Red Hat AMQ 7.4

Deploying AMQ Broker on OpenShift Container Platform

For Use with AMQ Broker 7.4

Last Updated: 2020-01-16

Red Hat AMQ 7.4 Deploying AMQ Broker on OpenShift Container Platform

For Use with AMQ Broker 7.4

Legal Notice

Abstract

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

Table of Contents

CHAPTER 1. INTRODUCTION

Red Hat AMQ Broker 7.4 is available as a containerized image that is provided for use with OpenShift Container Platform 4.1 (AMQ Broker on OCP).

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 4.1 image version compatibility, see the OpenShift and Atomic Platform Tested Integrations page.

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. INSTALLING AMQ BROKER ON OPENSHIFT CONTAINER PLATFORM

2.1. INSTALLING THE AMQ BROKER 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 and Tutorials.

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:

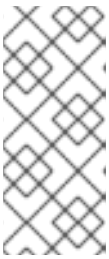
```
$ oc login -u system:admin
$ oc project openshift
```

   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.

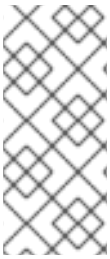
```
$ oc replace --force  -f \
https://raw.githubusercontent.com/jboss-container-images/jboss-amq-7-broker-openshift-image/74-7.4.0.GA/amq-broker-7-image-streams.yaml
```

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

```
$ for template in amq-broker-74-basic.yaml \
amq-broker-74-ssl.yaml \
amq-broker-74-custom.yaml \
amq-broker-74-persistence.yaml \
amq-broker-74-persistence-ssl.yaml \
amq-broker-74-persistence-clustered.yaml \
amq-broker-74-persistence-clustered-ssl.yaml;
 do
 oc replace --force -f \
https://raw.githubusercontent.com/jboss-container-images/jboss-amq-7-broker-openshift-image/74-7.4.0.GA/templates/${template}
 done
```

CHAPTER 3. DEPLOYING AMQ BROKER ON OPENSHIFT CONTAINER PLATFORM USING APPLICATION TEMPLATES

The procedures in this section show how to prepare a broker deployment and an example of using the OpenShift Container Platform web console to deploy a basic broker instance. For examples of deploying other broker configurations, see Tutorials.



NOTE

The following procedures assume that the broker image and application templates you installed in Installing AMQ Broker on OpenShift Container Platform 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**.

3.1. PREPARING AN AMQ 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 AMQ Broker image streams and application templates.

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. Add the secret to the service account created earlier:

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

3.2. 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 Tutorials.

Prerequisites

- You have already prepared the broker deployment. See Preparing an AMQ Broker deployment .
- 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.
- AMQ Broker 7.4 has been designated as a Long Term Support (LTS) release version. Bug fixes and security advisories will be made available for AMQ Broker 7.4 in a series of micro releases (7.4.1, 7.4.2, and so on) for a period of at least 12 months. This means that you will be able to get recent bug fixes and security advisories for AMQ Broker *without* having to upgrade to a new minor release. For more information, see Long Term Support for AMQ Broker .

3.2.1. Create 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-74-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 3.1. Basic broker application template

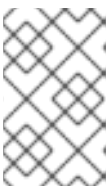
Environment variable	Display Name	Value	Description
AMQ_PROTOCOL	AMQ Protocols	openwire,amqp,stomp,mqtt,hornetq	The protocols to be accepted by the broker
AMQ_QUEUES	Queues	demoQueue	Creates an anycast queue called demoQueue
AMQ_ADDRESSES	Addresses	demoTopic	Creates an address (or topic) called demoTopic. By default, this address has no assigned routing type.
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

3.2.2. Deploy and start 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. From the search results, choose an image repository based on the information in this table.

AMQ Broker Version	Container Image Name	Repository Name
7.4.0	AMQ Broker	amq7/amq-broker
7.4.1, 7.4.2	AMQ Broker LTS	amq7/amq-broker-lts-rhel7

When you select a container image, the corresponding 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. The latest tags associated with each version of AMQ Broker 7.4 are shown in the following table.

AMQ Broker Version	Repository Name	Latest Container Image Tag
7.4.0	amq7/amq-broker	7.4-4
7.4.1 (previous LTS version)	amq7/amq-broker-lts-rhel7	7.4-14
7.4.2 (current LTS version)	amq7/amq-broker-lts-rhel7	7.4-15

c. Click the **Get This Image** tab.

d. 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-lts-rhel7:7.4-15'
    ..
    imagePullSecrets:
      - name: {PULL-SECRET-NAME}
```

e. Click **Save**.

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

```
$ oc import-image amq7/amq-broker-lts-rhel7:7.4-15 --from=registry.redhat.io/amq7/amq-broker-lts-rhel7 --confirm
```

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

9. 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.

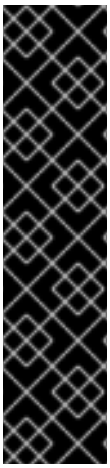
```
// Get the Pod names and internal IP Addresses
$ oc get pods -o wide

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

CHAPTER 4. UPGRADING AMQ BROKER ON OPENSHIFT CONTAINER PLATFORM

The procedures in this section show how to upgrade an existing broker deployment on OpenShift Container Platform.

IMPORTANT

AMQ Broker 7.4 has been designated as a Long Term Support (LTS) release version. Bug fixes and security advisories will be made available for AMQ Broker 7.4 in a series of micro releases (7.4.1, 7.4.2, and so on) for a period of at least 12 months. This means that you will be able to get recent bug fixes and security advisories for AMQ Broker *without* having to upgrade to a new minor release. For more information, see Long Term Support for AMQ Broker.



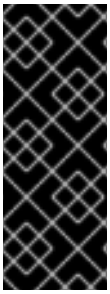
IMPORTANT

To upgrade an existing AMQ Broker deployment on OpenShift Container Platform 3.11 to run on OpenShift Container Platform 4.1, 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 (Technical Preview feature).
- Deploying AMQ Broker on OpenShift Container Platform using Application Templates

4.1. UPGRADING AN OPERATOR-BASED BROKER DEPLOYMENT

The following procedure shows how to upgrade an existing broker deployment that is based on the AMQ Broker Operator.



IMPORTANT

The AMQ Broker Operator is a Technology Preview feature for AMQ Broker 7.4, including all micro releases (7.4.1, 7.4.2, and so on). Technology Preview features are not supported with Red Hat production service-level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them for production. For more information about technology preview at Red Hat, see Red Hat Technology Preview Features Support Scope.

Prerequisites

- You have used the AMQ Broker Operator to create a broker deployment. For more information, see Deploying AMQ Broker on OpenShift Container Platform using an Operator .

Procedure

1. In the **deploy/crs** directory of the Operator repository that you cloned during your initial installation, open the **broker_v1alpha1_activemqartemis_cr.yaml** file.

2. Edit the **image** element to specify the latest broker container image. Specify an image based on the information in this table:

AMQ Broker Version	Container Image Name
7.4.0	registry.redhat.io/amq7/amq-broker:7.4-4
7.4.1 (previous LTS version)	registry.redhat.io/amq7/amq-broker-lts-rhel7:7.4-14
7.4.2 (current LTS version)	registry.redhat.io/amq7/amq-broker-lts-rhel7:7.4-15

For example:

```
image: registry.redhat.io/amq7/amq-broker-lts-rhel7:7.4-15
```

3. Apply the CR change.

```
$ oc apply -f deploy/crs/broker_v1alpha1_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.

4.2. UPGRADING TEMPLATES-BASED BROKER DEPLOYMENTS

The following procedures show how to upgrade an existing broker deployment that is based on application templates.

4.2.1. 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, 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 **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 a broker container image. Specify an image based on the information in this table.

AMQ Broker Version	Container Image Name
7.4.0	registry.redhat.io/amq7/amq-broker:7.4-4
7.4.1 (previous LTS version)	registry.redhat.io/amq7/amq-broker-lts-rhel7:7.4-14
7.4.2 (current LTS version)	registry.redhat.io/amq7/amq-broker-lts-rhel7:7.4-15

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-lts-rhel7:7.4-15'
..
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, 7.4-1, 7.4-2, and so on. If you specify a tag name without a final digit (**7.4**, 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.

4.2.2. 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, 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 **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 a broker container image. Specify an image based on the information in this table.

AMQ Broker Version	Container Image Name
7.4.0	registry.redhat.io/amq7/amq-broker:7.4-4
7.4.1 (previous LTS version)	registry.redhat.io/amq7/amq-broker-lts-rhel7:7.4-14

AMQ Broker Version	Container Image Name
7.4.2 (current LTS version)	registry.redhat.io/amq7/amq-broker-lts-rhel7:7.4-15

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-lts-rhel7:7.4-15'
..
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 5. CONFIGURING SSL FOR AMQ BROKER ON OPENSHIFT CONTAINER PLATFORM

5.1. CONFIGURING SSL

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

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

For a more complete example of deploying a broker instance that supports SSL, see Deploying a basic broker with SSL.

Procedure

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
```

5.2. GENERATING THE AMQ BROKER SECRET

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

Procedure

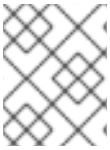
- At the command line, run the following commands:

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

5.3. CREATING AN SSL ROUTE

After the AMQ Broker on OpenShift Container Platform image has been deployed, you need to create an SSL Route for the AMQ Broker transport protocol port to allow connections to AMQ Broker outside of OpenShift. You can only expose SSL Routes because the OpenShift router requires SNI to send traffic to the correct Service.

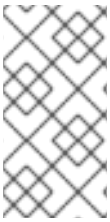
Selecting **Passthrough** for **TLS Termination** relays all communication to AMQ Broker without the OpenShift router decrypting and resending it.



NOTE

Regular HTTP traffic does not require a TLS passthrough Route because the OpenShift router uses **HAProxy**, which is a HTTP proxy.

External clients for AMQ Broker on OpenShift Container Platform must specify the OpenShift router port (443, by default) when setting the broker URL for SSL connections. Otherwise, AMQ Broker attempts to use the default SSL port (61617).



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 4.1 Routes .

Also, including the failover protocol in the URL preserves the client connection in case the pod is restarted or upgraded, or a disruption occurs on the router. Both of these settings are shown below.

```
...
factory.setBrokerURL("failover://ssl://<route-to-broker-pod>:443");
...
```

NOTE

External clients do not support HA.

The default ports for the various AMQ Broker transport protocols are shown in the table.

Table 5.1. Default ports for AMQ Broker transport protocols

AMQ Broker transport protocol	Default port
All protocols (OpenWire, AMQP, STOMP, MQTT, and HornetQ)	61616

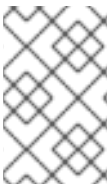
AMQ Broker transport protocol	Default port
All protocols -SSL (OpenWire AMQP, STOMP, MQTT, and HornetQ)	61617
AMQP	5672
AMQP -SSL	5671
MQTT	1883
MQTT -SSL	8883
STOMP	61613
STOMP -SSL	61612

Additional resources

- For more information on cluster networking, see Secured Routes.

CHAPTER 6. CUSTOMIZING AMQ BROKER CONFIGURATION FILES FOR DEPLOYMENT

If you are using a template from an alternative repository, AMQ Broker configuration files such as **artemis-users.properties** can be included. When the image is downloaded for deployment, these files are copied from **<amq-home>/conf/** to the **<broker-instance-dir>/etc/** directory on AMQ Broker, which is committed to the container and pushed to the OpenShift registry.



NOTE

If using this method, ensure that the placeholders in the configuration files (such as **AUTHENTICATION**) are not removed. The placeholders are necessary for building the AMQ Broker on OpenShift Container Platform image.

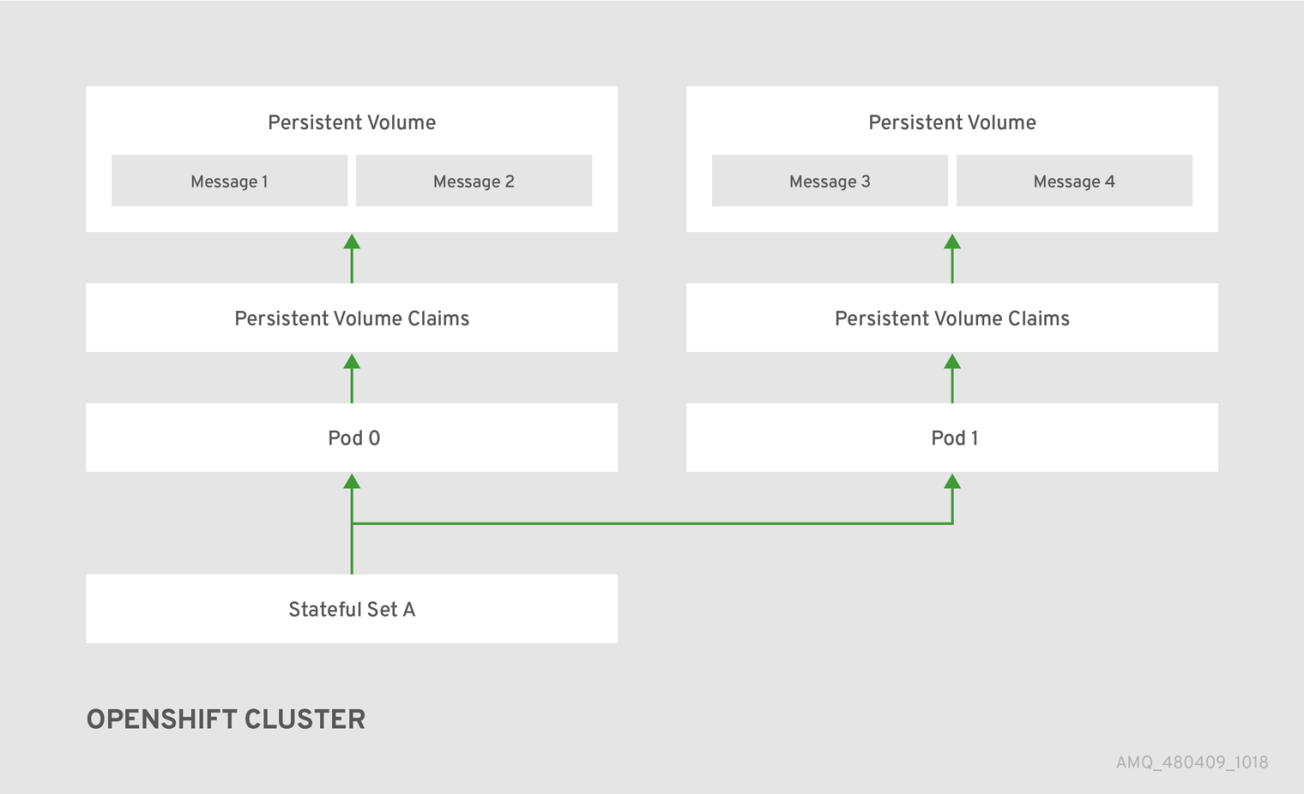
CHAPTER 7. HIGH AVAILABILITY

7.1. HIGH AVAILABILITY OVERVIEW

The term *high availability* refers to a system that is capable of remaining operational, even when part of that system fails or is taken offline. With Broker on OCP, specifically, HA refers to both maintaining the availability of brokers and the integrity of the messaging data if a broker fails.

In an HA configuration on AMQ Broker on OpenShift Container Platform, you run multiple instances of a broker pod simultaneously. Each individual broker pod writes its message data to a *persistent volume* (PVs), which logically define the storage volumes in the system. If a broker pod fails or is taken offline, the message data stored in that PV is redistributed to an alternative available broker, which then stores it in its own PV.

Figure 7.1. StatefulSet working normally



When you take a broker pod offline, the StatefulSet is scaled down and you must manage what happens to the message data in the unattached PV. To migrate the messages held in the PV associated with the now-offline pod, you use the scaledown controller. The process of migrating message data in this fashion is sometimes referred to as *pod draining*.

Additional resources

- To enable High Availability on AMQ Broker on OpenShift Container Platform, use the StatefulSets tutorial.

7.2. MESSAGE MIGRATION

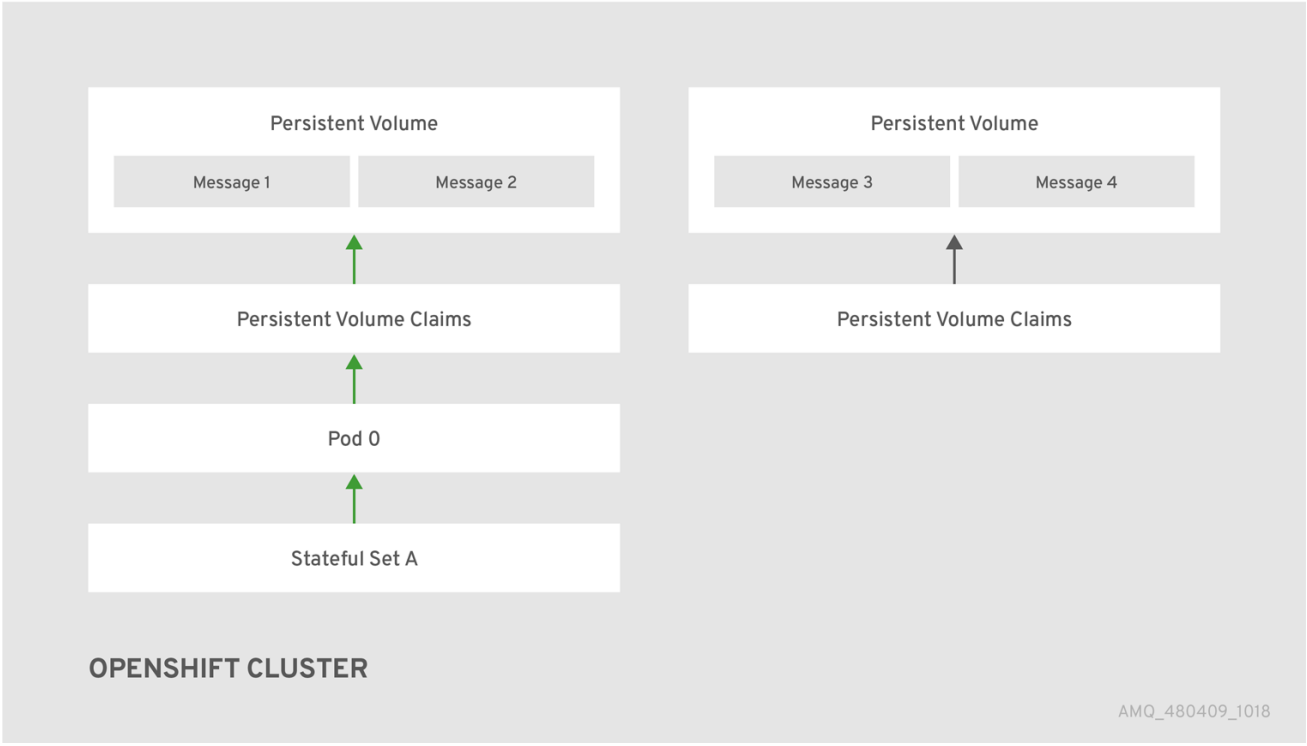
IMPORTANT

Message migration, which is enabled by use of a scaledown controller, is currently a Technology Preview feature. Technology Preview features are not supported with Red Hat production service-level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them for production. For more information about technology preview at Red Hat, see Technology Preview Support Scope.

Message migration is how you ensure the integrity of messaging data in a Kubernetes StatefulSet. Message migration, which uses a method called *pod draining*, refers to the removal and redistribution of "orphaned" messages from a persistent volume, due to broker pod failure or intentional scaledown.

To enable message migration in the event of broker pod failure or intentional scaledown , deploy a scaledown controller. For more information, see Migrating Messages Upon Scaledown.

Figure 7.2. One of the brokers has gone down



7.3. HOW DOES MESSAGE MIGRATION WORK?

A scaledown controller image exists for AMQ Broker on OpenShift Container Platform. It runs within the same project namespace as the broker Pods.

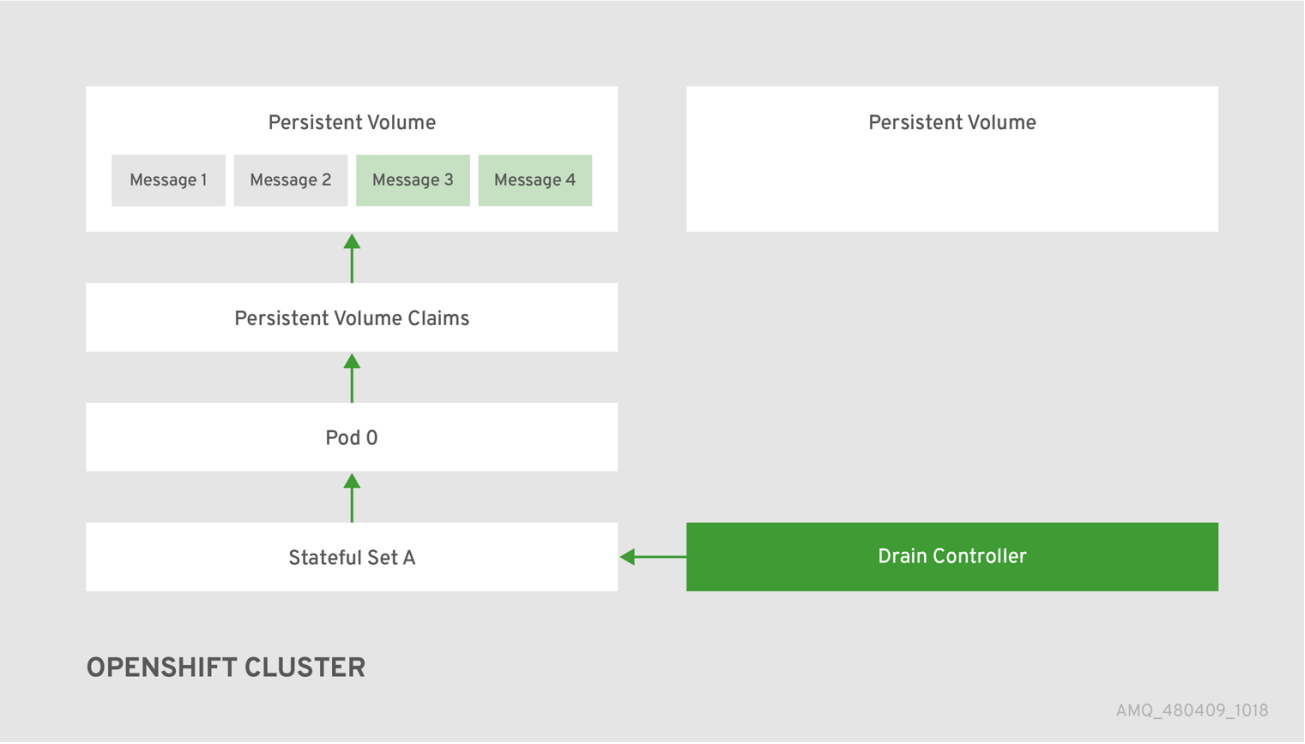
The scaledown controller registers itself and listens for Kubernetes events that are related to persistent volume claims (PVCs) in the project namespace.

The scaledown controller checks for PVCs that have been orphaned by looking at the ordinal on the volume claim. The ordinal on the volume claim is compared to the ordinal on the existing broker Pods, which are part of the StatefulSet in the project namespace.

If the ordinal on the volume claim is greater than the ordinal on the existing broker Pods, then the Pod at that ordinal has been terminated and the data must be migrated to another broker.

When these conditions are met, a drainer Pod is started. The drainer Pod runs the broker and executes the message migration. Then the drainer Pod identifies an alternative broker Pod to which the orphaned PVC messages can be migrated.

Figure 7.3. The scaledown controller registers itself, deletes the PVC, and redistributes messages on the PersistentVolume.



After the messages are successfully migrated to an operational broker Pod, the drainer Pod shuts down and the scaledown controller removes the orphaned PVC.

Additional resources

- To learn more about deploying a scaledown controller, see Migrating Messages Upon Scaledown.

CHAPTER 8. TUTORIALS

Prerequisites

- These procedures assume an OpenShift Container Platform 4.1 instance similar to that created in OpenShift Container Platform 4.1 Getting Started .

The following procedures example how to use application templates to create various deployments of brokers.

8.1. DEPLOYING A BASIC BROKER WITH SSL

Deploy a basic broker that is ephemeral and supports SSL.

8.1.1. Deploying the image and template

Prerequisites

- This tutorial builds upon Preparing a Broker.
- 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-74-ssl** template which is labeled **Red Hat AMQ Broker 7.4 (Ephemeral, with SSL)**.
6. Set the following values in the configuration and click **Create**.

Table 8.1. Example template

Environment variable	Display Name	Value	Description
AMQ_PROTOCOL	AMQ Protocols	openwire,amqp,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	The username the client uses
AMQ_PASSWORD	AMQ Password	password	The password the client uses with the username
AMQ_TRUSTSTORE	Trust Store Filename	broker.ts	The SSL truststore file name
AMQ_TRUSTSTORE_PASSWORD	Truststore Password	password	The password used when creating the Truststore
AMQ_KEYSTORE	AMQ Keystore Filename	broker.ks	The SSL keystore file name
AMQ_KEYSTORE_PASSWORD	AMQ Keystore Password	password	The password used when creating the Keystore

8.1.2. Deploying the application

After creating the application, deploy it to create a Pod and start the broker.

Procedure

1. Click **Deployments** in the OpenShift Container Platform web console.
2. Click the **broker-amq** deployment.
3. Click **Deploy** to deploy the application.
4. Click the broker Pod and then click the **Logs** tab to verify the state of the broker.
   If the broker logs have not loaded, and the Pod status shows **ErrImagePull** or **ImagePullBackOff**, your deployment configuration was not able to directly pull the specified broker image from the Red Hat Container Registry. In this case, edit your deployment configuration to reference the correct broker image name and the image pull secret name associated with the account used for authentication in the Red Hat Container Registry. Then, you can import the broker image and start the broker. To do this, complete steps similar to those in Deploy and start the broker application .

8.1.3. Creating a Route

Create a Route for the broker so that clients outside of OpenShift Container Platform can connect using SSL. By default, all 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.

Procedure

1. Click **Services → broker-amq-tcp-ssl**
2. Click **Actions → Create route**
3. To display the TLS parameters, select the **Secure route** check box .
4. From the **TLS Termination** drop-down menu, choose **Passthrough**. This selection relays all communication to AMQ Broker without the OpenShift router decrypting and resending it.
5. To view the Route, click **Routes**. For example:

```
https://broker-amq-tcp-amq-demo.router.default.svc.cluster.local
```

This hostname will be used by external clients to connect to the broker using SSL with SNI.

Additional resources

- For more information on Routes in the OpenShift Container Platform, see Routes.

8.2. 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 has an attached storage device that it uses for its journal. When a broker Pod is created, it is allocated 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 broker.
- Completion of the Deploying a basic broker tutorial is recommended.

8.2.1. Deploy the image and template

Procedure

1. Navigate to the OpenShift web console and log in.
2. Select the **amq-demo** project space.
3. Click **Add to Project → Browse catalog** to list all of the default image streams and templates.

4. Use the **Filter** search bar to limit the list to those that match **amq**. You might need to click **See all** to show the desired application template.

5. Select the **amq-broker-74-persistence-ssl** template, which is labelled **Red Hat AMQ Broker 7.4 (Persistence, with SSL)**.

6. Set the following values in the configuration and click **create**.

Table 8.2. Example template

Environment variable	Display Name	Value	Description
AMQ_PROTOCOL	AMQ Protocols	openwire,amqp,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
AMQ_TRUSTSTORE	Trust Store Filename	broker.ts	The SSL truststore file name
AMQ_TRUSTSTORE_PASSWORD	Truststore Password	password	The password used when creating the Truststore
AMQ_KEYSTORE	AMQ Keystore Filename	broker.ks	The SSL keystore file name
AMQ_KEYSTORE_PASSWORD	AMQ Keystore Password	password	The password used when creating the Keystore

8.2.2. Deploy the application

Once the application has been created it needs to be deployed. Deploying the application creates a Pod and starts the broker.

Procedure

1. Click **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 Deploy and start 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:

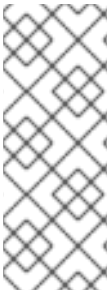
```
sh-4.2$ broker/bin/artemis consumer --destination queue://demoQueue
Consumer:: filter = null
Consumer ActiveMQQueue[demoQueue], thread=0 wait until 1000 messages are consumed
Received 1000
Consumer ActiveMQQueue[demoQueue], thread=0 Consumed: 1000 messages
Consumer ActiveMQQueue[demoQueue], thread=0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

Additional resources

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

8.2.3. Creating a Route

Create a Route for the broker so that clients outside of OpenShift Container Platform can connect using SSL. By default, the broker protocols are available through the 61617/TCP port.



NOTE

If you scale your deployment up to multiple brokers in a cluster, you must manually create a Service and a Route for each broker, and then use each Service-and-Route combination to direct a given client to a given broker, or broker list. For an example of configuring multiple Services and Routes to connect clustered brokers to their own instances of the AMQ Broker management console, see Creating routes for the AMQ Broker management console.

Procedure

1. Click **Services → broker-amq-tcp-ssl**.
2. Click **Actions → Create a route**.
3. To display the TLS parameters, select the **Secure route** check box.
4. From the **TLS Termination** drop-down menu, choose **Passthrough**. This selection relays all communication to AMQ Broker without the OpenShift router decrypting and resending it.
5. To view the route, click **Routes**. For example:

```
https://broker-amq-tcp-amq-demo.router.default.svc.cluster.local
```

This hostname will be used by external clients to connect to the broker using SSL with SNI.

Additional resources

- For more information on routes in the OpenShift Container Platform, see Routes.

8.3. DEPLOYING A SET OF CLUSTERED BROKERS

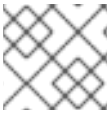
Deploy a clustered set of brokers where each broker runs in its own Pod.

8.3.1. Distributing messages

Message distribution is configured to use *ON_DEMAND*. This means that when messages arrive at a clustered broker, the messages are distributed in a round-robin fashion to any broker that has consumers.

This message distribution policy safeguards against messages getting stuck on a specific broker while a consumer, connected either directly or through the OpenShift router, is connected to a different broker.

The redistribution delay is zero by default. If a message is on a queue that has no consumers, it will be redistributed to another broker.



NOTE

When redistribution is enabled, messages can be delivered out of order.

8.3.2. Deploy the image and template

Prerequisites

- This procedure builds upon Preparing a broker.
- 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-74-persistence-clustered** template which is labeled **Red Hat AMQ Broker 7.4 (no SSL, clustered)**.
6. Set the following values in the configuration and click **create**.

Table 8.3. Example template

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

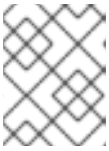
Environment variable	Display Name	Value	Description
VOLUME_CAPACITY	AMQ Volume Size	1Gi	The persistent volume size created for the journal
AMQ_CLUSTERED	Clustered	true	This needs to be true to ensure the brokers cluster
AMQ_CLUSTER_USER	cluster user	generated	The username the brokers use to connect with each other
AMQ_CLUSTER_PASSWORD	cluster password	generated	The password the brokers use to connect with each other
AMQ_USER	AMQ Username	amq-demo-user	The username the client uses
AMQ_PASSWORD	AMQ Password	password	The password the client uses with the username

8.3.3. Deploying the application

Once the application has been created it needs to be deployed. Deploying the application creates a Pod and starts the broker.

Procedure

1. Click **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 Deploy and start the broker application .

7. Verify that the brokers have clustered with the new Pod by checking the logs.

```
$ oc logs broker-amq-2
```

This shows the logs of the new broker and an entry for a clustered bridge created between the brokers:

```
2018-08-29 07:43:55,779 INFO  [org.apache.activemq.artemis.core.server] AMQ221027:
Bridge ClusterConnectionBridge@1b0e9e9d [name=$.artemis.internal.sf.my-
cluster.4333c830-ab5f-11e8-afb8-0a580a82006e,
queue=QueueImpl[name=$.artemis.internal.sf.my-cluster.4333c830-ab5f-11e8-afb8-
0a580a82006e, postOffice=PostOfficeImpl
[server=ActiveMQServerImpl::serverUUID=9cedb69d-ab5e-11e8-87a4-0a580a82006c],
temp=false]@5e0c0398 targetConnector=ServerLocatorImpl (identity=(Cluster-connection-
bridge::ClusterConnectionBridge@1b0e9e9d [name=$.artemis.internal.sf.my-
cluster.4333c830-ab5f-11e8-afb8-0a580a82006e,
queue=QueueImpl[name=$.artemis.internal.sf.my-cluster.4333c830-ab5f-11e8-afb8-
0a580a82006e, postOffice=PostOfficeImpl
[server=ActiveMQServerImpl::serverUUID=9cedb69d-ab5e-11e8-87a4-0a580a82006c],
temp=false]@5e0c0398 targetConnector=ServerLocatorImpl [initialConnectors=
[TransportConfiguration(name=artemis, factory=org-apache-activemq-artemis-core-remoting-
impl-netty-NettyConnectorFactory) ?port=61616&host=10-130-0-110],
discoveryGroupConfiguration=null]]::ClusterConnectionImpl@806813022[nodeUUID=9cedb69d
-ab5e-11e8-87a4-0a580a82006c, connector=TransportConfiguration(name=artemis,
factory=org-apache-activemq-artemis-core-remoting-impl-netty-NettyConnectorFactory) ?
port=61616&host=10-130-0-108, address=,
server=ActiveMQServerImpl::serverUUID=9cedb69d-ab5e-11e8-87a4-0a580a82006c]))
[initialConnectors=[TransportConfiguration(name=artemis, factory=org-apache-activemq-
artemis-core-remoting-impl-netty-NettyConnectorFactory) ?port=61616&host=10-130-0-110],
discoveryGroupConfiguration=null]] is connected
```

8.3.4. Creating a Route for the AMQ Broker management console

The clustering templates do not expose the console by default. This is because the the OpenShift proxy would load balance around each broker in the cluster and it would not be possible to control which broker console is connected.

The following procedure shows you how to create a Route for the AMQ Broker management console to connect to brokers in the cluster.

NOTE

If your cluster uses SSL, 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 more information, see Creating Routes for the AMQ Broker management console.

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: console-jolokia
spec:
  port:
    targetPort: console-jolokia
  to:
    kind: Service
    name: broker-amq-headless
    weight: 100
  wildcardPolicy: Subdomain
  host: star.broker-amq-headless.amq-demo.svc
```

NOTE

The **host: star.broker-amq-headless.amq-demo.svc** configuration is the hostname used for each Pod in the broker. The star is replaced by the Pod name, so if the Pod name is **broker-amq-0**, the hostname is **broker-amq-0.broker-amq-headless.amq-demo.svc**

3. Add entries into the /etc/hosts file to map the route names onto the IP address of the OpenShift cluster. For example, if you have three Pods, then add entries as shown below.

```
10.0.0.1 broker-amq-0.broker-amq-headless.amq-demo.svc
10.0.0.1 broker-amq-1.broker-amq-headless.amq-demo.svc
10.0.0.1 broker-amq-2.broker-amq-headless.amq-demo.svc
```

4. Navigate to the console using the address **http://broker-amq-0.broker-amq-headless.amq-demo.svc** in a browser.

Additional resources

- For more information on the clustering of brokers see Enabling Message Redistribution.

8.4. DEPLOYING A SET OF CLUSTERED SSL BROKERS

Deploy a clustered set of brokers, where each broker runs in its own Pod and the broker is configured to accept connections using SSL.

8.4.1. Distributing messages

Message distribution is configured to use *ON_DEMAND*. This means that when messages arrive at a clustered broker, the messages are distributed in a round-robin fashion to any broker that has consumers.

This message distribution policy safeguards against messages getting stuck on a specific broker while a consumer, connected either directly or through the OpenShift router, is connected to a different broker.

The redistribution delay is non-zero by default. If a message is on a queue that has no consumers, it will be redistributed to another broker.

NOTE

When redistribution is enabled, messages can be delivered out of order.

8.4.2. Deploying the image and template

Prerequisites

- This procedure builds upon Preparing a broker.
- 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-74-persistence-clustered-ssl** template which is labeled **Red Hat AMQ Broker 7.4 (SSL, clustered)**.
6. Set the following values in the configuration and click **create**.

Table 8.4. Example template

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

8.4.3. 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
```

8.4.4. 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 and the broker is configured to accept connections using SSL. See Deploying a set of clustered SSL brokers.

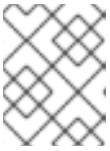
Procedure

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-74-persistence-clustered-ssl
  name: amq2-amq-console-XXXX
  namespace: amq74-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 is 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 using SSL.
   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-amq74-p-c-ssl-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 about messaging in a broker cluster, see Enabling Message Redistribution.

8.5. DEPLOYING A BROKER WITH CUSTOM CONFIGURATION

Deploy a broker with custom configuration. Although functionality can be obtained by using templates, broker configuration can be customized if needed.

Prerequisites

- This tutorial builds upon Preparing a broker.
- Completion of the Deploying a basic broker tutorial is recommended.

8.5.1. Deploy the image and template

Procedure

1. Navigate to the OpenShift web console and log in.
2. Select the **amq-demo** project space.
3. Click **Add to Project** > **Browse catalog** to list all of the default image streams and templates.
4. Use the **Filter** search bar to limit results to those that match **amq**. Click **See all** to show the desired application template.
5. Select the **amq-broker-74-custom** template which is labeled **Red Hat AMQ Broker 7.4(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.

8.5.2. Deploy the application

Once the application has been created it needs to be deployed. Deploying the application creates a Pod and starts the broker.

Procedure

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

8.6. BASIC SSL CLIENT EXAMPLE

Implement a client that sends and receives messages from a broker configured to use SSL, using the Qpid JMS client.

Prerequisites

- This tutorial builds upon Preparing a Broker.

- Completion of the Deploying a Basic Broker with SSL tutorial is recommended.
- AMQ JMS Examples

8.6.1. Configuring the client

Create a sample client that can be updated to connect to the SSL broker. The following procedure builds upon AMQ JMS Examples.

Procedure

1. Add an entry into your /etc/hosts file to map the route name onto the IP address of the OpenShift cluster:

```
10.0.0.1 broker-amq-tcp-amq-demo.router.default.svc.cluster.local
```

2. Update the jndi.properties configuration file to use the route, truststore and keystore created previously, for example:

```
connectionfactory.myFactoryLookup = amqps://broker-amq-tcp-amq-demo.router.default.svc.cluster.local:8443?transport.keyStoreLocation=<keystore-path>client.ks&transport.keyStorePassword=password&transport.trustStoreLocation=<truststore-path>/client.ts&transport.trustStorePassword=password&transport.verifyHost=false
```

3. Update the jndi.properties configuration file to use the queue created earlier.

```
queue.myDestinationLookup = demoQueue
```

4. Execute the sender client to send a text message.
5. Execute the receiver client to receive the text message. You should see:

```
Received message: Message Text!
```

8.7. EXTERNAL CLIENTS USING SUB-DOMAINS EXAMPLE

Expose a clustered set of brokers through a node port and connect to it using the core JMS client. This enables clients to connect to a set of brokers which are configured using the **amq-broker-74-persistence-clustered-ssl** template.

8.7.1. Exposing the brokers

Configure the brokers so that the cluster of brokers are externally available and can be connected to directly, bypassing the OpenShift router. This is done by creating a route that exposes each pod using its own hostname.

Prerequisites

- Deploying a set of clustered brokers

Procedure

1. Choose **import YAML/JSON** from **Add to Project** drop down
2. Enter the following and click create.

```
apiVersion: v1
kind: Route
metadata:
  labels:
    app: broker-amq
    application: broker-amq
  name: tcp-ssl
spec:
  port:
    targetPort: ow-multi-ssl
  tls:
    termination: passthrough
  to:
    kind: Service
    name: broker-amq-headless
    weight: 100
  wildcardPolicy: Subdomain
  host: star.broker-ssl-amq-headless.amq-demo.svc
```

NOTE

The important configuration here is the wildcard policy of **Subdomain**. This allows each broker to be accessible through its own hostname.

8.7.2. Connecting the clients

Create a sample client that can be updated to connect to the SSL broker. The steps in this procedure build upon the AMQ JMS Examples.

Procedure

1. Add entries into the /etc/hosts file to map the route name onto the actual IP addresses of the brokers:

```
10.0.0.1 broker-amq-0.broker-ssl-amq-headless.amq-demo.svc broker-amq-1.broker-ssl-amq-headless.amq-demo.svc broker-amq-2.broker-ssl-amq-headless.amq-demo.svc
```

2. Update the jndi.properties configuration file to use the route, truststore, and keystore created previously, for example:

```
connectionfactory.myFactoryLookup = amqps://broker-amq-0.broker-ssl-amq-headless.amq-demo.svc:443?transport.keyStoreLocation=/home/ataylor/projects/jboss-amq-7-broker-openshift-image/client.ks&transport.keyStorePassword=password&transport.trustStoreLocation=/home/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.

8.8. EXTERNAL CLIENTS USING PORT BINDING EXAMPLE

Expose a clustered set of brokers through a NodePort and connect to it using the core JMS client. This enables clients that do not support SNI or SSL. It is used with clusters configured using the **amq-broker-74-persistence-clustered** template.

8.8.1. Exposing the brokers

Configure the brokers so that the cluster of brokers are externally available and can be connected to directly, bypassing the OpenShift router. This is done by creating a service that uses a NodePort to load balance around the clusters.

Prerequisites

- Deploying a set of clustered brokers

Procedure

1. Choose **import YAML/JSON** from **Add to Project** drop down.

2. Enter the following and click create.

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    description: The broker's OpenWire port.
    service.alpha.openshift.io/dependencies: >-
      [{"name": "broker-amq-amqp", "kind": "Service"},{"name":
      "broker-amq-mqtt", "kind": "Service"},{"name": "broker-amq-stomp", "kind":
      "Service"}]
  creationTimestamp: '2018-08-29T14:46:33Z'
  labels:
    application: broker
    template: amq-broker-74-statefulset-clustered
  name: broker-external-tcp
  namespace: amq-demo
  resourceVersion: '2450312'
  selfLink: /api/v1/namespaces/amq-demo/services/broker-amq-tcp
  uid: 52631fa0-ab9a-11e8-9380-c280f77be0d0
spec:
  externalTrafficPolicy: Cluster
  ports:
```

```
  - nodePort: 30001
    port: 61616
    protocol: TCP
    targetPort: 61616
  selector:
    deploymentConfig: broker-amq
  sessionAffinity: None
  type: NodePort
status:
  loadBalancer: {}
```

NOTE

The NodePort configuration is important. The NodePort is the port in which the client will access the brokers and the type is **NodePort**.

8.8.2. Connecting the clients

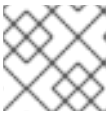
Create consumers that are round-robinned around the brokers in the cluster using the AMQ broker CLI.

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
```

8.9. MONITORING AMQ BROKER

This tutorial demonstrates how to monitor AMQ Broker.

Prerequisites

- This tutorial builds upon Preparing a broker.

- Completion of the Deploying a basic broker tutorial 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 9. DEPLOYING AMQ BROKER ON OPENSHIFT CONTAINER PLATFORM USING AN OPERATOR

IMPORTANT

The AMQ Broker Operator is a Technology Preview feature for AMQ Broker 7.4, including all micro releases (7.4.1, 7.4.2, and so on). Technology Preview features are not supported with Red Hat production service-level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them for production. For more information about technology preview at Red Hat, see Red Hat Technology Preview Features Support Scope.

9.1. OVERVIEW OF THE AMQ BROKER OPERATOR

Kubernetes - and, by extension, OpenShift Container Platform - includes features such as secret handling, load balancing, service discovery, and autoscaling that enable you to build complex distributed systems. Operators are programs that enable you to package, deploy, and manage Kubernetes 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

Operators use Kubernetes extension mechanisms called *Custom Resource Definitions* and corresponding *Custom Resources* to ensure that your custom objects look and act just like native, built-in Kubernetes objects. Custom Resource Definitions and Custom Resources are how you specify the configuration of the OpenShift objects that you plan to deploy.

Previously, you could use only application templates to deploy AMQ Broker on OpenShift Container Platform. While templates are effective for creating an initial deployment, they do not provide a mechanism for updating the deployment. Operators enable you to make changes while your broker instances are running, because they are always listening for changes to your Custom Resources, where you specify your configuration. When you make changes to a Custom Resource, the Operator reconciles the changes with the existing broker installation in your project, and makes it reflect the changes you have made.

9.2. OVERVIEW OF 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 AMQ Broker CRD is the **amq-broker-crd.yaml** file in the **deploy/crds** directory of the repository that you clone 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, queues, and a scaledown controller. Scaledown is desirable when you have more than one broker in a cluster, to ensure that messages are migrated from any broker that shuts down. When deployed, each CRD is a separate controller, running independently within the Operator.

9.2.1. Broker Custom Resource Definition overview

The broker CRD, **amq-broker-crd.yaml**, enables you to configure a broker deployment in a given project. The following sub-sections describe the main sections of the CRD.

spec.names

In the **spec.names** section, you specify names to associate with your CRD. You use these names with Kubernetes commands such as **kubectl** to request information or perform actions on an instance of the CRD deployed via a CR. An example configuration of the **spec.names** section is shown below.

```
spec:
  names:
    kind: ActiveMQArtemis
    listKind: ActiveMQArtemisList
    plural: activemqartemises
    singular: activemqartemis
```

OpenAPIV3Schema

Most of the configuration items in the CRD are in an OpenAPI schema section called **OpenAPIV3Schema**. The **kubectl** command validates the configuration specified in the **OpenAPIV3Schema** section, before the API is used to create a resource. Some specific properties of the schema are described in the sub-sections that follow.

OpenAPIV3Schema.properties.spec.properties.size

The **size** section specifies the maximum number of brokers that can be in an OpenShift project for one deployment of the CRD. This section also specifies the minimum number of brokers, which is **0**, by default. It it is useful, for example, to be able to scale a deployment down to zero brokers to perform some data management, or if a broker deployment is not currently in use.

OpenAPIV3Schema.properties.spec.properties.persistent

The **persistent** section specifies whether brokers deployed using the broker CRD will have persistent storage. Regardless of whether the persistence option is set to **true** or **false** in your CR, the broker deployment is always based on a StatefulSet. By contrast, when you use application templates, a non-persistent broker deployment is based on a different type of Kubernetes resource called a Deployment Config.

OpenAPIV3Schema.properties.spec.properties.aio

The **aio** section is used to specify whether to use asynchronous I/O (AIO). AIO enables you to get better performance from your file store, versus non-blocking I/O (NIO), which is synchronous.

OpenAPIV3Schema.properties.spec.properties.sslConfig

The **sslConfig** section is equivalent to the SSL configuration in an application template. You use this section to specify a secret name, a trust store file name and password, and a key store file name and password. An example configuration of the **sslConfig** section is shown below:

```
sslConfig:
   secretName: <secret name>
   trustStoreFilename: broker.ts
   trustStorePassword: <truststore password>
   keystoreFilename: broker.ks
   keystorePassword: <keystore password>
```

NOTE

If you do not include the **sslConfig** section in your CRD, transport layer security (TLS) is not used to secure your broker connections. Without TLS, you cannot configure secure OpenShift routes to expose the broker to clients outside the OpenShift internal network.

OpenAPIV3Schema.properties.spec.properties.clusterConfig

Use the **clusterConfig** section to specify a user name and password for your broker cluster. The configuration of the **clusterConfig** section in an accompanying CR looks as follows:

```
clusterConfig:
   clusterUserName: <cluster user>
   clusterPassword: <cluster password>
```

If **clusterConfig** is not included in your CRD, you cannot create a broker cluster.

NOTE

There is no option to use a randomly-generated user name or password for your broker cluster. In addition, the values you specify are not stored in a secret.

OpenAPIV3Schema.properties.spec.properties.commonConfig

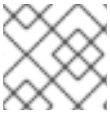
Use the **commonConfig** section to specify a user name and password for the broker management console.

OpenAPIV3Schema.properties.spec.required

The **required** section enforces the set of configuration elements that must be included in a corresponding CR.

By default, the **required** section is configured as follows:

```
required:
 - image
 - size
 - commonConfig
```



NOTE

Any configuration element not specified in the **required** section is optional in your CR.

9.2.2. Sample broker Custom Resources

The AMQ Broker Operator GitHub repository that you clone during installation includes sample CR files in the **deploy/crs** directory. These sample CR files enable you to:

- Deploy a minimal broker without SSL or clustering.
- Define addresses and queues.
- Deploy a broker cluster with a scaledown controller.

The broker Operator repository also includes example deployments in the **deploy/examples** directory. These examples show you how to combine CRs for various types of deployments, as listed below.

basic-deployment.yaml

Basic broker deployment.

persistence-deployment.yaml

Broker deployment with persistent storage.

cluster-deployment.yaml

Deployment of clustered brokers.

persistence-cluster-deployment.yaml

Deployment of clustered brokers with persistent storage.

ssl-deployment.yaml

Broker deployment with SSL security.

ssl-persistence-deployment.yaml

Broker deployment with SSL security and persistent storage.

address-queue-create.yaml

Address and queue creation.

aio-journal.yaml

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

The procedures in the following sections show you how to use an Operator, CRD, and some CRs to deploy some container-based instances of AMQ Broker 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 in a separate StatefulSet containing a Pod in the project. You will use a dedicated CR to define addresses and queues in your broker deployments.

9.3. INSTALLING THE BROKER OPERATOR

The procedures in this section show you how to install and deploy the AMQ Broker Operator on OpenShift Container Platform. In subsequent procedures, you use this Operator to deploy some broker instances.

NOTE

Installing an Operator requires administrator-level privileges for your OpenShift cluster. When the Operator is installed, a regular user can deploy broker instances via the provided Custom Resource.

9.3.1. Getting the Operator code

This procedure shows you how to access and prepare the code you need to install the AMQ Broker Operator.

Procedure

1. Clone the broker Operator repository and check out the latest tag.

```
$ git clone https://github.com/rh-messaging/activemq-artemis-operator
$ git checkout 0.6.3
```

2. Log in to OpenShift Container Platform as a cluster administrator.

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

NOTE

Installing the Operator requires cluster administration privileges. During installation, the Operator needs to be able to communicate with Kubernetes. This requires higher-level privileges than installing AMQ Broker on OpenShift using application templates.

3. Create or log in to the project in which you want to install the Operator.

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

or

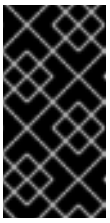
```
$ oc project <project_name>
```

4. Specify a service account to use with the Operator.

   a. In the **deploy** directory of the Operator repository that you cloned, open the **service_account.yaml** file.

   b. Set the **kind** element to **ServiceAccount**.

   c. In the **metadata** section, assign a name to the service account. The default name is **amq-broker-operator**.



IMPORTANT

If you plan to include a scaledown controller in your deployment, you must assign the name **activemq-artemis-operator** to the service account. For more information about the scaledown controller, see Migrating messages upon scaledown.

5. 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. Set the **kind** element to **Role**.

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

6. 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.
Open the **role_binding.yaml** file. Add lines that look like the following:

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

9.3.2. Deploying the Operator

The procedure in this section shows you how to deploy the AMQ Broker Operator in your OpenShift project.

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. 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 and ensure that they are available. For more information about persistent storage, see Understanding persistent storage .

Procedure

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

2. Assign storage to your Operator installation by claiming a persistent volume. In the OpenShift Container Platform web console, click **Storage → Persistent Volume Claims**.

NOTE

You need to execute this step if you do not have container-native storage deployed in your OpenShift cluster. If you do not have container-native storage, you need to manually provision persistent volumes and ensure that they are available. For example, if you want to create a cluster of two broker instances with persistent storage, you need to have two persistent volumes available. By default, each broker instance requires storage of 2GB. If you plan to deploy brokers with non-persistent storage (that is, by setting **persistent=false** in your CR), the broker uses ephemeral storage instead. Ephemeral storage means that that every time you restart the broker Pod, any existing data is lost.

3. Prepare the project to receive the operator.

   a. Create the service account.

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

   b. Create the role.

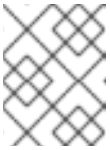
   ```
   $ oc create -f deploy/role.yaml
   ```

   c. Create the role binding.

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

   d. Deploy the broker CRD to the OpenShift cluster.

   ```
   $ oc create -f deploy/crds/broker_v1alpha1_activemqartemis_crd.yaml
   ```

   NOTE

   You must install the CRDs before deploying and starting the Operator. If not, the Operator logs will show messages instructing you to do so.

   e. Deploy the address CRD.

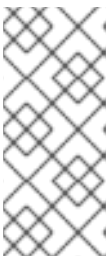
   ```
   $ oc create -f deploy/crds/broker_v1alpha1_activemqartemisaddress_crd.yaml
   ```

   f. Deploy the scaledown CRD.

   ```
   $ oc create -f deploy/crds/broker_v1alpha1_activemqartemisscaledown_crd.yaml
   ```

4. 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, 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 → Namespaces**. Specify the pull secret associated with the account that you use for authentication in the Red Hat Container Registry.

5. In the **deploy** directory of the Operator repository that you cloned, open the **operator.yaml** file. Update **spec.containers.image** with the full path to the Operator image in the Red Hat Container Registry.

```
spec:
   template:
      spec:
         containers:
            image: registry.redhat.io/amq7-tech-preview/amq-broker-operator:0.6
```

6. 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 includes line like 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, scaledown, and addressing are running, and that these controllers have started some workers.

9.4. DEPLOYING A BASIC BROKER

The following procedures show you how to deploy a basic broker instance in your OpenShift project when you have installed the AMQ Broker Operator.

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. Before you can follow the procedure in this section, you must first complete the steps described in Red Hat Container Registry Authentication .
- The Broker Operator is already installed. See Installing the Broker Operator.

Procedure

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

1. In the **deploy/crs** directory of the Operator repository that you cloned, open the **broker_v1alpha1_activemqartemis_cr.yaml** file. This file is an instance of a basic broker Custom Resource. By default, the contents of the file look as follows:

```
apiVersion: broker.amq.io/v1alpha1
kind: ActiveMQArtemis
metadata:
  name: example-activemqartemis
spec:
  # Add fields here
  size: 1
  image: registry.redhat.io/amq7/amq-broker:7.4
```

   The **size** value specifies the number of brokers to deploy. For a clustered deployment, you should set this value to **2** or greater. However, for the basic broker instance shown in the example, keep the default value of **1**.

   The **image** value specifies the container image to use to launch the broker.

2. Deploy a basic broker, based on the **broker_v1alpha1_activemqartemis_cr** CR.

```
$ oc create -f deploy/crs/broker_v1alpha1_activemqartemis_cr.yaml
```

   In the OpenShift Container Platform web console, click **Workloads → Stateful Sets**. You see a new Stateful Set called **example-activemqartemis-ss**.

   Expand the **example-activemqartemis-ss Stateful Set** section. You see that there is one Pod, corresponding to the single broker that you defined in the Custom Resource.

   On the **Events** tab of the running Pod, you see that the broker has started.

9.4.1. Networking services in your running broker

On the **Networking** pane of the OpenShift Container Platform web console, there are two running services; **amq-broker-amq-headless** and **ping**.

The **headless** service provides access to all protocol ports that the broker is listening on, as shown in the following list:

1883
: MQTT

5672
: AMQP

8161
: Web Console / Jolokia

61613
: STOMP

61616
: All protocols. Port 61616 is a multiplexed port that supports all protocols; AMQP, Core, HornetQ, MQTT, OpenWire and STOMP. To produce or consume messages to or from the broker outside OpenShift, you can use a URI formatted like **protocol://:<any.ocp.node.ip>:<muxProtocolPortNumber>**.

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

9.4.2. Accessing the management console for your running broker

The broker hosts its own management console at port 8161. The **example-activemqartemis-service-console-jolokia** service provides external access to the console. The port number chosen by OpenShift for the service is in the range 30000-32767.

To access the broker web console in the OpenShift Container Platform web console:

1. In the OpenShift Container Platform application console, click **Networking → Routes**. On the **Routes** pane, you see a route corresponding to **example-activemqartemis-service-console-jolokia**.
2. To access the broker web console, use the URL displayed under **Hostname**.

To access the broker web console outside the OpenShift Container Platform web console:

1. In the OpenShift Container Platform web console, click **Networking → Services**.
2. Click the **example-activemqartemis-console-jolokia-service** service to see the port number assigned to the service. You need this port number to access the console.
3. In your web browser, use a URL formatted like **http://ocp.node.ip:ConsolePortNumber** to access the running broker's management console.

9.4.3. Limitations of applying Custom Resource changes to running brokers

The following are some things to note when applying Custom Resource changes to running brokers:

- You cannot dynamically update the **image**, **persistent**, or **aio** attributes in your CR. To change these attributes, 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.
- If the **image** attribute in your CR uses a floating tag such as **7.4**, then your deployment automatically pulls new image versions as they become available in the Red Hat Container Registry, provided that the **imagePullPolicy** attribute in your deployment configuration is set to **always**. For example, if your deployment currently uses a broker image version, **7.4-4**, and a newer broker image version, **7.4-5**, becomes available, then your deployment automatically pulls and uses the new image version. To use the new image, each broker in the deployment scales down and then back up. If you have multiple brokers in your deployment, only one broker scales down at a time.
- To change the user name and password for an active broker cluster, scale the cluster down to zero brokers. Then, edit your CR, specifying a new user name and password. Redeploy the CR, also specifying a deployment size.
- All other Custom Resource changes – apart from adding brokers to your deployment – cause existing brokers to scale down and then back up. If you have multiple brokers in your deployment, only one broker scales down at a time.

9.5. DEPLOYING CLUSTERED BROKERS

If there are two or more broker Pods running in your project, then you can configure the Pods to 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 edit your Custom Resource (CR) to increase the number of brokers and deploy them in a cluster.

Prerequisites

- A basic broker is already deployed. See Deploying a basic broker.

Procedure

1. In the **deploy/crs** directory of the Operator repository that you cloned, open the **broker_v1alpha1_activemqartemis_cr.yaml** Custom Resource file.
2. To add another broker to your deployment, change the value of **spec.size** to **2**.
3. At the command line, apply the change:

```
$ oc apply -f deploy/crs/broker_v1alpha1_activemqartemis_cr.yaml
```

   In the OpenShift Container Platform web console, a second Pod starts in your project, for the additional broker that you specified in your CR. The two brokers running in your project are not yet clustered.

4. To add a clustered configuration, reopen the the **broker_v1alpha1_activemqartemis_cr.yaml** file. In the **spec** section of the file, add lines that look like the following:

```
clusterConfig:
  clusterUserName: <cluster user>
  clusterPassword: <cluster password>
```

   At the command line, apply the change:

```
$ oc apply -f deploy/crs/broker_v1alpha1_activemqartemis_cr.yaml
```

   In the OpenShift Container Platform web console, you see that one of the two Pods starts to shut down, while the other continues to run.

5. When the first broker Pod restarts, 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
```

9.6. DEPLOYING A SECURE BROKER CLUSTER

When you have established a broker cluster, you can secure the cluster by configuring SSL/TLS authentication on the broker connections.

This procedure shows you how to configure SSL/TLS security for the broker connections in your cluster.

Prerequisites

- A broker cluster is already deployed. See Deploying clustered brokers.

Procedure

1. AMQ Broker requires a broker keystore, a client keystore, and a client truststore that includes the broker keystore. This procedure uses Java Keytool, a package included with the Java Development Kit, to generate 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. Add the secret to the AMQ service account created earlier.

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

2. In the **deploy/crs** directory of the Operator repository that you cloned, open the **broker_v1alpha1_activemqartemis_cr.yaml** Custom Resource file.
   In the **spec** section of the Custom Resource file, add lines that look like the following:

   ```
   sslConfig:
     secretName: <secret name>
     trustStoreFilename: broker.ts
     trustStorePassword: <truststore password>
     keystoreFilename: broker.ks
     keystorePassword: <keystore password>
   ```

3. Apply the changes.

   ```
   $ oc apply -f deploy/crs/broker_v1alpha1_activemqartemis_cr.yaml
   ```

NOTE

If you apply the preceding changes to a running broker cluster, each broker in the cluster has to scale down and restart, in turn.

4. In the OpenShift Container Platform application, open the **broker** pod with the

4. In the OpenShift Container Platform application console, open the **Logs** tab of either broker Pod. The logs show that the brokers are now listening on the secure SSL-enabled ports listed below, in addition to the regular, non-SSL ports.

8883
: MQTT

5671
: AMQP

61612
: STOMP

61617
: All protocols

9.7. CREATING QUEUES IN A BROKER CLUSTER

The following procedure shows you how to use a Custom Resource Definition (CRD) and example Custom Resource (CR) to add and remove a queue from a broker cluster deployed using an Operator.

Prerequisites

- You have already deployed a broker cluster. See Deploying clustered brokers.

Procedure

1. Deploy the addressing CRD.

```
$ oc create -f deploy/crds/broker_v1alpha1_activemqartemisaddress_crd.yaml
```

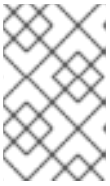
2. An example CR file, **broker_v1alpha1_activemqartemisaddress_cr.yaml**, was included in the Operator repository that you cloned. The example Custom Resource includes the following:

```
spec:
  # Add fields here
  statefulsetName: example-activemqartemis-ss
  addressName: myAddress0
  queueName: myQueue0
  routingType: anycast
```

With your broker cluster already already deployed and running via the Operator, use the example Custom Resource to create an address on every running broker in your cluster.

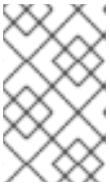
```
$ oc create -f deploy/crs/broker_v1alpha1_activemqartemisaddress_cr.yaml
```

Deploying the example CR creates an address **myAddress0** with a queue named **myQueue0** that has an **anycast** routing type. This address is created on every running broker.



NOTE

To create multiple addresses and/or queues in your broker cluster, you need to create separate CR files and deploy them individually, specifying new address and/or queue names in each case.



NOTE

If you add brokers to your cluster after deploying the addressing CR, the new brokers will not have the address you previously created. In this case, you need to delete the addresses and redeploy the addressing CR.

3. To delete queues created from the example CR, use the following command:

```
$ oc delete -f deploy/crs/broker_v1alpha1_activemqartemisaddress_cr.yaml
```

9.8. MIGRATING MESSAGES UPON SCALEDOWN

To migrate messages upon broker scaledown, deploy a scaledown controller with the AMQ Broker Operator. The Custom Resource Definition for the scaldedown controller is the **broker_v1alpha1_activemqartemisscaledown_crd.yaml** file in the **deploy/crds** directory of the Operator GitHub repository that you cloned.

When a broker Pod scales down, the scaledown controller detects this event and starts a drainer Pod. The drainer Pod connects to one of the live broker Pods in the cluster and migrates messages over to the live broker Pod. After migration is complete, the scaledown controller shuts down.

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

Prerequisites

- You have previously deployed the broker Operator. Specifically, you have linked the pull secret associated with your authenticated Red Hat Container Registry account with the **default**, **deployer**, and **builder** service accounts for your OpenShift project. You have also edited **operator.yaml** to reference the broker container image in the Red Hat Container Registry. For more information, see Deploying the Operator.

Procedure

1. Set up a service account, a role, and a role binding.

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

IMPORTANT

When including a scaledown controller in your deployment, you must assign the name **activemq-artemis-operator** to the service account. For more information about specifying a service account name before creating the service account, see Installing the Broker Operator.

2. Deploy CRDs for the broker and the scaledown controller.

```
$ oc create -f deploy/crds/broker_v1alpha1_activemqartemis_crd.yaml
$ oc create -f deploy/crds/broker_v1alpha1_activemqartemisscaledown_crd.yaml
```

3. Deploy the broker Operator.

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

4. Deploy clustered brokers.

```
$ oc create -f deploy/examples/amq-ssl-persistence-cluster-deployment.yaml
```

5. Once the broker Pod is running, scale it up to two replicas.

```
$ oc scale statefulset example-activemqartemis-ss --replicas 2
```

6. Verify which Pods are running.

```
$ oc get pods
```

You see output that looks like the following, which shows that there are three Pods running.

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

7. Log into each Pod and send some messages to each broker.

   a. Supposing that Pod **example-activemqartemis-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 **example-activemqartemis-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.

8. Scale the cluster down from two brokers to one.

```
$ oc scale statefulset example-activemqartemis-ss --replicas 1
```

You see that the Pod **example-activemqartemis-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 **example-activemqartemis-ss-1** to the other broker Pod in the cluster, **example-activemqartemis-ss-0**.

When the drainer Pod is shut down, check the message count on the **TEST** queue of broker Pod **example-activemqartemis-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.

9.9. MANAGING THE BROKER OPERATOR USING THE OPERATOR LIFECYCLE MANAGER

IMPORTANT

Deploying AMQ Broker on OpenShift Container Platform using an Operator is currently a Technology Preview feature. Technology Preview features are not supported with Red Hat production service-level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them for production. For more information about technology preview at Red Hat, see Red Hat Technology Preview Features Support Scope.

9.9.1. Overview of the Operator Lifecycle Manager

In OpenShift Container Platform 4.0 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 4.0, 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 install the AMQ Broker Operator in OperatorHub, you can use the graphical interface to create various broker deployments, such as a standalone broker, a broker cluster, and a cluster that includes a scaledown controller.

9.9.2. Installing the AMQ Broker Operator in OperatorHub

Procedure

This procedure shows you how to install the AMQ Broker Operator in OperatorHub.

1. Clone the broker Operator repository and check out the latest tag.

```
$ git clone https://github.com/rh-messaging/activemq-artemis-operator
$ git checkout 0.6.3
```

2. Log in to OpenShift Container Platform as a cluster administrator.

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

3. Deploy the AMQ Broker Operator source bundle from the **deploy** directory of the Operator repository that you cloned.

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

After a few minutes, the AMQ Broker Operator - named **ActiveMQArtemis** - is available in the **OperatorHub** section of the OpenShift Container Platform web console. You can then use the OperatorHub graphical interface to create various types of broker deployments.

CHAPTER 10. REFERENCE

10.1. APPLICATION TEMPLATE PARAMETERS

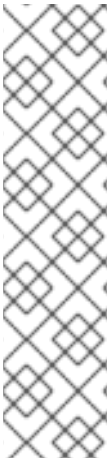
Configuration of the AMQ Broker on OpenShift Container Platform image is performed by specifying values of application template parameters. You can configure the following parameters:

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. If no value is specified, a random password is generated.
AMQ_CLUSTER_USER	Specifies the cluster user to use for clustering. If no value is specified, a random user name is generated.
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.
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.

Parameter	Description
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.
AMQ_PASSWORD	Specifies the password used for authentication to the broker. If no value is specified, a random password is generated.
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.
AMQ_USER	Specifies the user name used for authentication to the broker. If no value is specified, a random user name is generated.
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.

Parameter	Description
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.
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

10.2. SECURITY

Only SSL connections can connect from outside of the OpenShift instance. The non-SSL version of the protocols can only be used inside the OpenShift instance.

For security reasons, using the default keystore and truststore generated by the system is discouraged. Generate your own keystore and truststore and supply them to the image by using the OpenShift secrets mechanism.

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 2020-01-16 16:03:14 UTC