Red Hat AMQ Broker 7.12

OVERVIEW

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

KEY FEATURES

AMQ Broker provides the following features:

- Clustering and high availability options
- Fast, native-IO persistence
- Supports local transactions
- Supports XA transactions when using AMQ Core Protocol JMS and AMQ OpenWire JMS clients
- Written in Java for broad platform support
- Multiple management interfaces: AMQ Management Console, Management APIs, and JMX

SUPPORTED STANDARDS AND PROTOCOLS

AMQ Broker supports the following standards and protocols:

- Wire protocols:
  - Core Protocol
  - AMQP 1.0
  - MQTT
  - OpenWire (Used by A-MQ 6 clients)
  - STOMP
- JMS 2

UNDERSTANDING AMQ BROKER

AMQ Broker enables you to loosely couple heterogeneous systems together, while providing reliability, transactions, and many other features. Before using AMQ Broker, you should understand the capabilities it offers.

BROKER INSTANCES

In AMQ Broker, the installed AMQ Broker software serves as a "home" for one or more *broker instances*. This architecture provides several benefits, such as:

- You can create as many broker instances as you require from a single AMQ Broker installation. The AMQ Broker installation contains the necessary binaries and resources that each broker instance needs to run. These resources are then shared between the broker instances.
- When upgrading to a new version of AMQ Broker, you only need to update the software once, even if you are running multiple broker instances on that host.

You can think of a broker instance as a message broker. Each broker instance has its own directory containing its unique configuration and runtime data. This runtime data consists of logs and data files, and is associated with a unique broker process on the host.

MESSAGE PERSISTENCE

AMQ Broker persists message data to ensure that messages are never lost, even if the broker fails or shuts down unexpectedly. AMQ Broker provides two options for message persistence: journal-based persistence and database persistence.

Journal-based persistence

The default method, this option writes message data to message journal files stored on the file system. Initially, each of these journal files is created automatically with a fixed size and filled with empty data. As clients perform various broker operations, records are appended to the journal. When one of the journal files is full, the broker moves to the next journal file.

Journal-based persistence supports transactional operations, including both local and XA transactions.

Journal-based persistence requires an IO interface to the file system. AMQ Broker supports the following:

Linux Asynchronous IO (AIO)

AIO typically provides the best performance.

Java NIO

Java NIO provides good performance, and it can run on any platform with a Java 6 or later runtime.

Database persistence

This option stores message and bindings data in a database by using Java Database Connectivity (JDBC). This option is a good choice if you already have a reliable and high performing database platform in your environment, or if using a database is mandated by company policy.

The broker JDBC persistence store uses a standard JDBC driver to create a JDBC connection that stores message and bindings data in database tables. The data in the database tables is encoded using the same encoding as journal-based persistence. This means that messages stored in the database are not human-readable if accessed directly using SQL.

To use database persistence, you must use a supported database platform.

RESOURCE CONSUMPTION

AMQ Broker provides a number of options to limit memory and resource consumption on the broker.

Resource limits

You can set connection and queue limits for each user. This prevents users from consuming too many of the broker's resources and causing degraded performance for other users.

Message paging

Message paging enables AMQ Broker to support large queues containing millions of messages while also running with a limited amount of memory. When the broker receives a surge of messages that exceeds its memory capacity, it begins paging messages to disk. This paging process is transparent; the broker pages messages into and out of memory as needed.

Message paging is address-based. When the size of all messages in memory for an address exceeds the maximum size, each additional message for the address will be paged to the address's page file.

Large messages

With AMQ Broker, you can send and receive huge messages, even when running with limited memory resources. To avoid the overhead of storing large messages in memory, you can configure AMQ Broker to store these large messages in the file system or in a database table.

MONITORING AND MANAGEMENT

AMQ Broker provides several tools you can use to monitor and manage your brokers.

AMQ Management Console

AMQ Management Console is a web interface accessible through a web browser. You can use to monitor network health, view broker topology, and create and delete broker resources.

CLI

AMQ Broker provides the **artemis** CLI, which you can use to administer your brokers. Using the CLI, you can create, start, and stop broker instances. The CLI also provides several commands for managing the message journal.

Management API

AMQ Broker provides an extensive management API. You can use it to modify a broker's configuration, create new resources, inspect these resources, and interact with them. Clients can also use the management API to manage the broker and subscribe to management notifications. AMQ Broker provides the following methods for using the management API:

- Java Management Extensions (JMX) - JMX is a standard technology for managing Java applications. The broker's management operations are exposed through AMQ MBeans interfaces.
- JMS API - Management operations are sent using standard JMS messages to a special management JMS queue.

Logs

Each broker instance logs error messages, warnings, and other broker-related information and activities. You can configure the logging levels, the location of the log files, and log format. You can then use the resulting log files to monitor the broker and diagnose error conditions.

INSTALLING AMQ BROKER

AMQ Broker is distributed as a platform-independent archive file. To install AMQ Broker on your system, you must download the archive and extract the contents. You should also understand the directories included in the archive.

Prerequisites

- The host on which you are installing AMQ Broker must meet the AMQ Broker supported configurations.

DOWNLOADING THE AMQ BROKER ARCHIVE

AMQ Broker is distributed as a platform-independent archive file. You can download it from the Red Hat Customer Portal.

Prerequisites

- You must have a Red Hat subscription.

Procedure

1. In a web browser, navigate to https://access.redhat.com/downloads/ and log in. The Product Downloads page is displayed.
2. In the JBoss Integration and Automation section, click the Red Hat AMQ Brokerlink. The Software Downloads page is displayed.
3. Select the desired AMQ Broker version from the Version drop-down menu.
4. On the Releases tab, click the Download link for the specific AMQ Broker file you want to download.

EXTRACTING THE AMQ BROKER ARCHIVE ON LINUX

If you are installing AMQ Broker on Red Hat Enterprise Linux, create a new user account for AMQ Broker, and then extract the contents from the installation archive.

Procedure

5. Create a new user named **amq-broker** and provide it a password.

```
$ sudo useradd amq-broker
$ sudo passwd amq-broker
```

6. Create the directory **/opt/redhat/amq-broker** and make the new **amq-broker** user and group the owners of it.

```
$ sudo mkdir /opt/redhat
$ sudo mkdir /opt/redhat/amq-broker
$ sudo chown -R amq-broker:amq-broker /opt/redhat/amq-broker
```

7. Change the owner of the archive to the new user.

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

8. Move the installation archive to the directory you just created.

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

9. As the new **amq-broker** user, extract the contents by using the **unzip** command.

```
$ su - amq-broker
$ cd /opt/redhat/amq-broker
$ unzip <archive_name>.zip
$ exit
```

A directory named something similar to **apache-artemis-2.33.0.redhat-00010** is created. In the documentation, this location is referred to as ***<install_dir>***.

EXTRACTING THE AMQ BROKER ARCHIVE ON WINDOWS SYSTEMS

If you are installing AMQ Broker on a Windows system, create a new directory folder for AMQ Broker, and then extract the contents there.

Procedure

10. Use Windows Explorer to create the directory folder **\redhat\amq-broker** on the desired drive letter.
    For example: **C:\redhat\amq-broker**

11. Use Windows Explorer to move the installation archive to the directory you just created.

12. In the **\redhat\amq-broker** directory, right-click the installation archive zip file and select
    Extract All.
    A directory named something similar to **apache-artemis-2.33.0.redhat-00010** is created. In the documentation, this location is referred to as ***<install_dir>***.

UNDERSTANDING THE AMQ BROKER INSTALLATION ARCHIVE CONTENTS

The directory created by extracting the archive is the top-level directory for the AMQ Broker installation. This directory is referred to as ***<install_dir>***, and includes the following contents:

This directory…	Contains…
***<install_dir>*/web/api**	API documentation.
***<install_dir>*/bin**	Binaries and scripts needed to run AMQ Broker.
***<install_dir>*/etc**	Configuration files.

This directory…	Contains…
***<install_dir>*/lib**	JARs and libraries needed to run AMQ Broker.
***<install_dir>*/schema**	XML schemas used to validate AMQ Broker configuration.
***<install_dir>*/web**	The web context loaded when AMQ Broker runs.

CREATING A STANDALONE BROKER

You can get started quickly with AMQ Broker by creating a standalone broker instance on your local machine, starting it, and producing and consuming some test messages.

Prerequisites

- AMQ Broker must be installed.

CREATING A BROKER INSTANCE

A broker instance is a directory containing the configuration and runtime data for a broker. To create a new broker instance, you first create a directory for the broker instance, and then use the **artemis create** command to create the broker instance.

This procedure demonstrates how to create a simple, standalone broker on your local machine. The broker uses a basic, default configuration, and accepts connections from clients using any of the supported messaging protocols.

Procedure

1. Create a directory for the broker instance.

If you are using...	Do this...
Red Hat Enterprise Linux	1. Create a new directory to serve as the location for the broker instance. **$ sudo mkdir /var/opt/amq-broker** 2. Assign the user that you created during installation. **$ sudo chown -R amq-broker:amq-broker /var/opt/amq-broker**
Windows	Use Windows Explorer to create a new folder to serve as the location for the broker instance.

2. Use the **artemis create** command to create the broker.

If you are using...	Do this...
Red Hat Enterprise Linux	1. Switch to the user account you created during installation. **$ su - amq-broker** 2. Change to the directory you just created for the broker instance. **$ cd /var/opt/amq-broker** 3. From the broker instance's directory, create the broker instance. **$ <*install_dir*>/bin/artemis create mybroker**
Windows	1. Open a command prompt from the directory you just created forthe broker instance. 2. From the broker instance's directory, create the broker instance. **> <*install_dir*>\bin\artemis.cmd create mybroker**

3. Follow the **artemis create** prompts to configure the broker instance.

Example 4.1. Configuring a broker instance using**artemis create**

```
$ /opt/redhat/amq-broker/bin/artemis create mybroker

Creating ActiveMQ Artemis instance at: /var/opt/amq-broker/mybroker

--user: is mandatory with this configuration:
Please provide the default username:
admin

--password: is mandatory with this configuration:
Please provide the default password:

--role: is mandatory with this configuration:
Please provide the default role:
```

amq

--allow-anonymous | --require-login: is mandatory with this configuration: Allow anonymous access? (Y/N):
Y

Auto tuning journal ...
done! Your system can make 19.23 writes per millisecond, your journal-buffer-timeout will be 52000

You can now start the broker by executing:

"/var/opt/amq-broker/mybroker/bin/artemis" run

Or you can run the broker in the

background using: "/var/opt/amq-

broker/mybroker/bin/artemis-service" start

STARTING THE BROKER INSTANCE

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

Procedure

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

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

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

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

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

  ___|_|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/b
  indings,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.

PRODUCING AND CONSUMING TEST MESSAGES

After starting the broker, you should verify that it is running properly. This involves producing a few test messages, sending them to the broker, and then consuming them.

Procedure

5. Use the **artemis producer** command to produce a few test messages and send them to the broker.
   This command sends 100 messages to the **helloworld** address, which is created automatically on the broker. The producer connects to the broker by using the default port 61616, which accepts all supported messaging protocols.

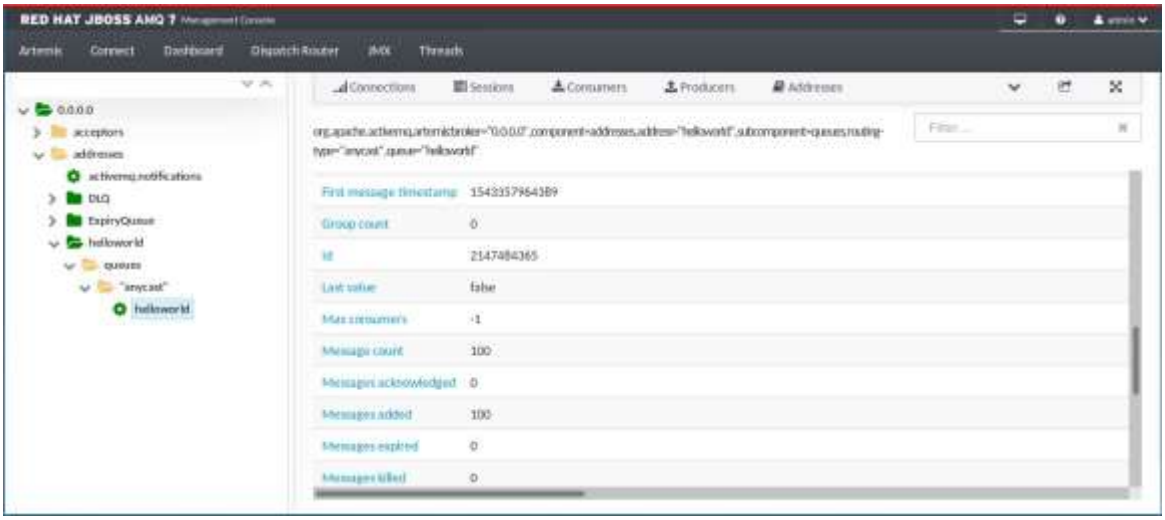
   ```
   $ /opt/redhat/amq-broker/amq-broker-7.2.0/bin/artemis producer --
   destination helloworld --message-count 100 --url
   tcp://localhost:61616
   Producer ActiveMQQueue[helloworld], thread=0 Started to calculate
   elapsed time ...

   Producer ActiveMQQueue[helloworld], thread=0
   Produced: 100 messages Producer
   ActiveMQQueue[helloworld], thread=0 Elapsed time in
   second : 1 s
   Producer ActiveMQQueue[helloworld], thread=0 Elapsed time in milli
   second : 1289 milli seconds
   ```

6. Use the web console to see the messages stored in the broker.
   a. In a web browser, navigate to http://localhost:8161.
   b. Log into the console using the default username and default password that you created when you created the broker instance.
      The **Attributes** tab is displayed.
   c. On the **Attributes** tab, navigate to addresses → helloworld → queues → "anycast" → helloworld.
      In the previous step, you sent messages to the **helloworld** address. This created a new anycast **helloworld** address with a queue (also named **helloworld**). The **Message count** attribute shows that all 100

messages that were sent to **helloworld** are currently stored in this queue.

Figure 4.1. Message count



Use the **artemis consumer** command to consume 50 of the messages stored on the broker. This command consumes 50 of the messages that you sent to the broker previously.

```
$ /opt/redhat/amq-broker/amq-broker-7.2.0/bin/artemis consumer --
destination helloworld -- message-count 50 --url
tcp://localhost:61616

Consumer:: filter = null
Consumer ActiveMQQueue[helloworld], thread=0 wait until 50
messages are consumedConsumer ActiveMQQueue[helloworld],
thread=0 Consumed: 50 messages
Consumer ActiveMQQueue[helloworld], thread=0 Consumer thread
finished
```

7. In the web console, verify that the **Message count** is now 50.
   50 of the messages were consumed, which leaves 50 messages stored in the **helloworld** queue.

8. Stop the broker and verify that the 50 remaining messages are still stored in the **helloworld**
   queue.

   a. In the terminal in which the broker is running, press **Ctrl+C** to stop the broker.

   b. Restart the broker.

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

c. In the web console, navigate back to the **helloworld** queue and verify that there are still 50 messages stored in the queue.

9. Consume the remaining 50 messages.

```
$ /opt/redhat/amq-broker/amq-broker-7.2.0/bin/artemis consumer --destination helloworld -- message-count 50 --url tcp://localhost:61616

Consumer:: filter = null
Consumer ActiveMQQueue[helloworld], thread=0 wait until 50 messages are consumedConsumer ActiveMQQueue[helloworld], thread=0 Consumed: 50 messages
Consumer ActiveMQQueue[helloworld], thread=0 Consumer thread finished
```

10. In the web console, verify that the **Message count** is 0.
All of the messages stored in the **helloworld** queue were consumed, and the queue is now empty.

STOPPING THE BROKER INSTANCE

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

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

Procedure

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

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

RUNNING THE AMQ BROKER EXAMPLES

AMQ Broker ships with many example programs that demonstrate basic and advanced features of the product. You can run these examples to become familiar with the capabilities of AMQ Broker.

To run the AMQ Broker examples, you must first set up your machine by installing and configuring Apache Maven and the AMQ Maven repository. Then, you use Maven to run the AMQ Broker example programs.

SETTING UP YOUR MACHINE TO RUN THE AMQ BROKER EXAMPLES

Before you can run the included AMQ Broker example programs, you must first download and install Maven and the AMQ Maven repository, and configure the Maven settings file.

Downloading and installing Maven

Maven is required to run the AMQ Broker examples.

Procedure

1. Go to the Apache Maven Download page and download the latest distribution for your operating system.
2. Install Maven for your operating system.

Downloading and installing the AMQ Maven repository

After Maven is installed on your machine, you download and install the AMQ Maven repository. This repository is available on the Red Hat Customer Portal.

3. In a web browser, navigate to https://access.redhat.com/downloads/ and log in. The Product Downloads page is displayed.
4. In the Integration and Automation section, click the Red Hat AMQ Broker link. The Software Downloads page is displayed.
5. Select the desired AMQ Broker version from the Version drop-down menu.

6. On the Releases tab, click the Download link for the AMQ Broker Maven Repository. The AMQ Maven repository file is downloaded as a zip file.

7. On your machine, unzip the AMQ repository file into a directory of your choosing.
   A new directory is created on your machine, which contains the Maven repository in a subdirectory named **maven-repository/**.

Configuring the Maven settings file

After downloading and installing the AMQ Maven repository, you must add the repository to the Maven settings file.

Procedure

8. Open the Maven **settings.xml** file.
   The **settings.xml** file is typically located in the **${user.home}/.m2/** directory.

   - For Linux, this is **~/.m2/**
   - For Windows, this is **\Documents and Settings\.m2** or **\Users\.m2**

   If you do not find a **settings.xml** file in **${user.home}/.m2/**, there is a default version located in the **conf/** directory of your Maven installation. Copy the default **settings.xml** file into the **${user.home}/.m2/** directory.

   In the **<profiles>** element, add a profile for the AMQ Maven repository.

```
<!-- Configure the JBoss AMQ Maven repository -->
<profile>
 <id>jboss-amq-maven-repository</id>
 <repositories>
  <repository>
   <id>jboss-amq-maven-repository</id>
   <url>file://<JBoss-AMQ-repository-path></url>
   <releases>
    <enabled>true</enabled>
   </releases>
   <snapshots>
    <enabled>false</enabled>
```

```
        </snapshots>
      </repository>
    </repositories>
    <pluginRepositories
                >
     <pluginRepository>
          <id>jboss-amq-maven-repository</id>
          <url>file://<JBoss-AMQ-repository-path></url>
          <releases>
           <enabled>true</enabled>
          </releases>
          <snapshots>
           <enabled>false</enabled>
          </snapshots>
     </pluginRepository>
    </pluginRepositories
                >
    </profile>
```

Replace **<JBoss-AMQ-repository-path>** with the location of the Maven repository that you installed. Typically, this location ends with **/maven-repository**. For example:

```
<url>file:///path/to/repo/amq-broker-7.2.0-maven-
        repository/maven-repository</url>
```

In the **<activeProfiles>** element, set the AMQ Maven repository to be active:

```
<activeProfiles>
 <activeProfile>jboss-amq-maven-repository</activeProfile>
...
</activeProfiles>
```

9. If you copied the default **settings.xml** from your Maven installation, uncomment the **<active- profiles>** section if it was commented out by default.

10. Save and close **settings.xml**.

11. Remove the cached **${user.home}/.m2/repository/** directory.
    If your Maven repository contains outdated artifacts, you may encounter one of the following Maven error messages when you build or deploy your project:

- **Missing artifact <artifact-name>**
- **[ERROR] Failed to execute goal on project <project-name>; Could not resolve dependencies for <project-name>**

AMQ BROKER EXAMPLE PROGRAMS

The AMQ Broker examples demonstrate how to use AMQ Broker features and the supported messaging protocols.

To find the example programs, see AMQ Broker example programs . The examples include the following:

- Features
  Broker-specific features such as:
  - Clustered - examples showing load balancing and distribution capabilities
  - HA - examples showing failover and reconnection capabilities
  - Perf - examples allowing you to run a few performance tests on the server
  - Standard - examples demonstrating various broker features
  - Sub-modules - examples of integrated external modules
- Protocols
  Examples for each of the supported messaging protocols:
  - AMQP
  - MQTT
  - OpenWire
  - STOMP

RUNNING AN AMQ BROKER EXAMPLE PROGRAM

Example programs for AMQ Broker demonstrate basic and advanced features of the product. You use Maven to run these programs.

Prerequisites

- Your machine is set up to run the AMQ Broker examples.
- You downloaded the AMQ Broker example programs .

Procedure

1. Navigate to the directory of the example that you want to run. The following example assumed that you downloaded the examples to a directory called **amq-broker-examples**.

```
$ cd amq-broker-examples/examples/features/standard/queue
```

2. Use the **mvn clean verify** command to run the example program. Maven starts the broker and runs the example program. The first time you run the example program, Maven downloads any missing dependencies, which may take a while to run.

   In this case, the **queue** example program is run, which creates a producer, sends a test message, and then creates a consumer that receives the message:

```
$ mvn clean verify
[INFO] Scanning for
projects...[INFO]
[INFO] ------------< org.apache.activemq.examples.broker:queue >---------
-----
[INFO] Building ActiveMQ Artemis JMS Queue Example
2.6.1.amq-720004-redhat-1[INFO] [ jar ]
...
server-out:2018-12-05 16:37:57,023 INFO
[org.apache.activemq.artemis.core.server] AMQ221001: Apache
ActiveMQ Artemis Message Broker version 2.6.1.amq-720004-
redhat- 1 [0.0.0.0, nodeID=06f529d3-f8d6-11e8-9bea-
0800271b03bd]
[INFO] Server
started[INFO]
```

[INFO] --- artemis-maven-plugin:2.6.1.amq-720004-redhat-1:runClient (runClient) @ queue --
-
Sent message: This is a text
message Received message: This
is a text message[INFO]
[INFO] --- artemis-maven-plugin:2.6.1.amq-720004-redhat-1:cli (stop) @ queue ---
server-out:2018-12-05 16:37:59,519 INFO [org.apache.activemq.artemis.core.server] AMQ221002: Apache ActiveMQ Artemis Message Broker version 2.6.1.amq-720004-redhat- 1 [06f529d3-f8d6-11e8-9bea-0800271b03bd] stopped, uptime 3.734 seconds
server-out:Server stopped!
[INFO] ----------
[INFO] BUILD SUCCESS
[INFO] ----------
[INFO] Total time: 48.681 s
[INFO] Finished at: 2018 12-05T16:37:59-05:00
[INFO] ----------

NOTE

Some of the example programs use UDP clustering, and may not work in yourenvironment by default. To run these examples successfully, redirect traffic directed to 224.0.0.0 to the loopback interface:

```
$ sudo route add -net 224.0.0.0 netmask 240.0.0.0 dev lo
```