# Red Hat AMQ 2020.Q4

## Migrating to Red Hat AMQ 7

For Use with Red Hat AMQ 7.8

Last Updated: 2020-12-03

# Red Hat AMQ 2020.Q4 Migrating to Red Hat AMQ 7

For Use with Red Hat AMQ 7.8

## Legal Notice



## Abstract

This guide describes the important changes that require your attention when transitioning from AMQ 6 to AMQ 7.

# Table of Contents

CHAPTER 1. INTRODUCTION ..... 4
1.1. WHEN TO GET ASSISTANCE BEFORE MIGRATING 4
1.2. SUPPORTED MIGRATION PATHS 4
1.3. UNDERSTANDING THE IMPORTANT NEW CONCEPTS IN AMQ 7 4
1.3.1. Architectural Changes in AMQ 7 4
Transport Connector Changes for Incoming Connections 4
Message Store and Paging Changes 4
Broker Deployment Changes 5
1.3.2. Message Address Changes in AMQ 7 5
1.4. REVIEWING NEW FEATURES AND KNOWN ISSUES IN AMQ 7 5
1.5. DOCUMENT CONVENTIONS 5
The sudo command 6
About the use of file paths in this document 6

CHAPTER 2. PREPARING FOR THE MIGRATION ..... 7
2.1. MIGRATION REQUIREMENTS 7
2.2. CREATING A BROKER INSTANCE 7
2.3. UNDERSTANDING THE BROKER INSTANCE DIRECTORY STRUCTURE 8
2.4. HOW BROKERS ARE CONFIGURED IN AMQ 7 9
2.5. VERIFYING THAT CLIENTS CAN CONNECT TO THE BROKER INSTANCE 10

CHAPTER 3. ACCEPTING INCOMING CONNECTIONS ..... 12
3.1. INCOMING NETWORK CONNECTIONS CHANGES 12
3.2. HOW ACCEPTORS ARE CONFIGURED 12

CHAPTER 4. USER AUTHENTICATION ..... 14
4.1. USER AUTHENTICATION CHANGES 14
4.2. HOW USER AUTHENTICATION IS CONFIGURED 14

CHAPTER 5. MESSAGE ADDRESSES AND QUEUES ..... 16
5.1. ADDRESSING CHANGES 16
5.2. HOW ADDRESSING IS CONFIGURED 17

CHAPTER 6. SECURITY ..... 19
6.1. HOW TRANSPORT LAYER SECURITY IS CONFIGURED 19
6.2. AUTHORIZATION 20
6.2.1. Authorization Changes 20
6.2.2. How Authorization is Configured 21

CHAPTER 7. RESOURCE LIMITS AND POLICIES ..... 23
7.1. HOW RESOURCE LIMITS AND POLICIES ARE CONFIGURED 23
7.2. RESOURCE LIMIT AND POLICY CONFIGURATION PROPERTIES 24
7.2.1. Queue Management Configuration Properties 24
7.2.2. Producer Policy Configuration Properties 25
7.2.3. Consumer Policy Configuration Properties 26
7.2.4. Slow Consumer Handling Configuration Properties 28
7.2.5. Message Paging Configuration Properties 29
7.2.6. Dead Letter Policy Configuration Properties 29
Dead Letter Policies in AMQ 6 29
Dead Letter Policies in AMQ 7 30

CHAPTER 8. MESSAGE PERSISTENCE AND PAGING ..... 32
8.1. MESSAGE PERSISTENCE CHANGES 32

8.2. HOW MESSAGE PERSISTENCE IS CONFIGURED 32
8.3. MESSAGE PERSISTENCE CONFIGURATION PROPERTY CHANGES 33
8.3.1. Journal Size and Management 33
8.3.2. Write Boundaries 35
8.3.3. Index Configuration 36
8.3.4. Journal Archival 37
8.3.5. Journal Recovery 37

**CHAPTER 9. BROKER CLUSTERS 38**
9.1. BROKER CLUSTERING CHANGES 38
9.2. HOW BROKER CLUSTERS ARE CONFIGURED 38
9.2.1. Creating a Broker Cluster 38
9.2.2. Additional Broker Cluster Topologies 40
9.3. BROKER CLUSTER CONFIGURATION PROPERTIES 42

**CHAPTER 10. HIGH AVAILABILITY AND FAILOVER 45**
10.1. HIGH AVAILABILITY AND FAILOVER CHANGES 45
10.2. HOW HIGH AVAILABILITY IS CONFIGURED 46

# CHAPTER 1. INTRODUCTION

This guide describes the new features and changes to behavior in AMQ 7. If you have an existing AMQ 6 environment, this guide will help you to understand the differences in AMQ 7 so that you are prepared to configure new broker instances in AMQ 7.

## 1.1. WHEN TO GET ASSISTANCE BEFORE MIGRATING

If you plan to migrate a production environment, you should seek further assistance and guidance from a Red Hat support representative. You can open a support case at https://access.redhat.com/support/.

## 1.2. SUPPORTED MIGRATION PATHS

You can use this guide to understand the configuration changes that might be required to create a AMQ Broker 7 configuration to which existing OpenWire JMS clients can connect.

This guide does not describe how to migrate the following features:

- The message store
  This guide provides information about configuration changes that will help you to configure a new AMQ 7 broker instance. Data, such as messages stored on the AMQ 6 broker, will not be migrated.

- Clients (other than OpenWire JMS clients)
  This guide helps you to configure a AMQ 7 broker instance to which existing OpenWire JMS clients can connect. For information about creating new clients that can connect to a AMQ 7 broker, see the client guides at the Red Hat Customer Portal .

## 1.3. UNDERSTANDING THE IMPORTANT NEW CONCEPTS IN AMQ 7

Before learning about the specific configuration changes in each AMQ feature area, you should first understand the important conceptual differences between AMQ 6 and AMQ 7.

There are several key architectural differences in AMQ 7. In addition, a new message addressing and routing model has been implemented in this release.

### 1.3.1. Architectural Changes in AMQ 7

AMQ 7 offers key architectural changes for how incoming network connections are made to the broker, the message store, and the way in which brokers are deployed.

**Transport Connector Changes for Incoming Connections**
AMQ 6 used different types of transport connectors, such as TCP (synchronous) and Java NIO (non-blocking).

In AMQ 7, you no longer have to choose which transport type to use: all incoming network connections between entities in different virtual machines use Netty connections. Netty is a high-performance, low-level network library that allows network connections to be configured to use Java IO, Java NIO, TCP sockets, SSL/TLS, HTTP, and HTTPS.

**Message Store and Paging Changes**
The process by which the broker stores messages in memory and pages them to disk is different in AMQ 7.

AMQ 6 used KahaDB for a message store, which consists of both a message journal for fast, sequential message storing, and an index to retrieve messages when needed.

AMQ 7 contains its own built-in message store, which consists of an append-only message journal. It does not use an index.

For more information about these changes, see Message Persistence.

**Broker Deployment Changes**
In AMQ Broker 7, broker deployment differs from AMQ 6 in the following ways:

- Deployment mechanism
  AMQ 6, by default, was deployed in Apache Karaf containers. AMQ Broker 7 is not.

- Deploying multiple brokers
  In AMQ 6, to deploy multiple brokers, you either had to deploy a collection of standalone brokers (which required you to install and configure each broker separately), or deploy a fabric of AMQ brokers using JBoss Fuse Fabric.

  In AMQ Broker 7, deploying multiple brokers involves installing AMQ Broker 7 once, and then on the same machine, creating as many *broker instances* as you require. AMQ Broker 7 is not intended to be deployed using fabrics.

### 1.3.2. Message Address Changes in AMQ 7

AMQ 7 introduces a new addressing and routing model to configure message routing semantics for any messaging protocol (or API in the case of JMS). However, this model does require you to configure address, queue, topic, and routing functionality differently than in AMQ 6. As part of your migration planning, you should be prepared to carefully review the new addressing model and its configuration elements.

AMQ Broker 7 does not distinguish between JMS and non-JMS configuration. AMQ Broker 7 implements addresses, routing mechanisms, and queues. Messages are delivered by routing messages to queues based on addresses and routing mechanisms.

Two new routing mechanisms—multicast and anycast—enable AMQ Broker 7 to route messages in standard messaging patterns. Multicast routing implements a publish-subscribe pattern in which all subscribers to an address receive messages sent to the address. Alternatively, anycast routing implements a point-to-point pattern in which only a single queue is attached to an address, and consumers subscribe to that queue to receive messages in round-robin order.

**Related Information**

- For more information about the new addressing model in AMQ Broker 7, see Configuring addresses and queues in *Configuring AMQ Broker*.

- For more information about how message addressing is configured in AMQ Broker 7, see Message Addresses and Queues.

## 1.4. REVIEWING NEW FEATURES AND KNOWN ISSUES IN AMQ 7

Before migrating to AMQ 7, you should understand the key new features, enhancements, and known issues. For a list, see the Release Notes for Red Hat AMQ Broker 7.8 .

## 1.5. DOCUMENT CONVENTIONS

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

### The sudo command

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

For more information about using **sudo**, see The **sudo** Command.

### About the use of file paths in this document

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

# CHAPTER 2. PREPARING FOR THE MIGRATION

Before learning about the configuration changes in each feature area, you should ensure that your environment meets the migration requirements and understand how broker instances are configured in AMQ Broker 7.

## 2.1. MIGRATION REQUIREMENTS

Before migrating to AMQ 7, your environment should meet the following requirements:

**AMQ 6 requirements**

- You should be running AMQ 6.2.x or later.
- OpenWire clients should use OpenWire version 10 or later.

**AMQ 7 requirements**

- You should have a supported operating system and JVM.
  You can view supported configurations for AMQ 7 at:
  https://access.redhat.com/articles/2791941
- AMQ Broker 7 should be installed.
  For more information, see Installing AMQ Broker in *Getting Started with AMQ Broker*.

## 2.2. CREATING A BROKER INSTANCE

Before migrating to AMQ 7, you should create a AMQ broker instance. You can configure this broker instance as you learn about the configuration differences in AMQ 7 that are described in this guide.

When you installed AMQ Broker, the binaries, libraries, and other important files needed to run AMQ Broker were installed. However, in AMQ 7, you must explicitly create a broker instance whenever a new broker is needed. Each broker instance is a separate directory containing its own configuration and runtime data.

**NOTE**

Keeping broker installation and configuration separate means that you can install AMQ Broker just once in a central location and then create as many broker instances as you require. Additionally, keeping installation and configuration separate makes it easier to manage and upgrade your brokers as needed.

**Prerequisites**

- AMQ Broker 7 must be installed.

**Procedure**

1. Navigate to the location where you want to create the broker instance.

```
$ sudo mkdir /var/lib/amq7
$ cd /var/lib/amq7
```

2. Do one of the following to create the broker instance:

| If... | Then... |
| --- | --- |
| AMQ Broker 7 is installed on the same machine as AMQ 6 | Use the **artemis create** command with the **--port-offset** parameter to create the new broker instance that will not conflict with your existing AMQ 6 broker.<br><br>  <br>**NOTE**<br>AMQ Broker 7 and AMQ 6 both listen for client traffic on the same set of default ports. Therefore, you must offset the default ports on the AMQ Broker broker instance to avoid potential conflicts.<br><br>This example creates a new broker instance that listens for client traffic on different ports than the AMQ 6 broker:<br><br>$ sudo *INSTALL_DIR*/bin/artemis create mybroker --port-offset 100 --user admin --password pass --role amq --allow-anonymous true |
| AMQ Broker 7 and AMQ 6 are installed on separate machines | Use the **artemis create** command to create the new broker instance.<br><br>This example creates a new broker instance and prompts you for any required values:<br><br>$ sudo *INSTALL_DIR*/bin/artemis create mybroker<br><br>Creating ActiveMQ Artemis instance at: /var/lib/amq7/mybroker<br><br>--user: is mandatory with this configuration:<br>Please provide the default username:<br>user<br><br>--password: is mandatory with this configuration:<br>Please provide the default password:<br>password<br><br>--role: is mandatory with this configuration:<br>Please provide the default role:<br>amq<br><br>--allow-anonymous |

### Related Information

For full details on creating broker instances, see Creating a broker instance in *Getting Started with AMQ Broker*.

## 2.3. UNDERSTANDING THE BROKER INSTANCE DIRECTORY STRUCTURE

Each AMQ 7 broker instance contains its own directory. You should understand the directory content and where to find the configuration files for the broker instance you created.

When you create a broker instance, the following directory structure is created:

```
$ ls /var/lib/amq7/mybroker
bin  data  etc  lock  log  tmp
```

***BROKER_INSTANCE_DIR***

The location where the broker instance was created. This is a different location than the AMQ Broker installation.

**/bin**

Shell scripts for starting and stopping the broker instance.

**/data**

Contains broker state data, such as the message store.

**/etc**

The broker instance's configuration files. These are the files you need to access to configure the broker instance.

**/lock**

Contains the **cli.lock** file.

**/log**

Log files for the broker instance.

**/tmp**

A utility directory for temporary files.

## 2.4. HOW BROKERS ARE CONFIGURED IN AMQ 7

You should understand how the broker instance you created should be configured and which configuration files you will need to edit.

Like AMQ 6, you configure AMQ 7 broker instances by editing plain text and XML files. Changing a broker's configuration involves opening the appropriate configuration file in the broker instance's directory, locating the proper element in the XML hierarchy, and then making the actual change—which typically involves adding or removing XML elements and attributes.

Within ***BROKER_INSTANCE_DIR*/etc**, there are several configuration files that you can edit:

| Configuration File | Description |
|---|---|
| **broker.xml** | The main configuration file. Similar to **activemq.xml** in AMQ 6, you use this file to configure most aspects of the broker, such as acceptors for incoming network connections, security settings, message addresses, and so on. |
| **bootstrap.xml** | The file that AMQ Broker uses to start the broker instance. You use it to change the location of the main broker configuration file, configure the web server, and set some security settings. |

| Configuration File | Description |
|---|---|
| **logging.properties** | You use this file to set logging properties for the broker instance. This file is similar to the **org.ops4j.pax.logging.cfg** file in AMQ 6. |
| JAAS configuration files (**login.config**, **users.properties**, **roles.properties**) | You use these files to set up authentication for user access to the broker instance. |

Migrating to AMQ 7 primarily involves editing the **broker.xml** file. For more information about the **broker.xml** structure and default configuration settings, see Understanding the default broker configuration in *Configuring AMQ Broker*.

## 2.5. VERIFYING THAT CLIENTS CAN CONNECT TO THE BROKER INSTANCE

To verify that your existing clients can connect to the broker instance you created, you should start the broker instance and send some test messages.

### Procedure

1. Start the broker instance by using one of the following commands:

| To... | Use this command... |
|---|---|
| Start the broker in the foreground | `$ sudo BROKER_INSTANCE_DIR/bin/artemis run` |
| Start the broker as a service | `$ sudo BROKER_INSTANCE_DIR/bin/artemis-service start` |

   The broker instance starts. By default, an OpenWire connector is started on the broker instance on the same port as your AMQ 6 broker. This should enable your existing clients to connect to the broker instance.

2. If you want to check the status of the broker instance, open the ***BROKER_INSTANCE_DIR*/logs/artemis.log** file.

3. In your AMQ 6 broker, use the **producer** command to send some test messages to the AMQ 7 broker instance.
   This command sends five test messages to a AMQ 7 broker instance hosted on localhost and listening on the default acceptor:

```
JBossA-MQ:karaf@root> producer --brokerUrl tcp://0.0.0.0:61616 --message "Test message" --messageCount 5
```

If you offset the port numbers when you created the broker instance (using **--port-offset**), make sure that you use the correct port number for the broker URL. For example, if you set the port offset to 100, then you would need to set **--brokerUrl** to **tcp://0.0.0.0:61716**.

4. In your AMQ 6 broker, use the **consumer** command to verify that you can consume the test messages that you sent to the AMQ 7 broker instance.
This command receives the five test messages sent to the AMQ 7 broker instance:

```
JBossA-MQ:karaf@root> consumer --brokerUrl tcp://0.0.0.0:61616
```

You can also verify that the messages were sent and received by checking the ***INSTALL_DIR*/data/log/amq.log** file on the AMQ 6 broker.

5. Stop the broker instance:

```
$ BROKER_INSTANCE_DIR/bin/artemis stop
```

# CHAPTER 3. ACCEPTING INCOMING CONNECTIONS

Network connections define how clients connect to your broker instance. In AMQ 7, these connections function differently and are configured differently than in AMQ 6.

## 3.1. INCOMING NETWORK CONNECTIONS CHANGES

AMQ 6 and AMQ Broker 7 both enable you to define the way that clients connect to the broker. These connection points were called *transport connectors* in AMQ 6, but now are called *acceptors* in AMQ Broker 7.

AMQ 6 provided multiple implementations of the transport layer (such as TCP and NIO), which meant that you had to use different transport connectors depending on whether you wanted a client connection point to use a blocking or non-blocking transport. In AMQ Broker 7, the transport layer uses Netty only, which is non-blocking by default. There are two types of acceptors in AMQ Broker 7:

**TCP**

Netty TCP connections are used when the client and broker are located in different virtual machines, whether on the same server or physically remote.
Netty uses non-blocking (Java NIO) by default, which means that all client connections to the broker instance are non-blocking. It also has built-in support for WebSockets.

**In-VM**

An In-VM connection is used when the client, whether an application or a server, resides within the same virtual machine as the broker.

AMQ 6 also required you to use separate transport connectors for each messaging protocol. In AMQ Broker 7, the low-level transport (either TCP or In-VM) is distinct from the messaging protocol used by the client (such as AMQP, MQTT, and so on). This means that a single acceptor can use multiple protocols on the same port. In fact, an acceptor will accept all supported message protocols unless you explicitly restrict the protocols that it can use.

For example, the default acceptor in AMQ Broker 7 automatically accepts all message protocols:

```
<acceptor name="artemis">tcp://0.0.0.0:61616?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=CORE,AMQP,STOMP,HORNE
TQ,MQTT,OPENWIRE;useEpoll=true;amqpCredits=1000;amqpLowCredits=300</acceptor>
```

## 3.2. HOW ACCEPTORS ARE CONFIGURED

You use the ***BROKER_INSTANCE_DIR*/etc/broker.xml** configuration file to configure acceptors to accept incoming client connections for your broker instance.

The **broker.xml** configuration file contains the following default acceptors in the **<acceptors>** section:

```
<configuration>
...
  <core>
  ...
     <acceptors>
       <!-- Acceptor for every supported protocol -->
       <acceptor name="artemis">tcp://0.0.0.0:61616?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=CORE,AMQP,STOMP,HORNE
```

```
TQ,MQTT,OPENWIRE;useEpoll=true;amqpCredits=1000;amqpLowCredits=300</acceptor> (1)

      <!-- AMQP Acceptor.  Listens on default AMQP port for AMQP traffic.-->
      <acceptor name="amqp">tcp://0.0.0.0:5672?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=AMQP;useEpoll=true;amqpCre
dits=1000;amqpMinCredits=300</acceptor>

      <!-- STOMP Acceptor. -->
      <acceptor name="stomp">tcp://0.0.0.0:61613?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=STOMP;useEpoll=true</acce
ptor>

      <!-- HornetQ Compatibility Acceptor.  Enables HornetQ Core and STOMP for legacy HornetQ
clients. -->
      <acceptor name="hornetq">tcp://0.0.0.0:5445?
protocols=HORNETQ,STOMP;useEpoll=true</acceptor>

      <!-- MQTT Acceptor -->
      <acceptor name="mqtt">tcp://0.0.0.0:1883?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=MQTT;useEpoll=true</accept
or>
      ...
   </core>
</configuration>
```

(1) The default acceptor, which accepts incoming client connections for any of the supported messaging protocols.

To configure the incoming client connections for your broker instance, you can modify the configuration properties for any of the default acceptors, or you can add new acceptors. This example shows a new acceptor configured to accept TCP connections using the OpenWire protocol:

```
<acceptor name="my-acceptor">tcp://0.0.0.0:61613?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=OPENWIRE;useEpoll=true</
acceptor>
```

**Related Information**

- For more information about the default acceptor configuration, see Default acceptor settings in *Configuring AMQ Broker*.
- For step-by-step details about configuring acceptors, see Network Connections: Acceptors and Connectors in *Configuring AMQ Broker*.
- For a description of every property you can use to configure an acceptor, see Acceptor and Connector Configuration Parameters in *Configuring AMQ Broker*.

# CHAPTER 4. USER AUTHENTICATION

User authentication enables you to verify the identity of users by adding usernames and assigning them to security roles. In AMQ Broker 7, this process is similar to AMQ 6. However, there are some differences in terminology, configuration file locations, and configuration syntax. Once you understand the differences, there are several methods you can use to configure user access to your broker instance.

## 4.1. USER AUTHENTICATION CHANGES

In both AMQ Broker 7 and AMQ 6, authentication is provided by pluggable login modules based on the Java Authentication and Authorization Service (JAAS). However, *groups* in AMQ 6 are now called *roles* in AMQ Broker 7.

In addition, the names and locations of the login modules have changed in AMQ Broker 7.

| Login Module | Location in AMQ 6 | Location in AMQ Broker 7 |
|---|---|---|
| Users | **etc/users.properties** | ***BROKER_INSTANCE_DIR*/etc/artemis-users.properties** |
| Roles (groups) | **etc/groups.properties** | ***BROKER_INSTANCE_DIR*/etc/artemis-roles.properties** |

The syntax for adding users and roles is also different.

**In AMQ 6**

Non-privileged users could be added and assigned a password and security role in the **users.properties** file:

```
USER=PASSWORD,ROLE
```

**In AMQ Broker 7**

Users and roles are assigned in separate login modules. You add users in the **artemis-users.properties** file:

```
USER=PASSWORD
```

You assign users to a security role in the **artemis-roles.properties** file:

```
ROLE=USER
```

## 4.2. HOW USER AUTHENTICATION IS CONFIGURED

You can access the AMQ 7 broker instance using the default username and password that you created when you created the broker instance. To enable additional users to access the broker instance, you can configure user authentication for the broker using any of the following methods:

| 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 access the broker instance using these credentials.<br><br>For more information, see Adding Users in *Configuring AMQ Broker*. |
| Certificate-Based Authentication | Users are authenticated using SSL certificates.<br><br>For more information, see Adding Certificate-based Authentication in *Configuring AMQ Broker*. |
| LDAP Authentication | Users are authenticated and authorized by checking the credentials against user data stored in a central X.500 directory server.<br><br>For more information, see Using LDAP for Authentication in *Configuring AMQ Broker*. |

# CHAPTER 5. MESSAGE ADDRESSES AND QUEUES

AMQ 7 introduces a new, flexible addressing model that enables you to define standard messaging patterns that work for any messaging protocol. Therefore, the process for configuring queues and topic-like behavior has changed significantly.

## 5.1. ADDRESSING CHANGES

AMQ 6 implemented JMS concepts such as queues, topics, and durable subscriptions as directly-configurable destinations.

### Example: Default Queue and Topic Configuration in AMQ 6

```
<destinations>
    <queue physicalName="my-queue" />
    <topic physicalName="my-topic" />
</destinations>
```

AMQ Broker 7 uses addresses, routing types, and queues to achieve queue and topic-like behavior. An *address* represents a messaging endpoint. *Queues* are associated with addresses. A *routing type* defines how messages are distributed to the queues associated with an address. There are two routing types: *Anycast* distributes messages to a single queue within the matching address, and *Multicast* distributes messages to every queue associated with the address.

By associating queues with addresses and routing types, you can implement a variety of messaging patterns, such as point-to-point (queues) and publish-subscribe (topic-like).

### Example: Point-to-Point Address Configuration in AMQ Broker 7

In this example, when the broker receives a message on **address.foo**, the message will be routed to **my-queue**. If multiple anycast queues are associated with the address, the messages are distributed evenly across the queues.

```
<address name="address.foo">
  <anycast>
    <queue name="my-queue"/>
  </anycast>
</address>
```

### Example: Publish-Subscribe Address Configuration in AMQ Broker 7

In this example, when the broker receives a message on **topic.foo**, a copy of the message will be routed to both **my-topic-1** and **my-topic-2**.

```
<address name="topic.foo">
  <multicast>
    <queue name="my-topic-1"/>
    <queue name="my-topic-2"/>
  </multicast>
</address>
```

### Related Information

- For full details about the addressing model in AMQ Broker 7, see Configuring addresses and queues in *Configuring AMQ Broker*.

## 5.2. HOW ADDRESSING IS CONFIGURED

You use the ***BROKER_INSTANCE_DIR*/etc/broker.xml** configuration file to configure addresses and queues for your broker instance.

The **broker.xml** configuration file contains the following default addressing configuration in the **<addresses>** section. There are default entries for the Dead Letter Queue ( **DLQ**) and Expiry Queue (**ExpiryQueue**):

```
<addresses>
  <address name="DLQ">
    <anycast>
      <queue name="DLQ" />
    </anycast>
  </address>
    <address name="ExpiryQueue">
      <anycast>
        <queue name="ExpiryQueue" />
      </anycast>
  </address>
</addresses>
```

You can configure addressing for your broker instance by using any of the following methods:

| Method | Description |
| --- | --- |
| Manually configure an address | You define the routing types and queues that the broker should use when receiving a message on the address. You can configure an address in the following ways:<br>• Configuring an Address for Point-to-Point Messaging in *Configuring AMQ Broker*<br>• Configuring a Point-to-Point Address with Two Queues in *Configuring AMQ Broker*<br>• Configuring an Address for Publish-Subscribe Messaging in *Configuring AMQ Broker*<br>• Configuring an Address to Use Point-to-Point and Publish-Subscribe in *Configuring AMQ Broker*<br>• Configuring Subscription Queues in *Configuring AMQ Broker* |

| Method | Description |
| --- | --- |
| Configure the broker to create addresses automatically | You specify an address prefix and routing type for which addresses you want to be created automatically. When the broker receives a message on an address that matches the prefix, the address and routing type will be created automatically. You can also specify that the address be deleted automatically when all of its queues have been deleted, and that its queues be deleted automatically when they have no consumers or messages.<br><br>For more information, see Configuring automatic creation and deletion of addresses and queues in *Configuring AMQ Broker*. |

# CHAPTER 6. SECURITY

AMQ Broker 7 provides transport layer security to secure incoming network connections, and authorization to secure access to queues based on their respective addresses. In both of these areas, the security model is very similar to AMQ 6. However, the configuration processes are different.

## 6.1. HOW TRANSPORT LAYER SECURITY IS CONFIGURED

Like AMQ 6, AMQ Broker 7 enables you to secure incoming network connections using SSL/TLS. However, there are some differences in configuration syntax and configuration properties.

In AMQ 6, transport layer security was configured by creating an SSL context to define the keystores and truststores, and then adding SSL attributes to each transport connector that you wanted to secure.

In AMQ Broker 7, the transport layer is based on Netty, which uses SSL natively. This means that to configure transport layer security, you just add the necessary SSL attributes to each acceptor that you want to secure. You do not need to add a separate SSL context.

For example, the following configuration accepts secure connections from an OpenWire client:

**In AMQ 6**

1. Define the SSL context in the ***INSTALL_DIR*/etc/activemq.xml** file:

```
<sslContext>
   <sslContext keyStore="file:${activemq.conf}/broker.ks" keyStorePassword="password"/>
</sslContext>
```

2. In the broker configuration file, create a transport connector to accept secure connections from the OpenWire client:

```
<transportConnector name="ssl" uri="ssl://localhost:61617?transport.needClientAuth=true"/>
```

**In AMQ Broker 7**

- In the ***BROKER_INSTANCE_DIR*/etc/broker.xml** configuration file, create or update an acceptor to accept secure connections from the OpenWire client:

```
<acceptor name="netty-ssl-acceptor">tcp://localhost:61617?
sslEnabled=true;keyStorePath=${data.dir}/../etc/broker.ks;keyStorePassword=password;needCl
ientAuth=true</acceptor>
```

You can configure either one-way or two-way TLS. The following table describes these methods:

| Method | Description |
|---|---|
| One-way TLS | Only the broker presents a certificate. This method requires you to have a Java KeyStore for the server-side certificates.<br><br>For more information, see Securing Network Connections in *Configuring AMQ Broker*. |

| Method | Description |
| --- | --- |
| Two-way TLS (mutual authentication) | Both the broker and the client present certificates. This method requires you to have a Java KeyStore for the server-side certificates, and a TrustStore that holds the keys of the clients that the broker trusts.<br><br>For more information, see Securing Network Connections in *Configuring AMQ Broker*. |

**NOTE**

To reuse your existing keystores and truststores for AMQ Broker 7, copy them to your AMQ Broker 7 broker instance.

**Related Information**

- For a full list of all transport layer security configuration properties, see Netty TLS Parameters in *Configuring AMQ Broker*.

## 6.2. AUTHORIZATION

AMQ Broker 7 provides a role-based security model in which you apply security settings to queues based on their addresses. This security model is similar to AMQ 6; however, the permissions and wildcard syntax are different, and authorization is configured differently.

### 6.2.1. Authorization Changes

AMQ Broker 7 uses a different set of permissions and a slightly different wildcard syntax than AMQ 6.

The following table describes the different types of permissions that you can apply in AMQ 6 and AMQ Broker 7:

| Permission in AMQ 6 | Corresponding Permissions in AMQ Broker 7 |
| --- | --- |
| **write** | **send** |
| **read** | **consume**<br><br>**browse** |

| Permission in AMQ 6 | Corresponding Permissions in AMQ Broker 7 |
|---|---|
| **admin** | **createAddress**<br>**deleteAddress**<br>**createNonDurableQueue**<br>**deleteNonDurableQueue**<br>**createDurableQueue**<br>**deleteDurableQueue**<br>**manage** |

For more information about permissions in AMQ Broker 7, see Setting Permissions in *Configuring AMQ Broker*.

The wildcard syntax for matching addresses is also different in AMQ Broker 7.

| To... | In AMQ 6 | In AMQ Broker 7 |
|---|---|---|
| Separate words in the path | **.** | **.** |
| Match a single word | ***** | ***** |
| Match any word recursively | **>** | **#** |

### 6.2.2. How Authorization is Configured

You use the ***BROKER_INSTANCE_DIR*/etc/broker.xml** configuration file to assign security settings to queues.

The **broker.xml** configuration file contains the following default security settings, which provide complete access to all addresses and queues for the default role that you created when you created the broker instance:

```
<configuration ...>
  <core ...>
    ...
    <security-settings>
      <security-setting match="#"> 1
        <permission type="createNonDurableQueue" roles="admin"/> 2
        <permission type="deleteNonDurableQueue" roles="admin"/>
        <permission type="createDurableQueue" roles="admin"/>
        <permission type="deleteDurableQueue" roles="admin"/>
        <permission type="createAddress" roles="admin"/>
        <permission type="deleteAddress" roles="admin"/>
        <permission type="consume" roles="admin"/>
        <permission type="browse" roles="admin"/>
```

```
      <permission type="send" roles="admin"/>
      <permission type="manage" roles="admin"/>
    </security-setting>
  </security-settings>
  ...
 </core>
</configuration>
```

1. The address or address prefix to which a set of security permissions are applied. The permissions are applied to the set of queues that match the address. In this example, the **#** wildcard matches all addresses.

2. A permission granted to a role. In this example, all users belonging to the **admin** role are granted permission to create non-durable queues.

You can configure authorization for a queue or set of queues by specifying an address that matches the queues, and then specifying the roles that should be granted each permission type.

**Related Information**

- Setting Permissions in *Configuring AMQ Broker*

# CHAPTER 7. RESOURCE LIMITS AND POLICIES

You can define resource limits and policies to control important aspects of how the broker instance should handle messages. The process for configuring these resource limits and policies is different in AMQ Broker 7 than in AMQ 6, and many of the configuration properties have changed.

## 7.1. HOW RESOURCE LIMITS AND POLICIES ARE CONFIGURED

In AMQ 6, resource limits and policies were configured as destination policies in the broker's configuration file.

In AMQ Broker 7, you define resource limits and policies for an address or *set* of addresses. When the broker instance receives a message, the resource limits and policies defined for the message's address are applied to the message.

To configure resource limits and policies in AMQ Broker 7, you use the ***BROKER_INSTANCE_DIR*/etc/broker.xml** configuration file to define **<address-setting>** elements with the appropriate configuration properties.

The **broker.xml** configuration file contains the following default address settings configuration:

```
<address-settings>
   <!-- if you define auto-create on certain queues, management has to be auto-create -->
   <address-setting match="activemq.management#"> (1)
      <dead-letter-address>DLQ</dead-letter-address>
      <expiry-address>ExpiryQueue</expiry-address>
      <redelivery-delay>0</redelivery-delay>
      <!-- with -1 only the global-max-size is in use for limiting -->
      <max-size-bytes>-1</max-size-bytes>
      <message-counter-history-day-limit>10</message-counter-history-day-limit>
      <address-full-policy>PAGE</address-full-policy>
      <auto-create-queues>true</auto-create-queues>
      <auto-create-addresses>true</auto-create-addresses>
      <auto-create-jms-queues>true</auto-create-jms-queues>
      <auto-create-jms-topics>true</auto-create-jms-topics>
   </address-setting>
   <!--default for catch all-->
   <address-setting match="#"> (2)
      <dead-letter-address>DLQ</dead-letter-address>
      <expiry-address>ExpiryQueue</expiry-address>
      <redelivery-delay>0</redelivery-delay>
      <!-- with -1 only the global-max-size is in use for limiting -->
      <max-size-bytes>-1</max-size-bytes>
      <message-counter-history-day-limit>10</message-counter-history-day-limit>
      <address-full-policy>PAGE</address-full-policy>
      <auto-create-queues>true</auto-create-queues>
      <auto-create-addresses>true</auto-create-addresses>
      <auto-create-jms-queues>true</auto-create-jms-queues>
      <auto-create-jms-topics>true</auto-create-jms-topics>
   </address-setting>
</address-settings>
```

(1) The default management address setting. The nested resource limits and policies are applied to all messages with an address that matches **activemq.management#**.

2 The default address setting. The **#** wildcard matches all addresses, so the defined resource limits and policies are applied to all messages.

To configure resource limits and policies, you specify an address or *set* of addresses (using **<address-setting>**), and then add resource limit and policy properties to it. These properties are applied to each message sent to the address (or addresses) that you specified.

**Related Information**

- For more information on using wildcards to match sets of addresses, see {Broker wildcard syntax in *Configuring AMQ Broker*.

## 7.2. RESOURCE LIMIT AND POLICY CONFIGURATION PROPERTIES

Like AMQ 6, in AMQ Broker 7, you can add resource limits and policies to control how the broker handles certain aspects of how and when messages are delivered, the number of delivery attempts that should be made, and when messages should expire. However, the configuration properties you use to define these resource limits and policies are different in AMQ Broker 7.

This section compares the **<policyEntry>** configuration properties in AMQ 6 to the equivalent **<address-setting>** properties in AMQ Broker 7. For complete details on each configuration property in AMQ Broker 7, see Address Setting Configuration Elements in *Configuring AMQ Broker*.

### 7.2.1. Queue Management Configuration Properties

The following table compares the queue management configuration properties in AMQ 6 to the equivalent properties in AMQ Broker 7:

| To set... | In AMQ 6 | In AMQ Broker 7 |
|---|---|---|
| The memory limit | **memoryLimit**<br><br>Sets a memory limit for the *destination*. The default is **none**. | **<max-size-bytes>**<br><br>Sets the memory limit for the *address*. The default is **-1** (no limit). |
| The order of the messages by priority within the queue | **prioritizedMessages**<br><br>This is off by default, which means that messages are prioritized on the consumer (not the broker), and therefore are ordered based on the priorities of the messages on the consumer. | Messages are automatically ordered by priority within the queue. |
| How often the broker should scan for expired messages | **expiredMessagesPeriod** | **<message-expiry-scan-period>**<br><br>The default is 30000 ms. |

| To set... | In AMQ 6 | In AMQ Broker 7 |
|---|---|---|
| Whether the broker should delete destinations that are inactive for a period of time | **gcInactiveDestinations**<br><br>The default is **false**. | No equivalent. However, for automatically-created queues, you can set the queue to be automatically deleted when the last consumer is detached. For more information, see Configuring automatic creation and deletion of addresses and queues in *Configuring AMQ Broker*. |
| The inactive timeout | **inactiveTimeoutBeforeGC**<br><br>The default is 60 seconds. | No equivalent. However, for automatically-created queues, you can set the queue to be automatically deleted when the last consumer is detached. For more information, see Creating and Deleting Queues and Addresses Automatically in *Configuring AMQ Broker*. |
| Whether the broker should use a separate thread when dispatching from a queue | **optimizedDispatch**<br><br>The default is **false**. | This cannot be set for an address or queue. However, you can control it from the incoming connection on which the message arrives. Use the **directDeliver** property on an acceptor or connector to control whether the message should be delivered on the same thread on which it arrived. For more information, see Acceptor and Connector Configuration Parameters in *Configuring AMQ Broker*. |

### 7.2.2. Producer Policy Configuration Properties

The following table compares the producer policy configuration properties in AMQ 6 to the equivalent properties in AMQ Broker 7:

| To set... | In AMQ 6 | In AMQ Broker 7 |
|---|---|---|

| To set... | In AMQ 6 | In AMQ Broker 7 |
|---|---|---|
| Producer flow control | **producerFlowControl**<br><br>Sets the broker to throttle the producer. The throttling is achieved by either withholding the producer's acknowledgement, or by raising a **javax.jms.ResourceAllocationException** exception and propagating it back to the client when local resources have been exhausted (such as memory or storage). The default is **true**. | For the address, set **<max-size-bytes>** to the size at which the producer should be throttled, and then set **<address-full-policy>** to **BLOCK**.<br><br>Configuring these two properties will also throttle your existing AMQ 6 OpenWire producers. |
| The amount of credits a producer can request at one time | No equivalent. | **<producer-window-size>**<br><br>Limiting the window size sets a limit on the number of bytes that the producer can have "in-flight" at any one time, which can prevent the remote connection from becoming overloaded. |

### 7.2.3. Consumer Policy Configuration Properties

The following table compares the server-side destination policy configuration properties in AMQ 6 to the equivalent properties in AMQ Broker 7. These properties only apply to OpenWire clients:

| To set... | In AMQ 6 | In AMQ Broker 7 |
|---|---|---|
| The queue prefetch | **queuePrefetch** | No equivalent on the broker. However, you can set the maximum size of messages (in bytes) that will be buffered on a consumer by setting the **consumerWindowSize** on the connection URL or directly on the **ActiveMQConnectionFactory** API. |
| Whether to use the priority of a consumer when dispatching messages from a queue | **useConsumerPriority**<br><br>The default is **true**. | This functionality does not exist in AMQ Broker 7. |
| Whether to use the prefetch extension to enable the broker to dispatch "prefetched" messages when the previous message is delivered but not acknowledged | **usePrefetchExtension**<br><br>The default is **true**. | This functionality does not exist in AMQ Broker 7. |

| To set... | In AMQ 6 | In AMQ Broker 7 |
|---|---|---|
| Initial redelivery delay | **initialRedeliveryDelay**<br><br>The default is 1000 ms. | No equivalent. The broker instance automatically handles this. |
| How long to wait before attempting to redeliver a canceled message | **redeliveryDelay**<br><br>The delivery delay if **initialRedeliveryDelay** is set to **0**. The default is 1000 ms. | **<redelivery-delay>**<br><br>The default is 0 ms. |
| Exponential back-off | **useExponentialBackoff**<br><br>The default is **false**. | No equivalent. You can use any of the other consumer policy configuration properties to configure redelivery for a consumer. |
| Backoff multiplier | **backOffMultiplier**<br><br>The default is 5. | **<redelivery-multiplier>**<br><br>The multiplier to apply to the redelivery delay. The default is 1.0. |
| The maximum number of times a cancelled message can be redelivered before it is returned to the broker's Dead Letter Queue | **maximumRedeliveries**<br><br>The default is 6. | **<max-delivery-attempts>**<br><br>The default is 10. |
| The maximum value for the redelivery delay | **maximumRedeliveryDelay**<br><br>This is only applied if the **useExponentialBackoff** property is set. The default is -1 (no maximum redelivery delay). | **<max-redelivery-delay>**<br><br>The default is 0. |
| The number of messages that a client can consume in a second | No equivalent. | No equivalent on the broker. However, you can set this on a consumer by setting the **consumerMaxRate** on the connection URL or directly on the **ActiveMQConnectionFactory** API.<br><br>The **consumerMaxRate** property does not affect the number of messages that a client has in its buffer. Therefore, if the client has a slow rate limit and a high window size, the client's internal buffer would quickly fill up with messages. |

### 7.2.4. Slow Consumer Handling Configuration Properties

Like AMQ 6, AMQ Broker 7 can detect slow consumers and automatically stop the ones that are consistently slow. This was enabled by default in AMQ 6, but is disabled by default in AMQ Broker 7.

The way in which the broker determines that a consumer is "slow" is also different. In AMQ Broker 7, a consumer is considered to be slow based on the number of messages the consumer has acknowledged. In AMQ 6, a consumer was considered to be slow based on the fullness of the prefetch buffer (if the buffer is consistently full, then the client may be consuming messages too slowly).

The following table compares the slow consumer handling configuration properties in AMQ 6 to the equivalent properties in AMQ Broker 7:

| To set... | In AMQ 6 | In AMQ Broker 7 |
|---|---|---|
| The number of times a consumer can be considered to be slow before it is aborted | **maxSlowCount**<br><br>The default is -1 (no limit). | No equivalent. You can use the other slow consumer handling properties to control slow consumers. |
| The amount of time a consumer can be continuously slow before it is aborted | **maxSlowDuration**<br><br>The default is 30000 ms. | **<slow-consumer-threshold>**<br><br>In AMQ Broker 7, this is the minimum rate of message consumption before a consumer is considered to be "slow" (measured in messages per second). The default is -1 (no threshold). |
| The amount of time the broker should wait before performing another check for slow consumers | **checkPeriod**<br><br>The default is 30000 ms. | **<slow-consumer-check-period>**<br><br>In AMQ Broker 7, this is measured in seconds. The default is 5. |
| Whether the broker should close the connection along with a slow consumer | **abortConnection**<br><br>The default is **false**. | No equivalent. In AMQ Broker 7, when a slow consumer is aborted, the connection is also closed. |
| The policy to apply if a slow consumer is detected. | No equivalent. | **<slow-consumer-policy>**<br><br>The default is **NOTIFY**, which will send a **CONSUMER_SLOW** management notification to the application.<br><br>You can also use the **KILL** policy to close the consumer's connection. However, this will impact any other client threads using that connection. |

**Related Information**

- For more information about how to handle slow consumers, see Handling Slow Consumers in *Configuring AMQ Broker*.

### 7.2.5. Message Paging Configuration Properties

In AMQ Broker 7, the process by which the broker stores messages in memory and stores them to disk is significantly different than AMQ 6. Therefore, most of the paging configuration properties in AMQ 6 do not apply to AMQ Broker 7.

In AMQ Broker 7, paging is configured on message addresses. Each address is configured to use a maximum number of bytes. When this limit is reached, messages sent to that address are paged to an on-disk buffer before they reach their queues. The queues are de-paged one page at a time when the address has enough available space.

The following table compares the message paging size limits in AMQ 6 to the equivalent properties in AMQ Broker 7:

| To set... | In AMQ 6 | In AMQ Broker 7 |
|---|---|---|
| The paging size | **maxPageSize**<br><br>This is measured in number of messages, and is variable based on the number of available messages. | **<page-size-bytes>**<br><br>This is measured in the physical page size in bytes (not messages). |

### 7.2.6. Dead Letter Policy Configuration Properties

AMQ Broker 7 handles undeliverable and expired messages much differently than AMQ 6. Dead letter policies are applied to addresses (instead of destinations), there are separate dead letter and expiry destinations (instead of a single dead letter queue), and the dead letter policy configuration is significantly different.

**Dead Letter Policies in AMQ 6**
In AMQ 6, an expired or undeliverable message would be sent to the *dead letter queue* (DLQ) configured for each message's destination. To configure the DLQ for a destination, you could use any of the following dead letter policies:

**sharedDeadLetterStrategy**
: The destination's undeliverable messages are sent to the shared, default DLQ called **ActiveMQ.DLQ**.

**individualDeadLetterStrategy**
: The destination's undeliverable messages are sent to a dedicated DLQ for this destination.

**discardingDeadLetterStrategy**
: The destination's undeliverable messages are discarded.

Within a destination's dead letter policy, you could add the following configuration properties to control the types of messages that should be sent to the destination's DLQ:

| AMQ 6 Configuration Property | Description |
|---|---|
| **processNotPersistent** | Whether non-persistent messages should be sent to the destination's DLQ. The default is **false**. |
| **processExpired** | Whether expired messages should be sent to the destination's DLQ. The default is **true**. |
| **expiration** | Whether an expiry should be applied to the messages sent to the destination's DLQ. The default is 0. |

### Dead Letter Policies in AMQ 7

In AMQ Broker 7, undeliverable messages are sent to the applicable *dead letter address*, and expired messages are sent to the applicable *expiry address*.

In the **broker.xml** configuration file, the default address setting specifies a dead letter address and expiry address. Undeliverable and expired messages will be delivered to the destinations specified by these settings:

```
...
<address-settings>
   <address-setting match="#">
      <dead-letter-address>DLQ</dead-letter-address>
      <expiry-address>ExpiryQueue</expiry-address>
      ...
   </address-setting>
   ...
</address-settings>
...
```

By default, the dead letter and expiry addresses specify the **DLQ** and **ExpiryQueue** destinations, which are defined in the **<addresses>** section:

```
...
<addresses>
   <address name="DLQ">
      <anycast>
         <queue name="DLQ" />
      </anycast>
   </address>
   <address name="ExpiryQueue">
      <anycast>
         <queue name="ExpiryQueue" />
      </anycast>
   </address>
...
</addresses>
...
```

To configure a non-default dead letter policy for an address, you can add a **<dead-letter-address>** and **<expiry-address>** to the address's **<address-setting>** and specify the DLQ and expiry queue it should use.

Unlike AMQ 6, in AMQ Broker 7, you cannot set an expiry time on messages sent to the DLQ. In addition, both persistent and non-persistent messages are sent to the DLQ specified by the address's **<dead-letter-address>**.

# CHAPTER 8. MESSAGE PERSISTENCE AND PAGING

AMQ Broker 7 provides persistence through either a message journal or a JDBC store. The method by which the broker stores messages and pages them to disk is different than AMQ 6, and the configuration properties you use to configure message persistence are changed.

## 8.1. MESSAGE PERSISTENCE CHANGES

AMQ Broker 7 uses a different type of message journal than AMQ 6, and it does not use a journal index.

AMQ 6 used KahaDB for a message store, and it maintained a message journal index to track the position of each message inside the journal. This index enabled the broker to pull paged messages from its journal in batches and place them in its cache.

By default, AMQ Broker 7 uses an in-memory message journal from which the broker can dispatch messages. Therefore, AMQ Broker 7 does not use a message journal index. If a broker instance runs out of memory, messages are paged as they arrive at the broker, but before they are queued. These message page files are stored on disk sequentially in the same order in which they arrived. Then, when memory is freed on the broker, the messages are moved from the page file to the journal on the broker. Because the journal is read sequentially, there is no need to keep an index of messages in the journal.

In addition, AMQ Broker 7 also offers a different JDBC-based message journal option that was not available in AMQ 6.

The AMQ Broker 7 message journal supports the following shared file systems:

- NFSv4
- GFS2

**Related Information**

- For more information about the default in-memory message journal, see About Journal-Based Persistence in *Configuring AMQ Broker*.
- For more information about the new JDBC-based persistence option, see Configuring JDBC Persistence in *Configuring AMQ Broker*.

## 8.2. HOW MESSAGE PERSISTENCE IS CONFIGURED

You use the ***BROKER_INSTANCE_DIR*/etc/broker.xml** configuration file to configure the broker instance's message journal.

The **broker.xml** configuration file contains the following default message journal configuration properties:

```
<core>
   <name>0.0.0.0</name>
   <persistence-enabled>true</persistence-enabled>
   <journal-type>ASYNCIO</journal-type>
   <paging-directory>./data/paging</paging-directory>
```

```
    <bindings-directory>./data/bindings</bindings-directory>

    <journal-directory>./data/journal</journal-directory>

    <large-messages-directory>./data/large-messages</large-messages-directory>

    <journal-datasync>true</journal-datasync>

    <journal-min-files>2</journal-min-files>

    <journal-pool-files>-1</journal-pool-files>

    <journal-buffer-timeout>744000</journal-buffer-timeout>

    <disk-scan-period>5000</disk-scan-period>

    <max-disk-usage>90</max-disk-usage>

    <global-max-size>104857600</global-max-size>

    ...

</core>
```

To configure the message journal, you can change the default values for any of the journal configuration properties. You can also add additional configuration properties.

## 8.3. MESSAGE PERSISTENCE CONFIGURATION PROPERTY CHANGES

AMQ 6 and AMQ Broker 7 both offer a number of configuration properties to control how the broker persists messages. This section compares the configuration properties in the AMQ 6 KahaDB journal to the equivalent properties in the AMQ Broker 7 in-memory message journal.

For complete details on each message persistence configuration property for the in-memory message journal, see the following:

- The Bindings Journal in *Configuring AMQ Broker*
- Messaging Journal Configuration Elements in *Configuring AMQ Broker*

### 8.3.1. Journal Size and Management

The following table compares the journal size and management configuration properties in AMQ 6 to the equivalent properties in AMQ Broker 7:

| To set... | In AMQ 6 | In AMQ Broker 7 |
|---|---|---|
| The time interval between cleaning up data logs that are no longer used | **cleanupInterval**<br><br>The default is 30000 ms. | No equivalent. In AMQ Broker 7, journal files that exceed the pool size are no longer used. |

| To set... | In AMQ 6 | In AMQ Broker 7 |
|---|---|---|
| The number of message store GC cycles that must be completed without cleaning up other files before compaction is triggered | **compactAcksAfterNoGC** | No equivalent. In AMQ Broker 7, compaction is not related to particular record types. |
| Whether compaction should be run when the message store is still growing, or if it should only occur when it has stopped growing | **compactAcksIgnoresStoreGrowth**<br><br>The default is **false**. | No equivalent. |
| The minimum number of journal files that can be stored on the broker before it will compact them | No equivalent. | **<journal-compact-min-files>**<br><br>The default is 10. If you set this value to 0, compaction will be deactivated. |
| The threshold to reach before compaction starts | No equivalent. | **<journal-compact-percentage>**<br><br>The default is 30%. When less than this percentage is considered to be live data, compaction will start. |
| The path to the top-level folder that holds the message store's data files | **directory** | AMQ Broker 7 has a separate directory for each type of journal:<br><br>• **<journal-directory>** - The default is **/data/journal**.<br>• **<bindings-directory>** - The default is **/data/bindings**.<br>• **<paging-directory>** - The default is **/data/paging**.<br>• **<large-message-directory>** - The default is **/data/large-messages**. |
| Whether the bindings directory should be created automatically if it does not already exist | No equivalent. | **<create-bindings-dir>**<br><br>The default is **true**. |
| Whether the journal directory should be created automatically if it does not already exist | No equivalent. | **<create-journal-dir>**<br><br>The default is **true**. |

| To set... | In AMQ 6 | In AMQ Broker 7 |
|---|---|---|
| Whether the message store should periodically compact older journal log files that contain only message acknowledgements | **enableAckCompaction** | No equivalent. |
| The maximum size of the data log files | **journalMaxFileLength**<br><br>The default is 32 MB. | **<journal-file-size>**<br><br>The default is 10485760 bytes (10 MiB). |
| The policy that the broker should use to preallocate the journal files when a new journal file is needed | **preallocationStrategy**<br><br>The default is **sparse_file**. | No equivalent. By default, preallocated journal files are typically filled with zeroes, but it can vary depending on the file system. |
| The policy the broker should use to preallocate the journal files | **preallocationScope**<br><br>The default is **entire_journal**. | AMQ Broker 7 automatically preallocates the journal files specified by **<journal-min-files>** when the broker instance is started. |
| The journal type (either NIO or AIO) | No equivalent. | **<journal-type>**<br><br>You can choose either **NIO** (Java NIO journal), or **ASYNCIO** (Linux asynchronous I/O journal). |
| The minimum number of files that the journal should maintain | No equivalent. | **<journal-min-files>** |
| The number of journal files the broker should keep when reclaiming files | No equivalent. | **<journal-pool-files>**<br><br>The default is -1, which means the broker instance will never delete files on the journal once created. |

### 8.3.2. Write Boundaries

The following table compares the write boundary configuration properties in AMQ 6 to the equivalent properties in AMQ Broker 7:

| To set... | In AMQ 6 | In AMQ Broker 7 |
|---|---|---|
| The time interval between writing the metadata cache to disk | **checkpointInterval**<br><br>The default is 5000 ms. | No equivalent. |

| To set... | In AMQ 6 | In AMQ Broker 7 |
|---|---|---|
| Whether the message store should dispatch queue messages to clients concurrently with message storage | **concurrentStoreAndDispatchQueues**<br><br>The default is **true**. | No equivalent. |
| Whether the message store should dispatch topic messages to interested clients concurrently with message storage | **concurrentStoreAndDispatchTopics**<br><br>The default is **false**. | No equivalent. |
| Whether a disk sync should be performed after each non-transactional journal write | **enableJournalDiskSyncs**<br><br>The default is **true**. | **<journal-sync-transactional>**<br>Flushes transaction data to disk whenever a transaction boundary is reached (commit, prepare, and rollback). The default is **true**.<br>**<journal-sync-nontransactional>**<br>Flushes non-transactional message data to disk (sends and acknowledgements). The default is **true**. |
| When to flush the entire journal buffer | No equivalent. | **<journal-buffer-timeout>**<br><br>The default for NIO is 3,333,333 nanoseconds, and the default for AIO is 500,000 nanoseconds. |
| The amount of data to buffer between journal disk writes | **journalMaxWriteBatchSize**<br><br>The default is 4000 bytes. | No equivalent. |
| The size of the task queue used to buffer the journal's write requests | **maxAsyncJobs**<br><br>The default is 10000. | **<journal-max-io>**<br><br>This property controls the maximum number of write requests that can be in the I/O queue at any given point. The default for NIO is 1, and the default for AIO is 500. |
| Whether to use **fdatasync** on journal writes | No equivalent. | **<journal-datasync>**<br><br>The default is **true**. |

### 8.3.3. Index Configuration

AMQ 6 has a number of configuration properties for configuring the journal index. Because AMQ Broker 7 does not use journal indexes, you do not need to configure any of these properties for your broker instance.

### 8.3.4. Journal Archival

AMQ 6 has several configuration properties for controlling which files are archived and where the archives are stored. In AMQ Broker 7, however, when old journal files are no longer needed, the broker reuses them instead of archiving them. Therefore, you do not need to configure any journal archival properties for your broker instance.

### 8.3.5. Journal Recovery

AMQ 6 has several configuration properties for controlling how the broker checks for corrupted journal files and what to do when it encounters a missing journal file.

In AMQ Broker 7, however, you do not need to configure any journal recovery properties for your broker instance. Journal files have a different format in AMQ Broker 7, which should prevent a corrupted entry in the journal from corrupting the entire journal file. Even if the journal is partially damaged, the broker should still be able to extract data from the undamaged entries.

# CHAPTER 9. BROKER CLUSTERS

You can connect brokers together to form a cluster. Broker clusters enable you to distribute message processing load and balance client connections. They also provide fault tolerance by increasing the number of brokers to which clients can connect.

## 9.1. BROKER CLUSTERING CHANGES

In AMQ Broker 7, broker networks are called broker clusters. The brokers in the cluster are connected by cluster connections (which reference **connector** elements). Members of a cluster can be configured to discover each other dynamically (using UDP or JGroups), or statically (by manually specifying a list of cluster members).

A cluster configuration is a required prerequisite for high-availability (HA). You must configure the cluster before you can configure HA, even if the cluster consists of only a single live broker.

You can configure broker clusters in many different topologies, though symmetric and chain clusters are the most common. Regardless of the topology, you can scale clusters up and down without message loss (as long as you have configured the broker to send its messages to another broker in the cluster).

Broker clusters distribute (and redistribute) messages differently than broker networks in AMQ 6. In AMQ 6, messages always arrived on a specific queue and were then pulled from one broker to another based on consumer interest. In AMQ Broker 7, queue definitions and consumers are shared across the cluster, and messages are routed across the cluster as they are received at the broker.

**IMPORTANT**

Do not attempt to combine AMQ 6 brokers and AMQ Broker 7 brokers in the same cluster.

## 9.2. HOW BROKER CLUSTERS ARE CONFIGURED

You configure a broker cluster by creating a broker instance for each member of the cluster, and then adding the cluster settings to each broker instance.

Cluster settings consist of the following:

**Discovery groups**

For use with dynamic discovery, a discovery group defines how the broker instance discovers other members in the cluster. Discovery can use either UDP or JGroups.

**Broadcast groups**

For use with dynamic discovery, a broadcast group defines how the broker instance transmits cluster-related information to other members in the cluster. Broadcast can use either UDP or JGroups, but it must match its discovery groups counterpart.

**Cluster connections**

How the broker instance should connect to other members of the cluster. You can specify a discovery group or a static list of cluster members. You can also specify message redistribution and max hop properties.

### 9.2.1. Creating a Broker Cluster

This procedure demonstrates how to create a basic, two-broker cluster with static discovery.

**Procedure**

1. Create the first broker instance by using the **artemis create** command.
   This example creates a new broker instance called **broker1**.

   ```
   $ sudo INSTALL_DIR/bin/artemis create broker1 --user user --password pass --role amq
   ```

2. Create a second broker instance for the second member of the cluster.
   For each additional broker instance, you should use the **--port-offset** parameter to avoid port collisions with the previous broker instances.

   This example creates a second broker instance called **broker2**.

   ```
   $ sudo INSTALL_DIR/bin/artemis create broker2 --port-offset 100 --user user --password pass --role amq
   ```

3. For the first broker instance, open the ***BROKER_INSTANCE_DIR*/etc/broker.xml** configuration file and add the cluster settings.
   For static discovery, you must add a connector and a static cluster connection. This example configures **broker1** to connect to **broker2**.

   ```
   <!-- Connectors -->
   <connectors>
      <connector name="netty-connector">tcp://localhost:61616</connector>
      <!-- connector to broker2 -->
      <connector name="broker2-connector">tcp://localhost:61617</connector>
   </connectors>

   <!-- Clustering configuration -->
   <cluster-connections>
      <cluster-connection name="my-cluster">
         <connector-ref>netty-connector</connector-ref>
         <retry-interval>500</retry-interval>
         <use-duplicate-detection>true</use-duplicate-detection>
         <message-load-balancing>STRICT</message-load-balancing>
         <max-hops>1</max-hops>
         <static-connectors>
            <connector-ref>broker2-connector</connector-ref>
         </static-connectors>
      </cluster-connection>
   </cluster-connections>
   ```

4. For the second broker instance, open the ***BROKER_INSTANCE_DIR*/etc/broker.xml** configuration file and add the cluster settings.
   This example configures **broker2** to connect to **broker1**.

   ```
   <!-- Connectors -->
   <connectors>
      <connector name="netty-connector">tcp://localhost:61617</connector>
      <!-- connector to broker1 -->
      <connector name="broker1-connector">tcp://localhost:61616</connector>
   </connectors>

   <!-- Clustering configuration -->
   ```

```
<cluster-connections>
   <cluster-connection name="my-cluster">
      <connector-ref>netty-connector</connector-ref>
      <retry-interval>500</retry-interval>
      <use-duplicate-detection>true</use-duplicate-detection>
      <message-load-balancing>STRICT</message-load-balancing>
      <max-hops>1</max-hops>
      <static-connectors>
         <connector-ref>broker1-connector</connector-ref>
      </static-connectors>
   </cluster-connection>
</cluster-connections>
```

**Related Information**

- For full details about creating broker clusters and configuring message redistribution and client load balancing, see Setting up a broker cluster in *Configuring AMQ Broker*.

### 9.2.2. Additional Broker Cluster Topologies

Broker clusters can be connected in many different topologies. In AMQ Broker 7, symmetric and chain clusters are the most common.

#### Example: Symmetric Cluster

In a full mesh topology, each broker is connected to every other broker in the cluster. This means that every broker in the cluster is no more than one hop away from every other broker.

This example uses dynamic discovery to enable the brokers in the cluster to discover each other. By setting **max-hops** to **1**, each broker will connect to every other broker:

```
<!-- Clustering configuration -->
<broadcast-groups>
   <broadcast-group name="my-broadcast-group">
      <group-address>${udp-address:231.7.7.7}</group-address>
      <group-port>9876</group-port>
      <broadcast-period>100</broadcast-period>
      <connector-ref>netty-connector</connector-ref>
   </broadcast-group>
</broadcast-groups>

<discovery-groups>
   <discovery-group name="my-discovery-group">
      <group-address>${udp-address:231.7.7.7}</group-address>
      <group-port>9876</group-port>
      <refresh-timeout>10000</refresh-timeout>
   </discovery-group>
</discovery-groups>

<cluster-connections>
   <cluster-connection name="my-cluster">
      <connector-ref>netty-connector</connector-ref>
      <retry-interval>500</retry-interval>
      <use-duplicate-detection>true</use-duplicate-detection>
      <message-load-balancing>ON_DEMAND</message-load-balancing>
```

```
      <max-hops>1</max-hops>
      <discovery-group-ref discovery-group-name="my-discovery-group"/>
    </cluster-connection>
</cluster-connections>
```

### Example: Chain Cluster

In a chain cluster, the brokers form a linear "chain" with a broker on each end and all other brokers connecting to the previous and next brokers in the chain (for example, A→B→C).

This example uses static discovery to connect three brokers into a chain cluster. Each broker connects to the next broker in the chain, and **max-hops** is set to **2** to enable messages to flow through the full chain.

The first broker is configured like this:

```
<connectors>
  <connector name="netty-connector">tcp://localhost:61616</connector>
  <!-- connector to broker2 -->
  <connector name="broker2-connector">tcp://localhost:61716</connector>
</connectors>


<cluster-connections>
  <cluster-connection name="my-cluster">
    <address>jms</address>
    <connector-ref>netty-connector</connector-ref>
    <retry-interval>500</retry-interval>
    <use-duplicate-detection>true</use-duplicate-detection>
    <message-load-balancing>STRICT</message-load-balancing>
    <max-hops>2</max-hops>
    <static-connectors allow-direct-connections-only="true">
      <connector-ref>broker2-connector</connector-ref>
    </static-connectors>
  </cluster-connection>
</cluster-connections>
```

The second broker is configured like this:

```
<connectors>
  <connector name="netty-connector">tcp://localhost:61716</connector>
  <!-- connector to broker3 -->
  <connector name="broker3-connector">tcp://localhost:61816</connector>
</connectors>


<cluster-connections>
  <cluster-connection name="my-cluster">
    <address>jms</address>
    <connector-ref>netty-connector</connector-ref>
    <retry-interval>500</retry-interval>
    <use-duplicate-detection>true</use-duplicate-detection>
    <message-load-balancing>STRICT</message-load-balancing>
    <max-hops>1</max-hops>
    <static-connectors allow-direct-connections-only="true">
      <connector-ref>broker3-connector</connector-ref>
```

```
      </static-connectors>
    </cluster-connection>
  </cluster-connections>
```

Finally, the third broker is configured like this:

```
<connectors>
  <connector name="netty-connector">tcp://localhost:61816</connector>
</connectors>

<cluster-connections>
  <cluster-connection name="my-cluster">
    <address>jms</address>
    <connector-ref>netty-connector</connector-ref>
    <retry-interval>500</retry-interval>
    <use-duplicate-detection>true</use-duplicate-detection>
    <message-load-balancing>STRICT</message-load-balancing>
    <max-hops>0</max-hops>
  </cluster-connection>
</cluster-connections>
```

## 9.3. BROKER CLUSTER CONFIGURATION PROPERTIES

The following table compares the broker network configuration properties in AMQ 6 to the equivalent **cluster-connection** properties in AMQ Broker 7:

| To set... | In AMQ 6 | In AMQ Broker 7 |
|---|---|---|
| Excluded destinations | **excludedDestinations** | No equivalent. |
| The number of hops that a message can make through the cluster | **networkTTL**<br><br>The default is **1**, which means that a message can make just one hop to a neighboring broker. | **<max-hops>**<br><br>Sets this broker instance to load balance messages to brokers which might be connected to it indirectly with other brokers are intermediaries in a chain. The default is **1**, which means that messages are distributed only to other brokers directly connected to this broker instance. |
| Replay messages when there are no consumers | **replayWhenNoConsumers** | No equivalent. However, you can set **<redistribution-delay>** to define the amount of time with no consumers (in milliseconds) after which messages should be redelivered as though arriving for the first time. |

| To set... | In AMQ 6 | In AMQ Broker 7 |
| --- | --- | --- |
| Whether to broadcast advisory messages for temporary destinations in the cluster | **bridgeTempDestinations**<br><br>The default is **true**. This property was typically used for temporary destinations created for request-reply messages. This would enable consumers of these messages to be connected to another broker in the network and still be able to send the reply to the temporary destination specified in the **JMSReplyTo** header. | No equivalent. In AMQ Broker 7, temporary destinations are never clustered. |
| The credentials to use to authenticate this broker with a remote broker | **userNamepassword** | **<cluster-user><cluster-password>** |
| Set the route priority for a connector | **decreaseNetworkConsumerPriority**<br><br>The default is **false**. If set to **true**, local consumers have a priority of **0**, and network subscriptions have a priority of **-5**. In addition, the priority of a network subscription is reduced by 1 for every network hop that it traverses. | No equivalent. |
| Whether and how messages should be distributed between other brokers in the cluster | No equivalent. | **<message-load-balancing>**<br><br>This can be set to **OFF** (no load balancing), **STRICT** (forward messages to all brokers in the cluster that have a matching queue), or **ON_DEMAND** (forward messages only to brokers in the cluster that have active consumers or a matching selector). The default is **ON_DEMAND**. |

| To set... | In AMQ 6 | In AMQ Broker 7 |
| --- | --- | --- |
| Enable a cluster network connection to both produce and consume messages | **duplex**<br><br>By default, network connectors are unidirectional. However, you could set them to duplex to enable messages to flow in both directions. This was typically used for hub-and-spoke networks in which the hub was behind a firewall. | No equivalent. Cluster connections are unidirectional only. However, you can configure a pair of cluster connections between each broker, one from each end. For more information about setting up a broker cluster, see Setting up a broker clusterin *Configuring AMQ Broker*. |

# CHAPTER 10. HIGH AVAILABILITY AND FAILOVER

After creating a cluster configuration, you can link broker instances together to form high availability (HA) pairs. An HA pair consists of a master broker that serves client requests, and one or more slave brokers that replace the master if it can no longer communicate with clients.

In AMQ Broker 7, a cluster configuration is required for HA. Broker clusters can consist of either a set of non-HA brokers or HA pairs.

AMQ Broker 7 provides the following HA policies:

**Replication**

Replication synchronizes the data between the master and slave brokers over the network. With replication, you can enable failback to return control to the master broker when it comes back online after a failure event and allow clients to fail back to it. You can also create HA groups in which multiple master brokers share one or more slave brokers, and colocate slave brokers in the same JVM as the master broker.

**IMPORTANT**

Starting in 7.5, network pinging, which was previously available for use with the replication HA policy, is a deprecated feature. Network pinging cannot protect a broker cluster from network isolation issues that can lead to irrecoverable message loss. This feature will be removed in a future release. Red Hat continues to support existing AMQ Broker deployments that use network pinging. However, Red Hat no longer recommends use of network pinging in new deployments. For guidance on configuring a broker cluster for high availability and to avoid network isolation issues, see Implementing high availability.

**Shared Store**

Shared store provides a location for the master and slave brokers to share messaging data. Using a shared store is generally preferable, as it offers the following benefits over replication:

- Performance (shared stores are faster)
- No split-brain issues
- Fewer brokers required to maintain quorum (replication requires at least three)
  Like with replication, you can enable failback to return control to the master broker after a failure event and allow clients to fail back to it. You can configure multiple slave brokers for a master broker, and colocate slave brokers.

For more information about HA and failover, see Implementing high availability in *Configuring AMQ Broker*.

## 10.1. HIGH AVAILABILITY AND FAILOVER CHANGES

High availability in AMQ Broker 7 differs from AMQ 6 based on how the master is determined and when the broker connections become active.

In AMQ Broker 7, the master and slave roles are fixed. You specify which broker instance is the master, and the slave only becomes active in certain conditions. In AMQ 6, the master and slave roles were not fixed. Instead, the brokers in an HA pair would compete for a lock, and the winner would become the master.

In AMQ Broker 7, in an HA pair, the slave broker's acceptors are active even if the broker is inactive. In AMQ 6, the slave broker's transport connectors did not become active until the broker became active.

## 10.2. HOW HIGH AVAILABILITY IS CONFIGURED

You configure HA by adding an HA policy configuration to the ***BROKER_INSTANCE_DIR*/etc/broker.xml** configuration file of each broker.

### Example: HA Pair with Shared Store

The master broker is configured like this. By setting **failover-on-shutdown** to **true**, the HA pair will fail over to the slave broker if the master broker is shut down:

```
<configuration>
  <core>
    ...
    <ha-policy>
      <shared-store>
        <master/>
          <failover-on-shutdown>true</failover-on-shutdown>
      </shared-store>
    </ha-policy>
    ...
  </core>
</configuration>
```

The slave broker is configured like this. By setting **failover-on-shutdown** to **true**, this slave broker will become the master if the current master broker is shut down:

```
<configuration>
  <core>
    ...
    <ha-policy>
      <shared-store>
        <slave/>
          <failover-on-shutdown>true</failover-on-shutdown>
      </shared-store>
    </ha-policy>
    ...
  </core>
</configuration>
```

### Related Information

For full details on configuring HA policies, see the following topics:

- Configuring high availability in *Configuring AMQ Broker*

*Revised on 2020-12-03 08:49:05 UTC*