Reference Architectures 2017

Clustering, Fault-Tolerance, and Messaging Patterns with Red Hat JBoss AMQ 7

Last Updated: 2018-08-10

Reference Architectures 2017 Clustering, Fault-Tolerance, and Messaging Patterns with Red Hat JBoss AMQ 7

Jeremy Ary
refarch-feedback@redhat.com

Legal Notice

Abstract

This reference architecture demonstrates basic messaging patterns with single-broker, clustered, and fault-tolerant Red Hat JBoss AMQ 7 topologies, as well as direct, closest, balanced and multicast Interconnect routing configurations.

Table of Contents

COMMENTS AND FEEDBACK

In the spirit of open source, we invite anyone to provide feedback and comments on any reference architecture. Although we review our papers internally, sometimes issues or typographical errors are encountered. Feedback allows us to not only improve the quality of the papers we produce, but allows the reader to provide their thoughts on potential improvements and topic expansion to the papers. Feedback on the papers can be provided by emailing refarch-feedback@redhat.com. Please refer to the title within the email.

CHAPTER 1. EXECUTIVE SUMMARY

This reference architecture demonstrates basic messaging patterns with single-broker, clustered, and fault-tolerant **Red Hat JBoss AMQ 7** topologies, as well as direct, closest, balanced and multicast **Interconnect** routing configurations.

The reference architecture paper documents the steps undertaken to utilize some of the prominent capabilities provided by AMQ 7, while the accompanying code, configurations, and S2I image for **OpenShift Container Platform** allows the reader to fully understand each step and replicate them at will.

WARNING

The AMQ/OpenShift configuration described and/or provided herein uses community software and thus is not an officially supported Red Hat configuration at the time of writing.

CHAPTER 2. RED HAT JBOSS AMQ 7

2.1. OVERVIEW

Based on the upstream **Apache ActiveMQ** and **Apache Qpid** community projects, **Red Hat JBoss AMQ 7** is a lightweight, standards-based open source messaging platform designed to enable real-time communication between different applications, services, devices, and Internet of Things (IoT) devices. It also serves as the messaging foundation for **Red Hat JBoss Fuse**, Red Hat's lightweight, flexible integration platform, and is designed to provide the real-time, distributed messaging capabilities needed to support an agile integration approach for modern application development.

AMQ 7 introduces technology enhancements across three core components: the broker, clients, and Interconnect router.

2.2. BROKER

The AMQ 7 broker, based on **Apache ActiveMQ Artemis**, manages addresses, queues, and routing semantics. The new broker has an asynchronous internal architecture which can increase performance and scalability, while enabling it to handle more concurrent connections and achieve greater message throughput. AMQ Broker is a full-featured, message-oriented middleware broker. It offers specialized queueing behaviors, message persistence, and manageability. Core messaging is provided with support for different messaging patterns such as publish-subscribe, point-to-point, and store-and-forward. AMQ 7 supports multiple protocols and client languages, allowing integration of many, if not all, application assets.

2.3. CLIENTS

Red Hat JBoss AMQ 7 expands its support of popular messaging APIs and protocols by adding new client libraries, including Java Message Service (JMS) 2.0, JavaScript, C++, .Net, and Python. With existing support for the popular open protocols MQTT and AMQP, AMQ 7 now offers broad interoperability across the IT landscape that can open up data in embedded devices to inspection, analysis, and control.

2.4. INTERCONNECT

The new **Interconnect** router in AMQ 7 enables users to create an internet-scale network of uniformly-addressed messaging paths spanning data centers, cloud services, and geographic zones. The **Interconnect** component serves as the backbone for distributed messaging, providing redundant network pathing for fault handling, traffic optimization, and more secure and reliable connectivity.

CHAPTER 3. REFERENCE ARCHITECTURE ENVIRONMENT

3.1. OVERVIEW

This reference architecture demonstrates a multi-service OpenShift Container Platform project housing various Red Hat JBoss AMQ 7 topologies. The provided environment builds on OpenShift, so the included test suite also assumes an OpenShift environment, but installation and configuration steps for any Red Hat Enterprise Linux server and/or cluster are also provided in order to establish a similar environment.

WARNING

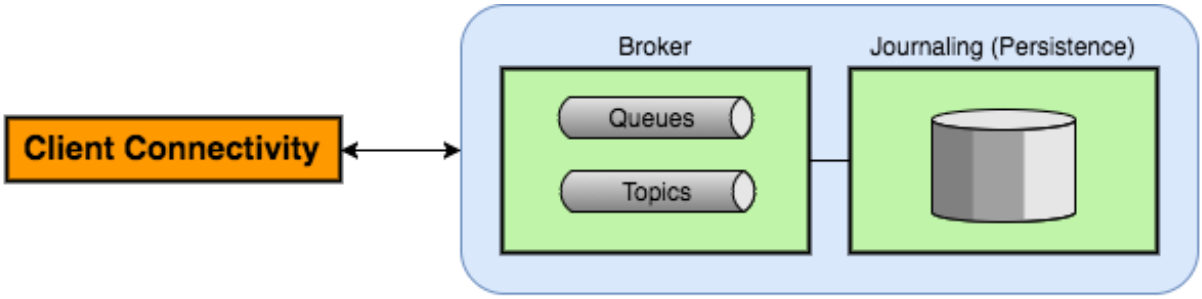
The AMQ/OpenShift configuration described and/or provided herein uses community software and thus is not an officially supported Red Hat configuration at the time of writing.

This reference architecture showcases 4 topology examples:

3.2. SINGLE-BROKER INSTANCE

This example features multiple JMS clients interacting with a single broker to demonstrate execution of basic send and receive messaging patterns.

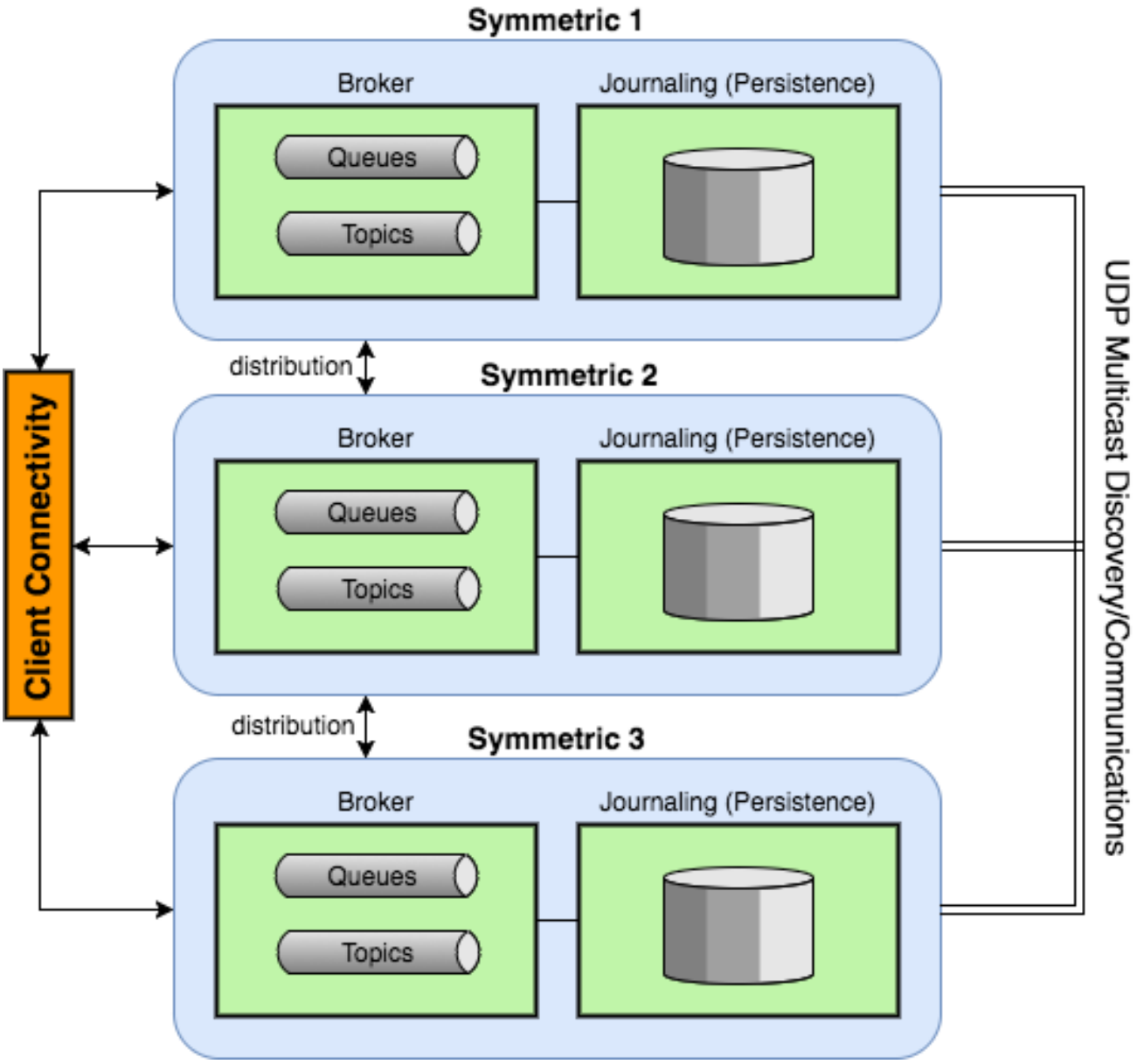
Figure 3.1. Single Broker



3.3. SYMMETRIC BROKER CLUSTER

This example features a cluster of 3 broker instances, all grouped together to allow for internal load balancing.

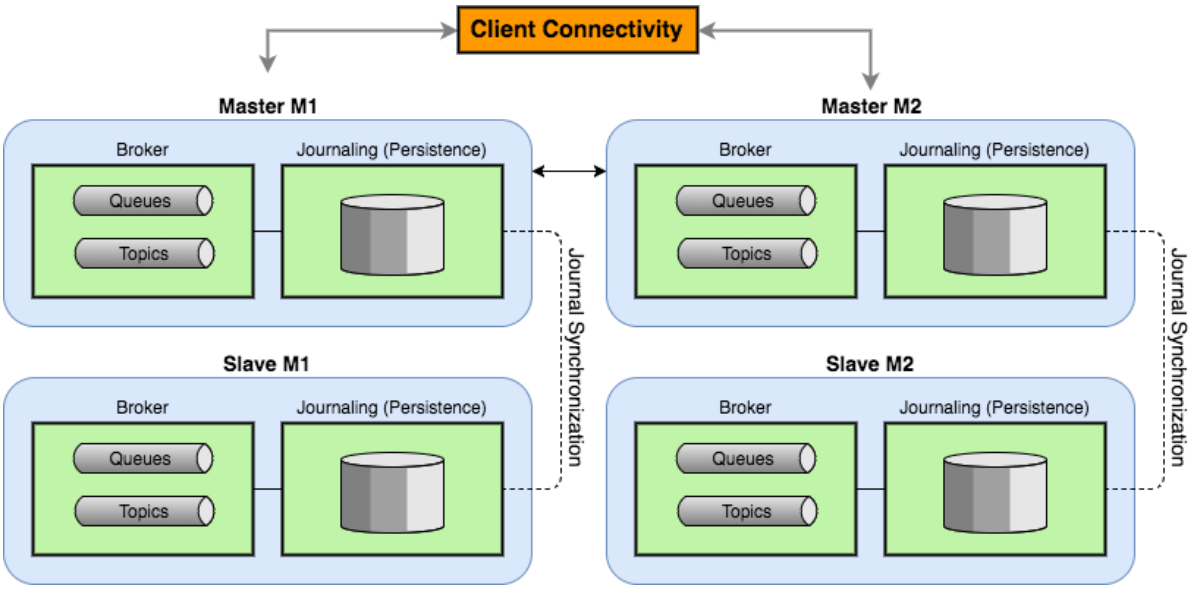
Figure 3.2. 3-Node Symmetric Cluster



3.4. FAULT-TOLERANT BROKER CLUSTER

This example features 3 master and 3 slave broker instances, all clustered within the same group, allowing automatic master/slave pairing and failover.

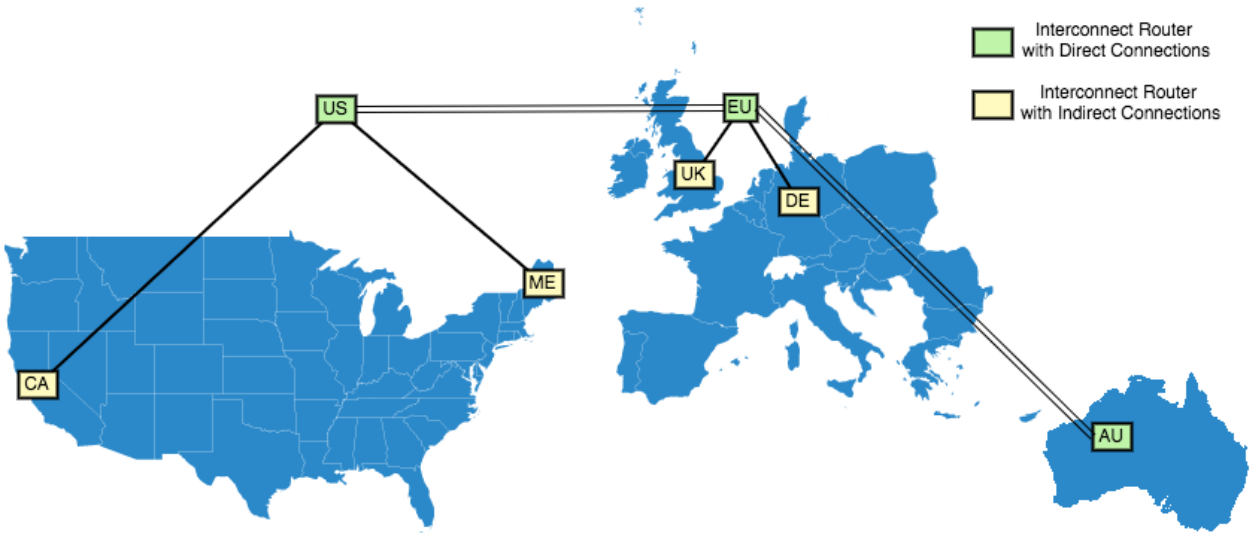
Figure 3.3. 3-Pair Master/Slave Cluster



3.5. MULTI-ROUTER INTERCONNECT NETWORK

This example features 7 nodes, simulating various geographic distributions in order to showcase the differences in various routing algorithm choices.

Figure 3.4. Interconnect Network



CHAPTER 4. CREATING THE ENVIRONMENT

4.1. OVERVIEW

This reference architecture provides an accompanying OpenShift S2I image and template set for the deployment of pre-configured services, that can be used as a target for the included test suite in its current form. The code may be found in a publicly-available repository.

```
$ git clone git@github.com:RHsyseng/amq7.git
```

While a pre-configured OpenShift Container Platform project is provided for convenience, the reader may also elect to utilize a single Red Hat Enterprise Linux server to recreate the single-broker, symmetric cluster, and replication topologies individually. A cluster of RHEL servers may also serve to replicate the Interconnect topology in lieu of OpenShift. If the reader's choice is to forego OpenShift Container Platform, you may skip ahead to Deployment on Red Hat Enterprise Linux at this time.

4.2. DEPLOYMENT ON OPENSHIFT CONTAINER PLATFORM

WARNING

The AMQ/OpenShift configuration described and/or provided herein uses community software and thus is not an officially supported Red Hat configuration at the time of writing.

4.2.1. Prerequisites

A pre-existing OpenShift Container Platform installation, with a configured user called **ocuser**, capable of project ownership, is assumed in the following steps. More information on installation and configuration of the Red Hat Enterprise Linux Operating System, OpenShift Container Platform, and more can be found in Section 3: **Creating the Environment** of a previous Reference Architecture, Building JBoss EAP 7 Microservices on OpenShift Container Platform .

The URL **cluster-ingress.example.com** is used throughout the provided templates to allow external access to the brokers and management consoles. Add the URL to your **/etc/hosts** file so that it resolves to the proper OpenShift Container Platform node for desired routing.

```
10.1.2.300    cluster-ingress.example.com #link to OpenShift-routable node
address
```

4.2.2. Create Project

Utilize remote or direct terminal access to log in to the OpenShift environment as the user who will create and have ownership of the new project:

```
$ oc login -u ocuser
```

Create the new project which will house the various builds, deployments and services:

```
$ oc new-project amq --display-name="AMQ 7 Reference Architecture Example"
--description="Showcase of various AMQ 7 features and topologies"
```

4.2.3. Base Image Template Deployment

Within the new project, execute the provided YAML template to configure and instantiate the base image stream and build config:

```
$ oc process -f
https://raw.githubusercontent.com/RHsyseng/amq7/master/S2I-Base-
Image/yaml_templates/amq_image_template.yaml | oc create -f -
```

4.2.4. Image Build

Kick off a build of the S2I base image stream and monitor for completion:

```
$ oc start-build amq7-image --follow
```

4.2.5. Topology Template Deployments

Once the initial build is complete, deploy the single-broker template:

```
$ oc process -f
https://raw.githubusercontent.com/RHsyseng/amq7/master/S2I-Base-
Image/yaml_templates/amq_single_template.yaml | oc create -f -
```

The remainder of the topology templates have been separated for convenience of cherry-picking, but in order to execute the full test suite, all three must be applied:

```
$ oc process -f
https://raw.githubusercontent.com/RHsyseng/amq7/master/S2I-Base-
Image/yaml_templates/amq_symmetric_template.yaml | oc create -f -
$ oc process -f
https://raw.githubusercontent.com/RHsyseng/amq7/master/S2I-Base-
Image/yaml_templates/amq_replicated_template.yaml | oc create -f -
$ oc process -f
https://raw.githubusercontent.com/RHsyseng/amq7/master/S2I-Base-
Image/yaml_templates/amq_interconnect_template.yaml | oc create -f -
```

NOTE

While any of the last three templates (symmetric, replication, and interconnect) can be skipped or deployed at-will, the single-broker service must be deployed for any of these to properly function, as the single-broker service provides ingress into the cluster for the various ports utilized by the remaining templates.

4.3. DEPLOYMENT ON RED HAT ENTERPRISE LINUX

4.3.1. Prerequisites

To install AMQ 7 to a Linux platform, you must first download the installation archive from the Red Hat Customer Portal.

4.3.2. Single Broker Configuration

The broker utilizes a few ports for inbound connectivity, including TCP ports **61616**, **5672**, **61613**, and **1883** for the various message protocols and TCP port **8161** for access to the HTTP AMQ Console. If the server is running a firewall, these ports must be allowed connectivity prior to beginning installation and configuration. More information on inspecting and configuring the firewall for Red Hat Enterprise Linux can be found here.

1) Create a new user named **amq-broker** and provide a password.

```
$ sudo useradd amq-broker
$ sudo passwd amq-broker
```

2) Create the directory **/opt/redhat/amq-broker** and make the new **amq-broker** user and group the owners.

```
$ sudo mkdir -p /opt/redhat/amq-broker
$ sudo chown -R amq-broker:amq-broker /opt/redhat/amq-broker
```

3) Change the owner of the archive to the new user.

```
$ sudo chown amq-broker:amq-broker amq-broker-7.x.x.zip
```

4) Move the installation archive to the directory you just created.

```
$ sudo mv amq-broker-7.x.x.zip /opt/redhat/amq-broker
```

5) As the new user **amq-broker**, extract the contents with a single unzip command.

```
$ su - amq-broker
$ cd /opt/redhat/amq-broker
$ unzip amq-broker-7.x.x.zip
```

6) A directory named something similar to **amq-broker-7.x.x** will be created, further referred to herein as the **INSTALL_DIR**. Next, create a directory location for the broker instance and assign the user you created during installation as its owner.

```
$ sudo mkdir /var/opt/amq-broker
$ sudo chown -R amq-broker:amq-broker /var/opt/amq-broker
```

7) Navigate to the new directory and use the **artemis create** command to create a broker. Note in the example below, the user that was created during installation is the one to run the create command.

```
$ su - amq-broker
$ cd /var/opt/amq-broker
$ /opt/redhat/amq-broker/bin/artemis create /var/opt/amq-broker/mybroker -
-user amq-broker --password password --allow-anonymous
```

NOTE

Use the **`artemis help create`** command to view a list/descriptions of available parameters for instance configuration.

8) If desired, view contents of the **`BROKER_INSTANCE_DIR/etc/broker.xml`** configuration file. A simple, single-broker localhost configuration has been already been configured by the **`artemis`** tool.

9) Next, use the **`artemis`** script in the **`BROKER_INSTANCE_DIR/bin`** directory to start the broker.

```
$ su - amq-broker
$ /var/opt/amq-broker/mybroker/bin/artemis run
```

NOTE

Use the **`artemis help run`** command to view a list/descriptions of available parameters for runtime configuration.

4.3.3. Symmetric Cluster Configuration

Typical production environments utilize multiple servers for clustering, however, for demonstration purposes, the following cluster is set up to reside on a single server utilizing unique, individual ports for inbound connections and a separate, shared UDP port for group connectivity.

4.3.3.1. Single-Server Cluster

To prepare multiple brokers on a single server, complete the follow steps.

1) As with the single-broker example, complete steps 1-6 to establish a user and prepare the AMQ artemis environment.

2) Next, create directories for each broker instance of the cluster:

```
$ mkdir /var/opt/amq-broker/run && cd "$_"
$ mkdir artemis_1 artemis_2 artemis_3
```

3) Similar to before, the **`artemis create`** command will be used to populate each directory with a broker instance.

```
$ /opt/redhat/amq-broker/bin/artemis create artemis_1 \
    --user amq-broker --password password --allow-anonymous --clustered \
    --cluster-user amqUser --cluster-password password --host localhost \
    --name artemis-1 --port-offset 10*
$ /opt/redhat/amq-broker/bin/artemis create artemis_2 \
    --user amq-broker --password password --allow-anonymous --clustered \
    --cluster-user amqUser --cluster-password password --host localhost \
    --name artemis-2 --port-offset 20*
$ /opt/redhat/amq-broker/bin/artemis create artemis_3 \
    --user amq-broker --password password --allow-anonymous --clustered \
    --cluster-user amqUser --cluster-password password --host localhost \
    --name artemis-3 --port-offset 30*
```

4) Once each broker directory has been populated, start each broker as a service.

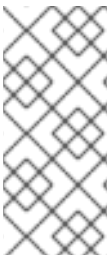
```
$ artemis_1/bin/artemis-service start
Starting artemis-service
artemis-service is now running (109195)

$ artemis_2/bin/artemis-service start
Starting artemis-service
artemis-service is now running (109307)

$ artemis_3/bin/artemis-service start
Starting artemis-service
artemis-service is now running (109432)
```

5) Examination of the logs reveals information relevant to the connectivity established between each of the servers.

```
$ cat artemis_3/log/artemis.log
...
AMQ221001: [truncated for brevity]...[artemis_3, nodeID=83c078d5-ad4f-
11e7-a3e2-000c2906f4ae]
AMQ221027: Bridge ClusterConnectionBridge@7a5f143
[name=$.artemis.internal.sf.my-cluster.77770624-ad4f-11e7-8acd-
000c2906f4ae
    [truncated for brevity]?port=61626&host=localhost],
discoveryGroupConfiguration=null]] is connected
AMQ221027: Bridge ClusterConnectionBridge@7241b488
[name=$.artemis.internal.sf.my-cluster.7d6ebb4d-ad4f-11e7-a379-
000c2906f4ae
    [truncated for brevity]?port=61636&host=localhost],
discoveryGroupConfiguration=null]] is connected
...
```

4.3.3.2. Multi-Server Cluster

To prepare multiple servers, each with its own broker, several ports must be utilized for intra-node and inbound communications. This includes **9876** for multicast UDP group connectivity, TCP via **61616**, **5672**, **61613**, and **1883** for the various message protocols, and TCP via 8161 for access to the HTTP AMQ Console. If the servers are running a firewall, these ports must be allowed connectivity prior to beginning installation and configuration. More information on inspecting and configuring the firewall for Red Hat Enterprise Linux can be found here.



NOTE

Ensure the network is properly configured for multicast and that multicast addresses have been verified during the configuration stage of the environment, otherwise HA functionality might not work as intended. If using Red Hat Enterprise Linux, also ensure that the http protocol has been enabled for viewing of AMQ Console, as RHEL does not allow HTTP traffic by default.

Each of the following steps should be performed on every server within the cluster.

1) Once port access has been configured, complete steps 1-6 to establish a user and prepare the AMQ artemis environment.

2) Next, create a directory for the broker instance:

```
$ mkdir /var/opt/amq-broker/run && cd "$_"
$ mkdir artemis_1
```

3) Similar to before, the **`artemis create`** command will be used to populate the directory with a broker instance.

```
$ /opt/redhat/amq-broker/bin/artemis create artemis_1 \
    --user amq-broker --password password --allow-anonymous --clustered \
    --cluster-user amqUser --cluster-password password --host 10.10.X.X \
    --name artemis-broker
```

4) Once the broker directory has been populated, start the broker as a service.

```
$ artemis_1/bin/artemis-service start
Starting artemis-service
artemis-service is now running (109195)
```

5) As with the single-server configuration above, cluster connectivity can be monitored by examining the logs of one of the servers.

```
$ tail -f artemis_1/log/artemis.log
```

4.3.4. Replication Cluster Configuration

Typical production environments utilize multiple servers for clustering, however, for demonstration purposes, the following cluster is set up to reside on a single server utilizing unique, individual ports for inbound connections and a separate, shared UDP or TCP (via JGroups) port for group connectivity.

4.3.4.1. Single-Server Cluster

To prepare multiple brokers on a single server, complete the follow steps.

1) As with the single-broker example, complete steps 1-6 to establish a user and prepare the AMQ artemis environment.

2) Next, create directories for each broker instance of the cluster:

```
$ mkdir /var/opt/amq-broker/run && cd "$_"
$ mkdir artemis_1 artemis_2 artemis_3 artemis_4 artemis_5 artemis_6
```

3) Similar to before, the **`artemis create`** command will be used to populate each directory with a broker instance.

```
$ /opt/redhat/amq-broker/bin/artemis create artemis_1 \
    --user amq-broker --password password --allow-anonymous --clustered \
    --cluster-user amqUser --cluster-password password --host localhost \
    --name artemis-1 --port-offset 10 --replicated

$ /opt/redhat/amq-broker/bin/artemis create artemis_2 \
    --user amq-broker --password password --allow-anonymous --clustered \
    --cluster-user amqUser --cluster-password password --host localhost \
    --name artemis-2 --port-offset 20 --replicated
```

```
$ /opt/redhat/amq-broker/bin/artemis create artemis_3 \
    --user amq-broker --password password --allow-anonymous --clustered \
    --cluster-user amqUser --cluster-password password --host localhost \
    --name artemis-3 --port-offset 30 --replicated

$ /opt/redhat/amq-broker/bin/artemis create artemis_4 \
    --user amq-broker --password password --allow-anonymous --clustered \
    --cluster-user amqUser --cluster-password password --host localhost \
    --name artemis-4 --port-offset 10 --replicated --slave

$ /opt/redhat/amq-broker/bin/artemis create artemis_5 \
    --user amq-broker --password password --allow-anonymous --clustered \
    --cluster-user amqUser --cluster-password password --host localhost \
    --name artemis-5 --port-offset 20 --replicated --slave

$ /opt/redhat/amq-broker/bin/artemis create artemis_6 \
    --user amq-broker --password password --allow-anonymous --clustered \
    --cluster-user amqUser --cluster-password password --host localhost \
    --name artemis-6 --port-offset 30 --replicated --slave
```

4) Once broker directories have been populated, start each broker as a service.

```
$ artemis_1/bin/artemis-service start
Starting artemis-service
artemis-service is now running (109195)

[repeat for artemis_2-6]
```

5) Examination of the logs reveals information relevant to group connectivity and replica information sent from master nodes to slave nodes.

```
$ cat artemis_3/log/artemis.log
...
AMQ221001: [truncated for brevity]...[artemis_2, nodeID=83c078d5-ad4f-
11e7-a3e2-000c2906f4ae]
AMQ221027: Bridge ClusterConnectionBridge@7a5f143
[name=$.artemis.internal.sf.my-cluster.77770624-ad4f-11e7-8acd-
000c2906f4ae
    [truncated for brevity]?port=61626&host=localhost],
discoveryGroupConfiguration=null]] is connected
AMQ221027: Bridge ClusterConnectionBridge@7241b488
[name=$.artemis.internal.sf.my-cluster.7d6ebb4d-ad4f-11e7-a379-
000c2906f4ae
    [truncated for brevity]?port=61646&host=localhost],
discoveryGroupConfiguration=null]] is connected
...
AMQ221025: Replication: sending
AIOSequentialFile:/var/opt/run/artemis_2/./data/journal/activemq-data-
2.amq (size=10,485,760) to replica.
AMQ221025: Replication: sending NIOSequentialFile
/var/opt/run/artemis_2/./data/bindings/activemq-bindings-4.bindings
(size=1,048,576) to replica.
AMQ221025: Replication: sending NIOSequentialFile
/var/opt/run/artemis_2/./data/bindings/activemq-bindings-2.bindings
(size=1,048,576) to replica.
```

4.3.4.1.1. Multi-Server Cluster

As with the multi-server symmetric cluster, once port access has been configured for all intended cluster members, the broker can be added to each server:

For **master** nodes, use the following **`artemis create`** command format, where **`N`** indicates the node number:

```
$ /opt/redhat/amq-broker/bin/artemis create artemis_N \
    --user amq-broker --password password --allow-anonymous --clustered \
    --cluster-user amqUser --cluster-password password --host localhost \
    --name artemis-N --replicated*
```

For **slave** nodes, use the following **`artemis create`** command format, where **`N`** indicates the node number:

```
$ /opt/redhat/amq-broker/bin/artemis create artemis_N \
    --user amq-broker --password password --allow-anonymous --clustered \
    --cluster-user amqUser --cluster-password password --host localhost \
    --name artemis-N --replicated --slave*
```

4.3.5. Interconnect Network Configuration

Given the nature of the Interconnect executable and the need to demonstrate various network proximity routing mechanisms, a multi-server configuration is recommended. However, the routers only utilize AMQP ports **5672**, **5673**, and **5674** for all communication needs, therefore multicast is not required and firewall rules are significantly simplified. Every server will need to expose the aforementioned three ports.

4.3.5.1. Installation

In order to install the Interconnect router on Red Hat Enterprise Linux, complete the following steps for each server intended to join the topology:

1) Ensure your subscription has been activated and your system is registered. For more information about using the customer portal to activate your Red Hat subscription and register your system for packages, refer to the documentation on *using your subscription*.

2) Subscribe to the required repositories:

```
$ sudo subscription-manager repos --enable=amq-interconnect-1-for-rhel-7-
server-rpms --enable=a-mq-clients-1-for-rhel-7-server-rpms
```

3) Use the **`yum`** command to install the **`qpid-dispatch-router`** and **`qpid-dispatch-tools`** packages and **`libaio`** dependency for **`ASYNCIO`** journaling:

```
$ sudo yum install libaio qpid-dispatch-router qpid-dispatch-tools
```

4) Use the **`which`** command to verify that the **`qdrouterd`** executable is present.

```
$ which qdrouterd
/usr/sbin/qdrouterd
```

The **qdrouterd** executable should be located at **/usr/sbin/qdrouterd**.

4.3.5.2. Configuration

The configuration file for the router is located at **etc/qpid-dispatch/qdrouterd.conf**. The following is a complete example configuration as used in the included OpenShift Container Platform example:

```
router { 1
    mode: interior
    id: US
}

listener { 2
    role: normal
    host: 0.0.0.0
    port: amqp
    authenticatePeer: no
    saslMechanisms: ANONYMOUS
}

listener { 3
    role: normal
    host: 0.0.0.0
    port: 5673
    http: yes
    authenticatePeer: no
    saslMechanisms: ANONYMOUS
}

listener { 4
    role: inter-router
    host: 0.0.0.0
    port: 5674
    authenticatePeer: no
    saslMechanisms: ANONYMOUS
}

connector { 5
    role: inter-router
    host: interconnect-eu.amq.svc.cluster.local
    port: 5674
    saslMechanisms: ANONYMOUS
}

address { 6
    prefix: multicast
    distribution: multicast
}

log { 7
    module: DEFAULT
    enable: debug+
    timestamp: yes
}
```

1. Basic router configuration. By default, a router operates in *standalone* mode, whereas this example indicates that the router is part of a network, or *interior* mode.
2. Listener for inbound AMQP client connections.
3. Listener for inbound AMQ Console connections allowing **HTTP**.
4. Listener for other router entity connections.
5. Connector request specifying another router entity that the router should attempt to connect with.
6. Address configuration indicating that any address beginning with the prefix **`multicast`** (such as a queue named should utilize the **`multicast`** routing pattern in delivery. Other options include **`closest`** and **`balanced`**.
7. Log module configuration: specifies that all Interconnect modules should log any debug or higher messages and include timestamp. Note that individual module configuration is possible.

Further information on Interconnect router configuration can be found in the Using AMQ Interconnect for Red Hat JBoss AMQ 7 Guide.

4.3.5.3. Execution

Once the router network has been configured as desired, start each with the following command:

```
$ systemctl start qdrouterd.service
```

4.3.5.4. Log Monitoring

Router status and log output can be viewed with the following command:

```
$ qdstat --log
```

CHAPTER 5. DESIGN AND DEVELOPMENT

5.1. OVERVIEW

The source code for the **Red Hat JBoss AMQ 7** example project is made available in a public github repository. This chapter briefly covers each topology, accompanying tests, and functionality. Note that in the example client tests, JMS is used exclusively for connectivity, however, AMQ 7 also offers clients for **C++**, **JavaScript**, **Python**, and **.NET**. More information on these clients can be found in their respective guides located on the AMQ 7 Documentation page.

5.2. PREREQUISITES

The following topology and routing explanations assume that the provided OpenShift Container Platform example environment has been utilized. If Red Hat Enterprise Linux instances have been used instead, the tests will require modification of server addresses and ports prior to usage.

WARNING

The AMQ/OpenShift configuration described and/or provided herein uses community software and thus is not an officially supported Red Hat configuration at the time of writing.

The included tests which showcase various features of each topology assume that **Maven 3.X** and **Java JDK 1.8+** are installed and configured. An example Maven **settings.xml** configuration file, allowing resolution of Red Hat's GA and Early Access repositories, is provided as an appendix.

The tests mentioned are part of the code available in the public github repository, under the **Test-Suite** directory. All **mvn** commands which follow are assumed to be ran from that directory.

5.3. COMMON BROKER CONFIGURATIONS

Various components within the **broker.xml** configuration file will be commonly seen across the different OpenShift Container Platform example scenarios. These sections are as follows:

5.3.1. Directory and Persistence Configuration

```
<persistence-enabled>true</persistence-enabled> (1)

<paging-directory>./data/paging</paging-directory> (2)
<bindings-directory>./data/bindings</bindings-directory>
<large-messages-directory>./data/large-messages</large-messages-directory>

<journal-type>ASYNCIO</journal-type> (3)
<journal-directory>./data/journal</journal-directory>
<journal-min-files>2</journal-min-files>
<journal-pool-files>-1</journal-pool-files>
<journal-buffer-timeout>59999</journal-buffer-timeout>
```

1. Enables the default highly performant option of writing messages to journals on the file system.
2. Configures multiple directories for journaling of paging, binding, and large messages.
3. Journaling configuration specifying the **ASYNCIO** type and various parameters.

5.3.2. Connectors and Acceptors

```
<connectors>
    <connector name="artemis">tcp://0.0.0.0:${ARTEMIS_PORT}</connector> (1)
</connectors>
<acceptors>
    <acceptor name="artemis">tcp://0.0.0.0:${ARTEMIS_PORT}</acceptor> (2)
</acceptors>
```

1. Configures server-to-server connectivity protocol via group discovery.
2. Configures a default endpoint for the broker listening for ActiveMQ Artemis, OpenWire, STOMP, AMQP, MQTT, and HornetQ protocol connections.

5.3.3. Security Configuration

```
<management-address>activemq.management</management-address> (1)
<management-notification-address> (2)
  jms.topic.notificationsTopic
</management-notification-address>

<security-settings>
    <security-setting match="#"> (3)
        <permission type="createNonDurableQueue" roles="admin"/>
        <permission type="deleteNonDurableQueue" roles="admin"/>
        <permission type="createDurableQueue" roles="admin"/>
        <permission type="deleteDurableQueue" roles="admin"/>
        <permission type="createAddress" roles="admin"/>
        <permission type="deleteAddress" roles="admin"/>
        <permission type="consume" roles="admin"/>
        <permission type="browse" roles="admin"/>
        <permission type="send" roles="admin"/>
        <permission type="manage" roles="admin"/>
    </security-setting>
    <security-setting match="jms.queue.activemq.management"> (4)
        <permission type="consume" roles="admin"/>
        <permission type="send" roles="admin"/>
        <permission type="manage" roles="admin"/>
    </security-setting>
</security-settings>
```

1. Indicates that a special, internal address should be enabled thus enabling AMQP clients to send Broker management commands to the server. A topic is also specified.
2. Specifies that authenticated clients may receive broker system notifications by subscribing to the specified topic.

3. Security settings for the wildcard matcher allowing the **admin** role for all functionalities.

4. Security settings used to authenticate clients wishing to utilize the **management-address** and **notification-address** mechanisms.

5.3.4. Queue and Topic Configuration

```
<addresses>
    <address name="test_topic"> 1
        <multicast/>
    </address>
    <address name="test_queue"> 2
        <anycast>
            <queue name="jms.queue.test_queue"/>
        </anycast>
    </address>
</addresses>
```

1. Configures an example multicast (publish-subscribe) topic on the broker.

2. Configures an example queue (point-to-point) on the broker.

5.3.5. Dead Letter and Expiry Queue Configuration

```
<address-settings>
    <address-setting match="activemq.management#"> 1
        <dead-letter-address>DLQ</dead-letter-address>
        <expiry-address>ExpiryQueue</expiry-address>
        <redelivery-delay>0</redelivery-delay>
        <max-size-bytes>-1</max-size-bytes>
        <message-counter-history-day-limit>10</message-counter-history-
day-limit>
        <address-full-policy>PAGE</address-full-policy>
        <auto-create-queues>true</auto-create-queues>
        <auto-create-addresses>true</auto-create-addresses>
        <auto-create-jms-queues>true</auto-create-jms-queues>
        <auto-create-jms-topics>true</auto-create-jms-topics>
    </address-setting>
    <address-setting match="#"> 2
        <dead-letter-address>DLQ</dead-letter-address>
        <expiry-address>ExpiryQueue</expiry-address>
        <redelivery-delay>0</redelivery-delay>
        <max-size-bytes>-1</max-size-bytes>
        <message-counter-history-day-limit>10</message-counter-history-
day-limit>
        <address-full-policy>PAGE</address-full-policy>
        <auto-create-queues>true</auto-create-queues>
        <auto-create-addresses>true</auto-create-addresses>
        <auto-create-jms-queues>true</auto-create-jms-queues>
        <auto-create-jms-topics>true</auto-create-jms-topics>
    </address-setting>
</address-settings>
```

1. Configures dead letter and expiry queues for the management addresses.
2. Configured dead letter and expiry queues with a wildcard matcher to serve all other queues and topics.

The full `broker.xml` configuration file used for the single broker example in the included OpenShift Container Platform environment can be found in the appendices for further review of overall file structure.

5.4. SINGLE-BROKER INSTANCE

The single-broker instance is configured for client connectivity via its `artemis` connector. The OpenShift service is configured with an ingress port of 30201 which forwards all requests to the container port 61616.

5.4.1. Broker Configuration

Given that no further highly-available or fault-tolerant options are required for the single broker, the common configuration elements detailed above make up the entirety of its configuration.

5.4.2. Exercising the Instance

The following tests demonstrate two basic messaging patterns encountered when working with AMQP via JMS: queues and topics. The first test demonstrates point-to-point messaging by establishing one client as a queue producer and another client as a queue consumer, then asserting that all messages sent to the queue only reach the intended recipient. The second test demonstrates the publish-subscribe pattern by similarly establishing a single topic publisher, but with multiple recipient subscribers who each receive a copy of all dispatched messages.

To run the tests, use the following command:

```
$ mvn -Dtest=SingleBrokerTest test
...
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.redhat.refarch.amq7.single.SingleBrokerTest
...SingleBrokerTest:24 - instantiating clients...
...SingleBrokerTest:33 - sending 25 messages to queue...
...SingleBrokerTest:36 - verifying single queue consumer received all
msgs...
...SingleBrokerTest:40 - terminating clients...
...SingleBrokerTest:49 - instantiating clients...
...SingleBrokerTest:60 - sending 25 messages to topic...
...SingleBrokerTest:63 - verifying both topic consumers received all
msgs...
...SingleBrokerTest:68 - terminating clients...
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 7.669 sec
```

5.5. SYMMETRIC CLUSTER TOPOLOGY

The highly-available symmetric cluster offers round-robin message distribution to all brokers sharing a common queue. Each broker is configured for client connectivity via its `artemis` connector. Each

OpenShift service representing a broker is configured with an ingress port of 3040X which forwards all requests to the corresponding container port 6161X.

5.5.1. Broker Configuration

```
<broadcast-groups>
    <broadcast-group name="test-broadcast-group"> (1)
        <group-address>${udp-address:231.7.7.7}</group-address> (2)
        <group-port>9876</group-port>
        <broadcast-period>100</broadcast-period>
        <connector-ref>artemis</connector-ref> (3)
    </broadcast-group>
</broadcast-groups>

<discovery-groups>
    <discovery-group name="test-discovery-group"> (4)
        <group-address>${udp-address:231.7.7.7}</group-address>
        <group-port>9876</group-port>
        <refresh-timeout>10000</refresh-timeout>
    </discovery-group>
</discovery-groups>

<cluster-connections>
    <cluster-connection name="test-cluster"> (5)
        <connector-ref>artemis</connector-ref>
        <retry-interval>500</retry-interval>
        <use-duplicate-detection>true</use-duplicate-detection>
        <message-load-balancing>ON_DEMAND</message-load-balancing> (6)
        <max-hops>1</max-hops>
        <discovery-group-ref discovery-group-name="test-discovery-
group"/>
    </cluster-connection>
</cluster-connections>
```

(1) Broadcast Group configuration; A broker uses a broadcast group to push information about its cluster-related connection to other potential cluster members on the network.

(2) A broadcast-group can use **TCP**, **UDP** or **JGroups**, but the choice must match its **discovery-group** counterpart.

(3) Connection information: matches a **connector** element as defined in the shared configuration section above.

(4) Discovery Group configuration: while the broadcast group defines how cluster-related information is transmitted, a discovery group defines how connector information is received.

(5) Uses a discovery-group to make the initial connection to each broker in the cluster.

(6) Cluster connections allow brokers to load balance their messages. If message load balancing is **OFF** or **ON_DEMAND**, messages are not moved to queues that do not have consumers to consume them. However, if a matching consumer on a queue closes after the messages have been sent to the queue, the messages will stay in the queue without being consumed.

Further information about configuring AMQ 7 clusters can be found in the Using Red Hat JBoss AMQ 7 Broker Guide.

5.5.2. Exercising the Cluster

The first included test demonstrates both point-to-point and publish-subscribe patterns utilizing JMS clients connecting to different cluster members. The second test demonstrates and asserts the round-robin distribution pattern of messages across all cluster members, from a client connected to a single broker.

```
$ mvn -Dtest=SymmetricClusterTest test
...
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.redhat.refarch.amq7.cluster.SymmetricClusterTest
- instantiate clients, s2 & s3 sub to queue, s1 writes to queue...
- sending 20 messages via s1 producer to queue...
- verifying both queue subscribers received half of the messages...
- verifying there are no more than half of the messages on one of the
receivers...
- terminating clients...
- instantiate clients, all sub to topic, s1 subs to queue, s2 writes to
both...
- sending 25 messages via s2 producer to queue & topic...
- verifying all 3 topic subscribers & single queue consumer received all
msgs...
- terminating clients...
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 18.386 sec
```

5.6. REPLICATION CLUSTER TOPOLOGY

The highly-available and fault-tolerant Replication topology offers the same features as seen with the Symmetric cluster, but with an added master-slave node pairing that serves to handle instances when a master broker node is shut down, stopped, or fails completely. This failover protection mechanism allows slave brokers to automatically match to master nodes within their broadcast group and receive replication of the master's journals as operations are performed. Should the cluster detect the loss of communication with a master node, a quorum vote is executed, potentially leading to the promotion of the paired slave node into the master broker position.

5.6.1. Broker Configuration

The only section differing from that already seen in the symmetric cluster example is a **<ha-policy>** configuration specifying master/slave positions and other possible failover parameters.

5.6.1.1. Master Nodes

```
<ha-policy>
    <replication>
        <master/>
    </replication>
</ha-policy>
```

5.6.1.2. Slave Nodes

```
<ha-policy>
    <replication>
        <slave/>
    </replication>
</ha-policy>
```

Further configuration options, such as possible failback scenarios where a master node returns to a cluster group and shared master/slave persistence, are detailed within the Using Red Hat JBoss AMQ 7 Broker Guide.

5.6.2. Exercising the Cluster

The included test for the Replication topology demonstrates the behavior visible to a JMS client in a scenario where the master broker fails after receiving messages from a clustered connection. The cluster consists of 3 master/slave pairs, with the master nodes connecting via a symmetric group. The test sends twenty messages to each master node, then acknowledges half of the messages received on each. The remaining ten messages from each node are received, but not acknowledged, an important differentiation when working within a cluster/fault-tolerant environment.

At this junction, the AMQP management queue is used by the client to send a **`forceFailover`** command to one master node in the cluster. The test then asserts that the ten remaining messages that had not been acknowledged via the now-failed master node still cannot be acknowledged since the node is offline, while the remaining ten messages on the 2 surviving master nodes are able to be acknowledged without issue. Lastly, the client that was connected to the failed master node demonstrates that seamless failover has occurred by re-consuming the ten non-acknowledged messages from the newly promoted replacement node without need for reestablishing a connection.

```
$ mvn -Dtest=ReplicatedFailoverTest test
...
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.redhat.refarch.amq7.cluster.ha.ReplicatedFailoverTest
- creating clients...
- send 20 messages via producer to all 3 master brokers...
- receiving all, but only acknowledging half of messages on all 3 master
brokers...
- shutting down master broker m2 via Mgmt API forceFailover()...
17-10-09 23:22:56:270  WARN Thread-0 (ActiveMQ-client-global-threads-
110771485)
    core.client:198 - AMQ212037: Connection failure has been detected:
    AMQ119015: The connection was disconnected because of server shutdown
- verifying rec'd-only messages fail to ack if on a failover broker...
- attempting to ack 10 messages from broker m1 post-shutdown...
- attempting to ack 10 messages from broker m2 post-shutdown...
- messages from m2 post-failover correctly failed to ack
- attempting to ack 10 messages from broker m3 post-shutdown...
- re-consuming the non-ack'd m2 messages with slave broker now promoted...
- terminating clients...
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 13.887 sec
```

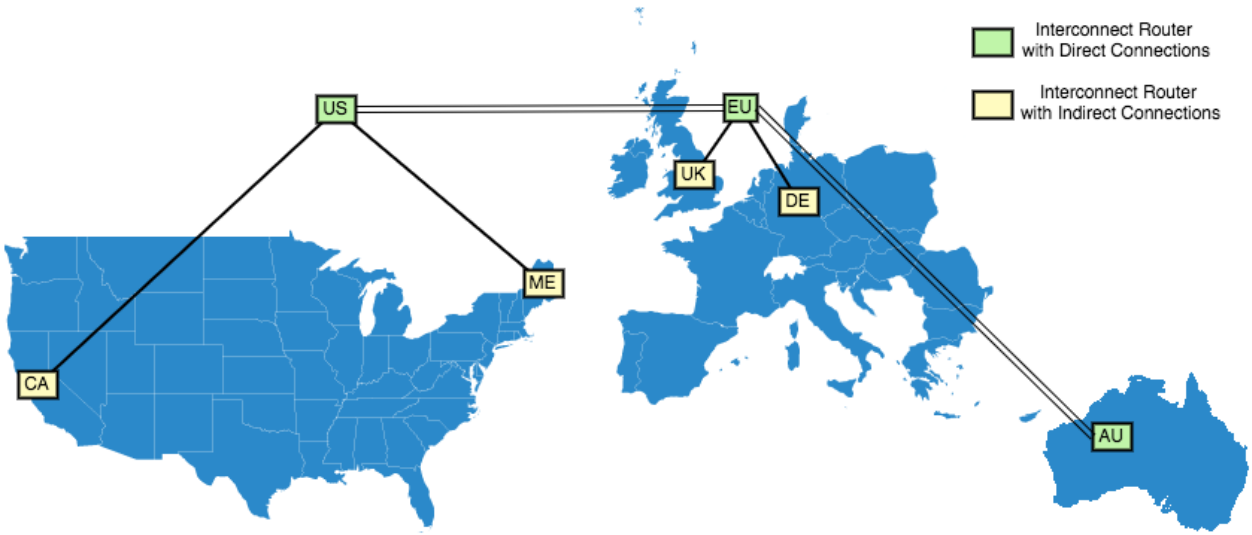
NOTE

Given the nature of the broker containers on OpenShift Container Platform, once the replication test has been ran, the pod will indicate a crashed state, thus requiring a redeployment of the `amq-replicated` deployment to reset the replication cluster back to its original 3-pair master/slave state for further test runs.

5.7. INTERCONNECT NETWORK TOPOLOGY

As a refresher, refer back to the following diagram showing the example Interconnect network and the role that each router plays.

Figure 5.1. Interconnect Router Network



As seen, three routers represent connection points to three major regions, **US**, **EU**, and **AU**. Each of these three routes are connected to one another directly. The **US** and **EU** routers each have two dependent connections representing an area within the larger defined region, while **AU** serves as a single entry point into the network for its whole region.

5.7.1. Router Configuration

Since each major region router and its dependent routers, if any, can accept client requests, a listener on the `amqp` port is defined at the top of each configuration. Since the network also needs to handle routing of any messages that need to reach non-direct portions of the network, the major region routers also feature an `inter-router` listener that allows router-to-router connectivity and defines any `connector` configurations needed to establish the bridges between regions. Note that `inter-router` connections do not need to be defined in both directions.

Those routers dependent on major region connections to reach other portions of the network solely connect to their respective region's router, meaning that no `inter-router` listener is required, but a single `inter-router` connector is set up to establish the necessary network connection.

Every router on the network is configured to handle three different types of routing patterns:

- `closest`: An anycast method in which even if there are more receivers for the same address, every message is sent along the shortest path to reach the destination. This means that only one receiver will get the message. Each message is delivered to the closest receivers in terms

of topology cost. If there are multiple receivers with the same lowest cost, deliveries will be spread evenly among those receivers.

- **`multicast`**: Having multiple consumers on the same address at the same time, messages are routed such that each consumer receives one copy of the message.
- **`balanced`**: An anycast method which allows multiple receivers to use the same address. In this case, messages (or links) are routed to exactly one of the receivers and the network attempts to balance the traffic load across the set of receivers using the same address. This routing delivers messages to receivers based on how quickly they settle the deliveries. Faster receivers get more messages.

Each of the separate router configurations can be viewed in full at the publicly-available repository for further examination.

5.7.2. Exercising the Network

The included tests represent several scenarios playing on different routing patterns and network pathings:

- **`testSendDirect`**: A client connects to the **CA** router and publishes ten messages intended for a receiving client connected to the **DE** router.
- **`testSendMulticast`**: A client connects to the **AU** router and publishes ten messages intended for multiple subscribed clients connected to the **CA**, **ME**, **UK**, and **DE** routers respectively.
- **`testSendBalancedSameHops`**: A client connects to the **AU** router and publishes twenty messages to be balanced between the **CA** and **ME** recipients equally, given their equal route pathing.
- **`testSendBalancedDifferentHops`**: A client connects to the **AU** router and publishes twenty messages to be balanced between the **CA**, **ME**, **UK**, and **DE** recipients, resulting in **UK** and **DE** consumers each receiving seven messages, while the **CA** and **ME** consumers each receive only three due to the extra links present in their route pathing.
- **`testSendClosest`**: Publisher clients are connected to the **CA** and **DE** routers, while consumer clients are simultaneously connected to their sister routers, **ME** and **UK**. A message is dispatched from each publisher with the **`closest`** prefix, resulting in **ME** always consuming from the **CA** publications and **UK** always consuming those from **DE**.

```
$ mvn -Dtest=InterconnectTest test
...
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.redhat.refarch.amq7.interconnect.InterconnectTest
- routing path: CA -> US -> EU -> DE
- creating CA and DE router connections...
- sending 10 messages via CA...
- receiving 10 messages from DE router...
- terminating clients...
- routing path: AU -> EU -> UK,DE,US -> CA,ME
- creating router connections...
- sending 10 messages via CA...
```

```
- receiving 10 messages from all 4 multicast recipient clients...
- terminating clients...
- creating router connections...
- sending 20 messages via AU...
- receiving equal number of balanced messages from CA and ME clients...
- verifying there are no more than half of msgs on one of the receivers...
- terminating clients...
- creating router connections...
- sending 20 messages via AU...
- due to routing weight, rcv fewer msgs from CA/ME than from UK/DE
clients...
- asserting yield is 3 messages to CA/ME each, and 7 messages to UK/DE
each...
- terminating clients...
- creating router connections...
- sending closest message from CA and asserting ME receives...
- sending closest message from DE and asserting UK receives...
- terminating clients...
Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 23.623 sec
```

5.8. AMQ CONSOLE

Each AMQ 7 broker instance can be configured to expose a web console based on **hawtio** that enables management and monitoring of AMQ Broker and Interconnect instances via web browser. By adding the configuration below to the **BROKER_INSTANCE_DIR/etc/bootstrap.xml** configuration file, the necessary port configuration and war deployments will be performed on broker initialization.

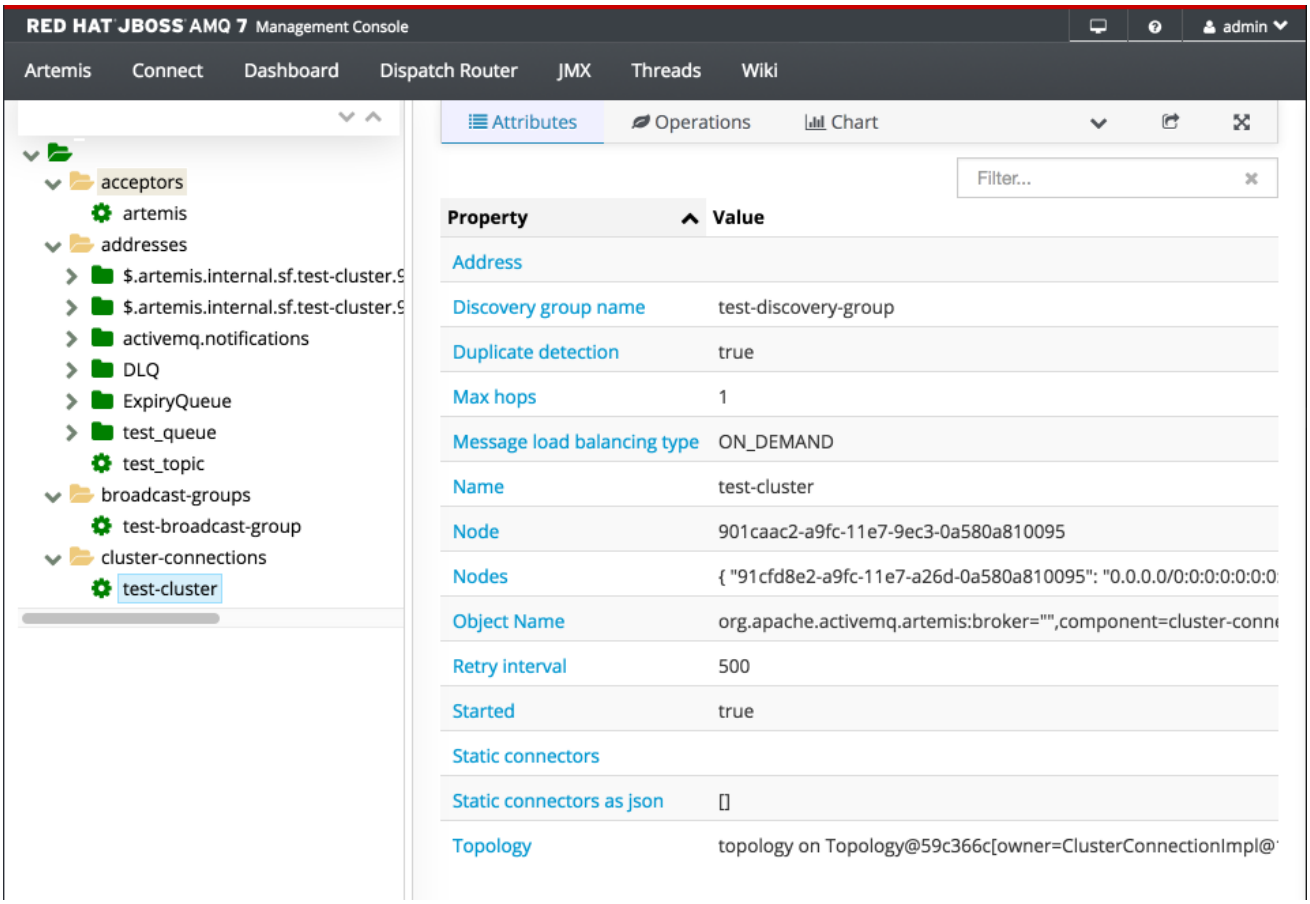
Example Web Binding Configuration:

```
<web bind="http://0.0.0.0:${JOLOKIA_PORT}" path="web">
    <app url="redhat-branding" war="redhat-branding.war"/>
    <app url="jolokia" war="jolokia.war"/>
    <app url="hawtio" war="hawtio-no-slf4j.war"/>
    <app url="artemis-plugin" war="artemis-plugin.war"/>
    <app url="dispatch-hawtio-console" war="dispatch-hawtio-console.war"/>
</web>
```

5.8.1. Broker Monitoring

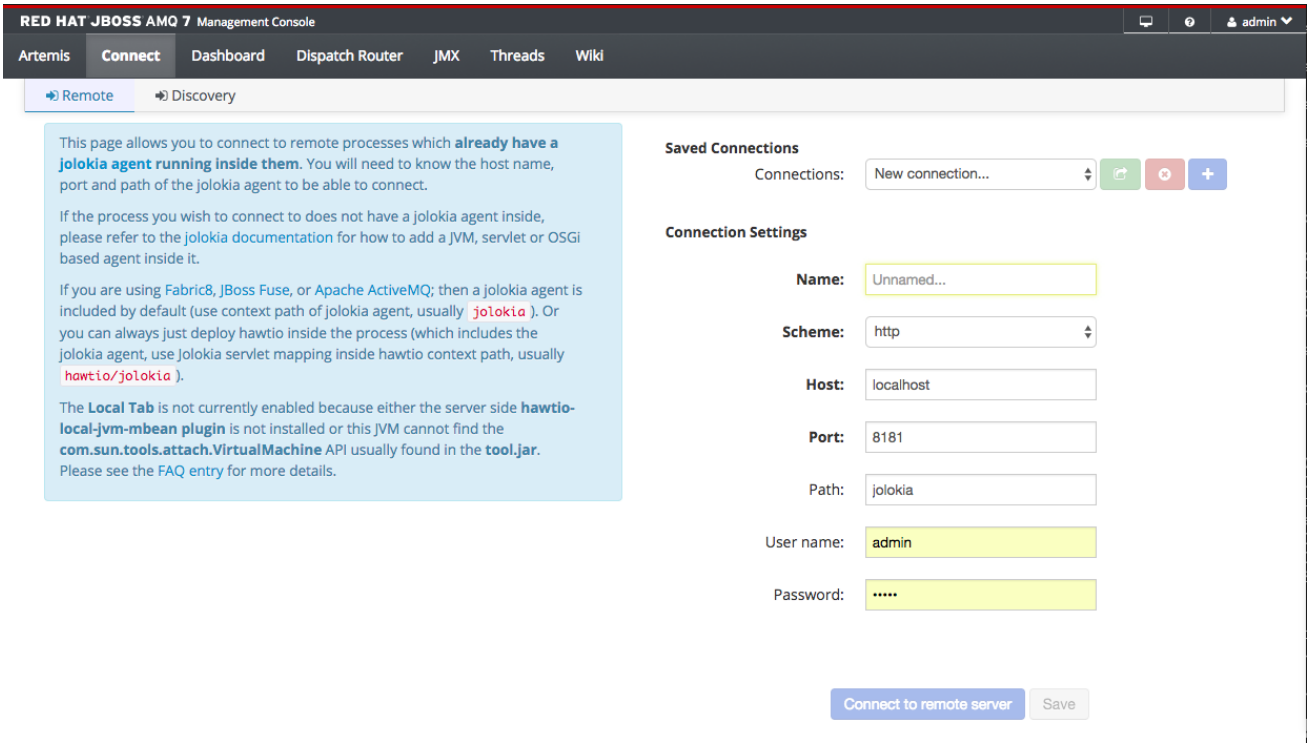
In the OpenShift Container Platform environment that's been established, the single-broker node exposes the AMQ Console via the URL cluster-ingress.example.com:30351/hawtio/. Credentials **admin/admin** can be used to log in. The web console offers various JMX operations, thread monitoring, and router, cluster, address, and acceptor statistics and properties lists for the hosting router.

Figure 5.2. AMQ Console



Other routers hosting AMQ Console can also be reached from within the web console via the **Connect** link, shown below:

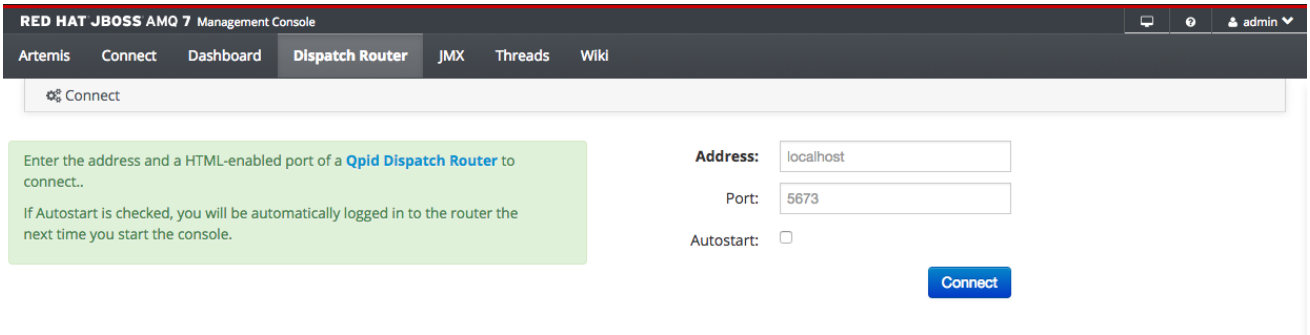
Figure 5.3. Broker Connect



5.8.2. Interconnect Network Monitoring

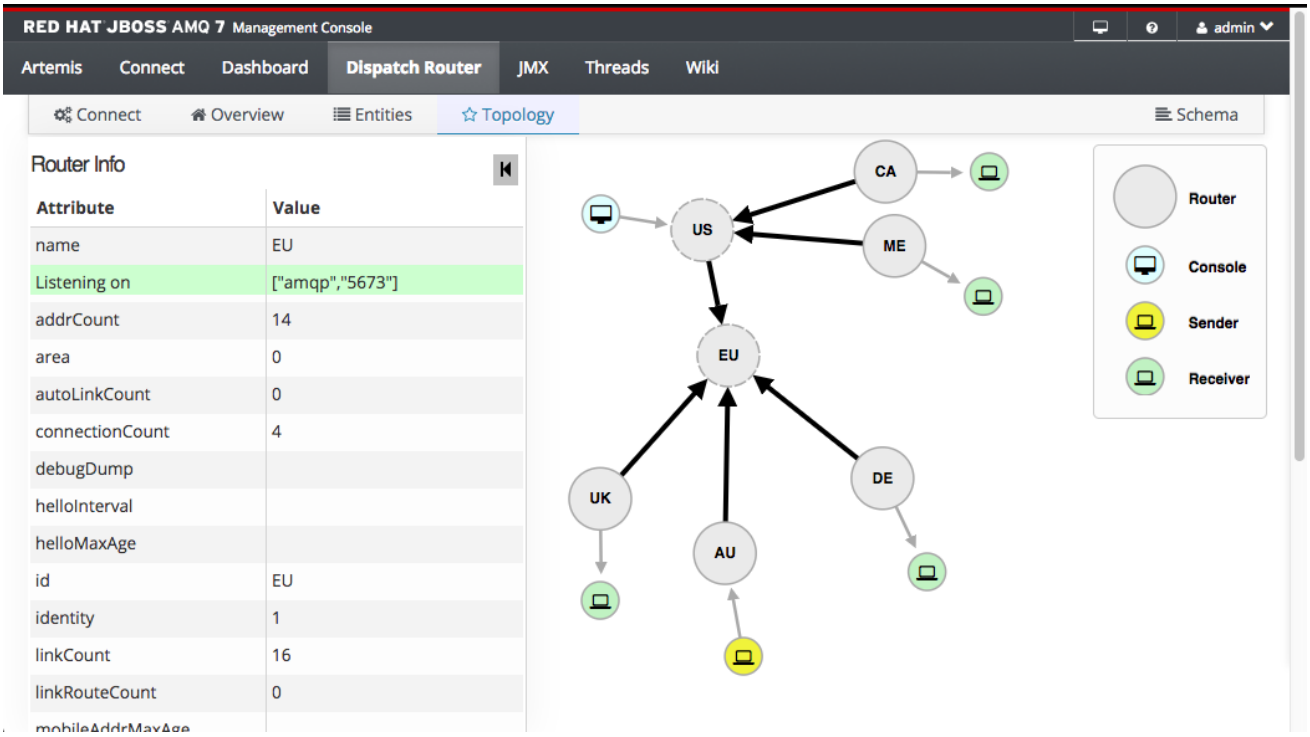
Given that an interconnect installation does not include the necessary components to host AMQ Console, a broker that exists on the same server, or different server altogether, can be used to connect to an http-enabled port listener of any single Interconnect router in order to view and manage the entire network topology.

Figure 5.4. Dispatch Router Connect



Once connected to the Interconnect network, various statistics on addresses, connections, and other components throughout the network may be viewed from the single connected AMQ Console. Additionally, a live network topology diagram may be viewed, which displays all realtime network members and clients.

Figure 5.5. Interconnect Live Topology



CHAPTER 6. CONCLUSION

This document has demonstrated the deployment and configuration of various **Red Hat JBoss AMQ 7** topologies supporting basic messaging via JMS, clustering, fault-tolerance, and brokerless routing. Furthermore, the paper documents interactivity with each environment via JMS, supporting AMQP 1.0, and provides a means of branching out into other languages via various supported clients and protocols to further integrate new and existing products with the newly-built, message-oriented middleware solution.

APPENDIX A. AUTHORSHIP HISTORY

Revision	Release Date	Author(s)
1.0	Oct 2017	Jeremy Ary

APPENDIX B. CONTRIBUTORS

We would like to thank the following individuals for their time and patience as we collaborated on this process. This document would not have been possible without their many contributions.

Contributor	Title	Contribution
Justin Bertram	Senior Software Engineer	Technical Content Review
Dejan Bosanac	Senior Software Engineer	Technical Content Review
Babak Mozaffari	Manager, Software Engineering & Consulting Engineer	Technical Content Review

APPENDIX C. MAVEN CONFIGURATION

```
<?xml version="1.0" encoding="UTF-8"?>

<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
http://maven.apache.org/xsd/settings-1.0.0.xsd">

  <profiles>

    <!-- Configure the JBoss GA Maven repository -->
    <profile>
      <id>jboss-ga-repository</id>
      <repositories>
        <repository>
          <id>jboss-ga-repository</id>
          <url>http://maven.repository.redhat.com/techpreview/all</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>jboss-ga-plugin-repository</id>
          <url>http://maven.repository.redhat.com/techpreview/all</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
        </pluginRepository>
      </pluginRepositories>
    </profile>
    <!-- Configure the JBoss Early Access Maven repository -->
    <profile>
      <id>jboss-earlyaccess-repository</id>
      <repositories>
        <repository>
          <id>jboss-earlyaccess-repository</id>
          <url>http://maven.repository.redhat.com/earlyaccess/all/</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
```

```
          <id>jboss-earlyaccess-plugin-repository</id>
          <url>http://maven.repository.redhat.com/earlyaccess/all/</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>

  <activeProfiles>
    <activeProfile>jboss-ga-repository</activeProfile>
    <activeProfile>jboss-earlyaccess-repository</activeProfile>
  </activeProfiles>

</settings>
```

APPENDIX D. EXAMPLE BROKER CONFIGURATION

```
<?xml version='1.0'?>

<configuration xmlns="urn:activemq"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="urn:activemq /schema/artemis-
configuration.xsd">

    <core xmlns="urn:activemq:core">

        <name>$CONTAINER_ID</name>
        <persistence-enabled>true</persistence-enabled>

        <!-- ========== DIRECTORY CONFIG ========== -->

        <paging-directory>./data/paging</paging-directory>
        <bindings-directory>./data/bindings</bindings-directory>
        <large-messages-directory>./data/large-messages</large-messages-
directory>

        <!-- ========== JOURNAL CONFIG ========== -->

        <journal-type>ASYNCIO</journal-type>
        <!--<journal-type>NIO</journal-type>-->
        <journal-directory>./data/journal</journal-directory>
        <journal-min-files>2</journal-min-files>
        <journal-pool-files>-1</journal-pool-files>
        <journal-buffer-timeout>59999</journal-buffer-timeout>

        <!-- ========== CONNECTION CONFIG ========== -->

        <connectors>
            <!-- default connector: ActiveMQ Artemis, OpenWire (buffered
below), STOMP, AMQP, MQTT, & HornetQ -->
            <connector name="artemis">tcp://0.0.0.0:61616</connector>
            <connector name="amqp">tcp://0.0.0.0:5672?
protocols=AMQP</connector>
        </connectors>
        <acceptors>
            <acceptor name="artemis">tcp://0.0.0.0:61616</acceptor>
            <acceptor name="amqp">tcp://0.0.0.0:5672?
protocols=AMQP</acceptor>
        </acceptors>

        <!-- ========== SECURITY CONFIG ========== -->

        <management-address>activemq.management</management-address>
        <management-notification-
address>jms.topic.notificationsTopic</management-notification-address>

        <security-settings>
            <security-setting match="#">
                <permission type="createNonDurableQueue" roles="admin"/>
                <permission type="deleteNonDurableQueue" roles="admin"/>
                <permission type="createDurableQueue" roles="admin"/>
```

```
                <permission type="deleteDurableQueue" roles="admin"/>
                <permission type="consume" roles="admin"/>
                <permission type="send" roles="admin"/>
                <permission type="manage" roles="admin"/>
            </security-setting>
            <security-setting match="jms.queue.activemq.management">
                <permission type="consume" roles="admin"/>
                <permission type="send" roles="admin"/>
                <permission type="manage" roles="admin"/>
            </security-setting>
        </security-settings>

        <!-- =========== QUEUE/TOPIC CONFIG =========== -->

        <address-settings>
            <!-- default dead letter -->
            <address-setting match="#">
                <dead-letter-address>jms.queue.DLQ</dead-letter-address>
                <expiry-address>jms.queue.ExpiryQueue</expiry-address>
                <redelivery-delay>0</redelivery-delay>
                <max-size-bytes>10485760</max-size-bytes>
                <message-counter-history-day-limit>10</message-counter-
history-day-limit>
                <address-full-policy>BLOCK</address-full-policy>
            </address-setting>
        </address-settings>

        <addresses>
            <address name="test_topic">
                <multicast/>
            </address>
            <address name="test_queue">
                <anycast>
                    <queue name="jms.queue.test_queue"/>
                </anycast>
            </address>
        </addresses>
    </core>
</configuration>
```

APPENDIX E. REVISION HISTORY

Revision 1.0-0	**October 2017**	**JA**