redhat. TRAINING

# CHAPTER 8

# CONFIGURATION OF MESSAGING SUBSYSTEM

| General description | |
|---|---|
| Meta | Describe the messaging architecture and configure different messaging resources. • |
| Goals | Describe the Java messaging service and the architecture of the messaging subsystem.<br>• Configure connection factories and destinations.<br>• Configure messaging journals and more Messaging subsystem configurations. |
| sections | • Exploration of the messaging subsystem (and questionnaire)<br>• Configuration of messaging resources (and exercise guided)<br>• Configuration of journals and other settings (and guided exercise) |
| Laboratory work | • Configuration of the messaging subsystem |

# Exploration of the messaging subsystem

## Goals

**After completing this section, students should be able to do the following:**

**• Describe the Java messaging service and the architecture of the messaging subsystem.**

## messaging concepts

**Messaging middleware, also known as Message Oriented Middleware (MOM) is responsible for managing application messages and delivering them to the intended recipient. This type of messaging is completely unrelated to the popular forms of messaging used by Internet users, such as email and instant messaging. MOM middleware typically offers features such as security, routing, guaranteed delivery, classes of services, and transactions.**

**The MOM middleware can be embedded within other products, similar to how EAP 7 integrates an Artemis ActiveMQ-based MOM. It can also be, and usually is, a stand-alone middleware server, requiring its own specialized administration, tuning, and clustering.**

**Messaging is the foundation of many Enterprise Integration Patterns (EIPs).**
**Building application architecture around messaging has been popular for decades in banking and other industries. Don't think of messaging as "old" technology; it's also a popular pattern for online stores and other types of web applications. The importance of messaging is increasing due to cloud and mobile applications.**

**MOMs and messaging patterns provide a loosely coupled communication mechanism for software components that facilitates business process change and makes the resulting applications more scalable and adaptable. As an example messaging use case, consider the checkout process for a web store. Trigger various actions:**

**• Authorize and confirm the payment.**

**• Remove products from inventory.**

**• Schedule the delivery of products.**

**All of these actions take time to process, and performing them as part of the purchase request means that users may have to wait a long time for their order to be confirmed. But you do NOT need to do them right away; the purchase action can send a message to a payment application and return to the user instantly.**
**When the payment application gets confirmation of store credit, it can send a message to an inventory component. When the products are available and ordered, the inventory application can send a message to the delivery component.**

**Each action can be handled by a different back-end system, making capacity planning easier. If a component cannot perform its action immediately (eg, items are missing from inventory), the action can be composed without canceling the entire process, or an alternate process flow can be triggered. If any item that is**

outside the control of the organization is offline (for example, the credit card operator's system), the process also does not need to be canceled; a new attempt can be made later.

# General description of the messaging service Java (JMS)

The Java Messaging Service (JMS) is a specification for accessing messaging middleware, which allows Java components to send and receive messages without being directly connected to each other.

JMS offers a standardized API for accessing MOM services, similar to how JDBC provides a standardized API for accessing relational databases.
It also defines some common semantics that the MOM must follow so that applications are not dependent on particular product idiosyncrasies.

The JMS API is based on three main resource types:

ConnectionFactory
: Encapsulates the connection to a MOM. JEE 7 indicates that the application server provides a default ConnectionFactory, so that applications that require messaging can be deployed immediately, without knowledge of the connection parameters. The administrator can configure additional ConnectionFactories based on the needs of specific applications on the server, or can point to external MOMs from different vendors.

Queue
: Messages are retrieved on a first-in, first-out basis. A producer puts a message on the queue, and a single consumer removes the message.

Topic
: Implements a publish-subscribe method where messages published to the topic are delivered to multiple recipients subscribed to the topic. If a consumer is offline, the MOM persists the messages for later delivery to that particular consumer.

JMS Queues and Topics are called destinations by JMS, and many operations apply to both, such as sending a message. Applications refer to them not by their internal MOM identifier, but by using a JNDI name. This name is under the control of the application server administrator, creating a layer of indirection that allows applications to be redirected to produce and consume from different destinations without touching the Java source code, making it easier to implement changes to implemented business processes. using messaging.

JMS ConnectionFactories are also referenced using a JNDI name.
In this way, applications can be directed to different MOMs, including different products when requirements change. It is very common to use this feature to connect to the application server's built-in MOM during development, but connect to an external MOM for production.

Like JDBC, the JMS API can be used by stand-alone Java SE applications, outside of a JEE application server. In this case, the application needs to incorporate the MOM-specific JMS provider and there is no standardized way to package and load this provider.

Within an application server, the JMS API collaborates with the Java Connection Architecture (JCA) API to provide a way to package the JMS provider as a Resource Adapter Archive (RAR) that can be deployed on any application server. JEE certified applications.

RAR files are generic middleware drivers, similar to JDBC drivers, but not tied to a specific type of middleware. Different vendors offer RAR files to connect JEE application servers to their middleware products, such as ERP systems and Transaction Monitors.

The JMS specification builds on this JCA capability, rather than providing similar functionality, unlike JDBC, whose origins predate Java EE.

A JMS description is not complete without presenting the message structure. A JMS message is made up of three parts:

- Headers: generally created and processed by the MOM. The JMS API defines some headers that applications can use, but they are generally treated as part of the MOM framework and should not be trusted by application logic in most cases.

- Properties: Defined by the producer to send the metadata to the MOM and to the consumers. An example is routing a message to different consumers based on the value of a property. The above example of a web store can use properties to deliver orders to different payment processing systems.

- Body – Contains the application-specific data that a MOM should not attempt to handle, but rather pass. The components that perform message transformations are, from the JMS point of view, regular client applications and not part of the MOM infrastructure.

## An introduction to the built-in messaging of EAP

Apache ActiveMQ Artemis is an open source project to create an embeddable, very high-performance, clustered, asynchronous, multi-protocol messaging system. The Artemis subproject is a next-generation Apache ActiveMQ messaging server and enterprise integration patterns, based on Red Hat's code donation from the HornetQ project.

ActiveMQ was already very popular due to its support of various protocols and native clients for a range of programming languages and operating systems. HornetQ was also very popular due to its disk I/O and network performance, although it supported only Java clients. The Artemis project brings the strengths of both offerings into the same product.

Red Hat acquired FuseSource in 2012, the company behind commercial support for ActiveMQ, and rebranded the product as JBoss A-MQ server. This left Red Hat with two competing MOM products, both heavily supported by Java EE developers.

In 2014 Red Hat and the HornetQ community decided to join forces with the ActiveMQ community and allow Red Hat to donate HornetQ code to the Apache Software Foundation, thus creating the Artemis project.

Current versions of Red Hat JBoss A-MQ, at the time of EAP 7.0.0 GA, were still based on the original ActiveMQ project. A future version is expected to be based on the Artemis project. This future version will have the external appearance of the current A-MQ product (fabric-based management interfaces, supported by the protocols, and native client libraries), but with many of the internal components of HornetQ (asynchronous I/O engine and disk storage with high-performance journals).

The messaging server built into EAP 7 does not incorporate ActiveMQ Artemis components that support protocols such as STOMP and AMQP. Native client libraries for non-Java languages are also absent in EAP 7.

To understand EAP versus A-MQ messaging from a product perspective: the former is positioned by Red Hat as a general messaging and integration platform, and the latter is positioned as a complete JMS messaging server solution.

Some Java EE application server vendors offer a built-in JMS server that is generally useful only for small-scale development and production. This is NOT the case for EAP 7; Although protocols are missing, the EAP 7 JMS server is fully supported and can address high throughput and mission critical production scenarios. It was not developed for use outside of the Java EE environment.

Clients that require backward compatibility with EAP 6 messaging, based on the HornetQ project, have native protocol support. EAP 6 servers can publish and consume messages from EAP 7 servers, and vice versa.

To use Red Hat JBoss A-MQ instead of EAP's embedded messaging, install the A-MQ JCA Resource Adapter in EAP 7 and configure JMS ConnectionFactories using it. The same procedure (use the product-specific JCA Resource Adapter) should be used when a future A-MQ release becomes Artemis-based.

# Subsistema activemq-messaging

ActiveMQ Artemis integrated into JBoss EAP 7 is the messaging-activemq subsystem, and is managed using the EAP 7 CLI and the administration console. The managed subsystem and resources follow a different syntax compared to the EAP 6 messaging subsystem, but administrators will recognize many of the same features and concepts.

Activation of the messaging-activemq subsystem involves using the standalone-full.xml or standalone-full-ha.xml configuration files for standalone server mode, or the full or full-ha profiles for managed domains.

The subsystem has two child objects in the default EAP 7 configuration, as can be seen from the following CLI command:

```
[domain@127.0.0.1:9990 /] /profile=full/subsystem=messaging-activemq:read-resource {
        "outcome" => "success",
        "result" => { "jms-
                bridge" => undefined, "server" =>
                {"default" => undefined}
        }
}
```

The child objects are:

- jms-bridge – Contains bridges to automatically transfer messages between the integrated JMS server and any other JMS servers known to the EAP 7 domain, including servers from other vendors. No jumpers are predefined in the standard configuration files.
- server=default – The embedded JMS server. It has a number of attributes and Child objects that configure all aspects of MOM, from JMS resources to networking, storage, and clustering. In the remainder of this chapter, we will cover the most common configurations.

Clustering with ActiveMQ is different from general EAP 7 clustering in that it supports a number of different approaches, such as in-memory replication and shared storage. This course does not pretend to describe in detail all these alternative configurations.

For now, suffice it to say that the default configuration of ActiveMQ clusters, which is the same in the standalone-full-ha.xml configuration file and the full-ha profile, does NOT use Infinispan for replication. Instead, the ActiveMQ server automatically creates bridges connecting cluster members and employs an intelligent message distribution algorithm, where non-consumer cluster members do not receive any replication traffic.

Creating these bridges requires shared authentication credentials stored in the cluster-password attribute. The default EAP configuration files set this attribute to the value of the jboss.messaging.cluster.password system property.

Applications deployed on an EAP 7 server instance connect to the local messaging server by default, to avoid network overhead. They are not affected by clustering configurations, in the sense that if the EAP server fails, this messaging server also fails. Nothing prevents administrators from configuring JMS ConnectionFactories to connect to remote servers, and in that case, the EAP server works like any other remote client.

Remote JMS clients discover all members of the messaging cluster in the same way that cluster members discover each other, using JGroups. Connections are distributed among cluster members so that no messaging server becomes a bottleneck. Remote JMS client connections automatically fail over to a surviving cluster member if their current connection fails, and no messages are lost in the process.

General EAP clustering is the subject of the next chapter in this book, and introduces configurations of JGroups. For more information about ActiveMQ configuration and tuning, not limited to clustering, see the EAP 7 and A-MQ product documentation.

# References

**For the JMS specification:**

> **JSR 343 a JCP**
> **https://jcp.org/aboutJava/communityprocess/final/jsr343/index.html**

**For more information about the ActiveMQ and Artemis open source projects:**

> **ActiveMQ project page at Apache Foundation http://activemq.apache.org/**

> **ActiveMQ Artemis project page http://activemq.apache.org/artemis/**

**For specific information about the ActiveMQ built into EAP 7:**

> **EAP 7 Product Documentation: Configuring Messaging https://access.redhat.com/documentation/en/red-hat-jboss-enterprise application-platform/7.0/configuring-messaging/configuring-messaging**

**For more information about Red Hat's JBoss A-MQ product:**

> **A-MQ product page on Red Hat https://www.redhat.com/en/technologies/jboss-middleware/amq**

> **A-MQ Product Documentation https://access.redhat.com/documentation/en/red-hat-jboss-a-mq/**

> **Course JB437 - Red Hat JBoss A-MQ Development and Implementation**
> **https://www.redhat.com/en/services/training/jb437-red-hat-jboss-mq development-and-deployment**

# Quiz: Messaging Subsystem



**Choose the correct answer to the following questions:**

**1. Which of the following is the integrated messaging server in EAP 7? (Choose one option).**

a. HornetQ
b. ActiveMQ Artemis
c. ActiveMQ Apollo
d. JBossMQ
e. Apache Qpid
f. RedÿHat MRG
g. RabbitMQ

**2. An administrator needs to connect EAP 7 to a standalone external MOM. What JMS resource is needed to implement this? (Choose one option).**

a. ConnectionFactory
b. DataSource
c. Queue
d. Topic
e. SecurityDomain
f. Destination

**3. Which of these JEE APIs are focused on client applications rather than the server side components? (Choose three options).**

a. JMS
b. JCA
c. EJB
d. JDBC
It is. Servlets

**4. Which of the following use cases would be good combinations for JMS what? (Choose two options.)**

a. Submit an employee expense report for reimbursement by the HR department. H H. of the organization.
b. Send updated aircraft locations to other airport traffic control systems.
c. Send the customer an email notification about the results of the credit analysis for loans.
d. Send news to social networking sites.

**5. Which of the following use cases would be good combinations for JMS Topics? (Choose two options.)**

a. Deliver an insurance application for risk analysis through an automated system.

b. Withdraw products from inventory to deliver to a customer.

c. Send stock price updates to stockbrokers.

d. Notification of project members about a new meeting invitation.

6. Where in a JMS message should an application place data to influence how the message is routed to specific consumers? (Choose one option).

a. Message body

b. Headers

c. finishers

d. Properties

7. How does an application reference JMS resources like ConnectionFactories and Queues? (Choose one option).

a. Following a naming convention, where the application implementation name is part of the resource name.

b. Using the name assigned to the resource in the JNDI tree of the application server.

c. By using the name assigned to the resource by the MOM.

d. Parsing a configuration directive in the web.xml application implementation description file.

8. Which of the following CLI paths would match the EAP 7 integrated messaging server in the default configuration files? (Choose two options.)

a. /profile=full/subsystem=messaging-activemq/server=default b. /profile=full/subsystem=messaging-activemq/server=artemis c. /profile=full-ha/subsystem=messaging/server=default d. /subsystem=messaging-activemq/server=artemis e. /subsystem=messaging/server=default f.

/subsystem=messaging-activemq/server=default

# Solution

**Choose the correct answer to the following questions:**

**1. Which of the following is the integrated messaging server in EAP 7? (Choose one option).**

a. HornetQ
**b. ActiveMQ Artemis**
c. ActiveMQ Apollo
**d.** JBossMQ **e.** Apache Qpid **f.** RedÿHat MRG
g. RabbitMQ

**2. An administrator needs to connect EAP 7 to a standalone external MOM. What JMS resource is needed to implement this? (Choose one option).**

**a. ConnectionFactory b.** DataSource **c.** Queue
**d.** Topic **e.** SecurityDomain **f.** Destination

**3. Which of these JEE APIs are focused on client applications rather than the server side components? (Choose three options).**

**a. JMS b.** JCA EJB
c. **d.** JDBC
It is. Servlets

**4. Which of the following use cases would be good combinations for JMS what? (Choose two options.)**

a. **Submit an employee expense report for reimbursement by the HR department. H H. of the organization.**
b. Send updated aircraft locations to other airport traffic control systems.
c. **Send the customer an email notification about the results of the credit analysis for loans.**
d. Send news to social networking sites.

**5. Which of the following use cases would be good combinations for JMS Topics? (Choose two options.)**

a. Deliver an insurance application for risk analysis through an automated system.

b. Withdraw products from inventory to deliver to a customer.
c. **Send stock price updates to stockbrokers.**
d. **Notification of project members about a new meeting invitation.**

**6. Where in a JMS message should an application place data to influence how the message is routed to specific consumers? (Choose one option).**

a. Message body
b. Headers
c. finishers
d. **Properties**

**7. How does an application reference JMS resources like ConnectionFactories and Queues? (Choose one option).**

a. Following a naming convention, where the application implementation name is part of the resource name.
b. **Using the name assigned to the resource in the JNDI tree of the application server.**
c. By using the name assigned to the resource by the MOM.
d. Parsing a configuration directive in the web.xml application implementation description file.

**8. Which of the following CLI paths would match the EAP 7 integrated messaging server in the default configuration files? (Choose two options.)**

**a. /profile=full/subsystem=messaging-activemq/server=default b. /**profile=full/subsystem=messaging-activemq/server=artemis **c. /**profile=full-ha/subsystem=messaging/server=default **d. /**subsystem=messaging-activemq/server=artemis **e. /**subsystem=messaging/server=default **f.**

**/subsystem=messaging-activemq/server=default**

# Configuring messaging resources

## Goals

**After completing this section, students should be able to do the following:**

**• Configure JMS connection factories and destinations.**

**• Describe ActiveMQ acceptors and connectors.**

## ActiveMQ Connectors and Acceptors

**For the remainder of this book, the Artemis ActiveMQ built into EAP 7 is referred to as ActiveMQ. Your network configuration is based on two components:**

**• Acceptors: define the protocols and network parameters to accept connections from courier clients.**

**• Connectors: define the protocols and network parameters to connect to servers ActiveMQ.**

### use

**Note that ActiveMQ uses the term connector to refer to a client-side component, while other EAP subsystems generally refer to a server-side component when using this term.**

**Because EAP 7 acts as both a server and a client for ActiveMQ, the default EAP 7 configuration files include out-of-the-box definitions of both types of components. Two types of ActiveMQ acceptors and connectors are provided:**

**• http – Uses the native ActiveMQ protocol that can be accessed over an HTTP connection. The HTTP connection is accepted by the undertow subsystem, which is the subject of a dedicated chapter in this book. In this way, an EAP 7 server instance does NOT require additional firewall ports to be opened to accept connections from remote messaging clients.**

**• in-vm – allows messaging clients to run on the same JVM as the server ActiveMQ to connect without network overhead.**

**The network parameters for an ActiveMQ connector are defined indirectly, by referencing an acceptor element. The connector connects to any IP address and TCP port on which the acceptor is configured to accept connections. This indirection is related to the way that remote ActiveMQ clients use a discovery mechanism to find servers to connect to. The following list shows the sockets and acceptors in the EAP 7 default configuration files:**

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:1.0"> <server
      name="default">
            ...
            <http-connector name="http-connector" socket-binding="http"
                  endpoint="http-acceptor" />  (1)
```

```
            ....
            <in-vm-connector name="in-vm" server-id="0"/>
            <http-acceptor name="http-acceptor" http-listener="default" />
            ....
            <in-vm-acceptor name="in-vm" server-id="0"/>
            ...
        </server> </
subsystem>
```

**The calls in the above list highlight the relationship between connectors and acceptors:**

**1 3 The <http-connector> element named http-connector references the <http-acceptor> element named http-acceptor using the endpoint attribute.**

**2 The <in-vm-connector> element does not need to reference any <in vm-acceptor> elements, since local clients do NOT need discovery to find the messaging server.**

**4 The <http-acceptor> element refers to the http-listener named default, defined by the undertow subsystem.**

**EAP 7 supports a third type of socket and acceptor: the remote type. Allows remote clients to connect to the messaging server using the native ActiveMQ protocol instead of HTTP tunneling. The native protocol is required by remote clients running outside of the EAP 7 server, as well as for connecting to standalone ActiveMQ servers.**

**Creating a new <remote-connector> and <remote-acceptor> pair in EAP 7 also requires the creation of a new <socket-binding> to provide the IP address and TCP port that the <remote-acceptor> should use. For more information, see the EAP product documentation.**

**The Remote Acceptor and Connector type also supports connections to EAP 6 and stand-alone HornetQ servers, although additional configuration is required. For more information, see the EAP 7 product documentation.**

# Defining ConnectionFactories and PooledConnectionFactories de JMS

**JMS ConnectionFactory resources defined within the messaging activemq subsystem refer to an ActiveMQ connector defined within the same subsystem.**
**The following list shows the connection-factory resources in the default EAP 7 configuration files:**

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:1.0"> <server
        name="default">
            ....
            <connection-factory name="InVmConnectionFactory" connectors="in-vm"
  entries="java:/ConnectionFactory"/> <connection-
            factory name="RemoteConnectionFactory" connectors="http-connector"
  entries="java:jboss/exported/jms/RemoteConnectionFactory"/>
            ...
        </server> </
subsystem>
```

The above list shows two <connection-factory> elements, each referring to a different connector and bound to different JNDI names using the entries attribute:

- The first <connection-factory> element is named InVmConnectionFactory and refers to the <in-vm-acceptor> element named in-vm. It supports applications that use the JMS API in a Java SE style, but is implemented in EAP 7. It should NOT be used by business components, such as Servlets and EJBs.
- The second <connection-factory> element is named RemoteConnectionFactory and refers to the <http-acceptor> element named http-acceptor. Supports remote JMS clients connecting to the ActiveMQ built into EAP 7.

Java EE applications must NOT use any of the JSM ConnectionFactory resources. Instead, they should use a JSM PooledConnectionFactory that provides:

- Reuse of connections to the messaging server.
- Integration of XA transactions.
- JAAS security context propagation.

The following list shows the pooled-connection-factory resource in the EAP 7 default configuration files:

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:1.0"> <server
        name="default">
                ....
                <pooled-connection-factory name="activemq-ra" transaction="xa"
                        connectors="in-vm"
                        entries="java:/JmsXA java:jboss/DefaultJMSConnectionFactory"/>
                ...
        </server> </
subsystem>
```

This will be used by Java EE applications when no JMS ConnectionFactory is declared. Notice that it refers to the in-vm connector. If a different messaging server will be used as the default, it must be modified to reference a different connector element or replaced with one that references a JCA Resource Adapter.

An application may refer to a JMS ConnectionFactory by a JNDI name that may not exist. Thus, the administrator must create a new pooled connection-factory resource on the same JNDI that refers to the correct connector. For example, to use the JNDI name java:/MyCF to indicate the integrated messaging server, add the following to the messaging-activemq subsystem:

```
<pooled-connection-factory name="mycf" transaction="xa" connectors="in-vm" entries="java:/MyCF"/
  >
```

The following CLI command creates the resource listed above in the full profile for an EAP 7 managed domain:

```
[domain@localhost:9990 /] cd /profile=full/subsystem=messaging-activemq/server=default
```

```
[domain@localhost:9990 server=default] ./pooled-connection-factory=mycf:add(\ connectors=[in-
vm], entries=[java:/jms/MyCF])
```

**If an application refers to a non-existent JNDI name, it can be implemented without errors, throwing Java NamingException errors only at runtime, when the application tries to connect to the messaging server.**

## use

**The first version of EAP 7 only allowed simple connection-factory resources to be configured using the web administration interface. You had to use the CLI to create or configure the pooled-connection factory resources.**

**The reason for adopting a JMS PooledConnectionFactory instead of a simple JMS ConnectionFactory is the same as a JDBC DataSource versus a DriverManager. Although Java EE applications can use the JMS ConnectionFactory and JDBC DataSource, they are not efficient because:**

- **The application includes additional code to manage the exchange of connections between multi-threaded, but loses scalability by doing so, or**
- **The application creates and destroys connections for each user request, which are expensive operations.**

**Both the JDBC DataSource and JMS PooledConnectionFactory objects provide connection pooling to improve application performance and scalability. The difference is that with JDBC, the developer has to code the correct interface, but in the case of JMS, the administrator only has to create the correct resource type: a JMS PooledConnectionFactory is also a JMS ConnectionFactory.**

## use

**To create a JMS PooledConnectionFactory that references an external, non-ActiveMQ MOM, a number of steps must be followed:**

1. **Package the MOM provider's JMS provider as a JBoss module and add it to the EAP installation or deploy the RAR package, if provided by the provider.**
2. **Define an external JNDI context to reference the JNDI tree of the JMS provider, in the naming subsystem.**
3. **Add a JCA resource adapter configuration to the subsystem resource-adapters.**
4. **Configure the ejb subsystem to use the new resource adapter for CIS.**

**The EAP 7 product documentation includes examples of configurations for popular MOMs, such as Tibco EMS and WebSphere MQ.**

# Definition of JMS destinations

**Using the EAP 7 built-in MOM, creating the JMS destination resources also creates the underlying MOM objects. This facility is provided by the messaging-activemq subsystem, which tightly integrates EAP 7 with the embedded ActiveMQ.**

## use

**If an external MOM is used, the JMS destinations are configured as part of the JMS Provider configuration in the naming and resource adapters subsystems, but creating them does NOT create the underlying objects in the MOM. This course covers only the integrated messaging server.**

**To create a destination using the CLI, create a jms-queue or jms-topic resource. For example, the following commands create a JMS Topic named actions, which is bound to the JNDI name java:/jms/broker/StockUpdates:**

```
[domain@localhost:9990 server=default] ./jms-topic=stocks:add(\ entries=[java:/
jms/broker/StockUpdates])
```

**The above command adds the following to the domain mode configuration files:**

```
<jms-topic name="stocks" entries="java:/jms/broker/StockUpdates"/>
```

**The attributes of the jms-topic resource are:**

- **name: The name of the managed object. This name is only useful for administration, and applications that use the JMS API do NOT use it.**
- **entries – Provides one or more JNDI names for the resource. These are the names visible to applications.**
- **legacy-entries – Allows this destination to be used by remote EAP 6 clients and HornetQ.**

**A jms-queue resource can also have all attributes by adding a jms topic resource:**

- **durable—If true, messages in this queue are saved to disk and not they are lost when you restart MOM. If false, messages are kept only in memory and can be deleted in out-of-memory situations.**
- **selector: Provides a filter for messages accepted by this queue. to get more information, see the JMS specification.**

**JMS destinations for the EAP 7 integrated MOM can also be created and configured using the administration console. In managed domain mode, enter Configuration > Profiles > full (or full-ha) > Messaging-ActiveMQ > default, to enter the Messaging Provider page, as shown in the following figure.**

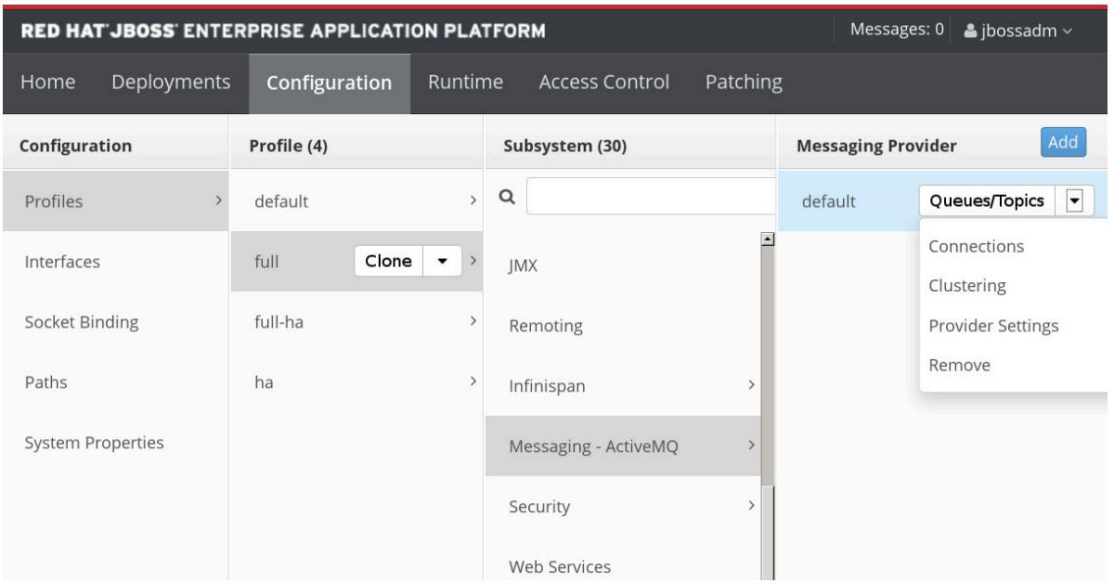


***Figure 8.1:***
***EAP Messaging Provider Selection Page***

**On the Messaging Provider page, click Queues/Topics to view the JMS Endpoints configuration page, as illustrated in the following figure:**

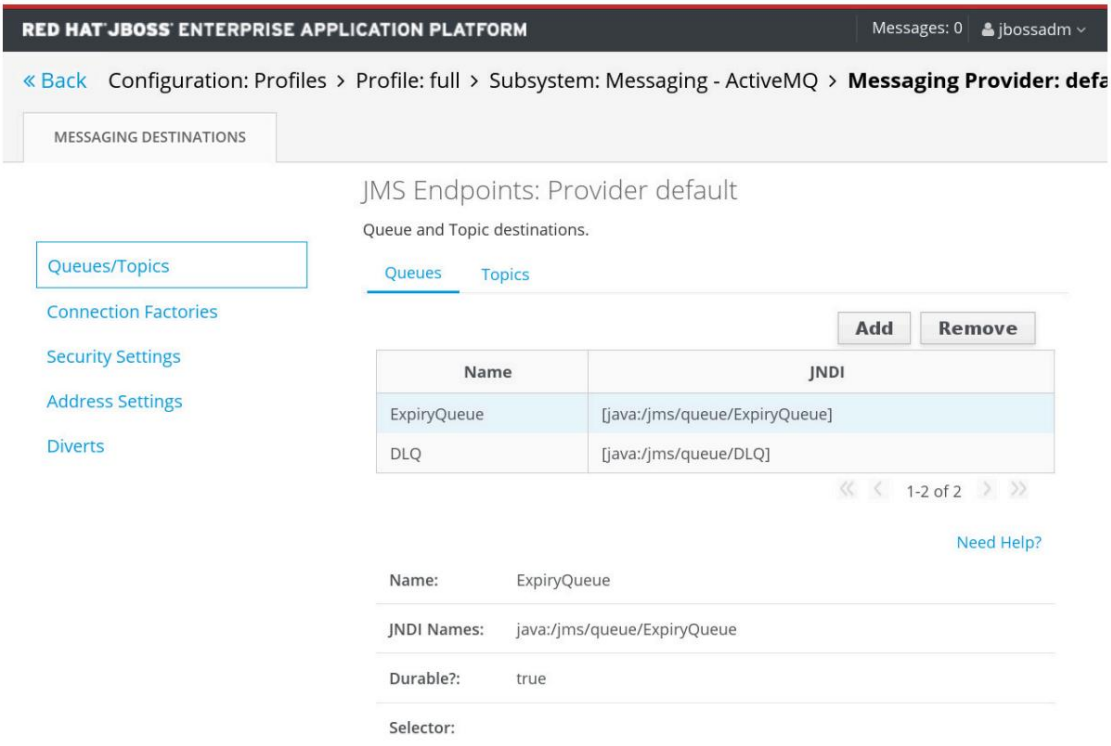


***Figure 8.2:***
***EAP 7 JMS Endpoints Page***

**From that page, new destinations can be created and existing ones can be created or modified. The links on the left also allow you to configure other messaging parameters and are explained later in this chapter.**

# JMS destination monitoring

**JMS destinations in EAP 7 integrated ActiveMQ can be monitored using some attributes and object runtime operations. Use the CLI operations read resource-description and read-resource-operations to learn about all the attributes and operations that EAP 7 offers to monitor JMS destinations. In this course only a few are presented.**

**In standalone server mode, monitoring attributes and operations are available from the same object that defines the resource as part of the server profile.**
**For example, to check the number of messages waiting to be consumed in the jms queue named MyQueue, use the following command to read the message count attribute:**

```
[standalone@localhost:9990 /] cd /subsystem=messaging-activemq/server=default
[standalone@localhost:9990 server=default] ./jms-queue=MyQueue:read-attribute(\ name=message-
count)
```

**Another interesting attribute is the messages-added attribute, which counts a number of messages published to the destination since the MOM was started; that is, since the EAP server instance was last started or reloaded.**

**In managed domain server mode, the same attributes and operations are available from a running server instance and NOT from the subsystem configuration. For example, to list the messages waiting to be consumed in the jms-queue named MyQueue from server srv1 on host devhost, use the following command to invoke the list-messages operation:**

```
[domain@localhost:9990 /] cd /host=devhost/server=srv1
[domain@localhost:9990 server=srv1] cd ./subsystem=messaging-activemq/server=default
[domain@localhost:9990 server=default] ./jms-queue=MyQueue:list-messages
```

**Messages are listed using the DMR syntax, but for each message, only the headers and properties as object attributes are displayed. There is no way to view the body of a message using the EAP CLI. The following is a sample output from the above operation:**

```
{
    "outcome" => "success",
    "result" => [ {

                "JMSPriority" => 4,
                "JMSMessageID" => "ID:b921a30-12e4-11e6-ae18-597e323e1397", "address"
                => "jms.queue.TestQueue", "JMSExpiration"
                => 0, "__AMQ_CID" =>
                "f51cfb7a-12e4-11e6-ae18-597e323e151397", "MyAppProperty" =>
                "MyAppValueXYZ", "JMSTimestamp" =>
                1462468442322L, "messageID" => 5,
                "JMSDeliveryMode"
                => "PERSISTENT"
```

```
            },
            {
                "JMSPriority" => 4, "JMSMessageID"
                => "ID:f5136421-12e4-11e6-ae18-597e323e2def", "address" => "jms.queue.TestQueue", "JMSExpiration" => 0,
                "__AMQ_CID" => "3d0cfb7a-12e4-11e6-ae18-597e323e2def",
                "OtherAppProperty" =>
                "OtherValueASD", "JMSTimestamp" => 1462468442339L, "messageID" => 9, "JMSDeliveryMode"
                => "PERSISTENT"



            }
        ]
}
```

**In the list above, the result attribute is an array containing two objects, each with a single message. The attributes of each message object are headers and properties of the JMS message. Some of them, for example those with the JMS prefix, are defined by the JMS specification. Other headers, such as address and messageID, are defined by the MOM. Finally, the MyAppProperty and OtherAppProperty properties are application-defined and have no meaning to the JMS APIs or the MOM.**

## use

**There is no way to infer, just from the output of the list messages operation, which attributes are message headers and which are message properties. It is also not possible to determine which are defined by the MOM and which are not. Check the EAP and A-MQ product documentation as well as the JMS specification for information about specific headers and properties. Also check with the application developer about application-defined properties.**

**Some target attributes can be viewed using the administration console. Select Runtime from the top navigation menu and navigate to Messaging-ActiveMQ within a running server. Click View to view the Messaging Statistics page, and select the default messaging provider. Then select a queue (or topic) to view the statistics. More information is usually available through the CLI.**

**The following figure illustrates navigating through the administration console in managed domain mode to reach the Messaging Statistics page:**

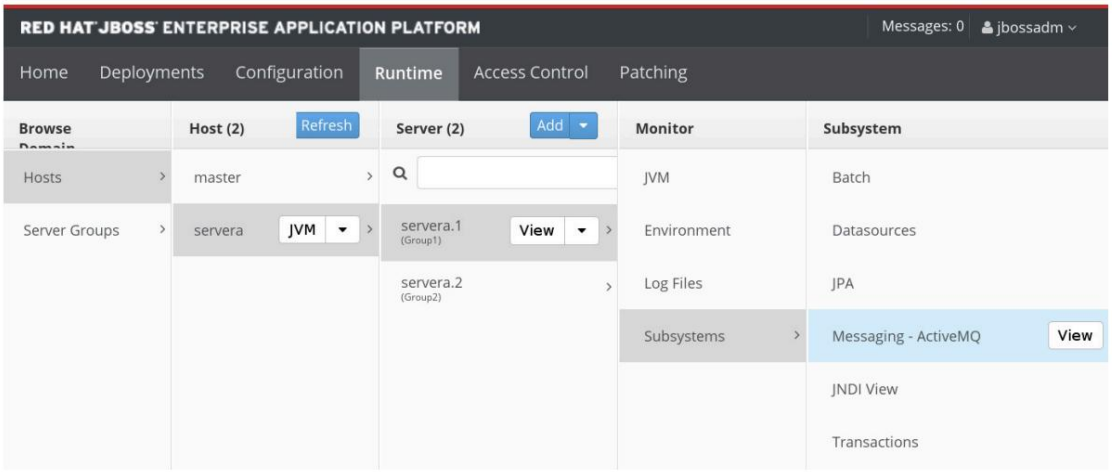


***Figure 8.3: Navigate to the Messaging-ActiveMQ monitoring page.***

**Of course monitoring attributes and operations is different for jms queue and jms-topic destinations. For example, only topics have subscribers.**