Red Hat AMQ 7.2

Deploying AMQ Broker on OpenShift Container Platform

For Use with AMQ Broker 7.2

Last Updated: 2018-10-23

Red Hat AMQ 7.2 Deploying AMQ Broker on OpenShift Container Platform

For Use with AMQ Broker 7.2

Legal Notice

Abstract

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

Table of Contents

CHAPTER 1. INTRODUCTION

Red Hat AMQ Broker 7.2 is available as a containerized image that is provided for use with OpenShift Container Platform 3.11 (AMQ Broker on OCP).

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

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

1.1. VERSION COMPATIBILITY AND SUPPORT

For details about OpenShift Container Platform 3.11 image version compatibility, see the OpenShift and Atomic Platform Tested Integrations page.

1.2. UNSUPPORTED FEATURES

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

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

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

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

CHAPTER 2. INSTALLING AND DEPLOYING AMQ BROKER ON OPENSHIFT CONTAINER PLATFORM

2.1. INSTALLING AMQ BROKER ON OPENSHIFT CONTAINER PLATFORM IMAGE STREAMS AND APPLICATION TEMPLATES

The AMQ Broker on OpenShift Container Platform images are not available in the service catalog. You must manually install them by using the procedures in this section.

Procedure

1. Log in to OpenShift as a cluster administrator (or as a user that has project administrator access to the global **openshift** project), for example:

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

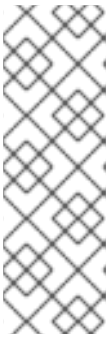
2. At the command line, run the following commands to update the AMQ Broker on OpenShift Container Platform image stream in the **openshift** project:

```
$ oc replace --force  -f \
https://raw.githubusercontent.com/jboss-container-images/jboss-amq-
7-broker-openshift-image/72-1.0.GA/amq-broker-7-image-streams.yaml

$ oc replace --force -f \
https://raw.githubusercontent.com/jboss-container-images/jboss-amq-
7-broker-openshift-image/72-1.0.GA/amq-broker-7-scaledown-
controller-image-streams.yaml

$ oc import-image amq-broker-72-openshift:1.0

$ oc import-image amq-broker-72-scaledown-controller-openshift:1.0
```



NOTE

AMQ Broker on OpenShift Container Platform leverages StatefulSets and Deployments resources for use with the ***-persistence** templates. These are Kubernetes-native resources that can consume image streams only from a local namespace, not the shared **openshift** namespace. This is because the image streams must be created in the same namespace where the template will be instantiated. Also, **-n openshift** is an optional parameter to use if you need to create a template in the shared namespace.

3. Run the following command to update the AMQ Broker templates. Using the **--force** option with the **oc replace** command creates or updates the resources

```
$ for template in amq-broker-72-basic.yaml \
amq-broker-72-ssl.yaml \
amq-broker-72-custom.yaml \
amq-broker-72-persistence.yaml \
amq-broker-72-persistence-ssl.yaml \
amq-broker-72-persistence-clustered.yaml \
amq-broker-72-persistence-clustered-ssl.yaml;
```

```
 do
 oc replace --force -f \
https://raw.githubusercontent.com/jboss-container-images/jboss-amq-
7-broker-openshift-image/72-1.0.GA/templates/${template}
 done
```

2.2. DEPLOYING AMQ BROKER ON OPENSHIFT CONTAINER PLATFORM IMAGE

The AMQ Broker on OpenShift Container Platform image requires a service account for deployments. Service accounts are API objects that exist within each project. Three service accounts are created automatically in every project:

- **`builder`**: This service account is used by build pods. It contains the **`system:image-builder`** role from which you can push images to any image stream in the project using the internal Docker registry.
- **`deployer`**: This service account is used by deployment pods. It contains the **`system:deployer`** role from which you can view and modify replication controllers and pods in the project.
- **`default`**: This service account is used to run all other pods unless you specify a different service account.

Service accounts can be created or deleted like any other API object. For multiple-node deployments, the service account must have the **`view`** role enabled so the various pods in the cluster can be discovered and managed.

In addition, you must configure SSL to enable connections to AMQ Broker from outside of the OpenShift Container Platform instance. The type of discovery protocol that is used for discovering AMQ Broker mesh endpoints is JGroups with OpenShift.dns ping protocol.

Procedure

- Add the **`view`** role to the service account:

```
$ oc policy add-role-to-user view -z default
```

Additional resources

- For more information on how to configure SSL, see Configuring SSL.

CHAPTER 3. CONFIGURING SSL FOR AMQ BROKER ON OPENSHIFT CONTAINER PLATFORM

3.1. 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.

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
```

3.2. 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>
```

3.3. CREATING AN SSL ROUTE

After the AMQ Broker on OpenShift Container Platform image has been deployed, an SSL route needs to be created for the AMQ Broker transport protocol port to allow connections to AMQ Broker outside of OpenShift.

In addition, selecting **Passthrough** for **TLS Termination** relays all communication to AMQ Broker without the OpenShift router decrypting and resending it. Only SSL routes can be exposed because the OpenShift router requires SNI to send traffic to the correct service.

The default ports for the various AMQ Broker transport protocols are:

Table 3.1. Default ports for AMQ Broker transport protocols

AMQ Broker transport protocol	Default port
All protocols	61616
All protocols (SSL)	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 4. 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.

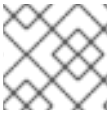
NOTE

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 5. CONFIGURING CLIENT CONNECTIONS

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

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



NOTE

External clients do not support HA.

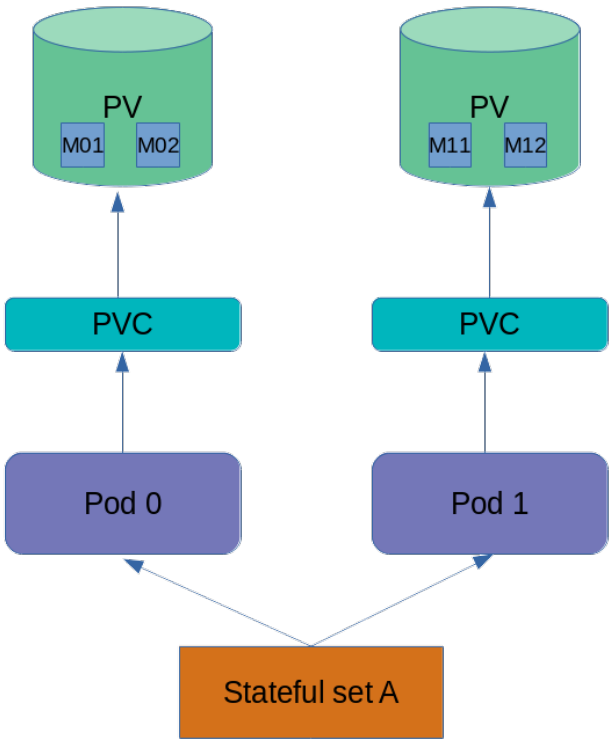
CHAPTER 6. HIGH AVAILABILITY

6.1. 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 6.1. StatefulSet working normally



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

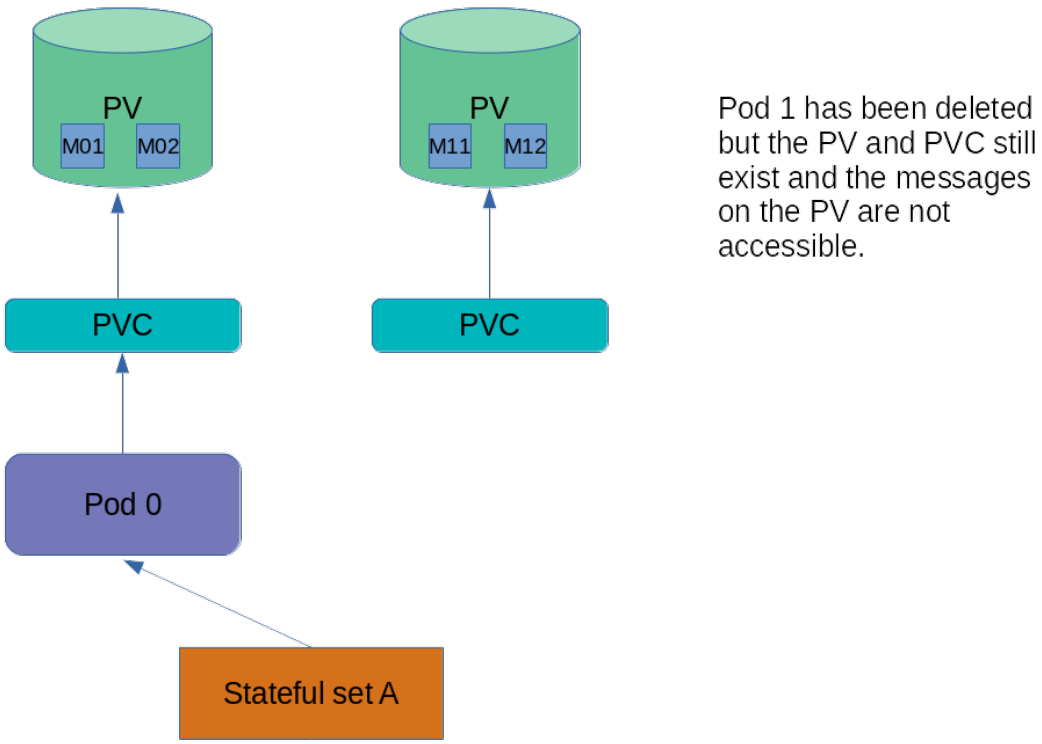
Additional resources

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

6.2. MESSAGE MIGRATION

Pod draining is the method used to manage the integrity of messaging data in a Kubernetes StatefulSet. Pod draining is used for message migration, which refers to the removal and redistribution of "orphaned" messages from a persistent volume, due to broker pod failure or intentional scale down.

Figure 6.2. One of the brokers has gone down



6.3. HOW DOES POD DRAINING AND MESSAGE MIGRATION WORK?

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

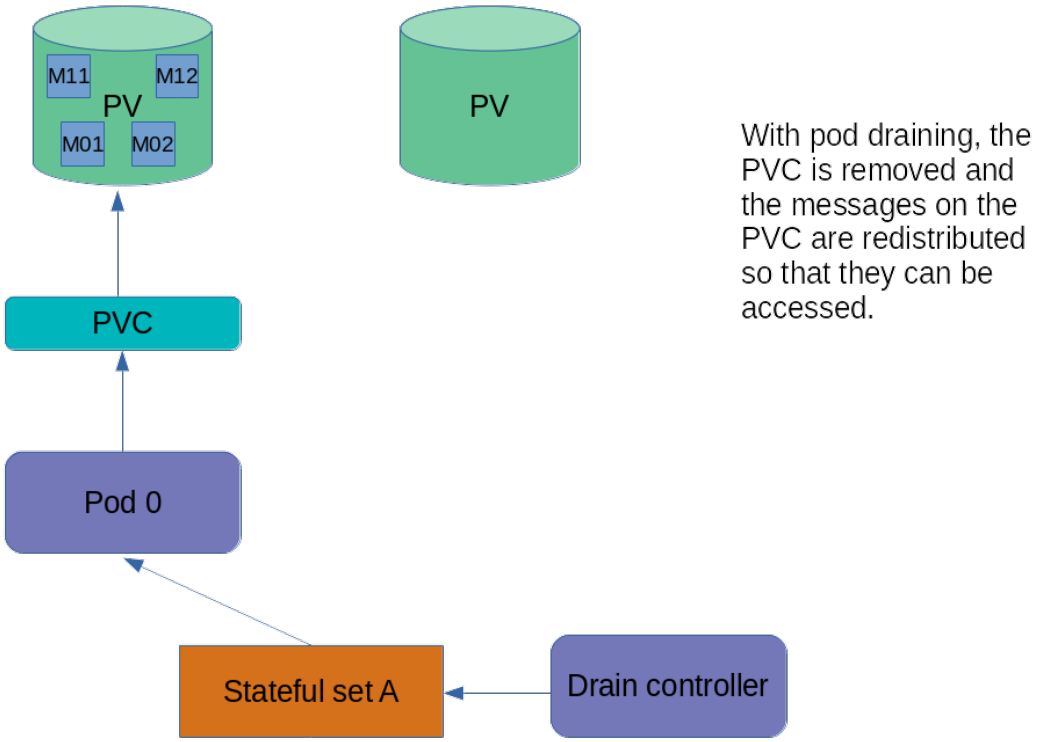
The scaledown controller registers itself and listens for Kubernetes events that are related to persistent volume claims (PVCs) in the project (openshift) namespace.

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

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

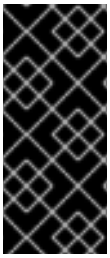
When these conditions are met, a drainer pod is started. The drainer pod runs the broker and executes the message migration. Then the drainer pod identifies an alternative broker pod to which the orphaned PVC messages can be migrated.

Figure 6.3. The scaledown controller registers itself, deletes the PVC, and redistributes messages on the PersistentVolume.



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

CHAPTER 7. MESSAGE MIGRATION WHEN SCALING DOWN PODS



IMPORTANT

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

When a persistent template is used to deploy a broker pod that uses a StatefulSet, that broker pod has its own external file system, which remains intact, even if the pod stops or restarts. However, if pods are scaled down and not restarted, data and information such as messages, destinations, or transactions are no longer available to clients.

Message migration addresses the issue of unavailable data and can be obtained by applying the scaledown controller image, which monitors each broker pod. If a broker pod is scaled down or stopped, the scaledown controller recovers the messages by transferring its contents to another broker running in the cluster.

If broker pods are scaled down to 0 (zero), message migration does not occur, since there is no running broker pod to which the message data can be migrated.

7.1. INSTALLING THE SCALEDOWN CONTROLLER

AMQ Broker on OCP message migration capabilites are packaged in the scaledown controller container image. This section describes how to enable the broker message migration capabilities on OpenShift Container Platform image streams and application templates.

Procedure

1. Log in to OpenShift as a cluster administrator (or as a user that has project administrator access to the global OpenShift project), for example:

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

2. Run the following command to update the AMQ Broker templates:

```
$ oc create -n amq-demo -f https://raw.githubusercontent.com/jboss-
container-images/jboss-amq-7-broker-openshift-image/72-
1.0.GA/templates/amq-broker-72-persistence-clustered-controller.yaml
```

NOTE

You could receive an "already exists" error messages after invoking the **`create`** command.

7.2. USING THE SCALEDOWN CONTROLLER

To deploy the scaledown controller to migrate messages and drain pods, run the the StatefulSet scaledown controller at the broker pod namespace level. The StatefulSet scaledown controller must be

deployed in the same namespace as the stateful applications (in this case, broker pods). It operates *only* on StatefulSets in that namespace.

NOTE

You *do not* need cluster-level privileges to complete this procedure. You must run the StatefulSet scaledown controller at the namespace level.

Prerequisites

- An understanding of Kubernetes StatefulSets definition and processing.

Procedure

1. Configure the Broker on OCP StatefulSet controller template in your namespace.
2. Configure the scaledown controller template in your StatefulSet definition. The following code example represents the drainer pod definition:

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: my-statefulset
  annotations:
    statefulsets.kubernetes.io/drainer-pod-template: |
      {
        "metadata": {
          "labels": {
            "app": "datastore-drainer"
          }
        },
        "spec": {
          "containers": [
            {
              "name": "drainer",
              "image": "my-drain-container",
              "volumeMounts": [
                {
                  "name": "data",
                  "mountPath": "/var/data"
                }
              ]
            }
          ]
        }
      }
spec:
```

CHAPTER 8. TUTORIALS

These procedures assume an OpenShift Container Platform 3.11 instance similar to that created in OpenShift Container Platform 3.11 Getting Started.

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

8.1. PREPARING AN AMQ BROKER DEPLOYMENT

Procedure

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

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

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

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

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

f. Add the secret to the service account created earlier:

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

8.2. CONNECTING TO THE AMQ CONSOLE

Connect to the AMQ Console from the OpenShift web console.

Procedure

1. In a web browser, navigate to the OpenShift web console and log in.
2. Navigate to the broker pod and click **Connect**, located in the template information.
3. Click **Open Java Console** for OpenShift Container Platform 3.11.

8.3. DEPLOYING A BASIC BROKER

Deploy a basic broker that is ephemeral and does not support SSL. This tutorial covers how to create transports, addresses, and queues.

NOTE

This broker does not support SSL and is not accessible to external clients. Only clients running internally on the OpenShift cluster are able to connect.

Prequisites

- Preparing a Broker

8.3.1. 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 find results that match **`amq`**. You might need to click **See all** to show the desired application template.
5. Select the **`amq-broker-72-basic`** template which is labeled **`Red Hat AMQ Broker 7.2(Ephemeral, no SSL)`**.
6. Set the following environment variables in the configuration and click **create**.

   Table 8.1. Example template

Environment variable	Display Name	Value	Description
AMQ_PROTOCOL	AMQ Protocols	openwire,amqp,stomp,mqtt,hornetq	The protocols to be accepted by the broker
AMQ_QUEUES	Queues	demoQueue	Creates an anycast queue called demoQueue
AMQ_ADDRESSES	Addresses	demoTopic	Creates a multicast address (or topic) called demoTopic
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

You can also deploy the image from the command line:

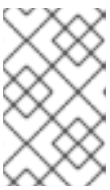
```
$ oc new-app --template=amq-broker-72-basic \
   -e AMQ_PROTOCOL=openwire,amqp,stomp,mqtt,hornetq \
   -e AMQ_QUEUES=demoQueue \
   -e AMQ_ADDRESSES=demoTopic \
   -e AMQ_USER=amq-demo-user \
   -e ADMIN_PASSWORD=password \
```

8.3.2. Deploy the application

After the application is created, you need to deploy it. Deploying the application creates a pod and starts the broker.

Procedure

1. After the deployment has been created, choose **Deployments** from the **Applications** menu.
2. Click on the **broker-amq** deployment.
3. Click the **deploy** button to deploy the application.



NOTE

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

4. After the deployment has appeared on the list, click on it to view the state of the pods. Click on the pod and then click the **log** tab to view the broker logs and verify its state. You should see the queue previously created.

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

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

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

You can also use the OpenShift client to access the shell by using the pod name.

```
oc rsh broker-amq-1-9x89r
```

8.4. DEPLOYING A BASIC BROKER WITH SSL

Deploy a basic broker that is ephemeral and supports SSL. This tutorial covers how to create transports, addresses, and queues.

8.4.1. 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-72-ssl`** template which is labeled **`Red Hat AMQ Broker 7.2 (Ephemeral, with SSL)`**.

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

Table 8.2. Example template

Environment variable	Display Name	Value	Description
AMQ_PROTOCOL	AMQ Protocols	openwire,amqp,stomp,mqtt,hornetq	The protocols to be accepted by the broker
AMQ_QUEUES	Queues	demoQueue	Creates an anycast queue called demoQueue
AMQ_ADDRESSES	Addresses	demoTopic	Creates a multicast address (or topic) called demoTopic
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_KEYSTORE_PASSWORD	AMQ Keystore Password	password	The password used when creating the Truststore
AMQ_TRUSTSTORE	AMQ Keystore Password	password	The password used when creating the Keystore

8.4.2. Deploying the application

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

Procedure

1. Choose **Deployments** from the **Applications** menu.
2. Click on the **broker-amq** deployment.
3. Click on the **deploy** button to deploy the application.

8.4.3. 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

Only one broker can be scaled up. You cannot scale up multiple brokers.

Procedure

1. From the **Services** menu choose **broker-amq-tcp-ssl**

2. From the **Action** menu and choose **Create a route** .
3. Select the **Secure route** check box to display the TLS parameters.
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. View the route by going to the **routes** menu. 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.

8.5. DEPLOYING A BASIC BROKER WITH PERSISTENCE AND SSL

Deploy a persistent broker that supports SSL. When a broker needs persistence it 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 the pod crashes or restarts. This means 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.

8.5.1. 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 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-72-persistence-ssl`** template which is labelled **`Red Hat AMQ Broker 7.2(Persistence, with SSL)`**.
6. Set the following values in the configuration and click **create**.

   Table 8.3. Example template

Environment variable	Display Name	Value	Description
AMQ_PROTOCOL	AMQ Protocols	openwire,amqp,stomp,mqtt,hornetq	The protocols to be accepted by the broker
AMQ_QUEUES	Queues	demoQueue	Creates an anycast queue called demoQueue
AMQ_ADDRESSES	Addresses	demoTopic	Creates a multicast address (or topic) called demoTopic
AMQ_Volume_Size	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_KEYSTORE_PASSWORD	AMQ Keystore Password	password	The password used when creating the Truststore
AMQ_TRUSTSTORE	AMQ Keystore Password	password	The password used when creating the Keystore

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. Once the deployment has been created, choose **StatefulSets** from the **Applications** menu
2. Click **broker-amq** deployment.
3. Click **deploy** to deploy the application.
4. Click the deployment to see the state of the pods.
5. Click the pod and then click the **log** tab to see the brokers logs to verify its state. You should see the queue we pre created via the template get deployed.
6. 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[TEST], thread=0 Started to calculate elapsed
time ...
```

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

You can also use the OpenShift client to access the shell by using the pod name

```
oc rsh broker-amq-1-9x89r
```

7. Now scale down the broker using the oc command.

```
sh-4.2$ oc scale statefulset broker-amq --replicas=0
statefulset "broker-amq" scaled
```

You can use the console to check that the pod count is 0

8. Now scale the broker back up to 1.

```
sh-4.2$ oc scale statefulset broker-amq --replicas=1
statefulset "broker-amq" scaled
```

9. 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[TEST], thread=0 wait until 1000 messages are
consumed
Received 1000
Consumer ActiveMQQueue[TEST], thread=0 Consumed: 1000 messages
Consumer ActiveMQQueue[TEST], thread=0 Consumer thread finished
```

Additional resources

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

8.5.3. 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

Only one broker can be scaled up. You cannot scale up multiple brokers.

Procedure

1. From the **Services** menu choose **broker-amq-tcp-ssl**
2. From the **Action** menu and choose **Create a route** .
3. Select the **Secure route** check box to display the TLS parameters.
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. View the route by going to the **routes** menu. 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.

8.6. DEPLOYING A SET OF CLUSTERED BROKERS

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

8.6.1. Distributing messages

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

This 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.

8.6.2. 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-72-persistence-clustered** template which is labeled **Red Hat AMQ Broker 7.2(no SSL, clustered)**.

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

Table 8.4. Example template

Environment variable	Display Name	Value	Description
AMQ_PROTOCOL	AMQ Protocols	openwire,amqp,stomp,mqtt,hornetq	The protocols to be accepted by the broker
AMQ_QUEUES	Queues	demoQueue	Creates an anycast queue called demoQueue
AMQ_ADDRESSES	Addresses	demoTopic	Creates a multicast address (or topic) called demoTopic
AMQ_Volume_Size	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

8.6.3. Deploying the application

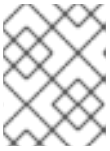
Once the application has been created it needs to be deployed. Deploying the application creates a pod and starts the broker.

Procedure

1. Once the deployment has been created choose **Stateful Sets** from the **Applications** menu.

2. Click on the **broker-amq** deployment.

3. Click on the **deploy** button 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.

```
sh-4.2$ oc scale statefulset broker-amq --replicas=3
statefulset "broker-amq" scaled
```

5. Check that there are three pods running.

```
sh-4.2$ jboss-amq-7-broker-openshift-image]$ 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. Verify that the brokers have clustered with the new pod by checking the logs.

```
sh-4.2$ jboss-amq-7-broker-openshift-image]$ 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
```

8.6.4. Creating a route for the 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.

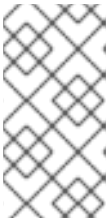
NOTE

In future releases each pod will have its own integrated console available through the use of the pod. It uses wildcard routing to expose each broker on its own hostname.

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 important configuration here is **host: star.broker-amq-headless.amq-demo.svc**. This 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 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-0.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.

8.7. DEPLOYING A SET OF CLUSTERED SSL BROKERS

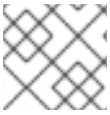
Deploy a clustered set of brokers, where each broker runs in its own pod and the broker is configured to accept connections using SSL.

8.7.1. Distributing messages

Message distribution is configured to use *ON_DEMAND*. This means that when a message arrives at a clustered broker, it is distributed in a round-robin fashion to any broker that has consumers.

This 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.

8.7.2. Deploying the image and template

Prerequisites

- This procredure 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-72-persistence-clustered-ssl`** template which is labeled **`Red Hat AMQ Broker 7.2(no SSL, clustered)`**.
6. Set the following values in the configuration and click **create**.

Table 8.5. Example template

Environment variable	Display Name	Value	Description

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 a multicast address (or topic) called demoTopic
AMQ_Volume_Size	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_KEYSTORE_PASSWORD	AMQ Keystore Password	password	The password used when creating the Truststore
AMQ_TRUSTSTORE	AMQ Keystore Password	password	The password used when creating the Keystore

8.7.3. Deploying the application

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

Procedure

1. Choose **StatefulSets** from the **Applications** menu, once the deployment has been created.
2. Click on the **broker-amq** deployment.
3. Click on the **deploy** button to deploy the application.

NOTE

The default number of replicas for a clustered template is 0, so you will not see any pods.

4. Scale up the pods to three to create a cluster of brokers.

```
sh-4.2$ oc scale statefulset broker-amq --replicas=3
statefulset "broker-amq" scaled
```

5. Check that there are three pods running.

```
sh-4.2$ jboss-amq-7-broker-openshift-image]$ 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. Verify the brokers have clustered with the new pod by checking the logs.

```
sh-4.2$ jboss-amq-7-broker-openshift-image]$ 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
```

8.7.4. Creating a route for the management console

The clustering templates do not expose the console by default. This is because 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.

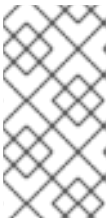
NOTE

In future releases each pod will have its own integrated console available through the pod details page. This is resolved by using wildcard routing to expose each broker on its own hostname.

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
  tls:
    termination: passthrough
  to:
    kind: Service
    name: broker-amq-headless
    weight: 100
  wildcardPolicy: Subdomain
  host: star.broker-amq-headless.amq-demo.svc
```



NOTE

The important configuration here is **host: star.broker-amq-headless.amq-demo.svc**. This is the hostname used for each pod in the broker. The star is replaced by the pod name. For instance, if the pod name is **broker-amq-0**, its hostname is **broker-amq-0.broker-amq-headless.amq-demo.svc**.

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

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

NOTE

The /etc/hosts entries do not point directly to the brokers, as the brokers running in the pods have IP addresses in the range of the pod-range for a given node (e.g. 10.128.x.y or 10.130.u.v). In the case of a nodePort configuration you can point the hostname to either of the node IP addresses and the name will get routed appropriately to the correct broker pod. In the case of having a headless service with a SSL route, point each of the names to the IP address of the node running the OpenShift router (i.e. haproxy instance).

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

Additional resources

- For more information on messaging, see Enabling Message Redistribution.

8.8. 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.

NOTE

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

Prerequisites

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

8.8.1. 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-72-custom`** template which is labeled **`Red Hat AMQ Broker 7.2(Ephemeral, no SSL)`**.
6. In the configuration, update the **`broker.xml`** with the custom **`configuration`** you would like to use and click **create**

NOTE

Use a text editor to create the broker's xml configuration. Cut and paste confguration details into the value field.

8.8.2. 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. Once the deployment has been created choose **Deployments** from the **Applications** menu
2. Click on the **broker-amq** deployment
3. Click on the **deploy** button to deploy the application.

8.9. 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

8.9.1. Configuring the client

```
Create a sample client that can be updated to connect to the SSL broker.
The following procedure builds upon
link:https://access.redhat.com/documentation/en-
us/red_hat_amq/7.2/html/using_the_amq_jms_client/examples[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.trustSt
```

```
oreLocation=<truststore-
path>/client.ts&transport.trustStorePassword=password&transport.veri
fyHost=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!
```

8.10. 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-72-persistence-clustered-ssl`** template.

8.10.1. 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.

8.10.2. Connecting the clients

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

Procedure

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

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

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

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

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

```
queue.myDestinationLookup = demoQueue
```

4. Execute the sender client code to send a text message.

5. Execute the receiver client code to receive the text message. You should see:

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

Additional resources

- For more information on using the AMQ JMS client, see AMQ JMS Examples.

8.11. 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-72-persistence-clustered`** template.

8.11.1. 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.

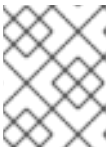
Prerequisites

- 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-72-statefulset-clustered
    xpaas: 1.4.12
  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**.

8.11.2. Connecting the clients

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

Procedure

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

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

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

NOTE

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

3. Create a producer to send messages.

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

4. Verify each consumer receives messages.

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

8.12. 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.artem
is:broker=%22broker%22,component=addresses,address=%22TESTQUEUE%22,s
ubcomponent=queues,routing-
type=%22anycast%22,queue=%22TESTQUEUE%22/MaxConsumers

{"request":
{"mbean":"org.apache.activemq.artemis:address=\"TESTQUEUE\",broker=\
"broker\",component=addresses,queue=\"TESTQUEUE\",routing-
type=\"anycast\",subcomponent=queues","attribute":"MaxConsumers","ty
pe":"read"},"value":-1,"timestamp":1528297825,"status":200}
```

CHAPTER 9. REFERENCE

9.1. APPLICATION TEMPLATE PARAMETERS

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

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_ADMIN_PASSWORD	Specifies the password used for authentication to the broker. If no value is specified, a random password is generated.
AMQ_ADMIN_USERNAME	Specifies the user name used as an administrator authentication to the broker. If no value is specified, a random user name is generated.
AMQ_ANYCAST_PREFIX	Specifies the anycast prefix applied to the multiplexed protocol ports 61616 and 61617.
AMQ_CLUSTERED	Enables clustering.
AMQ_CLUSTER_PASSWORD	Specifies the password to use for clustering. If no value is specified, a random password is generated.
AMQ_CLUSTER_USER	Specifies the cluster user to use for clustering. If no value is specified, a random user name is generated.
AMQ_DATA_DIR	Specifies the directory for the data. Used in stateful sets.
AMQ_DATA_DIR_LOGGING	Specifies the directory for the data directory logging.
AMQ_EXTRA_ARGS	Specifies additional arguments to pass to **`artemis create`**.
AMQ_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.

Parameter	Description
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_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**, anonymous access is permitted. If set to **false**, log in is required. The default value is **false**.
AMQ_RESET_CONFIG	If set to **true**, overwrites the configuration at the destination directory.
AMQ_ROLE	Specifies the name for the role created. The default value is **amq**.
AMQ_TRANSPORTS	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 available 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_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.

Parameter	Description
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.
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.
IMAGE	Specifies the image. Used in the **persistence**, **persistent-ssl**, and **statefulset-clustered** templates.
IMAGE_STREAM_NAMESPACE	Specifies the image stream name space. Used in the **ssl** and **basic** templates.
OPENSHIFT_DNS_PING_SERVICE_PORT	Specifies the port number for the OpenShift DNS ping.
VOLUME_CAPACITY	Specifies the size of the persistent storage for database volumes.

9.2. SECURITY

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

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

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 2018-10-23 14:31:13 UTC