Red Hat JBoss A-MQ 6.3

Integrating with JBoss Enterprise Application Platform

Installing the ActiveMQ resource adapter into the JBoss Enterprise Application Platform container

Last Updated: 2019-02-20

Red Hat JBoss A-MQ 6.3 Integrating with JBoss Enterprise Application Platform

Installing the ActiveMQ resource adapter into the JBoss Enterprise Application Platform container

JBoss A-MQ Docs Team
Content Services
fuse-docs-support@redhat.com

Abstract

This guide describes how to integrate the ActiveMQ resource adapter into a JBoss Enterprise Application Platform and how to run an example with Message Driven Beans.

Table of Contents

CHAPTER 1. DEPLOYING THE APACHE ACTIVEMQ RESOURCE ADAPTER

Abstract

This chapter explains how to install the Apache ActiveMQ resource adapter into JBoss EAP and how to integrate ActiveMQ messaging into your applications, taking the **`helloworld-mdb`** demonstration as an example.

1.1. SUPPORTED WEB SERVER PLATFORMS

Overview

The following Web server platforms are supported by JBoss A-MQ 6.3:

- *JBoss Enterprise Application Platform (JBoss EAP)*

Supported product versions

To see which versions of JBoss EAP are supported with JBoss A-MQ 6.3, please consult the Red Hat JBoss A-MQ Supported Configurations page.

NOTE

AMQP 1.0 is *not* a supported protocol for the JBoss A-MQ JCA connector (Apache ActiveMQ resource adapter). OpenWire is the only wire protocol supported by the JCA connector / resource adapter.

1.2. INSTALL THE ACTIVEMQ RESOURCE ADAPTER

Overview

This section describes how to find, install, and configure the ActiveMQ resource adapter into a standalone instance of the JBoss EAP.

A *resource adapter* is a kind of plug-in for a Java EE container. The Java EE standard defines the resource adapter framework, which makes it possible to expand the core Java EE container, adding new features and functionality. By installing the ActiveMQ resource adapter, you make it possible for message driven beans and servlets to communicate through an external JBoss A-MQ broker instance. The JBoss A-MQ broker can thus be used as the underlying messaging system in the container.

NOTE

Running an embedded ActiveMQ broker in JBoss EAP is not supported. The use of the resource adapter is restricted to connect to external broker instances only.

Resource adapter location

You can find the ActiveMQ resource adapter archive file, **`activemq-rar-5.11.0.redhat-630187.rar`**, at either of the following locations:

- In the following Zip archive file:

  ```
  InstallDir/extras/apache-activemq-5.11.0.redhat-630187-bin.zip
  ```

  After expanding the archive, the resource adapter file can be found in the following sub-directory:

  ```
  apache-activemq-5.11.0.redhat-630187/lib/optional
  ```

- Directly from the Red Hat JBoss Fuse Maven repository, at the following URL:

  ```
  https://maven.repository.redhat.com/ga/org/apache/activemq/activemq-
  rar/
  ```

  Download the **`.rar`** archive file from the appropriately versioned sub-directory, 5.11.0.redhat-630187.

Configuration files

The following configuration files are needed for the the ActiveMQ resource adapter (when installed in a standalone instance of the JBoss EAP):

`InstallDir`**`/standalone/configuration/standalone-full.xml`**

The **`standalone-full.xml`** file is the default (bare bones) configuration for the JBoss EAP container. You must edit this file to complete the installation of the ActiveMQ resource adapter.

NOTE

It is assumed that this file does *not* already configure the HornetQ messaging system (which would conflict with the ActiveMQ messaging system).

NOTE

JBoss EAP can be figured either as a standalone container, using **`standalone/configuration/standalone-full.xml`**, or as a managed domain, using **`domain/configuration/domain.xml`**. Throughout this section, we describe explicitly how to configure the standalone container, but it is understood that a similar approach could be used to configure a managed domain.

Steps to install the resource adapter

Perform the following steps to install the Apache ActiveMQ resource adapter into JBoss EAP (assuming that you will be running the container in standalone mode):

1. Extract the Apache ActiveMQ community distribution. You can find an archive of the Apache ActiveMQ distribution in the following location:

   ```
   InstallDir/extras/apache-activemq-5.11.0.redhat-630187-bin.zip
   ```

   Using a suitable archive utility, extract the archive file to a convenient location on your filesystem. The root of the extracted directory tree is called **`apache-activemq-5.11.0.redhat-630187`** by default.

2. The ActiveMQ resource adapter archive file can now be found under the **`/lib/optional`** sub-directory of the extracted archive.

   If you need to use the ActiveMQXAConnectionFactory, you need to add extra details to the ActiveMQ resource adapter file at this point. Extract the resource adapter archive, **`activemq-rar-5.11.0.redhat-630187.rar`** to a work space. In the expanded resource adapter archive, edit **`META_INF/ra.xml`**.

   Add the following entry to the adminobjects in the **`ra.xml`** file.

   ```
   <adminobject>
   <adminobject-interface>javax.jms.XAConnectionFactory</adminobject-
   interface>
   <adminobject-
   class>org.apache.activemq.ActiveMQXAConnectionFactory</adminobject-
   class>
   <config-property>
   <config-property-name>brokerURL</config-property-name>
   <config-property-type>java.lang.String</config-property-type>
   </config-property>
   </adminobject>
   ```

   Zip up the resource adapter archive, and save it to a temporary location.

3. Rename the updated and saved resource adapter archive file. The new file name should be the same as the original name, but without the version number in the filename. For example, on a UNIX or Linux platform, you can rename the **`activemq-rar-5.11.0.redhat-630187.rar`** archive file to **`activemq-rar.rar`**

   NOTE

   Renaming the resource adapter archive in this way is not strictly necessary. But because the resource adapter file name appears in the resource adapter configuration, using a versionless filename makes it easier to upgrade the resource adapter at a later date.

4. Install the ActiveMQ resource adapter by copying the resource adapter archive, **`activemq-rar.rar`**, to the relevant JBoss EAP deployment directory. For example, on a UNIX or Linux platform, you could copy the resource adapter archive to a standalone JBoss Enterprise Application Platform as follows:

   ```
   cp activemq-rar.rar EAPInstallDir/standalone/deployments/
   ```

5. Add the requisite resource adapter configuration to the **`urn:jboss:domain:resource-adapters:1.1`** subsystem in the JBoss EAP configuration, as follows.

   Open the ***`EAPInstallDir`*****`/standalone/configuration/standalone-full.xml`** file using a text editor and paste the **`resource-adapter`** element from Example 1.1, “ActiveMQ Resource Adapter Configuration in standalone-full.xml” into the **`urn:jboss:domain:resource-adapters:1.1`** subsystem, as a child of the **`resource-adapters`** element.

   Example 1.1. ActiveMQ Resource Adapter Configuration in standalone-full.xml

```
<server xmlns="urn:jboss:domain:1.4">
    ...
    <profile>
        ...
        <subsystem xmlns="urn:jboss:domain:resource-
adapters:1.1">
            <resource-adapters>
                <resource-adapter id="activemq-rar.rar">
                    <archive>
                        activemq-rar.rar
                    </archive>
                    <transaction-
support>XATransaction</transaction-support>
                    <config-property name="UserName">
                        defaultUser
                    </config-property>
                    <config-property name="Password">
                        defaultPassword
                    </config-property>
                    <config-property name="ServerUrl">
                        tcp://localhost:61616?
jms.rmIdFromConnectionId=true
                    </config-property>
                    <connection-definitions>
                        <connection-definition
                              class-
name="org.apache.activemq.ra.ActiveMQManagedConnectionFactory"
                              jndi-
name="java:/ConnectionFactory"
                              enabled="true"
                              pool-name="ConnectionFactory"
         tracking="false">
                            <xa-pool>
                                <min-pool-size>1</min-pool-size>
                                <max-pool-size>20</max-pool-
size>

                                <prefill>false</prefill>
                                <is-same-rm-override>false</is-
same-rm-override>
                            </xa-pool>
                            <recovery>
                                <recover-credential>
                                    <user-
name>defaultUser</user-name>

<password>defaultPassword</password>
                                </recover-credential>
                            </recovery>
                        </connection-definition>
                    </connection-definitions>
                    <admin-objects>
                        <admin-object
                              class-
name="org.apache.activemq.command.ActiveMQQueue"
                              jndi-
name="java:/queue/HELLOWORLDMDBQueue"
```

```
                              use-java-context="true"
                              pool-name="HELLOWORLDMDBQueue">
                            <config-property
name="PhysicalName">
                                HELLOWORLDMDBQueue
                            </config-property>
                        </admin-object>
                        <admin-object
                              class-
name="org.apache.activemq.command.ActiveMQTopic"
                              jndi-
name="java:/topic/HELLOWORLDMDBTopic"
                              use-java-context="true"
                              pool-name="HELLOWORLDMDBTopic">
                            <config-property
name="PhysicalName">
                                HELLOWORLDMDBTopic
                            </config-property>
                        </admin-object>
                    </admin-objects>
                </resource-adapter>
                ...
            </resource-adapters>
        </subsystem>
        ...
    </profile>
    ...
</server>
```

If you need to use the ActiveMQXAConnectionFactory, you must also add the following information to the **`admin-object`** section of **`standalone-full.xml`**.

```
<admin-object class-
name="org.apache.activemq.ActiveMQXAConnectionFactory" jndi-
name="java:/AMQXAConnectionFactory" enabled="true" use-java-
context="true" pool-name="AMQXAConnectionFactory">
<config-property name="brokerURL">
tcp://localhost:61616?jms.rmIdFromConnectionId=true
</config-property>
</admin-object>
```

If your resource adapter archive filename differs from **`activemq-rar.rar`**, you must change the content of the **`archive`** element in the preceding configuration to match the name of your archive file.

The values of the **`UserName`** and **`Password`** configuration properties must be chosen to match the credentials of a valid user in the external broker.

You might need to change the value of the **`ServerUrl`** configuration property to match the actual hostname and port exposed by the external broker.

IMPORTANT

In order to ensure that JMS transactions are integrated correctly, it is essential to include the **`jms.rmIdFromConnectionId=true`** option setting on the **`ServerUrl`** configuration property and to include the **`<is-same-rm-override>false</is-same-rm-override>`** setting in the **`xa-pool`** element, as shown above.

NOTE

The JMS administrative objects defined in the **`admin-objects`** element do not need to be defined yet. They serve as examples to show how you can define administrative objects for the ActiveMQ resource adapter (they are used later in the message-driven bean demonstration).

6. Add the requisite message driven bean configuration to the **`urn:jboss:domain:ejb3:1.5`** subsystem in the JBoss EAP configuration (where the subsystem version might vary, depending on which version of JBoss EAP you are using).

   Open the ***`EAPInstallDir`*****`/standalone/configuration/standalone-full.xml`** file using a text editor and paste the **`mdb`** element from Example 1.2, “Message Driven Bean Configuration in standalone-full.xml” into the **`urn:jboss:domain:ejb3:1.5`** subsystem.

Example 1.2. Message Driven Bean Configuration in standalone-full.xml

```
<server xmlns="urn:jboss:domain:1.4">
    ...
    <profile>
        ...
        <subsystem xmlns="urn:jboss:domain:ejb3:1.5">
            ...
            <mdb>
                <resource-adapter-ref resource-adapter-
name="activemq-rar.rar"/>
                <bean-instance-pool-ref pool-name="mdb-strict-
max-pool"/>
            </mdb>
            <pools>
                <bean-instance-pools>
                    <strict-max-pool name="slsb-strict-max-pool"
max-pool-size="20" instance-acquisition-timeout="5" instance-
acquisition-timeout-unit="MINUTES"/>
                    <strict-max-pool name="mdb-strict-max-pool"
max-pool-size="20" instance-acquisition-timeout="5" instance-
acquisition-timeout-unit="MINUTES"/>
                </bean-instance-pools>
            </pools>
            ...
        </subsystem>
        ...
    </profile>
    ...
</server>
```

7. Before starting the broker, check the broker configuration to make sure that there are valid user credentials defined in the broker's ***InstallDir*/etc/users.properties** file. For example, to match the **UserName** and **Password** credentials configured in Example 1.1, “ActiveMQ Resource Adapter Configuration in standalone-full.xml”, the **users.properties** file should contain an entry like the following:

   ```
   defaultUser=defaultPassword,admin
   ```

8. Start the external A-MQ broker. For example, on a UNIX or Linux platform, you can start the JBoss A-MQ broker instance as follows:

   ```
   cd InstallDir/bin
   ./amq
   ```

9. Start the standalone instance of JBoss EAP. For example, on a UNIX or Linux platform, you can start the standalone instance as follows:

   ```
   cd EAPInstallDir/bin
   ./standalone-full.sh
   ```

Steps to secure the resource adapter with SSL/TLS

The following steps describe how to modify the configuration of the ActiveMQ resource adapter to enable it to connect to a broker that is secured by an SSL/TLS protocol:

1. It is assumed that the remote broker is already configured with an SSL/TLS endpoint. For details of how to configure an OpenWire/SSL endpoint on the broker, see section "Setting Security Context for the Openwire/SSL Protocol" in "Security Guide".

2. Open the ***EAPInstallDir*/standalone/configuration/standalone-full.xml** file using a text editor and modify the **ServerUrl** property setting of the **activemq-rar.rar** resource adapter to point at the remote broker's OpenWire/SSL endpoint. For example:

   ```
   <resource-adapter id="activemq-rar.rar">
       ...
       <config-property name="ServerUrl">
           ssl://localhost:61617?jms.rmIdFromConnectionId=true
       </config-property>
       ...
   </resource-adapter>
   ```

   Where the **ssl** scheme selects the OpenWire/SSL protocol and **localhost:61617** should be customized to the host and port value of the remote broker's secure endpoint.

3. Specify the location and password for a trust store file, which will be used to verify the broker's certificate during the SSL/TLS handshake. Add the following property settings to the **activemq-rar.rar** resource adapter configuration and the **org.apache.activemq.ra.ActiveMQManagedConnectionFactory** connection definition configuration. Customize the settings as appropriate:

   ```
   <resource-adapter id="activemq-rar.rar">
       ...
       <config-property name="TrustStore">
   ```

```
        /opt/apache-activemq-5.11.0.redhat-630187/conf/truststore.ks
    </config-property>
    <config-property name="TrustStorePassword">
        password
    </config-property>
    ...
    <connection-definitions>
       <connection-definition class-
name="org.apache.activemq.ra.ActiveMQManagedConnectionFactory"
             jndi-name="java:/ConnectionFactory" enabled="true"
tracking="false"
             pool-name="ConnectionFactory">
             ...
           <config-property name="TrustStore">
               /opt/apache-activemq-5.11.0.redhat-
630187/conf/truststore.ks
           </config-property>
           <config-property name="TrustStorePassword">
               password
           </config-property>
           ...
      </connection-definition>
    </connection-definitions>
  </resource-adapter>
  ...
```

4. *(Optional)* If the remote broker has also been configured to require a client certificate, then it is necessary to configure the ActiveMQ resource adapter with a key store file containing a public/private key pair. In this case, you need to add the following property settings to the **activemq-rar.rar** resource adapter configuration and the **org.apache.activemq.ra.ActiveMQManagedConnectionFactory** connection definition configuration. Customize the settings as appropriate:

```
<resource-adapter id="activemq-rar.rar">
    ...
    <config-property name="KeyStore">
        /opt/apache-activemq-5.11.0.redhat-630187/conf/keystore.ks
    </config-property>
    <config-property name="KeyStorePassword">
        password
    </config-property>
    ...
      <connection-definitions>
          <connection-definition class-
name="org.apache.activemq.ra.ActiveMQManagedConnectionFactory"
                jndi-name="java:/ConnectionFactory" enabled="true"
tracking="false"
                pool-name="ConnectionFactory">
              ...
              <config-property name="KeyStore">
                /opt/apache-activemq-5.11.0.redhat-
630187/conf/keystore.ks
              </config-property>
              <config-property name="KeyStorePassword">
```

```
                password
              </config-property>
          </connection-definition>
      </connection-definitions>
    ...
</resource-adapter>
```

NOTE

In order for the resource adapter's certificate (the public key from the key store) to be accepted by the broker it is necessary for this certificate to be trusted by the broker. This can be achieved either by adding the resource adapter's certificate to the broker's trust store or by arranging to have the resource adapter's certificate signed by one of the certificates in the broker's trust store. For more background information on certificate signing and trust, see appendix "Managing Certificates" in "Security Guide".

Resource adapter configuration

In the configuration shown in Example 1.1, “ActiveMQ Resource Adapter Configuration in standalone-full.xml”, you use the **`config-property`** element to set resource adapter properties. The ActiveMQ resource adapter supports the following basic properties:

`UserName`

(Optional) Specifies the client username when connecting to the JBoss A-MQ broker (not required in this example, because the JBoss A-MQ broker configuration does not enable authentication).

`Password`

(Optional) Specifies the client password when connecting to the JBoss A-MQ broker (not required in this example, because the JBoss A-MQ broker configuration does not enable authentication).

`ServerUrl`

Specifies the URL used to connect to the JBoss A-MQ broker instance. This value must match one of the endpoints specified by a **`transportConnector`** element in the JBoss A-MQ broker configuration.

`BrokerXmlConfig`

(Optional) Specifies the location of an embedded broker's XML configuration file. To specify a location on the file system, use the format, **`xbean:file://`*****`AbsolutePath`***, where the path, ***`AbsolutePath`***, should be specified as an absolute pathname.

NOTE

This property is used to run an embedded ActiveMQ broker within resource adapter. Please note that running an embedded ActiveMQ broker in JBoss EAP is currently *not* supported.

`UseInboundSession`

(Optional) Sets a flag that specifies whether outbound connections should reuse the inbound connection's session for sending messages (useful for connections going through a firewall). Defaults to **`false`**.

`Clientid`

(Optional) Specifies a JMS client ID, which the resource adapter uses for the connection from the JBoss EAP container.

`TrustStore`

Specifies the location of a *trust store* file, using an absolute pathname. The trust store is required, if the ActiveMQ resource adapter connects to the broker using an SSL/TLS protocol. The trust store is provided in the form of a Java keystore file, which can contain one or multiple trusted certificates.

`TrustStorePassword`

Specifies the password for accessing the trust store file.

`KeyStore`

Specifies the location of a *key store* file, using an absolute pathname. The key store is required, if the ActiveMQ resource adapter connects to the broker using and SSL/TLS protocol *and* the remote broker is also configured to require a client certificate. The key store is provided in the form of a Java keystore file, which contains a single public/private key pair.

`KeyStorePassword`

Specifies the password that is used both for accessing the key store file and for unencrypting the private key entry (it is a common convention to use the same password for locking the keystore file and for encrypting the private key entry).

JBoss A-MQ broker configuration

Most of the options for customizing the ActiveMQ broker are provided by the JBoss A-MQ broker configuration file, at the following location:

```
InstallDir/etc/activemq.xml
```

This configuration file supports a huge range of features and settings which are beyond the scope of this guide. To learn more about JBoss A-MQ broker configuration, see the following guides from the *Red Hat JBoss A-MQ* documentation library:

- *Configuring Broker Persistence*
- *Tuning Guide*
- *Security Guide*
- *XML Configuration Reference*

1.3. INTEGRATING WITH AN ACTIVEMQ FAILOVER CLUSTER

Overview

This section describes how to configure the ActiveMQ resource adapter to connect to an ActiveMQ failover cluster (for example, a high-availability master/slave cluster). For details about how to set up and configure such a cluster, see "Fault Tolerant Messaging".

Failover URL

To connect to a cluster of JBoss A-MQ brokers (for example, a master/slave pair of brokers), you need to configure the **`ServerUrl`** configuration property with a failover URL, which lists the available endpoints in the cluster. The general form of the failover URL you should use is as follows:

```
failover:(uri1,...,uriN)?
maxReconnectAttempts=0&reconnectSupported=false&updateURIsSupported=false&
priorityBackup=false
```

NOTE

It is important to set the options as shown, in order to ensure a clean cutover when the master fails in a master/slave high-availability cluster. You do not want a failover client to attempt reconnection to the same endpoint. The client should always try the next URI in the failover list.

Sample scenario

Consider the scenario where a broker running on host **`amqhostA`** and a broker running on host **`amqhostB`** are configured to run as a high-availability master/slave cluster. In this scenario, the brokers expose the following TCP endpoints:

```
tcp://amqhostA:61616
tcp://amqhostB:61616
```

To connect to this cluster, the resource adapter should be configured with the following failover URL:

```
failover:(tcp://amqhostA:61616,tcp://amqhostB:61616)?
jms.rmIdFromConnectionId=true&maxReconnectAttempts=0&reconnectSupported=fa
lse&updateURIsSupported=false&priorityBackup=false
```

When setting the URL in an XML file, you must remember to escape the **`&`** symbol as **`&amp;`** giving the URL:

```
failover:(tcp://amqhostA:61616,tcp://amqhostB:61616)?
jms.rmIdFromConnectionId=true&amp;maxReconnectAttempts=0&amp;reconnectSupp
orted=false&amp;updateURIsSupported=false&amp;priorityBackup=false
```

Configuring the ActiveMQ resource adapter for failover

To configure the ActiveMQ resource adapter to connect to an ActiveMQ failover cluster, you must modify the following configuration settings:

- Set the **`ServerUrl`** configuration property to a correctly configured failover URL,
- Set the **`UseInboundSession`** configuration property to **`true`** for inbound connections (set as the direct child of the **`resource-adapter`** element), and
- Set the **`UseInboundSession`** configuration property to **`false`** for the connection factories (set as the child of a **`connection-definition`** element).

Open the ***`EAPInstallDir`*`/standalone/configuration/standalone-full.xml`** file using a text editor, search for the **`urn:jboss:domain:resource-adapters:1.1`** subsystem, and modify the **`ServerUrl`** property and the **`UseInboundSession`** property as shown in Example 1.3, "ActiveMQ Resource Adapter Configuration for Failover". You will need to customize the value of the failover URL, as appropriate, to match the configuration of your broker cluster.

Example 1.3. ActiveMQ Resource Adapter Configuration for Failover

```
<server xmlns="urn:jboss:domain:1.4">
    ...
    <profile>
        ...
        <subsystem xmlns="urn:jboss:domain:resource-adapters:1.1">
            <resource-adapters>
                <resource-adapter id="activemq-rar.rar">
                    ...
                    <config-property name="ServerUrl">
                        failover:
(tcp://amqhostA:61616,tcp://amqhostB:61616)?
jms.rmIdFromConnectionId=true&amp;maxReconnectAttempts=0
                    </config-property>
                    ...
                    <config-property name="UseInboundSession">
                        true
                    </config-property>
                    ...
                    <connection-definitions>
                        <connection-definition ... >
                            ...
                            <config-property name="UseInboundSession">
                                false
                            </config-property>
                            ...
                        </connection-definition>
                    </connection-definitions>
                    ...
                </resource-adapter>
                ...
            </resource-adapters>
        </subsystem>
        ...
    </profile>
    ...
</server>
```

1.4. INSTALL JBOSS AS QUICKSTARTS

Overview

The JBoss AS Quickstart examples consists of a collection of demonstrations that illustrate features of the JBoss EAP. The installation consists of the following parts:

- *JBoss EAP Maven repository*—an offline Maven repository for JBoss EAP, which contains the dependencies required by the quickstart examples.

- *JBoss AS Quickstart examples*—the quickstart examples themselves.

Prerequisites

To download, install, and build the JBoss AS Quickstart examples, you need the following prerequisites:

- *Subscription*—you must have a Red Hat subscription that includes support for the JBoss EAP product (or ask Red Hat support for access as part of an evaluation).
- *Maven installation*—you must have Apache Maven installed and the version must be 3.3 or later. You can get the latest copy of Maven from the Maven download page.
- *Internet access*—Maven is a distributed build system, which downloads packages from the Internet on the fly, whenever they are needed during a build. Consequently, you must have access to the Internet while performing a Maven build.

JBoss AS Quickstarts download location

You can download the JBoss AS Quickstart examples from the Quickstarts download page on the Red Hat Customer Portal site. Click the following link to download the **`jboss-eap-6.4.0-quickstarts.zip`** file:

- JBoss EAP 6.4.0 Quickstarts

NOTE

After following this link, you will be prompted to log on to the Red Hat customer access portal. If you do not have a subscription for JBoss EAP you will not be able to access this download, however.

Maven repository download location

The JBoss EAP Maven repository is required in order to run the quickstart examples.

You can download the Maven repository from the Maven Repository download page on the Red Hat Customer Portal site. Click the following link to download the **`jboss-eap-6.4.0-maven-repository.zip`** file:

- JBoss EAP 6.4.0 Maven Repository

Steps to install JBoss AS Quickstarts

To install the JBoss AS Quickstart examples, perform the following steps:

1. Download the **`jboss-eap-6.4.0-quickstarts.zip`** file from the customer portal site. Use an archive utility to unzip the downloaded file at a convenient location on your filesystem, ***`QuickInstallDir`***.
2. Download the **`jboss-eap-6.4.0-maven-repository.zip`** file from the customer portal site. Use an archive utility to unzip the downloaded file at a convenient location on your filesystem, ***`MvnRepoInstallDir`***.

NOTE

It is essential to download and install the Maven repository on your local machine. The quickstart examples require Maven artifacts that are *not* available from any public repositories online. You will not be able to build the quickstart examples unless you download, install, and configure the Maven repository.

3. Configure Maven to use the downloaded Maven repository by editing your local repository's **`settings.xml`** file (usually located at **`~/.m2/settings.xml`** on Linux and UNIX systems, or at **`C:\Documents and Settings\`*`Username`*`\.m2\settings.xml`** on Windows). Open the **`settings.xml`** file with a text editor and add the following profiles:

```
<?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 EAP Maven repository -->
    <profile>
      <id>jboss-eap-repository</id>
      <repositories>
        <repository>
          <id>jboss-eap-repository</id>
          <url>file:///path/to/jboss-eap-6.4.0.GA-maven-
repository</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>jboss-eap-plugin-repository</id>
          <url>file:///path/to/jboss-eap-6.4.0.GA-maven-
repository</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
        </pluginRepository>
      </pluginRepositories>
    </profile>

    <!-- Configure the JBoss Community Maven repository -->
    <profile>
      <id>jboss-community-repository</id>
      <repositories>
        <repository>
          <id>jboss-community-repository</id>

<url>http://repository.jboss.org/nexus/content/groups/public/</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>jboss-community-plugin-repository</id>

<url>http://repository.jboss.org/nexus/content/groups/public/</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
        </pluginRepository>
      </pluginRepositories>
    </profile>

  </profiles>

  <activeProfiles>
    <!-- Optionally, make the repositories active by default -->
    <activeProfile>jboss-eap-repository</activeProfile>
    <activeProfile>jboss-community-repository</activeProfile>
  </activeProfiles>

</settings>
```

NOTE

Alternatively, there is a sample settings file provided at ***MvnRepoInstallDir*/example-settings.xml** in the downloaded Maven repository, which you can use as a template for defining your **settings.xml** file.

4. Replace all occurrences of **file:///path/to/jboss-eap-6.4.0.GA-maven-repository** in the **settings.xml** file with the actual location of the Maven repository on your filesystem, ***MvnRepoInstallDir***.

Test the installation

To test the installation of the quickstart examples, try to build the **helloworld-mdb** example using Maven. Open a new command window, change directory to ***QuickInstallDir*/helloworld-mdb**, and enter the following command:

```
mvn clean package
```

If the project builds successfully, you should see a **`BUILD SUCCESS`** status and the generated **`jboss-as-helloworld-mdb.war`** package will be found under the ***`QuickInstallDir`*****`/target`** directory.

If the project does not build successfully, make sure that you have access to the Internet and check that the Maven **`settings.xml`** file is correctly configured.

1.5. BUILD AND DEPLOY THE HELLOWORLD-MDB EXAMPLE

Overview

In this tutorial, you will customize the **`helloworld-mdb`** quickstart example so that it works with the ActiveMQ resource adapter. You can then build and deploy the **`helloworld-mdb`** example into a standalone instance of JBoss Enterprise Application Platform (which already has an ActiveMQ resource adapter installed).

The **`helloworld-mdb`** example illustrates two kinds of integration with messaging: integration of message-driven beans; and integration of a servlet (which gets access to a JMS queue and a JMS topic using the standard JMS administered objects mechanism).

Prerequisites

The following prerequisites must be satisfied, before you can build and deploy the **`helloworld-mdb`** example:

- The ActiveMQ resource adapter is installed in JBoss EAP (as described in Section 1.2, "Install the ActiveMQ Resource Adapter"), and the installation has been verified.
- The JBoss EAP Maven repository and the JBoss AS Quickstart examples have been installed (as described in Section 1.4, "Install JBoss AS Quickstarts").
- You have Internet access (for the Maven build).

Customizations

The version of the **`helloworld-mdb`** demonstration provided in the quickstarts distribution is integrated with the HornetQ messaging platform by default. To refactor the demonstration so that it integrates with Apache ActiveMQ, it is necessary to customize the following aspects of the **`helloworld-mdb`** code:

- Delete the HornetQ XML configuration file (located in **`helloworld-mdb/src/main/webapp/WEB-INF/hornetq-jms.xml`**).
- In **`HelloWorldQueueMDB.java`**, customize the annotations on the message driven bean to integrate with the ActiveMQ resource adapter.
- In **`HelloWorldTopicMDB.java`**, customize the annotations on the message driven bean to integrate with the ActiveMQ resource adapter.
- Add additional Maven dependencies.

These customizations are described in more detail in the rest of this section.

Steps to build and deploy the example

To build and deploy the quickstart **helloworld-mdb** example, perform the following steps:

1. Delete the following HornetQ XML configuration file from the **helloworld-mdb** project:

   ```
   helloworld-mdb/src/main/webapp/WEB-INF/hornetq-jms.xml
   ```

2. Edit the annotations on the **HelloWorldQueueMDB** message driven bean class, so that it integrates with the ActiveMQ resource adapter (instead of HornetQ). Edit the **HelloWorldQueueMDB.java** file at the following location:

   ```
   helloworld-
   mdb/src/main/java/org/jboss/as/quickstarts/mdb/HelloWorldQueueMDB.ja
   va
   ```

   Open the **HelloWorldQueueMDB.java** file using a text editor and make the modifications highlighted in the following extract:

   ```
   import org.jboss.ejb3.annotation.ResourceAdapter;
   ...
   @MessageDriven(name = "HelloWorldQueueMDB", activationConfig = {
     @ActivationConfigProperty(propertyName = "destinationType",
   propertyValue = "javax.jms.Queue"),
     @ActivationConfigProperty(propertyName = "destination",
   propertyValue = "HELLOWORLDMDBQueue"),
     @ActivationConfigProperty(propertyName = "acknowledgeMode",
   propertyValue = "Auto-acknowledge") })
   @ResourceAdapter(value="activemq-rar.rar")
   public class HelloWorldQueueMDB implements MessageListener {
       ...
   ```

   Where the following changes are made to the code:

   - The **@ResourceAdapter** annotation explicitly associates the message driven bean with the ActiveMQ resource adapter. You *must* include this annotation, if you want to use the ActiveMQ resource adapter.
   - You need to add an **import** statement for the **@ResourceAdapter** annotation.
   - The value of the **destination** property is changed to **HELLOWORLDMDBQueue**, which is the *physical name* of the corresponding ActiveMQ queue that this message driven bean subscribes to. The physical name of the queue is the queue identifier used by the JBoss A-MQ broker.

   NOTE

   You must specify the queue's physical name here. In contrast to the case of HornetQ, the ActiveMQ messaging integration does *not* allow you to use a JNDI name for the **destination** value.

3. Edit the annotations on the **HelloWorldTopicMDB** message driven bean class, so that it integrates with the ActiveMQ resource adapter (instead of HornetQ). Edit the **HelloWorldTopicMDB.java** file at the following location:

```
helloworld-
mdb/src/main/java/org/jboss/as/quickstarts/mdb/HelloWorldTopicMDB.ja
va
```

Open the **HelloWorldTopicMDB.java** file using a text editor and make the modifications highlighted in the following extract:

```
import org.jboss.ejb3.annotation.ResourceAdapter;
...
@MessageDriven(name = "HelloWorldQTopicMDB", activationConfig = {
  @ActivationConfigProperty(propertyName = "destinationType",
propertyValue = "javax.jms.Topic"),
  @ActivationConfigProperty(propertyName = "destination",
propertyValue = "HELLOWORLDMDBTopic"),
  @ActivationConfigProperty(propertyName = "acknowledgeMode",
propertyValue = "Auto-acknowledge") })
@ResourceAdapter(value="activemq-rar.rar")
public class HelloWorldTopicMDB implements MessageListener {
    ...
```

4. Add the Maven dependency for the **jboss-ejb3-ext-api** artifact, which is needed for the **@ResourceAdapter** annotation. Open the **helloworld-mdb/pom.xml** file using a text editor and add the following **dependency** element as a child of the **dependencies** element in the POM file:

```
<project ...>
    ...
    <dependencies>
        ...
        <dependency>
            <groupId>org.jboss.ejb3</groupId>
            <artifactId>jboss-ejb3-ext-api</artifactId>
            <!-- to get the right version, look in the EAP offline
Maven repo -->
            <version>2.1.0.redhat-1</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
    ...
</project>
```

The **provided** scope value implies there is no need to embed the **jboss-ejb3-ext-api** JAR file in the generated WAR, because this library is already provided by the JBoss EAP container.

NOTE

If you ever need to update the version of the **jboss-ejb3-ext-api** artifact, you can discover which version to use by drilling down to the **org/jboss/ejb3/jboss-ejb3-ext-api** subdirectory of the Maven repository you downloaded and installed in Section 1.4, “Install JBoss AS Quickstarts”.

5. Build the **helloworld-mdb** example as follows. Open a new command prompt, change directory to the **helloworld-mdb** directory, and enter the following Maven command:

```
mvn clean package
```

If the build is successful, you should find the **jboss-as-helloworld-mdb.war** WAR file in the **helloworld-mdb/target** directory.

6. If you have not already done so, register the administered objects for the queues and topics used by the example, by editing the JBoss EAP configuration.

   In your JBoss EAP installation, open the **standalone/configuration/standalone-full.xml** configuration file with a text editor, and add the following highlighted administered objects to the **activemq-rar.rar** resource adapter:

```
<server xmlns="urn:jboss:domain:1.4">
    ...
    <profile>
        ...
        <subsystem xmlns="urn:jboss:domain:resource-adapters:1.1">
            <resource-adapters>
                <resource-adapter id="activemq-rar.rar">
                    ...
                    <admin-objects>
                        <admin-object
                              class-
name="org.apache.activemq.command.ActiveMQQueue"
                              jndi-
name="java:/queue/HELLOWORLDMDBQueue"
                              use-java-context="true"
                              pool-name="HELLOWORLDMDBQueue">
                            <config-property name="PhysicalName">
                                HELLOWORLDMDBQueue
                            </config-property>
                        </admin-object>
                        <admin-object
                              class-
name="org.apache.activemq.command.ActiveMQTopic"
                              jndi-
name="java:/topic/HELLOWORLDMDBTopic"
                              use-java-context="true"
                              pool-name="HELLOWORLDMDBTopic">
                            <config-property name="PhysicalName">
                                HELLOWORLDMDBTopic
                            </config-property>
                        </admin-object>
                    </admin-objects>
                </resource-adapter>
            </resource-adapters>
        </subsystem>
        ...
    </profile>
    ...
</server>
```

   Where the preceding configuration adds the following entries to the JNDI registry:

   java:/queue/HELLOWORLDMDBQueue

References a **javax.jms.Queue** object that connects to the **HELLOWORLDMDBQueue** ActiveMQ queue (where the queue name on the JBoss A-MQ broker is specified by the **PhysicalName** config property).

java:/queue/HELLOWORLDMDBTopic

References a **javax.jms.Topic** object that connects to the **HELLOWORLDMDBTopic** ActiveMQ topic (where the topic name on the JBoss A-MQ broker is specified by the **PhysicalName** config property).

In the **helloworld-mdb** example, these administered objects are accessed by the servlet class, **HelloWorldMDBServletClient** (but *not* by the message driven bean classes). For example, the **HelloWorldMDBServletClient** class injects these JNDI entries, using the **@Resource** annotation, as follows:

```
import javax.annotation.Resource;
...
import javax.jms.ConnectionFactory;
...
import javax.jms.Queue;
import javax.jms.Topic;
...
public class HelloWorldMDBServletClient extends HttpServlet {
    ...
 @Resource(mappedName = "java:/ConnectionFactory")
 private ConnectionFactory connectionFactory;

 @Resource(mappedName = "java:/queue/HELLOWORLDMDBQueue")
 private Queue queue;

 @Resource(mappedName = "java:/topic/HELLOWORLDMDBTopic")
 private Topic topic;
    ...
```

7. Deploy the **helloworld-mdb** example to the running Web server. Manually copy the **jboss-as-helloworld-mdb.war** WAR file from the **helloworld-mdb/target** directory to the Web server's deployment directory, **standalone/deployments**.

Access the helloworld-mdb service

You can now test the **helloworld-mdb** service, as follows:

1. If you have not already started the JBoss EAP standalone container, do so by entering the following commands at the command line:

```
cd EAPInstallDir/bin
./standalone.sh
```

2. You should now be able to access the **helloworld-mdb** service from your browser, by navigating to the following URL:

```
http://localhost:8080/jboss-helloworld-
mdb/HelloWorldMDBServletClient
```

When you navigate to the preceding URL, the servlet sends five messages to the **`HelloWorldQueueMDB`** message-driven bean. If you look at the container console window, you should see some output like the following:

```
14:41:20,739 INFO  [class
org.jboss.as.quickstarts.mdb.HelloWorldQueueMDB] (default-threads -
7) Received Message from queue: This is message 5
14:41:20,739 INFO  [class
org.jboss.as.quickstarts.mdb.HelloWorldQueueMDB] (default-threads -
5) Received Message from queue: This is message 3
14:41:20,739 INFO  [class
org.jboss.as.quickstarts.mdb.HelloWorldQueueMDB] (default-threads -
4) Received Message from queue: This is message 2
14:41:20,741 INFO  [class
org.jboss.as.quickstarts.mdb.HelloWorldQueueMDB] (default-threads -
6) Received Message from queue: This is message 4
14:41:20,742 INFO  [class
org.jboss.as.quickstarts.mdb.HelloWorldQueueMDB] (default-threads -
3) Received Message from queue: This is message 1
```

These console messages are written by the **`HelloWorldQueueMDB`** message-driven bean, thus demonstrating that the messages have successfully propagated from the servlet, through the JBoss A-MQ broker, to the message-driven bean.

3. To send messages to the **`HelloWorldTopicMDB`** message-driven bean, navigate to the following URL:

```
http://localhost:8080/jboss-helloworld-
mdb/HelloWorldMDBServletClient?topic
```

When you navigate to the preceding URL, the servlet sends five messages to the **`HelloWorldTopicMDB`** message-driven bean. If you look at the container console window, you should see some output like the following:

```
14:53:02,464 INFO  [class
org.jboss.as.quickstarts.mdb.HelloWorldTopicMDB] (default-threads -
9) Received Message from topic: This is message 2
14:53:02,466 INFO  [class
org.jboss.as.quickstarts.mdb.HelloWorldTopicMDB] (default-threads -
10) Received Message from topic: This is message 3
14:53:02,468 INFO  [class
org.jboss.as.quickstarts.mdb.HelloWorldTopicMDB] (default-threads -
8) Received Message from topic: This is message 1
14:53:02,471 INFO  [class
org.jboss.as.quickstarts.mdb.HelloWorldTopicMDB] (default-threads -
11) Received Message from topic: This is message 4
14:53:02,472 INFO  [class
org.jboss.as.quickstarts.mdb.HelloWorldTopicMDB] (default-threads -
12) Received Message from topic: This is message 5
```

These console messages are written by the **`HelloWorldTopicMDB`** message-driven bean, thus demonstrating that the messages have successfully propagated from the servlet, through the JBoss A-MQ broker, to the message-driven bean.