Red Hat AMQ 7.7

Deploying AMQ Broker on OpenShift

For Use with AMQ Broker 7.7

Last Updated: 2021-08-04

Red Hat AMQ 7.7 Deploying AMQ Broker on OpenShift

For Use with AMQ Broker 7.7

Legal Notice

Abstract

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

Table of Contents

CHAPTER 1. INTRODUCTION TO AMQ BROKER ON OPENSHIFT CONTAINER PLATFORM

Red Hat AMQ Broker 7.7 is available as a containerized image that is provided for use with OpenShift Container Platform (OCP) 3.11 and later.

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.

AMQ Broker on OCP provides similar functionality to Red Hat AMQ Broker, but some aspects of the functionality need to be configured specifically for use with OpenShift Container Platform.

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 OPENSHIFT 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

Deployment using application templates

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.

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.

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 4.6, “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. An accompanying Custom Resource (CR) file enables you to 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. The Operator also interacts with Kubernetes via the **kubectl** command, using HTTP requests.

The main broker CRD for AMQ Broker 7.7 is the **broker_activemqartemis_crd** file in the **deploy/crds** directory of the archive that you download and extract when installing the Operator. This CRD enables you to configure a broker deployment in a given OpenShift project. The other CRDs in the **deploy/crds** directory are for configuring addresses and for the Operator to use when instantiating a scaledown controller .

When deployed, each CRD is a separate controller, running independently within the Operator.

For a complete configuration reference for each CRD, see:

- Section 10.1.1, “Broker Custom Resource configuration reference”
- Section 10.1.2, “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. OPERATOR DEPLOYMENT NOTES

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

- 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.
- 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.
- 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 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 4.1 and later)
  - Persistent Storage (OpenShift Container Platform 3.11).
- In AMQ Broker 7.7, you must add configuration for the following items to the main broker CR instance **before** deploying the CR for the first time.

  - Address settings
  - 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

  You **cannot** add configuration for any of the listed items to a broker deployment that is already running.

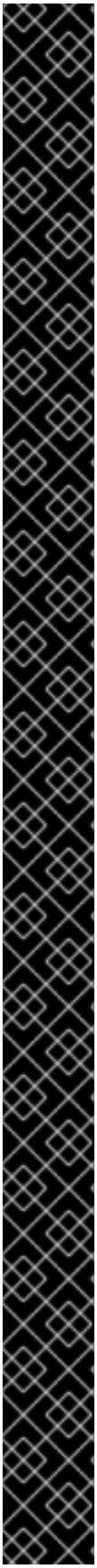
The procedures in the next section show you how to install the Operator and use Custom Resources (CRs) to create broker deployments on OpenShift Container Platform. When you have successfully completed the procedures, you will have the Operator running in an individual Pod. Each broker instance that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

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

3.1. INSTALLING THE OPERATOR USING THE CLI

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 7.7 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 3.2, “Installing the Operator using the Operator Lifecycle Manager” .



IMPORTANT

- 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.

- If you previously deployed:

  - Version 0.15 of the Operator (that is, the first version available for AMQ Broker 7.7) you can *upgrade* your project to use the latest version of the Operator, rather than performing a new installation. When you upgrade, you do not need to delete any existing CRDs in your OpenShift cluster. However, as part of the upgrade, you must delete the main broker Custom Resource (CR) instance from your project. Then, when you have upgraded the Operator, you must deploy a new CR to recreate your broker deployment. For more information, see Section 5.2.1, "Upgrading version 0.15 of the Operator".

  - Version 0.13 of the Operator (that is, the version available for AMQ Broker 7.6) you can *upgrade* your project to use the latest version of the Operator, rather than performing a new installation. When you upgrade, you do not need to delete any existing CRDs in your OpenShift cluster. However, as part of the upgrade, you must delete the main broker Custom Resource (CR) instance from your project. Then, when you have upgraded the Operator, you must deploy a new CR to recreate your broker deployment. For more information, see Section 5.2.2, "Upgrading version 0.13 of the Operator" .

  - Version 0.9 of the Operator (that is, the version available for AMQ Broker 7.5 or the Long Term Suppport (LTS) version available for AMQ Broker 7.4), you can *upgrade* your project to use the latest version of the Operator, rather than performing a new installation. When you upgrade, you do not need to delete any existing CRDs in your OpenShift cluster. However, as part of the upgrade, you must delete the main broker Custom Resource (CR) instance from your project. Then, when you have upgraded the Operator, you must deploy a new CR to recreate your broker deployment. For more information, see Section 5.2.3, "Upgrading version 0.9 of the Operator" .

- 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.

3.1.1. 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 7.7.

Procedure

1. In your web browser, navigate to the **Software Downloads** page for AMQ Broker 7.7.0 patches.
2. Ensure that the value of the **Version** drop-down list is set to **7.7.0** and the **Patches** tab is selected.
3. Next to **AMQ Broker 7.7.0 Operator Installation and Example Files**, click **Download**. Download of the **amq-broker-operator-7.7.0-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-7.7.0-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-7.7.0-ocp-install-examples.zip
```

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

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

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

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

   b. Or, switch to an existing project:

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

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

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

9. Specify a role name for the Operator.
   a. Open the **role.yaml** file. This file specifies the resources that the Operator can use and modify.

b. Ensure that the **kind** element is set to **Role**.

c. In the **metadata** section, assign a custom name to the role, or use the default name. The default name is **amq-broker-operator**.

d. Create the role in your project.

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

10. 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:

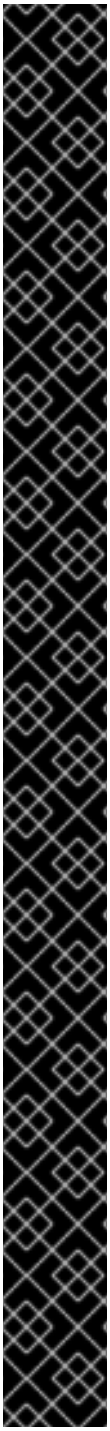
```
metadata:
    name: amq-broker-operator
subjects:
    kind: ServiceAccount
    name: amq-broker-operator
roleRef:
    kind: Role
    name: amq-broker-operator
```

b. Create the role binding in your project.

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

3.1.2. 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 7.7 in your OpenShift project.



IMPORTANT

If you previously deployed:

- Version 0.15 of the Operator (that is, the first version available for AMQ Broker 7.7) you can *upgrade* your project to use the latest version of the Operator, rather than performing a new installation. When you upgrade, you do not need to delete any existing CRDs in your OpenShift cluster. However, as part of the upgrade, you must delete the main broker Custom Resource (CR) instance from your project. Then, when you have upgraded the Operator, you must deploy a new CR to recreate your broker deployment. For more information, see Section 5.2.1, "Upgrading version 0.15 of the Operator" .

- Version 0.13 of the Operator (that is, the version available for AMQ Broker 7.6) you can *upgrade* your project to use the latest version of the Operator, rather than performing a new installation. When you upgrade, you do not need to delete any existing CRDs in your OpenShift cluster. However, as part of the upgrade, you must delete the main broker Custom Resource (CR) instance from your project. Then, when you have upgraded the Operator, you must deploy a new CR to recreate your broker deployment. For more information, see Section 5.2.2, "Upgrading version 0.13 of the Operator".

- Version 0.9 of the Operator (that is, the version available for AMQ Broker 7.5 or the Long Term Suppport (LTS) version available for AMQ Broker 7.4), you can *upgrade* your project to use the latest version of the Operator, rather than performing a new installation. When you upgrade, you do not need to delete any existing CRDs in your OpenShift cluster. However, as part of the upgrade, you must delete the main broker Custom Resource (CR) instance from your project. Then, when you have upgraded the Operator, you must deploy a new CR to recreate your broker deployment. For more information, see Section 5.2.3, "Upgrading version 0.9 of the Operator".

- Version 0.6 of the Operator (that is, the Technical Preview version for AMQ Broker 7.4.0) you **cannot** directly upgrade this Operator to later versions, including the latest version available for AMQ Broker 7.7. The Custom Resource Definitions (CRDs) used by version 0.6 are not compatible with later versions of the Operator. Instead, you must perform a new installation of the Operator. This involves deleting any existing Custom Resource Definitions (CRDs) installed in your OpenShift cluster. These steps are described in the procedure in this section.

Prerequisites

- You have already prepared your OpenShift project for the Operator deployment. See Section 3.1.1, "Getting the Operator code" .

- Starting in AMQ Broker 7.3, you use a new version of the Red Hat Container Registry 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 4.1 and later)
- Persistent Storage (OpenShift Container Platform 3.11)

- To learn how to specify the size of Persistent Volume Claim (PVC) that each broker in your deployment requires for persistent message storage, see Section 4.2.1, "Configuring broker storage size".

Procedure

1. In the OpenShift Container Platform web console, open the project in which you want your broker deployment.
   If you created a new project, it is currently empty. Observe that there are no Deployments, StatefulSets, Pods, Services, or Routes.

2. If you deployed an earlier version of the AMQ Broker Operator in the project, remove the main broker Custom Resource (CR) that you used to create your broker deployment. Deleting the main CR removes the existing broker deployment in the project. For example:

```
oc delete -f deploy/crs/broker_v1alpha1_activemqartemis_cr.yaml.
```

3. If you deployed an earlier version of the AMQ Broker Operator in the project, delete this Operator instance. For example:

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

4. If you deployed Custom Resource Definitions (CRDs) in your OpenShift cluster for an earlier version of the AMQ Broker Operator, remove these CRDs from the cluster. For example:

```
oc delete -f deploy/crds/broker_v1alpha1_activemqartemis_crd.yaml
oc delete -f deploy/crds/broker_v1alpha1_activemqartemisaddress_crd.yaml
oc delete -f deploy/crds/broker_v1alpha1_activemqartemisscaledown_crd.yaml
```

5. Deploy the CRDs that are included in the **deploy/crds** directory of the Operator archive that you downloaded and extracted. You must install the latest 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
```

6. Link the pull secret associated with the account used for authentication in the Red Hat Container Registry 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>
```

NOTE

In OpenShift Container Platform 4.1 or later, you can also use the web console to associate a pull secret with a project in which you want to deploy container images such as the AMQ Broker Operator. To do this, click **Administration → Service Accounts**. Specify the pull secret associated with the account that you use for authentication in the Red Hat Container Registry.

7. In the **deploy** directory of the Operator archive that you downloaded and extracted, open the **operator.yaml** file. Ensure that the value of the **spec.containers.image** property is set to the latest Operator image for AMQ Broker 7.7, as shown below.

```
spec:
   template:
      spec:
         containers:
            image: registry.redhat.io/amq7/amq-broker-rhel7-operator:0.17
```

8. Deploy the Operator.

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

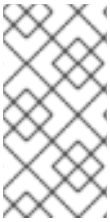
In your OpenShift project, the **amq-broker-operator** image that you deployed starts in a new Pod.

The information on the **Events** tab of the new Pod confirms that OpenShift has deployed the Operator image you specified, assigned a new container to a node in your OpenShift cluster, and 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. Specifically, setting the **replicas** element 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 3.2, "Installing the Operator using the Operator Lifecycle Manager".

3.2. INSTALLING THE OPERATOR USING THE OPERATOR LIFECYCLE MANAGER

3.2.1. Overview of the Operator Lifecycle Manager

In OpenShift Container Platform 4.1 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 4.1 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. 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.

3.2.2. Installing the Operator in OperatorHub

In OperatorHub, the name of the Operator for AMQ Broker 7.7 is **Red Hat Integration - AMQ Broker**. If you do not see the Operator automatically available in OperatorHub, follow this procedure to manually install the Operator in OperatorHub.

Procedure

1. In your web browser, navigate to the AMQ Broker **Software Downloads** page.
2. In the **Version** drop-down box, ensure the value is set to the latest AMQ Broker version, **7.7.0**.
3. Click the **Patches** tab.

4. Next to **AMQ Broker 7.7.0 Operator Installation and Example Files**, click **Download**. Download of the **amq-broker-operator-7.7.0-ocp-install-examples.zip** compressed archive automatically begins.

5. 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-7.7.0-ocp-install-examples.zip ~/broker/operator
```

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

```
cd ~/broker/operator
unzip amq-broker-operator-7.7.0-ocp-install-examples.zip
```

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

```
cd amq-broker-operator-7.7.0-ocp-install-examples
```

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

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

9. Install the Operator in OperatorHub.

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

After a few minutes, the Operator for AMQ Broker 7.7 is available in the **OperatorHub** section of the OpenShift Container Platform web console. The name of the Operator is **Red Hat Integration - AMQ Broker**.

3.2.3. 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 for AMQ Broker using the 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 3.2.2, “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 7.7, the latest version tag of this Operator is **0.17**.

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**, ensure that the radio button entitled **current** is selected. This option specifies the channel used to track and receive updates for the Operator. The **current** value specifies that the AMQ Broker 7.7 channel is used.

   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 3.3.1, “Deploying a basic broker instance” .

3.3. CREATING OPERATOR-BASED BROKER DEPLOYMENTS

3.3.1. 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 7.7, if you want to configure any of the following items, you must add the appropriate configuration to the main broker CR instance **before** deploying the CR for the first time.
  - Address settings
  - 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 3.1, “Installing the Operator using the CLI” .
  - To use the OperatorHub graphical interface to install the AMQ Broker Operator, see Section 3.2, “Installing the Operator using the Operator Lifecycle Manager” .
- Starting in AMQ Broker 7.3, you use a new version of the Red Hat Container Registry 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 with administrator privileges for 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 with administrator privileges for 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, the configuration might resemble that shown below. This configuration is the default content of the **broker_activemqartemis_cr.yaml** sample CR.

```
apiVersion: broker.amq.io/v2alpha3
kind: ActiveMQArtemis
metadata:
  name: ex-aao
  application: ex-aao-app
spec:
    version: 7.7.0
    deploymentPlan:
      size: 2
      image: registry.redhat.io/amq7/amq-broker:7.7
      ...
```

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. The **size** value specifies the number of brokers to deploy. The default value of **2** specifies a clustered broker deployment of two brokers. However, to deploy a single broker instance, change the value to **1**.

3. The **image** value specifies the container image to use to launch the broker. Ensure that this value specifies the latest version of the AMQ Broker 7.7 broker container image in the Red Hat Container Registry, as shown below.

```
image: registry.redhat.io/amq7/amq-broker:7.7
```

NOTE

In the preceding step, the **image** attribute specifies a *floating* image tag (that is, **7.7**) rather than a full image tag (for example, **7.7-2**). When you specify this floating tag, your deployment uses the latest image available in the 7.7 image stream. In addition, when you specify a floating tag such as this, if the **imagePullPolicy** attribute in your Stateful Set is set to **Always**, your deployment automatically pulls and uses new *micro* image versions (for example, **7.7-3**, **7-7-4**, and so on) when they become available in the Red Hat Container Registry.

4. Deploy the CR instance.

a. Using the OpenShift command-line interface:

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.

```
$ oc create -f <path/to/custom-resource-instance>.yaml
```

b. Using the OpenShift Container Platform web console:

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

5. In the OpenShift Container Platform web console, click **Workloads → Stateful Sets** (OpenShift Container Platform 4.1 or later) or **Applications → Stateful Sets** (OpenShift Container Platform 3.11). You see a new Stateful Set called **ex-aao-ss**.
Expand the **ex-aao-ss Stateful Set** section. You see that there is one Pod, corresponding to the single broker that you defined in the CR.

   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.

Additional resources

- For a complete configuration reference for the main broker Custom Resource (CR), see Section 10.1, "Custom Resource configuration reference" .
- To learn how to connect a running broker to the AMQ Broker management console, see Section 4.4.5, "Connecting to the AMQ Broker management console" .

3.3.2. 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

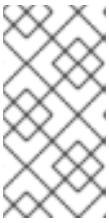
- A basic broker instance is already deployed. See Section 3.3.1, "Deploying a basic broker instance".

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/v2alpha3
```

```
kind: ActiveMQArtemis
metadata:
  name: ex-aao
  application: ex-aao-app
spec:
    version: 7.7.0
    deploymentPlan:
        size: 4
        image: registry.redhat.io/amq7/amq-broker:7.7
        ...
```



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 with administrator privileges for 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
```

3.3.3. 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 3.1.2, "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 Releasing volumes in the OpenShift documentation.

- In AMQ Broker 7.7, if you want to configure any of the following items, you must add the appropriate configuration to the main CR instance **before** deploying the CR for the first time.
  - Address settings
  - 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. 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 **ActiveMQAretmisAddress** 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 **ActiveMQAretmis** CRD listed under **Administration → Custom Resource Definitions** in the OpenShift Container Platform web console.

IMPORTANT

- To configure address settings for an Operator-based deployment, you must be using 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, see Chapter 5, *Upgrading an Operator-based broker deployment*.
- In AMQ Broker 7.7, you must add address settings configuration to your main broker CR instance **before** deploying the CR for the first time. You **cannot** add this configuration to a broker deployment that is already running.

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.

4.1.1. 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.

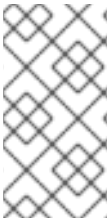
Configuration item for standalone broker deployment	Configuration item for OpenShift broker deployment
address-full-policy	addressFullPolicy
auto-create-queues	autoCreateQueues
default-queue-routing-type	defaultQueueRoutingType
last-value-queue	lastValueQueue

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 10.1, “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.

4.1.2. 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 3.1, "Installing the Operator using the CLI" .
  - Section 3.2, "Installing the Operator using the Operator Lifecycle Manager" .
- You should be familiar with how to use a sample CR included with AMQ Broker to create a basic broker deployment. For more information, see Section 3.3.1, "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 with administrator privileges for the project that contains 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 with administrator privileges for the project that contains 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 that contains the broker deployment.

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

      iii. Create the 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. (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
```

4.1.3. 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 and then match those settings to addresses in your broker deployment.

IMPORTANT

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

Prerequisites

- You must be using 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, see Chapter 5, *Upgrading an Operator-based broker deployment*.
- You should be familiar with how to use a sample CR included with AMQ Broker to create a basic broker deployment. For more information, see Section 3.3.1, "Deploying a basic broker instance".

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 with administrator privileges for the project that will contain 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 with administrator privileges for the project that will contain 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-aaoaddress
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.

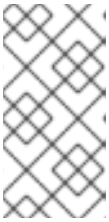
```
$ oc create -f <path/to/address-custom-resource-instance>.yaml
```

   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 the following:

```
apiVersion: broker.amq.io/v2alpha3
```

```
kind: ActiveMQArtemis
metadata:
  name: ex-aao
  application: ex-aao-app
spec:
    version: 7.7.0
    deploymentPlan:
        size: 3
        image: registry.redhat.io/amq7/amq-broker:7.7
        requireLogin: false
        persistenceEnabled: true
        journalType: nio
        messageMigration: true
```



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: 7.7.0
    deploymentPlan:
        size: 3
        image: registry.redhat.io/amq7/amq-broker:7.7
        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: 7.7.0
    deploymentPlan:
        size: 3
        image: registry.redhat.io/amq7/amq-broker:7.7
        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: 7.7.0
  deploymentPlan:
    size: 3
    image: registry.redhat.io/amq7/amq-broker:7.7
    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.

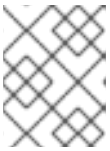
maxDeliveryAttempts

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: 7.7.0
  deploymentPlan:
    size: 3
    image: registry.redhat.io/amq7/amq-broker:7.7
    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: 7.7.0
  deploymentPlan:
    size: 3
    image: registry.redhat.io/amq7/amq-broker:7.7
    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, megred 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.

```
$ 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**.

4.1.3.1. Understanding the default address settings configuration

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

If you have included an address settings configuration in the Custom Resource (CR) instance for your broker deployment, the Operator also 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. Typically, Init Containers include utilities or setup scripts that are not present in the application image.

If you have configured address settings in the CR, the initialization of each broker Pod follows this sequence:

1. The Init Container runs. The Init Container processes the address settings configuration in the CR and converts the configuration to XML.

2. When the Init Container has finished, the broker application container starts. When the broker starts, it creates a **default** address settings configuration, included in the generated **broker.xml** configuration file. The default address settings configuration looks as follows:

```
<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>
```

3. Based on the value of the **applyRule** property in the CR, the Operator merges or replaces the default address settings configuration shown above with the configuration that you have specified in the CR.
   When the broker Pod is initialized and running, you can inspect the resulting address settings configuration in the **broker.xml** configuration file. For a running broker Pod, this file is located in the **/home/jboss/amq-broker/etc** directory.

Additional resources

- To learn how to configure the **applyRule** property for address settings in your CR, see Section 4.1.3, "Matching address settings to configured addresses in an Operator-based broker deployment".
- To learn about all of the configuration options for addresses, queues, and address settings for OpenShift Container Platform broker deployments, see Section 10.1, "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)

4.2. CONFIGURING BROKER STORAGE REQUIREMENTS

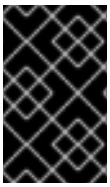
To use persistent storage in an Operator-based broker deployment, you set **persistenceEnabled** to **true** in the Custom Resource (CR) instance used to create the deployment. If you do not have container-native storage in your OpenShift cluster, you need to manually provision Persistent Volumes (PVs) 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 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, see Chapter 5, *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.

4.2.1. 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 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, see Chapter 5, *Upgrading an Operator-based broker deployment*.
- You should be familiar with how to use a sample CR included with AMQ Broker to create a basic broker deployment. See Section 3.3.1, “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 4.1 and later)
  - 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 with administrator privileges for the project in which you are creating the deployment:

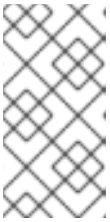
      ```
      oc login -u <user> -p <password> --server=<host:port>
      ```

      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 with administrator privileges for 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 the following:

```
apiVersion: broker.amq.io/v2alpha3
kind: ActiveMQArtemis
metadata:
  name: ex-aao
  application: ex-aao-app
spec:
    version: 7.7.0
    deploymentPlan:
        size: 3
        image: registry.redhat.io/amq7/amq-broker:7.7
        requireLogin: false
        persistenceEnabled: true
        journalType: nio
        messageMigration: true
```



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: 7.7.0
  deploymentPlan:
    size: 3
    image: registry.redhat.io/amq7/amq-broker:7.7
    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**.

4.3. 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 Stateful Set 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 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, see Chapter 5, *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.
- If you have included an address settings configuration in the Custom Resource (CR) instance for your broker deployment, the Operator runs a type of container called an *Init Container* (in addition to the broker container) when initializing each broker Pod. In this case, 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 4.1.3.1, "Understanding the default address settings 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.

4.3.1. 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 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, see Chapter 5, *Upgrading an Operator-based broker deployment*.
- You should be familiar with how to use a sample CR included with AMQ Broker to create a basic broker deployment. See Section 3.3.1, “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 with administrator privileges for 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 with administrator privileges for 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 the following:

```
apiVersion: broker.amq.io/v2alpha3
kind: ActiveMQArtemis
metadata:
  name: ex-aao
  application: ex-aao-app
spec:
    version: 7.7.0
    deploymentPlan:
        size: 3
        image: registry.redhat.io/amq7/amq-broker:7.7
        requireLogin: false
        persistenceEnabled: true
        journalType: nio
        messageMigration: true
```

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: 7.7.0
    deploymentPlan:
        size: 3
        image: registry.redhat.io/amq7/amq-broker:7.7
        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**.

4.4. CONFIGURING OPERATOR-BASED BROKER DEPLOYMENTS FOR CLIENT CONNECTIONS

4.4.1. 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 3.1, “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

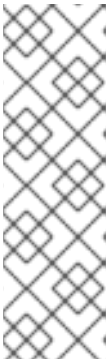
1. In the **deploy/crs** directory of the Operator archive that you downloaded and extracted during your initial installation, open the **broker_activemqartemis_cr.yaml** Custom Resource (CR) file.

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 the 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 Section 4.4.5.1, “Accessing the broker management console” .

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.

4. 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
...
```

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 4.4.2, “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 4.4.2, "Securing broker-client connections".
- For a complete Custom Resource configuration reference, including configuration of acceptors and connectors, see Section 10.1, "Custom Resource configuration reference" .

4.4.2. 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.

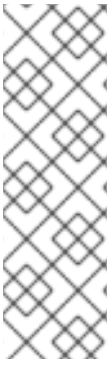
Two-way TLS

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 secrete name, you must still generate this secret yourself. It is not automatically created.

4.4.2.1. 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 a wildcard, *, 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,..."
```

4.4.2.2. 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 4.4.2.1, "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
...
```

4.4.2.3. 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 4.4.2.1, “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
...
```

4.4.3. 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 4.4.4.1, “Connecting to the broker from internal clients” .

4.4.4. 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).

4.4.4.1. 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 Stateful Sets 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 4.4.3, “Networking Services in your broker deployments” .

4.4.4.2. 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 protcols, 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>
```

4.4.4.3. 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-specifc 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)

4.4.5. Connecting to the AMQ Broker management console

The broker hosts its own management console at port 8161. Each broker Pod in your deployment has a Service and Route that provide access to the console.

The following procedure shows how to connect a running broker instance to the AMQ Broker management console.

Prerequisites

- You have deployed a basic broker using the AMQ Broker Operator. For more information, see Section 3.3.1, “Deploying a basic broker instance” .

4.4.5.1. Accessing the broker management console

Each broker Pod in your deployment has a service that provides access to the console. The default name of this service uses the format **<Custom Resource name>-wconsj-<broker Pod ordinal>-svc**. For example, **my-broker-deployment-wconsj-0-svc**. Each Service has a corresponding Route that uses the format `**<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 AMQ Broker management console for a running broker instance.

Procedure

1. In the OpenShift Container Platform web console, click **Networking → Routes** (OpenShift Container Platform 4.1 or later) or **Applications → Routes** (OpenShift Container Platform 3.11). On the **Routes** pane, you see a Route corresponding to the **wconsj** Service.

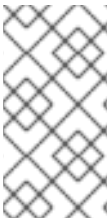
2. Under **Hostname**, note the complete URL. You need to specify this URL to access the console.

3. In a web browser, enter the host name URL.

   a. 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.

   b. 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.

4. To log in to the management console, enter the user name and password specified in the **adminUser** and **adminPassword** parameters of the Custom Resource (CR) instance used to create your broker deployment.
   If there are no values explictly specified for **adminUser** and **adminPassword** in the CR, follow the instructions in Accessing 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 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.

4.4.5.2. Accessing 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 has a format of **<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 4.1 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 4.1 or later) or **Actions → Edit YAML** (OpenShift Container Platform 3.11).

   b. From the command line:

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

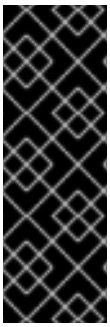
3. To decode a value in the secret, use a command such as the following:

```
$ echo 'dXNlcl9uYW1l' | base64 --decode
console_admin
```

4.5. 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 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, see Chapter 5, *Upgrading an Operator-based broker deployment*.
- For Operator-based broker deployments in AMQ Broker 7.7, large message handling is available only for the AMQP protocol.

4.5.1. 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 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, see Chapter 5, *Upgrading an Operator-based broker deployment*.

- You should be familiar with how to configure acceptors for Operator-based broker deployments. See Section 4.4.1, "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 2.4, "Operator deployment notes"
  - Section 10.1, "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.

4.6. HIGH AVAILABILITY AND MESSAGE MIGRATION

4.6.1. High availability

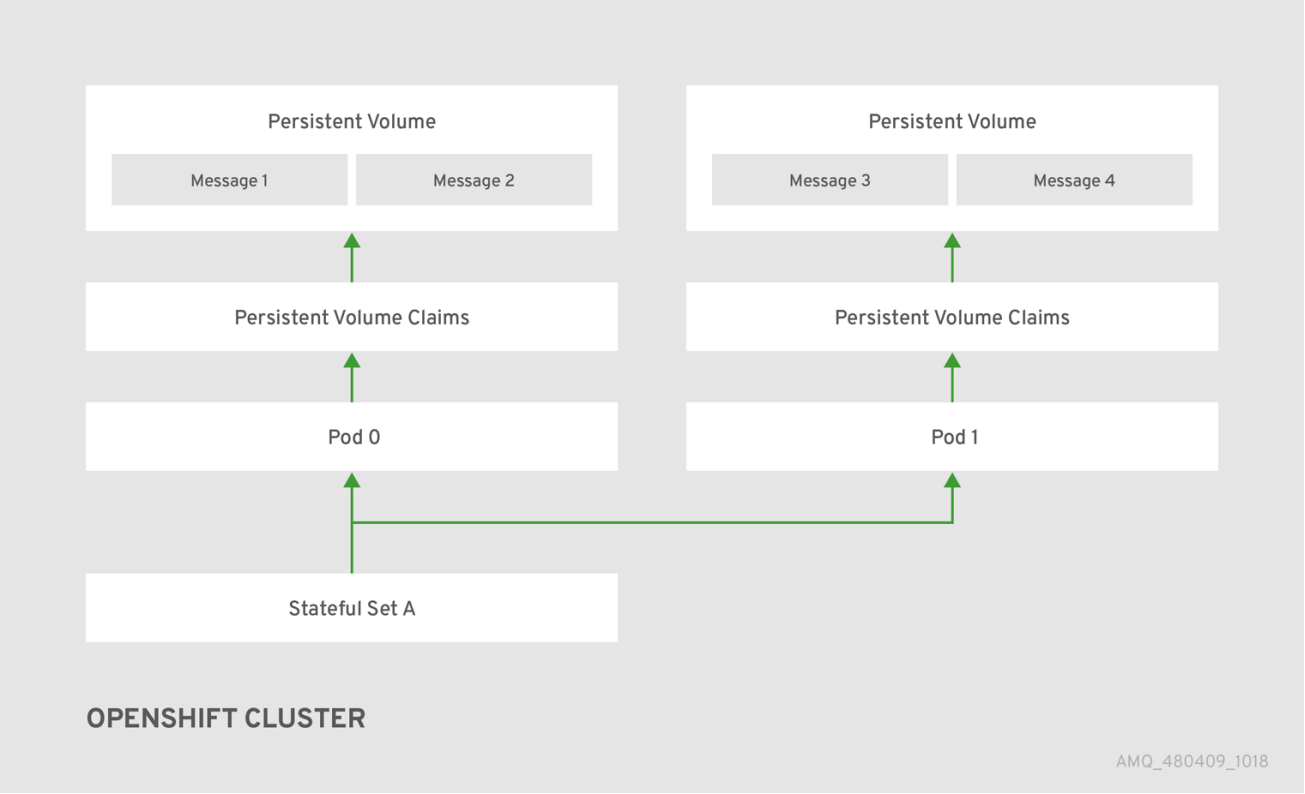
The term *high availability* refers to a system that can remain operational even when part of that system fails or is shut down. For AMQ Broker on OpenShift Container Platform, this means ensuring the integrity and availability of messaging data if a broker Pod fails, or shuts down due to intentional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

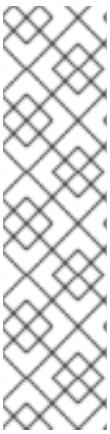
The following figure shows a StatefulSet-based broker deployment. In this case, the two broker Pods in the broker cluster are still running.



When a broker Pod shuts down, the AMQ Broker Operator automatically starts a *scaledown controller* that performs the migration of messages to an 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.

4.6.2. 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.

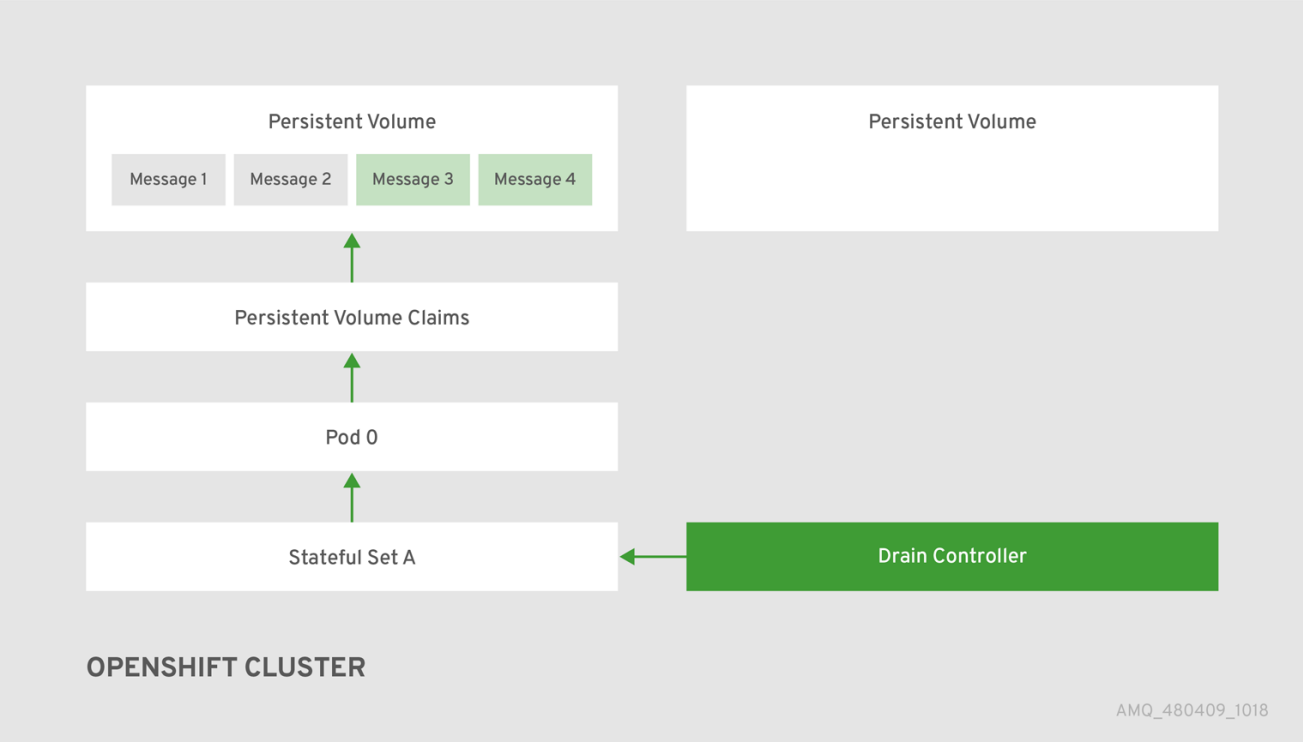
3. To check for Persistent Volumes (PVs) that have been orphaned, the scaledown controller looks at the ordinal on the volume claim. The controller compares the ordinal on the volume claim to that of the broker Pods that are still running in the StatefulSet (that is, the broker cluster) in the project.
   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.

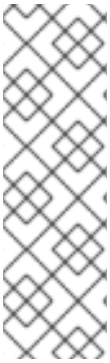
Additional resources

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

4.6.3. 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 3.3.1, “Deploying a basic broker instance”.
- You should understand how message migration works. For more information, see Section 4.6.2, “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-aao-ss-0                                  1/1   Running   0   112s
ex-aao-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-aao-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-aao-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
```

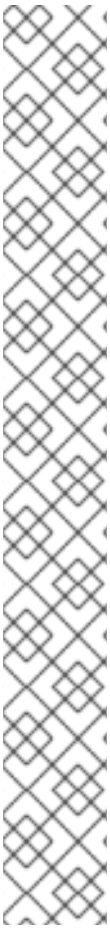
   You see that the Pod **ex-aao-ss-1** starts to shut down. The scaledown controller starts a new drainer Pod of the same name. This drainer Pod also shuts down after it migrates all messages from broker Pod **ex-aao-ss-1** to the other broker Pod in the cluster, **ex-aao-ss-0**.

6. When the drainer Pod is shut down, check the message count on the **TEST** queue of broker Pod **ex-aao-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. 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



NOTE

- To upgrade an existing AMQ Broker deployment on OpenShift Container Platform 3.11 to run on OpenShift Container Platform 4.1 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 in the Red Hat Container Registry, provided that the **imagePullPolicy** attribute in your Stateful Set or Deployment Config is set to **Always**.
  For example, suppose that the **image** attribute of your deployment specifies a floating tag of **7.7**. If the deployment currently uses minor version **7.7-2**, and a newer minor version, **7.7-3**, 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.

5.1. ABOUT UPGRADING TECHNICAL PREVIEW-BASED DEPLOYMENTS

Version 0.6 of the AMQ Broker Operator, which was first available in AMQ Broker 7.4.0, was a Technical Preview feature only. If you intend to upgrade the broker container image or Operator version for a broker deployment based on this version of the Operator, the list below describes some important considerations.

- You **cannot** directly upgrade version 0.6 of the Operator to later versions, including the latest version for AMQ Broker 7.7. The Custom Resource Definitions (CRDs) used by version 0.6 are not compatible with later versions of the Operator. To upgrade your Operator from version 0.6, you must delete your previously-deployed CRDs before performing a new installation of the Operator. When you have installed the new version of the Operator, you then need to recreate your broker deployments. To learn how to install the latest version of the Operator , see Section 3.1, “Installing the Operator using the CLI” .
- When you update your cluster with the CRDs for the latest Operator version, this update affects

all projects in the cluster. Any broker Pods previously deployed from version 0.6 of the Operator 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.

- Section 5.4, "Upgrading the broker container image" shows how to upgrade the **broker container image** for an Operator-based deployment. That procedure assumes that you used version 0.13 of the Operator to create your previous deployment. You can use similar instructions to upgrade the broker container image for a deployment based on version 0.6 of the Operator. The main broker Custom Resource (CR) instance included with version 0.6 was **broker_v1alpha1_activemqartemis_cr.yaml**.

5.2. 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 7.7.

NOTE

- For an alternative method of upgrading the AMQ Broker Operator that uses the OperatorHub graphical interface, see Section 5.3, "Upgrading the Operator using OperatorHub".
- Upgrading a AMQ Broker Operator deployment using either the OpenShift CLI or OperatorHub requires cluster administrator privileges for your OpenShift cluster.

5.2.1. 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 previous version available for AMQ Broker 7.7) to the latest version for AMQ Broker 7.7.

Procedure

1. In your web browser, navigate to the **Software Downloads** page for AMQ Broker 7.7.0 patches.
2. Ensure that the value of the **Version** drop-down list is set to **7.7.0** and the **Patches** tab is selected.
3. Next to **AMQ Broker 7.7.0 Operator Installation and Example Files**, click **Download**. Download of the **amq-broker-operator-7.7.0-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-7.7.0-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-7.7.0-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.

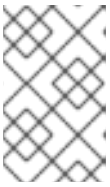
```
$ oc project <project-name>
```

8. Delete the main Custom Resource (CR) for the existing broker deployment in your project. 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 Operator archive that you downloaded and extracted during your previous installation, open the **operator.yaml** file.

11. Update the **spec.containers.image** property to specify the full path to the latest Operator image for AMQ Broker 7.7, as shown below.

```
spec:
   template:
      spec:
         containers:
            image: registry.redhat.io/amq7/amq-broker-rhel7-operator:0.17
```

12. When you have updated **spec.containers.image**, apply the changes.

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

OpenShift updates your project to use the latest Operator version.

13. To recreate your previous broker deployment, deploy a new instance of the main broker CR in your project. For more information, see Section 3.3.1, “Deploying a basic broker instance” .

5.2.2. 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 7.7.

Procedure

1. In your web browser, navigate to the **Software Downloads** page for AMQ Broker 7.7.0 patches.
2. Ensure that the value of the **Version** drop-down list is set to **7.7.0** and the **Patches** tab is selected.
3. Next to **AMQ Broker 7.7.0 Operator Installation and Example Files**, click **Download**. Download of the **amq-broker-operator-7.7.0-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-7.7.0-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-7.7.0-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 Custom Resource (CR) for the existing broker deployment in your project. 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
```

10. Update the **address** CRD in your OpenShift cluster to the latest version included with AMQ Broker 7.7.

```
$ 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 7.7, this CRD is fully compatible with the one that was included with the Operator for AMQ Broker 7.6.

11. In the **deploy** directory of the Operator archive that you downloaded and extracted during your previous installation, open the **operator.yaml** file.

12. Update the **spec.containers.image** property to specify the full path to the latest Operator image for AMQ Broker 7.7, as shown below.

```
spec:
  template:
    spec:
      containers:
        image: registry.redhat.io/amq7/amq-broker-rhel7-operator:0.17
```

13. When you have updated **spec.containers.image**, apply the changes.

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

OpenShift updates your project to use the latest Operator version.

5.2.3. 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 7.7.

IMPORTANT

As part of this Operator upgrade, you must delete and then recreate any existing broker deployment in your project. These steps are described in the procedure that follows.

Procedure

1. In your web browser, navigate to the **Software Downloads** page for AMQ Broker 7.7.0 patches.

2. Ensure that the value of the **Version** drop-down list is set to **7.7.0** and the **Patches** tab is selected.

3. Next to **AMQ Broker 7.7.0 Operator Installation and Example Files**, click **Download**. Download of the **amq-broker-operator-7.7.0-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-7.7.0-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-7.7.0-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 Custom Resource (CR) for the existing broker deployment in your project. For example:

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

9. Update the main broker Custom Resource Definition (CRD) in your OpenShift cluster to the latest version included with AMQ Broker 7.7.

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

10. Update the address CRD in your OpenShift cluster to the latest version included with AMQ Broker 7.7.

```
$ 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 7.7, this CRD is fully compatible with the one included with the previous Operator version.

11. In the **deploy** directory of the Operator archive that you downloaded and extracted during your previous installation, open the **operator.yaml** file.

12. Update the **spec.containers.image** property to specify the full path to the latest Operator image for AMQ Broker 7.7, as shown below.

```
spec:
   template:
      spec:
         containers:
            image: registry.redhat.io/amq7/amq-broker-rhel7-operator:0.17
```

13. When you have updated **spec.containers.image**, apply the changes.

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

OpenShift updates your project to use the latest Operator version.

14. To recreate your previous broker deployment, deploy a new instance of the main broker CR in your project. For more information, see Section 3.3.1, “Deploying a basic broker instance” .

5.3. UPGRADING THE OPERATOR USING OPERATORHUB

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

IMPORTANT

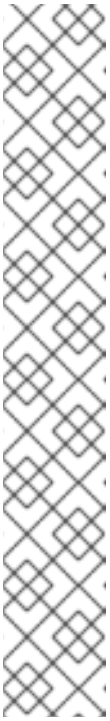
- Upgrading the AMQ Broker Operator using OperatorHub requires cluster administrator privileges for your OpenShift cluster.
- You cannot use OperatorHub to *seamlessly* upgrade the Operator to the latest version for AMQ Broker 7.7. Instead, you must uninstall the existing Operator and then install the latest version from OperatorHub, as described in the procedure that follows. As part of this upgrade, you must also delete and then recreate any existing broker deployment in your project.
- If you are upgrading version 0.6 of the Operator (that is, the Technical Preview version that was available for AMQ Broker 7.4), you must also delete the Custom Resource Definitions (CRDs) previously deployed in your cluster, before performing a new installation of the Operator. These steps are described in the procedure that follows.

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 **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 for 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. If you are upgrading version 0.6 of the Operator (that is, the Technical Preview version that was available for AMQ Broker 7.4):
   a. Access the latest CRDs included with AMQ Broker 7.7. See Section 3.1.1, “Getting the Operator code”

Operator code .

b. Remove existing CRDs in your cluster and manually deploy the latest CRDs. See Section 3.1.2, "Deploying the Operator using the CLI" .



NOTE

- If you are upgrading version 0.15 of the Operator (that is, the first version available for AMQ Broker 7.7), version 0.13 (that is, the version available for AMQ Broker 7.6), or version 0.9 (that is, the version available for AMQ Broker 7.5, or the Long Term Support version available for AMQ Broker 7.4), the Operator Lifecycle Manager (OLM) **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.

5. Use OperatorHub to install the latest version of the Operator for AMQ Broker 7.7. For more information, see Section 3.2.3, "Deploying the Operator from OperatorHub".

6. To recreate your previous broker deployment, deploy a new instance of the main broker CR in your project. For more information, see Section 3.3.1, "Deploying a basic broker instance" .

5.4. UPGRADING THE BROKER CONTAINER IMAGE

For an Operator-based broker deployment, the procedures in this section show how to:

- Upgrade the broker container image by specifying an image tag
- Upgrade the broker container image by specifying the AMQ Broker version

Both approaches to upgrading the broker container image involve editing the Custom Resource (CR) instance used to create your deployment.

An advantage of upgrading the broker container image by specifying the AMQ Broker version is that you are not responsible for ensuring that the image tag specified in the CR corresponds to a valid image in the Red Hat Container Registry. Instead, when you update the CR to specify a AMQ Broker version (for example, **7.7.0**), the Operator first validates that a direct upgrade to the specified version of AMQ Broker is possible for your existing deployment. If an upgrade is possible, the Operator updates the CR to specify a broker container image for the specified version of AMQ Broker, for example, **registry.redhat.io/amq7/amq-broker:7.7-2**. The broker image that the Operator specifies is defined by default in an environment variable in the Operator deployment. The environment variable name aligns with the AMQ Broker version, for example, **BROKER_IMAGE_770**. 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.

A potential disadvantage of upgrading the broker container image by specifying the AMQ Broker

version is that the broker image is statically defined in the Operator deployment. The Operator is **not** able to dynamically search the Red Hat Container Registry for newer image versions that have become available. For this reason, you might still want the option of manually updating the image tag after the Operator has performed an initial upgrade. Therefore, a strategy to ensure that a broker deployment continues to use the latest version of the broker container image might mix both upgrade approaches.

The procedures that follow show both approaches.

5.4.1. Upgrading the broker container image by specifying an image tag

The following procedure shows how to upgrade the broker container image for an Operator-based deployment by specifying an image tag.

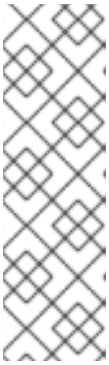
Prerequisites

- The procedure assumes that you used version 0.15 or 0.13 of the Operator (that is, the *first* version available for AMQ Broker 7.7 or the version available for AMQ Broker 7.6) to create your previous broker deployment. The name of the main broker Custom Resource (CR) instance included with those versions of the Operator was **broker_activemqartemis_cr.yaml**.
  You can use similar instructions to upgrade the broker container image for a deployment based on 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). The name of the main broker CR included with version 0.9 of the Operator was **broker_v2alpha1_activemqartemis_cr.yaml**.

Procedure

1. In the **deploy/crs** directory of the Operator archive that you downloaded and extracted during your previous installation, open the main broker CR, **broker_activemqartemis_cr.yaml**.
2. Edit the **image** element to specify the latest AMQ Broker 7.7 container image, as shown below.

```
spec:
    ...
    deploymentPlan:
        ...
        image: registry.redhat.io/amq7/amq-broker:7.7
```



NOTE

In the preceding step, you specify a *floating* image tag (that is, **7.7**) rather than a full image tag (for example, **7.7-2**). When you specify this floating tag and apply the changes, your deployment *automatically* pulls and uses the latest image available in the 7.7 image stream. In addition, when you specify a floating tag, if the **imagePullPolicy** attribute in your Stateful Set is set to **Always**, your deployment automatically pulls and uses new *micro* image versions (for example, **7.7-3**, **7-7-4**, and so on) when they become available in the Red Hat Container Registry.

3. Save the CR file.
4. Apply the CR change.

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

When you apply the CR change, each broker Pod in your deployment shuts down and then restarts using the new broker container image. If you have multiple brokers in your deployment, only one broker Pod shuts down and restarts at a time.

5.4.2. 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 deployment by specifying an AMQ Broker version.

Prerequisites

- Before you can configure your Custom Resource (CR) instance for upgrade of the broker container image based on a specified AMQ Broker version, you must be using *at least* version 0.15 of the Operator (that is, the first version available for AMQ Broker 7.7). To learn how to upgrade the Operator to the *latest* version for AMQ Broker 7.7, see:
  - Section 5.2, "Upgrading the Operator using the CLI"
  - Section 5.3, "Upgrading the Operator using OperatorHub".

Procedure

1. In the **deploy/crs** directory of the Operator archive that you downloaded and extracted during your previous installation, open the main broker CR, **broker_activemqartemis_cr.yaml**.
2. In the **upgrades** section of the CR, enable upgrades of the broker container image between minor versions of AMQ Broker, for example between **7.6.0** and **7.7.0**.

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

NOTE

Both of the preceding options must be set to **true** to enable upgrades between minor versions of AMQ Broker.

3. Update the **version** property of the CR to specify the minor version of AMQ Broker that you want the upgraded broker container image to correspond to. For example:

```
spec:
  version: 7.7.0
  ...
  upgrades:
    enabled: true
    minor: true
```

4. Save the CR file.
5. Apply the CR change.

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

When you apply the CR change, the Operator first validates that a direct upgrade to the specified version of AMQ Broker is possible 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 possible, the Operator automatically modifies the CR instance for your deployment to specify a broker container image for the specified version of AMQ Broker, for example, **registry.redhat.io/amq7/amq-broker:7.7-2**. The default broker image that the Operator uses is defined in an environment variable in the Operator deployment. The environment variable name aligns with the AMQ Broker version, for example, **BROKER_IMAGE_770**. 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.

When you reopen or reload the CR, you see that the Operator has updated the **image** attribute. For example:

```
spec:
  ...
  deploymentPlan:
    ...
    image: registry.redhat.io/amq7/amq-broker:7.7-2
```

CHAPTER 6. DEPLOYING AMQ BROKER ON OPENSHIFT CONTAINER PLATFORM USING APPLICATION TEMPLATES

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.

6.1. 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:

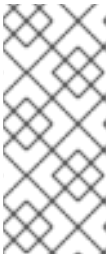
```
$ oc login -u system:admin
$ oc project openshift
```

   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.

2. 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/77-7.7.0.GA/amq-broker-7-image-streams.yaml
```

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

```
$ for template in amq-broker-77-basic.yaml \
amq-broker-77-ssl.yaml \
amq-broker-77-custom.yaml \
amq-broker-77-persistence.yaml \
amq-broker-77-persistence-ssl.yaml \
amq-broker-77-persistence-clustered.yaml \
amq-broker-77-persistence-clustered-ssl.yaml;
 do
 oc replace --force -f \
https://raw.githubusercontent.com/jboss-container-images/jboss-amq-7-broker-openshift-image/77-7.7.0.GA/templates/${template}
 done
```

6.2. 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
   ```

6.3. 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 Container Registry 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.

6.3.1. 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-77-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 6.1. Basic broker application template

Environment variable	Display Name	Value	Description
AMQ_PROTOCOL	AMQ Protocols	openwire,amqp,stomp,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.

Environment variable	Display Name	Value	Description
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

6.3.2. 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 'dXNlcl9uYW1l' | base64 --decode
user_name
```

6.3.3. 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 **Deployment Configs**.
   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 **Deployment Configs → broker-amq**.
   b. Click **Actions → Edit Deployment Configs**.
   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	Container image name	Repository name
OpenShift Container Platform	AMQ Broker	amq7/amq-broker
OpenShift Container Platform on IBM Z	AMQ Broker on Eclipse OpenJ9 11	amq7/amq-broker-openj9-11-rhel8

Platform	Container image name	Repository name
OpenShift Container Platform on IBM Power Systems	AMQ Broker on Eclipse OpenJ9 11	amq7/amq-broker-openj9-11-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:7.7'
    ..
    imagePullSecrets:
      - name: {PULL-SECRET-NAME}
```

NOTE

To use the latest version of the broker container image for OpenShift Container Platform on IBM Z or OpenShift Container Platform on IBM Power Systems, specify the image attribute as follows:

- **registry.redhat.io/amq7/amq-broker-openj9-11-rhel8:7.7** (OpenShift Container Platform on IBM Power Systems)
- **registry.redhat.io/amq7/amq-broker-openj9-11-rhel8:7.7** (OpenShift Container Platform on IBM Z)

a. Click **Save**.

1. Import the latest broker image version to your project namespace. For example:

```
$ oc import-image amq7/amq-broker:7.7 --from=registry.redhat.io/amq7/amq-broker --confirm
```

2. 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.

3. 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>
```

6.4. 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.

6.4.1. 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

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

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

2. 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
```

6.4.2. 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>
```

6.4.3. Creating an SSL Route

To enable client applications outside your OpenShift cluster to connnect 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 6.2. 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 7. 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.

7.1. DEPLOYING A BASIC BROKER WITH SSL

Deploy a basic broker that is ephemeral and supports SSL.

7.1.1. 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-77-ssl** template which is labeled **Red Hat AMQ Broker 7.7 (Ephemeral, with SSL)**.
6. Set the following values in the configuration and click **Create**.

Table 7.1. Example template

Environment variable	Display Name	Value	Description
AMQ_PROTOCOL	AMQ Protocols	openwire,amqp,stomp,mqtt,hornetq	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

7.1.2. 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 .

7.1.3. 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.

7.2. 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.

7.2.1. 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-77-persistence-ssl** template, which is labelled **Red Hat AMQ Broker 7.7 (Persistence, with SSL)**.
6. Set the following values in the configuration and click **create**.

Table 7.2. Example template

Environment variable	Display Name	Value	Description
AMQ_PROTOCOL	AMQ Protocols	openwire,amqp,stomp,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.
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

7.2.2. 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 **Stateful Sets** 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>
```

6. 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

7. Now scale the broker back up to **1**.

```
$ oc scale statefulset broker-amq --replicas=1
statefulset "broker-amq" scaled
```

8. Consume the messages again by using the terminal. For example:

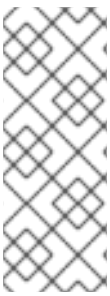
```
sh-4.2$ broker/bin/artemis consumer --destination queue://demoQueue
Consumer:: filter = null
Consumer ActiveMQQueue[demoQueue], thread=0 wait until 1000 messages are consumed
Received 1000
Consumer ActiveMQQueue[demoQueue], thread=0 Consumed: 1000 messages
Consumer ActiveMQQueue[demoQueue], thread=0 Consumer thread finished
```

Additional resources

- For more information on managing stateful applications, see StatefulSets (external).

7.2.3. 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.

7.3. DEPLOYING A SET OF CLUSTERED BROKERS

Deploy a clustered set of brokers where each broker runs in its own Pod.

7.3.1. 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.

7.3.2. 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-77-persistence-clustered** template which is labeled **Red Hat AMQ Broker 7.7 (no SSL, clustered)**.
6. Set the following values in the configuration and click **create**.

Table 7.3. Example template

Environment variable	Display Name	Value	Description
AMQ_PROTOCOL	AMQ Protocols	openwire,amqp,stomp,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.
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

7.3.3. 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 **Stateful Sets** 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 Stateful Set 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
```

7.3.4. 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 .

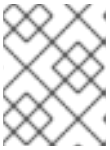
Procedure

1. 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-77-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.

7.4. 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.

7.4.1. 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.

7.4.2. 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-77-persistence-clustered-ssl** template which is labeled **Red Hat AMQ Broker 7.7 (SSL, clustered)**.
6. Set the following values in the configuration and click **create**.

Table 7.4. Example template

Environment variable	Display Name	Value	Description
AMQ_PROTOCOL	AMQ Protocols	openwire,amqp,stomp,mqtt,hornetq	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

7.4.3. Deploying the application

Deploy after creating the application. Deploying the application creates a Pod and starts the broker.

Procedure

1. Click **Stateful Sets** 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.

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 Stateful Set 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 .

7. 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.

7.5. 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.

7.5.1. 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-77-custom** template which is labeled **Red Hat AMQ Broker 7.7(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 confguration 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.

7.5.2. 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.

7.6. 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

7.6.1. 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!
```

7.7. 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-77-persistence-clustered-ssl** template.

7.7.1. 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.

7.7.2. 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/ataylor/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.

7.8. 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-77-persistence-clustered** template.

7.8.1. 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-77-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**.

7.8.2. Connecting the clients

Create consumers that are round-robinned 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.

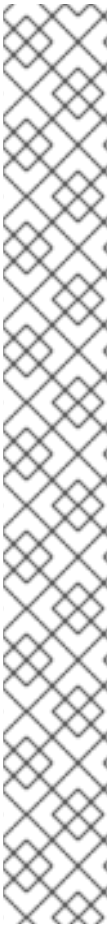
```
artemis producer --url tcp://<IP_ADDRESS>:30001 --message-count 300 --destination
queue://demoQueue
```

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 8. 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 4.1 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 in the Red Hat Container Registry, provided that the **imagePullPolicy** attribute in your Stateful Set or Deployment Config is set to **Always**.
  For example, suppose that the **image** attribute of your deployment specifies a floating tag of **7.7**. If the deployment currently uses minor version **7.7-2**, and a newer minor version, **7.7-3**, 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.

8.1. 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 Container Registry 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 Deployment Config (DC) corresponding to your broker deployment.
   a. In OpenShift Container Platform 4.1 or later, click **Workloads → Deployment Configs**.
   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 Deployment Config** (OpenShift Container Platform 4.1 or later) or **Edit YAML** (OpenShift Container Platform 3.11).
   The **YAML** tab of the Deployment Config opens, with the **.yaml** file in an editable mode.
5. Edit the **image** attribute to specify the latest AMQ Broker 7.7 container image, **registry.redhat.io/amq7/amq-broker:7.7**.
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:7.7'
..
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, 7.7-1, 7.7-2, and so on. If you specify a tag name without a final digit (**7.7**, 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 in the Red Hat Container Registry, 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.

8.2. 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 Container Registry 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 persistent broker deployment.
3. Select the StatefulSet (SS) corresponding to your broker deployment.
   a. In OpenShift Container Platform 4.1 or later, click **Workloads → Stateful Sets**.
   b. In OpenShift Container Platform 3.11, click **Applications → Stateful Sets**.
4. From the **Actions** menu, click **Edit Stateful Set** (OpenShift Container Platform 4.1 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.
5. To prepare your broker deployment for upgrade, scale the deployment down to zero brokers.
   a. If the **replicas** attribute is currently set to **1** or greater, set it to **0**.
   b. Click **Save**.
6. When all broker Pods have shut down, edit the Stateful Set **.yaml** file again. Edit the **image** attribute to specify the latest AMQ Broker 7.7 container image, **registry.redhat.io/amq7/amq-broker:7.7**.
7. 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:7.7'
..
imagePullSecrets:
  - name: {PULL-SECRET-NAME}
```

8. Set the **replicas** attribute back to the original value.
9. Click **Save**.
   If a newer broker image than the one currently installed is available in the Red Hat Container Registry, OpenShift Container Platform upgrades your broker deployment. To do this, OpenShift Container Platform restarts the broker Pod.

CHAPTER 9. MONITORING YOUR BROKERS

To monitor runtime data for brokers in your deployment, use one of these approaches:

- Section 9.1, “Monitoring broker runtime metrics using Prometheus”
- Section 9.2, “Monitoring broker runtime data using JMX”

In general, using Prometheus is the recommended approach. However, you might choose to use the Jolokia REST interface to JMX if a metric that you need to monitor is not exported by the Prometheus plugin. For more information about the broker runtime metrics that the Prometheus plugin exports, see Section 9.1.1, “Overview of broker metrics” .

9.1. 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.

9.1.1. Overview of broker metrics

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 listed below. A description of each metric is exported along with the metric itself.

Broker Metrics

- **address.memory.usage**
- **connection.count**
- **total.connection.count**

Address Metrics

- **routed.message.count**
- **unrouted.message.count**

Queue Metrics

- **consumer.count**
- **delivering.durable.message.count**
- **delivering.durable.persistent.size**
- **delivering.message.count**
- **delivering.persistent.size**
- **durable.message.count**
- **durable.persistent.size**
- **messages.acknowledged**
- **messages.added**
- **message.count**
- **messages.killed**
- **messages.expired**
- **persistent.size**
- **scheduled.durable.message.count**
- **scheduled.durable.persistent.size**
- **scheduled.message.count**
- **scheduled.persistent.size**

For higher-level broker metrics that are not listed above, you can calculate these by aggregating lower-level metrics. For example, to calculate total message count, you can aggregate the **message.count** metrics from all queues in your broker deployment.

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

9.1.2. 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 8, *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 4.1 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 Stateful Sets or Deployment Configs in your project, click:
   a. **Workloads → Stateful Sets** or **Workloads → Deployment Configs** (OpenShift Container Platform 4.1 or later).
   b. **Applications → Stateful Sets** or **Applications → Deployments** (OpenShift Container Platform 3.11).
4. Click the Stateful Set or Deployment Config 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 Stateful Set or Deployment Config. 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 6.1, “Installing the image streams and application templates”

9.1.3. 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 9.1.2, “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 4.1 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.

9.2. 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\",component=addresses,queue=\"TESTQUEUE\",routing-type=\"anycast\",subcomponent=queues","attribute":"MaxConsumers","type":"read"},"value":-1,"timestamp":1528297825,"status":200}
```

CHAPTER 10. REFERENCE

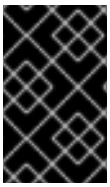
10.1. CUSTOM RESOURCE CONFIGURATION REFERENCE

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.

10.1.1. 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*		Administrator user name required for connecting to the broker and management console. 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**. **Type**: string **Example**: my-user **Default value**: Automatically-generated, random value

Entry	Sub-entry	Description and usage
adminPassword*		Administrator password required for connecting to the broker and management console. 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**. **Type**: string **Example**: my-password **Default value**: Automatically-generated, random value
deploymentPlan*		Broker deployment configuration
	image*	Full path of the broker container image to be pulled from the Red Hat Container Registry. The default container image tag matches the AMQ Broker version. **Type**: string **Example**: registry.redhat.io/amq7/amq-broker:latest **Default value**: registry.redhat.io/amq7/amq-broker:7.7

Entry	Sub-entry	Description and usage
	size*	Number of broker Pods to create in the deployment. 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. **Type**: int **Example**: 1 **Default value**: 2
	requireLogin	Specify whether login credentials are required to connect to the broker. **Type**: Boolean **Example**: false **Default value**: true
	persistenceEnabled	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). **Type**: Boolean **Example**: false **Default value**: true

Entry	Sub-entry	Description and usage
	journalType	Specify whether to use asynchronous I/O (AIO) or non-blocking I/O (NIO). **Type**: string **Example**: aio **Default value**: nio
	messageMigration	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. **Type**: Boolean **Example**: false **Default value**: true
	resources.limits.cpu	Maximum amount of host-node CPU, in millicores, that each broker container running in a Pod in a deployment can consume. **Type**: string **Example**: "500m" **Default value**: Uses the same default value that your version of OpenShift Container Platform uses. Consult a cluster administrator.

Entry	Sub-entry	Description and usage
	resources.limits.memory	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). **Type**: string **Example**: "1024M" **Default value**: Uses the same default value that your version of OpenShift Container Platform uses. Consult a cluster administrator.
	resources.requests.cpu	Amount of host-node CPU, in millicores, that each broker container running in a Pod in a deployment explicitly requests. **Type**: string **Example**: "250m" **Default value**: Uses the same default value that your version of OpenShift Container Platform uses. Consult a cluster administrator.

Entry	Sub-entry	Description and usage
	resources.requests.memory	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). **Type**: string **Example**: "512M" **Default value**: Uses the same default value that your version of OpenShift Container Platform uses. Consult a cluster administrator.
	storage.size	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). **Type**: string **Example**: 4Gi **Default value**: 2Gi
acceptors.acceptor		A single acceptor configuration instance.

Entry	Sub-entry	Description and usage
	name*	Name of acceptor. **Type**: string **Example**: my-acceptor **Default value**: Not applicable
	port	Port number to use for the acceptor instance. **Type**: int **Example**: 5672 **Default value**: 61626 for the first acceptor that you define. The default value then increments by 10 for every subsequent acceptor that you define.
	protocols	Messaging protocols to be enabled on the acceptor instance. **Type**: string **Example**: amqp,core **Default value**: all
	sslEnabled	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. **Type**: Boolean **Example**: true **Default value**: false

Entry	Sub-entry	Description and usage
	sslSecret	Secret where broker key store, trust store, and their corresponding passwords (all Base64-encoded) are stored. 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**. You must always create this secret yourself, even when the acceptor assumes a default name. **Type**: string **Example**: my-broker-deployment-my-acceptor-secret **Default value**: *<Custom-Resource-name>-<acceptor-name>*-secret

Entry	Sub-entry	Description and usage
	enabledCipherSuites	Comma-separated list of cipher suites to use for TLS/SSL communication. 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. **Type**: string **Default value**: Not specified
	enabledProtocols	Comma-separated list of protocols to use for TLS/SSL communication. **Type**: string **Example**: TLSv1,TLSv1.1,TLSv1.2 **Default value**: Not specified

Entry	Sub-entry	Description and usage
	needClientAuth	Specify whether the broker informs clients that two-way TLS is required on the acceptor. This property overrides **wantClientAuth**. **Type**: Boolean **Example**: true **Default value**: Not specified
	wantClientAuth	Specify whether the broker informs clients that two-way TLS is *requested* on the acceptor, but not required. This property is overridden by **needClientAuth**. **Type**: Boolean **Example**: true **Default value**: Not specified
	verifyHost	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. **Type**: Boolean **Example**: true **Default value**: Not specified

Entry	Sub-entry	Description and usage
	sslProvider	Specify whether the SSL provider is JDK or OPENSSL. **Type**: string **Example**: OPENSSL **Default value**: JDK
	sniHost	Regular expression to match against the **server_name** extension on incoming connections. If the names don't match, connection to the acceptor is rejected. **Type**: string **Example**: some_regular_expression **Default value**: Not specified
	expose	Specify whether to expose the acceptor to clients outside OpenShift Container Platform. **Type**: Boolean **Example**: true **Default value**: false
	anycastPrefix	Prefix used by a client to specify that the **anycast** routing type should be used. **Type**: string **Example**: jms.queue **Default value**: Not specified

Entry	Sub-entry	Description and usage
	multicastPrefix	Prefix used by a client to specify that the **multicast** routing type should be used. **Type**: string **Example**: /topic/ **Default value**: Not specified
	connectionsAllowed	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. **Type**: integer **Example**: 2 **Default value**: 0 (unlimited connections)

Entry	Sub-entry	Description and usage
	amqpMinLargeMessageSize	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 (/**opt**/***<custom-resource-name>***/**data**/**large-messages**, 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. **Type**: integer **Example**: 204800 **Default value**: 102400 (100 KB)
connectors.connector		A single connector configuration instance.
	name*	Name of connector. **Type**: string **Example**: my-connector **Default value**: Not applicable
	type	The type of connector to create; **tcp** or **vm**. **Type**: string **Example**: vm **Default value**: tcp

Entry	Sub-entry	Description and usage
	host*	Host name or IP address to connect to. **Type**: string **Example**: 192.168.0.58 **Default value**: Not specified
	port*	Port number to be used for the connector instance. **Type**: int **Example**: 22222 **Default value**: Not specified
	sslEnabled	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. **Type**: Boolean **Example**: true **Default value**: false

Entry	Sub-entry	Description and usage
	sslSecret	Secret where broker key store, trust store, and their corresponding passwords (all Base64-encoded) are stored. 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**. You must always create this secret yourself, even when the connector assumes a default name. **Type**: string **Example**: my-broker-deployment-my-connector-secret **Default value**: *<Custom-Resource-name>-<connector-name>*-secret
	enabledCipherSuites	Comma-separated list of cipher suites to use for TLS/SSL communication. **Type**: string **NOTE**: For a connector, it is recommended that you **do not** specify a list of cipher suites. **Default value**: Not specified

Entry	Sub-entry	Description and usage
	enabledProtocols	Comma-separated list of protocols to use for TLS/SSL communication. **Type**: string **Example**: TLSv1,TLSv1.1,TLSv1.2 **Default value**: Not specified
	needClientAuth	Specify whether the broker informs clients that two-way TLS is required on the connector. This property overrides **wantClientAuth**. **Type**: Boolean **Example**: true **Default value**: Not specified
	wantClientAuth	Specify whether the broker informs clients that two-way TLS is *requested* on the connector, but not required. This property is overridden by **needClientAuth**. **Type**: Boolean **Example**: true **Default value**: Not specified

Entry	Sub-entry	Description and usage
	verifyHost	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. **Type**: Boolean **Example**: true **Default value**: Not specified
	sslProvider	Specify whether the SSL provider is **JDK** or **OPENSSL**. **Type**: string **Example**: OPENSSL **Default value**: JDK
	sniHost	Regular expression to match against the **server_name** extension on outgoing connections. If the names don't match, the connector connection is rejected. **Type**: string **Example**: some_regular_expression **Default value**: Not specified
	expose	Specify whether to expose the connector to clients outside OpenShift Container Platform. **Type**: Boolean **Example**: true **Default value**: false

addressSettings.applyRule Entry	Sub-entry	Description and usage
		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.

Entry	Sub-entry	Description and usage
		Do not include any [illegible] 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, megred configuration corresponds exactly to that specified in the CR. **Type**: string **Example**: replace_all **Default value**: merge_all
addressSettings.addressSetting		Address settings for a matching address or *set* of addresses.

Entry	Sub-entry	Description and usage
	addressFullPolicy	Specify what happens when an address configured with **maxSizeBytes** becomes full. The available policies are: **PAGE** Messages sent to a full address are paged to disk. **DROP** Messages sent to a full address are silently dropped. **FAIL** Messages sent to a full address are dropped and the message producers receive an exception. **BLOCK** Message producers will block when they try to send any further messages. The BLOCK policy works only for the AMQP, OpenWire, and Core protocols because they feature flow control. **Type**: string **Example**: DROP **Default value**: PAGE
	autoCreateAddresses	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. **Type**: Boolean **Example**: false **Default value**: true

Entry	Sub-entry	Description and usage
	autoCreateDeadLetterResources	Specify whether the broker automatically creates a dead letter address and queue to receive undelivered messages. 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**. **Type**: Boolean **Example**: true **Default value**: false
	autoCreateExpiryResources	Specify whether the broker automatically creates an address and queue to receive expired messages. 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**. **Type**: Boolean **Example**: true **Default value**: false
	autoCreateJmsQueues	This property is deprecated. Use **autoCreateQueues** instead.

Entry	Sub-entry	Description and usage
	autoCreateJmsTopics	This property is deprecated. Use **autoCreateQueues** instead.
	autoCreateQueues	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. **Type**: Boolean **Example**: false **Default value**: true
	autoDeleteAddresses	Specify whether the broker automatically deletes automatically-created addresses when the broker no longer has any queues. **Type**: Boolean **Example**: false **Default value**: true
	autoDeleteAddressDelay	Time, in milliseconds, that the broker waits before automatically deleting an automatically-created address when the address has no queues. **Type**: integer **Example**: 100 **Default value**: 0
	autoDeleteJmsQueues	This property is deprecated. Use **autoDeleteQueues** instead.

Entry	Sub-entry	Description and usage
	autoDeleteJmsTopics	This property is deprecated. Use **autoDeleteQueues** instead.
	autoDeleteQueues	Specify whether the broker automatically deletes an automatically-created queue when the queue has no consumers and no messages. **Type**: Boolean **Example**: false **Default value**: true
	autoDeleteCreatedQueues	Specify whether the broker automatically deletes a manually-created queue when the queue has no consumers and no messages. **Type**: Boolean **Example**: true **Default value**: false
	autoDeleteQueuesDelay	Time, in milliseconds, that the broker waits before automatically deleting an automatically-created queue when the queue has no consumers. **Type**: integer **Example**: 10 **Default value**: 0

Entry	Sub-entry	Description and usage
	autoDeleteQueuesMessageCount	Maximum number of messages that can be in a queue before the broker evaluates whether the queue can be automatically deleted. **Type**: integer **Example**: 5 **Default value**: 0
	configDeleteAddresses	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: **OFF** The broker does not delete the address when the configuration file is reloaded. **FORCE** 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. **Type**: string **Example**: FORCE **Default value**: OFF

Entry	Sub-entry	Description and usage
	configDeleteQueues	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: **OFF** The broker does not delete the queue when the configuration file is reloaded. **FORCE** The broker deletes the queue when the configuration file is reloaded. If there are any messages in the queue, they are removed also. **Type**: string **Example**: FORCE **Default value**: OFF
	deadLetterAddress	The address to which the broker sends dead (that is, *undelivered*) messages. **Type**: string **Example**: DLA **Default value**: None
	deadLetterQueuePrefix	Prefix that the broker applies to the name of an automatically-created dead letter queue. **Type**: string **Example**: myDLQ. **Default value**: DLQ.

Entry	Sub-entry	Description and usage
	deadLetterQueueSuffix	Suffix that the broker applies to an automatically-created dead letter queue. **Type**: string **Example**: .DLQ **Default value**: None
	defaultAddressRoutingType	Routing type used on automatically-created addresses. **Type**: string **Example**: ANYCAST **Default value**: MULTICAST
	defaultConsumersBeforeDispatch	Number of consumers needed before message dispatch can begin for queues on an address. **Type**: integer **Example**: 5 **Default value**: 0
	defaultConsumerWindowSize	Default window size, in bytes, for a consumer. **Type**: integer **Example**: 300000 **Default value**: 1048576 (1024*1024)

Entry	Sub-entry	Description and usage
	defaultDelayBeforeDispatch	Default time, in milliseconds, that the broker waits before dispatching messages if the value specified for **defaultConsumersBeforeDispatch** has not been reached. **Type**: integer **Example**: 5 **Default value**: -1 (no delay)
	defaultExclusiveQueue	Specifies whether all queues on an address are exclusive queues by default. **Type**: Boolean **Example**: true **Default value**: false
	defaultGroupBuckets	Number of buckets to use for message grouping. **Type**: integer **Example**: 0 (message grouping disabled) **Default value**: -1 (no limit)
	defaultGroupFirstKey	Key used to indicate to a consumer which message in a group is first. **Type**: string **Example**: firstMessageKey **Default value**: None

Entry	Sub-entry	Description and usage
	defaultGroupRebalance	Specifies whether to rebalance groups when a new consumer connects to the broker. **Type**: Boolean **Example**: true **Default value**: false
	defaultGroupRebalancePauseDispatch	Specifies whether to pause message dispatch while the broker is rebalancing groups. **Type**: Boolean **Example**: true **Default value**: false
	defaultLastValueQueue	Specifies whether all queues on an address are last value queues by default. **Type**: Boolean **Example**: true **Default value**: false
	defaultLastValueKey	Default key to use for a last value queue. **Type**: string **Example**: stock_ticker **Default value**: None
	defaultMaxConsumers	Maximum number of consumers allowed on a queue at any time. **Type**: integer **Example**: 100 **Default value**: -1 (no limit)

Entry	Sub-entry	Description and usage
	defaultNonDestructive	Specifies whether all queues on an address are non-destructive by default. **Type**: Boolean **Example**: true **Default value**: false
	defaultPurgeOnNoConsumers	Specifies whether the broker purges the contents of a queue once there are no consumers. **Type**: Boolean **Example**: true **Default value**: false
	defaultQueueRoutingType	Routing type used on automatically-created queues. The default value is **MULTICAST**. **Type**: string **Example**: ANYCAST **Default value**: MULTICAST
	defaultRingSize	Default ring size for a matching queue that does not have a ring size explicitly set. **Type**: integer **Example**: 3 **Default value**: -1 (no size limit)

Entry	Sub-entry	Description and usage
	enableMetrics	Specifies whether a configured metrics plugin such as the Prometheus plugin collects metrics for a matching address or set of addresses. **Type**: Boolean **Example**: false **Default value**: true
	expiryAddress	Address that receives expired messages. **Type**: string **Example**: myExpiryAddress **Default value**: None
	expiryDelay	Expiration time, in milliseconds, applied to messages that are using the default expiration time. **Type**: integer **Example**: 100 **Default value**: -1 (no expiration time applied)
	expiryQueuePrefix	Prefix that the broker applies to the name of an automatically-created expiry queue. **Type**: string **Example**: myExp. **Default value**: EXP.

Entry	Sub-entry	Description and usage
	expiryQueueSuffix	Suffix that the broker applies to the name of an automatically-created expiry queue. **Type**: string **Example**: .EXP **Default value**: None
	lastValueQueue	Specify whether a queue uses only last values or not. **Type**: Boolean **Example**: true **Default value**: false
	managementBrowsePageSize	Specify how many messages a management resource can browse. **Type**: integer **Example**: 100 **Default value**: 200

Entry	Sub-entry	Description and usage
	match*	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. 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*'**. **Type**: string **Example**: 'myAddresses*' **Default value**: None
	maxDeliveryAttempts	Specifies how many times the broker attempts to deliver a message before sending the message to the configured dead letter address. **Type**: integer **Example**: 20 **Default value**: 10
	maxExpiryDelay	Expiration time, in milliseconds, applied to messages that are using an expiration time greater than this value. **Type**: integer **Example**: 20 **Default value**: -1 (no maximum expiration time applied)

Entry	Sub-entry	Description and usage
	maxRedeliveryDelay	Maximum value, in milliseconds, between message redelivery attempts made by the broker. **Type**: integer **Example**: 100 **Default value**: The default value is ten times the value of **redeliveryDelay**, which has a default value of **0**.
	maxSizeBytes	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". **Type**: string **Example**: 10Mb **Default value**: -1 (no limit)
	maxSizeBytesRejectThreshold	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. **Type**: integer **Example**: 500 **Default value**: -1 (no maximum size)

Entry	Sub-entry	Description and usage
	messageCounterHistoryDayLimit	Number of days for which a broker keeps a message counter history for an address. **Type**: integer **Example**: 5 **Default value**: 0
	minExpiryDelay	Expiration time, in milliseconds, applied to messages that are using an expiration time lower than this value. **Type**: integer **Example**: 20 **Default value**: -1 (no minimum expiration time applied)
	pageMaxCacheSize	Number of page files to keep in memory to optimize I/O during paging navigation. **Type**: integer **Example**: 10 **Default value**: 5
	pageSizeBytes	Paging size in bytes. Also supports byte notation such as **K**, **Mb**, and **GB**. **Type**: string **Example**: 20971520 **Default value**: 10485760 (approximately 10.5 MB)

Entry	Sub-entry	Description and usage
	redeliveryDelay	Time, in milliseconds, that the broker waits before redelivering a cancelled message. **Type**: integer **Example**: 100 **Default value**: 0
	redeliveryDelayMultiplier	Multiplying factor to apply to the value of **redeliveryDelay**. **Type**: number **Example**: 5 **Default value**: 1
	redeliveryCollisionAvoidanceFactor	Multiplying factor to apply to the value of **redeliveryDelay** to avoid collisions. **Type**: number **Example**: 1.1 **Default value**: 0
	redistributionDelay	Time, in milliseconds, that the broker waits after the last consumer is closed on a queue before redistributing any remaining messages. **Type**: integer **Example**: 100 **Default value**: -1 (not set)

Entry	Sub-entry	Description and usage
	retroactiveMessageCount	Number of messages to keep for future queues created on an address. **Type**: integer **Example**: 100 **Default value**: 0
	sendToDlaOnNoRoute	Specify whether a message will be sent to the configured dead letter address if it cannot be routed to any queues. **Type**: Boolean **Example**: true **Default value**: false
	slowConsumerCheckPeriod	How often, in **seconds**, that the broker checks for slow consumers. **Type**: integer **Example**: 15 **Default value**: 5
	slowConsumerPolicy	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. **Type**: string **Example**: KILL **Default value**: NOTIFY

Entry	Sub-entry	Description and usage
	slowConsumerThreshold	Minimum rate of message consumption, in messages per second, before a consumer is considered slow. **Type**: integer **Example**: 100 **Default value**: -1 (not set)
console		Configuration of broker management console.
	expose	Specify whether to expose the management console port. **Type**: Boolean **Example**: true **Default value**: false
	sslEnabled	Specify whether to use SSL on the management console port. **Type**: Boolean **Example**: true **Default value**: false

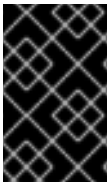
Entry	Sub-entry	Description and usage
	sslSecret	Secret where broker key store, trust store, and their corresponding passwords (all Base64-encoded) are stored. If you do not specify a value for **sslSecret**, the console uses a default secret name. The default secret name has a format of ***<Custom-Resource-name>*-console-secret**. **Type**: string **Example**: my-broker-deployment-console-secret **Default value**: Not specified
	useClientAuth	Specify whether the management console requires client authorization. **Type**: Boolean **Example**: true **Default value**: false
upgrades		

Entry	Sub-entry	Description and usage
	enabled	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. **Type**: Boolean **Example**: true **Default value**: false
	minor	Specify whether to allow the Operator to automatically update the **deploymentPlan.image** value when you update the value of **version** from one *minor* version of AMQ Broker to another, for example, from **7.6.0** to **7.7.0**. **Type**: Boolean **Example**: true **Default value**: false

Entry	Sub-entry	Description and usage
version		Specify a target *minor* 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**. **Type**: string **Example**: 7.7.0 **Default value**: Current version of AMQ Broker

10.1.2. 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*	Address name to be created on broker. **Type**: string **Example**: address0 **Default value**: Not specified

Entry	Description and usage
queueName*	Queue name to be created on broker. **Type**: string **Example**: queue0 **Default value**: Not specified
removeFromBrokerOnDelete*	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. **Type**: Boolean **Example**: true **Default value**: false
routingType*	Routing type to be used; **anycast** or **multicast**. **Type**: string **Example**: anycast **Default value**: Not specified

10.2. 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 10.1. 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_ANYCAST_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 AMQ_CREDENTIAL_SECRET.

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 stateful sets.
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.
OPENSHIFT_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

10.3. 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 2021-08-04 13:47:28 UTC