



Red Hat AMQ 7.4

Deploying AMQ Broker on OpenShift Container Platform

For Use with AMQ Broker 7.4

Red Hat AMQ 7.4 Deploying AMQ Broker on OpenShift Container Platform

For Use with AMQ Broker 7.4

Legal Notice

Copyright © 2021 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

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

Table of Contents

CHAPTER 1. INTRODUCTION	4
1.1. VERSION COMPATIBILITY AND SUPPORT	4
1.2. UNSUPPORTED FEATURES	4
CHAPTER 2. DEPLOYING AMQ BROKER ON OPENSIFT CONTAINER PLATFORM USING AN OPERATOR	6
2.1. OVERVIEW OF THE AMQ BROKER OPERATOR	6
2.2. OVERVIEW OF CUSTOM RESOURCE DEFINITIONS	6
2.2.1. Sample broker Custom Resources	7
2.3. INSTALLING THE AMQ BROKER OPERATOR	7
2.3.1. Getting the Operator code	8
2.3.2. Deploying the Operator	10
2.4. DEPLOYING A BASIC BROKER	12
2.5. APPLYING CUSTOM RESOURCE CHANGES TO RUNNING BROKER DEPLOYMENTS	14
2.6. CONFIGURING OPERATOR-BASED BROKER DEPLOYMENTS FOR CLIENT CONNECTIONS	14
2.6.1. Configuring acceptors	14
2.6.1.1. Additional acceptor configuration notes	15
2.6.2. Connecting to the broker from internal and external clients	16
2.6.3. Generating credentials for SSL connections	17
2.6.4. Networking services in your broker deployments	18
2.6.5. Connecting a broker to the AMQ Broker management console	18
2.6.5.1. Accessing the broker management console	18
2.6.5.2. Accessing management console login credentials	19
2.7. BROKER DEPLOYMENT EXAMPLES	19
2.7.1. Deploying clustered brokers	19
2.7.2. Creating queues in a broker cluster	20
2.8. MIGRATING MESSAGES UPON SCALEDOWN	21
2.9. MANAGING THE BROKER OPERATOR USING THE OPERATOR LIFECYCLE MANAGER	23
2.9.1. Overview of the Operator Lifecycle Manager	23
2.9.2. Installing the AMQ Broker Operator in OperatorHub	23
CHAPTER 3. DEPLOYING AMQ BROKER ON OPENSIFT CONTAINER PLATFORM USING APPLICATION TEMPLATES	25
3.1. INSTALLING THE AMQ BROKER IMAGE STREAMS AND APPLICATION TEMPLATES	25
3.2. PREPARING A TEMPLATES-BASED BROKER DEPLOYMENT	26
3.3. DEPLOYING A BASIC BROKER	27
3.3.1. Create the broker application	28
3.3.2. Deploy and start the broker application	29
CHAPTER 4. UPGRADING AMQ BROKER ON OPENSIFT CONTAINER PLATFORM	32
4.1. UPGRADING OPERATOR-BASED BROKER DEPLOYMENTS	32
4.1.1. Upgrading the Operator	32
4.2. UPGRADING A TEMPLATES-BASED BROKER DEPLOYMENT	33
4.2.1. Upgrading non-persistent broker deployments	33
4.2.2. Upgrading persistent broker deployments	34
CHAPTER 5. CONNECTING EXTERNAL CLIENTS TO TEMPLATES-BASED BROKER DEPLOYMENTS ...	37
5.1. CONFIGURING SSL	37
5.2. GENERATING THE AMQ BROKER SECRET	37
5.3. CREATING AN SSL ROUTE	38
CHAPTER 6. CUSTOMIZING AMQ BROKER CONFIGURATION FILES FOR DEPLOYMENT	40
CHAPTER 7. HIGH AVAILABILITY	41

7.1. HIGH AVAILABILITY OVERVIEW	41
7.2. MESSAGE MIGRATION	41
7.2.1. Message migration overview	41
7.2.1.1. How does message migration work?	42
CHAPTER 8. TUTORIALS	44
8.1. DEPLOYING A BASIC BROKER WITH SSL	44
8.1.1. Deploying the image and template	44
8.1.2. Deploying the application	45
8.1.3. Creating a Route	45
8.2. DEPLOYING A BASIC BROKER WITH PERSISTENCE AND SSL	46
8.2.1. Deploy the image and template	46
8.2.2. Deploy the application	47
8.2.3. Creating a Route	49
8.3. DEPLOYING A SET OF CLUSTERED BROKERS	49
8.3.1. Distributing messages	49
8.3.2. Deploy the image and template	50
8.3.3. Deploying the application	51
8.3.4. Creating a Route for the AMQ Broker management console	52
8.4. DEPLOYING A SET OF CLUSTERED SSL BROKERS	53
8.4.1. Distributing messages	54
8.4.2. Deploying the image and template	54
8.4.3. Deploying the application	55
8.4.4. Creating Routes for the AMQ Broker management console	57
8.5. DEPLOYING A BROKER WITH CUSTOM CONFIGURATION	58
8.5.1. Deploy the image and template	59
8.5.2. Deploy the application	59
8.6. BASIC SSL CLIENT EXAMPLE	59
8.6.1. Configuring the client	60
8.7. EXTERNAL CLIENTS USING SUB-DOMAINS EXAMPLE	60
8.7.1. Exposing the brokers	60
8.7.2. Connecting the clients	61
8.8. EXTERNAL CLIENTS USING PORT BINDING EXAMPLE	62
8.8.1. Exposing the brokers	62
8.8.2. Connecting the clients	63
8.9. MONITORING AMQ BROKER	63
CHAPTER 9. REFERENCE	65
9.1. CUSTOM RESOURCE DEFINITION CONFIGURATION REFERENCE	65
9.1.1. Broker CRD configuration reference	65
9.1.2. Addressing CRD configuration reference	75
9.2. APPLICATION TEMPLATE PARAMETERS	75
9.3. LOGGING	77

CHAPTER 1. INTRODUCTION

Red Hat AMQ Broker 7.4 is available as a containerized image that is provided for use with OpenShift Container Platform (OCP) 3.11 and later.



NOTE

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, 7.4.3, 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.

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. DEPLOYING AMQ BROKER ON OPENSIFT CONTAINER PLATFORM USING AN OPERATOR

2.1. 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.

2.2. 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 broker CRD is the **broker_v2alpha1_activemqartemis** file in the **deploy/crds** directory of the archive that you download and extract when installing the Operator. This CRD enables you to configure a broker deployment in a given OpenShift project. The other CRDs in the **deploy/crds** directory are for configuring addresses and for the Operator to use when instantiating a scaledown controller .

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

For a complete configuration reference for each CRD see:

- [Broker CRD configuration reference](#)

- [Addressing CRD configuration reference](#)

2.2.1. Sample broker Custom Resources

The AMQ Broker Operator archive that you download and extract 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.

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

artemis-basic-deployment.yaml

Basic broker deployment.

artemis-persistence-deployment.yaml

Broker deployment with persistent storage.

artemis-cluster-deployment.yaml

Deployment of clustered brokers.

artemis-persistence-cluster-deployment.yaml

Deployment of clustered brokers with persistent storage.

artemis-ssl-deployment.yaml

Broker deployment with SSL security.

artemis-ssl-persistence-deployment.yaml

Broker deployment with SSL security and persistent storage.

artemis-aio-journal.yaml

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

address-queue-create.yaml

Address and queue creation.

The procedures in the following sections show you how to use an Operator, CRD, and some CRs to create some container-based broker deployments on OpenShift Container Platform. When you have successfully completed the procedures, you will have the Operator running in an individual Pod. Each broker instance that you create will run in a separate StatefulSet containing a Pod in the project. You will use a dedicated CR to define addresses in your broker deployments.



NOTE

You cannot create more than one broker deployment in a given OpenShift project by deploying multiple broker CR instances. However, when you have created a broker deployment in a project, you **can** deploy multiple CR instances for addresses.

2.3. INSTALLING THE AMQ BROKER OPERATOR

The procedures in this section show how to:

- Install the latest version of the Operator for AMQ Broker 7.4
- Specify the latest broker container image for AMQ Broker 7.4 for your broker deployments

Described below are some important things to note about installing the Operator.

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, 7.4.3, 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.
- To remain in a supported configuration, you must upgrade to the latest micro release in the LTS release stream. This means that you must use the latest broker container image from the LTS stream in your broker deployments. This section shows how to specify the latest broker container image for a broker deployment based on the AMQ Broker Operator.
- Version 0.6 of the AMQ Broker Operator for AMQ Broker 7.4 was a Technology Preview feature only. If you have version 0.6 of the Operator installed in an OpenShift project, it is recommended that you update the Operator to the latest Long Term Support (LTS) version. The LTS version of the Operator includes fixes for bugs and security advisories. Red Hat supports LTS versions of the Operator for production use.
- The Custom Resource Definitions (CRDs) used by version 0.6 of the AMQ Broker Operator are not compatible with the Long Term Support (LTS) version. To install the latest LTS version of the Operator, you must delete any CRDs previously deployed in your OpenShift cluster before deploying the latest CRDs. You must also remove the existing Operator and broker deployment from any project in which you want to install the LTS version of the Operator. These steps are shown in the procedures in this section.
- Deploying the latest CRDs requires cluster administrator privileges for your OpenShift cluster. Non-administrator users can create broker instances in OpenShift projects by deploying Custom Resources (CRs).
- When you update an OpenShift cluster with the latest CRDs, this update affects **all projects** in the cluster. Any broker Pods previously deployed from version 0.6 no longer work. You should update each affected project in the OpenShift cluster to use the LTS version of the Operator. Then, you can deploy Custom Resources (CRs) included with the LTS version of the Operator to recreate your previous broker deployments.

2.3.1. Getting the Operator code

This procedure shows you how to access and prepare the code you need to install the latest LTS version of the Operator for AMQ Broker 7.4.

Procedure

1. In your web browser, navigate to the [AMQ Broker Software Downloads page](#).
2. In the **Version** drop-down box, select **7.4.6**.
3. Next to **AMQ Broker 7.4.6 Operator Installation and Example Files** click **Download**.

Download of the **amq-broker-operator-7.4.6-ocp-install-examples.zip** compressed archive automatically begins.

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

```
sudo mv amq-broker-operator-7.4.6-ocp-install-examples.zip /broker/operator
```

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

```
cd /broker/operator
sudo unzip amq-broker-operator-7.4.6-ocp-install-examples.zip
```

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

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

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

- Create a new project:

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

- Or, switch to an existing project:

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

- Specify a service account to use with the Operator.

- In the **deploy** directory of the Operator archive that you extracted, open the **service_account.yaml** file.

- Ensure that the **kind** element is set to **ServiceAccount**.

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

- Create the service account in your project.

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

- Specify a role name for the Operator.

- Open the **role.yaml** file. This file specifies the resources that the Operator can use and modify.

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

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

- Create the role in your project.

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

-
10. Specify a role binding for the Operator. The role binding binds the previously-created service account to the Operator role, based on the names you specified.
 - a. Open the **role_binding.yaml** file. Ensure that the **name** values for **ServiceAccount** and **Role** match those specified in the **service_account.yaml** and **role.yaml** files. For example:

```

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

```

- b. Create the role binding in your project.

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

2.3.2. Deploying the Operator

The procedure in this section shows how to deploy the latest LTS version of the Operator for AMQ Broker 7.4 in your OpenShift project.

Prerequisites

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

For more information about provisioning persistent storage in OpenShift Container Platform, see [Understanding persistent storage](#) in the OpenShift Container Platform documentation.

Procedure

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

- If you deployed an earlier version of the AMQ Broker Operator in the project, remove the main broker Custom Resource (CR) from the project. Deleting the main CR removes the existing broker deployment in the project. For example:

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

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

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

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

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

- Deploy the CRDs that are included in the **deploy/crds** directory of the Operator archive that you downloaded and extracted. You must install the latest CRDs in your OpenShift cluster before deploying and starting the Operator.

- Deploy the main broker CRD.

```
$ oc create -f deploy/crds/broker_v2alpha1_activemqartemis_crd.yaml
```

- Deploy the addressing CRD.

```
$ oc create -f deploy/crds/broker_v2alpha1_activemqartemisaddress_crd.yaml
```

- Deploy the scaledown controller CRD.

```
$ oc create -f deploy/crds/broker_v2alpha1_activemqartemisscaledown_crd.yaml
```

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

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



NOTE

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

- In the **deploy** directory of the Operator archive that you downloaded and extracted, open the **operator.yaml** file. Update **spec.containers.image** with the full path to the image for the latest LTS version of the Operator for AMQ Broker 7.4 in the Red Hat Container Registry.

```
spec:
  template:
    spec:
      containers:
        image: registry.redhat.io/amq7/amq-broker-lts-rhel7-operator:0.9
```

8. Deploy the Operator.

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

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

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

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

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

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



NOTE

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

2.4. 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.



NOTE

You cannot create more than one broker deployment in a given OpenShift project by deploying multiple broker CR instances. However, when you have created a broker deployment in a project, you **can** deploy multiple CR instances for addresses.

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 AMQ Broker Operator is already installed. See [Installing the AMQ 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 archive that you downloaded and extracted, open the **broker_v2alpha1_activemqartemis_cr.yaml** file. This file is an instance of a basic broker CR.

The default contents of the file look as follows:

```
apiVersion: broker.amq.io/v2alpha1
kind: ActiveMQArtemis
metadata:
  name: ex-aao
  application: ex-aao-app
...
spec:
  deploymentPlan:
    size: 2
    image: registry.redhat.io/amq7/amq-broker-lts-rhel7:7.4
```

size

Specifies the number of brokers to deploy. For a clustered deployment, this value is **2** or greater. However, for a basic broker instance, change the value to **1**.

image

Specifies the container image to use to launch the broker. By default, the CR uses a floating tag of **7.4**. The floating tag means that the CR uses the latest broker container image that is available for the LTS image stream in the Red Hat Container Registry.

2. Deploy a basic broker, based on the **broker_v2alpha1_activemqartemis** CR.

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

In the OpenShift Container Platform web console, click **Workloads** → **Stateful Sets** (OpenShift Container Platform 4.1) or **Applications** → **Stateful Sets** (OpenShift Container Platform 3.11). You see a new Stateful Set called **ex-aao-ss**.

Expand the **ex-aao-ss Stateful Set** section. You see that there is at least one Pod, corresponding to the value that you set for the **size** attribute in the CR.

On the **Events** tab for each broker Pod, you see that the broker has started.



NOTE

To delete a broker deployment, delete the Custom Resource instance that you created for the deployment. It is not sufficient to delete only the Stateful Set.

Additional resources

- To learn how to connect a running broker to the AMQ Broker management console, see [Connecting a broker to the AMQ Broker management console](#) .

2.5. APPLYING CUSTOM RESOURCE CHANGES TO RUNNING BROKER DEPLOYMENTS

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

- You cannot dynamically update the **persistenceEnabled** attribute in your CR. To change this attribute, scale your cluster down to zero brokers. Delete the existing CR. Then, recreate and redeploy the CR with your changes, also specifying a deployment size.
- 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 or Stateful Set is set to **Always**. For example, if your deployment currently uses broker image version **7.4-6**, and a newer broker image version, **7.4-7**, becomes available, then your deployment automatically pulls and uses the new image version. To use the new image, each broker in the deployment restarts. If you have multiple brokers in your deployment, each broker restarts in sequence.
- The value of the **deploymentPlan.size** attribute in your CR overrides any change you make to size of your broker deployment via the **oc scale** command. For example, suppose you use **oc scale** to change the size of a deployment from three brokers to two, but the value of **deploymentPlan.size** in your CR is still **3**. In this case, OpenShift initially scales the deployment down to two brokers. However, when the scaledown operation is complete, the Operator restores the deployment to three brokers, as specified in the CR.
- During an active scaling event, any further changes that you apply are queued by the Operator and executed only when scaling is complete. For example, suppose you scale the size of your deployment down from four brokers to one. Then, while scaledown is taking place, you also change the values of the broker administrator user name and password. In this case, the Operator queues the user name and password changes until the deployment is running with one active broker.
- All Custom Resource changes – apart from changing the size of your deployment, or changing the value of the **expose** attribute for acceptors, connectors, or the console – cause existing brokers to scale down and then back up. If you have multiple brokers in your deployment, only one broker scales down at a time.

2.6. CONFIGURING OPERATOR-BASED BROKER DEPLOYMENTS FOR CLIENT CONNECTIONS

2.6.1. Configuring acceptors

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

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

Prerequisites

- To configure acceptors, your broker deployment must be based on based on the LTS version of the AMQ Broker Operator. For more information about installing the LTS version of the Operator, see [Installing the AMQ Broker Operator](#).
- The information in this section applies only to broker deployments based on the AMQ Broker Operator. If you used application templates to create your broker deployment, you cannot define individual protocol-specific acceptors. For more information, see [Connecting external clients to templates-based broker deployments](#).

Procedure

1. In the **deploy/crs** directory of the Operator archive that you downloaded and extracted during your initial installation, open the **broker_v2alpha1_activemqartemis_cr.yaml** Custom Resource (CR).
2. In the **acceptors** element, add a named acceptor. Typically, you also specify a minimal set of attributes such as the protocols to be used by the acceptor and the port on the broker Pod to expose for those protocols. An example is shown below.

```
spec:
...
  acceptors:
  - name: amqp_acceptor
    protocols: amqp
    port: 5672
    sslEnabled: false
  ...
```

The preceding example shows configuration of a simple AMQP acceptor. The acceptor exposes port 5672 to AMQP clients.

2.6.1.1. Additional acceptor configuration notes

This section describes some additional things to note about acceptor configurations.

- You can define acceptors either for internal clients (that is, client applications in the same OpenShift cluster as the broker Pod), or for both internal and external clients (that is, applications outside OpenShift). To also expose an acceptor to external clients, set the **expose** parameter of the acceptor configuration to **true**. The default value of this parameter is **false**.
- A single acceptor can accept multiple client connections, up to a maximum limit specified by the **connectionsAllowed** parameter of your acceptor configuration.
- If you do not define any acceptors in your CR, the broker Pods in your deployment use a single acceptor, created by default, on port 61616. This default acceptor has only the Core protocol specified.
- Port 8161 is automatically exposed on the broker Pod for use by the AMQ Broker management console. Within the OpenShift network, this port can be accessed via the *headless* service that runs in your broker deployment. For more information, see [Accessing the broker management console](#).

- You can enable SSL on the acceptor by setting **sslEnabled** to **true**. You can specify additional information such as:
 - The secret name used to store SSL credentials (required).
 - The cipher suites and protocols to use for SSL communication.
 - Whether the acceptor uses two-way SSL, that is, mutual authentication between the broker and the client.

If the acceptor that you define uses SSL, then the SSL credentials used by the acceptor must be stored in a secret. You must create your own secret and specify this secret name in the **sslSecret** parameter of your acceptor configuration. If you do not explicitly specify a secret name in the **sslSecret** parameter, the acceptor assumes a default secret name. The default secret name uses the format **<CustomResourceName>-<AcceptorName>-secret**. For example, **ex-aa0-amqp-secret**.

The SSL credentials required in the secret are **broker.ks**, which must be a base64-encoded keystore, **client.ts**, which must be a base64-encoded truststore, and **keyStorePassword** and **trustStorePassword**, which are passwords specified in raw text. This requirement is the same for any connectors that you configure. For information about generating credentials for SSL connections, see [Generating credentials for SSL connections](#).

Additional resources

- For a complete configuration reference for the main broker Custom Resource Definition (CRD), including configuration of acceptors, see [Custom Resource Definition configuration reference](#).

2.6.2. Connecting to the broker from internal and external clients

- An internal client can connect to the broker Pod by specifying an address in the format **<PodName>:<AcceptorPortNumber>**. OpenShift DNS successfully resolves addresses in this format because the Stateful Sets created by Operator-based broker deployments provide stable Pod names.
- When you expose an acceptor to external clients, a dedicated Service and Route are automatically created. To see the Routes configured on a given broker Pod, select the Pod in the OpenShift Container Platform web console and click the **Routes** tab. An external client can connect to the broker by specifying the full host name of the Route created for the acceptor. You can use a **curl** command to test external access to this full host name. For example:

```
$ curl https://ex-aa0-0-svc-my_project.my_openshift_domain
```

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

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

By contrast, a messaging client that uses TCP must explicitly specify the port number as part of the connection URL. For example:

```
tcp://ex-aa0-0-svc-my_project.my_openshift_domain:443
```

-
- As an alternative to using a Route, an OpenShift administrator can configure a NodePort to connect to a broker Pod from a client outside OpenShift. The NodePort should map to one of the protocol-specific ports specified by the acceptors configured for the broker. By default, NodePorts are in the range 30000 to 32767, which means that a NodePort typically does not match the intended port on the broker Pod. To connect from a client outside OpenShift to the broker via a NodePort, you specify a URI in the format **<Protocol>://<OCPNodeIP>:<NodePortNumber>**.

Additional resources

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

2.6.3. Generating credentials for SSL connections

For SSL connections, AMQ Broker requires a broker keystore, a client keystore, and a client truststore that includes the broker keystore. This procedure shows you how to generate the credentials. The procedure uses Java Keytool, a package included with the Java Development Kit.

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. Use the broker keystore file to create a secret to store the SSL credentials, as shown in the example below.

```
$ oc secrets new ex-aa0-amqp-secret broker.ks client.ts
```

6. Add the secret to the service account that you created when installing the Operator, as shown in the example below.

```
$ oc secrets add sa/amq-broker-operator secret/ex-aa0-amqp-secret
```

2.6.4. Networking services in your broker deployments

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

The headless service provides access to ports 8161 and 61616 on each broker Pod. Port 8161 is used by the broker management console, and port 61616 is used for broker clustering.

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.

2.6.5. Connecting a broker to the AMQ Broker management console

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

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

Prerequisites

- You have deployed a basic broker using the AMQ Broker Operator. For more information, see [Deploying a basic broker](#).

2.6.5.1. Accessing the broker management console

Each broker Pod in your deployment has a service that provides access to the console. The default name of this service uses the format **<Custom Resource name>-wconsj-<broker Pod ordinal>-svc**. For example, for broker Pod **0** of your deployment, the service name is **ex-ao-wconsj-0-svc**. Each Service has a corresponding Route that uses the format **<Custom Resource name>-wconsj-<broker Pod ordinal>-svc-rte**. For example, **ex-ao-wconsj-0-svc-rte**.

This procedure shows you how to access the AMQ Broker management console for a running broker instance.

Procedure

1. In the OpenShift Container Platform web console, click **Networking** → **Routes** (OpenShift Container Platform 4.1) or **Applications** → **Routes** (OpenShift Container Platform 3.11). On the **Routes** pane, you see a Route corresponding to the **wconsj** Service.
2. Under **Hostname**, note the complete URL. You need to specify this URL to access the console.
3. In a web browser, enter the host name URL.
 - a. If your console configuration does not use SSL, specify **http** in the URL. In this case, DNS resolution of the host name directs traffic to port 80 of the OpenShift router.
 - b. If your console configuration uses SSL, specify **https** in the URL. In this case, your browser defaults to port 443 of the OpenShift router. This enables a successful connection to the console if the OpenShift router also uses port 443 for SSL traffic, which the router does by default.

4. To log in to the management console, enter the user name and password specified in the **adminUser** and **adminPassword** parameters of your broker deployment Custom Resource. If there are no values specified for **adminUser** and **adminPassword**, follow the instructions in [Accessing management console login credentials](#) to retrieve the credentials required to log in to the console.

2.6.5.2. Accessing management console login credentials

If you did not specify a value for **adminUser** and **adminPassword** in your broker Custom Resource (CR), the Operator automatically generates the broker user name and password (required to log in to the AMQ Broker management console) and stores these credentials in a secret. The default secret name has a format of **<Custom Resource name>-credentials-secret**, for example, **ex-aa-credentials-secret**.

This procedure shows you how to access the login credentials required to log in to the management console.

Procedure

1. See the complete list of secrets in your OpenShift project.
 - a. From the OpenShift Container Platform web console, click **Workload → Secrets** (OpenShift Container Platform 4.1) or **Resources → Secrets** (OpenShift Container Platform 3.11).
 - b. From the command line:

```
$ oc get secrets
```

2. Open the appropriate secret to reveal the console login credentials.
 - a. From the OpenShift Container Platform web console, click the secret that includes your broker Custom Resource instance in its name. To see the encrypted user name and password values, click the **YAML** tab (OpenShift Container Platform 4.1) or **Actions → Edit YAML** (OpenShift Container Platform 3.11).
 - b. From the command line:

```
$ oc edit secret <my_custom_resource_name-credentials-secret>
```

2.7. BROKER DEPLOYMENT EXAMPLES

2.7.1. Deploying clustered brokers

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

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

Prerequisites

- A basic broker is already deployed. See [Deploying a basic broker](#).

Procedure

1. In the **deploy/crs** directory of the Operator archive that you downloaded and extracted, open the **broker_v2alpha1_activemqartemis_cr.yaml** Custom Resource file.
2. For a minimally-sized clustered deployment, ensure that the value of **deploymentPlan.size** is **2**.
3. At the command line, apply the change:

```
$ oc apply -f deploy/crs/broker_v2alpha1_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. By default, the two brokers running in your project are clustered.

4. 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
```

2.7.2. 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_v2alpha1_activemqartemisaddress_cr.yaml
```

2. An example CR file, **broker_v2alpha1_activemqartemisaddress_cr.yaml**, was included in the Operator archive that you downloaded and extracted. The example Custom Resource includes the following:

```
spec:
  # Add fields here
  spec:
    addressName: myAddress0
    queueName: myQueue0
    routingType: anycast
```

With your broker cluster 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_v2alpha1_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.

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

```
$ oc delete -f deploy/crs/broker_v2alpha1_activemqartemisaddress_cr.yaml
```

2.8. MIGRATING MESSAGES UPON SCALEDOWN

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

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



NOTE

A scaledown controller operates only within a single OpenShift project. The controller cannot migrate messages between brokers in separate projects.



NOTE

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

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

Prerequisites

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

Procedure

1. In the **deploy/crs** directory of the Operator repository that you originally downloaded and extracted, open the main broker CR, **broker_v2alpha1_activemqartemis_cr.yaml**.

In the main broker CR set **messageMigration** and **persistenceEnabled** to **true**.

+ These settings mean that when you later scale down the size of your clustered broker deployment, the Operator automatically starts a scaledown controller and migrate messages to a broker Pod that is still running.

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

```
$ oc get pods
```

You see output that looks like the following.

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

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

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

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

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

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

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

3. Scale the cluster down from two brokers to one.
 - a. Open the main broker CR, **broker_v2alpha1_activemqartemis_cr.yaml**.
 - b. In the CR, set **deploymentPlan.size** to **1**.
 - c. At the command line, apply the change:

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

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

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

2.9. MANAGING THE BROKER OPERATOR USING THE OPERATOR LIFECYCLE MANAGER

2.9.1. 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.

2.9.2. Installing the AMQ Broker Operator in OperatorHub

If you do not see the latest LTS version of the Operator for AMQ Broker 7.4 automatically available in OperatorHub, follow this procedure to manually install the Operator in OperatorHub.

Procedure

1. In your web browser, navigate to the [AMQ Broker Software Downloads](#) page.
2. In the **Version** drop-down box, select, **7.4.6**.
3. Next to **AMQ Broker 7.4.6 Operator Installation and Example Files** click **Download**. Download of the **amq-broker-operator-7.4.6-ocp-install-examples.zip** compressed archive automatically begins.
4. When the download has completed, move the archive to your chosen installation directory. The following example moves the archive to a directory called **/broker/operator**.

```
sudo mv amq-broker-operator-7.4.6-ocp-install-examples.zip /broker/operator
```

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

```
cd /broker/operator
unzip amq-broker-operator-7.4.6-ocp-install-examples.zip
```

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

```
┃ $ oc login -u system:admin
```

7. Deploy the AMQ Broker Operator source bundle from the **deploy** directory of the Operator archive that you downloaded and extracted.

```
┃ $ oc create -f deploy/catalog_resources/courier/amq-broker-operatorsource.yaml
```

After a few minutes, the AMQ Broker Operator is available in the **OperatorHub** section of the OpenShift Container Platform web console.

CHAPTER 3. DEPLOYING AMQ BROKER ON OPENSIFT 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. 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.

- 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/74-7.4.6.GA/amq-broker-7-image-streams.yaml
```

- 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/74-7.4.6.GA/templates/${template}
done
```

3.2. PREPARING A TEMPLATES-BASED BROKER DEPLOYMENT

Prerequisites

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

Procedure

- Use the command prompt to create a new project:

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

- 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 -
```

- 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
```

3.3. DEPLOYING A BASIC BROKER

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



NOTE

This broker does not support SSL and is not accessible to external clients. Only clients running internally on the OpenShift cluster can connect to the broker. For examples of creating broker configurations that support SSL, see [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, 7.4.3, 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.

- To remain in a supported configuration, you must install the latest micro release in the LTS release stream. This means that you must use the latest broker container image from the LTS stream in your broker deployments. This section shows how to specify the latest broker container image for a broker deployment based on application templates.

3.3.1. 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

Environment variable	Display Name	Value	Description
AMQ_PASSWORD	AMQ Password	password	The password the client uses with the username

3.3.2. 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. In the search box, enter **AMQ Broker LTS**. Click **Search**.
- c. In the search results, click **AMQ Broker LTS**. The **amq7/amq-broker-lts-rhel7** repository opens, with the **latest** image tag automatically selected.
- d. Click the **Get this image** tab.
- e. Under **Authentication with registry tokens**, review the on-page instructions in the **Using OpenShift secrets** section. The instructions describe how to specify the broker container 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 container 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'
  ..
  imagePullSecrets:
    - name: {PULL-SECRET-NAME}
```

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

```
$ oc import-image amq7/amq-broker-lts-rhel7:7.4 --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 OPENSIFT 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, 7.4.3, 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.
- To remain in a supported configuration, you must upgrade to the latest micro release in the LTS release stream.
 - To learn about updating the Operator to the latest LTS version, see [Upgrading the Operator](#).
 - To update templates-based deployments to specify the latest broker container image in the LTS stream, see [Upgrading the broker container image for templates-based deployments](#).
- 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 \(Recommended\)](#).
 - [Deploying AMQ Broker on OpenShift Container Platform using Application Templates](#)

4.1. UPGRADING OPERATOR-BASED BROKER DEPLOYMENTS

This section provides information about updating the Operator version used by your OpenShift project.

4.1.1. Upgrading the Operator

Version 0.6 of the AMQ Broker Operator was a Technology Preview feature only. If you have version 0.6 of the Operator installed in an OpenShift project, it is recommended that you update the Operator to the latest Long Term Support (LTS) version. The latest LTS version of the Operator includes fixes for bugs and security advisories. Red Hat supports LTS versions of the Operator for production use.

Described below are some important things to note about upgrading the Technical Preview version of the Operator to the latest LTS version.



IMPORTANT

- The Custom Resource Definitions (CRDs) used by version 0.6 of the Operator are not compatible with the Long Term Support (LTS) version. For this reason, you cannot perform a seamless upgrade of the Operator. To upgrade your Operator from version 0.6, you must delete the CRDs previously deployed in your OpenShift cluster. You must also remove the existing Operator and broker deployment from any project in which you want to install the LTS version of the Operator.
- When you update an OpenShift cluster with the latest CRDs, this update affects **all projects** in the cluster. Any broker Pods previously deployed from version 0.6 no longer work. You should update each affected project in the OpenShift cluster to use the LTS version of the Operator. Then, you can deploy Custom Resources (CRs) included with the LTS version of the Operator to recreate your previous broker deployments.
- When you deploy a new CR to recreate a previous broker deployment, you can specify the latest broker container image in the LTS stream in your CR.

For instructions on installing the LTS version of the Operator and creating new broker deployments, see [Deploying AMQ Broker on OpenShift Container Platform using an Operator](#) .

4.2. UPGRADING A TEMPLATES-BASED BROKER DEPLOYMENT

The following procedures show how to upgrade the broker container image for broker deployments 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 the latest broker container image in the Long Term Support (LTS) image stream.

```
...
spec:
  containers:
    image: 'registry.redhat.io/amq7/amq-broker-lts-rhel7:7.4'
```



NOTE

If you specify a tag format such as **7.4** for the **image** attribute (rather than a tag with a specific version identifier, such as **7.4-6**), this tag format is known as a *floating tag*. When you specify a floating tag, OpenShift Container Platform automatically identifies the most recent available image in the specified repository and uses this image to upgrade your broker deployment.

6. Add the **imagePullSecrets** attribute to specify the image pull secret associated with the account used for authentication in the Red Hat Container Registry.

```
...
spec:
  containers:
    image: 'registry.redhat.io/amq7/amq-broker-lts-rhel7:7.4'
..
imagePullSecrets:
  - name: {PULL-SECRET-NAME}
```

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

- Navigate to the OpenShift Container Platform web console and log in.
- Click the project in which you want to upgrade a persistent broker deployment.
- Select the StatefulSet (SS) corresponding to your broker deployment.
 - In OpenShift Container Platform 4.1, click **Workloads** → **Stateful Sets**.
 - In OpenShift Container Platform 3.11, click **Applications** → **Stateful Sets**.
- 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.
- To prepare your broker deployment for upgrade, scale the deployment down to zero brokers.
 - If the **replicas** attribute is currently set to **1** or greater, set it to **0**.
 - Click **Save**.
- When all broker Pods have shut down, edit the Stateful Set **.yaml** file again. Edit the **image** attribute to specify the latest broker container image in the LTS image stream.

```
...
spec:
  containers:
    image: 'registry.redhat.io/amq7/amq-broker-lts-rhel7:7.4'
```



NOTE

If you specify a tag format such as **7.4** for the **image** attribute (rather than a tag with a specific version identifier, such as **7.4-6**), this tag format is known as a *floating tag*. When you specify a floating tag, OpenShift Container Platform automatically identifies the most recent available image in the specified repository and uses this image to upgrade your broker deployment.

- Add the **imagePullSecrets** attribute to specify the image pull secret associated with the account used for authentication in the Red Hat Container Registry.

```
...
spec:
  containers:
    image: 'registry.redhat.io/amq7/amq-broker-lts-rhel7:7.4'
  ..
  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. CONNECTING EXTERNAL CLIENTS TO TEMPLATES-BASED BROKER DEPLOYMENTS

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.

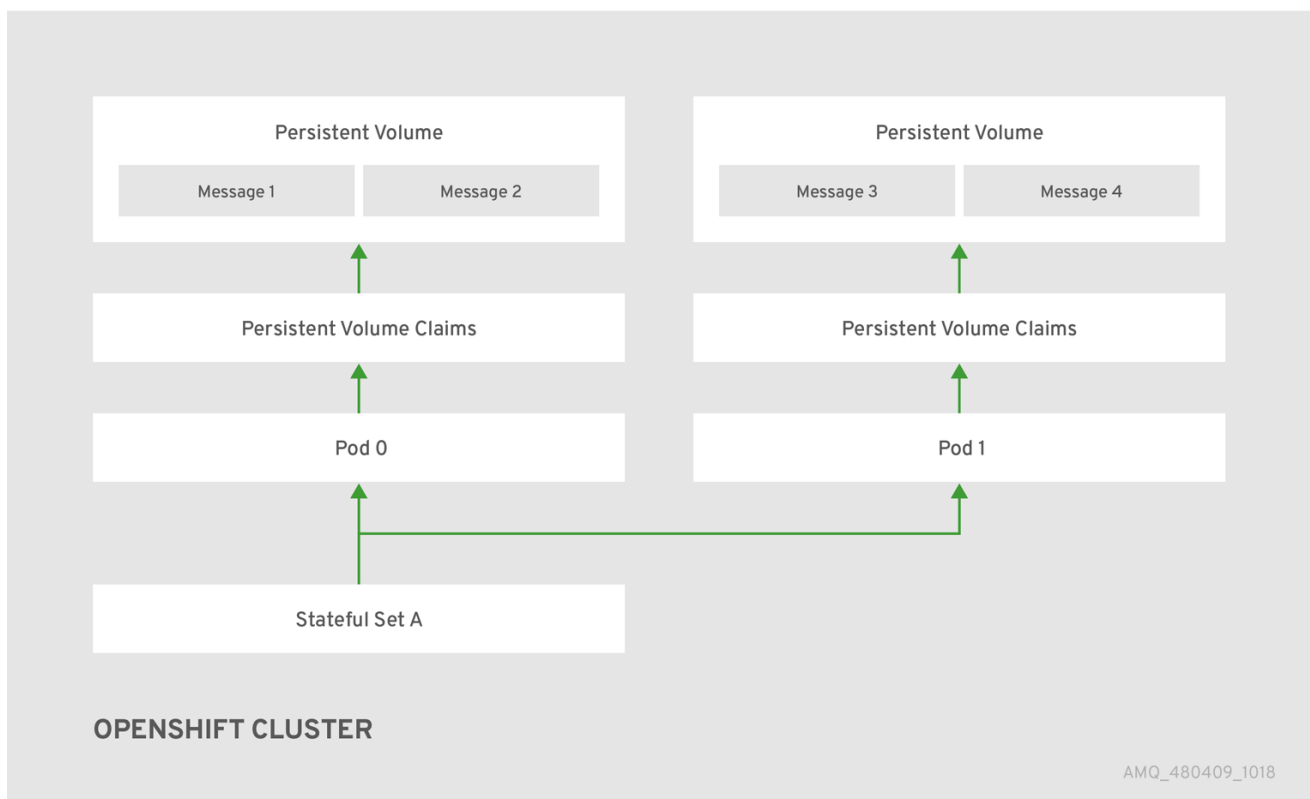
CHAPTER 7. HIGH AVAILABILITY

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*.

7.2. MESSAGE MIGRATION

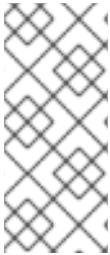
7.2.1. Message migration overview

Message migration is how you ensure the integrity of messaging data when a broker in a clustered deployment shuts down due to failure or intentional scaledown of the deployment. Message migration, which uses a method called *Pod draining*, refers to the removal and redistribution of "orphaned" messages from the persistent volume used by the broker to store messaging data. With message migration enabled, the scaledown controller, which is part of the AMQ Broker Operator, detects

shutdown of any broker Pods in your deployment. The scaledown controller starts a dedicated drainer Pod for each broker Pod that is shut down, to prepare for message migration. Each drainer Pod connects to one of the remaining live broker Pods in the cluster and migrates messages over to that live broker Pod. After migration is complete, each drainer Pod shuts down. Persistent volumes previously used by running brokers are returned to a "Released" state.

**NOTE**

The scaledown controller within the AMQ Broker Operator can operate only within a single OpenShift project. The controller cannot migrate messages between brokers in separate projects.

**NOTE**

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

7.2.1.1. How does message migration work?

When you enable message migration in a broker deployment created using the AMQ Broker Operator, a scaledown controller is started by the Operator 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.

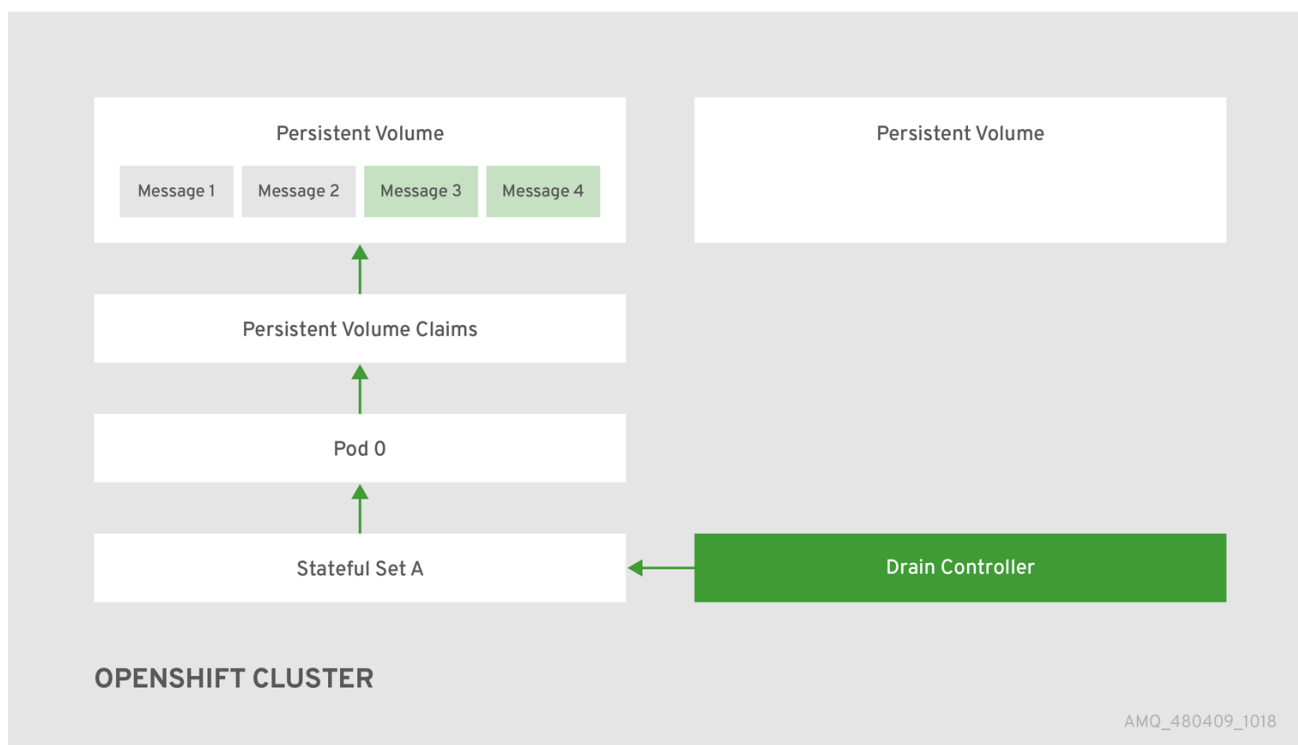
If the ordinal on the volume claim is greater than the ordinal on the existing broker Pods, then the Pod at that ordinal has been terminated and the data must be migrated to another broker.

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.

**NOTE**

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

Figure 7.2. The scaledown controller registers itself, deletes the PVC, and redistributes messages on the persistent volume.



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

Additional resources

- For an example of message migration when you scale down a broker deployment, 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,storm,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,stormq,mqtt,horntq	The protocols to be accepted by the broker
AMQ_QUEUES	Queues	demoQueue	Creates an anycast queue called demoQueue
AMQ_ADDRESSES	Addresses	demoTopic	Creates an address (or topic) called demoTopic. By default, this address has no assigned routing type.
VOLUME_CAPACITY	AMQ Volume Size	1Gi	The persistent volume size created for the journal
AMQ_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

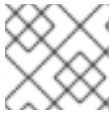
Deploy a clustered set of brokers where each broker runs in its own Pod.

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,horntq	The protocols to be accepted by the broker
AMQ_QUEUES	Queues	demoQueue	Creates an anycast queue called demoQueue
AMQ_ADDRESSES	Addresses	demoTopic	Creates an address (or topic) called demoTopic. By default, this address has no assigned routing type.

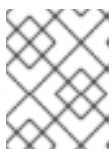
Environment variable	Display Name	Value	Description
VOLUME_CAPACITY	AMQ Volume Size	1Gi	The persistent volume size created for the journal
AMQ_CLUSTERED	Clustered	true	This needs to be true to ensure the brokers cluster
AMQ_CLUSTER_USER	cluster user	generated	The username the brokers use to connect with 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,stopmp,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.

- Scale up the Pods to three to create a cluster of brokers.

```
$ oc scale statefulset broker-amq --replicas=3
statefulset "broker-amq" scaled
```

- Check that there are three Pods running.

```
$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
broker-amq-0  1/1     Running   0           33m
broker-amq-1  1/1     Running   0           33m
broker-amq-2  1/1     Running   0           29m
```

- If the Pod status shows **ErrImagePull** or **ImagePullBackOff**, your deployment was not able to directly pull the specified broker image from the Red Hat Container Registry. In this case, edit your 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](#).

- 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](#).

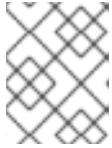
Procedure

- 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 configuration details into the **broker.xml** field.



NOTE

OpenShift Container Platform does not use a **ConfigMap** object to store the custom configuration that you specify in the **broker.xml** field, as is common for many applications deployed on this platform. Instead, OpenShift temporarily stores the specified configuration in an environment variable, before transferring the configuration to a standalone file when the broker container starts.

8.5.2. Deploy the application

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

Procedure

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

8.6. BASIC SSL CLIENT EXAMPLE

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

Prerequisites

- This tutorial builds upon [Preparing a 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-
opshift-
image/client.ks&transport.keyStorePassword=password&transport.trustStoreLocation=/home/at
aylor/projects/jboss-amq-7-broker-opshift-
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-robbined around the brokers in the cluster using the AMQ broker CLI.

Procedure

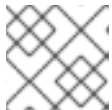
1. In a terminal create a consumer and attach it to the IP address where OpenShift is running.

```

artemis consumer --url tcp://<IP_ADDRESS>:30001 --message-count 100 --destination
queue://demoQueue

```

2. Repeat step 1 twice to start another two consumers.



NOTE

You should now have three consumers load balanced across the three brokers.

3. Create a producer to send messages.

```

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

```

4. Verify each consumer receives messages.

```

Consumer:: filter = null
Consumer ActiveMQQueue[demoQueue], thread=0 wait until 100 messages are consumed
Consumer ActiveMQQueue[demoQueue], thread=0 Consumed: 100 messages
Consumer ActiveMQQueue[demoQueue], thread=0 Consumer thread finished

```

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

CHAPTER 9. REFERENCE

9.1. CUSTOM RESOURCE DEFINITION CONFIGURATION REFERENCE

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.

The following sub-sections detail the configuration items available in the broker and addressing CRDs.

9.1.1. Broker CRD configuration reference

The broker Custom Resource Definition (CRD) enables you to configure a broker for deployment in an OpenShift project. The following table details the items that you can configure.



IMPORTANT

Configuration items marked with an asterisk (*) are required in any corresponding Custom Resource (CR) that you deploy. If you do not explicitly specify a value for a non-required item, the configuration uses the default value.

Entry	Sub-entry	Type	Example	Default value	Description
adminUser*		string	my_user	Automatic ally-generated, random value	<p>Password required for connecting to broker and management console.</p> <p>If you do not specify a value, the value is automatically generated and stored in a secret. The default secret name has a format of <Custom Resource name>-credentials-secret. For example, ex-aaocredentials-secret.</p>

Entry	Sub-entry	Type	Example	Default value	Description
adminPassword *		string	my_password	Automatically-generated, random value	<p>Password required for connecting to broker and management console.</p> <p>If you do not specify a value, the value is automatically generated and stored in a secret. The default secret name has a format of <Custom Resource name>-credentials-secret. For example, ex-aa0-credentials-secret.</p>
deploymentPlan *					Broker deployment configuration
	image *	string	registry.redhat.io/amq7/amq-broker:latest	registry.redhat.io/amq7/amq-broker:7.5	URL of broker container image to pull from Red Hat Container Registry. The default tag matches the broker Operator version.

Entry	Sub-entry	Type	Example	Default value	Description
	size*	int	2	2	Number of broker Pods to create in deployment. If you specify a value of 2 or greater, your broker deployment is clustered by default. The cluster user name and password are automatically generated and stored in the same secret as adminUser and adminPassword , by default.
	requireLogin	Boolean	true	true	Specify whether login credentials are required to connect to broker.
	persistenceEnabled	Boolean	false	true	Specify whether to use journal storage via a persistent volume (PV) created with a persistent volume claim (PVC).
	journalType	string	aio	aio	Specify whether to use asynchronous I/O (AIO) or non-blocking I/O (NIO).
	messageMigration	Boolean	true	true	Specify whether to migrate messages upon broker scaledown.
acceptors.acceptor		object			A single acceptor configuration instance.

Entry	Sub-entry	Type	Example	Default value	Description
	name*	string	my_acceptor	Not specified	Name of acceptor.
	port	int	5672	61626 for the first acceptor that you define. Default value increments by 10 for every subsequent acceptor that you define.	Port number to be used for acceptor instance.
	protocols	string	amqp,core	all	Messaging protocols to enable on acceptor instance.
	sslEnabled	Boolean	false	false	Specify whether SSL is enabled on acceptor port. If set to true , look in secret for data required to enable SSL on acceptor.

Entry	Sub-entry	Type	Example	Default value	Description
	sslSecret	string	ex-aao-my_acceptor-secret	Not specified	Secret where client truststore and broker keystore (base64-encoded) and keyStorePassword and trustStorePassword (non-encoded) are stored. If you do not specify a value for sslSecret , the acceptor uses the default secret. The default secret name has a format of <Custom Resource name>-<acceptor name>-secret .
	enabledCipherSuites	string	SSL_RSA_WITH_RC4_128_SHA, SSL_DH_anon_WITH_3DES_EDE_CBC_SHA	Not specified	Comma-separated list of cipher suites to use for SSL communication.
	enabledProtocols	string	TLSv1,TLSv1.1,TLSv1.2	Not specified	Comma-separated list of protocols to use for SSL communication.
	needClientAuth	Boolean	true	Not specified	Specify whether broker informs client that two-way SSL is required on acceptor. This property overrides wantClientAuth .

Entry	Sub-entry	Type	Example	Default value	Description
	wantClientAuth	Boolean	true	Not specified	Specify whether broker tells client that two-way SSL is requested on acceptor, but not required. Overridden by needClientAuth .
	verifyHost	Boolean	true	Not specified	Specify whether to compare the Common Name (CN) of client's SSL certificate to its host name, to verify that they match. This option applies only when two-way SSL is used.
	sslProvider	string	JDK	JDK	Specify whether SSL provider is JDK or OPENSSL.
	sniHost	string	some_regular_expression	Not specified	Regular expression to match against server_name extension on incoming SSL connections. If the names don't match, connection to the acceptor is rejected.
	expose	Boolean	true	false	Specify whether to expose acceptor outside OpenShift Container Platform
	anycastPrefix	string	.jms.topic.	Not specified	Prefix used by client to specify that the anycast routing type should be used.

Entry	Sub-entry	Type	Example	Default value	Description
	multicastPrefix	string	/queue/	Not specified	Prefix used by client to specify that the multicast routing type should be used.
	connectionsAllowed	integer	2	0	Number of connections allowed on acceptor. When this limit is reached, a DEBUG message is issued to the log, and the connection is refused. The type of client in use determines what happens when the connection is refused.
connectors.connector		object			A single connector configuration instance.
	name*	string	my_connector	N/A	Name of connector
	type	string	tcp	tcp	The type of connector to create, tcp or vm .
	host*	string	localhost	Not specified	Host name or IP address to connect to.
	port*	int	22222	Not specified	Port number to be used for connector instance.

Entry	Sub-entry	Type	Example	Default value	Description
	sslEnabled	Boolean	false	false	Specify whether SSL is enabled on connector port. If set to true , look in secret for data required to enable SSL on connector.
	sslSecret	string	ex-aaomy_connector-secret	Not specified	Secret where client truststore and broker keystore (base64-encoded) and keyStorePassword and trustStorePassword (non-encoded) are stored. If you do not specify a value for sslSecret , the connector uses the default secret. The default secret name has a format of <Custom Resource name>-<connector name>-secret .
	enabledCipherSuites	string	SSL_RSA_WITH_RC4_128_SHA, SSL_DH_anon_WITH_3DES_EDE_CBC_SHA	Not specified	Comma-separated list of cipher suites to use for SSL communication.
	enabledProtocols	string	TLSv1,TLSv1.1,TLSv1.2	Not specified	Comma-separated list of protocols to use for SSL communication.

Entry	Sub-entry	Type	Example	Default value	Description
	needClientAuth	Boolean	true	Not specified	Specify whether broker informs client that two-way SSL is required on connector. This property overrides wantClientAuth .
	wantClientAuth	Boolean	true	Not specified	Specify whether broker informs client that two-way SSL is requested on connector, but not required. Overridden by needClientAuth .
	verifyHost	Boolean	true	Not specified	Specify whether to compare Common Name (CN) of client's SSL certificate to its host name, to verify that they match. This option applies only when two-way SSL is used.
	sslProvider	string	JDK	JDK	Specify whether SSL provider is JDK or OPENSSL .
	sniHost	string	some_regular_expression	Not specified	Regular expression to match against server_name extension on SSL connection. If the names don't match, the connector connection is rejected.

Entry	Sub-entry	Type	Example	Default value	Description
	expose	Boolean	true	false	Specify whether to expose connector outside OpenShift Container Platform.
console					Configuration of broker management console.
	expose	Boolean	true	false	Specify whether to expose management console port.
	sslEnabled	Boolean	true	false	Specify whether to use SSL on management console port.
	sslSecret	string	ex-aao-my_console-secret	Not specified	Secret where client truststore and broker keystore (base64-encoded) and keyStorePassword and trustStorePassword (non-encoded) are stored. If you do not specify a value for sslSecret , the console uses the default secret. The default secret name has a format of <Custom Resource name>-console-secret .
	useClientAuth	Boolean	true	false	Specify whether management console requires client authorization.

9.1.2. Addressing CRD configuration reference

The addressing Custom Resource Definition (CRD) enables you to define addresses and queues and associated routing types to be created in your broker. The following table details the items that you can configure.



IMPORTANT

Configuration items marked with an asterisk (*) are required in any corresponding Custom Resource (CR) that you deploy. If you do not explicitly specify a value for a non-required item, the configuration uses the default value.

Entry	Type	Example	Default value	Description
addressName*	string	address0	Not specified	Address name to be created in broker.
queueName*	string	queue0	Not specified	Queue name to be created in broker.
routingType*	string	anycast	Not specified	Routing type to be used - anycast or multicast.

9.2. APPLICATION TEMPLATE PARAMETERS

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

Table 9.1. Application template parameters

Parameter	Description
AMQ_ADDRESSES	Specifies the addresses available by default on the broker on its startup, in a comma-separated list.
AMQ_ANICAST_PREFIX	Specifies the anycast prefix applied to the multiplexed protocol ports 61616 and 61617.
AMQ_CLUSTERED	Enables clustering.
AMQ_CLUSTER_PASSWORD	Specifies the password to use for clustering. 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.

Parameter	Description
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.
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.

Parameter	Description
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.
IMAGE_STREAM_NAMESPACE	Specifies the image stream name space. Used in the ssl and basic templates.
OPENSIFT_DNS_PING_SERVICE_PORT	Specifies the port number for the OpenShift DNS ping.
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

9.3. LOGGING

In addition to viewing the OpenShift logs, you can troubleshoot a running AMQ Broker on OpenShift Container Platform image by viewing the AMQ logs that are output to the container's console.

Procedure

- At the command line, run the following command:

```
$ oc logs -f <pass:quotes[<pod-name>]> <pass:quotes[<container-name>]>
```

Revised on 2021-05-14 10:13:54 UTC