Purge all messages from the queue | 1. On the navigation bar, click **Delete**.<br><br>2. Click the **Purge queue** button. |
| Delete the queue | 1. On the navigation bar, click **Delete**.<br><br>2. Click the **Delete queue** button. |

# CHAPTER 3. USING 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 |/ __ \ | _ \    | |
    / \ | \ / | | | | | |_) |_ __ ___ | | ____ _ __
   / /\ \ | |\/| | | | | |  _ <| '__/ _ \| | // _ ' __|
  / ____ \| | | | |__| | | |_) | | | () | |  < __/ |
 /_/    \_\_| |_|\___\_\ |____/|_| \___/|_|\_\___|_|


   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 a broker instance

Stop the broker manually by issuing the **stop** command. Immediately after the command is entered, all connections to the broker are forcefully stopped and the shutdown process begins.

**Procedure**

- Stop the broker.

  - If you are running the broker on Linux, issue the following command:

    *<broker-instance-dir>*\bin\artemis stop

  - If you are running the broker on Windows as a service, issue the following command:

    *<broker-instance-dir>*\bin\artemis-service.exe stop

### 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.
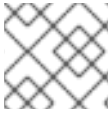


NOTE

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.

   ```
   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 the Core protocol, your interceptor must implement the **org.apache.artemis.activemq.api.core.Interceptor** interface.

   ```
   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. 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**). |
| 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 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.

## 4.1. 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*.

## 4.2. 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.

## 4.2.1. 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.

## 4.2.2. 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.

## 4.2.3. 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 4.1. 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.

### 4.2.4. 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 4.5, "Management notifications".

## 4.3. 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.

### 4.3.1. 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>
   ```

## 4.3.2. 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 4.2. 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");
```

## 4.4. 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.

### 4.4.1. 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.

## 4.4.2. 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.

## 4.4.3. 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 will be sent to it, otherwise they are discarded. The queue's expiry address can be set with the **setExpiryAddress()** method.

Send messages to a dead letter address with the **sendMessagesToDeadLetterAddress()** method. It returns the number of messages which are sent to the dead letter address. If a dead letter address is not defined, messages are removed from the queue and discarded. The queue's dead letter address can be set with the **setDeadLetterAddress()** method.

Move messages from one queue to another by 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 4.6, "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.

## 4.4.4. 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.

## 4.5. 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**<br><br>**_AMQ_Address**<br><br>**_AMQ_ClusterName**<br><br>**_AMQ_RoutingName**<br><br>**_AMQ_Binding_ID**<br><br>**_AMQ_Distance**<br><br>**_AMQ_FilterString** |
| **BINDING_REMOVED** (1) | **_AMQ_Address**<br><br>**_AMQ_ClusterName**<br><br>**_AMQ_RoutingName**<br><br>**_AMQ_Binding_ID**<br><br>**_AMQ_Distance**<br><br>**_AMQ_FilterString** |

| Notification type | Headers |
|---|---|
| **CONSUMER_CREATED** (2) | **_AMQ_Address**<br><br>**_AMQ_ClusterName**<br><br>**_AMQ_RoutingName**<br><br>**_AMQ_Distance**<br><br>**_AMQ_ConsumerCount**<br><br>**_AMQ_User**<br><br>**_AMQ_RemoteAddress**<br><br>**_AMQ_SessionName**<br><br>**_AMQ_FilterString** |
| **CONSUMER_CLOSED** (3) | **_AMQ_Address**<br><br>**_AMQ_ClusterName**<br><br>**_AMQ_RoutingName**<br><br>**_AMQ_Distance**<br><br>**_AMQ_ConsumerCount**<br><br>**_AMQ_User**<br><br>**_AMQ_RemoteAddress**<br><br>**_AMQ_SessionName**<br><br>**_AMQ_FilterString** |
| **SECURITY_AUTHENTICATION_VIOLATION** (6) | **_AMQ_User** |
| **SECURITY_PERMISSION_VIOLATION** (7) | **_AMQ_Address**<br><br>**_AMQ_CheckType**<br><br>**_AMQ_User** |
| **DISCOVERY_GROUP_STARTED** (8) | **name** |
| **DISCOVERY_GROUP_STOPPED** (9) | **name** |
| **BROADCAST_GROUP_STARTED** (10) | **name** |
| **BROADCAST_GROUP_STOPPED** (11) | **name** |

| Notification type | Headers |
|---|---|
| **BRIDGE_STARTED** (12) | **name** |
| **BRIDGE_STOPPED** (13) | **name** |
| **CLUSTER_CONNECTION_STARTED** (14) | **name** |
| **CLUSTER_CONNECTION_STOPPED** (15) | **name** |
| **ACCEPTOR_STARTED** (16) | **factory**<br><br>**id** |
| **ACCEPTOR_STOPPED** (17) | **factory**<br><br>**id** |
| **PROPOSAL** (18) | **_JBM_ProposalGroupId**<br><br>**_JBM_ProposalValue**<br><br>**_AMQ_Binding_Type**<br><br>**_AMQ_Address**<br><br>**_AMQ_Distance** |
| **PROPOSAL_RESPONSE** (19) | **_JBM_ProposalGroupId**<br><br>**_JBM_ProposalValue**<br><br>**_JBM_ProposalAltValue**<br><br>**_AMQ_Binding_Type**<br><br>**_AMQ_Address**<br><br>**_AMQ_Distance** |
| **CONSUMER_SLOW** (21) | **_AMQ_Address**<br><br>**_AMQ_ConsumerCount**<br><br>**_AMQ_RemoteAddress**<br><br>**_AMQ_ConnectionName**<br><br>**_AMQ_ConsumerName**<br><br>**_AMQ_SessionName** |

## 4.6. 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.

## 4.6.1. 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.

**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.

**lastAddTimestamp**

The time stamp of the last time a message was added to 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**.

## 4.6.2. 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 (10 seconds).

## 4.6.3. 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 4.6.2, "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 4.4.3, "Queue management operations".

# CHAPTER 5. MONITORING BROKER RUNTIME DATA USING PROMETHEUS

## 5.1. 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. Prometheus is software built for monitoring large, scalable systems and storing historical runtime data over an extended time period. 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.

For AMQ Broker, the metrics exported to Prometheus format by the plugin are listed below. A description of each metric is exported along with the metric itself.

**Broker Metrics**

- **address.memory.usage**

- **connection.count**

- **total.connection.count**

**Address Metrics**

- **routed.message.count**

- **unrouted.message.count**

**Queue Metrics**

- **consumer.count**

- **delivering.durable.message.count**

- **delivering.durable.persistent.size**

- **delivering.message.count**

- **delivering.persistent.size**

- **durable.message.count**

- **durable.persistent.size**

- **messages.acknowledged**

- **messages.added**

- **message.count**

- **messages.killed**

- **messages.expired**

- **persistent.size**

- **scheduled.durable.message.count**

- **scheduled.durable.persistent.size**

- **scheduled.message.count**

- **scheduled.persistent.size**

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 **message.count** metrics from all queues in your broker deployment.

Java Virtual Machine (JVM) memory metrics are also exported to Prometheus format.

## 5.2. ENABLING THE PROMETHEUS PLUGIN FOR AMQ BROKER

When you install AMQ Broker, the Prometheus plugin is included in your installation and is already configured for use. However, you need to enable the plugin in your broker configuration.

The following procedure shows you how to enable the Prometheus plugin and start gathering metrics for running broker instances.

**Procedure**

1. Copy the Prometheus metrics plugin **.jar** file from your AMQ Broker 7.5 extracted archive to the **lib** directory of your broker instance.

   ```
   $ cp amq-broker-7.5.0/lib/artemis-prometheus-metrics-plugin-1.0.0.CR1-redhat-00002.jar
   BROKER_INSTANCE_DIR/lib
   ```

2. Add the following line to your **broker.xml** configuration file.

   ```
   <metrics-plugin class-
   name="org.apache.activemq.artemis.core.server.metrics.plugins.ArtemisPrometheusMetricsPlu
   gin"/>
   ```

When you have enabled the Prometheus plugin, the plugin starts to gather the metrics listed in the previous section, for any broker that you start.

## 5.3. 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.

**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.

*Revised on 2020-04-22 14:05:31 UTC*