



Red Hat AMQ 2020.Q4

Deploying AMQ Broker on OpenShift

For Use with AMQ Broker 7.8

Red Hat AMQ 2020.Q4 Deploying AMQ Broker on OpenShift

For Use with AMQ Broker 7.8

Legal Notice

Copyright © 2022 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

Learn how to install and deploy AMQ Broker on OpenShift Container Platform.

Table of Contents

CHAPTER 1. INTRODUCTION TO AMQ BROKER ON OPENSIFT CONTAINER PLATFORM	5
1.1. VERSION COMPATIBILITY AND SUPPORT	5
1.2. UNSUPPORTED FEATURES	5
CHAPTER 2. PLANNING A DEPLOYMENT OF AMQ BROKER ON OPENSIFT CONTAINER PLATFORM ..	6
2.1. COMPARISON OF DEPLOYMENT METHODS	6
2.2. OVERVIEW OF THE AMQ BROKER OPERATOR CUSTOM RESOURCE DEFINITIONS	7
2.3. OVERVIEW OF THE AMQ BROKER OPERATOR SAMPLE CUSTOM RESOURCES	8
2.4. HOW THE OPERATOR CHOOSES CONTAINER IMAGES	9
2.4.1. Environment variables for broker container images	9
2.4.2. Environment variables for Init Container images	11
2.5. OPERATOR DEPLOYMENT NOTES	12
CHAPTER 3. DEPLOYING AMQ BROKER ON OPENSIFT CONTAINER PLATFORM USING THE AMQ BROKER OPERATOR	14
3.1. PREREQUISITES	14
3.2. INSTALLING THE OPERATOR USING THE CLI	14
3.2.1. Getting the Operator code	14
3.2.2. Deploying the Operator using the CLI	16
3.3. INSTALLING THE OPERATOR USING OPERATORHUB	18
3.3.1. Overview of the Operator Lifecycle Manager	18
3.3.2. Installing the Operator in OperatorHub	18
3.3.3. Deploying the Operator from OperatorHub	19
3.4. CREATING OPERATOR-BASED BROKER DEPLOYMENTS	21
3.4.1. Deploying a basic broker instance	21
3.4.2. Deploying clustered brokers	24
3.4.3. Applying Custom Resource changes to running broker deployments	26
CHAPTER 4. CONFIGURING OPERATOR-BASED BROKER DEPLOYMENTS	27
4.1. HOW THE OPERATOR GENERATES THE BROKER CONFIGURATION	27
4.1.1. How the Operator generates the address settings configuration	27
4.1.2. Directory structure of a broker Pod	28
4.2. CONFIGURING ADDRESSES AND QUEUES FOR OPERATOR-BASED BROKER DEPLOYMENTS	29
4.2.1. Differences in configuration of address and queue settings between OpenShift and standalone broker deployments	30
4.2.2. Creating addresses and queues for an Operator-based broker deployment	31
4.2.3. Matching address settings to configured addresses in an Operator-based broker deployment	33
4.3. CONFIGURING BROKER STORAGE REQUIREMENTS	39
4.3.1. Configuring broker storage size	39
4.4. CONFIGURING RESOURCE LIMITS AND REQUESTS FOR OPERATOR-BASED BROKER DEPLOYMENTS	42
4.4.1. Configuring broker resource limits and requests	43
4.5. SPECIFYING A CUSTOM INIT CONTAINER IMAGE	46
4.6. CONFIGURING OPERATOR-BASED BROKER DEPLOYMENTS FOR CLIENT CONNECTIONS	49
4.6.1. Configuring acceptors	49
4.6.2. Securing broker-client connections	52
4.6.2.1. Configuring a broker certificate for host name verification	52
4.6.2.2. Configuring one-way TLS	53
4.6.2.3. Configuring two-way TLS	55
4.6.3. Networking Services in your broker deployments	56
4.6.4. Connecting to the broker from internal and external clients	57
4.6.4.1. Connecting to the broker from internal clients	57

4.6.4.2. Connecting to the broker from external clients	57
4.6.4.3. Connecting to the Broker using a NodePort	59
4.7. CONFIGURING LARGE MESSAGE HANDLING FOR AMQP MESSAGES	59
4.7.1. Configuring AMQP acceptors for large message handling	59
4.8. HIGH AVAILABILITY AND MESSAGE MIGRATION	61
4.8.1. High availability	61
4.8.2. Message migration	62
4.8.3. Migrating messages upon scaledown	64
CHAPTER 5. CONNECTING TO AMQ MANAGEMENT CONSOLE FOR AN OPERATOR-BASED BROKER DEPLOYMENT	66
5.1. CONNECTING TO AMQ MANAGEMENT CONSOLE	66
5.2. ACCESSING AMQ MANAGEMENT CONSOLE LOGIN CREDENTIALS	67
CHAPTER 6. UPGRADING AN OPERATOR-BASED BROKER DEPLOYMENT	69
6.1. BEFORE YOU BEGIN	69
6.2. UPGRADING THE OPERATOR USING THE CLI	69
6.2.1. Prerequisites	69
6.2.2. Upgrading version 0.19 of the Operator	69
6.2.3. Upgrading version 0.18 of the Operator	71
6.2.4. Upgrading version 0.17 of the Operator	72
6.2.5. Upgrading version 0.15 of the Operator	74
6.2.6. Upgrading version 0.13 of the Operator	75
6.2.7. Upgrading version 0.9 of the Operator	77
6.3. UPGRADING THE OPERATOR USING OPERATORHUB	78
6.3.1. Prerequisites	78
6.3.2. Before you begin	79
6.3.3. Upgrading the Operator using OperatorHub	79
6.4. UPGRADING THE BROKER CONTAINER IMAGE BY SPECIFYING AN AMQ BROKER VERSION	80
CHAPTER 7. DEPLOYING AMQ BROKER ON OPENSIFT CONTAINER PLATFORM USING APPLICATION TEMPLATES	84
7.1. PREREQUISITES	84
7.2. INSTALLING THE IMAGE STREAMS AND APPLICATION TEMPLATES	84
7.3. PREPARING A TEMPLATE-BASED BROKER DEPLOYMENT	85
7.4. DEPLOYING A BASIC BROKER	86
7.4.1. Creating the broker application	87
7.4.2. About sensitive credentials	88
7.4.3. Deploying and starting the broker application	89
7.5. CONNECTING EXTERNAL CLIENTS TO TEMPLATE-BASED BROKER DEPLOYMENTS	91
7.5.1. Configuring SSL	91
7.5.2. Generating the AMQ Broker secret	92
7.5.3. Creating an SSL Route	92
CHAPTER 8. TEMPLATE-BASED BROKER DEPLOYMENT EXAMPLES	95
8.1. DEPLOYING A BASIC BROKER WITH SSL	95
8.1.1. Deploying the image and template	95
8.1.2. Deploying the application	96
8.1.3. Creating a Route	97
8.2. DEPLOYING A BASIC BROKER WITH PERSISTENCE AND SSL	97
8.2.1. Deploy the image and template	98
8.2.2. Deploy the application	99
8.2.3. Creating a Route	100
8.3. DEPLOYING A SET OF CLUSTERED BROKERS	101

8.3.1. Distributing messages	101
8.3.2. Deploy the image and template	101
8.3.3. Deploying the application	102
8.3.4. Creating Routes for the AMQ Broker management console	104
8.4. DEPLOYING A SET OF CLUSTERED SSL BROKERS	105
8.4.1. Distributing messages	106
8.4.2. Deploying the image and template	106
8.4.3. Deploying the application	107
8.5. DEPLOYING A BROKER WITH CUSTOM CONFIGURATION	109
8.5.1. Deploy the image and template	109
8.5.2. Deploy the application	110
8.6. BASIC SSL CLIENT EXAMPLE	110
8.6.1. Configuring the client	110
8.7. EXTERNAL CLIENTS USING SUB-DOMAINS EXAMPLE	111
8.7.1. Exposing the brokers	111
8.7.2. Connecting the clients	111
8.8. EXTERNAL CLIENTS USING PORT BINDING EXAMPLE	112
8.8.1. Exposing the brokers	112
8.8.2. Connecting the clients	113
CHAPTER 9. UPGRADING A TEMPLATE-BASED BROKER DEPLOYMENT	115
9.1. UPGRADING NON-PERSISTENT BROKER DEPLOYMENTS	115
9.2. UPGRADING PERSISTENT BROKER DEPLOYMENTS	116
CHAPTER 10. MONITORING YOUR BROKERS	118
10.1. VIEWING BROKERS IN FUSE CONSOLE	118
10.2. MONITORING BROKER RUNTIME METRICS USING PROMETHEUS	120
10.2.1. Metrics overview	120
10.2.2. Enabling the Prometheus plugin for a running broker deployment	122
10.2.3. Accessing Prometheus metrics for a running broker Pod	123
10.3. MONITORING BROKER RUNTIME DATA USING JMX	123
CHAPTER 11. REFERENCE	125
11.1. CUSTOM RESOURCE CONFIGURATION REFERENCE	125
11.1.1. Broker Custom Resource configuration reference	125
11.1.2. Address Custom Resource configuration reference	167
11.2. APPLICATION TEMPLATE PARAMETERS	168
11.3. LOGGING	171

CHAPTER 1. INTRODUCTION TO AMQ BROKER ON OPENSIFT CONTAINER PLATFORM

Red Hat AMQ Broker 7.8 is available as a containerized image for use with OpenShift Container Platform (OCP) 3.11 and 4.5 and 4.6.

AMQ Broker is based on Apache ActiveMQ Artemis. It provides a message broker that is JMS-compliant. After you have set up the initial broker pod, you can quickly deploy duplicates by using OpenShift Container Platform features.

1.1. VERSION COMPATIBILITY AND SUPPORT

For details about OpenShift Container Platform image version compatibility, see:

- [OpenShift and Atomic Platform 3.x Tested Integrations](#)
- [OpenShift Container Platform 4.x Tested Integrations](#)

1.2. UNSUPPORTED FEATURES

- Master-slave-based high availability
High availability (HA) achieved by configuring master and slave pairs is not supported. Instead, when pods are scaled down, HA is provided in OpenShift by using the scaledown controller, which enables message migration.

External Clients that connect to a cluster of brokers, either through the OpenShift proxy or by using bind ports, may need to be configured for HA accordingly. In a clustered scenario, a broker will inform certain clients of the addresses of all the broker's host and port information. Since these are only accessible internally, certain client features either will not work or will need to be disabled.

Client	Configuration
Core JMS Client	Because external Core Protocol JMS clients do not support HA or any type of failover, the connection factories must be configured with useTopologyForLoadBalancing=false .
AMQP Clients	AMQP clients do not support failover lists

- Durable subscriptions in a cluster
When a durable subscription is created, this is represented as a durable queue on the broker to which a client has connected. When a cluster is running within OpenShift the client does not know on which broker the durable subscription queue has been created. If the subscription is durable and the client reconnects there is currently no method for the load balancer to reconnect it to the same node. When this happens, it is possible that the client will connect to a different broker and create a duplicate subscription queue. For this reason, using durable subscriptions with a cluster of brokers is not recommended.

CHAPTER 2. PLANNING A DEPLOYMENT OF AMQ BROKER ON OPENSIFT CONTAINER PLATFORM

2.1. COMPARISON OF DEPLOYMENT METHODS

There are two ways to deploy AMQ Broker on OpenShift Container Platform:

- [Using the AMQ Broker Operator](#) (recommended)
- [Using application templates](#)

This section describes each of these deployment methods.

Deployment using the AMQ Broker Operator (recommended)

Operators are programs that enable you to package, deploy, and manage OpenShift applications. Often, Operators automate common or complex tasks. Commonly, Operators are intended to provide:

- Consistent, repeatable installations
- Health checks of system components
- Over-the-air (OTA) updates
- Managed upgrades

The AMQ Broker Operator is the recommended way to create broker deployments on OpenShift Container Platform. Operators enable you to make changes while your broker instances are running, because they are always listening for changes to the Custom Resource (CR) instances that you used to configure your deployment. When you make changes to a CR, the Operator reconciles the changes with the existing broker deployment and updates the deployment to reflect the changes. In addition, the Operator provides a message migration capability, which ensures the integrity of messaging data. When a broker in a clustered deployment shuts down due to failure or intentional scaledown of the deployment, this capability migrates messages to a broker Pod that is still running in the same broker cluster.

Deployment using application templates



IMPORTANT

Starting in 7.8, the use of application templates for deploying AMQ Broker on OpenShift Container Platform is a deprecated feature. This feature will be removed in a future release. Red Hat continues to support existing deployments that are based on application templates. However, Red Hat does not recommend using application templates for new deployments. For new deployments, Red Hat recommends using the AMQ Broker Operator.

A *template* is a way to describe objects that can be parameterized and processed for creation by OpenShift Container Platform. You can use a template to describe anything that you have permission to create within an OpenShift project, for example, Services or build configurations. AMQ Broker has some sample *application templates* that enable you to create various types of broker deployments as DeploymentConfig- or StatefulSet-based applications. You configure your broker deployments by specifying values for the environment variables included in the application templates. A limitation of templates is that while they are effective for creating an initial broker

deployment, they do not provide a mechanism for updating the deployment. In addition, because AMQ Broker does not provide a message migration capability for template-based deployments, templates are not recommended for use in a production environment.

Additional resources

- To learn how to use the AMQ Broker Operator to create a broker deployment, see [Chapter 3, *Deploying AMQ Broker on OpenShift Container Platform using the AMQ Broker Operator*](#) .
- For more information about message migration using the Operator, see [Section 4.8, “High availability and message migration”](#).

2.2. OVERVIEW OF THE AMQ BROKER OPERATOR CUSTOM RESOURCE DEFINITIONS

In general, a Custom Resource Definition (CRD) is a schema of configuration items that you can modify for a custom OpenShift object deployed with an Operator. By creating a corresponding Custom Resource (CR) instance, you can specify values for configuration items in the CRD. If you are an Operator developer, what you expose through a CRD essentially becomes the API for how a deployed object is configured and used. You can directly access the CRD through regular HTTP **curl** commands, because the CRD gets exposed automatically through Kubernetes.

You can install the AMQ Broker Operator using either the OpenShift command-line interface (CLI), or the Operator Lifecycle Manager, through the OperatorHub graphical interface. In either case, the AMQ Broker Operator includes the CRDs described below.

Main broker CRD

You deploy a CR instance based on this CRD to create and configure a broker deployment. Based on how you install the Operator, this CRD is:

- The **broker_activemqartemis_crd** file in the **crds** directory of the Operator installation archive (OpenShift CLI installation method)
- The **ActiveMQArtemis** CRD in the **Custom Resource Definitions** section of the OpenShift Container Platform web console (OperatorHub installation method)

Address CRD

You deploy a CR instance based on this CRD to create addresses and queues for a broker deployment.

Based on how you install the Operator, this CRD is:

- The **broker_activemqartemisaddress_crd** file in the **crds** directory of the Operator installation archive (OpenShift CLI installation method)
- The **ActiveMQArtemisAddresss** CRD in the **Custom Resource Definitions** section of the OpenShift Container Platform web console (OperatorHub installation method)

Scaledown CRD

The **Operator** automatically creates a CR instance based on this CRD when instantiating a scaledown controller for message migration.

Based on how you install the Operator, this CRD is:

- The **broker_activemqartemiscaledown_crd** file in the **crds** directory of the Operator installation archive (OpenShift CLI installation method)
- The **ActiveMQArtemisScaledown** CRD in the **Custom Resource Definitions** section of the OpenShift Container Platform web console (OperatorHub installation method).

Additional resources

- To learn how to install the AMQ Broker Operator (and all included CRDs) using:
 - The OpenShift CLI, see [Section 3.2, “Installing the Operator using the CLI”](#)
 - The Operator Lifecycle Manager and OperatorHub graphical interface, see [Section 3.3, “Installing the Operator using OperatorHub”](#).
- For complete configuration references to use when creating CR instances based on the main broker and address CRDs, see:
 - [Section 11.1.1, “Broker Custom Resource configuration reference”](#)
 - [Section 11.1.2, “Address Custom Resource configuration reference”](#)

2.3. OVERVIEW OF THE AMQ BROKER OPERATOR SAMPLE CUSTOM RESOURCES

The AMQ Broker Operator archive that you download and extract during installation includes sample Custom Resource (CR) files in the **deploy/crs** directory. These sample CR files enable you to:

- Deploy a minimal broker without SSL or clustering.
- Define addresses.

The broker Operator archive that you download and extract also includes CRs for example deployments in the **deploy/examples** directory, as listed below.

artemis-basic-deployment.yaml

Basic broker deployment.

artemis-persistence-deployment.yaml

Broker deployment with persistent storage.

artemis-cluster-deployment.yaml

Deployment of clustered brokers.

artemis-persistence-cluster-deployment.yaml

Deployment of clustered brokers with persistent storage.

artemis-ssl-deployment.yaml

Broker deployment with SSL security.

artemis-ssl-persistence-deployment.yaml

Broker deployment with SSL security and persistent storage.

artemis-aio-journal.yaml

Use of asynchronous I/O (AIO) with the broker journal.

address-queue-create.yaml

Address and queue creation.

2.4. HOW THE OPERATOR CHOOSES CONTAINER IMAGES

When you create a Custom Resource (CR) instance for a broker deployment based on *at least* version 7.8.5-opr-2 of the Operator, you **do not** need to explicitly specify broker or Init Container image names in the CR. By default, if you deploy a CR and do not explicitly specify container image values, the Operator automatically chooses the appropriate container images to use.



NOTE

If you install the Operator using the OpenShift command-line interface, the Operator installation archive includes a sample CR file called **broker_activemqartemis_cr.yaml**. In the sample CR, the **spec.deploymentPlan.image** property is included and set to its default value of **placeholder**. This value indicates that the Operator does not choose a broker container image until you deploy the CR.

The **spec.deploymentPlan.initImage** property, which specifies the Init Container image, is **not** included in the **broker_activemqartemis_cr.yaml** sample CR file. If you do not explicitly include the **spec.deploymentPlan.initImage** property in your CR and specify a value, the Operator chooses an appropriate built-in Init Container image to use when you deploy the CR.

How the Operator chooses these images is described in this section.

To choose broker and Init Container images, the Operator first determines an AMQ Broker version to which the images should correspond. The Operator determines the version as follows:

- If the **spec.upgrades.enabled** property in the main CR is already set to **true** and the **spec.version** property specifies **7.7.0**, **7.8.0**, **7.8.1**, or **7.8.2**, the Operator uses that specified version.
- If **spec.upgrades.enabled** is **not** set to **true**, or **spec.version** is set to an AMQ Broker version earlier than **7.7.0**, the Operator uses the **latest** version of AMQ Broker (that is, **7.8.5**).

Note: For IBM Z and IBM Power Systems, **7.8.1** and **7.8.2** are the only valid value for **spec.version**.

The Operator then detects your container platform. The AMQ Broker Operator can run on the following container platforms:

- OpenShift Container Platform (x86_64)
- OpenShift Container Platform on IBM Z (s390x)
- OpenShift Container Platform on IBM Power Systems (ppc64le)

Based on the version of AMQ Broker and your container platform, the Operator then references two sets of environment variables in the **operator.yaml** configuration file. These sets of environment variables specify broker and Init Container images for various versions of AMQ Broker, as described in the following sub-sections.

2.4.1. Environment variables for broker container images

The environment variables included in the **operator.yaml** configuration file for broker container images have the following naming convention:

OpenShift Container Platform

RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_<AMQ_Broker_version_identifier>

OpenShift Container Platform on IBM Z

RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_<AMQ_Broker_version_identifier>_s390x

OpenShift Container Platform on IBM Power Systems

RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_<AMQ_Broker_version_identifier>_ppc64le

Environment variable names for each supported container platform and specific AMQ Broker versions are shown in the table.

Container platform	Environment variable names
OpenShift Container Platform	<ul style="list-style-type: none"> ● RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_770 ● RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_780 ● RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_781
OpenShift Container Platform on IBM Z	<ul style="list-style-type: none"> ● RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_770_s390x ● RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_780_s390x ● RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_781_s390x
OpenShift Container Platform on IBM Power Systems	<ul style="list-style-type: none"> ● RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_770_ppc64le ● RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_780_ppc64le ● RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_781_ppc64le

The value of each environment variable specifies a broker container image that is available from Red Hat. For example:

```
- name: RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_787
  #value: registry.redhat.io/amq7/amq-broker:7.8-33
  value: registry.redhat.io/amq7/amq-broker@sha256:4d60775cd384067147ab105f41855b5a7af855c4d9cbef1d4dea566cbe214558
```

Therefore, based on an AMQ Broker version and your container platform, the Operator determines the applicable environment variable name. The Operator uses the corresponding image value when starting the broker container.



NOTE

In the **operator.yaml** file, the Operator uses an image that is represented by a *Secure Hash Algorithm* (SHA) value. The comment line, which begins with a number sign (#) symbol, denotes that the SHA value corresponds to a specific container image tag.

2.4.2. Environment variables for Init Container images

The environment variables included in the **operator.yaml** configuration file for Init Container images have the following naming convention:

OpenShift Container Platform

RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_<AMQ_Broker_version_identifier>

OpenShift Container Platform on IBM Z

RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_s390x_<AMQ_Broker_version_identifier>

OpenShift Container Platform on IBM Power Systems

RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_ppc64le_<AMQ_Broker_version_identifier>

Environment variable names for each supported container platform and specific AMQ Broker versions are shown in the table.

Container platform	Environment variable names
OpenShift Container Platform	<ul style="list-style-type: none"> ● RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_770 ● RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_780 ● RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_781
OpenShift Container Platform on IBM Z	<ul style="list-style-type: none"> ● RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_s390x_770 ● RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_s390x_780 ● RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_s390x_781
OpenShift Container Platform on IBM Power Systems	<ul style="list-style-type: none"> ● RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_ppc64le_770 ● RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_ppc64le_780 ● RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_ppc64le_781

The value of each environment variable specifies an Init Container image that is available from Red Hat. For example:

```
- name: RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_787
  #value: registry.redhat.io/amq7/amq-broker-init-rhel7:7.8-1
  value: registry.redhat.io/amq7/amq-broker-init-
  rhel7@sha256:f7482d07ecaa78d34c37981447536e6f73d4013ec0c64ff787161a75e4ca3567
```

Therefore, based on an AMQ Broker version and your container platform, the Operator determines the applicable environment variable name. The Operator uses the corresponding image value when starting the Init Container.



NOTE

As shown in the example, the Operator uses an image that is represented by a *Secure Hash Algorithm* (SHA) value. The comment line, which begins with a number sign (#) symbol, denotes that the SHA value corresponds to a specific container image tag. Observe that the corresponding container image tag is **not** a floating tag in the form of **0.2**. This means that the container image used by the Operator remains fixed. The Operator **does not** automatically pull and use a new *micro* image version (that is, **0.2-n**, where *n* is the latest micro version) when it becomes available from Red Hat.

Additional resources

- To learn how to use the AMQ Broker Operator to create a broker deployment, see [Chapter 3, Deploying AMQ Broker on OpenShift Container Platform using the AMQ Broker Operator](#).
- For more information about how the Operator uses an Init Container to generate the broker configuration, see [Section 4.1, “How the Operator generates the broker configuration”](#).
- To learn how to build and specify a *custom* Init Container image, see [Section 4.5, “Specifying a custom Init Container image”](#).

2.5. OPERATOR DEPLOYMENT NOTES

This section describes some important considerations when planning an Operator-based deployment

- Deploying the Custom Resource Definitions (CRDs) that accompany the AMQ Broker Operator requires cluster administrator privileges for your OpenShift cluster. When the Operator is deployed, non-administrator users can create broker instances via corresponding Custom Resources (CRs). To enable regular users to deploy CRs, the cluster administrator must first assign roles and permissions to the CRDs. For more information, see [Creating cluster roles for Custom Resource Definitions](#) in the OpenShift Container Platform documentation.
- When you update your cluster with the CRDs for the latest Operator version, this update affects **all** projects in the cluster. Any broker Pods deployed from previous versions of the Operator might become unable to update their status. When you click the Logs tab of a running broker Pod in the OpenShift Container Platform web console, you see messages indicating that 'UpdatePodStatus' has failed. However, the broker Pods and Operator in that project continue to work as expected. To fix this issue for an affected project, you must also upgrade that project to use the latest version of the Operator.
- You cannot create more than one broker deployment in a given OpenShift project by deploying multiple broker Custom Resource (CR) instances. However, when you have created a broker deployment in a project, you **can** deploy multiple CR instances for addresses.

- If you intend to deploy brokers with persistent storage and do not have container-native storage in your OpenShift cluster, you need to manually provision Persistent Volumes (PVs) and ensure that these are available to be claimed by the Operator. For example, if you want to create a cluster of two brokers with persistent storage (that is, by setting **persistenceEnabled=true** in your CR), you need to have two persistent volumes available. By default, each broker instance requires storage of 2 GiB.

If you specify **persistenceEnabled=false** in your CR, the deployed brokers uses *ephemeral* storage. Ephemeral storage means that every time you restart the broker Pods, any existing data is lost.

For more information about provisioning persistent storage in OpenShift Container Platform, see:

- [Understanding persistent storage](#) (OpenShift Container Platform 4.5)
 - [Persistent Storage](#) (OpenShift Container Platform 3.11).
- You must add configuration for the items listed below to the main broker CR instance **before** deploying the CR for the first time. You **cannot** add configuration for these items to a broker deployment that is already running.
 - [The size of the Persistent Volume Claim \(PVC\) required by each broker in a deployment for persistent storage](#)
 - [Limits and requests for memory and CPU for each broker in a deployment](#)

The procedures in the next section show you how to install the Operator and use Custom Resources (CRs) to create broker deployments on OpenShift Container Platform. When you have successfully completed the procedures, you will have the Operator running in an individual Pod. Each broker instance that you create will run as an individual Pod in a StatefulSet in the same project as the Operator. Later, you will see how to use a dedicated addressing CR to define addresses in your broker deployment.

CHAPTER 3. DEPLOYING AMQ BROKER ON OPENSIFT CONTAINER PLATFORM USING THE AMQ BROKER OPERATOR

3.1. PREREQUISITES

- Before you install the Operator and use it to create a broker deployment, you should consult the Operator deployment notes in [Section 2.5, “Operator deployment notes”](#).

3.2. INSTALLING THE OPERATOR USING THE CLI



NOTE

Each Operator release requires that you download the latest **AMQ Broker 7.8.5 .3 Operator Installation and Example Files** as described below.

The procedures in this section show how to use the OpenShift command-line interface (CLI) to install and deploy the latest version of the Operator for AMQ Broker 7.8 in a given OpenShift project. In subsequent procedures, you use this Operator to deploy some broker instances.

- For an alternative method of installing the AMQ Broker Operator that uses the OperatorHub graphical interface, see [Section 3.3, “Installing the Operator using OperatorHub”](#).
- To learn about *upgrading* existing Operator-based broker deployments, see [Chapter 6, *Upgrading an Operator-based broker deployment*](#).

3.2.1. Getting the Operator code

This procedure shows how to access and prepare the code you need to install the latest version of the Operator for AMQ Broker 7.8.

Procedure

1. In your web browser, navigate to the **Software Downloads** page for [AMQ Broker 7.8.5 patches](#).
2. Ensure that the value of the **Version** drop-down list is set to **7.8.5** and the **Patches** tab is selected.
3. Next to **AMQ Broker 7.8.5 .3 Operator Installation and Example Files** click **Download**. Download of the **amq-broker-operator-7.8.5-ocp-install-examples.zip** compressed archive automatically begins.
4. When the download has completed, move the archive to your chosen installation directory. The following example moves the archive to a directory called **~/broker/operator**.

```
$ mkdir ~/broker/operator
$ mv amq-broker-operator-7.8.5-ocp-install-examples.zip ~/broker/operator
```

5. In your chosen installation directory, extract the contents of the archive. For example:

```
$ cd ~/broker/operator  
$ unzip amq-broker-operator-7.8.5-ocp-install-examples.zip
```

- Switch to the directory that was created when you extracted the archive. For example:

```
$ cd amq-broker-operator-7.8.5-ocp-install-examples
```

- Log in to OpenShift Container Platform as a cluster administrator. For example:

```
$ oc login -u system:admin
```

- Specify the project in which you want to install the Operator. You can create a new project or switch to an existing one.

- Create a new project:

```
$ oc new-project <project-name>
```

- Or, switch to an existing project:

```
$ oc project <project-name>
```

- Specify a service account to use with the Operator.

- In the **deploy** directory of the Operator archive that you extracted, open the **service_account.yaml** file.
- Ensure that the **kind** element is set to **ServiceAccount**.
- In the **metadata** section, assign a custom name to the service account, or use the default name. The default name is **amq-broker-operator**.
- Create the service account in your project.

```
$ oc create -f deploy/service_account.yaml
```

- Specify a role name for the Operator.

- Open the **role.yaml** file. This file specifies the resources that the Operator can use and modify.
- Ensure that the **kind** element is set to **Role**.
- In the **metadata** section, assign a custom name to the role, or use the default name. The default name is **amq-broker-operator**.
- Create the role in your project.

```
$ oc create -f deploy/role.yaml
```

- Specify a role binding for the Operator. The role binding binds the previously-created service account to the Operator role, based on the names you specified.

- a. Open the **role_binding.yaml** file. Ensure that the **name** values for **ServiceAccount** and **Role** match those specified in the **service_account.yaml** and **role.yaml** files. For example:

```

metadata:
  name: amq-broker-operator
subjects:
  kind: ServiceAccount
  name: amq-broker-operator
roleRef:
  kind: Role
  name: amq-broker-operator

```

- b. Create the role binding in your project.

```
$ oc create -f deploy/role_binding.yaml
```

In the procedure that follows, you deploy the Operator in your project.

3.2.2. Deploying the Operator using the CLI

The procedure in this section shows how to use the OpenShift command-line interface (CLI) to deploy the latest version of the Operator for AMQ Broker 7.8 in your OpenShift project.

Prerequisites

- You must have already prepared your OpenShift project for the Operator deployment. See [Section 3.2.1, "Getting the Operator code"](#).
- Starting in AMQ Broker 7.3, you use a new version of the Red Hat Ecosystem Catalog to access container images. This new version of the registry requires you to become an authenticated user before you can access images. Before you can follow the procedure in this section, you must first complete the steps described in [Red Hat Container Registry Authentication](#).
- If you intend to deploy brokers with persistent storage and do not have container-native storage in your OpenShift cluster, you need to manually provision Persistent Volumes (PVs) and ensure that they are available to be claimed by the Operator. For example, if you want to create a cluster of two brokers with persistent storage (that is, by setting **persistenceEnabled=true** in your Custom Resource), you need to have two PVs available. By default, each broker instance requires storage of 2 GiB.

If you specify **persistenceEnabled=false** in your Custom Resource, the deployed brokers uses *ephemeral* storage. Ephemeral storage means that that every time you restart the broker Pods, any existing data is lost.

For more information about provisioning persistent storage, see:

- [Understanding persistent storage](#) (OpenShift Container Platform 4.5)
- [Persistent Storage](#) (OpenShift Container Platform 3.11)

Procedure

1. In the OpenShift command-line interface (CLI), log in to OpenShift as a cluster administrator. For example:

```
$ oc login -u system:admin
```

-
2. Switch to the project that you previously prepared for the Operator deployment. For example:

```
$ oc project <project_name>
```

3. Switch to the directory that was created when you previously extracted the Operator installation archive. For example:

```
$ cd ~/broker/operator/amq-broker-operator-7.8.5-ocp-install-examples
```

4. Deploy the CRDs that are included with the Operator. You must install the CRDs in your OpenShift cluster before deploying and starting the Operator.

- a. Deploy the main broker CRD.

```
$ oc create -f deploy/crds/broker_activemqartemis_crd.yaml
```

- b. Deploy the address CRD.

```
$ oc create -f deploy/crds/broker_activemqartemisaddress_crd.yaml
```

- c. Deploy the scaledown controller CRD.

```
$ oc create -f deploy/crds/broker_activemqartemisscaledown_crd.yaml
```

5. Link the pull secret associated with the account used for authentication in the Red Hat Ecosystem Catalog with the **default**, **deployer**, and **builder** service accounts for your OpenShift project.

```
$ oc secrets link --for=pull default <secret_name>
$ oc secrets link --for=pull deployer <secret_name>
$ oc secrets link --for=pull builder <secret_name>
```

6. In the **deploy** directory of the Operator archive that you downloaded and extracted, open the **operator.yaml** file.



NOTE

In the **operator.yaml** file, the Operator uses an image that is represented by a *Secure Hash Algorithm* (SHA) value. The comment line, which begins with a number sign (**#**) symbol, denotes that the SHA value corresponds to a specific container image tag.

7. Deploy the Operator.

```
$ oc create -f deploy/operator.yaml
```

In your OpenShift project, the Operator starts in a new Pod.

In the OpenShift Container Platform web console, the information on the **Events** tab of the Operator Pod confirms that OpenShift has deployed the Operator image that you specified, has assigned a new container to a node in your OpenShift cluster, and has started the new container.

In addition, if you click the **Logs** tab within the Pod, the output should include lines resembling the following:

```
...
{"level":"info","ts":1553619035.8302743,"logger":"kubebuilder.controller","msg":"Starting Controller","controller":"activemqartemisaddress-controller"}
{"level":"info","ts":1553619035.830541,"logger":"kubebuilder.controller","msg":"Starting Controller","controller":"activemqartemis-controller"}
{"level":"info","ts":1553619035.9306898,"logger":"kubebuilder.controller","msg":"Starting workers","controller":"activemqartemisaddress-controller","worker count":1}
{"level":"info","ts":1553619035.9311671,"logger":"kubebuilder.controller","msg":"Starting workers","controller":"activemqartemis-controller","worker count":1}
```

The preceding output confirms that the newly-deployed Operator is communicating with Kubernetes, that the controllers for the broker and addressing are running, and that these controllers have started some workers.



NOTE

It is recommended that you deploy only a **single instance** of the AMQ Broker Operator in a given OpenShift project. Setting the **spec.replicas** property of your Operator deployment to a value greater than **1**, or deploying the Operator more than once in the same project is **not** recommended.

Additional resources

- For an alternative method of installing the AMQ Broker Operator that uses the OperatorHub graphical interface, see [Section 3.3, “Installing the Operator using OperatorHub”](#).

3.3. INSTALLING THE OPERATOR USING OPERATORHUB

3.3.1. Overview of the Operator Lifecycle Manager

In OpenShift Container Platform 4.5 and later, the *Operator Lifecycle Manager* (OLM) helps users install, update, and generally manage the lifecycle of all Operators and their associated services running across their clusters. It is part of the Operator Framework, an open source toolkit designed to manage Kubernetes-native applications (Operators) in an effective, automated, and scalable way.

The OLM runs by default in OpenShift Container Platform 4.5 and later, which aids cluster administrators in installing, upgrading, and granting access to Operators running on their cluster. The OpenShift Container Platform web console provides management screens for cluster administrators to install Operators, as well as grant specific projects access to use the catalog of Operators available on the cluster.

OperatorHub is the graphical interface that OpenShift cluster administrators use to discover, install, and upgrade Operators using the OLM. With one click, these Operators can be pulled from OperatorHub, installed on the cluster, and managed by the OLM, ready for engineering teams to self-service manage the software in development, test, and production environments.

When you have deployed the Operator, you can use Custom Resource (CR) instances to create broker deployments such as standalone and clustered brokers.

3.3.2. Installing the Operator in OperatorHub

In OperatorHub, the name of the Operator for AMQ Broker 7.8 is **Red Hat Integration - AMQ Broker**. You should see the Operator automatically available in OperatorHub. However, if you do not see it, follow this procedure to manually install the Operator in OperatorHub.



NOTE

This section describes how to install the RHEL 7 Operator. There is also an Operator for RHEL 8 that provides RHEL 8 images.

To determine which Operator to choose, see the [Red Hat Enterprise Linux Container Compatibility Matrix](#).

Procedure

1. In your web browser, navigate to the **Software Downloads** page for [AMQ Broker 7.8.5 releases](#).
2. Ensure that the value of the **Version** drop-down list is set to **7.8.5** and the **Releases** tab is selected.
3. Next to **AMQ Broker 7.8.5 .3 Operator Installation and Example Files** click **Download**. Download of the **amq-broker-operator-7.8.5-ocp-install-examples.zip** compressed archive automatically begins.
4. When the download has completed, move the archive to your chosen installation directory. The following example moves the archive to a directory called **~/broker/operator**.

```
mkdir ~/broker/operator
mv amq-broker-operator-7.8.5-ocp-install-examples.zip ~/broker/operator
```

5. In your chosen installation directory, extract the contents of the archive. For example:

```
cd ~/broker/operator
unzip amq-broker-operator-7.8.5-ocp-install-examples.zip
```

6. Switch to the directory for the Operator archive that you extracted. For example:

```
cd amq-broker-operator-7.8.5-ocp-install-examples
```

7. Log in to OpenShift Container Platform as a cluster administrator. For example:

```
$ oc login -u system:admin
```

8. Install the Operator in OperatorHub.

```
$ oc create -f deploy/catalog_resources/activemq-artemis-operatorsourcesource.yaml
```

After a few minutes, the Operator for AMQ Broker 7.8 is available in the **OperatorHub** section of the OpenShift Container Platform web console. The name of the Operator is **Red Hat Integration - AMQ Broker**.

3.3.3. Deploying the Operator from OperatorHub

This procedure shows how to use OperatorHub to deploy the latest version of the Operator for AMQ Broker to a specified OpenShift project.



IMPORTANT

Deploying the Operator using OperatorHub requires cluster administrator privileges.

Prerequisites

- The **Red Hat Integration - AMQ Broker** Operator must be available in OperatorHub. If you do not see the Operator automatically available, see [Section 3.3.2, “Installing the Operator in OperatorHub”](#) for instructions on manually installing the Operator in OperatorHub.

Procedure

1. Log in to the OpenShift Container Platform web console as a cluster administrator.
2. In left navigation menu, click **Operators** → **OperatorHub**.
3. On the **Project** drop-down menu at the top of the **OperatorHub** page, select the project in which you want to deploy the Operator.
4. On the **OperatorHub** page, use the **Filter by keyword...** box to find the **Red Hat Integration - AMQ Broker** Operator.



NOTE

In OperatorHub, you might find more than one Operator than includes **AMQ Broker** in its name. Ensure that you click the **Red Hat Integration - AMQ Broker** Operator. When you click this Operator, review the information pane that opens. For AMQ Broker 7.8, the latest minor version tag of this Operator is **7.8.5-opr-2**.

The Operator for RHEL 8 that provides RHEL 8 images is named **Red Hat Integration - AMQ Broker for RHEL 8** and has the version **7.8.5-opr-2**.

To determine which Operator to choose, see the [Red Hat Enterprise Linux Container Compatibility Matrix](#).

5. Click the **Red Hat Integration - AMQ Broker** Operator. On the dialog box that appears, click **Install**.
6. On the **Install Operator** page:
 - a. Under **Update Channel**, specify the channel used to track and receive updates for the Operator by selecting one of the following radio buttons:
 - **7.x** - This channel will update to **7.9** when available.
 - **7.8.x** - This is the Long Term Support (LTS) channel.
 - b. Under **Installation Mode**, ensure that the radio button entitled **A specific namespace on the cluster** is selected.
 - c. From the **Installed Namespace** drop-down menu, select the project in which you want to install the Operator.

- d. Under **Approval Strategy**, ensure that the radio button entitled **Automatic** is selected. This option specifies that updates to the Operator do not require manual approval for installation to take place.
- e. Click **Install**.

When the Operator installation is complete, the **Installed Operators** page opens. You should see that the **Red Hat Integration - AMQ Broker** Operator is installed in the project namespace that you specified.

Additional resources

- To learn how to create a broker deployment in a project that has the Operator for AMQ Broker installed, see [Section 3.4.1, “Deploying a basic broker instance”](#).

3.4. CREATING OPERATOR-BASED BROKER DEPLOYMENTS

3.4.1. Deploying a basic broker instance

The following procedure shows how to use a Custom Resource (CR) instance to create a basic broker deployment.



NOTE

- You cannot create more than one broker deployment in a given OpenShift project by deploying multiple Custom Resource (CR) instances. However, when you have created a broker deployment in a project, you **can** deploy multiple CR instances for addresses.
- In AMQ Broker 7.8, if you want to configure the following items, you must add the appropriate configuration to the main broker CR instance **before** deploying the CR for the first time.
 - [The size of the Persistent Volume Claim \(PVC\) required by each broker in a deployment for persistent storage](#)
 - [Limits and requests for memory and CPU for each broker in a deployment](#)

Prerequisites

- You must have already installed the AMQ Broker Operator.
 - To use the OpenShift command-line interface (CLI) to install the AMQ Broker Operator, see [Section 3.2, “Installing the Operator using the CLI”](#).
 - To use the OperatorHub graphical interface to install the AMQ Broker Operator, see [Section 3.3, “Installing the Operator using OperatorHub”](#).
- You should understand how the Operator chooses a broker container image to use for your broker deployment. For more information, see [Section 2.4, “How the Operator chooses container images”](#).
- Starting in AMQ Broker 7.3, you use a new version of the Red Hat Ecosystem Catalog to access container images. This new version of the registry requires you to become an authenticated user before you can access images. Before you can follow the procedure in this section, you must

first complete the steps described in [Red Hat Container Registry Authentication](#) .

Procedure

When you have successfully installed the Operator, the Operator is running and listening for changes related to your CRs. This example procedure shows how to use a CR instance to deploy a basic broker in your project.

1. Start configuring a Custom Resource (CR) instance for the broker deployment.
 - a. Using the OpenShift command-line interface:
 - i. Log in to OpenShift as a user that has privileges to deploy CRs in the project in which you are creating the deployment.


```
oc login -u <user> -p <password> --server=<host:port>
```
 - ii. Open the sample CR file called **broker_activemqartemis_cr.yaml** that was included in the **deploy/crs** directory of the Operator installation archive that you downloaded and extracted.
 - b. Using the OpenShift Container Platform web console:
 - i. Log in to the console as a user that has privileges to deploy CRs in the project in which you are creating the deployment.
 - ii. Start a new CR instance based on the main broker CRD. In the left pane, click **Administration** → **Custom Resource Definitions**
 - iii. Click the **ActiveMQArtemis** CRD.
 - iv. Click the **Instances** tab.
 - v. Click **Create ActiveMQArtemis**.
Within the console, a YAML editor opens, enabling you to configure a CR instance.

For a basic broker deployment, a configuration might resemble that shown below. This configuration is the default content of the **broker_activemqartemis_cr.yaml** sample CR file.

```
apiVersion: broker.amq.io/v2alpha4
kind: ActiveMQArtemis
metadata:
  name: ex-aa0
  application: ex-aa0-app
spec:
  version: 7.8.5
  deploymentPlan:
    size: 1
    image: placeholder
    requireLogin: false
    persistenceEnabled: true
    journalType: nio
    messageMigration: true
```

Observe that in the **broker_activemqartemis_cr.yaml** sample CR file, the **image** property is set to a default value of **placeholder**. This value indicates that, by default, the **image** property does not specify a broker container image to use for the deployment. To learn how the

Operator determines the appropriate broker container image to use, see [Section 2.4, “How the Operator chooses container images”](#).



NOTE

In the **metadata** section, you need to include the **namespace** property and specify a value **only** if you are using the OpenShift Container Platform web console to create your CR instance. The value that you should specify is the name of the OpenShift project for your broker deployment.



NOTE

The **broker_activemqartemis_cr.yaml** sample CR uses a naming convention of **ex-aa0**. This naming convention denotes that the CR is an **example** resource for the AMQ Broker **Operator**. AMQ Broker is based on the **ActiveMQ Artemis** project. When you deploy this sample CR, the resulting StatefulSet uses the name **ex-aa0-ss**. Furthermore, broker Pods in the deployment are directly based on the StatefulSet name, for example, **ex-aa0-ss-0**, **ex-aa0-ss-1**, and so on. The application name in the CR appears in the deployment as a label on the StatefulSet. You might use this label in a Pod selector, for example.

2. The **size** property specifies the number of brokers to deploy. A value of **2** or greater specifies a clustered broker deployment. However, to deploy a single broker instance, ensure that the value is set to **1**.
3. Deploy the CR instance.
 - a. Using the OpenShift command-line interface:
 - i. Save the CR file.
 - ii. Switch to the project in which you are creating the broker deployment.


```
$ oc project <project_name>
```
 - iii. Create the CR instance.


```
$ oc create -f <path/to/custom_resource_instance>.yaml
```
 - b. Using the OpenShift web console:
 - i. When you have finished configuring the CR, click **Create**.
4. In the OpenShift Container Platform web console, click **Workloads** → **StatefulSets** (OpenShift Container Platform 4.5 or later) or **Applications** → **StatefulSets** (OpenShift Container Platform 3.11). You see a new StatefulSet called **ex-aa0-ss**.
 - a. Click the **ex-aa0-ss** StatefulSet. You see that there is one Pod, corresponding to the single broker that you defined in the CR.
 - b. Within the StatefulSet, click the **Pods** tab. Click the **ex-aa0-ss** Pod. On the **Events** tab of the running Pod, you see that the broker container has started. The **Logs** tab shows that the broker itself is running.

5. To test that the broker is running normally, access a shell on the broker Pod to send some test messages.
 - a. Using the OpenShift Container Platform web console:
 - i. Click **Workloads** → **Pods** (OpenShift Container Platform 4.5 or later) or **Applications** → **Pods** (OpenShift Container Platform 3.11).
 - ii. Click the **ex-aao-ss** Pod.
 - iii. Click the **Terminal** tab.
 - b. Using the OpenShift command-line interface:
 - i. Get the Pod names and internal IP addresses for your project.

```
$ oc get pods -o wide
```

NAME	STATUS	IP
amq-broker-operator-54d996c	Running	10.129.2.14
ex-aao-ss-0	Running	10.129.2.15

- ii. Access the shell for the broker Pod.

```
$ oc rsh ex-aao-ss-0
```

6. From the shell, use the **artemis** command to send some test messages. Specify the internal IP address of the broker Pod in the URL. For example:

```
sh-4.2$ ./amq-broker/bin/artemis producer --url tcp://10.129.2.15:61616 --destination queue://demoQueue
```

The preceding command automatically creates a queue called **demoQueue** on the broker and sends a default quantity of 1000 messages to the queue.

You should see output that resembles the following:

```
Connection brokerURL = tcp://10.129.2.15:61616
Producer ActiveMQQueue[demoQueue], thread=0 Started to calculate elapsed time ...

Producer ActiveMQQueue[demoQueue], thread=0 Produced: 1000 messages
Producer ActiveMQQueue[demoQueue], thread=0 Elapsed time in second : 3 s
Producer ActiveMQQueue[demoQueue], thread=0 Elapsed time in milli second : 3492 milli
seconds
```

Additional resources

- For a complete configuration reference for the main broker Custom Resource (CR), see [Section 11.1, “Custom Resource configuration reference”](#).
- To learn how to connect a running broker to AMQ Management Console, see [Chapter 5, Connecting to AMQ Management Console for an Operator-based broker deployment](#).

3.4.2. Deploying clustered brokers

If there are two or more broker Pods running in your project, the Pods automatically form a broker cluster. A clustered configuration enables brokers to connect to each other and redistribute messages as needed, for load balancing.

The following procedure shows you how to deploy clustered brokers. By default, the brokers in this deployment use *on demand* load balancing, meaning that brokers will forward messages only to other brokers that have matching consumers.

Prerequisites

- A basic broker instance is already deployed. See [Section 3.4.1, “Deploying a basic broker instance”](#).

Procedure

1. Open the CR file that you used for your basic broker deployment.
2. For a clustered deployment, ensure that the value of **deploymentPlan.size** is **2** or greater. For example:

```
apiVersion: broker.amq.io/v2alpha4
kind: ActiveMQArtemis
metadata:
  name: ex-aa0
  application: ex-aa0-app
spec:
  version: 7.8.5
  deploymentPlan:
    size: 4
    image: placeholder
  ...
```



NOTE

In the **metadata** section, you need to include the **namespace** property and specify a value **only** if you are using the OpenShift Container Platform web console to create your CR instance. The value that you should specify is the name of the OpenShift project for your broker deployment.

3. Save the modified CR file.
4. Log in to OpenShift as a user that has privileges to deploy CRs in the project in which you previously created your basic broker deployment.

```
$ oc login -u <user> -p <password> --server=<host:port>
```

5. Switch to the project in which you previously created your basic broker deployment.

```
$ oc project <project_name>
```

6. At the command line, apply the change:

```
$ oc apply -f <path/to/custom_resource_instance>.yaml
```

In the OpenShift Container Platform web console, additional broker Pods starts in your project, according to the number specified in your CR. By default, the brokers running in the project are clustered.

7. Open the **Logs** tab of each Pod. The logs show that OpenShift has established a cluster connection bridge on each broker. Specifically, the log output includes a line like the following:

```
targetConnector=ServerLocatorImpl (identity=(Cluster-connection-bridge::ClusterConnectionBridge@6f13fb88
```

3.4.3. Applying Custom Resource changes to running broker deployments

The following are some important things to note about applying Custom Resource (CR) changes to running broker deployments:

- You cannot dynamically update the **persistenceEnabled** attribute in your CR. To change this attribute, scale your cluster down to zero brokers. Delete the existing CR. Then, recreate and redeploy the CR with your changes, also specifying a deployment size.
- The value of the **deploymentPlan.size** attribute in your CR overrides any change you make to size of your broker deployment via the **oc scale** command. For example, suppose you use **oc scale** to change the size of a deployment from three brokers to two, but the value of **deploymentPlan.size** in your CR is still **3**. In this case, OpenShift initially scales the deployment down to two brokers. However, when the scaledown operation is complete, the Operator restores the deployment to three brokers, as specified in the CR.
- As described in [Section 3.2.2, "Deploying the Operator using the CLI"](#), if you create a broker deployment with persistent storage (that is, by setting **persistenceEnabled=true** in your CR), you might need to provision Persistent Volumes (PVs) for the AMQ Broker Operator to claim for your broker Pods. If you scale down the size of your broker deployment, the Operator releases any PVs that it previously claimed for the broker Pods that are now shut down. However, if you *remove* your broker deployment by deleting your CR, AMQ Broker Operator **does not** release Persistent Volume Claims (PVCs) for any broker Pods that are still in the deployment when you remove it. In addition, these unreleased PVs are unavailable to any new deployment. In this case, you need to manually release the volumes. For more information, see [Release a persistent volume](#) in the OpenShift documentation.
- In AMQ Broker 7.8, if you want to configure the following items, you must add the appropriate configuration to the main CR instance **before** deploying the CR for the first time.
 - [The size of the Persistent Volume Claim \(PVC\) required by each broker in a deployment for persistent storage](#)
 - [Limits and requests for memory and CPU for each broker in a deployment](#)
- During an active scaling event, any further changes that you apply are queued by the Operator and executed only when scaling is complete. For example, suppose that you scale the size of your deployment down from four brokers to one. Then, while scaledown is taking place, you also change the values of the broker administrator user name and password. In this case, the Operator queues the user name and password changes until the deployment is running with one active broker.
- All CR changes – apart from changing the size of your deployment, or changing the value of the **expose** attribute for acceptors, connectors, or the console – cause existing brokers to be restarted. If you have multiple brokers in your deployment, only one broker restarts at a time.

CHAPTER 4. CONFIGURING OPERATOR-BASED BROKER DEPLOYMENTS

4.1. HOW THE OPERATOR GENERATES THE BROKER CONFIGURATION

Before you use Custom Resource (CR) instances to configure your broker deployment, you should understand how the Operator generates the broker configuration.

When you create an Operator-based broker deployment, a Pod for each broker runs in a StatefulSet in your OpenShift project. An application container for the broker runs within each Pod.

The Operator runs a type of container called an *Init Container* when initializing each Pod. In OpenShift Container Platform, Init Containers are specialized containers that run before application containers. Init Containers can include utilities or setup scripts that are not present in the application image.

By default, the AMQ Broker Operator uses a built-in Init Container. The Init Container uses the main CR instance for your deployment to generate the configuration used by each broker application container.

If you have specified address settings in the CR, the Operator generates a default configuration and then merges or replaces that configuration with the configuration specified in the CR. This process is described in the section that follows.

4.1.1. How the Operator generates the address settings configuration

If you have included an address settings configuration in the main Custom Resource (CR) instance for your deployment, the Operator generates the address settings configuration for each broker as described below.

1. The Operator runs the Init Container before the broker application container. The Init Container generates a **default** address settings configuration. The default address settings configuration is shown below.

```
<address-settings>
  <!--
  if you define auto-create on certain queues, management has to be auto-create
  -->
  <address-setting match="activemq.management#">
    <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="#">
  <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>

```

2. If you have also specified an address settings configuration in your Custom Resource (CR) instance, the Init Container processes that configuration and converts it to XML.
3. Based on the value of the **applyRule** property in the CR, the Init Container *merges* or *replaces* the default address settings configuration shown above with the configuration that you have specified in the CR. The result of this merge or replacement is the final address settings configuration that the broker will use.
4. When the Init Container has finished generating the broker configuration (including address settings), the broker application container starts. When starting, the broker container copies its configuration from the installation directory previously used by the Init Container. You can inspect the address settings configuration in the **broker.xml** configuration file. For a running broker, this file is located in the **/home/jboss/amq-broker/etc** directory.

Additional resources

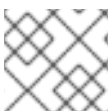
- For an example of using the **applyRule** property in a CR, see [Section 4.2.3, "Matching address settings to configured addresses in an Operator-based broker deployment"](#).

4.1.2. Directory structure of a broker Pod

When you create an Operator-based broker deployment, a Pod for each broker runs in a StatefulSet in your OpenShift project. An application container for the broker runs within each Pod.

The Operator runs a type of container called an *Init Container* when initializing each Pod. In OpenShift Container Platform, Init Containers are specialized containers that run before application containers. Init Containers can include utilities or setup scripts that are not present in the application image.

When generating the configuration for a broker instance, the Init Container uses files contained in a default installation directory. This installation directory is on a volume that the Operator mounts to the broker Pod and which the Init Container and broker container share. The path that the Init Container uses to mount the shared volume is defined in an environment variable called **CONFIG_INSTANCE_DIR**. The default value of **CONFIG_INSTANCE_DIR** is **/amq/init/config**. In the documentation, this directory is referred to as **<install_dir>**.



NOTE

You cannot change the value of the **CONFIG_INSTANCE_DIR** environment variable.

By default, the installation directory has the following sub-directories:

Sub-directory	Contents
<code><install_dir>/bin</code>	Binaries and scripts needed to run the broker.
<code><install_dir>/etc</code>	Configuration files.
<code><install_dir>/data</code>	The broker journal.
<code><install_dir>/lib</code>	JARs and libraries needed to run the broker.
<code><install_dir>/log</code>	Broker log files.
<code><install_dir>/tmp</code>	Temporary web application files.

When the Init Container has finished generating the broker configuration, the broker application container starts. When starting, the broker container copies its configuration from the installation directory previously used by the Init Container. When the broker Pod is initialized and running, the broker configuration is located in the `/home/jboss/amq-broker` directory (and subdirectories) of the broker.

Additional resources

- For more information about how the Operator chooses a container image for the built-in Init Container, see [Section 2.4, “How the Operator chooses container images”](#).
- To learn how to build and specify a custom Init Container image, see [Section 4.5, “Specifying a custom Init Container image”](#).

4.2. CONFIGURING ADDRESSES AND QUEUES FOR OPERATOR-BASED BROKER DEPLOYMENTS

For an Operator-based broker deployment, you use two separate Custom Resource (CR) instances to configure address and queues and their associated settings.

- To create address and queues on your brokers, you deploy a CR instance based on the address Custom Resource Definition (CRD).
 - If you used the OpenShift command-line interface (CLI) to install the Operator, the address CRD is the `broker_activemqartemisaddress_crd.yaml` file that was included in the `deploy/crds` of the Operator installation archive that you downloaded and extracted.
 - If you used OperatorHub to install the Operator, the address CRD is the **ActiveMQArtemisAddress** CRD listed under **Administration** → **Custom Resource Definitions** in the OpenShift Container Platform web console.
- To configure address and queue settings that you then match to specific addresses, you include configuration in the main Custom Resource (CR) instance used to create your broker deployment.

- If you used the OpenShift CLI to install the Operator, the main broker CRD is the **broker_activemqartemis_crd.yaml** file that was included in the **deploy/crds** of the Operator installation archive that you downloaded and extracted.
- If you used OperatorHub to install the Operator, the main broker CRD is the **ActiveMQartemis** CRD listed under **Administration** → **Custom Resource Definitions** in the OpenShift Container Platform web console.



NOTE

To configure address settings for an Operator-based deployment, you must be using *at least* the latest version of the Operator for AMQ Broker 7.7 (that is, version 0.17). To learn how to upgrade the Operator to the latest version for AMQ Broker 7.8, see [Chapter 6, Upgrading an Operator-based broker deployment](#).

In general, the address and queue settings that you can configure for a broker deployment on OpenShift Container Platform are **fully equivalent** to those of standalone broker deployments on Linux or Windows. However, you should be aware of some differences in *how* those settings are configured. Those differences are described in the following sub-section.

4.2.1. Differences in configuration of address and queue settings between OpenShift and standalone broker deployments

- To configure address and queue settings for broker deployments on OpenShift Container Platform, you add configuration to an **addressSettings** section of the main Custom Resource (CR) instance for the broker deployment. This contrasts with standalone deployments on Linux or Windows, for which you add configuration to an **address-settings** element in the **broker.xml** configuration file.
- The format used for the names of configuration items differs between OpenShift Container Platform and standalone broker deployments. For OpenShift Container Platform deployments, configuration item names are in *camel case*, for example, **defaultQueueRoutingType**. By contrast, configuration item names for standalone deployments are in lower case and use a dash (-) separator, for example, **default-queue-routing-type**.

The following table shows some further examples of this naming difference.

Configuration item for standalone broker deployment	Configuration item for OpenShift broker deployment
address-full-policy	addressFullPolicy
auto-create-queues	autoCreateQueues
default-queue-routing-type	defaultQueueRoutingType
last-value-queue	lastValueQueue

Additional resources

- For examples of creating addresses and queues and matching settings for OpenShift Container Platform broker deployments, see:

- [Creating addresses and queues for a broker deployment on OpenShift Container Platform](#)
- [Matching address settings to configured addresses for a broker deployment on OpenShift Container Platform](#)
- To learn about all of the configuration options for addresses, queues, and address settings for OpenShift Container Platform broker deployments, see [Section 11.1, “Custom Resource configuration reference”](#).
- For comprehensive information about configuring addresses, queues, and associated address settings for **standalone** broker deployments, see [Addresses, Queues, and Topics](#) in *Configuring AMQ Broker*. You can use this information to create equivalent configurations for broker deployments on OpenShift Container Platform.

4.2.2. Creating addresses and queues for an Operator-based broker deployment

The following procedure shows how to use a Custom Resource (CR) instance to add an address and associated queue to an Operator-based broker deployment.



NOTE

To create multiple addresses and/or queues in your broker deployment, you need to create separate CR files and deploy them individually, specifying new address and/or queue names in each case. In addition, the **name** attribute of each CR instance must be unique.

Prerequisites

- You must have already installed the AMQ Broker Operator, including the dedicated Custom Resource Definition (CRD) required to create addresses and queues on your brokers. For information on two alternative ways to install the Operator, see:
 - [Section 3.2, “Installing the Operator using the CLI”](#).
 - [Section 3.3, “Installing the Operator using OperatorHub”](#).
- You should be familiar with how to use a CR instance to create a basic broker deployment. For more information, see [Section 3.4.1, “Deploying a basic broker instance”](#).

Procedure

1. Start configuring a Custom Resource (CR) instance to define addresses and queues for the broker deployment.
 - a. Using the OpenShift command-line interface:
 - i. Log in to OpenShift as a user that has privileges to deploy CRs in the project for the broker deployment.

```
oc login -u <user> -p <password> --server=<host:port>
```

- ii. Open the sample CR file called **broker_activemqartemisaddress_cr.yaml** that was included in the **deploy/crs** directory of the Operator installation archive that you downloaded and extracted.
 - b. Using the OpenShift Container Platform web console:

- i. Log in to the console as a user that has privileges to deploy CRs in the project for the broker deployment.
 - ii. Start a new CR instance based on the address CRD. In the left pane, click **Administration** → **Custom Resource Definitions**
 - iii. Click the **ActiveMQArtemisAddresss** CRD.
 - iv. Click the **Instances** tab.
 - v. Click **Create ActiveMQArtemisAddress**
Within the console, a YAML editor opens, enabling you to configure a CR instance.
2. In the **spec** section of the CR, add lines to define an address, queue, and routing type. For example:

```

apiVersion: broker.amq.io/v2alpha2
kind: ActiveMQArtemisAddress
metadata:
  name: myAddressDeployment0
  namespace: myProject
spec:
  ...
  addressName: myAddress0
  queueName: myQueue0
  routingType: anycast
  ...

```

The preceding configuration defines an address named **myAddress0** with a queue named **myQueue0** and an **anycast** routing type.



NOTE

In the **metadata** section, you need to include the **namespace** property and specify a value **only** if you are using the OpenShift Container Platform web console to create your CR instance. The value that you should specify is the name of the OpenShift project for your broker deployment.

3. Deploy the CR instance.
 - a. Using the OpenShift command-line interface:
 - i. Save the CR file.
 - ii. Switch to the project for the broker deployment.

```
$ oc project <project_name>
```

- iii. Create the CR instance.

```
$ oc create -f <path/to/address_custom_resource_instance>.yaml
```

- b. Using the OpenShift web console:
 - i. When you have finished configuring the CR, click **Create**.

4. (Optional) To delete an address and queue previously added to your deployment using a CR instance, use the following command:

```
$ oc delete -f <path/to/address_custom_resource_instance>.yaml
```

4.2.3. Matching address settings to configured addresses in an Operator-based broker deployment

If delivery of a message to a client is unsuccessful, you might not want the broker to make ongoing attempts to deliver the message. To prevent infinite delivery attempts, you can define a *dead letter address* and an associated *dead letter queue*. After a specified number of delivery attempts, the broker removes an undelivered message from its original queue and sends the message to the configured dead letter address. A system administrator can later consume undelivered messages from a dead letter queue to inspect the messages.

The following example shows how to configure a dead letter address and queue for an Operator-based broker deployment. The example demonstrates how to:

- Use the **addressSetting** section of the main broker Custom Resource (CR) instance to configure address settings.
- Match those address settings to addresses in your broker deployment.

Prerequisites

- You must be using the latest version of the Operator for AMQ Broker 7.8 (that is, version 7.8.5-opr-2). To learn how to upgrade the Operator to the latest version, see [Chapter 6, Upgrading an Operator-based broker deployment](#).
- You should be familiar with how to use a CR instance to create a basic broker deployment. For more information, see [Section 3.4.1, “Deploying a basic broker instance”](#).
- You should be familiar with the **default** address settings configuration that the Operator merges or replaces with the configuration specified in your CR instance. For more information, see [Section 4.1.1, “How the Operator generates the address settings configuration”](#).

Procedure

1. Start configuring a CR instance to add a dead letter address and queue to receive undelivered messages for each broker in the deployment.
 - a. Using the OpenShift command-line interface:
 - i. Log in to OpenShift as a user that has privileges to deploy CRs in the project for the broker deployment.

```
oc login -u <user> -p <password> --server=<host:port>
```

- ii. Open the sample CR file called **broker_activemqartemisaddress_cr.yaml** that was included in the **deploy/crs** directory of the Operator installation archive that you downloaded and extracted.
 - b. Using the OpenShift Container Platform web console:

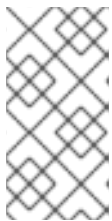
- i. Log in to the console as a user that has privileges to deploy CRs in the project for the broker deployment.
 - ii. Start a new CR instance based on the address CRD. In the left pane, click **Administration → Custom Resource Definitions**
 - iii. Click the **ActiveMQArtemisAddresss** CRD.
 - iv. Click the **Instances** tab.
 - v. Click **Create ActiveMQArtemisAddress**.
Within the console, a YAML editor opens, enabling you to configure a CR instance.
2. In the **spec** section of the CR, add lines to specify a dead letter address and queue to receive undelivered messages. For example:

```

apiVersion: broker.amq.io/v2alpha2
kind: ActiveMQArtemisAddress
metadata:
  name: ex-aaaddress
spec:
  ...
  addressName: myDeadLetterAddress
  queueName: myDeadLetterQueue
  routingType: anycast
  ...

```

The preceding configuration defines a dead letter address named **myDeadLetterAddress** with a dead letter queue named **myDeadLetterQueue** and an **anycast** routing type.



NOTE

In the **metadata** section, you need to include the **namespace** property and specify a value **only** if you are using the OpenShift Container Platform web console to create your CR instance. The value that you should specify is the name of the OpenShift project for your broker deployment.

3. Deploy the address CR instance.
 - a. Using the OpenShift command-line interface:
 - i. Save the CR file.
 - ii. Switch to the project for the broker deployment.

```
$ oc project <project_name>
```

- iii. Create the address CR.

```
$ oc create -f <path/to/address_custom_resource_instance>.yaml
```

- b. Using the OpenShift web console:
 - i. When you have finished configuring the CR, click **Create**.

4. Start configuring a Custom Resource (CR) instance for a broker deployment.
 - a. From a sample CR file:
 - i. Open the sample CR file called **broker_activemqartemis_cr.yaml** that was included in the **deploy/crs** directory of the Operator installation archive that you downloaded and extracted.
 - b. Using the OpenShift Container Platform web console:
 - i. Start a new CR instance based on the main broker CRD. In the left pane, click **Administration** → **Custom Resource Definitions**
 - ii. Click the **ActiveMQArtemis** CRD.
 - iii. Click the **Instances** tab.
 - iv. Click **Create ActiveMQArtemis**.
Within the console, a YAML editor opens, enabling you to configure a CR instance.

For a basic broker deployment, a configuration might resemble that shown below. This configuration is the default content of the **broker_activemqartemis_cr.yaml** sample CR file.

```
apiVersion: broker.amq.io/v2alpha4
kind: ActiveMQArtemis
metadata:
  name: ex-aa0
  application: ex-aa0-app
spec:
  version: 7.8.5
  deploymentPlan:
    size: 1
    image: placeholder
    requireLogin: false
    persistenceEnabled: true
    journalType: nio
    messageMigration: true
```

Observe that in the **broker_activemqartemis_cr.yaml** sample CR file, the **image** property is set to a default value of **placeholder**. This value indicates that, by default, the **image** property does not specify a broker container image to use for the deployment. To learn how the Operator determines the appropriate broker container image to use, see [Section 2.4, “How the Operator chooses container images”](#).



NOTE

In the **metadata** section, you need to include the **namespace** property and specify a value **only** if you are using the OpenShift Container Platform web console to create your CR instance. The value that you should specify is the name of the OpenShift project for your broker deployment.

5. In the **deploymentPlan** section of the CR, add a new **addressSettings** section that contains a single **addressSetting** section, as shown below.

```
spec:
  version: 7.8.5
```

```

deploymentPlan:
  size: 1
  image: placeholder
  requireLogin: false
  persistenceEnabled: true
  journalType: nio
  messageMigration: true
  addressSettings:
    addressSetting:

```

6. Add a single instance of the **match** property to the **addressSetting** block. Specify an address-matching expression. For example:

```

spec:
  version: 7.8.5
  deploymentPlan:
    size: 1
    image: placeholder
    requireLogin: false
    persistenceEnabled: true
    journalType: nio
    messageMigration: true
    addressSettings:
      addressSetting:
        - match: myAddress

```

match

Specifies the address, or set of address to which the broker applies the configuration that follows. In this example, the value of the **match** property corresponds to a single address called **myAddress**.

7. Add properties related to undelivered messages and specify values. For example:

```

spec:
  version: 7.8.5
  deploymentPlan:
    size: 1
    image: placeholder
    requireLogin: false
    persistenceEnabled: true
    journalType: nio
    messageMigration: true
    addressSettings:
      addressSetting:
        - match: myAddress
        deadLetterAddress: myDeadLetterAddress
        maxDeliveryAttempts: 5

```

deadLetterAddress

Address to which the broker sends undelivered messages.

maxDeliveryAttempts

Maximum number of delivery attempts that a broker makes before moving a message to the configured dead letter address.

In the preceding example, if the broker makes five unsuccessful attempts to deliver a message to an address that begins with **myAddress**, the broker moves the message to the specified dead letter address, **myDeadLetterAddress**.

8. (Optional) Apply similar configuration to another address or set of addresses. For example:

```
spec:
  version: 7.8.5
  deploymentPlan:
    size: 1
    image: placeholder
    requireLogin: false
    persistenceEnabled: true
    journalType: nio
    messageMigration: true
    addressSettings:
      addressSetting:
        - match: myAddress
          deadLetterAddress: myDeadLetterAddress
          maxDeliveryAttempts: 5
        - match: 'myOtherAddresses*'
          deadLetterAddress: myDeadLetterAddress
          maxDeliveryAttempts: 3
```

In this example, the value of the second **match** property includes an asterisk wildcard character. The wildcard character means that the preceding configuration is applied to **any** address that begins with the string **myOtherAddresses**.



NOTE

If you use a wildcard expression as a value for the **match** property, you must enclose the value in single quotation marks, for example, **'myOtherAddresses*'**.

9. At the beginning of the **addressSettings** section, add the **applyRule** property and specify a value. For example:

```
spec:
  version: 7.8.5
  deploymentPlan:
    size: 1
    image: placeholder
    requireLogin: false
    persistenceEnabled: true
    journalType: nio
    messageMigration: true
    addressSettings:
      applyRule: merge_all
      addressSetting:
        - match: myAddress
          deadLetterAddress: myDeadLetterAddress
          maxDeliveryAttempts: 5
        - match: 'myOtherAddresses*'
          deadLetterAddress: myDeadLetterAddress
          maxDeliveryAttempts: 3
```

The **applyRule** property specifies how the Operator applies the configuration that you add to the CR for each matching address or set of addresses. The values that you can specify are:

merge_all

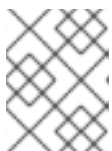
- For address settings specified in both the CR **and** the default configuration that match the same address or set of addresses:
 - Replace any property values specified in the default configuration with those specified in the CR.
 - Keep any property values that are specified uniquely in the CR **or** the default configuration. Include each of these in the final, merged configuration.
- For address settings specified in either the CR **or** the default configuration that uniquely match a particular address or set of addresses, include these in the final, merged configuration.

merge_replace

- For address settings specified in both the CR **and** the default configuration that match the same address or set of addresses, include the settings specified in the **CR** in the final, merged configuration. **Do not** include any properties specified in the default configuration, even if these are not specified in the CR.
- For address settings specified in either the CR **or** the default configuration that uniquely match a particular address or set of addresses, include these in the final, merged configuration.

replace_all

Replace **all** address settings specified in the default configuration with those specified in the CR. The final, merged configuration corresponds exactly to that specified in the CR.



NOTE

If you do not explicitly include the **applyRule** property in your CR, the Operator uses a default value of **merge_all**.

10. Deploy the broker CR instance.
 - a. Using the OpenShift command-line interface:
 - i. Save the CR file.
 - ii. Create the CR instance.

```
$ oc create -f <path/to/broker_custom_resource_instance>.yaml
```

- b. Using the OpenShift web console:
 - i. When you have finished configuring the CR, click **Create**.

Additional resources

- To learn about all of the configuration options for addresses, queues, and address settings for OpenShift Container Platform broker deployments, see [Section 11.1, “Custom Resource configuration reference”](#).
- If you installed the AMQ Broker Operator using the OpenShift command-line interface (CLI), the installation archive that you downloaded and extracted contains some additional examples of configuring address settings. In the **deploy/examples** folder of the installation archive, see:
 - **artemis-basic-address-settings-deployment.yaml**
 - **artemis-merge-replace-address-settings-deployment.yaml**
 - **artemis-replace-address-settings-deployment.yaml**
- For comprehensive information about configuring addresses, queues, and associated address settings for **standalone** broker deployments, see [Addresses, Queues, and Topics](#) in *Configuring AMQ Broker*. You can use this information to create equivalent configurations for broker deployments on OpenShift Container Platform.
- For more information about Init Containers in OpenShift Container Platform, see:
 - [Using Init Containers to perform tasks before a pod is deployed](#) (OpenShift Container Platform 4.1 and later)
 - [Init Containers](#) (OpenShift Container Platform 3.11)

4.3. CONFIGURING BROKER STORAGE REQUIREMENTS

To use persistent storage in an Operator-based broker deployment, you set **persistenceEnabled** to **true** in the Custom Resource (CR) instance used to create the deployment. If you do not have container-native storage in your OpenShift cluster, you need to manually provision Persistent Volumes (PVs) and ensure that these are available to be claimed by the Operator using a Persistent Volume Claim (PVC). If you want to create a cluster of two brokers with persistent storage, for example, then you need to have two PVs available. By default, each broker in your deployment requires storage of 2 GiB. However, you can configure the CR for your broker deployment to specify the size of PVC required by each broker.



IMPORTANT

- To configure the size of the PVC required by the brokers in an Operator-based deployment, you must be using *at least* the latest version of the Operator for AMQ Broker 7.7 (that is, version 0.17). To learn how to upgrade the Operator to the latest version for AMQ Broker 7.8, see [Chapter 6, Upgrading an Operator-based broker deployment](#).
- You must add the configuration for broker storage size to the main CR for your broker deployment **before** deploying the CR for the first time. You **cannot** add the configuration to a broker deployment that is already running.

4.3.1. Configuring broker storage size

The following procedure shows how to configure the Custom Resource (CR) instance for your broker deployment to specify the size of the Persistent Volume Claim (PVC) required by each broker for persistent message storage.



IMPORTANT

You must add the configuration for broker storage size to the main CR for your broker deployment **before** deploying the CR for the first time. You **cannot** add the configuration to a broker deployment that is already running.

Prerequisites

- You must be using *at least* the latest version of the Operator for AMQ Broker 7.7 (that is, version 0.17). To learn how to upgrade the Operator to the latest version for AMQ Broker 7.8, see [Chapter 6, Upgrading an Operator-based broker deployment](#).
- You should be familiar with how to use a CR instance to create a basic broker deployment. See [Section 3.4.1, “Deploying a basic broker instance”](#).
- You must have already provisioned Persistent Volumes (PVs) and made these available to be claimed by the Operator. For example, if you want to create a cluster of two brokers with persistent storage, you need to have two PVs available.
For more information about provisioning persistent storage, see:
 - [Understanding persistent storage](#) (OpenShift Container Platform 4.5)
 - [Persistent Storage](#) (OpenShift Container Platform 3.11).

Procedure

1. Start configuring a Custom Resource (CR) instance for the broker deployment.
 - a. Using the OpenShift command-line interface:
 - i. Log in to OpenShift as a user that has privileges to deploy CRs in the project in which you are creating the deployment.


```
oc login -u <user> -p <password> --server=<host:port>
```
 - ii. Open the sample CR file called **broker_activemqartemis_cr.yaml** that was included in the **deploy/crs** directory of the Operator installation archive that you downloaded and extracted.
 - b. Using the OpenShift Container Platform web console:
 - i. Log in to the console as a user that has privileges to deploy CRs in the project in which you are creating the deployment.
 - ii. Start a new CR instance based on the main broker CRD. In the left pane, click **Administration → Custom Resource Definitions**
 - iii. Click the **ActiveMQArtemis** CRD.
 - iv. Click the **Instances** tab.
 - v. Click **Create ActiveMQArtemis**.
Within the console, a YAML editor opens, enabling you to configure a CR instance.

For a basic broker deployment, a configuration might resemble that shown below. This configuration is the default content of the **broker_activemqartemis_cr.yaml** sample CR file.

■

```

apiVersion: broker.amq.io/v2alpha4
kind: ActiveMQArtemis
metadata:
  name: ex-aao
  application: ex-aao-app
spec:
  version: 7.8.5
  deploymentPlan:
    size: 1
    image: placeholder
    requireLogin: false
    persistenceEnabled: true
    journalType: nio
    messageMigration: true

```

Observe that in the **broker_activemqartemis_cr.yaml** sample CR file, the **image** property is set to a default value of **placeholder**. This value indicates that, by default, the **image** property does not specify a broker container image to use for the deployment. To learn how the Operator determines the appropriate broker container image to use, see [Section 2.4, “How the Operator chooses container images”](#).



NOTE

In the **metadata** section, you need to include the **namespace** property and specify a value **only** if you are using the OpenShift Container Platform web console to create your CR instance. The value that you should specify is the name of the OpenShift project for your broker deployment.

2. To specify broker storage requirements, in the **deploymentPlan** section of the CR, add a **storage** section. Add a **size** property and specify a value. For example:

```

spec:
  version: 7.8.5
  deploymentPlan:
    size: 1
    image: placeholder
    requireLogin: false
    persistenceEnabled: true
    journalType: nio
    messageMigration: true
    storage:
      size: 4Gi

```

storage.size

Size, in bytes, of the Persistent Volume Claim (PVC) that each broker Pod requires for persistent storage. This property applies only when **persistenceEnabled** is set to **true**. The value that you specify **must** include a unit. Supports byte notation (for example, K, M, G), or the binary equivalents (Ki, Mi, Gi).

3. Deploy the CR instance.
 - a. Using the OpenShift command-line interface:
 - i. Save the CR file.

- ii. Switch to the project in which you are creating the broker deployment.

```
$ oc project <project_name>
```

- iii. Create the CR instance.

```
$ oc create -f <path/to/custom_resource_instance>.yaml
```

- b. Using the OpenShift web console:

- i. When you have finished configuring the CR, click **Create**.

4.4. CONFIGURING RESOURCE LIMITS AND REQUESTS FOR OPERATOR-BASED BROKER DEPLOYMENTS

When you create an Operator-based broker deployment, the broker Pods in the deployment run in a StatefulSet on a node in your OpenShift cluster. You can configure the Custom Resource (CR) instance for the deployment to specify the host-node compute resources used by the broker container that runs in each Pod. By specifying limit and request values for CPU and memory (RAM), you can ensure satisfactory performance of the broker Pods.

IMPORTANT

- To configure resource limits and requests for the brokers in an Operator-based deployment, you must be using *at least* the latest version of the Operator for AMQ Broker 7.7 (that is, version 0.17). To learn how to upgrade the Operator to the latest version for AMQ Broker 7.8, see [Chapter 6, Upgrading an Operator-based broker deployment](#).
- You must add configuration for limits and requests to the CR instance for your broker deployment **before** deploying the CR for the first time. You **cannot** add the configuration to a broker deployment that is already running.
- It is not possible for Red Hat to recommend values for limits and requests because these are based on your specific messaging system use-cases and the resulting architecture that you have implemented. However, it *is* recommended that you test and tune these values in a development environment before configuring them for your production environment.
- The Operator runs a type of container called an *Init Container* when initializing each broker Pod. Any resource limits and requests that you configure for each broker container also apply to each Init Container. For more information about the use of Init Containers in broker deployments, see [Section 4.1, "How the Operator generates the broker configuration"](#).

You can specify the following limit and request values:

CPU limit

For each broker container running in a Pod, this value is the maximum amount of host-node CPU that the container can consume. If a broker container attempts to exceed the specified CPU limit, OpenShift throttles the container. This ensures that containers have consistent performance, regardless of the number of Pods running on a node.

Memory limit

For each broker container running in a Pod, this value is the maximum amount of host-node memory that the container can consume. If a broker container attempts to exceed the specified memory limit, OpenShift terminates the container. The broker Pod restarts.

CPU request

For each broker container running in a Pod, this value is the amount of host-node CPU that the container requests. The OpenShift scheduler considers the CPU request value during Pod placement, to bind the broker Pod to a node with sufficient compute resources.

The CPU request value is the *minimum* amount of CPU that the broker container requires to run. However, if there is no contention for CPU on the node, the container can use all available CPU. If you have specified a CPU limit, the container cannot exceed that amount of CPU usage. If there is CPU contention on the node, CPU request values provide a way for OpenShift to weigh CPU usage across all containers.

Memory request

For each broker container running in a Pod, this value is the amount of host-node memory that the container requests. The OpenShift scheduler considers the memory request value during Pod placement, to bind the broker Pod to a node with sufficient compute resources.

The memory request value is the *minimum* amount of memory that the broker container requires to run. However, the container can consume as much available memory as possible. If you have specified a memory limit, the broker container cannot exceed that amount of memory usage.

CPU is measured in units called millicores. Each node in an OpenShift cluster inspects the operating system to determine the number of CPU cores on the node. Then, the node multiplies that value by 1000 to express the total capacity. For example, if a node has two cores, the CPU capacity of the node is expressed as **2000m**. Therefore, if you want to use one-tenth of a single core, you specify a value of **100m**.

Memory is measured in bytes. You can specify the value using byte notation (E, P, T, G, M, K) or the binary equivalents (Ei, Pi, Ti, Gi, Mi, Ki). The value that you specify must include a unit.

4.4.1. Configuring broker resource limits and requests

The following example shows how to configure the main Custom Resource (CR) instance for your broker deployment to set limits and requests for CPU and memory for each broker container that runs in a Pod in the deployment.



IMPORTANT

- You must add configuration for limits and requests to the CR instance for your broker deployment **before** deploying the CR for the first time. You **cannot** add the configuration to a broker deployment that is already running.
- It is not possible for Red Hat to recommend values for limits and requests because these are based on your specific messaging system use-cases and the resulting architecture that you have implemented. However, it *is* recommended that you test and tune these values in a development environment before configuring them for your production environment.

Prerequisites

- You must be using *at least* the latest version of the Operator for AMQ Broker 7.7 (that is, version 0.17). To learn how to upgrade the Operator to the latest version for AMQ Broker 7.8, see [Chapter 6, Upgrading an Operator-based broker deployment](#).

- You should be familiar with how to use a CR instance to create a basic broker deployment. See [Section 3.4.1, “Deploying a basic broker instance”](#).

Procedure

1. Start configuring a Custom Resource (CR) instance for the broker deployment.
 - a. Using the OpenShift command-line interface:
 - i. Log in to OpenShift as a user that has privileges to deploy CRs in the project in which you are creating the deployment.


```
oc login -u <user> -p <password> --server=<host:port>
```
 - ii. Open the sample CR file called **broker_activemqartemis_cr.yaml** that was included in the **deploy/crs** directory of the Operator installation archive that you downloaded and extracted.
 - b. Using the OpenShift Container Platform web console:
 - i. Log in to the console as a user that has privileges to deploy CRs in the project in which you are creating the deployment.
 - ii. Start a new CR instance based on the main broker CRD. In the left pane, click **Administration → Custom Resource Definitions**
 - iii. Click the **ActiveMQArtemis** CRD.
 - iv. Click the **Instances** tab.
 - v. Click **Create ActiveMQArtemis**.
Within the console, a YAML editor opens, enabling you to configure a CR instance.

For a basic broker deployment, a configuration might resemble that shown below. This configuration is the default content of the **broker_activemqartemis_cr.yaml** sample CR file.

```
apiVersion: broker.amq.io/v2alpha4
kind: ActiveMQArtemis
metadata:
  name: ex-aa0
  application: ex-aa0-app
spec:
  version: 7.8.5
  deploymentPlan:
    size: 1
    image: placeholder
    requireLogin: false
    persistenceEnabled: true
    journalType: nio
    messageMigration: true
```

Observe that in the **broker_activemqartemis_cr.yaml** sample CR file, the **image** property is set to a default value of **placeholder**. This value indicates that, by default, the **image** property does not specify a broker container image to use for the deployment. To learn how the Operator determines the appropriate broker container image to use, see [Section 2.4, “How the Operator chooses container images”](#).

**NOTE**

In the **metadata** section, you need to include the **namespace** property and specify a value **only** if you are using the OpenShift Container Platform web console to create your CR instance. The value that you should specify is the name of the OpenShift project for your broker deployment.

2. In the **deploymentPlan** section of the CR, add a **resources** section. Add **limits** and **requests** sub-sections. In each sub-section, add a **cpu** and **memory** property and specify values. For example:

```
spec:
  version: 7.8.5
  deploymentPlan:
    size: 1
    image: placeholder
    requireLogin: false
    persistenceEnabled: true
    journalType: nio
    messageMigration: true
    resources:
      limits:
        cpu: "500m"
        memory: "1024M"
      requests:
        cpu: "250m"
        memory: "512M"
```

limits.cpu

Each broker container running in a Pod in the deployment cannot exceed this amount of host-node CPU usage.

limits.memory

Each broker container running in a Pod in the deployment cannot exceed this amount of host-node memory usage.

requests.cpu

Each broker container running in a Pod in the deployment requests this amount of host-node CPU. This value is the *minimum* amount of CPU required for the broker container to run.

requests.memory

Each broker container running in a Pod in the deployment requests this amount of host-node memory. This value is the *minimum* amount of memory required for the broker container to run.

3. Deploy the CR instance.
 - a. Using the OpenShift command-line interface:
 - i. Save the CR file.
 - ii. Switch to the project in which you are creating the broker deployment.

```
$ oc project <project_name>
```

- iii. Create the CR instance.

```
$ oc create -f <path/to/custom_resource_instance>.yaml
```

- b. Using the OpenShift web console:
 - i. When you have finished configuring the CR, click **Create**.

4.5. SPECIFYING A CUSTOM INIT CONTAINER IMAGE

As described in [Section 4.1, “How the Operator generates the broker configuration”](#), the AMQ Broker Operator uses a default, built-in Init Container to generate the broker configuration. To generate the configuration, the Init Container uses the main Custom Resource (CR) instance for your deployment. The **only** items that you can specify in the CR are those that are exposed in the main broker Custom Resource Definition (CRD).

However, there might a case where you need to include configuration that is **not** exposed in the CRD. In this case, in your main CR instance, you can specify a *custom* Init Container. The custom Init Container can modify or add to the configuration that has already been created by the Operator. For example, you might use a custom Init Container to modify the broker logging settings. Or, you might use a custom Init Container to include extra runtime dependencies (that is, **.jar** files) in the broker installation directory.

When you build a custom Init Container image, you must follow these important guidelines:

- In the build script (for example, a Docker Dockerfile or Podman Containerfile) that you create for the custom image, the **FROM** instruction must specify the latest version of the AMQ Broker Operator built-in Init Container as the base image. In your script, include the following line:

```
FROM registry.redhat.io/amq7/amq-broker-init-rhel7:0.2-13
```

- The custom image must include a script called **post-config.sh** that you include in a directory called **/amq/scripts**. The **post-config.sh** script is where you can modify or add to the initial configuration that the Operator generates. When you specify a custom Init Container, the Operator runs the **post-config.sh** script **after** it uses your CR instance to generate a configuration, but **before** it starts the broker application container.
- As described in [Section 4.1.2, “Directory structure of a broker Pod”](#), the path to the installation directory used by the Init Container is defined in an environment variable called **CONFIG_INSTANCE_DIR**. The **post-config.sh** script should use this environment variable name when referencing the installation directory (for example, **/\${CONFIG_INSTANCE_DIR}/lib**) and **not** the actual value of this variable (for example, **/amq/init/config/lib**).
- If you want to include additional resources (for example, **.xml** or **.jar** files) in your custom broker configuration, you must ensure that these are included in the custom image and accessible to the **post-config.sh** script.

The following procedure describes how to specify a custom Init Container image.

Prerequisites

- You must be using *at least* version 7.8.5-opr-2 of the Operator. To learn how to upgrade to the latest Operator version, see [Chapter 6, Upgrading an Operator-based broker deployment](#).
- You must have built a custom Init Container image that meets the guidelines described above. For a complete example of building and specifying a custom Init Container image for the ArtemisCloud Operator, see [custom Init Container image for JDBC-based persistence](#).

- To provide a custom Init Container image for the AMQ Broker Operator, you need to be able to add the image to a repository in a container registry such as the [Quay container registry](#).
- You should understand how the Operator uses an Init Container to generate the broker configuration. For more information, see [Section 4.1, “How the Operator generates the broker configuration”](#).
- You should be familiar with how to use a CR to create a broker deployment. For more information, see [Section 3.4, “Creating Operator-based broker deployments”](#).

Procedure

1. Start configuring a Custom Resource (CR) instance for the broker deployment.
 - a. Using the OpenShift command-line interface:
 - i. Log in to OpenShift as a user that has privileges to deploy CRs in the project in which you are creating the deployment.


```
oc login -u <user> -p <password> --server=<host:port>
```
 - ii. Open the sample CR file called **broker_activemqartemis_cr.yaml** that was included in the **deploy/crs** directory of the Operator installation archive that you downloaded and extracted.
 - b. Using the OpenShift Container Platform web console:
 - i. Log in to the console as a user that has privileges to deploy CRs in the project in which you are creating the deployment.
 - ii. Start a new CR instance based on the main broker CRD. In the left pane, click **Administration → Custom Resource Definitions**
 - iii. Click the **ActiveMQArtemis** CRD.
 - iv. Click the **Instances** tab.
 - v. Click **Create ActiveMQArtemis**.
Within the console, a YAML editor opens, enabling you to configure a CR instance.

For a basic broker deployment, a configuration might resemble that shown below. This configuration is the default content of the **broker_activemqartemis_cr.yaml** sample CR file.

```
apiVersion: broker.amq.io/v2alpha4
kind: ActiveMQArtemis
metadata:
  name: ex-aa0
  application: ex-aa0-app
spec:
  version: 7.8.5
  deploymentPlan:
    size: 1
    image: placeholder
    requireLogin: false
```

```
persistenceEnabled: true
journalType: nio
messageMigration: true
```

Observe that in the `broker_activemqartemis_cr.yaml` sample CR file, the `image` property is set to a default value of `placeholder`. This value indicates that, by default, the `image` property does not specify a broker container image to use for the deployment. To learn how the Operator determines the appropriate broker container image to use, see [Section 2.4, "How the Operator chooses container images"](#).



NOTE

In the `metadata` section, you need to include the `namespace` property and specify a value **only** if you are using the OpenShift Container Platform web console to create your CR instance. The value that you should specify is the name of the OpenShift project for your broker deployment.

- In the `deploymentPlan` section of the CR, add the `initImage` property.

```
apiVersion: broker.amq.io/v2alpha4
kind: ActiveMQArtemis
metadata:
  name: ex-aa0
  application: ex-aa0-app
spec:
  version: 7.8.5
  deploymentPlan:
    size: 1
    image: placeholder
    initImage:
      requireLogin: false
      persistenceEnabled: true
      journalType: nio
      messageMigration: true
```

- Set the value of the `initImage` property to the URL of your custom Init Container image.

```
apiVersion: broker.amq.io/v2alpha4
kind: ActiveMQArtemis
metadata:
  name: ex-aa0
  application: ex-aa0-app
spec:
  version: 7.8.5
  deploymentPlan:
    size: 1
    image: placeholder
    initImage: <custom_init_container_image_url>
      requireLogin: false
      persistenceEnabled: true
      journalType: nio
      messageMigration: true
```

initImage

Specifies the full URL for your custom Init Container image, which you must have added to repository in a container registry.

4. Deploy the CR instance.

a. Using the OpenShift command-line interface:

i. Save the CR file.

ii. Switch to the project in which you are creating the broker deployment.

```
$ oc project <project_name>
```

iii. Create the CR instance.

```
$ oc create -f <path/to/custom_resource_instance>.yaml
```

b. Using the OpenShift web console:

i. When you have finished configuring the CR, click **Create**.

Additional resources

- For a complete example of building and specifying a custom Init Container image for the ArtemisCloud Operator, see [custom Init Container image for JDBC-based persistence](#).

4.6. CONFIGURING OPERATOR-BASED BROKER DEPLOYMENTS FOR CLIENT CONNECTIONS

4.6.1. Configuring acceptors

To enable client connections to broker Pods in your OpenShift deployment, you define *acceptors* for your deployment. Acceptors define how a broker Pod accepts connections. You define acceptors in the main Custom Resource (CR) used for your broker deployment. When you create an acceptor, you specify information such as the messaging protocols to enable on the acceptor, and the port on the broker Pod to use for these protocols.

The following procedure shows how to define a new acceptor in the CR for your broker deployment.

Prerequisites

- To configure acceptors, your broker deployment must be based on version 0.9 or greater of the AMQ Broker Operator. For more information about installing the latest version of the Operator, see [Section 3.2, "Installing the Operator using the CLI"](#).
- The information in this section applies only to broker deployments based on the AMQ Broker Operator. If you used application templates to create your broker deployment, you cannot define individual protocol-specific acceptors. For more information about configuring this type of deployment for client connections, see [Chapter 6, "Connecting external clients to template-based broker deployments"](#).

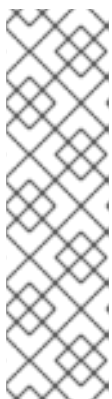
Procedure

1. In the **deploy/crs** directory of the Operator archive that you downloaded and extracted during your initial installation, open the **broker_activemqartemis_cr.yaml** Custom Resource (CR) file.
2. In the **acceptors** element, add a named acceptor. Add the **protocols** and **port** parameters. Set values to specify the messaging protocols to be used by the acceptor and the port on each broker Pod to expose for those protocols. For example:

```
spec:
...
  acceptors:
  - name: my-acceptor
    protocols: amqp
    port: 5672
...
```

The configured acceptor exposes port 5672 to AMQP clients. The full set of values that you can specify for the **protocols** parameter is shown in the table.

Protocol	Value
Core Protocol	core
AMQP	amqp
OpenWire	openwire
MQTT	mqtt
STOMP	stomp
All supported protocols	all



NOTE

- For each broker Pod in your deployment, the Operator also creates a default acceptor that uses port 61616. This default acceptor is required for broker clustering and has Core Protocol enabled.
- By default, the AMQ Broker management console uses port 8161 on the broker Pod. Each broker Pod in your deployment has a dedicated Service that provides access to the console. For more information, see [Chapter 5, Connecting to AMQ Management Console for an Operator-based broker deployment](#).

3. To use another protocol on the same acceptor, modify the **protocols** parameter. Specify a comma-separated list of protocols. For example:

```
spec:
...
  acceptors:
  - name: my-acceptor
```

```
protocols: amqp,openwire
```

```
port: 5672
```

```
...
```

The configured acceptor now exposes port 5672 to AMQP and OpenWire clients.

- To specify the number of concurrent client connections that the acceptor allows, add the **connectionsAllowed** parameter and set a value. For example:

```
spec:
```

```
...
```

```
acceptors:
```

```
- name: my-acceptor
```

```
  protocols: amqp,openwire
```

```
  port: 5672
```

```
  connectionsAllowed: 5
```

```
...
```

- By default, an acceptor is exposed only to clients in the same OpenShift cluster as the broker deployment. To also expose the acceptor to clients outside OpenShift, add the **expose** parameter and set the value to **true**.

In addition, to enable secure connections to the acceptor from clients outside OpenShift, add the **sslEnabled** parameter and set the value to **true**.

```
spec:
```

```
...
```

```
acceptors:
```

```
- name: my-acceptor
```

```
  protocols: amqp,openwire
```

```
  port: 5672
```

```
  connectionsAllowed: 5
```

```
  expose: true
```

```
  sslEnabled: true
```

```
...
```

```
...
```

When you enable SSL (that is, Secure Sockets Layer) security on an acceptor (or connector), you can add related configuration, such as:

- The secret name used to store authentication credentials in your OpenShift cluster. A secret is **required** when you enable SSL on the acceptor. For more information on generating this secret, see [Section 4.6.2, "Securing broker-client connections"](#).
- The Transport Layer Security (TLS) protocols to use for secure network communication. TLS is an updated, more secure version of SSL. You specify the TLS protocols in the **enabledProtocols** parameter.
- Whether the acceptor uses two-way TLS, also known as *mutual authentication*, between the broker and the client. You specify this by setting the value of the **needClientAuth** parameter to **true**.

Additional resources

- To learn how to configure TLS to secure broker-client connections, including generating a secret to store authentication credentials, see [Section 4.6.2, "Securing broker-client connections"](#).
- For a complete Custom Resource configuration reference, including configuration of acceptors and connectors, see [Section 11.1, "Custom Resource configuration reference"](#).

4.6.2. Securing broker-client connections

If you have enabled security on your acceptor or connector (that is, by setting **sslEnabled** to **true**), you must configure Transport Layer Security (TLS) to allow certificate-based authentication between the broker and clients. TLS is an updated, more secure version of SSL. There are two primary TLS configurations:

One-way TLS

Only the broker presents a certificate. The certificate is used by the client to authenticate the broker. This is the most common configuration.

Two-way TLS

Both the broker and the client present certificates. This is sometimes called *mutual authentication*.

The sections that follow describe:

- [Configuration requirements for the broker certificate used by one-way and two-way TLS](#)
- [How to configure one-way TLS](#)
- [How to configure two-way TLS](#)

For both one-way and two-way TLS, you complete the configuration by generating a secret that stores the credentials required for a successful TLS handshake between the broker and the client. This is the secret name that you must specify in the **sslSecret** parameter of your secured acceptor or connector. The secret must contain a Base64-encoded broker key store (both one-way and two-way TLS), a Base64-encoded broker trust store (two-way TLS only), and the corresponding passwords for these files, also Base64-encoded. The one-way and two-way TLS configuration procedures show how to generate this secret.



NOTE

If you do not explicitly specify a secret name in the **sslSecret** parameter of a secured acceptor or connector, the acceptor or connector assumes a default secret name. The default secret name uses the format **<CustomResourceName>-<AcceptorName>-secret** or **<CustomResourceName>-<ConnectorName>-secret**. For example, **my-broker-deployment-my-acceptor-secret**.

Even if the acceptor or connector assumes a default secret name, you must still generate this secret yourself. It is not automatically created.

4.6.2.1. Configuring a broker certificate for host name verification



NOTE

This section describes some requirements for the broker certificate that you must generate when configuring one-way or two-way TLS.

When a client tries to connect to a broker Pod in your deployment, the **verifyHost** option in the client connection URL determines whether the client compares the Common Name (CN) of the broker's certificate to its host name, to verify that they match. The client performs this verification if you specify **verifyHost=true** or similar in the client connection URL.

You might omit this verification in rare cases where you have no concerns about the security of the connection, for example, if the brokers are deployed on an OpenShift cluster in an isolated network. Otherwise, for a secure connection, it is advisable for a client to perform this verification. In this case, correct configuration of the broker key store certificate is essential to ensure successful client connections.

In general, when a client is using host verification, the CN that you specify when generating the broker certificate must match the full host name for the Route on the broker Pod that the client is connecting to. For example, if you have a deployment with a single broker Pod, the CN might look like the following:

```
CN=my-broker-deployment-0-svc-rte-my-openshift-project.my-openshift-domain
```

To ensure that the CN can resolve to **any** broker Pod in a deployment with multiple brokers, you can specify an asterisk (*) wildcard character in place of the ordinal of the broker Pod. For example:

```
CN=my-broker-deployment-*-svc-rte-my-openshift-project.my-openshift-domain
```

The CN shown in the preceding example successfully resolves to any broker Pod in the **my-broker-deployment** deployment.

In addition, the Subject Alternative Name (SAN) that you specify when generating the broker certificate must **individually list** all broker Pods in the deployment, as a comma-separated list. For example:

```
"SAN=DNS:my-broker-deployment-0-svc-rte-my-openshift-project.my-openshift-domain,DNS:my-broker-deployment-1-svc-rte-my-openshift-project.my-openshift-domain,..."
```

4.6.2.2. Configuring one-way TLS

The procedure in this section shows how to configure one-way Transport Layer Security (TLS) to secure a broker-client connection.

In one-way TLS, only the broker presents a certificate. This certificate is used by the client to authenticate the broker.

Prerequisites

- You should understand the requirements for broker certificate generation when clients use host name verification. For more information, see [Section 4.6.2.1, "Configuring a broker certificate for host name verification"](#).

Procedure

1. Generate a self-signed certificate for the broker key store.

```
$ keytool -genkey -alias broker -keyalg RSA -keystore ~/broker.ks
```

2. Export the certificate from the broker key store, so that it can be shared with clients. Export the certificate in the Base64-encoded **.pem** format. For example:

```
$ keytool -export -alias broker -keystore ~/broker.ks -file ~/broker_cert.pem
```

3. On the client, create a client trust store that imports the broker certificate.

```
$ keytool -import -alias broker -keystore ~/client.ts -file ~/broker_cert.pem
```

4. Log in to OpenShift Container Platform as an administrator. For example:

```
$ oc login -u system:admin
```

5. Switch to the project that contains your broker deployment. For example:

```
$ oc project my-openshift-project
```

6. Create a secret to store the TLS credentials. For example:

```
$ oc create secret generic my-tls-secret \
  --from-file=broker.ks=~/.broker.ks \
  --from-file=client.ts=~/.broker.ks \
  --from-literal=keyStorePassword=<password> \
  --from-literal=trustStorePassword=<password>
```



NOTE

When generating a secret, OpenShift requires you to specify both a key store and a trust store. The trust store key is generically named **client.ts**. For one-way TLS between the broker and a client, a trust store is not actually required. However, to successfully generate the secret, you need to specify *some* valid store file as a value for **client.ts**. The preceding step provides a "dummy" value for **client.ts** by reusing the previously-generated broker key store file. This is sufficient to generate a secret with all of the credentials required for one-way TLS.

7. Link the secret to the service account that you created when installing the Operator. For example:

```
$ oc secrets link sa/amq-broker-operator secret/my-tls-secret
```

8. Specify the secret name in the **sslSecret** parameter of your secured acceptor or connector. For example:

```
spec:
  ...
  acceptors:
  - name: my-acceptor
    protocols: amqp,openwire
    port: 5672
    sslEnabled: true
    sslSecret: my-tls-secret
    expose: true
    connectionsAllowed: 5
  ...
```

4.6.2.3. Configuring two-way TLS

The procedure in this section shows how to configure two-way Transport Layer Security (TLS) to secure a broker-client connection.

In two-way TLS, both the broker and client presents certificates. The broker and client use these certificates to authenticate each other in a process sometimes called *mutual authentication*.

Prerequisites

- You should understand the requirements for broker certificate generation when clients use host name verification. For more information, see [Section 4.6.2.1, “Configuring a broker certificate for host name verification”](#).

Procedure

1. Generate a self-signed certificate for the broker key store.

```
$ keytool -genkey -alias broker -keyalg RSA -keystore ~/broker.ks
```

2. Export the certificate from the broker key store, so that it can be shared with clients. Export the certificate in the Base64-encoded **.pem** format. For example:

```
$ keytool -export -alias broker -keystore ~/broker.ks -file ~/broker_cert.pem
```

3. On the client, create a client trust store that imports the broker certificate.

```
$ keytool -import -alias broker -keystore ~/client.ts -file ~/broker_cert.pem
```

4. On the client, generate a self-signed certificate for the client key store.

```
$ keytool -genkey -alias broker -keyalg RSA -keystore ~/client.ks
```

5. On the client, export the certificate from the client key store, so that it can be shared with the broker. Export the certificate in the Base64-encoded **.pem** format. For example:

```
$ keytool -export -alias broker -keystore ~/client.ks -file ~/client_cert.pem
```

6. Create a broker trust store that imports the client certificate.

```
$ keytool -import -alias broker -keystore ~/broker.ts -file ~/client_cert.pem
```

7. Log in to OpenShift Container Platform as an administrator. For example:

```
$ oc login -u system:admin
```

8. Switch to the project that contains your broker deployment. For example:

```
$ oc project my-openshift-project
```

9. Create a secret to store the TLS credentials. For example:

```
$ oc create secret generic my-tls-secret \
--from-file=broker.ks=~/.broker.ks \
--from-file=client.ts=~/.broker.ts \
--from-literal=keyStorePassword=<password> \
--from-literal=trustStorePassword=<password>
```



NOTE

When generating a secret, OpenShift requires you to specify both a key store and a trust store. The trust store key is generically named **client.ts**. For two-way TLS between the broker and a client, you must generate a secret that includes the broker trust store, because this holds the client certificate. Therefore, in the preceding step, the value that you specify for the **client.ts** key is actually the **broker** trust store file.

10. Link the secret to the service account that you created when installing the Operator. For example:

```
$ oc secrets link sa/amq-broker-operator secret/my-tls-secret
```

11. Specify the secret name in the **sslSecret** parameter of your secured acceptor or connector. For example:

```
spec:
...
acceptors:
- name: my-acceptor
  protocols: amqp,openwire
  port: 5672
  sslEnabled: true
  sslSecret: my-tls-secret
  expose: true
  connectionsAllowed: 5
...
```

4.6.3. Networking Services in your broker deployments

On the **Networking** pane of the OpenShift Container Platform web console for your broker deployment, there are two running Services; a *headless* Service and a *ping* Service. The default name of the headless Service uses the format **<Custom Resource name>-hdls-svc**, for example, **my-broker-deployment-hdls-svc**. The default name of the ping Service uses a format of **<Custom Resource name>-ping-svc**, for example, **my-broker-deployment-ping-svc**.

The headless Service provides access to ports 8161 and 61616 on each broker Pod. Port 8161 is used by the broker management console, and port 61616 is used for broker clustering. You can also use the headless Service to connect to a broker Pod from an internal client (that is, a client inside the same OpenShift cluster as the broker deployment).

The ping Service is used by the brokers for discovery, and enables brokers to form a cluster within the OpenShift environment. Internally, this Service exposes port 8888.

Additional resources

- To learn about using the headless Service to connect to a broker Pod from an internal client, see [Section 4.6.4.1, “Connecting to the broker from internal clients”](#) .

4.6.4. Connecting to the broker from internal and external clients

The examples in this section show how to connect to the broker from internal clients (that is, clients in the same OpenShift cluster as the broker deployment) and external clients (that is, clients outside the OpenShift cluster).

4.6.4.1. Connecting to the broker from internal clients

An internal client can connect to the broker Pod using the *headless* Service that is running for the broker deployment.

To connect to a broker Pod using the headless Service, specify an address in the format **<Protocol>://<PodName>.<HeadlessServiceName>.<ProjectName>.svc.cluster.local**. For example:

```
$ tcp://my-broker-deployment-0.my-broker-deployment-hdls-svc.my-openshift-project.svc.cluster.local
```

OpenShift DNS successfully resolves addresses in this format because the StatefulSets created by Operator-based broker deployments provide stable Pod names.

Additional resources

- For more information about the headless Service that runs by a default in a broker deployment, see [Section 4.6.3, “Networking Services in your broker deployments”](#) .

4.6.4.2. Connecting to the broker from external clients

When you expose an acceptor to external clients (that is, by setting the value of the **expose** parameter to **true**), a dedicated Service and Route are automatically created for each broker Pod in the deployment. To see the Routes configured on a given broker Pod, select the Pod in the OpenShift Container Platform web console and click the **Routes** tab.

An external client can connect to the broker by specifying the full host name of the Route created for the the broker Pod. You can use a basic **curl** command to test external access to this full host name. For example:

```
$ curl https://my-broker-deployment-0-svc-rte-my-openshift-project.my-openshift-domain
```

The full host name for the Route must resolve to the node that’s hosting the OpenShift router. The OpenShift router uses the host name to determine where to send the traffic inside the OpenShift internal network.

By default, the OpenShift router listens to port 80 for non-secured (that is, non-SSL) traffic and port 443 for secured (that is, SSL-encrypted) traffic. For an HTTP connection, the router automatically directs traffic to port 443 if you specify a secure connection URL (that is, **https**), or to port 80 if you specify a non-secure connection URL (that is, **http**).

For non-HTTP connections:

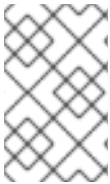
- Clients must explicitly specify the port number (for example, port 443) as part of the connection URL.

- For one-way TLS, the client must specify the path to its trust store and the corresponding password, as part of the connection URL.
- For two-way TLS, the client must **also** specify the path to its **key** store and the corresponding password, as part of the connection URL.

Some example client connection URLs, for supported messaging protocols, are shown below.

External Core client, using one-way TLS

```
tcp://my-broker-deployment-0-svc-rte-my-openshift-project.my-openshift-domain:443?
useTopologyForLoadBalancing=false&sslEnabled=true \
&trustStorePath=~/.client.ts&trustStorePassword=<password>
```



NOTE

The **useTopologyForLoadBalancing** key is explicitly set to **false** in the connection URL because an external Core client cannot use topology information returned by the broker. If this key is set to **true** or you do not specify a value, it results in a DEBUG log message.

External Core client, using two-way TLS

```
tcp://my-broker-deployment-0-svc-rte-my-openshift-project.my-openshift-domain:443?
useTopologyForLoadBalancing=false&sslEnabled=true \
&keyStorePath=~/.client.ks&keyStorePassword=<password> \
&trustStorePath=~/.client.ts&trustStorePassword=<password>
```

External OpenWire client, using one-way TLS

```
ssl://my-broker-deployment-0-svc-rte-my-openshift-project.my-openshift-domain:443"
# Also, specify the following JVM flags
-Djavax.net.ssl.trustStore=~/.client.ts -Djavax.net.ssl.trustStorePassword=<password>
```

External OpenWire client, using two-way TLS

```
ssl://my-broker-deployment-0-svc-rte-my-openshift-project.my-openshift-domain:443"
# Also, specify the following JVM flags
-Djavax.net.ssl.keyStore=~/.client.ks -Djavax.net.ssl.keyStorePassword=<password> \
-Djavax.net.ssl.trustStore=~/.client.ts -Djavax.net.ssl.trustStorePassword=<password>
```

External AMQP client, using one-way TLS

```
amqps://my-broker-deployment-0-svc-rte-my-openshift-project.my-openshift-domain:443?
transport.verifyHost=true \
&transport.trustStoreLocation=~/.client.ts&transport.trustStorePassword=<password>
```

External AMQP client, using two-way TLS

```
amqps://my-broker-deployment-0-svc-rte-my-openshift-project.my-openshift-domain:443?
transport.verifyHost=true \
```

```
&transport.keyStoreLocation=~/.client.ks&transport.keyStorePassword=<password> \
&transport.trustStoreLocation=~/.client.ts&transport.trustStorePassword=<password>
```

4.6.4.3. Connecting to the Broker using a NodePort

As an alternative to using a Route, an OpenShift administrator can configure a NodePort to connect to a broker Pod from a client outside OpenShift. The NodePort should map to one of the protocol-specific ports specified by the acceptors configured for the broker.

By default, NodePorts are in the range 30000 to 32767, which means that a NodePort typically does not match the intended port on the broker Pod.

To connect from a client outside OpenShift to the broker via a NodePort, you specify a URL in the format **<Protocol>://<OCPNodeIP>:<NodePortNumber>**.

Additional resources

- For more information about using methods such as Routes and NodePorts for communicating from outside an OpenShift cluster with services running in the cluster, see:
 - [Configuring ingress cluster traffic overview](#) (OpenShift Container Platform 4.1 and later)
 - [Getting Traffic into a Cluster](#) (OpenShift Container Platform 3.11)

4.7. CONFIGURING LARGE MESSAGE HANDLING FOR AMQP MESSAGES

Clients might send large AMQP messages that can exceed the size of the broker's internal buffer, causing unexpected errors. To prevent this situation, you can configure the broker to store messages as files when the messages are larger than a specified minimum value. Handling large messages in this way means that the broker does not hold the messages in memory. Instead, the broker stores the messages in a dedicated directory used for storing large message files.

For a broker deployment on OpenShift Container Platform, the large messages directory is **/opt/<custom-resource-name>/data/large-messages** on the Persistent Volume (PV) used by the broker for message storage. When the broker stores a message as a large message, the queue retains a reference to the file in the large messages directory.



IMPORTANT

- To configure large message handling for AMQP messages, you must be using *at least* the latest version of the Operator for AMQ Broker 7.7 (that is, version 0.17). To learn how to upgrade the Operator to the latest version for AMQ Broker 7.8, see [Chapter 6, Upgrading an Operator-based broker deployment](#).
- For Operator-based broker deployments in AMQ Broker 7.8, large message handling is available only for the AMQP protocol.

4.7.1. Configuring AMQP acceptors for large message handling

The following procedure shows how to configure an acceptor to handle an AMQP message larger than a specified size as a large message.

Prerequisites

- You must be using *at least* the latest version of the Operator for AMQ Broker 7.7 (that is, version 0.17). To learn how to upgrade the Operator to the latest version for AMQ Broker 7.8, see [Chapter 6, Upgrading an Operator-based broker deployment](#).
- You should be familiar with how to configure acceptors for Operator-based broker deployments. See [Section 4.6.1, "Configuring acceptors"](#).
- To store large AMQP messages in a dedicated large messages directory, your broker deployment must be using persistent storage (that is, **persistenceEnabled** is set to **true** in the Custom Resource (CR) instance used to create the deployment). For more information about configuring persistent storage, see:
 - [Section 2.5, "Operator deployment notes"](#)
 - [Section 11.1, "Custom Resource configuration reference"](#)

Procedure

1. Open the Custom Resource (CR) instance in which you previously defined an AMQP acceptor.
 - a. Using the OpenShift command-line interface:


```
$ oc edit -f <path/to/custom-resource-instance>.yaml
```
 - b. Using the OpenShift Container Platform web console:
 - i. In the left navigation menu, click **Administration** → **Custom Resource Definitions**
 - ii. Click the **ActiveMQArtemis** CRD.
 - iii. Click the **Instances** tab.
 - iv. Locate the CR instance that corresponds to your project namespace.

A previously-configured AMQP acceptor might resemble the following:

```
spec:
...
acceptors:
- name: my-acceptor
  protocols: amqp
  port: 5672
  connectionsAllowed: 5
  expose: true
  sslEnabled: true
...
```

2. Specify the minimum size, in bytes, of an AMQP message that the broker handles as a large message. For example:

```
spec:
...
acceptors:
- name: my-acceptor
  protocols: amqp
  port: 5672
```



```

connectionsAllowed: 5
expose: true
sslEnabled: true
amqpMinLargeMessageSize: 204800
...

```

In the preceding example, the broker is configured to accept AMQP messages on port 5672. Based on the value of **amqpMinLargeMessageSize**, if the acceptor receives an AMQP message with a body larger than or equal to 204800 bytes (that is, 200 kilobytes), the broker stores the message as a large message.

The broker stores the message in the large messages directory (**/opt/<custom-resource-name>/data/large-messages**, by default) on the persistent volume (PV) used by the broker for message storage.

If you do not explicitly specify a value for the **amqpMinLargeMessageSize** property, the broker uses a default value of 102400 (that is, 100 kilobytes).

If you set **amqpMinLargeMessageSize** to a value of **-1**, large message handling for AMQP messages is disabled.

4.8. HIGH AVAILABILITY AND MESSAGE MIGRATION

4.8.1. High availability

The term *high availability* refers to a system that can remain operational even when part of that system fails or is shut down. For AMQ Broker on OpenShift Container Platform, this means ensuring the integrity and availability of messaging data if a broker Pod fails, or shuts down due to intentional scaledown of your deployment.

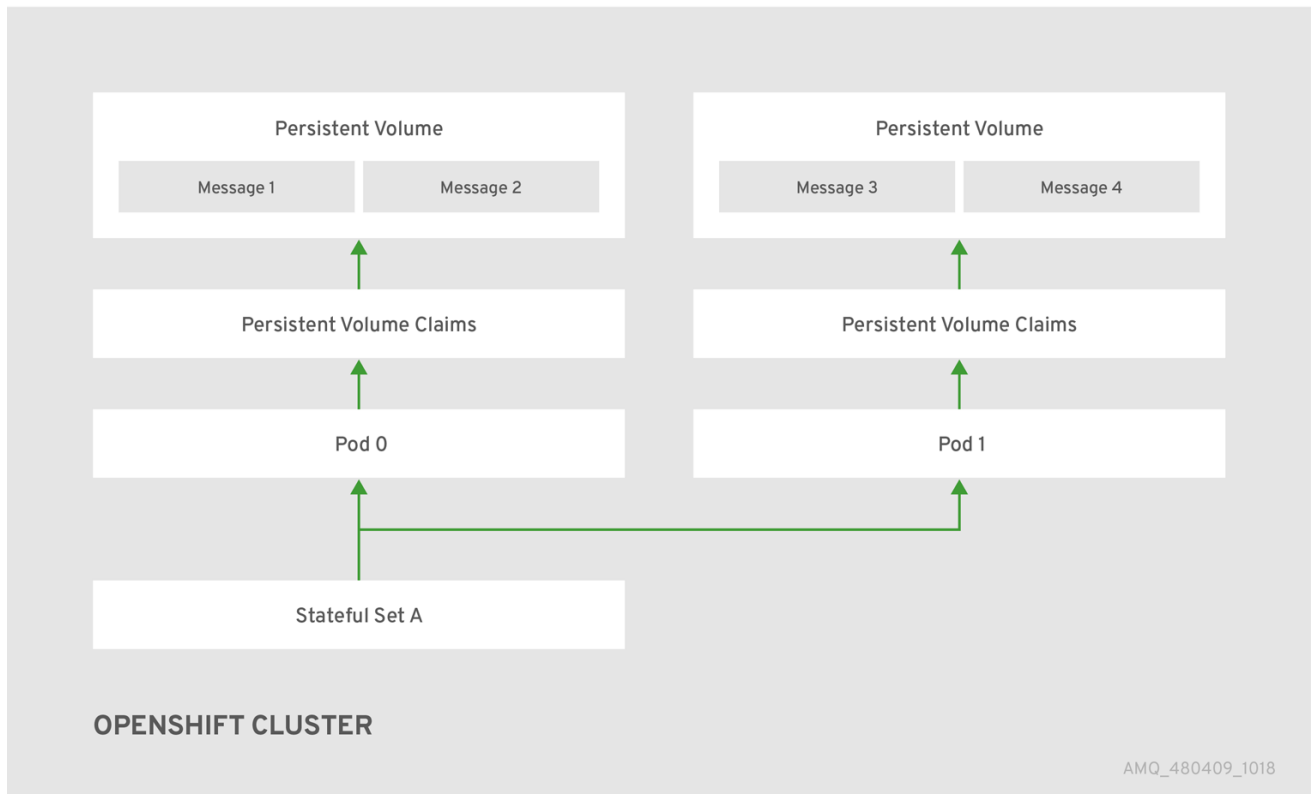
To allow high availability for AMQ Broker on OpenShift Container Platform, you run multiple broker Pods in a broker cluster. Each broker Pod writes its message data to an available Persistent Volume (PV) that you have claimed for use with a Persistent Volume Claim (PVC). If a broker Pod fails or is shut down, the message data stored in the PV is migrated to another available broker Pod in the broker cluster. The other broker Pod stores the message data in its own PV.



NOTE

Message migration is available **only** for deployments based on the AMQ Broker Operator. Deployments based on application templates **do not** have a message migration capability.

The following figure shows a StatefulSet-based broker deployment. In this case, the two broker Pods in the broker cluster are still running.



When a broker Pod shuts down, the AMQ Broker Operator automatically starts a *scaledown controller* that performs the migration of messages to another broker Pod that is still running in the broker cluster. This message migration process is also known as *Pod draining*. The section that follows describes message migration.

4.8.2. Message migration

Message migration is how you ensure the integrity of messaging data when a broker in a clustered deployment shuts down due to failure or intentional scaledown of the deployment. Also known as *Pod draining*, this process refers to removal and redistribution of messages from a broker Pod that has shut down.



NOTE

- Message migration is available **only** for deployments based on the AMQ Broker Operator. Deployments based on application templates **do not** have a message migration capability.
- The scaledown controller that performs message migration can operate only within a single OpenShift project. The controller cannot migrate messages between brokers in separate projects.
- To use message migration, you must have a minimum of two brokers in your deployment. A broker with two or more brokers is clustered by default.

For an Operator-based broker deployment, you enable message migration by setting **messageMigration** to **true** in the main broker Custom Resource for your deployment.

The message migration process follows these steps:

1. When a broker Pod in the deployment shuts down due to failure or intentional scaledown of the

deployment, the Operator automatically starts a scaledown controller to prepare for message migration. The scaledown controller runs in the same OpenShift project name as the broker cluster.

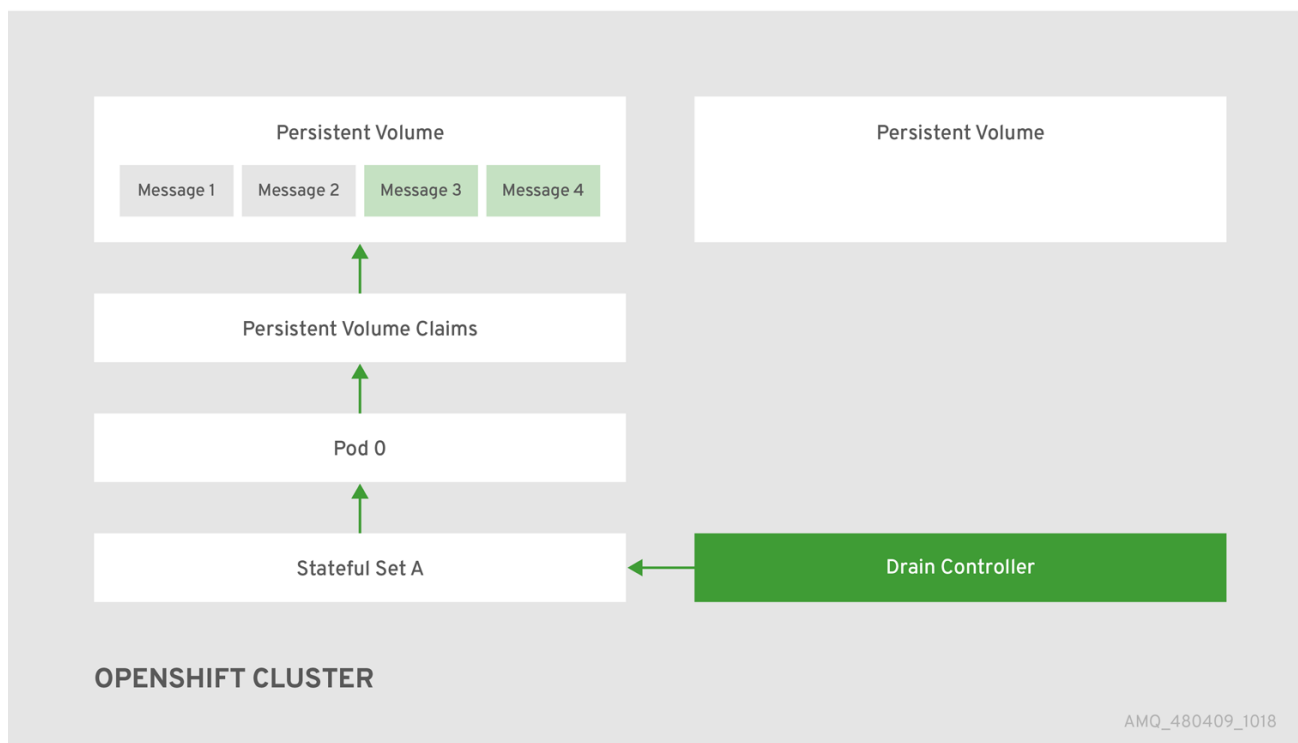
2. The scaledown controller registers itself and listens for Kubernetes events that are related to Persistent Volume Claims (PVCs) in the project.
3. To check for Persistent Volumes (PVs) that have been orphaned, the scaledown controller looks at the ordinal on the volume claim. The controller compares the ordinal on the volume claim to that of the broker Pods that are still running in the StatefulSet (that is, the broker cluster) in the project.
If the ordinal on the volume claim is higher than the ordinal on any of the broker Pods still running in the broker cluster, the scaledown controller determines that the broker Pod at that ordinal has been shut down and that messaging data must be migrated to another broker Pod.
4. The scaledown controller starts a drainer Pod. The drainer Pod runs the broker and executes the message migration. Then, the drainer Pod identifies an alternative broker Pod to which the orphaned messages can be migrated.



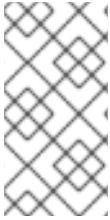
NOTE

There must be at least one broker Pod still running in your deployment for message migration to occur.

The following figure illustrates how the scaledown controller (also known as a *drain controller*) migrates messages to a running broker Pod.



After the messages are successfully migrated to an operational broker Pod, the drainer Pod shuts down and the scaledown controller removes the PVC for the orphaned PV. The PV is returned to a "Released" state.

**NOTE**

If you scale a broker deployment down to 0 (zero), message migration does not occur, since there is no running broker Pod to which messaging data can be migrated. However, if you scale a deployment down to zero and then back up to a size that is smaller than the original deployment, drainer Pods are started for the brokers that remain shut down.

Additional resources

- For an example of message migration when you scale down a broker deployment, see [Migrating messages upon scaledown](#).

4.8.3. Migrating messages upon scaledown

To migrate messages upon scaledown of your broker deployment, use the main broker Custom Resource (CR) to enable message migration. The AMQ Broker Operator automatically runs a dedicated scaledown controller to execute message migration when you scale down a clustered broker deployment.

With message migration enabled, the scaledown controller within the Operator detects shutdown of a broker Pod and starts a drainer Pod to execute message migration. The drainer Pod connects to one of the other live broker Pods in the cluster and migrates messages to that live broker Pod. After migration is complete, the scaledown controller shuts down.

**NOTE**

- A scaledown controller operates only within a single OpenShift project. The controller cannot migrate messages between brokers in separate projects.
- If you scale a broker deployment down to 0 (zero), message migration does not occur, since there is no running broker Pod to which the messaging data can be migrated. However, if you scale a deployment down to zero brokers and then back up to only some of the brokers that were in the original deployment, drainer Pods are started for the brokers that remain shut down.

The following example procedure shows the behavior of the scaledown controller.

Prerequisites

- You already have a basic broker deployment. See [Section 3.4.1, “Deploying a basic broker instance”](#).
- You should understand how message migration works. For more information, see [Section 4.8.2, “Message migration”](#).

Procedure

1. In the **deploy/crs** directory of the Operator repository that you originally downloaded and extracted, open the main broker CR, **broker_activemqartemis_cr.yaml**.
2. In the main broker CR set **messageMigration** and **persistenceEnabled** to **true**.
These settings mean that when you later scale down the size of your clustered broker deployment, the Operator automatically starts a scaledown controller and migrates messages to a broker Pod that is still running.

3. In your existing broker deployment, verify which Pods are running.

```
$ oc get pods
```

You see output that looks like the following.

```
activemq-artemis-operator-8566d9bf58-9g25l 1/1 Running 0 3m38s
ex-aa0-ss-0                               1/1 Running 0 112s
ex-aa0-ss-1                               1/1 Running 0 8s
```

The preceding output shows that there are three Pods running; one for the broker Operator itself, and a separate Pod for each broker in the deployment.

4. Log into each Pod and send some messages to each broker.
 - a. Supposing that Pod **ex-aa0-ss-0** has a cluster IP address of **172.17.0.6**, run the following command:

```
$ /opt/amq-broker/bin/artemis producer --url tcp://172.17.0.6:61616 --user admin --
password admin
```

- b. Supposing that Pod **ex-aa0-ss-1** has a cluster IP address of **172.17.0.7**, run the following command:

```
$ /opt/amq-broker/bin/artemis producer --url tcp://172.17.0.7:61616 --user admin --
password admin
```

The preceding commands create a queue called **TEST** on each broker and add 1000 messages to each queue.

5. Scale the cluster down from two brokers to one.
 - a. Open the main broker CR, **broker_activemqartemis_cr.yaml**.
 - b. In the CR, set **deploymentPlan.size** to **1**.
 - c. At the command line, apply the change:

```
$ oc apply -f deploy/crs/broker_activemqartemis_cr.yaml
```

You see that the Pod **ex-aa0-ss-1** starts to shut down. The scaledown controller starts a new drainer Pod of the same name. This drainer Pod also shuts down after it migrates all messages from broker Pod **ex-aa0-ss-1** to the other broker Pod in the cluster, **ex-aa0-ss-0**.

6. When the drainer Pod is shut down, check the message count on the **TEST** queue of broker Pod **ex-aa0-ss-0**. You see that the number of messages in the queue is 2000, indicating that the drainer Pod successfully migrated 1000 messages from the broker Pod that shut down.

CHAPTER 5. CONNECTING TO AMQ MANAGEMENT CONSOLE FOR AN OPERATOR-BASED BROKER DEPLOYMENT

Each broker Pod in an Operator-based deployment hosts its own instance of AMQ Management Console at port 8161. To provide access to the console for each broker, you can configure the Custom Resource (CR) instance for the broker deployment to instruct the Operator to automatically create a dedicated Service and Route for each broker Pod.

The following procedures describe how to connect to AMQ Management Console for a deployed broker.

Prerequisites

- You must have created a broker deployment using the AMQ Broker Operator. For example, to learn how to use a sample CR to create a basic broker deployment, see [Section 3.4.1, “Deploying a basic broker instance”](#).
- To instruct the Operator to automatically create a Service and Route for each broker Pod in a deployment for console access, you must set the value of the **console.expose** property to **true** in the Custom Resource (CR) instance used to create the deployment. The default value of this property is **false**. For a complete Custom Resource configuration reference, including configuration of the **console** section of the CR, see [Section 11.1, “Custom Resource configuration reference”](#).

5.1. CONNECTING TO AMQ MANAGEMENT CONSOLE

When you set the value of the **console.expose** property to **true** in the Custom Resource (CR) instance used to create a broker deployment, the Operator automatically creates a dedicated Service and Route for each broker Pod, to provide access to AMQ Management Console.

The default name of the automatically-created Service is in the form **<custom-resource-name>wconsj-<broker-pod-ordinal>-svc**. For example, **my-broker-deployment-wconsj-0-svc**. The default name of the automatically-created Route is in the form **<custom-resource-name>wconsj-<broker-pod-ordinal>-svc-rte**. For example, **my-broker-deployment-wconsj-0-svc-rte**.

This procedure shows you how to access the console for a running broker Pod.

Procedure

1. In the OpenShift Container Platform web console, click **Networking** → **Routes** (OpenShift Container Platform 4.5 or later) or **Applications** → **Routes** (OpenShift Container Platform 3.11). On the **Routes** page, identify the **wconsj** Route for the given broker Pod. For example, **my-broker-deployment-wconsj-0-svc-rte**.
2. Under **Location** (OpenShift Container Platform 4.5 or later) or **Hostname** (OpenShift Container Platform 3.11), click the link that corresponds to the Route. A new tab opens in your web browser.
3. Click the **Management Console** link. The AMQ Management Console login page opens.
4. To log in to the console, enter the values specified for the **adminUser** and **adminPassword** properties in the Custom Resource (CR) instance used to create your broker deployment.

If there are no values explicitly specified for **adminUser** and **adminPassword** in the CR, follow the instructions in [Section 5.2, “Accessing AMQ Management Console login credentials”](#) to retrieve the credentials required to log in to the console.



NOTE

Values for **adminUser** and **adminPassword** are required to log in to the console **only** if the **requireLogin** property of the CR is set to **true**. This property specifies whether login credentials are required to log in to the broker **and** the console. If **requireLogin** is set to **false**, any user with administrator privileges for the OpenShift project can log in to the console.

5.2. ACCESSING AMQ MANAGEMENT CONSOLE LOGIN CREDENTIALS

If you do not specify a value for **adminUser** and **adminPassword** in the Custom Resource (CR) instance used for your broker deployment, the Operator automatically generates these credentials and stores them in a secret. The default secret name is in the form **<custom-resource-name>-credentials-secret**, for example, **my-broker-deployment-credentials-secret**.



NOTE

Values for **adminUser** and **adminPassword** are required to log in to the management console **only** if the **requireLogin** parameter of the CR is set to **true**. If **requireLogin** is set to **false**, any user with administrator privileges for the OpenShift project can log in to the console.

This procedure shows how to access the login credentials.

Procedure

1. See the complete list of secrets in your OpenShift project.
 - a. From the OpenShift Container Platform web console, click **Workload** → **Secrets** (OpenShift Container Platform 4.5 or later) or **Resources** → **Secrets** (OpenShift Container Platform 3.11).
 - b. From the command line:

```
$ oc get secrets
```

2. Open the appropriate secret to reveal the Base64-encoded console login credentials.
 - a. From the OpenShift Container Platform web console, click the secret that includes your broker Custom Resource instance in its name. Click the **YAML** tab (OpenShift Container Platform 4.5 or later) or **Actions** → **Edit YAML** (OpenShift Container Platform 3.11).
 - b. From the command line:

```
$ oc edit secret <my-broker-deployment-credentials-secret>
```

3. To decode a value in the secret, use a command such as the following:

```
$ echo 'dXNlcl9uYW11' | base64 --decode  
console_admin
```

Additional resources

- To learn more about using AMQ Management Console to view and manage brokers, see [Managing brokers using AMQ Management Console](#) in Managing AMQ Broker

CHAPTER 6. UPGRADING AN OPERATOR-BASED BROKER DEPLOYMENT

The procedures in this section show how to upgrade:

- The AMQ Broker Operator version, using both the OpenShift command-line interface (CLI) and OperatorHub
- The broker container image for an Operator-based broker deployment

6.1. BEFORE YOU BEGIN

This section describes some important considerations before you upgrade the Operator and broker container images for an Operator-based broker deployment.

- To upgrade an Operator-based broker deployment running on OpenShift Container Platform 3.11 to run on OpenShift Container Platform 4.5 or later, you must first upgrade your OpenShift Container Platform installation. Then, you must create a new Operator-based broker deployment that matches your existing deployment. To learn how to create a new Operator-based broker deployment, see [Chapter 3, Deploying AMQ Broker on OpenShift Container Platform using the AMQ Broker Operator](#).
- Upgrading the Operator using either the OpenShift command-line interface (CLI) or OperatorHub requires cluster administrator privileges for your OpenShift cluster.
- If you originally used the CLI to *install* the Operator, you should also use the CLI to *upgrade* the Operator. If you originally used OperatorHub to install the Operator (that is, it appears under **Operators → Installed Operators** for your project in the OpenShift Container Platform web console), you should also use OperatorHub to upgrade the Operator. For more information about these upgrade methods, see:
 - [Section 6.2, “Upgrading the Operator using the CLI”](#)
 - [Section 6.3.3, “Upgrading the Operator using OperatorHub”](#)

6.2. UPGRADING THE OPERATOR USING THE CLI

The procedures in this section show how to use the OpenShift command-line interface (CLI) to upgrade different versions of the Operator to the latest version available for AMQ Broker 7.8.

6.2.1. Prerequisites

- You should use the CLI to upgrade the Operator only if you originally used the CLI to *install* the Operator. If you originally used OperatorHub to install the Operator (that is, the Operator appears under **Operators → Installed Operators** for your project in the OpenShift Container Platform web console), you should use OperatorHub to upgrade the Operator. To learn how to upgrade the Operator using OperatorHub, see [Section 6.3, “Upgrading the Operator using OperatorHub”](#).

6.2.2. Upgrading version 0.19 of the Operator

This procedure shows to how to use the OpenShift command-line interface (CLI) to upgrade version 0.19 of the Operator to the latest version for AMQ Broker 7.8.

Procedure

1. In your web browser, navigate to the **Software Downloads** page for [AMQ Broker 7.8.5 patches](#).
2. Ensure that the value of the **Version** drop-down list is set to **7.8.5** and the **Patches** tab is selected.
3. Next to **AMQ Broker 7.8.5 .3 Operator Installation and Example Files** click **Download**. Download of the **amq-broker-operator-7.8.5-ocp-install-examples.zip** compressed archive automatically begins.
4. When the download has completed, move the archive to your chosen installation directory. The following example moves the archive to a directory called **~/broker/operator**.

```
mkdir ~/broker/operator
mv amq-broker-operator-7.8.5-ocp-install-examples.zip ~/broker/operator
```

5. In your chosen installation directory, extract the contents of the archive. For example:

```
cd ~/broker/operator
unzip amq-broker-operator-7.8.5-ocp-install-examples.zip
```

6. Log in to OpenShift Container Platform as an administrator for the project that contains your existing Operator deployment.

```
$ oc login -u <user>
```

7. Switch to the OpenShift project in which you want to upgrade your Operator version.

```
$ oc project <project-name>
```

8. In the **deploy** directory of the latest Operator archive that you downloaded and extracted, open the **operator.yaml** file.



NOTE

In the **operator.yaml** file, the Operator uses an image that is represented by a *Secure Hash Algorithm* (SHA) value. The comment line, which begins with a number sign (**#**) symbol, denotes that the SHA value corresponds to a specific container image tag.

9. Open the **operator.yaml** file for your **previous** Operator deployment. Check that any non-default values that you specified in your previous configuration are replicated in the **new operator.yaml** configuration file.
10. If you have made any updates to the **new operator.yaml** file, save the file.
11. Apply the updated Operator configuration.

```
$ oc apply -f deploy/operator.yaml
```

OpenShift updates your project to use the latest Operator version.

12. To recreate your previous broker deployment, create a new CR yaml file to match the purpose

of your original CR and apply it. [Section 3.4.1, “Deploying a basic broker instance”](#). describes how to apply the `deploy/crs/broker_activemqartemis_cr.yaml` file in the Operator installation archive, you can use that file as a basis for your new CR yaml file.

6.2.3. Upgrading version 0.18 of the Operator

This procedure shows to how to use the OpenShift command-line interface (CLI) to upgrade version 0.18 of the Operator (that is, the *first* version available for AMQ Broker 7.8) to the latest version for AMQ Broker 7.8.

Procedure

1. In your web browser, navigate to the **Software Downloads** page for [AMQ Broker 7.8.5 patches](#).
2. Ensure that the value of the **Version** drop-down list is set to **7.8.5** and the **Patches** tab is selected.
3. Next to **AMQ Broker 7.8.5 .3 Operator Installation and Example Files** click **Download**. Download of the `amq-broker-operator-7.8.5-ocp-install-examples.zip` compressed archive automatically begins.
4. When the download has completed, move the archive to your chosen installation directory. The following example moves the archive to a directory called `~/broker/operator`.

```
mkdir ~/broker/operator
mv amq-broker-operator-7.8.5-ocp-install-examples.zip ~/broker/operator
```

5. In your chosen installation directory, extract the contents of the archive. For example:

```
cd ~/broker/operator
unzip amq-broker-operator-7.8.5-ocp-install-examples.zip
```

6. Log in to OpenShift Container Platform as an administrator for the project that contains your existing Operator deployment.

```
$ oc login -u <user>
```

7. Switch to the OpenShift project in which you want to upgrade your Operator version.

```
$ oc project <project-name>
```

8. In the `deploy` directory of the latest Operator archive that you downloaded and extracted, open the `operator.yaml` file.



NOTE

In the `operator.yaml` file, the Operator uses an image that is represented by a *Secure Hash Algorithm* (SHA) value. The comment line, which begins with a number sign (`#`) symbol, denotes that the SHA value corresponds to a specific container image tag.

- Open the **operator.yaml** file for your **previous** Operator deployment. Check that any non-default values that you specified in your previous configuration are replicated in the **new operator.yaml** configuration file.



NOTE

The **operator.yaml** file for version 0.18 of the Operator includes environment variables whose names begin with **BROKER_IMAGE**. **Do not** replicate these environment variables in your new configuration. The latest version of the Operator for AMQ Broker 7.8 no longer uses these environment variables.

- If you have made any updates to the **new operator.yaml** file, save the file.
- Apply the updated Operator configuration.

```
$ oc apply -f deploy/operator.yaml
```

OpenShift updates your project to use the latest Operator version.

- To recreate your previous broker deployment, create a new CR yaml file to match the purpose of your original CR and apply it. [Section 3.4.1, "Deploying a basic broker instance"](#) describes how to apply the **deploy/crs/broker_activemqartemis_cr.yaml** file in the Operator installation archive, you can use that file as a basis for your new CR yaml file.

6.2.4. Upgrading version 0.17 of the Operator

This procedure shows to how to use the OpenShift command-line interface (CLI) to upgrade version 0.17 of the Operator (that is, the latest version available for AMQ Broker 7.7) to the latest version for AMQ Broker 7.8.

Procedure

- In your web browser, navigate to the **Software Downloads** page for [AMQ Broker 7.8.5 patches](#).
- Ensure that the value of the **Version** drop-down list is set to **7.8.5** and the **Patches** tab is selected.
- Next to **AMQ Broker 7.8.5 .3 Operator Installation and Example Files** click **Download**. Download of the **amq-broker-operator-7.8.5-ocp-install-examples.zip** compressed archive automatically begins.
- When the download has completed, move the archive to your chosen installation directory. The following example moves the archive to a directory called **~/broker/operator**.

```
mkdir ~/broker/operator
mv amq-broker-operator-7.8.5-ocp-install-examples.zip ~/broker/operator
```

- In your chosen installation directory, extract the contents of the archive. For example:

```
cd ~/broker/operator
unzip amq-broker-operator-7.8.5-ocp-install-examples.zip
```

- Log in to OpenShift Container Platform as a cluster administrator. For example:

```
$ oc login -u system:admin
```

- Switch to the OpenShift project in which you want to upgrade your Operator version.

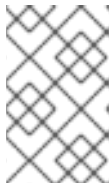
```
$ oc project <project-name>
```

- Delete the main broker Custom Resource (CR) instance in your project. This also deletes the broker deployment. For example:

```
$ oc delete -f deploy/crs/broker_activemqartemis_cr.yaml
```

- Update the main broker Custom Resource Definition (CRD) in your OpenShift cluster to the latest version.

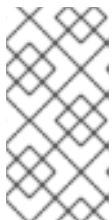
```
$ oc apply -f deploy/crds/broker_activemqartemis_crd.yaml
```



NOTE

You **do not** need to update your cluster with the latest versions of the CRDs for addressing or the scaledown controller. These CRDs are fully compatible with the ones included with the previous Operator version.

- In the **deploy** directory of the latest Operator archive that you downloaded and extracted, open the **operator.yaml** file.



NOTE

In the **operator.yaml** file, the Operator uses an image that is represented by a *Secure Hash Algorithm* (SHA) value. The comment line, which begins with a number sign (#) symbol, denotes that the SHA value corresponds to a specific container image tag.

- Open the **operator.yaml** file for your **previous** Operator deployment. Check that any non-default values that you specified in your previous configuration are replicated in the **new operator.yaml** configuration file.



NOTE

The **operator.yaml** file for version 0.17 of the Operator includes environment variables whose names begin with **BROKER_IMAGE**. **Do not** replicate these environment variables in your new configuration. The latest version of the Operator for AMQ Broker 7.8 no longer uses these environment variables.

- If you have made any updates to the **new operator.yaml** file, save the file.
- Apply the updated Operator configuration.

```
$ oc apply -f deploy/operator.yaml
```

OpenShift updates your project to use the latest Operator version.

- To recreate your previous broker deployment, create a new CR yaml file to match the purpose

of your original CR and apply it. [Section 3.4.1, "Deploying a basic broker instance"](#) . describes how to apply the **deploy/crs/broker_activemqartemis_cr.yaml** file in the Operator installation archive, you can use that file as a basis for your new CR yaml file.

6.2.5. Upgrading version 0.15 of the Operator

This procedure shows to how to use the OpenShift command-line interface (CLI) to upgrade version 0.15 of the Operator (that is, the first version available for AMQ Broker 7.7) to the latest version for AMQ Broker 7.8.

Procedure

1. In your web browser, navigate to the **Software Downloads** page for [AMQ Broker 7.8.5 patches](#) .
2. Ensure that the value of the **Version** drop-down list is set to **7.8.5** and the **Patches** tab is selected.
3. Next to **AMQ Broker 7.8.5 .3 Operator Installation and Example Files** click **Download**. Download of the **amq-broker-operator-7.8.5-ocp-install-examples.zip** compressed archive automatically begins.
4. When the download has completed, move the archive to your chosen installation directory. The following example moves the archive to a directory called **~/broker/operator**.

```
mkdir ~/broker/operator
mv amq-broker-operator-7.8.5-ocp-install-examples.zip ~/broker/operator
```

5. In your chosen installation directory, extract the contents of the archive. For example:

```
cd ~/broker/operator
unzip amq-broker-operator-7.8.5-ocp-install-examples.zip
```

6. Log in to OpenShift Container Platform as a cluster administrator. For example:

```
$ oc login -u system:admin
```

7. Switch to the OpenShift project in which you want to upgrade your Operator version.

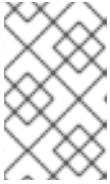
```
$ oc project <project-name>
```

8. Delete the main broker Custom Resource (CR) instance in your project. This also deletes the broker deployment. For example:

```
$ oc delete -f deploy/crs/broker_activemqartemis_cr.yaml
```

9. Update the main broker Custom Resource Definition (CRD) in your OpenShift cluster to the latest version.

```
$ oc apply -f deploy/crds/broker_activemqartemis_crd.yaml
```

**NOTE**

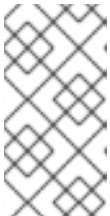
You **do not** need to update your cluster with the latest versions of the CRDs for addressing or the scaledown controller. These CRDs are fully compatible with the ones included with the previous Operator version.

10. In the **deploy** directory of the latest Operator archive that you downloaded and extracted, open the **operator.yaml** file.

**NOTE**

In the **operator.yaml** file, the Operator uses an image that is represented by a *Secure Hash Algorithm* (SHA) value. The comment line, which begins with a number sign (**#**) symbol, denotes that the SHA value corresponds to a specific container image tag.

11. Open the **operator.yaml** file for your **previous** Operator deployment. Check that any non-default values that you specified in your previous configuration are replicated in the **new operator.yaml** configuration file.

**NOTE**

The **operator.yaml** file for version 0.15 of the Operator includes environment variables whose names begin with **BROKER_IMAGE**. **Do not** replicate these environment variables in your new configuration. The latest version of the Operator for AMQ Broker 7.8 no longer uses these environment variables.

12. If you have made any updates to the **new operator.yaml** file, save the file.
13. Apply the updated Operator configuration.

```
$ oc apply -f deploy/operator.yaml
```

OpenShift updates your project to use the latest Operator version.

14. To recreate your previous broker deployment, create a new CR yaml file to match the purpose of your original CR and apply it. [Section 3.4.1, “Deploying a basic broker instance”](#). describes how to apply the **deploy/crs/broker_activemqartemis_cr.yaml** file in the Operator installation archive, you can use that file as a basis for your new CR yaml file.

6.2.6. Upgrading version 0.13 of the Operator

This procedure shows to how to use the OpenShift command-line interface (CLI) to upgrade version 0.13 of the Operator (that is, the version available for AMQ Broker 7.6) to the latest version for AMQ Broker 7.8.

Procedure

1. In your web browser, navigate to the **Software Downloads** page for [AMQ Broker 7.8.5 patches](#).
2. Ensure that the value of the **Version** drop-down list is set to **7.8.5** and the **Patches** tab is selected.
3. Next to **AMQ Broker 7.8.5 .3 Operator Installation and Example Files** click **Download**.

Download of the **amq-broker-operator-7.8.5-ocp-install-examples.zip** compressed archive automatically begins.

- When the download has completed, move the archive to your chosen installation directory. The following example moves the archive to a directory called **~/broker/operator**.

```
mkdir ~/broker/operator
mv amq-broker-operator-7.8.5-ocp-install-examples.zip ~/broker/operator
```

- In your chosen installation directory, extract the contents of the archive. For example:

```
cd ~/broker/operator
unzip amq-broker-operator-7.8.5-ocp-install-examples.zip
```

- Log in to OpenShift Container Platform as a cluster administrator. For example:

```
$ oc login -u system:admin
```

- Switch to the OpenShift project in which you want to upgrade your Operator version.

```
$ oc project <project-name>
```

- Delete the main broker Custom Resource (CR) instance in your project. This also deletes the broker deployment. For example:

```
$ oc delete -f deploy/crs/broker_activemqartemis_cr.yaml
```

- Update the main broker Custom Resource Definition (CRD) in your OpenShift cluster to the latest version.

```
$ oc apply -f deploy/crds/broker_activemqartemis_crd.yaml
```

- Update the **address** CRD in your OpenShift cluster to the latest version included with AMQ Broker 7.8.

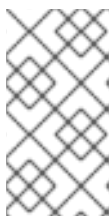
```
$ oc apply -f deploy/crds/broker_activemqartemisaddress_crd.yaml
```



NOTE

You **do not** need to update your cluster with the latest version of the CRD for the scaledown controller. In AMQ Broker 7.8, this CRD is fully compatible with the one that was included with the Operator for AMQ Broker 7.6.

- In the **deploy** directory of the latest Operator archive that you downloaded and extracted, open the **operator.yaml** file.



NOTE

In the **operator.yaml** file, the Operator uses an image that is represented by a *Secure Hash Algorithm* (SHA) value. The comment line, which begins with a number sign (**#**) symbol, denotes that the SHA value corresponds to a specific container image tag.

12. Open the **operator.yaml** file for your **previous** Operator deployment. Check that any non-default values that you specified in your previous configuration are replicated in the **new operator.yaml** configuration file.
13. If you have made any updates to the **new operator.yaml** file, save the file.
14. Apply the updated Operator configuration.

```
$ oc apply -f deploy/operator.yaml
```

OpenShift updates your project to use the latest Operator version.

6.2.7. Upgrading version 0.9 of the Operator

The following procedure shows how to use the OpenShift command-line interface (CLI) to upgrade version 0.9 of the Operator (that is, the version available for AMQ Broker 7.5 or the Long Term Support version available for AMQ Broker 7.4) to the latest version for AMQ Broker 7.8.

Procedure

1. In your web browser, navigate to the **Software Downloads** page for [AMQ Broker 7.8.5 patches](#).
2. Ensure that the value of the **Version** drop-down list is set to **7.8.5** and the **Patches** tab is selected.
3. Next to **AMQ Broker 7.8.5 .3 Operator Installation and Example Files** click **Download**. Download of the **amq-broker-operator-7.8.5-ocp-install-examples.zip** compressed archive automatically begins.
4. When the download has completed, move the archive to your chosen installation directory. The following example moves the archive to a directory called **~/broker/operator**.

```
mkdir ~/broker/operator
mv amq-broker-operator-7.8.5-ocp-install-examples.zip ~/broker/operator
```

5. In your chosen installation directory, extract the contents of the archive. For example:

```
cd ~/broker/operator
unzip amq-broker-operator-7.8.5-ocp-install-examples.zip
```

6. Log in to OpenShift Container Platform as a cluster administrator. For example:

```
$ oc login -u system:admin
```

7. Switch to the OpenShift project in which you want to upgrade your Operator version.

```
$ oc project <project-name>
```

8. Delete the main broker Custom Resource (CR) instance in your project. This also deletes the broker deployment. For example:

```
$ oc delete -f deploy/crs/broker_v2alpha1_activemqartemis_cr.yaml
```

- Update the main broker Custom Resource Definition (CRD) in your OpenShift cluster to the latest version included with AMQ Broker 7.8.

```
$ oc apply -f deploy/crds/broker_activemqartemis_crd.yaml
```

- Update the address CRD in your OpenShift cluster to the latest version included with AMQ Broker 7.8.

```
$ oc apply -f deploy/crds/broker_activemqartemisaddress_crd.yaml
```



NOTE

You **do not** need to update your cluster with the latest version of the CRD for the scaledown controller. In AMQ Broker 7.8, this CRD is fully compatible with the one included with the previous Operator version.

- In the **deploy** directory of the latest Operator archive that you downloaded and extracted, open the **operator.yaml** file.



NOTE

In the **operator.yaml** file, the Operator uses an image that is represented by a *Secure Hash Algorithm* (SHA) value. The comment line, which begins with a number sign (**#**) symbol, denotes that the SHA value corresponds to a specific container image tag.

- Open the **operator.yaml** file for your **previous** Operator deployment. Check that any non-default values that you specified in your previous configuration are replicated in the **new operator.yaml** configuration file.
- If you have made any updates to the **new operator.yaml** file, save the file.
- Apply the updated Operator configuration.

```
$ oc apply -f deploy/operator.yaml
```

OpenShift updates your project to use the latest Operator version.

- To recreate your previous broker deployment, create a new CR yaml file to match the purpose of your original CR and apply it. [Section 3.4.1, "Deploying a basic broker instance"](#) describes how to apply the **deploy/crds/broker_activemqartemis_cr.yaml** file in the Operator installation archive, you can use that file as a basis for your new CR yaml file.

6.3. UPGRADING THE OPERATOR USING OPERATORHUB

This section describes how to use OperatorHub to upgrade different versions of the Operator to the latest version available for AMQ Broker 7.8.

6.3.1. Prerequisites

- You should use OperatorHub to upgrade the Operator only if you originally used OperatorHub to *install* the Operator (that is, the Operator appears under **Operators** → **Installed Operators** for your project in the OpenShift Container Platform web console). By contrast, if you originally

used the OpenShift command-line interface (CLI) to install the Operator, you should also use the CLI to upgrade the Operator. To learn how to upgrade the Operator using the CLI, see [Section 6.2, “Upgrading the Operator using the CLI”](#).

- Upgrading the AMQ Broker Operator using OperatorHub requires cluster administrator privileges for your OpenShift cluster.

6.3.2. Before you begin

This section describes some important considerations before you use OperatorHub to upgrade an instance of the AMQ Broker Operator.

- The Operator Lifecycle Manager **automatically** updates the CRDs in your OpenShift cluster when you install the latest Operator version from OperatorHub. You do not need to remove existing CRDs.
- When you update your cluster with the CRDs for the latest Operator version, this update affects **all** projects in the cluster. Any broker Pods deployed from previous versions of the Operator might become unable to update their status in the OpenShift Container Platform web console. When you click the **Logs** tab of a running broker Pod, you see messages indicating that 'UpdatePodStatus' has failed. However, the broker Pods and Operator in that project continue to work as expected. To fix this issue for an affected project, you must also upgrade that project to use the latest version of the Operator.

6.3.3. Upgrading the Operator using OperatorHub

This procedure shows how to use OperatorHub to upgrade an instance of the AMQ Broker Operator.

Procedure

1. Log in to the OpenShift Container Platform web console as a cluster administrator.
2. Delete the main Custom Resource (CR) instance for the broker deployment in your project. This action deletes the broker deployment.
 - a. In the left navigation menu, click **Administration** → **Custom Resource Definitions**
 - b. On the **Custom Resource Definitions** page, click the **ActiveMQArtemis** CRD.
 - c. Click the **Instances** tab.
 - d. Locate the CR instance that corresponds to your project namespace.
 - e. For your CR instance, click the **More Options** icon (three vertical dots) on the right-hand side. Select **Delete ActiveMQArtemis**.
3. Uninstall the existing AMQ Broker Operator from your project.
 - a. In the left navigation menu, click **Operators** → **Installed Operators**.
 - b. From the **Project** drop-down menu at the top of the page, select the project in which you want to uninstall the Operator.
 - c. Locate the **Red Hat Integration - AMQ Broker** instance that you want to uninstall.

- d. For your Operator instance, click the **More Options** icon (three vertical dots) on the right-hand side. Select **Uninstall Operator**.
 - e. On the confirmation dialog box, click **Uninstall**.
4. Use OperatorHub to install the latest version of the Operator for AMQ Broker 7.8. For more information, see [Section 3.3.3, "Deploying the Operator from OperatorHub"](#).
 5. To recreate your previous broker deployment, create a new CR yaml file to match the purpose of your original CR and apply it. [Section 3.4.1, "Deploying a basic broker instance"](#) describes how to apply the **deploy/crs/broker_activemqartemis_cr.yaml** file in the Operator installation archive, you can use that file as a basis for your new CR yaml file.

6.4. UPGRADING THE BROKER CONTAINER IMAGE BY SPECIFYING AN AMQ BROKER VERSION

The following procedure shows how to upgrade the broker container image for an Operator-based broker deployment by specifying an AMQ Broker version. You might do this, for example, if you upgrade the Operator to the latest version for AMQ Broker 7.8.5 but the **spec.upgrades.enabled** property in your CR is already set to **true** and the **spec.version** property specifies **7.7.0** or **7.8.0**. To *upgrade* the broker container image, you need to manually specify a new AMQ Broker version (for example, **7.8.5**). When you specify a new version of AMQ Broker, the Operator automatically chooses the broker container image that corresponds to this version.

Prerequisites

- You must be using the latest version of the Operator for 7.8.5. To learn how to upgrade the Operator to the latest version, see:
 - [Section 6.2, "Upgrading the Operator using the CLI"](#)
 - [Section 6.3.3, "Upgrading the Operator using OperatorHub"](#).
- As described in [Section 2.4, "How the Operator chooses container images"](#), if you deploy a CR and do not explicitly specify a broker container image, the Operator automatically chooses the appropriate container image to use. To use the upgrade process described in this section, you **must** use this default behavior. If you override the default behavior by directly specifying a broker container image in your CR, the Operator **cannot** automatically upgrade the broker container image to correspond to an AMQ Broker version as described below.

Procedure

1. Edit the main broker CR instance for the broker deployment.
 - a. Using the OpenShift command-line interface:
 - i. Log in to OpenShift as a user that has privileges to edit and deploy CRs in the project for the broker deployment.

```
$ oc login -u <user> -p <password> --server=<host:port>
```

- ii. In a text editor, open the CR file that you used for your broker deployment. For example, this might be the **broker_activemqartemis_cr.yaml** file that was included in the **deploy/crs** directory of the Operator installation archive that you previously downloaded and extracted.

- b. Using the OpenShift Container Platform web console:
 - i. Log in to the console as a user that has privileges to edit and deploy CRs in the project for the broker deployment.
 - ii. In the left pane, click **Administration** → **Custom Resource Definitions**
 - iii. Click the **ActiveMQArtemis** CRD.
 - iv. Click the **Instances** tab.
 - v. Locate the CR instance that corresponds to your project namespace.
 - vi. For your CR instance, click the **More Options** icon (three vertical dots) on the right-hand side. Select **Edit ActiveMQArtemis**.
Within the console, a YAML editor opens, enabling you to edit the CR instance.

2. To specify a version of AMQ Broker to which to upgrade the broker container image, set a value for the **spec.version** property of the CR. For example:

```
spec:
  version: 7.8.5
  ...
```

3. In the **spec** section of the CR, locate the **upgrades** section. If this section is not already included in the CR, add it.

```
spec:
  version: 7.8.5
  ...
  upgrades:
```

4. Ensure that the **upgrades** section includes the **enabled** and **minor** properties.

```
spec:
  version: 7.8.5
  ...
  upgrades:
    enabled:
    minor:
```

5. To enable an upgrade of the broker container image based on a specified version of AMQ Broker, set the value of the **enabled** property to **true**.

```
spec:
  version: 7.8.5
  ...
  upgrades:
    enabled: true
    minor:
```

6. To define the upgrade behavior of the broker, set a value for the **minor** property.
 - a. To allow upgrades between **minor** AMQ Broker versions, set the value of **minor** to **true**.

```
spec:
  version: 7.8.5
  ...
  upgrades:
    enabled: true
    minor: true
```

For example, suppose that the current broker container image corresponds to **7.7.0**, and a new image, corresponding to the **7.8.5** version specified for **spec.version**, is available. In this case, the Operator determines that there is an available upgrade between the **7.7** and **7.8** minor versions. Based on the preceding settings, which allow upgrades between minor versions, the Operator upgrades the broker container image.

By contrast, suppose that the current broker container image corresponds to **7.8.0**, and a new image, corresponding to the **7.8.5** version specified for **spec.version**, is available. In this case, the Operator determines that there is an available upgrade between **7.8.0** and **7.8.5** micro versions. Based on the preceding settings, which allow upgrades only between minor versions, the Operator **does not** upgrade the broker container image.

- b. To allow upgrades between **micro** AMQ Broker versions, set the value of **minor** to **false**.

```
spec:
  version: 7.8.5
  ...
  upgrades:
    enabled: true
    minor: false
```

For example, suppose that the current broker container image corresponds to **7.7.0**, and a new image, corresponding to the **7.8.5** version specified for **spec.version**, is available. In this case, the Operator determines that there is an available upgrade between the **7.7** and **7.8** minor versions. Based on the preceding settings, which do not allow upgrades between minor versions (that is, only between micro versions), the Operator **does not** upgrade the broker container image.

By contrast, suppose that the current broker container image corresponds to **7.8.0**, and a new image, corresponding to the **7.8.5** version specified for **spec.version**, is available. In this case, the Operator determines that there is an available upgrade between **7.8.0** and **7.8.5** micro versions. Based on the preceding settings, which allow upgrades between micro versions, the Operator upgrades the broker container image.

7. Apply the changes to the CR.

- a. Using the OpenShift command-line interface:
 - i. Save the CR file.
 - ii. Switch to the project for the broker deployment.

```
$ oc project <project_name>
```

- iii. Apply the CR.

```
$ oc apply -f <path/to/broker_custom_resource_instance>.yaml
```

- b. Using the OpenShift web console:
 - i. When you have finished editing the CR, click **Save**.

When you apply the CR change, the Operator first validates that an upgrade to the AMQ Broker version specified for **spec.version** is available for your existing deployment. If you have specified an invalid version of AMQ Broker to which to upgrade (for example, a version that is not yet available), the Operator logs a warning message, and takes no further action.

However, if an upgrade to the specified version **is** available, and the values specified for **upgrades.enabled** and **upgrades.minor** allow the upgrade, then the Operator upgrades each broker in the deployment to use the broker container image that corresponds to the new AMQ Broker version.

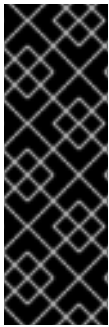
The broker container image that the Operator uses is defined in an environment variable in the **operator.yaml** configuration file of the Operator deployment. The environment variable name includes an identifier for the AMQ Broker version. For example, the environment variable **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_781** corresponds to AMQ Broker 7.8.1.

When the Operator has applied the CR change, it restarts each broker Pod in your deployment so that each Pod uses the specified image version. If you have multiple brokers in your deployment, only one broker Pod shuts down and restarts at a time.

Additional resources

- To learn how the Operator uses environment variables to choose a broker container image, see [Section 2.4, “How the Operator chooses container images”](#).

CHAPTER 7. DEPLOYING AMQ BROKER ON OPENSIFT CONTAINER PLATFORM USING APPLICATION TEMPLATES



IMPORTANT

Starting in 7.8, the use of application templates for deploying AMQ Broker on OpenShift Container Platform is a deprecated feature. This feature will be removed in a future release. Red Hat continues to support existing deployments that are based on application templates. However, Red Hat does not recommend using application templates for new deployments. For new deployments, Red Hat recommends using the AMQ Broker Operator. For information on installing and deploying the Operator, see [Chapter 3, Deploying AMQ Broker on OpenShift Container Platform using the AMQ Broker Operator](#).

The procedures in this section show:

- How to install the AMQ Broker image streams and application templates
- How to prepare a template-based broker deployment
- An example of using the OpenShift Container Platform web console to deploy a basic broker instance using an application template. For examples of deploying other broker configurations using templates, see [template-based broker deployment examples](#).

7.1. PREREQUISITES

- You should have read the comparison of methods for deploying AMQ Broker on OpenShift Container Platform. For new deployments, Red Hat recommends using the AMQ Broker Operator. For more information, see [Chapter 2, Planning a deployment of AMQ Broker on OpenShift Container Platform](#).

7.2. INSTALLING THE IMAGE STREAMS AND APPLICATION TEMPLATES

The AMQ Broker on OpenShift Container Platform image streams and application templates are not available in OpenShift Container Platform by default. You must manually install them using the procedure in this section. When you have completed the manual installation, you can then instantiate a template that enables you to deploy a chosen broker configuration on your OpenShift cluster. For examples of creating various broker configurations in this way, see [Deploying AMQ Broker on OpenShift Container Platform using application templates](#) and [template-based broker deployment examples](#).

Procedure

1. At the command line, log in to OpenShift as a cluster administrator (or as a user that has namespace-specific administrator access for the global **openshift** project namespace), for example:

```
$ oc login -u system:admin
$ oc project openshift
```

Using the **openshift** project makes the image stream and application templates that you install later in this procedure globally available to all projects in your OpenShift cluster. If you want to explicitly specify that image streams and application templates are imported to the **openshift**

project, you can also add **-n openshift** as an optional parameter with the **oc replace** commands that you use later in the procedure.

As an alternative to using the **openshift** project (e.g., if a cluster administrator is unavailable), you can log in to a specific OpenShift project to which you have administrator access and in which you want to create a broker deployment, for example:

```
$ oc login -u <USERNAME>
$ oc project <PROJECT_NAME>
```

Logging into a specific project makes the image stream and templates that you install later in this procedure available only in that project's namespace.



NOTE

AMQ Broker on OpenShift Container Platform uses StatefulSet resources with all ***-persistence*.yaml** templates. For templates that are not ***-persistence*.yaml**, AMQ Broker uses Deployments resources. Both types of resources are Kubernetes-native resources that can consume image streams **only** from the same project namespace in which the template will be instantiated.

- At the command line, run the following commands to import the broker image streams to your project namespace. Using the **--force** option with the **oc replace** command updates the resources, or creates them if they don't already exist.

```
$ oc replace --force -f \
https://raw.githubusercontent.com/jboss-container-images/jboss-amq-7-broker-openshift-
image/78-7.8.5.GA/amq-broker-7-image-streams.yaml
```

- Run the following command to update the AMQ Broker application templates.

```
$ for template in amq-broker-78-basic.yaml \
amq-broker-78-ssl.yaml \
amq-broker-78-custom.yaml \
amq-broker-78-persistence.yaml \
amq-broker-78-persistence-ssl.yaml \
amq-broker-78-persistence-clustered.yaml \
amq-broker-78-persistence-clustered-ssl.yaml;
do
oc replace --force -f \
https://raw.githubusercontent.com/jboss-container-images/jboss-amq-7-broker-openshift-
image/78-7.8.5.GA/templates/${template}
done
```

7.3. PREPARING A TEMPLATE-BASED BROKER DEPLOYMENT

Prerequisites

- Before deploying a broker instance on OpenShift Container Platform, you must have installed the AMQ Broker image streams and application templates. For more information, see [Installing the image streams and application templates](#).
- The following procedure assumes that the broker image stream and application templates you

installed are available in the global **openshift** project. If you installed the image and application templates in a specific project namespace, then continue to use that project instead of creating a new project such as **amq-demo**.

Procedure

1. Use the command prompt to create a new project:

```
$ oc new-project amq-demo
```

2. Create a service account to be used for the AMQ Broker deployment:

```
$ echo '{"kind": "ServiceAccount", "apiVersion": "v1", "metadata": {"name": "amq-service-account"}}' | oc create -f -
```

3. Add the view role to the service account. The view role enables the service account to view all the resources in the amq-demo namespace, which is necessary for managing the cluster when using the OpenShift dns-ping protocol for discovering the broker cluster endpoints.

```
$ oc policy add-role-to-user view system:serviceaccount:amq-demo:amq-service-account
```

4. AMQ Broker requires a broker keystore, a client keystore, and a client truststore that includes the broker keystore. This example uses Java Keytool, a package included with the Java Development Kit, to generate dummy credentials for use with the AMQ Broker installation.

- a. Generate a self-signed certificate for the broker keystore:

```
$ keytool -genkey -alias broker -keyalg RSA -keystore broker.ks
```

- b. Export the certificate so that it can be shared with clients:

```
$ keytool -export -alias broker -keystore broker.ks -file broker_cert
```

- c. Generate a self-signed certificate for the client keystore:

```
$ keytool -genkey -alias client -keyalg RSA -keystore client.ks
```

- d. Create a client truststore that imports the broker certificate:

```
$ keytool -import -alias broker -keystore client.ts -file broker_cert
```

- e. Use the broker keystore file to create the AMQ Broker secret:

```
$ oc create secret generic amq-app-secret --from-file=broker.ks
```

- f. Link the secret to the service account created earlier:

```
$ oc secrets link sa/amq-service-account secret/amq-app-secret
```

7.4. DEPLOYING A BASIC BROKER

The procedure in this section shows you how to deploy a basic broker that is ephemeral and does not support SSL.



NOTE

This broker does not support SSL and is not accessible to external clients. Only clients running internally on the OpenShift cluster can connect to the broker. For examples of creating broker configurations that support SSL, see [template-based broker deployment examples](#).

Prerequisites

- You have already prepared the broker deployment. See [Preparing a template-based broker deployment](#).
- The following procedure assumes that the broker image stream and application templates you installed in [Installing the image streams and application templates](#) are available in the global **openshift** project. If you installed the image and application templates in a specific project namespace, then continue to use that project instead of creating a new project such as **amq-demo**.
- Starting in AMQ Broker 7.3, you use a new version of the Red Hat Ecosystem Catalog to access container images. This new version of the registry requires you to become an authenticated user before you can access images and pull them into an OpenShift project. Before following the procedure in this section, you must first complete the steps described in [Red Hat Container Registry Authentication](#).

7.4.1. Creating the broker application

Procedure

1. Log in to the **amq-demo** project space, or another, existing project in which you want to deploy a broker.

```
$ oc login -u <USER_NAME>
$ oc project <PROJECT_NAME>
```

2. Create a new broker application, based on the template for a basic broker. The broker created by this template is ephemeral and does not support SSL.

```
$ oc new-app --template=amq-broker-78-basic \
-p AMQ_PROTOCOL=openwire,amqp,stomp,mqtt,hornetq \
-p AMQ_QUEUES=demoQueue \
-p AMQ_ADDRESSES=demoTopic \
-p AMQ_USER=amq-demo-user \
-p AMQ_PASSWORD=password \
```

The basic broker application template sets the environment variables shown in the following table.

Table 7.1. Basic broker application template

Environment variable	Display Name	Value	Description
AMQ_PROTOCOL	AMQ Protocols	openwire,amqp,stormq,mqtt,hornetq	The protocols to be accepted by the broker
AMQ_QUEUES	Queues	demoQueue	Creates an anycast queue called demoQueue
AMQ_ADDRESSES	Addresses	demoTopic	Creates an address (or topic) called demoTopic. By default, this address has no assigned routing type.
AMQ_USER	AMQ Username	amq-demo-user	User name that the client uses to connect to the broker
AMQ_PASSWORD	AMQ Password	password	Password that the client uses with the user name to connect to the broker

7.4.2. About sensitive credentials

In the AMQ Broker application templates, the values of the following environment variables are stored in a secret:

- AMQ_USER
- AMQ_PASSWORD
- AMQ_CLUSTER_USER (clustered broker deployments)
- AMQ_CLUSTER_PASSWORD (clustered broker deployments)
- AMQ_TRUSTSTORE_PASSWORD (SSL-enabled broker deployments)
- AMQ_KEYSTORE_PASSWORD (SSL-enabled broker deployments)

To retrieve and use the values for these environment variables, the AMQ Broker application templates access the secret specified in the AMQ_CREDENTIAL_SECRET environment variable. By default, the secret name specified in this environment variable is **amq-credential-secret**. Even if you specify a custom value for any of these variables when deploying a template, OpenShift Container Platform uses the value currently stored in the named secret. Furthermore, the application templates always use the default values stored in **amq-credential-secret** unless you edit the secret to change the values, or create and specify a new secret with new values. You can edit a secret using the OpenShift command-line interface, as shown in this example:

```
$ oc edit secrets amq-credential-secret
```

Values in the **amq-credential-secret** use base64 encoding. To decode a value in the secret, use a command that looks like this:

```
$ echo 'dXNlcl9uYW11' | base64 --decode
user_name
```

7.4.3. Deploying and starting the broker application

After the broker application is created, you need to deploy it. Deploying the application creates a Pod for the broker to run in.

Procedure

1. Click **Deployments** in the OpenShift Container Platform web console.
2. Click the **broker-amq** application.
3. Click **Deploy**.



NOTE

If the application does not deploy, you can check the configuration by clicking the **Events** tab. If something is incorrect, edit the deployment configuration by clicking the **Actions** button.

4. After you deploy the broker application, inspect the current state of the broker Pod.
 - a. Click **DeploymentConfigs**.
 - b. Click the **broker-amq** Pod and then click the **Logs** tab to verify the state of the broker. You should see the queue previously created via the application template. If the logs show that:
 - The broker is running, skip to step 9 of this procedure.
 - The broker logs have not loaded, and the Pod status shows **ErrImagePull** or **ImagePullBackOff**, your deployment configuration was not able to directly pull the specified broker image from the Red Hat Container Registry. In this case, continue to step 5 of this procedure.
5. To prepare the Pod for installation of the broker container image, scale the number of running brokers to **0**.
 - a. Click **DeploymentConfigs** → **broker-amq**.
 - b. Click **Actions** → **Edit DeploymentConfigs**.
 - c. In the deployment config **.yaml** file, set the value of the **replicas** attribute to **0**.
 - d. Click **Save**.
 - e. The pod restarts, with zero broker instances running.
6. Install the latest broker container image.
 - a. In your web browser, navigate to the [Red Hat Container Catalog](#).
 - b. In the search box, enter **AMQ Broker**. Click **Search**.

- c. Choose an image repository based on the information in the following table.

Platform (Architecture)	Container image name	Repository name
OpenShift Container Platform (amd64)	AMQ Broker or AMQ Broker for RHEL 8	amq7/amq-broker or amq7/amq-broker-rhel8
OpenShift Container Platform on IBM Z (s390x)	AMQ Broker for RHEL 8 on OpenJDK 11	amq7/amq-broker-openjdk-11-rhel8
OpenShift Container Platform on IBM Power Systems (ppc64le)	AMQ Broker for RHEL 8 on OpenJDK 11	amq7/amq-broker-openjdk-11-rhel8

For example, for the OpenShift Container Platform broker container image, click **AMQ Broker**. The **amq7/amq-broker** repository opens, with the most recent image version automatically selected. If you want to change to an earlier image version, click the **Tags** tab and choose another version tag.

- d. Click the **Get This Image** tab.
- e. Under **Authentication with registry tokens**, review the on-page instructions in the **Using OpenShift secrets** section. The instructions describe how to add references to the broker image and the image pull secret name associated with the account used for authentication in the Red Hat Container Registry to your Pod deployment configuration file. For example, to reference the broker image and pull secret in the **broker-amq** deployment configuration in the **amq-demo** project namespace, include lines that look like the following:

```
apiVersion: apps.openshift.io/v1
kind: DeploymentConfig
..
metadata:
  name: broker-amq
  namespace: amq-demo
..
spec:
  containers:
    name: broker-amq
    image: 'registry.redhat.io/amq7/amq-broker:7.8'
  ..
  imagePullSecrets:
    - name: {PULL-SECRET-NAME}
```

- f. Click **Save**.
7. Import the latest broker image version to your project namespace. For example:

```
$ oc import-image amq7/amq-broker:7.8 --from=registry.redhat.io/amq7/amq-broker --confirm
```

8. Edit the **broker-amq** deployment config again, as previously described. Set the value of the **replicas** attribute back to its original value. The broker Pod restarts, with all running brokers referencing the new broker image.

- Click the **Terminal** tab to access a shell where you can start the broker and use the CLI to test sending and consuming messages.

```
sh-4.2$ ./broker/bin/artemis run
sh-4.2$ ./broker/bin/artemis producer --destination queue://demoQueue
Producer ActiveMQQueue[demoQueue], thread=0 Started to calculate elapsed time ...

Producer ActiveMQQueue[demoQueue], thread=0 Produced: 1000 messages
Producer ActiveMQQueue[demoQueue], thread=0 Elapsed time in second : 4 s
Producer ActiveMQQueue[demoQueue], thread=0 Elapsed time in milli second : 4584 milli
seconds
sh-4.2$ ./broker/bin/artemis consumer --destination queue://demoQueue
Consumer:: filter = null
Consumer ActiveMQQueue[demoQueue], thread=0 wait until 1000 messages are consumed
Received 1000
Consumer ActiveMQQueue[demoQueue], thread=0 Consumed: 1000 messages
Consumer ActiveMQQueue[demoQueue], thread=0 Consumer thread finished
```

Alternatively, use the OpenShift client to access the shell using the Pod name, as shown in the following example.

```
// Get the Pod names and internal IP Addresses
$ oc get pods -o wide

// Access a broker Pod by name
$ oc rsh <broker-pod-name>
```

7.5. CONNECTING EXTERNAL CLIENTS TO TEMPLATE-BASED BROKER DEPLOYMENTS

This section describes how to configure SSL to enable connections from clients outside OpenShift Container Platform to brokers deployed using application templates.

7.5.1. Configuring SSL

For a minimal SSL configuration to allow connections outside of OpenShift Container Platform, AMQ Broker requires a broker keystore, a client keystore, and a client truststore that includes the broker keystore. The broker keystore is also used to create a secret for the AMQ Broker on OpenShift Container Platform image, which is added to the service account.

The following example commands use Java KeyTool, a package included with the Java Development Kit, to generate the necessary certificates and stores.

For a more complete example of deploying a broker instance that supports SSL, see [Deploying a basic broker with SSL](#).

Procedure

- Generate a self-signed certificate for the broker keystore:

```
$ keytool -genkey -alias broker -keyalg RSA -keystore broker.ks
```

- Export the certificate so that it can be shared with clients:

■

```
$ keytool -export -alias broker -keystore broker.ks -file broker_cert
```

3. Generate a self-signed certificate for the client keystore:

```
$ keytool -genkey -alias client -keyalg RSA -keystore client.ks
```

4. Create a client truststore that imports the broker certificate:

```
$ keytool -import -alias broker -keystore client.ts -file broker_cert
```

5. Export the client's certificate from the keystore:

```
$ keytool -export -alias client -keystore client.ks -file client_cert
```

6. Import the client's exported certificate into a broker SERVER truststore:

```
$ keytool -import -alias client -keystore broker.ts -file client_cert
```

7.5.2. Generating the AMQ Broker secret

The broker keystore can be used to generate a secret for the namespace, which is also added to the service account so that the applications can be authorized.

Procedure

- At the command line, run the following commands:

```
$ oc create secret generic <secret-name> --from-file=<broker-keystore> --from-file=<broker-truststore>
$ oc secrets link sa/<service-account-name> secret/<secret-name>
```

7.5.3. Creating an SSL Route

To enable client applications outside your OpenShift cluster to connect to a broker, you need to create an SSL Route for the broker Pod. You can expose only SSL-enabled Routes to external clients because the OpenShift router requires Server Name Indication (SNI) to send traffic to the correct Service.

When you use an application template to deploy a broker on OpenShift Container Platform, you use the **AMQ_PROTOCOL** template parameter to specify the messaging protocols that the broker uses, in a comma-separated list. Available options are **amqp**, **mqtt**, **openwire**, **stomp**, and **hornetq**. If you do not specify any protocols, all protocols are made available.

For each messaging protocol that the broker uses, OpenShift exposes a dedicated port on the broker Pod. In addition, OpenShift automatically creates a multiplexed, *all protocols* port. Client applications outside OpenShift always use the multiplexed, all protocols port to connect to the broker, regardless of which of the supported protocols they are using.

Connections to the all protocols port are via a Service that OpenShift automatically creates, and an SSL Route that you create. A headless service within the broker Pod provides access to the other protocol-specific ports, which do not have their own Services and Routes that clients can access directly.

The ports that OpenShift exposes for the various AMQ Broker transport protocols are shown in the following table. Brokers listen on the non-SSL ports for traffic within the OpenShift cluster. Brokers

listen on the SSL-enabled ports for traffic from clients outside OpenShift, if you created your deployment using an SSL-based (that is, ***-ssl.yaml**) template.

Table 7.2. Default ports for AMQ Broker transport protocols

AMQ Broker transport protocol	Default port
All protocols (OpenWire, AMQP, STOMP, MQTT, and HornetQ)	61616
All protocols -SSL (OpenWire AMQP, STOMP, MQTT, and HornetQ)	61617
AMQP	5672
AMQP (SSL)	5671
MQTT	1883
MQTT (SSL)	8883
STOMP	61613
STOMP (SSL)	61612

Below are some other things to note when creating an SSL Route on your broker Pod:

- When you create a Route, setting **TLS Termination** to **Passthrough** relays all communication to AMQ Broker without the OpenShift router decrypting and resending it.



NOTE

Regular HTTP traffic does not require a TLS passthrough Route because the OpenShift router uses **HAProxy**, which is an HTTP proxy.

- External broker clients must specify the OpenShift router port (443, by default) when setting the broker URL for SSL connections. When a client connection specifies the OpenShift router port, the router determines the appropriate port on the broker Pod to which the client traffic should be directed.



NOTE

By default, the OpenShift router uses port 443. However, the router might be configured to use a different port number, based on the value specified for the **ROUTER_SERVICE_HTTPS_PORT** environment variable. For more information, see [OpenShift Container Platform Routes](#).

- Including the failover protocol in the broker URL preserves the client connection in case the Pod is restarted or upgraded, or a disruption occurs on the router. Both of the previous settings are shown in the example below.

■

```
...  
factory.setBrokerURL("failover://ssl://<broker-pod-route-name>:443");  
...
```

Additional resources

- For a complete example of deploying a broker that supports SSL and of creating an SSL Route to enable external client access, see [Deploying a basic broker with SSL](#).
- For an example of creating Routes for clustered brokers to connect to their own instances of the AMQ Broker management console, see [Creating routes for the AMQ Broker management console](#).

CHAPTER 8. TEMPLATE-BASED BROKER DEPLOYMENT EXAMPLES

Prerequisites

- These procedures assume an OpenShift Container Platform instance similar to that created in [OpenShift Container Platform Getting Started](#).
- In the AMQ Broker application templates, the values of the AMQ_USER, AMQ_PASSWORD, AMQ_CLUSTER_USER, AMQ_CLUSTER_PASSWORD, AMQ_TRUSTSTORE_PASSWORD, and AMQ_KEYSTORE_PASSWORD environment variables are stored in a secret. To learn more about using and modifying these environment variables when you deploy a template in any of tutorials that follow, see [About sensitive credentials](#).

The following procedures example how to use application templates to create various deployments of brokers.

8.1. DEPLOYING A BASIC BROKER WITH SSL

Deploy a basic broker that is ephemeral and supports SSL.

8.1.1. Deploying the image and template

Prerequisites

- This tutorial builds upon [Preparing a template-based broker deployment](#).
- Completion of the [Deploying a basic broker](#) tutorial is recommended.

Procedure

1. Navigate to the OpenShift web console and log in.
2. Select the **amq-demo** project space.
3. Click **Add to Project** > **Browse Catalog** to list all of the default image streams and templates.
4. Use the **Filter** search bar to limit the list to those that match **amq**. You might need to click **See all** to show the desired application template.
5. Select the **amq-broker-78-ssl** template which is labeled **Red Hat AMQ Broker 7.8 (Ephemeral, with SSL)**.
6. Set the following values in the configuration and click **Create**.

Table 8.1. Example template

Environment variable	Display Name	Value	Description
AMQ_PROTOCOL	AMQ Protocols	openwire,amqp,storm,mqtt,horntq	The protocols to be accepted by the broker

Environment variable	Display Name	Value	Description
AMQ_QUEUES	Queues	demoQueue	Creates an anycast queue called demoQueue
AMQ_ADDRESSES	Addresses	demoTopic	Creates an address (or topic) called demoTopic. By default, this address has no assigned routing type.
AMQ_USER	AMQ Username	amq-demo-user	The username the client uses
AMQ_PASSWORD	AMQ Password	password	The password the client uses with the username
AMQ_TRUSTSTORE	Trust Store Filename	broker.ts	The SSL truststore file name
AMQ_TRUSTSTORE_PASSWORD	Truststore Password	password	The password used when creating the Truststore
AMQ_KEYSTORE	AMQ Keystore Filename	broker.ks	The SSL keystore file name
AMQ_KEYSTORE_PASSWORD	AMQ Keystore Password	password	The password used when creating the Keystore

8.1.2. Deploying the application

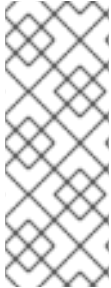
After creating the application, deploy it to create a Pod and start the broker.

Procedure

1. Click **Deployments** in the OpenShift Container Platform web console.
2. Click the **broker-amq** deployment.
3. Click **Deploy** to deploy the application.
4. Click the broker Pod and then click the **Logs** tab to verify the state of the broker.
If the broker logs have not loaded, and the Pod status shows **ErrImagePull** or **ImagePullBackOff**, your deployment configuration was not able to directly pull the specified broker image from the Red Hat Container Registry. In this case, edit your deployment configuration to reference the correct broker image name and the image pull secret name associated with the account used for authentication in the Red Hat Container Registry. Then, you can import the broker image and start the broker. To do this, complete steps similar to those in [Deploying and starting the broker application](#).

8.1.3. Creating a Route

Create a Route for the broker so that clients outside of OpenShift Container Platform can connect using SSL. By default, the secured broker protocols are available through the 61617/TCP port. In addition, there are SSL and non-SSL ports exposed on the broker Pod for each messaging protocol that the broker supports. However, external client cannot connect directly to these ports on the broker. Instead, external clients connect to OpenShift via the Openshift router, which determines how to forward traffic to the appropriate port on the broker Pod.



NOTE

If you scale your deployment up to multiple brokers in a cluster, you must manually create a Service and a Route for each broker, and then use each Service-and-Route combination to direct a given client to a given broker, or broker list. For an example of configuring multiple Services and Routes to connect clustered brokers to their own instances of the AMQ Broker management console, see [Creating Routes for the AMQ Broker management console](#).

Prerequisites

- Before creating an SSL Route, you should understand how external clients use this Route to connect to the broker. For more information, see [Creating an SSL Route](#).

Procedure

1. Click **Services** → **broker-amq-tcp-ssl**.
2. Click **Actions** → **Create a route**.
3. To display the TLS parameters, select the **Secure route** check box.
4. From the **TLS Termination** drop-down menu, choose **Passthrough**. This selection relays all communication to AMQ Broker without the OpenShift router decrypting and resending it.
5. To view the Route, click **Routes**. For example:

```
https://broker-amq-tcp-amq-demo.router.default.svc.cluster.local
```

This hostname will be used by external clients to connect to the broker using SSL with SNI.

Additional resources

- For more information about creating SSL Routes, see [Creating an SSL Route](#).
- For more information on Routes in the OpenShift Container Platform, see [Routes](#).

8.2. DEPLOYING A BASIC BROKER WITH PERSISTENCE AND SSL

Deploy a persistent broker that supports SSL. When a broker needs persistence, the broker is deployed as a StatefulSet and stores messaging data on a persistent volume associated with the broker Pod via a persistent volume claim. When a broker Pod is created, it uses storage that remains in the event that you shut down the Pod, or if the Pod shuts down unexpectedly. This configuration means that messages are not lost, as they would be with a standard deployment.

Prerequisites

- This tutorial builds upon [Preparing a template-based broker deployment](#).
- Completion of the [Deploying a basic broker](#) tutorial is recommended.
- You must have sufficient persistent storage provisioned to your OpenShift cluster to associate with your broker Pod via a persistent volume claim. For more information, see [Understanding persistent storage](#) (OpenShift Container Platform 4.5)

8.2.1. Deploy the image and template

Procedure

1. Navigate to the OpenShift web console and log in.
2. Select the **amq-demo** project space.
3. Click **Add to Project** → **Browse catalog** to list all of the default image streams and templates.
4. Use the **Filter** search bar to limit the list to those that match **amq**. You might need to click **See all** to show the desired application template.
5. Select the **amq-broker-78-persistence-ssl** template, which is labelled **Red Hat AMQ Broker 7.8 (Persistence, with SSL)**.
6. Set the following values in the configuration and click **create**.

Table 8.2. Example template

Environment variable	Display Name	Value	Description
AMQ_PROTOCOL	AMQ Protocols	openwire,amqp,storm,mqtt,horntq	The protocols to be accepted by the broker
AMQ_QUEUES	Queues	demoQueue	Creates an anycast queue called demoQueue
AMQ_ADDRESSES	Addresses	demoTopic	Creates an address (or topic) called demoTopic. By default, this address has no assigned routing type.
VOLUME_CAPACITY	AMQ Volume Size	1Gi	The persistent volume size created for the journal
AMQ_USER	AMQ Username	amq-demo-user	The username the client uses
AMQ_PASSWORD	AMQ Password	password	The password the client uses with the username

Environment variable	Display Name	Value	Description
AMQ_TRUSTSTORE	Trust Store Filename	broker.ts	The SSL truststore file name
AMQ_TRUSTSTORE_PASSWORD	Truststore Password	password	The password used when creating the Truststore
AMQ_KEYSTORE	AMQ Keystore Filename	broker.ks	The SSL keystore file name
AMQ_KEYSTORE_PASSWORD	AMQ Keystore Password	password	The password used when creating the Keystore

8.2.2. Deploy the application

Once the application has been created it needs to be deployed. Deploying the application creates a Pod and starts the broker.

Procedure

1. Click **StatefulSets** in the OpenShift Container Platform web console.
2. Click the **broker-amq** deployment.
3. Click **Deploy** to deploy the application.
4. Click the broker Pod and then click the **Logs** tab to verify the state of the broker. You should see the queue created via the template.

If the broker logs have not loaded, and the Pod status shows **ErrImagePull** or **ImagePullBackOff**, your configuration was not able to directly pull the specified broker image from the Red Hat Container Registry. In this case, edit your deployment configuration to reference the correct broker image name and the image pull secret name associated with the account used for authentication in the Red Hat Container Registry. Then, you can import the broker image and start the broker. To do this, complete steps similar to those in [Deploying and starting the broker application](#).

5. Click the **Terminal** tab to access a shell where you can use the CLI to send some messages.

```
sh-4.2$ ./broker/bin/artemis producer --destination queue://demoQueue
Producer ActiveMQQueue[demoQueue], thread=0 Started to calculate elapsed time ...

Producer ActiveMQQueue[demoQueue], thread=0 Produced: 1000 messages
Producer ActiveMQQueue[demoQueue], thread=0 Elapsed time in second : 4 s
Producer ActiveMQQueue[demoQueue], thread=0 Elapsed time in milli second : 4584 milli
seconds

sh-4.2$ ./broker/bin/artemis consumer --destination queue://demoQueue
```

```

Consumer:: filter = null
Consumer ActiveMQQueue[demoQueue], thread=0 wait until 1000 messages are consumed
Received 1000
Consumer ActiveMQQueue[demoQueue], thread=0 Consumed: 1000 messages
Consumer ActiveMQQueue[demoQueue], thread=0 Consumer thread finished

```

Alternatively, use the OpenShift client to access the shell using the Pod name, as shown in the following example.

```

// Get the Pod names and internal IP Addresses
oc get pods -o wide

// Access a broker Pod by name
oc rsh <broker-pod-name>

```

- Now scale down the broker using the oc command.

```

$ oc scale statefulset broker-amq --replicas=0
statefulset "broker-amq" scaled

```

You can use the console to check that the Pod count is 0

- Now scale the broker back up to **1**.

```

$ oc scale statefulset broker-amq --replicas=1
statefulset "broker-amq" scaled

```

- Consume the messages again by using the terminal. For example:

```

sh-4.2$ broker/bin/artemis consumer --destination queue://demoQueue
Consumer:: filter = null
Consumer ActiveMQQueue[demoQueue], thread=0 wait until 1000 messages are consumed
Received 1000
Consumer ActiveMQQueue[demoQueue], thread=0 Consumed: 1000 messages
Consumer ActiveMQQueue[demoQueue], thread=0 Consumer thread finished

```

Additional resources

- For more information on managing stateful applications, see [StatefulSets](#) (external).

8.2.3. Creating a Route

Create a Route for the broker so that clients outside of OpenShift Container Platform can connect using SSL. By default, the broker protocols are available through the 61617/TCP port.



NOTE

If you scale your deployment up to multiple brokers in a cluster, you must manually create a Service and a Route for each broker, and then use each Service-and-Route combination to direct a given client to a given broker, or broker list. For an example of configuring multiple Services and Routes to connect clustered brokers to their own instances of the AMQ Broker management console, see [Creating Routes for the AMQ Broker management console](#).

Prerequisites

- Before creating an SSL Route, you should understand how external clients use this Route to connect to the broker. For more information, see [Creating an SSL Route](#).

Procedure

1. Click **Services** → **broker-amq-tcp-ssl**.
2. Click **Actions** → **Create a route**.
3. To display the TLS parameters, select the **Secure route** check box.
4. From the **TLS Termination** drop-down menu, choose **Passthrough**. This selection relays all communication to AMQ Broker without the OpenShift router decrypting and resending it.
5. To view the Route, click **Routes**. For example:

`https://broker-amq-tcp-amq-demo.router.default.svc.cluster.local`

This hostname will be used by external clients to connect to the broker using SSL with SNI.

Additional resources

- For more information on Routes in the OpenShift Container Platform, see [Routes](#).

8.3. DEPLOYING A SET OF CLUSTERED BROKERS

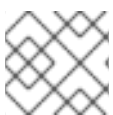
Deploy a clustered set of brokers where each broker runs in its own Pod.

8.3.1. Distributing messages

Message distribution is configured to use *ON_DEMAND*. This means that when messages arrive at a clustered broker, the messages are distributed in a round-robin fashion to any broker that has consumers.

This message distribution policy safeguards against messages getting stuck on a specific broker while a consumer, connected either directly or through the OpenShift router, is connected to a different broker.

The redistribution delay is zero by default. If a message is on a queue that has no consumers, it will be redistributed to another broker.



NOTE

When redistribution is enabled, messages can be delivered out of order.

8.3.2. Deploy the image and template

Prerequisites

- This procedure builds upon [Preparing a template-based broker deployment](#).
- Completion of the [Deploying a basic broker](#) tutorial is recommended.

Procedure

1. Navigate to the OpenShift web console and log in.
2. Select the **amq-demo** project space.
3. Click **Add to Project > Browse catalog** to list all of the default image streams and templates
4. Use the **Filter** search bar to limit the list to those that match **amq**. Click **See all** to show the desired application template.
5. Select the **amq-broker-78-persistence-clustered** template which is labeled **Red Hat AMQ Broker 7.8 (no SSL, clustered)**.
6. Set the following values in the configuration and click **create**.

Table 8.3. Example template

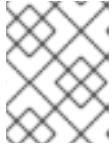
Environment variable	Display Name	Value	Description
AMQ_PROTOCOL	AMQ Protocols	openwire,amqp,stormq,mqtt,horntq	The protocols to be accepted by the broker
AMQ_QUEUES	Queues	demoQueue	Creates an anycast queue called demoQueue
AMQ_ADDRESSES	Addresses	demoTopic	Creates an address (or topic) called demoTopic. By default, this address has no assigned routing type.
VOLUME_CAPACITY	AMQ Volume Size	1Gi	The persistent volume size created for the journal
AMQ_CLUSTERED	Clustered	true	This needs to be true to ensure the brokers cluster
AMQ_CLUSTER_USER	cluster user	generated	The username the brokers use to connect with each other
AMQ_CLUSTER_PASSWORD	cluster password	generated	The password the brokers use to connect with each other
AMQ_USER	AMQ Username	amq-demo-user	The username the client uses
AMQ_PASSWORD	AMQ Password	password	The password the client uses with the username

8.3.3. Deploying the application

Once the application has been created it needs to be deployed. Deploying the application creates a Pod and starts the broker.

Procedure

1. Click **StatefulSets** in the OpenShift Container Platform web console.
2. Click the **broker-amq** deployment.
3. Click **Deploy** to deploy the application.



NOTE

The default number of replicas for a clustered template is 0. You should not see any Pods.

4. Scale up the Pods to three to create a cluster of brokers.

```
$ oc scale statefulset broker-amq --replicas=3
statefulset "broker-amq" scaled
```

5. Check that there are three Pods running.

```
$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
broker-amq-0  1/1     Running  0           33m
broker-amq-1  1/1     Running  0           33m
broker-amq-2  1/1     Running  0           29m
```

6. If the Pod status shows **ErrImagePull** or **ImagePullBackOff**, your deployment was not able to directly pull the specified broker image from the Red Hat Container Registry. In this case, edit your StatefulSet to reference the correct broker image name and the image pull secret name associated with the account used for authentication in the Red Hat Container Registry. Then, you can import the broker image and start the brokers. To do this, complete steps similar to those in [Deploying and starting the broker application](#).

7. Verify that the brokers have clustered with the new Pod by checking the logs.

```
$ oc logs broker-amq-2
```

This shows the logs of the new broker and an entry for a clustered bridge created between the brokers:

```
2018-08-29 07:43:55,779 INFO [org.apache.activemq.artemis.core.server] AMQ221027:
Bridge ClusterConnectionBridge@1b0e9e9d [name=$.artemis.internal.sf.my-
cluster.4333c830-ab5f-11e8-afb8-0a580a82006e,
queue=QueueImpl[name=$.artemis.internal.sf.my-cluster.4333c830-ab5f-11e8-afb8-
0a580a82006e, postOffice=PostOfficeImpl
[server=ActiveMQServerImpl::serverUUID=9cedb69d-ab5e-11e8-87a4-0a580a82006c],
temp=false]@5e0c0398 targetConnector=ServerLocatorImpl (identity=(Cluster-connection-
bridge::ClusterConnectionBridge@1b0e9e9d [name=$.artemis.internal.sf.my-
cluster.4333c830-ab5f-11e8-afb8-0a580a82006e,
queue=QueueImpl[name=$.artemis.internal.sf.my-cluster.4333c830-ab5f-11e8-afb8-
0a580a82006e, postOffice=PostOfficeImpl
```

```
[server=ActiveMQServerImpl::serverUUID=9cedb69d-ab5e-11e8-87a4-0a580a82006c],
temp=false]@5e0c0398 targetConnector=ServerLocatorImpl [initialConnectors=
[TransportConfiguration(name=artemis, factory=org-apache-activemq-artemis-core-remoting-
impl-netty-NettyConnectorFactory) ?port=61616&host=10-130-0-110],
discoveryGroupConfiguration=null]]::ClusterConnectionImpl@806813022[nodeUUID=9cedb69d
-ab5e-11e8-87a4-0a580a82006c, connector=TransportConfiguration(name=artemis,
factory=org-apache-activemq-artemis-core-remoting-impl-netty-NettyConnectorFactory) ?
port=61616&host=10-130-0-108, address=,
server=ActiveMQServerImpl::serverUUID=9cedb69d-ab5e-11e8-87a4-0a580a82006c]))
[initialConnectors=[TransportConfiguration(name=artemis, factory=org-apache-activemq-
artemis-core-remoting-impl-netty-NettyConnectorFactory) ?port=61616&host=10-130-0-110],
discoveryGroupConfiguration=null]] is connected
```

8.3.4. Creating Routes for the AMQ Broker management console

The clustering templates do not expose the AMQ Broker management console by default. This is because the OpenShift proxy performs load balancing across each broker in the cluster and it would not be possible to control which broker console is connected at a given time.

The following example procedure shows how to configure each broker in the cluster to connect to its own management console instance. You do this by creating a dedicated Service-and-Route combination for each broker Pod in the cluster.

Prerequisites

- You have already deployed a clustered set of brokers, where each broker runs in its own Pod. See [Deploying a set of clustered brokers](#).

Procedure

- Create a regular Service for each Pod in the cluster, using a StatefulSet selector to select between Pods. To do this, deploy a Service template, in **.yaml** format, that looks like the following:

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    description: 'Service for the management console of broker pod XXXX'
  labels:
    app: application2
    application: application2
    template: amq-broker-78-persistence-clustered
  name: amq2-amq-console-XXXX
  namespace: amq75-p-c-ssl-2
spec:
  ports:
    - name: console-jolokia
      port: 8161
      protocol: TCP
      targetPort: 8161
  selector:
    deploymentConfig: application2-amq
    statefulset.kubernetes.io/pod-name: application2-amq-XXXX
  type: ClusterIP
```

In the preceding template, replace **XXXX** with the ordinal value of the broker Pod you want to associate with the Service. For example, to associate the Service with the first Pod in the cluster, set **XXXX** to **0**. To associate the Service with the second Pod, set **XXXX** to **1**, and so on.

Save and deploy an instance of the template for each broker Pod in your cluster.



NOTE

In the example template shown above, the selector uses the Kubernetes-defined Pod name.

2. Create a Route for each broker Pod, so that the AMQ Broker management console can connect to the Pod.

Click **Routes** → **Create Route**.

The **Edit Route** page opens.

- a. In the **Services** drop-down menu, select the previously created broker Service that you want to associate the Route with, for example, **amq2-amq-console-0**.
- b. Set **Target Port** to **8161**, to enable access for the AMQ Broker management console.
- c. To display the TLS parameters, select the **Secure route** check box.
 - i. From the **TLS Termination** drop-down menu, choose **Passthrough**.
This selection relays all communication to AMQ Broker without the OpenShift router decrypting and resending it.
- d. Click **Create**.

When you create a Route associated with one of broker Pods, the resulting **.yaml** file includes lines that look like the following:

```
spec:
  host: amq2-amq-console-0-amq75-p-c-2.apps-ocp311.example.com
  port:
    targetPort: console-jolokia
  tls:
    termination: passthrough
  to:
    kind: Service
    name: amq2-amq-console-0
    weight: 100
  wildcardPolicy: None
```

3. To access the management console for a specific broker instance, copy the **host** URL shown above to a web browser.

Additional resources

- For more information on the clustering of brokers see [Configuring message redistribution](#).

8.4. DEPLOYING A SET OF CLUSTERED SSL BROKERS

Deploy a clustered set of brokers, where each broker runs in its own Pod and the broker is configured to accept connections using SSL.

8.4.1. Distributing messages

Message distribution is configured to use *ON_DEMAND*. This means that when messages arrive at a clustered broker, the messages are distributed in a round-robin fashion to any broker that has consumers.

This message distribution policy safeguards against messages getting stuck on a specific broker while a consumer, connected either directly or through the OpenShift router, is connected to a different broker.

The redistribution delay is non-zero by default. If a message is on a queue that has no consumers, it will be redistributed to another broker.



NOTE

When redistribution is enabled, messages can be delivered out of order.

8.4.2. Deploying the image and template

Prerequisites

- This procedure builds upon [Preparing a template-based broker deployment](#).
- Completion of the [Deploying a basic broker](#) example is recommended.

Procedure

1. Navigate to the OpenShift web console and log in.
2. Select the **amq-demo** project space.
3. Click **Add to Project** > **Browse catalog** to list all of the default image streams and templates.
4. Use the **Filter** search bar to limit the list to those that match **amq**. Click **See all** to show the desired application template.
5. Select the **amq-broker-78-persistence-clustered-ssl** template which is labeled **Red Hat AMQ Broker 7.8 (SSL, clustered)**.
6. Set the following values in the configuration and click **create**.

Table 8.4. Example template

Environment variable	Display Name	Value	Description
AMQ_PROTOCOL	AMQ Protocols	openwire,amqp,storm,mqtt,horntq	The protocols to be accepted by the broker
AMQ_QUEUES	Queues	demoQueue	Creates an anycast queue called demoQueue

Environment variable	Display Name	Value	Description
AMQ_ADDRESSES	Addresses	demoTopic	Creates an address (or topic) called demoTopic. By default, this address has no assigned routing type.
VOLUME_CAPACITY	AMQ Volume Size	1Gi	The persistent volume size created for the journal
AMQ_CLUSTERED	Clustered	true	This needs to be true to ensure the brokers cluster
AMQ_CLUSTER_USER	cluster user	generated	The username the brokers use to connect with each other
AMQ_CLUSTER_PASSWORD	cluster password	generated	The password the brokers use to connect with each other
AMQ_USER	AMQ Username	amq-demo-user	The username the client uses
AMQ_PASSWORD	AMQ Password	password	The password the client uses with the username
AMQ_TRUSTSTORE	Trust Store Filename	broker.ts	The SSL truststore file name
AMQ_TRUSTSTORE_PASSWORD	Truststore Password	password	The password used when creating the Truststore
AMQ_KEYSTORE	AMQ Keystore Filename	broker.ks	The SSL keystore file name
AMQ_KEYSTORE_PASSWORD	AMQ Keystore Password	password	The password used when creating the Keystore

8.4.3. Deploying the application

Deploy after creating the application. Deploying the application creates a Pod and starts the broker.

Procedure

1. Click **StatefulSets** in the OpenShift Container Platform web console.
2. Click the **broker-amq** deployment.
3. Click **Deploy** to deploy the application.

**NOTE**

The default number of replicas for a clustered template is **0**, so you will not see any Pods.

- Scale up the Pods to three to create a cluster of brokers.

```
$ oc scale statefulset broker-amq --replicas=3
statefulset "broker-amq" scaled
```

- Check that there are three Pods running.

```
$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
broker-amq-0  1/1     Running   0           33m
broker-amq-1  1/1     Running   0           33m
broker-amq-2  1/1     Running   0           29m
```

- If the Pod status shows **ErrImagePull** or **ImagePullBackOff**, your deployment was not able to directly pull the specified broker image from the Red Hat Container Registry. In this case, edit your StatefulSet to reference the correct broker image name and the image pull secret name associated with the account used for authentication in the Red Hat Container Registry. Then, you can import the broker image and start the brokers. To do this, complete steps similar to those in [Deploy and start the broker application](#).

- Verify the brokers have clustered with the new Pod by checking the logs.

```
$ oc logs broker-amq-2
```

This shows all the logs of the new broker and an entry for a clustered bridge created between the brokers, for example:

```
2018-08-29 07:43:55,779 INFO [org.apache.activemq.artemis.core.server] AMQ221027:
Bridge ClusterConnectionBridge@1b0e9e9d [name=$.artemis.internal.sf.my-
cluster.4333c830-ab5f-11e8-afb8-0a580a82006e,
queue=QueueImpl[name=$.artemis.internal.sf.my-cluster.4333c830-ab5f-11e8-afb8-
0a580a82006e, postOffice=PostOfficeImpl
[server=ActiveMQServerImpl::serverUUID=9cedb69d-ab5e-11e8-87a4-0a580a82006c],
temp=false]@5e0c0398 targetConnector=ServerLocatorImpl (identity=(Cluster-connection-
bridge::ClusterConnectionBridge@1b0e9e9d [name=$.artemis.internal.sf.my-
cluster.4333c830-ab5f-11e8-afb8-0a580a82006e,
queue=QueueImpl[name=$.artemis.internal.sf.my-cluster.4333c830-ab5f-11e8-afb8-
0a580a82006e, postOffice=PostOfficeImpl
[server=ActiveMQServerImpl::serverUUID=9cedb69d-ab5e-11e8-87a4-0a580a82006c],
temp=false]@5e0c0398 targetConnector=ServerLocatorImpl [initialConnectors=
[TransportConfiguration(name=artemis, factory=org-apache-activemq-artemis-core-remoting-
impl-netty-NettyConnectorFactory) ?port=61616&host=10-130-0-110],
discoveryGroupConfiguration=null]]::ClusterConnectionImpl@806813022[nodeUUID=9cedb69d
-ab5e-11e8-87a4-0a580a82006c, connector=TransportConfiguration(name=artemis,
factory=org-apache-activemq-artemis-core-remoting-impl-netty-NettyConnectorFactory) ?
port=61616&host=10-130-0-108, address=,
server=ActiveMQServerImpl::serverUUID=9cedb69d-ab5e-11e8-87a4-0a580a82006c]]))
```



```
[initialConnectors=[TransportConfiguration(name=artemis, factory=org-apache-activemq-
artemis-core-remoting-impl-netty-NettyConnectorFactory) ?port=61616&host=10-130-0-110],
discoveryGroupConfiguration=null]] is connected
```

Additional resources

- To learn how to configure each broker in the cluster to connect to its own management console instance, see [Creating Routes for the AMQ Broker management console](#).
- For more information about messaging in a broker cluster, see [Enabling Message Redistribution](#).

8.5. DEPLOYING A BROKER WITH CUSTOM CONFIGURATION

Deploy a broker with custom configuration. Although functionality can be obtained by using templates, broker configuration can be customized if needed.

Prerequisites

- This tutorial builds upon [Preparing a template-based broker deployment](#).
- Completion of the [Deploying a basic broker](#) tutorial is recommended.

8.5.1. Deploy the image and template

Procedure

1. Navigate to the OpenShift web console and log in.
2. Select the **amq-demo** project space.
3. Click **Add to Project** > **Browse catalog** to list all of the default image streams and templates.
4. Use the **Filter** search bar to limit results to those that match **amq**. Click **See all** to show the desired application template.
5. Select the **amq-broker-78-custom** template which is labeled **Red Hat AMQ Broker 7.8(Ephemeral, no SSL)**.
6. In the configuration, update **broker.xml** with the custom configuration you would like to use. Click **Create**.



NOTE

Use a text editor to create the broker's XML configuration. Then, cut and paste configuration details into the **broker.xml** field.



NOTE

OpenShift Container Platform does not use a **ConfigMap** object to store the custom configuration that you specify in the **broker.xml** field, as is common for many applications deployed on this platform. Instead, OpenShift temporarily stores the specified configuration in an environment variable, before transferring the configuration to a standalone file when the broker container starts.

8.5.2. Deploy the application

Once the application has been created it needs to be deployed. Deploying the application creates a Pod and starts the broker.

Procedure

1. Click **Deployments** in the OpenShift Container Platform web console.
2. Click the **broker-amq** deployment
3. Click **Deploy** to deploy the application.

8.6. BASIC SSL CLIENT EXAMPLE

Implement a client that sends and receives messages from a broker configured to use SSL, using the Qpid JMS client.

Prerequisites

- This tutorial builds upon [Preparing a template-based broker deployment](#).
- Completion of the [Deploying a basic broker with SSL](#) tutorial is recommended.
- [AMQ JMS Examples](#)

8.6.1. Configuring the client

Create a sample client that can be updated to connect to the SSL broker. The following procedure builds upon [AMQ JMS Examples](#).

Procedure

1. Add an entry into your `/etc/hosts` file to map the route name onto the IP address of the OpenShift cluster:

```
10.0.0.1 broker-amq-tcp-amq-demo.router.default.svc.cluster.local
```

2. Update the `jndi.properties` configuration file to use the route, truststore and keystore created previously, for example:

```
connectionfactory.myFactoryLookup = amqps://broker-amq-tcp-amq-  
demo.router.default.svc.cluster.local:8443?transport.keyStoreLocation=<keystore-  
path>client.ks&transport.keyStorePassword=password&transport.trustStoreLocation=  
<truststore-  
path>/client.ts&transport.trustStorePassword=password&transport.verifyHost=false
```

3. Update the `jndi.properties` configuration file to use the queue created earlier.

```
queue.myDestinationLookup = demoQueue
```

4. Execute the sender client to send a text message.
5. Execute the receiver client to receive the text message. You should see:

Received message: Message Text!

8.7. EXTERNAL CLIENTS USING SUB-DOMAINS EXAMPLE

Expose a clustered set of brokers through a node port and connect to it using the core JMS client. This enables clients to connect to a set of brokers which are configured using the **amq-broker-78-persistence-clustered-ssl** template.

8.7.1. Exposing the brokers

Configure the brokers so that the cluster of brokers are externally available and can be connected to directly, bypassing the OpenShift router. This is done by creating a route that exposes each pod using its own hostname.

Prerequisites

- [Deploying a set of clustered brokers](#)

Procedure

1. Choose **import YAML/JSON** from **Add to Project** drop down
2. Enter the following and click create.

```
apiVersion: v1
kind: Route
metadata:
  labels:
    app: broker-amq
    application: broker-amq
  name: tcp-ssl
spec:
  port:
    targetPort: ow-multi-ssl
  tls:
    termination: passthrough
  to:
    kind: Service
    name: broker-amq-headless
    weight: 100
  wildcardPolicy: Subdomain
  host: star.broker-ssl-amq-headless.amq-demo.svc
```



NOTE

The important configuration here is the wildcard policy of **Subdomain**. This allows each broker to be accessible through its own hostname.

8.7.2. Connecting the clients

Create a sample client that can be updated to connect to the SSL broker. The steps in this procedure build upon the [AMQ JMS Examples](#).

Procedure

1. Add entries into the `/etc/hosts` file to map the route name onto the actual IP addresses of the brokers:

```
10.0.0.1 broker-amq-0.broker-ssl-amq-headless.amq-demo.svc broker-amq-1.broker-ssl-
amq-headless.amq-demo.svc broker-amq-2.broker-ssl-amq-headless.amq-demo.svc
```

2. Update the `jndi.properties` configuration file to use the route, truststore, and keystore created previously, for example:

```
connectionfactory.myFactoryLookup = amqps://broker-amq-0.broker-ssl-amq-headless.amq-
demo.svc:443?transport.keyStoreLocation=/home/ataylor/projects/jboss-amq-7-broker-
openshift-
image/client.ks&transport.keyStorePassword=password&transport.trustStoreLocation=/home/at
aylor/projects/jboss-amq-7-broker-openshift-
image/client.ts&transport.trustStorePassword=password&transport.verifyHost=false
```

3. Update the `jndi.properties` configuration file to use the queue created earlier.

```
queue.myDestinationLookup = demoQueue
```

4. Execute the sender client code to send a text message.
5. Execute the receiver client code to receive the text message. You should see:

```
Received message: Message Text!
```

Additional resources

- For more information on using the AMQ JMS client, see [AMQ JMS Examples](#).

8.8. EXTERNAL CLIENTS USING PORT BINDING EXAMPLE

Expose a clustered set of brokers through a NodePort and connect to it using the core JMS client. This enables clients that do not support SNI or SSL. It is used with clusters configured using the **amq-broker-78-persistence-clustered** template.

8.8.1. Exposing the brokers

Configure the brokers so that the cluster of brokers are externally available and can be connected to directly, bypassing the OpenShift router. This is done by creating a service that uses a NodePort to load balance around the clusters.

Prerequisites

- [Deploying a set of clustered brokers](#)

Procedure

1. Choose **import YAML/JSON** from **Add to Project** drop down.
2. Enter the following and click create.

```

apiVersion: v1
kind: Service
metadata:
  annotations:
    description: The broker's OpenWire port.
    service.alpha.openshift.io/dependencies: >-
      [{"name": "broker-amq-amqp", "kind": "Service"}, {"name":
        "broker-amq-mqtt", "kind": "Service"}, {"name": "broker-amq-stomp", "kind":
        "Service"}]
    creationTimestamp: '2018-08-29T14:46:33Z'
  labels:
    application: broker
    template: amq-broker-78-statefulset-clustered
  name: broker-external-tcp
  namespace: amq-demo
  resourceVersion: '2450312'
  selfLink: /api/v1/namespaces/amq-demo/services/broker-amq-tcp
  uid: 52631fa0-ab9a-11e8-9380-c280f77be0d0
spec:
  externalTrafficPolicy: Cluster
  ports:
    - nodePort: 30001
      port: 61616
      protocol: TCP
      targetPort: 61616
  selector:
    deploymentConfig: broker-amq
  sessionAffinity: None
  type: NodePort
status:
  loadBalancer: {}

```

**NOTE**

The NodePort configuration is important. The NodePort is the port in which the client will access the brokers and the type is **NodePort**.

8.8.2. Connecting the clients

Create consumers that are round-robbined around the brokers in the cluster using the AMQ broker CLI.

Procedure

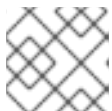
1. In a terminal create a consumer and attach it to the IP address where OpenShift is running.

```

artemis consumer --url tcp://<IP_ADDRESS>:30001 --message-count 100 --destination
queue://demoQueue

```

2. Repeat step 1 twice to start another two consumers.

**NOTE**

You should now have three consumers load balanced across the three brokers.

3. Create a producer to send messages.

```
artemis producer --url tcp://<IP_ADDRESS>:30001 --message-count 300 --destination  
queue://demoQueue
```

4. Verify each consumer receives messages.

```
Consumer:: filter = null  
Consumer ActiveMQQueue[demoQueue], thread=0 wait until 100 messages are consumed  
Consumer ActiveMQQueue[demoQueue], thread=0 Consumed: 100 messages  
Consumer ActiveMQQueue[demoQueue], thread=0 Consumer thread finished
```

CHAPTER 9. UPGRADING A TEMPLATE-BASED BROKER DEPLOYMENT

The following procedures show how to upgrade the broker container image for a deployment that is based on application templates.



NOTE

- To upgrade an existing AMQ Broker deployment on OpenShift Container Platform 3.11 to run on OpenShift Container Platform 4.5 or later, you must first upgrade your OpenShift Container Platform installation, before performing a clean installation of AMQ Broker that matches your existing deployment. To perform a clean AMQ Broker installation, use one of these methods:
 - [Deploying AMQ Broker on OpenShift Container Platform using an Operator \(Recommended\)](#)
 - [Deploying AMQ Broker on OpenShift Container Platform using Application Templates](#)
- The procedures show how to manually upgrade your image specifications between *minor* versions (for example, from **7.x** to **7.y**). If you use a floating tag such as **7.y** in your image specification, your deployment *automatically* pulls and uses new *micro* image versions (that is, **7.y-z**) when they become available from Red Hat, provided that the **imagePullPolicy** attribute in your StatefulSet or DeploymentConfig is set to **Always**.
For example, suppose that the **image** attribute of your deployment specifies a floating tag of **7.8**. If the deployment currently uses minor version **7.8-5**, and a newer minor version, **7.8-6**, becomes available in the registry, then your deployment automatically pulls and uses the new minor version. To use the new image, each broker Pod in the deployment is restarted. If you have multiple brokers in your deployment, brokers Pods are restarted one at a time.

9.1. UPGRADING NON-PERSISTENT BROKER DEPLOYMENTS

This procedure shows you how to upgrade a non-persistent broker deployment. The non-persistent broker templates in the OpenShift Container Platform service catalog have labels that resemble the following:

- Red Hat AMQ Broker 7.x (Ephemeral, no SSL)
- Red Hat AMQ Broker 7.x (Ephemeral, with SSL)
- Red Hat AMQ Broker 7.x (Custom Config, Ephemeral, no SSL)

Prerequisites

- Starting in AMQ Broker 7.3, you use a new version of the Red Hat Ecosystem Catalog to access container images. This new version of the registry requires you to become an authenticated user before you can access images and pull them into an OpenShift project. Before following the procedure in this section, you must first complete the steps described in [Red Hat Container Registry Authentication](#).

Procedure

1. Navigate to the OpenShift Container Platform web console and log in.
2. Click the project in which you want to upgrade a non-persistent broker deployment.
3. Select the DeploymentConfig (DC) corresponding to your broker deployment.
 - a. In OpenShift Container Platform 4.5 or later, click **Workloads** → **DeploymentConfigs**.
 - b. In OpenShift Container Platform 3.11, click **Applications** → **Deployments**. Within your broker deployment, click the **Configuration** tab.
4. From the **Actions** menu, click **Edit DeploymentConfig** (OpenShift Container Platform 4.5 or later) or **Edit YAML** (OpenShift Container Platform 3.11).
The **YAML** tab of the DeploymentConfig opens, with the **.yaml** file in an editable mode.
5. Edit the **image** attribute to specify the latest AMQ Broker 7.8 container image, **registry.redhat.io/amq7/amq-broker:7.8**.
6. Add the **imagePullSecrets** attribute to specify the image pull secret associated with the account used for authentication in the Red Hat Container Registry.
Changes based on the previous two steps are shown in the example below:

```

...
spec:
  containers:
    image: 'registry.redhat.io/amq7/amq-broker:7.8'
..
imagePullSecrets:
- name: {PULL-SECRET-NAME}

```



NOTE

In AMQ Broker, container image tags increment by **1** for each new version of the container image added to the Red Hat image registry, for example, 7.8-1, 7.8-2, and so on. If you specify a tag name without a final digit (**7.8**, for example), this tag is known as a *floating tag*. When you specify a floating tag, OpenShift Container Platform automatically identifies the most recent available image (that is, the image tag with the highest final number) and uses this image to upgrade your broker deployment.

7. Click **Save**.
If a newer broker image than the one currently installed is available from Red Hat, OpenShift Container Platform upgrades your broker deployment. To do this, OpenShift Container Platform stops the existing broker Pod and then starts a new Pod that uses the new image.

9.2. UPGRADING PERSISTENT BROKER DEPLOYMENTS

This procedure shows you how to upgrade a persistent broker deployment. The persistent broker templates in the OpenShift Container Platform service catalog have labels that resemble the following:

- Red Hat AMQ Broker 7.x (Persistence, clustered, no SSL)
- Red Hat AMQ Broker 7.x (Persistence, clustered, with SSL)
- Red Hat AMQ Broker 7.x (Persistence, with SSL)

Prerequisites

- Starting in AMQ Broker 7.3, you use a new version of the Red Hat Ecosystem Catalog to access container images. This new version of the registry requires you to become an authenticated user before you can access images and pull them into an OpenShift project. Before following the procedure in this section, you must first complete the steps described in [Red Hat Container Registry Authentication](#).

Procedure

- Navigate to the OpenShift Container Platform web console and log in.
- Click the project in which you want to upgrade a persistent broker deployment.
- Select the StatefulSet (SS) corresponding to your broker deployment.
 - In OpenShift Container Platform 4.5 or later, click **Workloads** → **StatefulSets**.
 - In OpenShift Container Platform 3.11, click **Applications** → **StatefulSets**.
- From the **Actions** menu, click **Edit StatefulSet** (OpenShift Container Platform 4.5 or later) or **Edit YAML** (OpenShift Container Platform 3.11).
The **YAML** tab of the StatefulSet opens, with the **.yaml** file in an editable mode.
- To prepare your broker deployment for upgrade, scale the deployment down to zero brokers.
 - If the **replicas** attribute is currently set to **1** or greater, set it to **0**.
 - Click **Save**.
- When all broker Pods have shut down, edit the StatefulSet **.yaml** file again. Edit the **image** attribute to specify the latest AMQ Broker 7.8 container image, **registry.redhat.io/amq7/amq-broker:7.8**.
- Add the **imagePullSecrets** attribute to specify the image pull secret associated with the account used for authentication in the Red Hat Container Registry.
Changes based on the previous two steps are shown in the example below:

```

...
spec:
  containers:
    image: 'registry.redhat.io/amq7/amq-broker:7.8'
  ..
  imagePullSecrets:
    - name: {PULL-SECRET-NAME}

```

- Set the **replicas** attribute back to the original value.
- Click **Save**.
If a newer broker image than the one currently installed is available from Red Hat, OpenShift Container Platform upgrades your broker deployment. To do this, OpenShift Container Platform restarts the broker Pod.

CHAPTER 10. MONITORING YOUR BROKERS

10.1. VIEWING BROKERS IN FUSE CONSOLE

You can configure an Operator-based broker deployment to use Fuse Console for OpenShift instead of the AMQ Management Console. When you have configured your broker deployment appropriately, Fuse Console discovers the brokers and displays them on a dedicated **Artemis** tab. You can view the same broker runtime data that you do in the AMQ Management Console. You can also perform the same basic management operations, such as creating addresses and queues.

The following procedure describes how to configure the Custom Resource (CR) instance for a broker deployment to enable Fuse Console for OpenShift to discover and display brokers in the deployment.



IMPORTANT

Viewing brokers from Fuse Console is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process. For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

Prerequisites

- Fuse Console for OpenShift must be deployed to an OCP cluster, or to a specific namespace on that cluster. If you have deployed the console to a specific namespace, your broker deployment must be in the **same** namespace, to enable the console to discover the brokers. Otherwise, it is sufficient for Fuse Console and the brokers to be deployed on the same OCP cluster. For more information on installing Fuse Online on OCP, see [Installing and Operating Fuse Online on OpenShift Container Platform](#).
- You must have already created a broker deployment. For example, to learn how to use a Custom Resource (CR) instance to create a basic Operator-based deployment, see [Section 3.4.1, "Deploying a basic broker instance"](#).

Procedure

1. Open the CR instance that you used for your broker deployment. For example, the CR for a basic deployment might resemble the following:

```
apiVersion: broker.amq.io/v2alpha4
kind: ActiveMQArtemis
metadata:
  name: ex-aao
  application: ex-aao-app
spec:
  version: 7.8.0
  deploymentPlan:
    size: 4
    image: registry.redhat.io/amq7/amq-broker:7.8
  ...
```

- In the **deploymentPlan** section, add the **jolokiaAgentEnabled** and **managementRBACEnabled** properties and specify values, as shown below.

```

apiVersion: broker.amq.io/v2alpha4
kind: ActiveMQArtemis
metadata:
  name: ex-aa0
  application: ex-aa0-app
spec:
  version: 7.8.0
  deploymentPlan:
    size: 4
    image: registry.redhat.io/amq7/amq-broker:7.8
    ...
    jolokiaAgentEnabled: true
    managementRBACEnabled: false

```

jolokiaAgentEnabled

Specifies whether Fuse Console can discover and display runtime data for the brokers in the deployment. To use Fuse Console, set the value to **true**.

managementRBACEnabled

Specifies whether role-based access control (RBAC) is enabled for the brokers in the deployment. You **must** set the value to **false** to use Fuse Console because Fuse Console uses its own role-based access control.



IMPORTANT

If you set the value of **managementRBACEnabled** to **false** to enable use of Fuse Console, management MBeans for the brokers no longer require authorization. You **should not** use the AMQ management console while **managementRBACEnabled** is set to **false** because this potentially exposes all management operations on the brokers to unauthorized use.

- Save the CR instance.
- Switch to the project in which you previously created your broker deployment.

```
$ oc project <project-name>
```

- At the command line, apply the change.

```
$ oc apply -f <path/to/custom-resource-instance>.yaml
```

- In Fuse Console, to view Fuse applications, click the **Online** tab. To view running brokers, in the left navigation menu, click **Artemis**.

Additional resources

- For more information about using Fuse Console for OpenShift, see [Monitoring and managing Red Hat Fuse applications on OpenShift](#).

- To learn about using AMQ Management Console to view and manage brokers in the same way that you can in Fuse Console, see [Managing brokers using AMQ Management Console](#).

10.2. MONITORING BROKER RUNTIME METRICS USING PROMETHEUS

The sections that follow describe how to configure the Prometheus metrics plugin for AMQ Broker on OpenShift Container Platform. You can use the plugin to monitor and store broker runtime metrics. You might also use a graphical tool such as Grafana to configure more advanced visualizations and dashboards of the data that the Prometheus plugin collects.



NOTE

The Prometheus metrics plugin enables you to collect and export broker metrics in Prometheus **format**. However, Red Hat **does not** provide support for installation or configuration of Prometheus itself, nor of visualization tools such as Grafana. If you require support with installing, configuring, or running Prometheus or Grafana, visit the product websites for resources such as community support and documentation.

10.2.1. Metrics overview

To monitor the health and performance of your broker instances, you can use the Prometheus plugin for AMQ Broker to monitor and store broker runtime metrics. The AMQ Broker Prometheus plugin exports the broker runtime metrics to Prometheus format, enabling you to use Prometheus itself to visualize and run queries on the data.

You can also use a graphical tool, such as Grafana, to configure more advanced visualizations and dashboards for the metrics that the Prometheus plugin collects.

The metrics that the plugin exports to Prometheus format are described below.

Broker metrics

artemis_address_memory_usage

Number of bytes used by all addresses on this broker for in-memory messages.

artemis_address_memory_usage_percentage

Memory used by all the addresses on this broker as a percentage of the **global-max-size** parameter.

artemis_connection_count

Number of clients connected to this broker.

artemis_total_connection_count

Number of clients that have connected to this broker since it was started.

Address metrics

artemis_routed_message_count

Number of messages routed to one or more queue bindings.

artemis_unrouted_message_count

Number of messages *not* routed to any queue bindings.

Queue metrics

artemis_consumer_count

Number of clients consuming messages from a given queue.

artemis_delivering_durable_message_count

Number of durable messages that a given queue is currently delivering to consumers.

artemis_delivering_durable_persistent_size

Persistent size of durable messages that a given queue is currently delivering to consumers.

artemis_delivering_message_count

Number of messages that a given queue is currently delivering to consumers.

artemis_delivering_persistent_size

Persistent size of messages that a given queue is currently delivering to consumers.

artemis_durable_message_count

Number of durable messages currently in a given queue. This includes scheduled, paged, and in-delivery messages.

artemis_durable_persistent_size

Persistent size of durable messages currently in a given queue. This includes scheduled, paged, and in-delivery messages.

artemis_messages_acknowledged

Number of messages acknowledged from a given queue since the queue was created.

artemis_messages_added

Number of messages added to a given queue since the queue was created.

artemis_message_count

Number of messages currently in a given queue. This includes scheduled, paged, and in-delivery messages.

artemis_messages_killed

Number of messages removed from a given queue since the queue was created. The broker kills a message when the message exceeds the configured maximum number of delivery attempts.

artemis_messages_expired

Number of messages expired from a given queue since the queue was created.

artemis_persistent_size

Persistent size of all messages (both durable and non-durable) currently in a given queue. This includes scheduled, paged, and in-delivery messages.

artemis_scheduled_durable_message_count

Number of durable, scheduled messages in a given queue.

artemis_scheduled_durable_persistent_size

Persistent size of durable, scheduled messages in a given queue.

artemis_scheduled_message_count

Number of scheduled messages in a given queue.

artemis_scheduled_persistent_size

Persistent size of scheduled messages in a given queue.

For higher-level broker metrics that are not listed above, you can calculate these by aggregating lower-level metrics. For example, to calculate total message count, you can aggregate the

artemis_message_count metrics from all queues in your broker deployment.

For an on-premise deployment of AMQ Broker, metrics for the Java Virtual Machine (JVM) hosting the broker are also exported to Prometheus format. This does not apply to a deployment of AMQ Broker on OpenShift Container Platform.

10.2.2. Enabling the Prometheus plugin for a running broker deployment

This procedure shows how to enable the Prometheus plugin for a broker Pod in a given deployment.

Prerequisites

- You can enable the Prometheus plugin for a broker Pod created with application templates or with the AMQ Broker Operator. However, your deployed broker must use the broker container image for AMQ Broker 7.5 or later. For more information about ensuring that your broker deployment uses the latest broker container image, see [Chapter 9, Upgrading a template-based broker deployment](#).

Procedure

1. Log in to the OpenShift Container Platform web console with administrator privileges for the project that contains your broker deployment.
2. In the web console, click **Home** → **Projects** (OpenShift Container Platform 4.5 or later) or the drop-down list in the top-left corner (OpenShift Container Platform 3.11). Choose the project that contains your broker deployment.
3. To see the StatefulSets or DeploymentConfigs in your project, click:
 - a. **Workloads** → **StatefulSets** or **Workloads** → **DeploymentConfigs** (OpenShift Container Platform 4.5 or later).
 - b. **Applications** → **StatefulSets** or **Applications** → **Deployments** (OpenShift Container Platform 3.11).
4. Click the StatefulSet or DeploymentConfig that corresponds to your broker deployment.
5. To access the environment variables for your broker deployment, click the **Environment** tab.
6. Add a new environment variable, **AMQ_ENABLE_METRICS_PLUGIN**. Set the value of the variable to **true**.

When you set the **AMQ_ENABLE_METRICS_PLUGIN** environment variable, OpenShift restarts each broker Pod in the StatefulSet or DeploymentConfig. When there are multiple Pods in the deployment, OpenShift restarts each Pod in turn. When each broker Pod restarts, the Prometheus plugin for that broker starts to gather broker runtime metrics.



NOTE

The **AMQ_ENABLE_METRICS_PLUGIN** environment variable is included by default in the application templates for AMQ Broker 7.5 or later. To enable the plugin for each broker in a *new* template-based deployment, ensure that the value of **AMQ_ENABLE_METRICS_PLUGIN** is set to **true** when deploying the application template.

Additional resources

- For information about installing the latest application templates, see [Section 7.2, “Installing the image streams and application templates”](#)

10.2.3. Accessing Prometheus metrics for a running broker Pod

This procedure shows how to access Prometheus metrics for a running broker Pod.

Prerequisites

- You must have already enabled the Prometheus plugin for your broker Pod. See [Section 10.2.2, “Enabling the Prometheus plugin for a running broker deployment”](#).

Procedure

1. For the broker Pod whose metrics you want to access, you need to identify the Route you previously created to connect the Pod to the AMQ Broker management console. The Route name forms part of the URL needed to access the metrics.
 - a. Click **Networking** → **Routes** (OpenShift Container Platform 4.5 or later) or **Applications** → **Routes** (OpenShift Container Platform 3.11).
 - b. For your chosen broker Pod, identify the Route created to connect the Pod to the AMQ Broker management console. Under **Hostname**, note the complete URL that is shown. For example:

```
http://rte-console-access-pod1.openshiftdomain
```

2. To access Prometheus metrics, in a web browser, enter the previously noted Route name appended with **“/metrics”**. For example:

```
http://rte-console-access-pod1.openshiftdomain/metrics
```



NOTE

If your console configuration does not use SSL, specify **http** in the URL. In this case, DNS resolution of the host name directs traffic to port 80 of the OpenShift router. If your console configuration uses SSL, specify **https** in the URL. In this case, your browser defaults to port 443 of the OpenShift router. This enables a successful connection to the console if the OpenShift router also uses port 443 for SSL traffic, which the router does by default.

10.3. MONITORING BROKER RUNTIME DATA USING JMX

This example shows how to monitor a broker using the Jolokia REST interface to JMX.

Prerequisites

- This example builds upon [Preparing a template-based broker deployment](#).
- Completion of [Deploying a basic broker](#) is recommended.

Procedure

1. Get the list of running pods:

```
$ oc get pods
```

```
NAME           READY   STATUS    RESTARTS   AGE
broker-amq-1-ftqmk 1/1     Running  0          14d
```

2. Run the **oc logs** command:

```
$ oc logs -f broker-amq-1-ftqmk
```

```
Running /amq-broker-71-openshift image, version 1.3-5
INFO: Loading '/opt/amq/bin/env'
INFO: Using java '/usr/lib/jvm/java-1.8.0/bin/java'
INFO: Starting in foreground, this is just for debugging purposes (stop process by pressing CTRL+C)
...
INFO | Listening for connections at: tcp://broker-amq-1-ftqmk:61616?
maximumConnections=1000&wireFormat.maxFrameSize=104857600
INFO | Connector openwire started
INFO | Starting OpenShift discovery agent for service broker-amq-tcp transport type tcp
INFO | Network Connector DiscoveryNetworkConnector:NC:BrokerService[broker-amq-1-
ftqmk] started
INFO | Apache ActiveMQ 5.11.0.redhat-621084 (broker-amq-1-ftqmk, ID:broker-amq-1-
ftqmk-41433-1491445582960-0:1) started
INFO | For help or more information please see: http://activemq.apache.org
WARN | Store limit is 102400 mb (current store usage is 0 mb). The data directory:
/opt/amq/data/kahadb only has 9684 mb of usable space - resetting to maximum available
disk space: 9684 mb
WARN | Temporary Store limit is 51200 mb, whilst the temporary data directory:
/opt/amq/data/broker-amq-1-ftqmk/tmp_storage only has 9684 mb of usable space - resetting
to maximum available 9684 mb.
```

3. Run your query to monitor your broker for **MaxConsumers**:

```
$ curl -k -u admin:admin http://console-broker.amq-
demo.apps.example.com/console/jolokia/read/org.apache.activemq.artemis:broker=%22broker
%22,component=addresses,address=%22TESTQUEUE%22,subcomponent=queues,routing-
type=%22anycast%22,queue=%22TESTQUEUE%22/MaxConsumers
```

```
{"request":
{"mbean":"org.apache.activemq.artemis:address=\"TESTQUEUE\",broker=\"broker\",compone
nt=addresses,queue=\"TESTQUEUE\",routing-
type=\"anycast\",subcomponent=queues\",\"attribute\":\"MaxConsumers\",\"type\":\"read\"},\"value\":-
1,\"timestamp\":1528297825,\"status\":200}
```


CHAPTER 11. REFERENCE

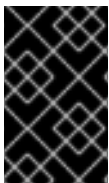
11.1. CUSTOM RESOURCE CONFIGURATION REFERENCE

A Custom Resource Definition (CRD) is a schema of configuration items for a custom OpenShift object deployed with an Operator. By deploying a corresponding Custom Resource (CR) instance, you specify values for configuration items shown in the CRD.

The following sub-sections detail the configuration items that you can set in Custom Resource instances based on the main broker and addressing CRDs.

11.1.1. Broker Custom Resource configuration reference

A CR instance based on the main broker CRD enables you to configure brokers for deployment in an OpenShift project. The following table describes the items that you can configure in the CR instance.



IMPORTANT

Configuration items marked with an asterisk (*) are required in any corresponding Custom Resource (CR) that you deploy. If you do not explicitly specify a value for a non-required item, the configuration uses the default value.

Entry	Sub-entry	Description and usage
adminUser*		<p>Administrator user name required for connecting to the broker and management console.</p> <p>If you do not specify a value, the value is automatically generated and stored in a secret. The default secret name has a format of <custom_resource_name>credentials-secret. For example, my-broker-deployment-credentials-secret.</p> <p>Type: string</p> <p>Example: my-user</p> <p>Default value: Automatically-generated, random value</p>

Entry	Sub-entry	Description and usage
adminPassword*		<p>Administrator password required for connecting to the broker and management console.</p> <p>If you do not specify a value, the value is automatically generated and stored in a secret. The default secret name has a format of <custom_resource_name>credentials-secret. For example, my-broker-deployment-credentials-secret.</p> <p>Type: string</p> <p>Example: my-password</p> <p>Default value: Automatically-generated, random value</p>
deploymentPlan*		Broker deployment configuration

Entry	Sub-entry	Description and usage
	image*	<p>Full path of the broker container image used for each broker in the deployment.</p> <p>You do not need to explicitly specify a value for image in your CR. The default value of placeholder indicates that the Operator has not yet determined the appropriate image to use.</p> <p>To learn how the Operator chooses a broker container image to use, see Section 2.4, "How the Operator chooses container images".</p> <p>Type: string</p> <p>Example: registry.redhat.io/amq7/a mq- broker@sha256:4d60775c d384067147ab105f41855b 5a7af855c4d9cbef1d4dea 566cbe214558</p> <p>Default value: placeholder</p>
	size*	<p>Number of broker Pods to create in the deployment.</p> <p>If you specify a value of 2 or greater, your broker deployment is clustered by default. The cluster user name and password are automatically generated and stored in the same secret as adminUser and adminPassword, by default.</p> <p>Type: int</p> <p>Example: 1</p> <p>Default value: 2</p>

Entry	Sub-entry	Description and usage
	requireLogin	<p>Specify whether login credentials are required to connect to the broker.</p> <p>Type: Boolean</p> <p>Example: false</p> <p>Default value: true</p>
	persistenceEnabled	<p>Specify whether to use journal storage for each broker Pod in the deployment. If set to true, each broker Pod requires an available Persistent Volume (PV) that the Operator can claim using a Persistent Volume Claim (PVC).</p> <p>Type: Boolean</p> <p>Example: false</p> <p>Default value: true</p>

Entry	Sub-entry	Description and usage
	initImage	<p>Init Container image used to configure the broker.</p> <p>You do not need to explicitly specify a value for initImage in your CR, unless you want to provide a custom image.</p> <p>To learn how the Operator chooses a built-in Init Container image to use, see Section 2.4, “How the Operator chooses container images”.</p> <p>To learn how to specify a <i>custom</i> Init Container image, see Section 4.5, “Specifying a custom Init Container image”.</p> <p>Type: string</p> <p>Example: registry.redhat.io/amq7/amq-broker-init-rhel7@sha256:f7482d07ecaa78d34c37981447536e6f73d4013ec0c64ff787161a75e4ca3567</p> <p>Default value: Not specified</p>
	journalType	<p>Specify whether to use asynchronous I/O (AIO) or non-blocking I/O (NIO).</p> <p>Type: string</p> <p>Example: aio</p> <p>Default value: nio</p>

Entry	Sub-entry	Description and usage
	messageMigration	<p>When a broker Pod shuts down due to a failure or intentional scaledown of the broker deployment, specify whether to migrate messages to another broker Pod that is still running in the broker cluster.</p> <p>Type: Boolean</p> <p>Example: false</p> <p>Default value: true</p>
	resources.limits.cpu	<p>Maximum amount of host-node CPU, in millicores, that each broker container running in a Pod in a deployment can consume.</p> <p>Type: string</p> <p>Example: "500m"</p> <p>Default value: Uses the same default value that your version of OpenShift Container Platform uses. Consult a cluster administrator.</p>
	resources.limits.memory	<p>Maximum amount of host-node memory, in bytes, that each broker container running in a Pod in a deployment can consume. Supports byte notation (for example, K, M, G), or the binary equivalents (Ki, Mi, Gi).</p> <p>Type: string</p> <p>Example: "1024M"</p> <p>Default value: Uses the same default value that your version of OpenShift Container Platform uses. Consult a cluster administrator.</p>

Entry	Sub-entry	Description and usage
	resources.requests.cpu	<p>Amount of host-node CPU, in millicores, that each broker container running in a Pod in a deployment explicitly requests.</p> <p>Type: string</p> <p>Example: "250m"</p> <p>Default value: Uses the same default value that your version of OpenShift Container Platform uses. Consult a cluster administrator.</p>
	resources.requests.memory	<p>Amount of host-node memory, in bytes, that each broker container running in a Pod in a deployment explicitly requests. Supports byte notation (for example, K, M, G), or the binary equivalents (Ki, Mi, Gi).</p> <p>Type: string</p> <p>Example: "512M"</p> <p>Default value: Uses the same default value that your version of OpenShift Container Platform uses. Consult a cluster administrator.</p>

Entry	Sub-entry	Description and usage
	storage.size	<p>Size, in bytes, of the Persistent Volume Claim (PVC) that each broker in a deployment requires for persistent storage. This property applies only when persistenceEnabled is set to true. The value that you specify must include a unit. Supports byte notation (for example, K, M, G), or the binary equivalents (Ki, Mi, Gi).</p> <p>Type: string</p> <p>Example: 4Gi</p> <p>Default value: 2Gi</p>
	jolokiaAgentEnabled	<p>Specifies whether the Jolokia JVM Agent is enabled for the brokers in the deployment. If the value of this property is set to true, Fuse Console can discover and display runtime data for the brokers.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: false</p>

Entry	Sub-entry	Description and usage
	managementRBACEnabled	<p>Specifies whether role-based access control (RBAC) is enabled for the brokers in the deployment. To use Fuse Console, you must set the value to false, because Fuse Console uses its own role-based access control.</p> <p>Type: Boolean</p> <p>Example: false</p> <p>Default value: true</p>
console		Configuration of broker management console.
	expose	<p>Specify whether to expose the management console port for each broker in a deployment.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: false</p>
	sslEnabled	<p>Specify whether to use SSL on the management console port.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: false</p>

Entry	Sub-entry	Description and usage
	sslSecret	<p>Secret where broker key store, trust store, and their corresponding passwords (all Base64-encoded) are stored. If you do not specify a value for sslSecret, the console uses a default secret name. The default secret name is in the form of <custom_resource_name>-console-secret.</p> <p>Type: string</p> <p>Example: my-broker-deployment-console-secret</p> <p>Default value: Not specified</p>
	useClientAuth	<p>Specify whether the management console requires client authorization.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: false</p>
acceptors.acceptor		<p>A single acceptor configuration instance.</p>
	name*	<p>Name of acceptor.</p> <p>Type: string</p> <p>Example: my-acceptor</p> <p>Default value: Not applicable</p>

Entry	Sub-entry	Description and usage
	port	<p>Port number to use for the acceptor instance.</p> <p>Type: int</p> <p>Example: 5672</p> <p>Default value: 61626 for the first acceptor that you define. The default value then increments by 10 for every subsequent acceptor that you define.</p>
	protocols	<p>Messaging protocols to be enabled on the acceptor instance.</p> <p>Type: string</p> <p>Example: amqp,core</p> <p>Default value: all</p>
	sslEnabled	<p>Specify whether SSL is enabled on the acceptor port. If set to true, look in the secret name specified in sslSecret for the credentials required by TLS/SSL.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: false</p>

Entry	Sub-entry	Description and usage
	sslSecret	<p>Secret where broker key store, trust store, and their corresponding passwords (all Base64-encoded) are stored.</p> <p>If you do not specify a custom secret name for sslSecret, the acceptor assumes a default secret name. The default secret name has a format of <custom_resource_name><acceptor_name>-secret.</p> <p>You must always create this secret yourself, even when the acceptor assumes a default name.</p> <p>Type: string</p> <p>Example: my-broker-deployment-my-acceptor-secret</p> <p>Default value: <custom_resource_name>-<acceptor_name>-secret</p>

Entry	Sub-entry	Description and usage
	enabledCipherSuites	<p>Comma-separated list of cipher suites to use for TLS/SSL communication.</p> <p>Specify the most secure cipher suite(s) supported by your client application. If you use a comma-separated list to specify a set of cipher suites that is common to both the broker and the client, or you do not specify any cipher suites, the broker and client mutually negotiate a cipher suite to use. If you do not know which cipher suites to specify, it is recommended that you first establish a broker-client connection with your client running in debug mode, to verify the cipher suites that are common to both the broker and the client. Then, configure enabledCipherSuites on the broker.</p> <p>Type: string</p> <p>Default value: Not specified</p>
	enabledProtocols	<p>Comma-separated list of protocols to use for TLS/SSL communication.</p> <p>Type: string</p> <p>Example: TLSv1,TLSv1.1,TLSv1.2</p> <p>Default value: Not specified</p>

Entry	Sub-entry	Description and usage
	needClientAuth	<p>Specify whether the broker informs clients that two-way TLS is required on the acceptor. This property overrides wantClientAuth.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: Not specified</p>
	wantClientAuth	<p>Specify whether the broker informs clients that two-way TLS is <i>requested</i> on the acceptor, but not required. This property is overridden by needClientAuth.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: Not specified</p>
	verifyHost	<p>Specify whether to compare the Common Name (CN) of a client's certificate to its host name, to verify that they match. This option applies only when two-way TLS is used.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: Not specified</p>

Entry	Sub-entry	Description and usage
	sslProvider	<p>Specify whether the SSL provider is JDK or OPENSSL.</p> <p>Type: string</p> <p>Example: OPENSSL</p> <p>Default value: JDK</p>
	sniHost	<p>Regular expression to match against the server_name extension on incoming connections. If the names don't match, connection to the acceptor is rejected.</p> <p>Type: string</p> <p>Example: some_regular_expression</p> <p>Default value: Not specified</p>
	expose	<p>Specify whether to expose the acceptor to clients outside OpenShift Container Platform.</p> <p>When you expose an acceptor to clients outside OpenShift, the Operator automatically creates a dedicated Service and Route for each broker Pod in the deployment.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: false</p>

Entry	Sub-entry	Description and usage
	anycastPrefix	<p>Prefix used by a client to specify that the anycast routing type should be used.</p> <p>Type: string</p> <p>Example: <code>jms.queue</code></p> <p>Default value: Not specified</p>
	multicastPrefix	<p>Prefix used by a client to specify that the multicast routing type should be used.</p> <p>Type: string</p> <p>Example: <code>/topic/</code></p> <p>Default value: Not specified</p>
	connectionsAllowed	<p>Number of connections allowed on the acceptor. When this limit is reached, a DEBUG message is issued to the log, and the connection is refused. The type of client in use determines what happens when the connection is refused.</p> <p>Type: integer</p> <p>Example: 2</p> <p>Default value: 0 (unlimited connections)</p>

Entry	Sub-entry	Description and usage
	amqpMinLargeMessageSize	<p>Minimum message size, in bytes, required for the broker to handle an AMQP message as a large message. If the size of an AMQP message is equal or greater to this value, the broker stores the message in a large messages directory (<code>/opt/<custom_resource_name>/data/large-messages</code>, by default) on the persistent volume (PV) used by the broker for message storage. Setting the value to -1 disables large message handling for AMQP messages.</p> <p>Type: integer</p> <p>Example: 204800</p> <p>Default value: 102400 (100 KB)</p>
connectors.connector		A single connector configuration instance.
	name*	<p>Name of connector.</p> <p>Type: string</p> <p>Example: my-connector</p> <p>Default value: Not applicable</p>
	type	<p>The type of connector to create; tcp or vm.</p> <p>Type: string</p> <p>Example: vm</p> <p>Default value: tcp</p>

Entry	Sub-entry	Description and usage
	host*	<p>Host name or IP address to connect to.</p> <p>Type: string</p> <p>Example: 192.168.0.58</p> <p>Default value: Not specified</p>
	port*	<p>Port number to be used for the connector instance.</p> <p>Type: int</p> <p>Example: 22222</p> <p>Default value: Not specified</p>
	sslEnabled	<p>Specify whether SSL is enabled on the connector port. If set to true, look in the secret name specified in sslSecret for the credentials required by TLS/SSL.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: false</p>

Entry	Sub-entry	Description and usage
	sslSecret	<p>Secret where broker key store, trust store, and their corresponding passwords (all Base64-encoded) are stored.</p> <p>If you do not specify a custom secret name for sslSecret, the connector assumes a default secret name. The default secret name has a format of <custom_resource_name><connector_name>secret.</p> <p>You must always create this secret yourself, even when the connector assumes a default name.</p> <p>Type: string</p> <p>Example: my-broker-deployment-my-connector-secret</p> <p>Default value: <custom_resource_name>-<connector_name>-secret</p>
	enabledCipherSuites	<p>Comma-separated list of cipher suites to use for TLS/SSL communication.</p> <p>Type: string</p> <p>NOTE: For a connector, it is recommended that you do not specify a list of cipher suites.</p> <p>Default value: Not specified</p>

Entry	Sub-entry	Description and usage
	enabledProtocols	<p>Comma-separated list of protocols to use for TLS/SSL communication.</p> <p>Type: string</p> <p>Example: TLsv1,TLsv1.1,TLsv1.2</p> <p>Default value: Not specified</p>
	needClientAuth	<p>Specify whether the broker informs clients that two-way TLS is required on the connector. This property overrides wantClientAuth.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: Not specified</p>
	wantClientAuth	<p>Specify whether the broker informs clients that two-way TLS is <i>requested</i> on the connector, but not required. This property is overridden by needClientAuth.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: Not specified</p>

Entry	Sub-entry	Description and usage
	verifyHost	<p>Specify whether to compare the Common Name (CN) of client's certificate to its host name, to verify that they match. This option applies only when two-way TLS is used.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: Not specified</p>
	sslProvider	<p>Specify whether the SSL provider is JDK or OPENSSL.</p> <p>Type: string</p> <p>Example: OPENSSL</p> <p>Default value: JDK</p>
	sniHost	<p>Regular expression to match against the server_name extension on outgoing connections. If the names don't match, the connector connection is rejected.</p> <p>Type: string</p> <p>Example: some_regular_expression</p> <p>Default value: Not specified</p>
	expose	<p>Specify whether to expose the connector to clients outside OpenShift Container Platform.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: false</p>

addressSettings.applyRule Entry	Sub-entry	Description and usage
		<p>Specifies how the Operator applies the configuration that you add to the CR for each matching address or set of addresses.</p> <p>The values that you can specify are:</p> <p>merge_all</p> <p>For address settings specified in both the CR and the default configuration that match the same address or set of addresses:</p> <ul style="list-style-type: none"> ● Replace any property values specified in the default configuration with those specified in the CR. ● Keep any property values that are specified uniquely in the CR or the default configuration. Include each of these in the final, merged configuration. <p>For address settings specified in either the CR or the default configuration that uniquely match a particular address or set of addresses, include these in the final, merged configuration.</p> <p>merge_replace</p> <p>For address settings specified in both the CR and the default configuration that match the same address or set of addresses, include the settings specified in the CR in the final, merged configuration.</p>

Entry	Sub-entry	Description and usage
		<p>Do not include any properties specified in the default configuration, even if these are not specified in the CR. For address settings specified in either the CR or the default configuration that uniquely match a particular address or set of addresses, include these in the final, merged configuration.</p> <p>replace_all Replace all address settings specified in the default configuration with those specified in the CR. The final, merged configuration corresponds exactly to that specified in the CR.</p> <p>Type: string</p> <p>Example: replace_all</p> <p>Default value: merge_all</p>
addressSettings.addressSetting		Address settings for a matching address or set of addresses.

Entry	Sub-entry	Description and usage
	addressFullPolicy	<p>Specify what happens when an address configured with maxSizeBytes becomes full. The available policies are:</p> <p>PAGE</p> <p>Messages sent to a full address are paged to disk.</p> <p>DROP</p> <p>Messages sent to a full address are silently dropped.</p> <p>FAIL</p> <p>Messages sent to a full address are dropped and the message producers receive an exception.</p> <p>BLOCK</p> <p>Message producers will block when they try to send any further messages. The BLOCK policy works only for AMQP, OpenWire, and Core Protocol, because those protocols support flow control.</p> <p>Type: string</p> <p>Example: DROP</p> <p>Default value: PAGE</p>

Entry	Sub-entry	Description and usage
	autoCreateAddresses	<p>Specify whether the broker automatically creates an address when a client sends a message to, or attempts to consume a message from, a queue that is bound to an address that does not exist.</p> <p>Type: Boolean</p> <p>Example: false</p> <p>Default value: true</p>
	autoCreateDeadLetterResources	<p>Specify whether the broker automatically creates a dead letter address and queue to receive undelivered messages.</p> <p>If the parameter is set to true, the broker automatically creates a dead letter address and an associated dead letter queue. The name of the automatically-created address matches the value that you specify for deadLetterAddress.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: false</p>

Entry	Sub-entry	Description and usage
	autoCreateExpiryResources	<p>Specify whether the broker automatically creates an address and queue to receive expired messages.</p> <p>If the parameter is set to true, the broker automatically creates an expiry address and an associated expiry queue. The name of the automatically-created address matches the value that you specify for expiryAddress.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: false</p>
	autoCreateJmsQueues	<p>This property is deprecated. Use autoCreateQueues instead.</p>
	autoCreateJmsTopics	<p>This property is deprecated. Use autoCreateQueues instead.</p>
	autoCreateQueues	<p>Specify whether the broker automatically creates a queue when a client sends a message to, or attempts to consume a message from, a queue that does not yet exist.</p> <p>Type: Boolean</p> <p>Example: false</p> <p>Default value: true</p>

Entry	Sub-entry	Description and usage
	autoDeleteAddresses	<p>Specify whether the broker automatically deletes automatically-created addresses when the broker no longer has any queues.</p> <p>Type: Boolean</p> <p>Example: false</p> <p>Default value: true</p>
	autoDeleteAddressDelay	<p>Time, in milliseconds, that the broker waits before automatically deleting an automatically-created address when the address has no queues.</p> <p>Type: integer</p> <p>Example: 100</p> <p>Default value: 0</p>
	autoDeleteJmsQueues	<p>This property is deprecated. Use autoDeleteQueues instead.</p>
	autoDeleteJmsTopics	<p>This property is deprecated. Use autoDeleteQueues instead.</p>
	autoDeleteQueues	<p>Specify whether the broker automatically deletes an automatically-created queue when the queue has no consumers and no messages.</p> <p>Type: Boolean</p> <p>Example: false</p> <p>Default value: true</p>

Entry	Sub-entry	Description and usage
	autoDeleteCreatedQueues	<p>Specify whether the broker automatically deletes a manually-created queue when the queue has no consumers and no messages.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: false</p>
	autoDeleteQueuesDelay	<p>Time, in milliseconds, that the broker waits before automatically deleting an automatically-created queue when the queue has no consumers.</p> <p>Type: integer</p> <p>Example: 10</p> <p>Default value: 0</p>
	autoDeleteQueuesMessageCount	<p>Maximum number of messages that can be in a queue before the broker evaluates whether the queue can be automatically deleted.</p> <p>Type: integer</p> <p>Example: 5</p> <p>Default value: 0</p>

Entry	Sub-entry	Description and usage
	configDeleteAddresses	<p>When the configuration file is reloaded, this parameter specifies how to handle an address (and its queues) that has been deleted from the configuration file. You can specify the following values:</p> <p>OFF</p> <p>The broker does not delete the address when the configuration file is reloaded.</p> <p>FORCE</p> <p>The broker deletes the address and its queues when the configuration file is reloaded. If there are any messages in the queues, they are removed also.</p> <p>Type: string</p> <p>Example: FORCE</p> <p>Default value: OFF</p>

Entry	Sub-entry	Description and usage
	configDeleteQueues	<p>When the configuration file is reloaded, this setting specifies how the broker handles queues that have been deleted from the configuration file. You can specify the following values:</p> <p>OFF</p> <p>The broker does not delete the queue when the configuration file is reloaded.</p> <p>FORCE</p> <p>The broker deletes the queue when the configuration file is reloaded. If there are any messages in the queue, they are removed also.</p> <p>Type: string</p> <p>Example: FORCE</p> <p>Default value: OFF</p>
	deadLetterAddress	<p>The address to which the broker sends dead (that is, <i>undelivered</i>) messages.</p> <p>Type: string</p> <p>Example: DLA</p> <p>Default value: None</p>
	deadLetterQueuePrefix	<p>Prefix that the broker applies to the name of an automatically-created dead letter queue.</p> <p>Type: string</p> <p>Example: myDLQ.</p> <p>Default value: DLQ.</p>

Entry	Sub-entry	Description and usage
	deadLetterQueueSuffix	<p>Suffix that the broker applies to an automatically-created dead letter queue.</p> <p>Type: string</p> <p>Example: .DLQ</p> <p>Default value: None</p>
	defaultAddressRoutingType	<p>Routing type used on automatically-created addresses.</p> <p>Type: string</p> <p>Example: ANYCAST</p> <p>Default value: MULTICAST</p>
	defaultConsumersBeforeDispatch	<p>Number of consumers needed before message dispatch can begin for queues on an address.</p> <p>Type: integer</p> <p>Example: 5</p> <p>Default value: 0</p>
	defaultConsumerWindowSize	<p>Default window size, in bytes, for a consumer.</p> <p>Type: integer</p> <p>Example: 300000</p> <p>Default value: 1048576 (1024*1024)</p>

Entry	Sub-entry	Description and usage
	defaultDelayBeforeDispatch	<p>Default time, in milliseconds, that the broker waits before dispatching messages if the value specified for defaultConsumersBeforeDispatch has not been reached.</p> <p>Type: integer</p> <p>Example: 5</p> <p>Default value: -1 (no delay)</p>
	defaultExclusiveQueue	<p>Specifies whether all queues on an address are exclusive queues by default.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: false</p>
	defaultGroupBuckets	<p>Number of buckets to use for message grouping.</p> <p>Type: integer</p> <p>Example: 0 (message grouping disabled)</p> <p>Default value: -1 (no limit)</p>
	defaultGroupFirstKey	<p>Key used to indicate to a consumer which message in a group is first.</p> <p>Type: string</p> <p>Example: firstMessageKey</p> <p>Default value: None</p>

Entry	Sub-entry	Description and usage
	defaultGroupRebalance	<p>Specifies whether to rebalance groups when a new consumer connects to the broker.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: false</p>
	defaultGroupRebalancePauseDispatch	<p>Specifies whether to pause message dispatch while the broker is rebalancing groups.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: false</p>
	defaultLastValueQueue	<p>Specifies whether all queues on an address are last value queues by default.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: false</p>
	defaultLastValueKey	<p>Default key to use for a last value queue.</p> <p>Type: string</p> <p>Example: stock_ticker</p> <p>Default value: None</p>
	defaultMaxConsumers	<p>Maximum number of consumers allowed on a queue at any time.</p> <p>Type: integer</p> <p>Example: 100</p> <p>Default value: -1 (no limit)</p>

Entry	Sub-entry	Description and usage
	defaultNonDestructive	<p>Specifies whether all queues on an address are non-destructive by default.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: false</p>
	defaultPurgeOnNoConsumers	<p>Specifies whether the broker purges the contents of a queue once there are no consumers.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: false</p>
	defaultQueueRoutingType	<p>Routing type used on automatically-created queues. The default value is MULTICAST.</p> <p>Type: string</p> <p>Example: ANYCAST</p> <p>Default value: MULTICAST</p>
	defaultRingSize	<p>Default ring size for a matching queue that does not have a ring size explicitly set.</p> <p>Type: integer</p> <p>Example: 3</p> <p>Default value: -1 (no size limit)</p>

Entry	Sub-entry	Description and usage
	enableMetrics	<p>Specifies whether a configured metrics plugin such as the Prometheus plugin collects metrics for a matching address or set of addresses.</p> <p>Type: Boolean</p> <p>Example: false</p> <p>Default value: true</p>
	expiryAddress	<p>Address that receives expired messages.</p> <p>Type: string</p> <p>Example: myExpiryAddress</p> <p>Default value: None</p>
	expiryDelay	<p>Expiration time, in milliseconds, applied to messages that are using the default expiration time.</p> <p>Type: integer</p> <p>Example: 100</p> <p>Default value: -1 (no expiration time applied)</p>
	expiryQueuePrefix	<p>Prefix that the broker applies to the name of an automatically-created expiry queue.</p> <p>Type: string</p> <p>Example: myExp.</p> <p>Default value: EXP.</p>

Entry	Sub-entry	Description and usage
	expiryQueueSuffix	<p>Suffix that the broker applies to the name of an automatically-created expiry queue.</p> <p>Type: string</p> <p>Example: .EXP</p> <p>Default value: None</p>
	lastValueQueue	<p>Specify whether a queue uses only last values or not.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: false</p>
	managementBrowsePageSize	<p>Specify how many messages a management resource can browse.</p> <p>Type: integer</p> <p>Example: 100</p> <p>Default value: 200</p>

Entry	Sub-entry	Description and usage
	match*	<p>String that matches address settings to addresses configured on the broker. You can specify an exact address name or use a wildcard expression to match the address settings to a set of addresses.</p> <p>If you use a wildcard expression as a value for the match property, you must enclose the value in single quotation marks, for example, 'myAddresses*.</p> <p>Type: string</p> <p>Example: 'myAddresses*'</p> <p>Default value: None</p>
	maxDeliveryAttempts	<p>Specifies how many times the broker attempts to deliver a message before sending the message to the configured dead letter address.</p> <p>Type: integer</p> <p>Example: 20</p> <p>Default value: 10</p>
	maxExpiryDelay	<p>Expiration time, in milliseconds, applied to messages that are using an expiration time greater than this value.</p> <p>Type: integer</p> <p>Example: 20</p> <p>Default value: -1 (no maximum expiration time applied)</p>

Entry	Sub-entry	Description and usage
	maxRedeliveryDelay	<p>Maximum value, in milliseconds, between message redelivery attempts made by the broker.</p> <p>Type: integer</p> <p>Example: 100</p> <p>Default value: The default value is ten times the value of redeliveryDelay, which has a default value of 0.</p>
	maxSizeBytes	<p>Maximum memory size, in bytes, for an address. Used when addressFullPolicy is set to PAGING, BLOCK, or FAIL. Also supports byte notation such as "K", "Mb", and "GB".</p> <p>Type: string</p> <p>Example: 10Mb</p> <p>Default value: -1 (no limit)</p>
	maxSizeBytesRejectThreshold	<p>Maximum size, in bytes, that an address can reach before the broker begins to reject messages. Used when the address-full-policy is set to BLOCK. Works in combination with maxSizeBytes for the AMQP protocol only.</p> <p>Type: integer</p> <p>Example: 500</p> <p>Default value: -1 (no maximum size)</p>

Entry	Sub-entry	Description and usage
	messageCounterHistoryDayLimit	<p>Number of days for which a broker keeps a message counter history for an address.</p> <p>Type: integer</p> <p>Example: 5</p> <p>Default value: 0</p>
	minExpiryDelay	<p>Expiration time, in milliseconds, applied to messages that are using an expiration time lower than this value.</p> <p>Type: integer</p> <p>Example: 20</p> <p>Default value: -1 (no minimum expiration time applied)</p>
	pageMaxCacheSize	<p>Number of page files to keep in memory to optimize I/O during paging navigation.</p> <p>Type: integer</p> <p>Example: 10</p> <p>Default value: 5</p>
	pageSizeBytes	<p>Paging size in bytes. Also supports byte notation such as K, Mb, and GB.</p> <p>Type: string</p> <p>Example: 20971520</p> <p>Default value: 10485760 (approximately 10.5 MB)</p>

Entry	Sub-entry	Description and usage
	redeliveryDelay	<p>Time, in milliseconds, that the broker waits before redelivering a cancelled message.</p> <p>Type: integer</p> <p>Example: 100</p> <p>Default value: 0</p>
	redeliveryDelayMultiplier	<p>Multiplying factor to apply to the value of redeliveryDelay.</p> <p>Type: number</p> <p>Example: 5</p> <p>Default value: 1</p>
	redeliveryCollisionAvoidanceFactor	<p>Multiplying factor to apply to the value of redeliveryDelay to avoid collisions.</p> <p>Type: number</p> <p>Example: 1.1</p> <p>Default value: 0</p>
	redistributionDelay	<p>Time, in milliseconds, that the broker waits after the last consumer is closed on a queue before redistributing any remaining messages.</p> <p>Type: integer</p> <p>Example: 100</p> <p>Default value: -1 (not set)</p>

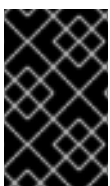
Entry	Sub-entry	Description and usage
	retroactiveMessageCount	<p>Number of messages to keep for future queues created on an address.</p> <p>Type: integer</p> <p>Example: 100</p> <p>Default value: 0</p>
	sendToDlaOnNoRoute	<p>Specify whether a message will be sent to the configured dead letter address if it cannot be routed to any queues.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: false</p>
	slowConsumerCheckPeriod	<p>How often, in seconds, that the broker checks for slow consumers.</p> <p>Type: integer</p> <p>Example: 15</p> <p>Default value: 5</p>
	slowConsumerPolicy	<p>Specifies what happens when a slow consumer is identified. Valid options are KILL or NOTIFY. KILL kills the consumer's connection, which impacts any client threads using that same connection. NOTIFY sends a CONSUMER_SLOW management notification to the client.</p> <p>Type: string</p> <p>Example: KILL</p> <p>Default value: NOTIFY</p>

Entry	Sub-entry	Description and usage
	slowConsumerThreshold	<p>Minimum rate of message consumption, in messages per second, before a consumer is considered slow.</p> <p>Type: integer</p> <p>Example: 100</p> <p>Default value: -1 (not set)</p>
upgrades		
	enabled	<p>When you update the value of version to specify a new target version of AMQ Broker, specify whether to allow the Operator to automatically update the deploymentPlan.image value to a broker container image that corresponds to that version of AMQ Broker.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: false</p>
	minor	<p>Specify whether to allow the Operator to automatically update the deploymentPlan.image value when you update the value of version from one <i>minor</i> version of AMQ Broker to another, for example, from 7.6.0 to 7.8.5.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: false</p>

Entry	Sub-entry	Description and usage
version		<p>Specify a target <i>minor</i> version of AMQ Broker for which you want the Operator to automatically update the CR to use a corresponding broker container image. For example, if you change the value of version from 7.6.0 to 7.7.0 (and upgrades.enabled and upgrades.minor are both set to true), then the Operator updates deploymentPlan.image to a broker image of the form registry.redhat.io/amq7/amq-broker:7.7-x.</p> <p>Type: string</p> <p>Example: 7.7.0</p> <p>Default value: Current version of AMQ Broker</p>

11.1.2. Address Custom Resource configuration reference

A CR instance based on the address CRD enables you to define addresses and queues for the brokers in your deployment. The following table details the items that you can configure.



IMPORTANT

Configuration items marked with an asterisk (*) are required in any corresponding Custom Resource (CR) that you deploy. If you do not explicitly specify a value for a non-required item, the configuration uses the default value.

Entry	Description and usage
addressName*	<p>Address name to be created on broker.</p> <p>Type: string</p> <p>Example: address0</p> <p>Default value: Not specified</p>

Entry	Description and usage
queueName*	<p>Queue name to be created on broker.</p> <p>Type: string</p> <p>Example: queue0</p> <p>Default value: Not specified</p>
removeFromBrokerOnDelete*	<p>Specify whether the Operator removes existing addresses for all brokers in a deployment when you remove the address CR instance for that deployment. The default value is false, which means the Operator does not delete existing addresses when you remove the CR.</p> <p>Type: Boolean</p> <p>Example: true</p> <p>Default value: false</p>
routingType*	<p>Routing type to be used; anycast or multicast.</p> <p>Type: string</p> <p>Example: anycast</p> <p>Default value: Not specified</p>

11.2. APPLICATION TEMPLATE PARAMETERS

Configuration of the AMQ Broker on OpenShift Container Platform image is performed by specifying values of application template parameters. You can configure the following parameters:

Table 11.1. Application template parameters

Parameter	Description
AMQ_ADDRESSES	Specifies the addresses available by default on the broker on its startup, in a comma-separated list.
AMQ_ANICAST_PREFIX	Specifies the anycast prefix applied to the multiplexed protocol ports 61616 and 61617.
AMQ_CLUSTERED	Enables clustering.
AMQ_CLUSTER_PASSWORD	Specifies the password to use for clustering. The AMQ Broker application templates use the value of this parameter stored in the secret named in <code>AMQ_CREDENTIAL_SECRET</code> .

Parameter	Description
AMQ_CLUSTER_USER	Specifies the cluster user to use for clustering. The AMQ Broker application templates use the value of this parameter stored in the secret named in AMQ_CREDENTIAL_SECRET .
AMQ_CREDENTIAL_SECRET	Specifies the secret in which sensitive credentials such as broker user name/password, cluster user name/password, and truststore and keystore passwords are stored.
AMQ_DATA_DIR	Specifies the directory for the data. Used in StatefulSets.
AMQ_DATA_DIR_LOGGING	Specifies the directory for the data directory logging.
AMQ_EXTRA_ARGS	Specifies additional arguments to pass to artemis create .
AMQ_GLOBAL_MAX_SIZE	Specifies the maximum amount of memory that message data can consume. If no value is specified, half of the system's memory is allocated.
AMQ_KEYSTORE	Specifies the SSL keystore file name. If no value is specified, a random password is generated but SSL will not be configured.
AMQ_KEYSTORE_PASSWORD	(Optional) Specifies the password used to decrypt the SSL keystore. The AMQ Broker application templates use the value of this parameter stored in the secret named in AMQ_CREDENTIAL_SECRET .
AMQ_KEYSTORE_TRUSTSTORE_DIR	Specifies the directory where the secrets are mounted. The default value is /etc/amq-secret-volume .
AMQ_MAX_CONNECTIONS	For SSL only, specifies the maximum number of connections that an acceptor will accept.
AMQ_MULTICAST_PREFIX	Specifies the multicast prefix applied to the multiplexed protocol ports 61616 and 61617.
AMQ_NAME	Specifies the name of the broker instance. The default value is amq-broker .
AMQ_PASSWORD	Specifies the password used for authentication to the broker. The AMQ Broker application templates use the value of this parameter stored in the secret named in AMQ_CREDENTIAL_SECRET .

Parameter	Description
AMQ_PROTOCOL	Specifies the messaging protocols used by the broker in a comma-separated list. Available options are amqp , mqtt , openwire , stomp , and hornetq . If none are specified, all protocols are available. Note that for integration of the image with Red Hat JBoss Enterprise Application Platform, the OpenWire protocol must be specified, while other protocols can be optionally specified as well.
AMQ_QUEUES	Specifies the queues available by default on the broker on its startup, in a comma-separated list.
AMQ_REQUIRE_LOGIN	If set to true , login is required. If not specified, or set to false , anonymous access is permitted. By default, the value of this parameter is not specified.
AMQ_ROLE	Specifies the name for the role created. The default value is amq .
AMQ_TRUSTSTORE	Specifies the SSL truststore file name. If no value is specified, a random password is generated but SSL will not be configured.
AMQ_TRUSTSTORE_PASSWORD	(Optional) Specifies the password used to decrypt the SSL truststore. The AMQ Broker application templates use the value of this parameter stored in the secret named in AMQ_CREDENTIAL_SECRET .
AMQ_USER	Specifies the user name used for authentication to the broker. The AMQ Broker application templates use the value of this parameter stored in the secret named in AMQ_CREDENTIAL_SECRET .
APPLICATION_NAME	Specifies the name of the application used internally within OpenShift. It is used in names of services, pods, and other objects within the application.
IMAGE	Specifies the image. Used in the persistence , persistent-ssl , and statefulset-clustered templates.
IMAGE_STREAM_NAMESPACE	Specifies the image stream name space. Used in the ssl and basic templates.
OPENSIFT_DNS_PING_SERVICE_PORT	Specifies the port number for the OpenShift DNS ping service.

Parameter	Description
PING_SVC_NAME	Specifies the name of the OpenShift DNS ping service. The default value is \$APPLICATION_NAME-ping if you have specified a value for APPLICATION_NAME . Otherwise, the default value is ping . If you specify a custom value for PING_SVC_NAME , this value overrides the default value. If you want to use templates to deploy multiple broker clusters in the same OpenShift project namespace, you must ensure that PING_SVC_NAME has a unique value for each deployment.
VOLUME_CAPACITY	Specifies the size of the persistent storage for database volumes.



NOTE

If you use **broker.xml** for a custom configuration, any values specified in that file for the following parameters will override values specified for the same parameters in the your application templates.

- AMQ_NAME
- AMQ_ROLE
- AMQ_CLUSTER_USER
- AMQ_CLUSTER_PASSWORD

11.3. LOGGING

In addition to viewing the OpenShift logs, you can troubleshoot a running AMQ Broker on OpenShift Container Platform image by viewing the AMQ logs that are output to the container's console.

Procedure

- At the command line, run the following command:

```
$ oc logs -f <pass:quotes[<pod-name>]> <pass:quotes[<container-name>]>
```

Revised on 2022-08-19 13:14:15 UTC