Red Hat JBoss A-MQ 6.2

Red Hat JBoss A-MQ for OpenShift

Learn to install and develop with Red Hat JBoss A-MQ for OpenShift

Last Updated: 2017-11-16

Red Hat JBoss A-MQ 6.2 Red Hat JBoss A-MQ for OpenShift

Learn to install and develop with Red Hat JBoss A-MQ for OpenShift

Legal Notice

Abstract

Guide to using the Red Hat JBoss A-MQ for OpenShift

Table of Contents

CHAPTER 1. INTRODUCTION

1.1. WHAT IS RED HAT JBOSS A-MQ?

Red Hat JBoss A-MQ (A-MQ) is available as a containerized image that is designed for use with OpenShift. It allows developers to quickly deploy an A-MQ message broker in a hybrid cloud environment.

A-MQ, based on Apache ActiveMQ, is a JMS 1.1-compliant messaging system. It consists of a broker and client-side libraries that enable remote communication among distributed client applications. A-MQ provides numerous connectivity options and can communicate with a wide variety of non-JMS clients through its support of the OpenWire and STOMP wire protocols.

CHAPTER 2. BEFORE YOU BEGIN

2.1. COMPARISON: RED HAT JBOSS A-MQ FOR OPENSHIFT AND A-MQ FOR OPENSHIFT IMAGE

This topic details the differences between the regular release of JBoss A-MQ and the A-MQ for OpenShift image, and provides instructions specific to running and configuring the A-MQ for OpenShift image. Documentation for other JBoss A-MQ functionality not specific to the A-MQ for OpenShift image can be found in the Red Hat JBoss A-MQ documentation on the Red Hat Customer Portal .

Differences between the regular release of JBoss A-MQ and the A-MQ for OpenShift image:

- The Karaf shell is not available.
- The Fuse Management Console (Hawtio) is not available.
- To connect to the A-MQ web console, click the **Connect** button in the A-MQ pod of the OpenShift web console, or the **Open Java Console** button in OpenShift Container Platform.
- Configuration of the broker can be performed:
  - using parameters specified in the A-MQ application template, as described in Section 6.1, “Application Template Parameters for Persistent Configuration”.
  - using the S2I (Source-to-image) tool, as described in Section 6.2, “Configuration using S2I”.
- Clients for the A-MQ xPaaS image need to specify the OpenShift API port (443) when setting the broker URL for SSL connections. Otherwise, A-MQ will attempt to use the default SSL port (61617).

CHAPTER 3. USING THE A-MQ FOR OPENSHIFT IMAGE STREAMS AND APPLICATION TEMPLATES

Red Hat JBoss A-MQ images were automatically created during the installation of OpenShift along with the other default image streams and templates.

3.1. VERSION COMPATIBILITY AND SUPPORT

See the xPaaS part of the OpenShift and Atomic Platform Tested Integrations page for details about OpenShift image version compatibility.

3.2. INITIAL SETUP

The Tutorials in this guide follow on from and assume an OpenShift instance similar to that created in the OpenShift Primer.

CHAPTER 4. GET STARTED

4.1. USING THE A-MQ FOR OPENSHIFT IMAGE STREAMS AND APPLICATION TEMPLATES

Red Hat JBoss A-MQ images were automatically created during the installation of OpenShift along with the other default image streams and templates.

4.2. DEPLOYMENT CONSIDERATIONS FOR THE A-MQ FOR OPENSHIFT IMAGE

4.2.1. Service Accounts

The A-MQ for OpenShift image requires a service account for deployments. Service accounts are API objects that exists within each project. Three service accounts are created automatically in every project: builder, deployer, and default.

- builder: This service account is used by build pods. It has **system:image-builder** role which allows pushing images to any image stream in the project using the internal Docker registry.
- deployer: This service account is used by deployment pods. It has **system:deployer** role which allows viewing and modifying replication controllers and pods in the project.
- default: This service account used to run all other pods unless you specify a different service account.

4.2.2. Creating the Service Account

Service accounts are API objects that exists within each project and can be created or deleted like any other API object. For multiple node deployments, the service account must have the **view** role enabled so that it can discover and manage the various pods in the cluster. In addition, you will need to configure SSL to enable connections to A-MQ from outside of the OpenShift instance. There are two types of discovery protocols that can be possibly used for discovering of AMQ mesh endpoints. To use OpenShift DNS service, DNS based discovery protocol is used and in case of Kubernets REST API, Kubernetes based discovery protocol is used. To use the Kubernetes based discovery protocol, create a new service account and grant a 'view' role for the newly created service account.

1. Create the service account:

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

   OpenShift 3.2 users can use the following command to create the service account:

```
$ oc create serviceaccount <service-account-name>
```

2. Add the **view** role to the service account:

```
$ oc policy add-role-to-user view system:serviceaccount:<project-
name>:<service-account-name>
```

3. Edit the deployment configuration to run the AMQ pod with newly created service account.

```
$ oc edit dc/<deployment_config>
```

Add the serviceAccount and serviceAccountName parameters to the **spec** field, and specify the service account you want to use.

```
spec:
      securityContext: {}
      serviceAccount: <service_account>
      serviceAccountName: <service_account>
```

4.2.3. Configuring SSL

For a minimal SSL configuration to allow for connections outside of OpenShift, A-MQ 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 A-MQ for OpenShift image, which is added to the service account.
The following example commands use ***keytool***, a package included with the Java Development Kit, to generate the necessary certificates and stores:

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
```

4.2.4. Generating the A-MQ Secret

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

```
$ oc secrets new <secret-name> <broker-keystore> <broker-truststore>
```

```
$ oc secrets add sa/<service-account-name> secret/<secret-name>
```

4.2.5. Creating a Route

After the A-MQ for OpenShift image has been deployed, an SSL route needs to be created for the A-MQ transport protocol port to allow connections to A-MQ outside of OpenShift.
In addition, selecting **Passthrough** for **TLS Termination** relays all communication to the A-MQ 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. See Secured Routes for more information.
The default ports for the various A-MQ transport protocols are:
61616/TCP (OpenWire)
61617/TCP (OpenWire+SSL)
5672/TCP (AMQP)
5671/TCP (AMQP+SSL)
1883/TCP (MQTT)
8883/TCP (MQTT+SSL)
61613/TCP (STOMP)
61612/TCP (STOMP+SSL)

4.2.6. Scaling Up and Persistent Storage Partitioning

There are two methods for deploying A-MQ with persistent storage: single-node and multi-node partitioning. Single-node partitioning stores the A-MQ logs and the kahadb store directory, with the message queue data, in the storage volume. Multi-node partitioning creates additional, independent **split-*n*** directories to store the messaging queue data for each broker, where ***n*** is an incremental integer. This communication is not altered if a broker pod is updated, goes down unexpectedly, or is redeployed. When the broker pod is operational again, it reconnects to the associated split directory and continues as before. If a new broker pod is added, a corresponding **split-*n*** directory is created for that broker.

NOTE

In order to enable a multi-node configuration it is necessary to set the **AMQ_SPLIT** parameter to true, this will result in the server creating independent **split-*n*** directories for each instance within the Persistent Volume which can then be used as their data store. This is now the default setting in all persistent templates.

IMPORTANT

Due to the different storage methods of single-node and multi-node partitioning, changing a deployment from single-node to multi-node results in the application losing all previously stored messages. This is also true if changing a deployment from multi-node to single-node, as the storage paths will not match.

Similarly, if a Rolling Strategy is implemented, the **maxSurge** parameter must be set to **0%**, otherwise the new broker creates a new partition and be unable to connect to the stored messages.

In multi-node partitioning, OpenShift routes new connections to the broker pod with the least amount of connections. Once this connection has been made, messages from that client are sent to the same broker every time, even if the client is run multiple times. This is because the OpenShift router is set to route requests from a client with the same IP to the same pod.

You can see which broker pod is connected to which split directory by viewing the logs for the pod, or by connecting to the broker console. In the **ActiveMQ** tab of the console, the **PersistenceAdapter** shows the **KahaDBPersistenceAdapter**, which includes the split directory as part of it's name.

4.2.7. Scaling Down and Message Migration

When A-MQ is deployed using a multi-node configuration it is possible for messages to be left in the kahadb store directory of a terminating pod should the cluster be scaled down. In order to prevent

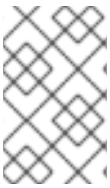
messages from remaining within the kahadb store of the terminating pod until the cluster next scales up, each A-MQ persistent template creates a second deployment containing a drainer pod which is responsible for managing the migration of messages. The drainer pod will scan each independent **split-*n*** directory within the A-MQ persistent volume, identify data stores associated with those pods which are terminating, and execute an application to migrate the remaining messages from those pods to other active members of the cluster.

IMPORTANT

Only messages sent through Message Queues will be migrated to other instances of the cluster when scaling down. Messages sent via topics will remain in storage until the cluster scales back up. Support for migrating Virtual Topics will be introduced in a future release.

4.2.8. Customizing A-MQ Configuration Files for Deployment

If using a template from an alternate repository, A-MQ configuration files such as **user.properties** can be included. When the image is downloaded for deployment, these files are copied to the ***<amq-home>*/amq/conf/** directory on the broker, which are committed to the container and pushed to the registry.



NOTE

If using this method, it is important that the placeholders in the configuration files (such as **##### AUTHENTICATION #####**) are not removed as these placeholders are necessary for building the A-MQ for OpenShift image.

4.2.9. Configuring Client Connections

Clients for the A-MQ for OpenShift image must specify the OpenShift router port (443) when setting the broker URL for SSL connections. Otherwise, A-MQ 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 there is a disruption on the router.

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

4.3. UPGRADING THE IMAGE REPOSITORY

On your master host(s), ensure you are logged into the CLI as a cluster administrator or user that has project administrator access to the global **openshift** project. For example:

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

Then, run the following command to update the core A-MQ OpenShift image stream in the **openshift** project:

```
$ oc -n openshift import-image jboss-amq-62
```

Depending on the deployment configuration, OpenShift deletes one of the broker pods and start a new upgraded pod. The new pod connects to the same persistent storage so that no messages are lost in the process. Once the upgraded pod is running, the process is repeated for the next pod until all of the

pods have been upgraded.

If a Rolling Strategy has been configured, OpenShift deletes and recreate pods based on the rolling update settings. Any new pod will only connect to the same persistent storage if the **maxSurge** parameter is set to **0%**, otherwise the new pod creates a new partition and will not be able to connect to the stored messages in the previous partition.

4.4. BINARY BUILDS

To deploy existing applications on OpenShift, you can use the binary source capability.

The following example uses the helloworld-mdb quickstart to deploy an A-MQ 6.2-based broker together with a JBoss EAP 6.4 messaging application, using JMS 1.1.

4.4.1. Prerequisite

1. Create a new project:

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

2. Create a service account to be used for A-MQ broker deployment:

```
$ oc create serviceaccount eap-service-account
serviceaccount "eap-service-account" created
```

3. Grant the **view** role to the service account. This enables the service account to view all resources in the namespace, which is necessary for managing the cluster.

```
$ oc policy add-role-to-user view system:serviceaccount:$(oc project
-q):eap-service-account
role "view" added: "system:serviceaccount:amq-bin-demo:eap-service-
account"
```

4.4.2. Deploy the A-MQ 6.2 Broker

1. Identify the image stream for the A-MQ broker:

```
$ oc get is -n openshift | grep amq | cut -d ' ' -f 1
jboss-amq-62
```

2. Deploy the broker, specifying the following:

   a. Application name and image stream,

   b. User name and password for standard broker user,

   c. A-MQ protocols to configure,

   d. Names of the A-MQ queues and topics (separated by commas),

   e. The discovery agent type to use for discovering mesh endpoints,

   f. Name of the service used for mesh creation,

g. The namespace in which the service resides, and

h. The A-MQ storage usage limit.

```
$ oc new-app --name=eap-app-amq \
--image-stream=jboss-amq-62 \
-e AMQ_USER=admin \
-e AMQ_PASSWORD=admin \
-e AMQ_TRANSPORTS=openwire \
-e AMQ_QUEUES=HELLOWORLDMDBQueue \
-e AMQ_TOPICS=HELLOWORLDMDBTopic \
-e AMQ_MESH_DISCOVERY_TYPE=kube \
-e AMQ_MESH_SERVICE_NAME=eap-app-amq \
-e AMQ_MESH_SERVICE_NAMESPACE=$(oc project -q) \
-e AMQ_STORAGE_USAGE_LIMIT="100 gb"
--> Found image 884d69b (4 months old) in image stream
"openshift/jboss-amq-62" under tag "latest" for "jboss-amq-62"

    JBoss A-MQ 6.2
    --------------
    A reliable messaging platform that supports standard
messaging paradigms for a real-time enterprise.

    Tags: messaging, amq, java, jboss, xpaas

    * This image will be deployed in deployment config "eap-app-
amq"
    * Ports 1883/tcp, 5672/tcp, 61613/tcp, 61616/tcp, 8778/tcp
will be load balanced by service "eap-app-amq"
      * Other containers can access this service through the
hostname "eap-app-amq"

--> Creating resources ...
    deploymentconfig "eap-app-amq" created
    service "eap-app-amq" created
--> Success
    Run 'oc status' to view your app.
```

3. Modify the **eap-app-amq** deployment config to run the pods under the **eap-service-account** service account created above:

```
$ oc patch dc/eap-app-amq --type=json \
-p '[{"op": "add", "path": "/spec/template/spec/serviceAccountName",
"value": "eap-service-account"}]'
"eap-app-amq" patched
```

4.4.3. Deploy Binary Build of EAP 6.4 Messaging Application

1. Clone the source code:

```
$ git clone -b 6.4.x https://github.com/jboss-developer/jboss-eap-
quickstarts.git
```

2. Configure the Red Hat JBoss Middleware Maven repository .

3. Build the **helloworld-mdb** application.

```
$ cd jboss-eap-quickstarts/helloworld-mdb
```

```
$ mvn clean package
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------
--------------
[INFO] Building JBoss EAP Quickstart: helloworld-mdb 6.4.0-SNAPSHOT
[INFO] ------------------------------------------------------------
--------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ jboss-
helloworld-mdb ---
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources)
@ jboss-helloworld-mdb ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /tmp/github/jboss-eap-
quickstarts/helloworld-mdb/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @
jboss-helloworld-mdb ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 3 source files to /tmp/github/jboss-eap-
quickstarts/helloworld-mdb/target/classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-
testResources) @ jboss-helloworld-mdb ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /tmp/github/jboss-eap-
quickstarts/helloworld-mdb/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-
testCompile) @ jboss-helloworld-mdb ---
[INFO] No sources to compile
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ jboss-
helloworld-mdb ---
[INFO] No tests to run.
[INFO]
[INFO] --- maven-war-plugin:2.1.1:war (default-war) @ jboss-
helloworld-mdb ---
[INFO] Packaging webapp
[INFO] Assembling webapp [jboss-helloworld-mdb] in
[/tmp/github/jboss-eap-quickstarts/helloworld-mdb/target/jboss-
helloworld-mdb]
[INFO] Processing war project
[INFO] Copying webapp resources [/tmp/github/jboss-eap-
quickstarts/helloworld-mdb/src/main/webapp]
[INFO] Webapp assembled in [22 msecs]
[INFO] Building war: /tmp/github/jboss-eap-quickstarts/helloworld-
mdb/target/jboss-helloworld-mdb.war
[INFO] ------------------------------------------------------------
--------------
```

```
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
-------------
[INFO] Total time: 1.536 s
[INFO] Finished at: 2017-05-26T16:50:49+02:00
[INFO] Final Memory: 17M/284M
[INFO] ------------------------------------------------------------
-------------
```

4. Prepare the directory structure on the local file system.
   Application archives in the **deployments/** subdirectory of the main binary build directory are copied directly to the standard deployments folder of the image being built on OpenShift. For the application to deploy, the directory hierarchy containing the web application data must be correctly structured.

   Create main directory for the binary build on the local file system and **deployments/** subdirectory within it. Copy the previously built WAR archive for the **helloworld-mdb** quickstart to the **deployments/** subdirectory:

```
$ ls
pom.xml  README.html  README.md  src  target
```

```
$ mkdir -p amq-binary-demo/deployments
```

```
$ cp target/jboss-helloworld-mdb.war amq-binary-demo/deployments/
```

NOTE

Location of the standard deployments directory depends on the underlying base image, that was used to deploy the application. See the following table:

Table 4.1. Standard Location of the Deployments Directory

Name of the Underlying Base Image(s)	Standard Location of the Deployments Directory
EAP for OpenShift 6.4 and 7.0	***$JBOSS_HOME/standalone/deployments***
Java S2I for OpenShift	***/deployments***
JWS for OpenShift	***$JWS_HOME/webapps***

5. Identify the image stream for the EAP 6.4 image:

```
$ oc get is -n openshift | grep eap64 | cut -d ' ' -f 1
jboss-eap64-openshift
```

6. Create a new binary build, specifying the image stream and the application name:

```
$ oc new-build --binary=true \
```

```
--image-stream=jboss-eap64-openshift \
--name=eap-app
--> Found image 8fbf0f7 (2 months old) in image stream
"openshift/jboss-eap64-openshift" under tag "latest" for "jboss-
eap64-openshift"

    JBoss EAP 6.4
    -------------
    Platform for building and running JavaEE applications on JBoss
EAP 6.4

    Tags: builder, javaee, eap, eap6

    * A source build using binary input will be created
      * The resulting image will be pushed to image stream "eap-
app:latest"
      * A binary build was created, use 'start-build --from-dir' to
trigger a new build

--> Creating resources with label build=eap-app ...
    imagestream "eap-app" created
    buildconfig "eap-app" created
--> Success
```

7. Start the binary build. Instruct the **oc** executable to use the main directory created in a previous step as the directory containing the binary input for the OpenShift build:

```
$ oc start-build eap-app --from-dir=amq-binary-demo/ --follow
Uploading directory "amq-binary-demo" as binary input for the build
...
build "eap-app-1" started
Receiving source from STDIN as archive ...
Copying all war artifacts from /home/jboss/source/. directory into
/opt/eap/standalone/deployments for later deployment...
Copying all ear artifacts from /home/jboss/source/. directory into
/opt/eap/standalone/deployments for later deployment...
Copying all rar artifacts from /home/jboss/source/. directory into
/opt/eap/standalone/deployments for later deployment...
Copying all jar artifacts from /home/jboss/source/. directory into
/opt/eap/standalone/deployments for later deployment...
Copying all war artifacts from /home/jboss/source/deployments
directory into /opt/eap/standalone/deployments for later
deployment...
'/home/jboss/source/deployments/jboss-helloworld-mdb.war' ->
'/opt/eap/standalone/deployments/jboss-helloworld-mdb.war'
Copying all ear artifacts from /home/jboss/source/deployments
directory into /opt/eap/standalone/deployments for later
deployment...
Copying all rar artifacts from /home/jboss/source/deployments
directory into /opt/eap/standalone/deployments for later
deployment...
Copying all jar artifacts from /home/jboss/source/deployments
directory into /opt/eap/standalone/deployments for later
deployment...
Pushing image 172.30.82.129:5000/amq-bin-demo/eap-app:latest ...
Pushed 0/7 layers, 6% complete
```

```
Pushed 1/7 layers, 14% complete
Pushed 2/7 layers, 29% complete
Pushed 3/7 layers, 45% complete
Pushed 4/7 layers, 73% complete
Pushed 5/7 layers, 84% complete
Pushed 6/7 layers, 96% complete
Pushed 7/7 layers, 100% complete
Push successful
```

8. Create a new OpenShift application based on the build, specifying the following:

   a. Application name,

   b. A-MQ service prefix mapping,

   c. JNDI name for connection factory used by applications to connect to the A-MQ broker,

   d. User name and password for standard broker user,

   e. A-MQ protocols to configure, and

   f. Names of the A-MQ queues and topics (separated by commas).

```
$ oc new-app eap-app \
-e MQ_SERVICE_PREFIX_MAPPING="eap-app-amq=MQ" \
-e MQ_JNDI=java:/ConnectionFactory \
-e MQ_USERNAME=admin \
-e MQ_PASSWORD=admin \
-e MQ_PROTOCOL=tcp \
-e MQ_QUEUES=HELLOWORLDMDBQueue \
-e MQ_TOPICS=HELLOWORLDMDBTopic
--> Found image 490167e (5 minutes old) in image stream "amq-bin-
demo/eap-app" under tag "latest" for "eap-app"

    amq-bin-demo/eap-app-1:fbe4b2ea
    -------------------------------
    Platform for building and running JavaEE applications on
JBoss EAP 6.4

    Tags: builder, javaee, eap, eap6

    * This image will be deployed in deployment config "eap-app"
    * Ports 8080/tcp, 8443/tcp, 8778/tcp will be load balanced by
service "eap-app"
      * Other containers can access this service through the
hostname "eap-app"

--> Creating resources ...
    deploymentconfig "eap-app" created
    service "eap-app" created
--> Success
    Run 'oc status' to view your app.
```

9. Expose the service as a route:

```
$ oc get svc -o name
service/eap-app
service/eap-app-amq
```

```
$ oc get route
No resources found.
```

```
$ oc expose svc/eap-app
route "eap-app" exposed
```

```
$ oc get route
NAME      HOST/PORT                                        PATH
SERVICES   PORT       TERMINATION   WILDCARD
eap-app   eap-app-amq-bin-demo.openshift.example.com
eap-app    8080-tcp                 None
```

10. Access the application.
    Access the EAP 6.4 messaging application in your browser using the URL **http://eap-app-amq-bin-demo.openshift.example.com/jboss-helloworld-mdb/**.

11. Check the log of the EAP 6.4 pod to see the result of message processing.

```
$ oc get pods
NAME                  READY     STATUS      RESTARTS   AGE
eap-app-1-build       0/1       Completed   0          15m
eap-app-1-f8w3r       1/1       Running     0          9m
eap-app-amq-2-8q1r6   1/1       Running     0          2h
```

```
$ oc logs eap-app-1-f8w3r | grep 'Received Message'
17:18:48,370 INFO  [class
org.jboss.as.quickstarts.mdb.HelloWorldQueueMDB] (Thread-7 (HornetQ-
client-global-threads-2098309660)) Received Message from queue: This
is message 4
17:18:48,369 INFO  [class
org.jboss.as.quickstarts.mdb.HelloWorldQueueMDB] (Thread-5 (HornetQ-
client-global-threads-2098309660)) Received Message from queue: This
is message 2
17:18:48,376 INFO  [class
org.jboss.as.quickstarts.mdb.HelloWorldQueueMDB] (Thread-4 (HornetQ-
client-global-threads-2098309660)) Received Message from queue: This
is message 1
17:18:48,379 INFO  [class
org.jboss.as.quickstarts.mdb.HelloWorldQueueMDB] (Thread-6 (HornetQ-
client-global-threads-2098309660)) Received Message from queue: This
is message 3
17:18:48,388 INFO  [class
org.jboss.as.quickstarts.mdb.HelloWorldQueueMDB] (Thread-9 (HornetQ-
client-global-threads-2098309660)) Received Message from queue: This
is message 5
```

CHAPTER 5. TUTORIALS

5.1. EXAMPLE DEPLOYMENT WORKFLOW

This tutorial prepares and deploys a multi-node A-MQ instance with persistent storage.

5.1.1. Preparing A-MQ Deployment

1. Create a new project:

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

2. Create a service account to be used for the A-MQ deployment:

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

3. Add the view role to the service account. This enables the service account to view all the resources in the **amq-demo** namespace, which is necessary for managing the cluster when using the Kubernetes REST API agent for discovering the mesh endpoints.

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

4. Edit the deployment configuration to run the AMQ pod with newly created service account.

```
$ oc edit dc/<deployment_config>
```

Add the serviceAccount and serviceAccountName parameters to the **spec** field, and specify the service account you want to use.

```
spec:
      securityContext: {}
      serviceAccount: serviceaccount
      serviceAccountName: amq-service-account
```

5. A-MQ requires a broker keyStore, a client keyStore, and a client trustStore that includes the broker keyStore.
This example uses 'keytool', a package included with the Java Development Kit, to generate dummy credentials for use with the A-MQ 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 trust store that imports the broker certificate:

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

6. Use the broker keyStore file to create the A-MQ secret:

```
$ oc secrets new amq-app-secret broker.ks
```

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

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

5.1.2. Deployment

1. Log in to the OpenShift web console and select the **amq-demo** project space.
2. Click **Add to Project** to list all of the default image streams and templates.
3. Use the Filter by keyword search bar to limit the list to those that match amq. You may need to click **See all** to show the desired application template.
4. Select the template. This example uses the ***amq62-persistent-ssl*** template to allow for persistent storage.

Example Template:
APPLICATION_NAME
broker
MQ_PROTOCOL
openwire
MQ_USERNAME
amq-demo-user
MQ_PASSWORD
password
VOLUME_CAPACITY
512Mi
AMQ_SECRET
amq-app-secret
AMQ_TRUSTSTORE
broker.ks
AMQ_TRUSTSTORE_PASSWORD
password
AMQ_KEYSTORE
broker.ks
AMQ_KEYSTORE_PASSWORD
password
AMQ_MESH_DISCOVERY_TYPE
kube
AMQ_MESH_SERVICE_NAME
broker
AMQ_MESH_SERVICE_NAMESPACE

amq-demo
AMQ_STORAGE_USAGE_LIMIT
1 gb
AMQ_SPLIT
true
IMAGE_STREAM_NAMESPACE
openshift

5.1.3. Post-Deployment

Creating a route
Create a route for the broker so that clients outside of OpenShift can connect using SSL. By default, the OpenWire protocol uses the 61617/TCP port.

1. Click **Create a Route** and click **Show options for secured routes** to display all parameters.
2. Use the **Target Port** drop-down menu to select **61617/TCP**
3. Use the **TLS Termination** drop-down menu to select **Passthrough**. This will relay all communication to the A-MQ broker without the OpenShift router decrypting and resending it.
4. Clients can now connect to the broker by specifying the following in their configuration:

```
factory.setBrokerURL("failover://ssl://broker-amq-
demo.example.com:443");
```

Scaling up
Scale up by clicking the **Scale up** arrow in the ***amq-demo*** project **Overview** in the web console. Or, using the OpenShift command line:

```
$ oc scale dc amq-demo --replicas=3
```

Connecting to the A-MQ Console
To connect to the A-MQ console from the OpenShift web console, navigate to the broker pod and click the **Connect** button located in the **Template** information.

For OpenShift Container Platform, click the **Open Java Console** button. === Example How to Monitor A-MQ This tutorial demonstrates how to monitor A-MQ.

5.1.4. Prerequisite

Make sure you have created a project, service account, and added the view role to the service account for A-MQ deployment, as mentioned in the section Example Deployment Workflow.

5.1.5. Monitoring A-MQ

1. Go to your project:

```
$ oc project monitoramq
```

2. Deploy a new broker instance to the **monitoramq** project, using the **amq62-basic** template from the **openshift** namespace:

```
$ oc process openshift//amq62-basic -v
APPLICATION_NAME=broker,MQ_USERNAME=admin,MQ_PASSWORD=admin,MQ_QUEUE
S=TESTQUEUE -n monitoramq | oc create -f  -

services "broker-amq-amqp" created
services "broker-amq-mqtt" created
services "broker-amq-stomp" created
services "broker-amq-tcp" created
deploymentconfigs "broker-amq" created
```

3. Get the list of running pods:

```
$ oc get pods

NAME                 READY     STATUS    RESTARTS   AGE
broker-amq-1-ftqmk   1/1       Running   0          14d
```

4. Run the command **oc logs**:

```
oc logs -f broker-amq-1-ftqmk

Running jboss-amq-6/amq62-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.
```

5. Run your query to monitor your broker for ServiceHealth:

```
$ curl -k -H "Authorization: Bearer $(oc whoami -t)"
https://10.1.2.2:8443/api/v1/namespaces/monitoramq/pods/https:broker
-amq-1-
ftqmk:8778/proxy/jolokia/read/org.apache.activemq:type=Broker,broker
Name=*,service=Health/CurrentStatus

{"request":
```

```
{"mbean":"org.apache.activemq:brokerName=*,service=Health,type=Broke
r","attribute":"CurrentStatus","type":"read"},"value":
{"org.apache.activemq:brokerName=broker-amq-1-
ftqmk,service=Health,type=Broker":
{"CurrentStatus":"Good"}},"timestamp":1491451600,"status":200}
```

where,

- **10.1.2.2** is the master host.
- **monitoramq** is the name of your namespace or project.
- **broker-amq-1-ftqmk** is the name of your running pod (got from **`oc get pods`** in previous step-4).
- **8778** Jolokia exposed port for AMQ xPaaS image.

6. Run your query to monitor your broker's memoryLimit for a queue:

```
$ curl -k -H "Authorization: Bearer $(oc whoami -t)"
https://10.1.2.2:8443/api/v1/namespaces/monitoramqlatest/pods/https:
broker-amq-1-
ftqmk:8778/proxy/jolokia/read/org.apache.activemq:type=Broker,broker
Name=*,destinationType=Queue,destinationName=ABEL/MemoryLimit

{"request":
{"mbean":"org.apache.activemq:brokerName=*,destinationName=ABEL,dest
inationType=Queue,type=Broker","attribute":"MemoryLimit","type":"rea
d"},"value":{"org.apache.activemq:brokerName=broker-amq-1-
ftqmk,destinationName=ABEL,destinationType=Queue,type=Broker":
{"MemoryLimit":1048576}},"timestamp":1491451792,"status":200}
```

7. Go to your Openshift Web Console, select your project, go to the running pod, and click **`Open Java Console`**.
8. You will be redirected to Openshift Container Platform Console, where you can see the listed queues under your broker.

CHAPTER 6. REFERENCE

6.1. APPLICATION TEMPLATE PARAMETERS FOR PERSISTENT CONFIGURATION

Configuration of the A-MQ for OpenShift image is performed by specifying values of application template parameters. Different A-MQ images require different subsets of these parameters. The following parameters can be configured:

AMQ_ADMIN_PASSWORD

The password used for authentication to the broker. If no value is specified, a random password is generated.

AMQ_ADMIN_USERNAME

The user name used as an admin authentication to the broker. If no value is specified, a random user name is generated.

AMQ_KEYSTORE

The SSL keyStore filename. If no value is specified, a random password is generated but SSL will not be configured.

AMQ_KEYSTORE_PASSWORD

The password used to decrypt the SSL keyStore (optional).

AMQ_MESH_DISCOVERY_TYPE

The discovery agent type to use for discovering mesh endpoints. 'dns' will use the OpenShift DNS service to resolve endpoints. 'kube' will use Kubernetes REST API to resolve service endpoints. If using 'kube' the service account for the pod must have the 'view' role.

AMQ_MESH_SERVICE_NAME

Name of service used for mesh creation.

AMQ_MESH_SERVICE_NAMESPACE

The namespace in which the service resides. Must be specified if using **kube** discovery.

AMQ_QUEUE_MEMORY_LIMIT

Specifies the memory limit set on a destination (limits the amount of memory on each queue listed in the **MQ_QUEUES** environment variable). The value of **AMQ_QUEUE_MEMORY_LIMIT** can be a string, such as **10 MB** or **512 KB**.

AMQ_RELEASE

The A-MQ release version. This determines which A-MQ image will be used as a basis for the application.

AMQ_SECRET

The name of a secret containing SSL related files. If no value is specified, a random password is generated.

AMQ_SPLIT

Boolean. Setting to ‘true’ partitions the persistent volume, allowing for multiple A-MQ pods for scalability.

AMQ_STORAGE_USAGE_LIMIT

The A-MQ storage usage limit.

AMQ_TRUSTSTORE

The SSL trustStore filename. If no value is specified, a random password is generated but SSL will not be configured.

AMQ_TRUSTSTORE_PASSWORD

The password used to decrypt the SSL trustStore (optional).

APPLICATION_NAME

The name of the application used internally in OpenShift. It is used in names of services, pods, and other objects within the application.

MQ_PASSWORD

The password used for authentication to the broker. In a standard non-containerized JBoss A-MQ, you would specify the password in the ***<amq-home>*/opt/user.properties** file. If no value is specified, a random password is generated.

MQ_PROTOCOL

Comma-separated list of the messaging protocols used by the broker. Available options are amqp, mqtt, openwire, and stomp. If left empty, all available protocols will be available. Please 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.

MQ_QUEUES

Comma-separated list of queues available by default on the broker on its startup.

MQ_TOPICS

Comma-separated list of topics available by default on the broker on its startup.

MQ_USERNAME

The user name used for authentication to the broker. In a standard non-containerized JBoss A-MQ, you would specify the user name in the ***<amq-home>*/opt/user.properties** file. If no value is specified, a random user name is generated.

VOLUME_CAPACITY

The size of the persistent storage for database volumes.

6.2. CONFIGURATION USING S2I

You can modify the configuration of the A-MQ for OpenShift image using the Source-to-image feature, described in full detail at S2I Requirements.

To specify a custom A-MQ broker configuration, create a new folder called **configuration** inside the root directory of your project. Then specify your custom A-MQ configuration by creating an **openshift-activemq.xml** file inside the 'configuration' folder. On each commit, this file gets copied to the **conf** directory in the A-MQ root and its contents are used to configure the broker.

NOTE

Please be careful when making a changes to the **openshift-activemq.xml** file and do not remove the placeholders, as automated configuration may not work properly.

6.3. SECURITY

Only SSL connections can connect from outside of the OpenShift instance, regardless of the protocol specified in the **MQ_PROTOCOL** property of the A-MQ application templates. 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 using the OpenShift secrets mechanism or S2I.

6.4. LOGGING

In addition to viewing the OpenShift logs, you can troubleshoot a running A-MQ image by viewing the A-MQ logs that are outputted to the container's console:

```
$ oc logs -f <pod-name> <container-name>
```

NOTE

By default, the A-MQ for OpenShift image does not have a file log handler configured. Logs are only sent to the console.