Appendix A - Deploying to JBoss EAP locally

What Will You Learn Here?

This appendix demonstrates how to import, develop and deploy the TicketMonster example using JBoss Developer Studio:

- Obtain and import the TicketMonster example source code
- Deploy the application to JBoss EAP with JBoss Server Tools

Pre-requisites

We don't recommend using the **Internal Web Browser**, although it is configured as the default web browser in the IDE. In certain environments, it may lack features present in modern web browsers, thus providing a sub-optimal user and developer experience.

We shall therefore set the IDE default web browser to be your default system web browser. Click **Window → Web Browser → Default system web browser**.

Import the Project source code

Once the TicketMonster source code is obtained and unpackaged, you must import it into JBoss Developer Studio, as detailed in the procedure below. TicketMonster is a Maven-based project so a specific Import Maven Project wizard is used for the import.

1. Click **File → Import** to open the **Import** wizard.
2. Expand **Maven**, select **Existing Maven Projects** and click **Next**.
3. In the **Root Directory** field, enter the path to the TicketMonster source code. Alternatively, click **Browse** to navigate to the source code location. The **Import Maven Project** wizard recursively searches the path for a **pom.xml** file. The **pom.xml** file identifies the project as a Maven project. The file is listed under **Projects** once it is found.

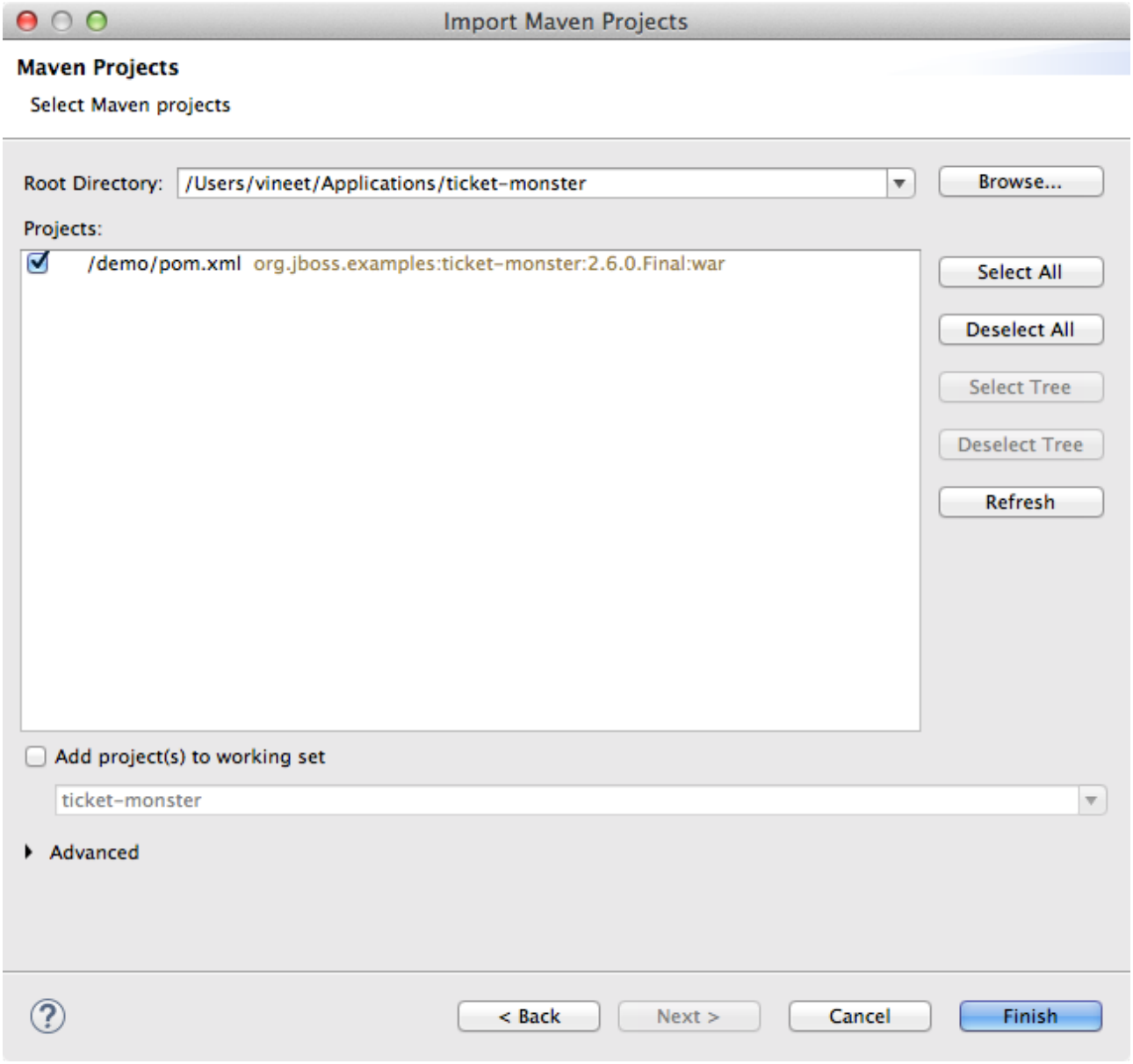

Figure 1. pom.xml File Listed in the Projects Pane

4. Click **Finish**. When the import process is complete, the project is listed in the **Project Explorer** view.

Deploying to JBoss EAP using JBoss Developer Studio

Once you have imported the TicketMonster source code into JBoss Developer Studio, the project application can be deployed to the JBoss EAP server and the running application viewed in the default system web browser, as detailed in the procedure below:

1. In the **Project Explorer** view, right-click **ticket-monster** and click **Run As** → **Run on Server**.
2. Under **How do you want to select the server?**, ensure **Choose an existing server** is selected.
3. In the **Server** table, expand **localhost**, select **jboss-eap-version** where version denotes the JBoss EAP version, and click **Next**.

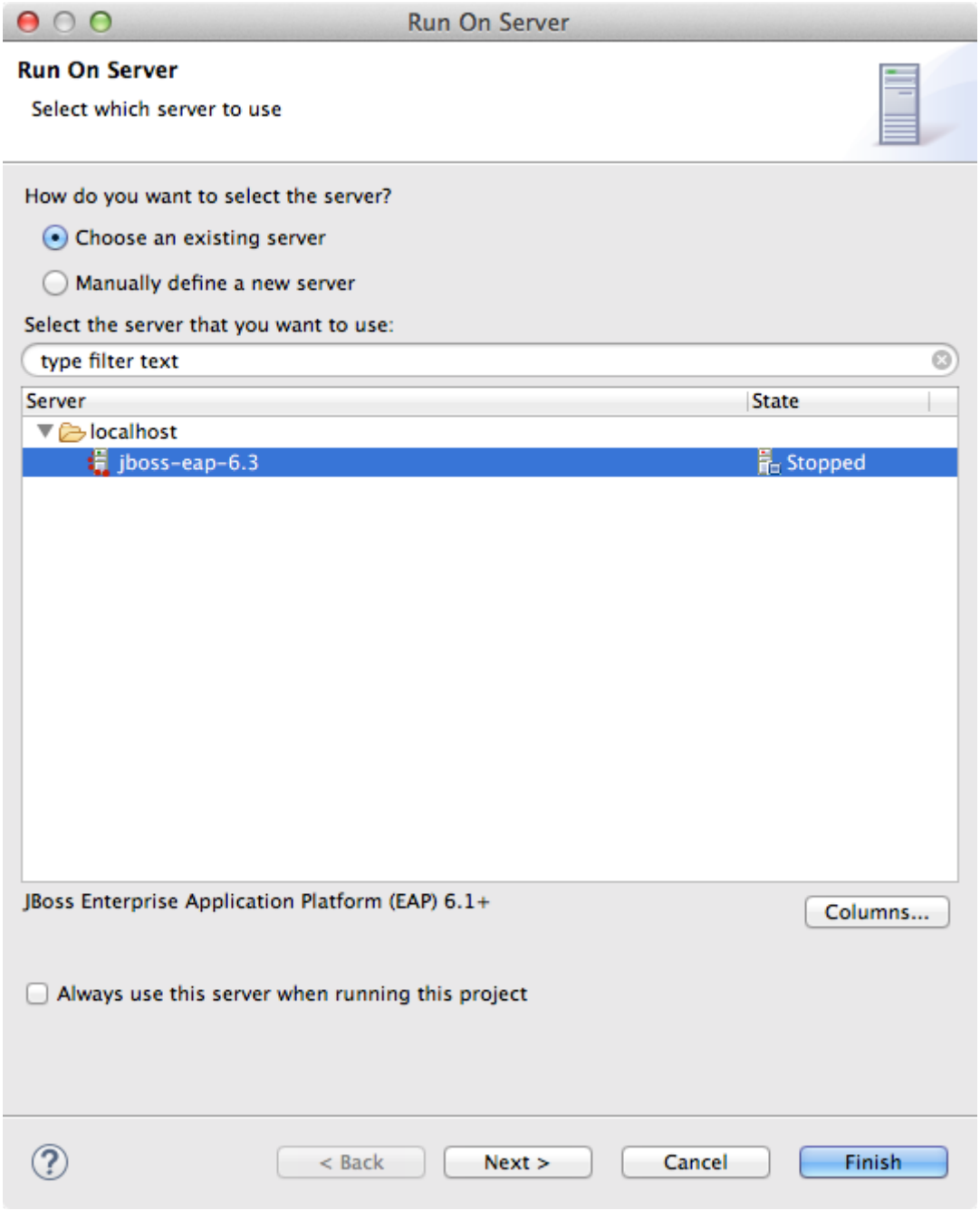

Figure 2. JBoss EAP 6.x Server Selected

4. Ensure **ticket-monster** is listed in the **Configured** column and click Finish. The **Console** view automatically becomes the view in focus and displays the output from the JBoss EAP server. Once deploying is complete, the web application opens in the default system web browser.

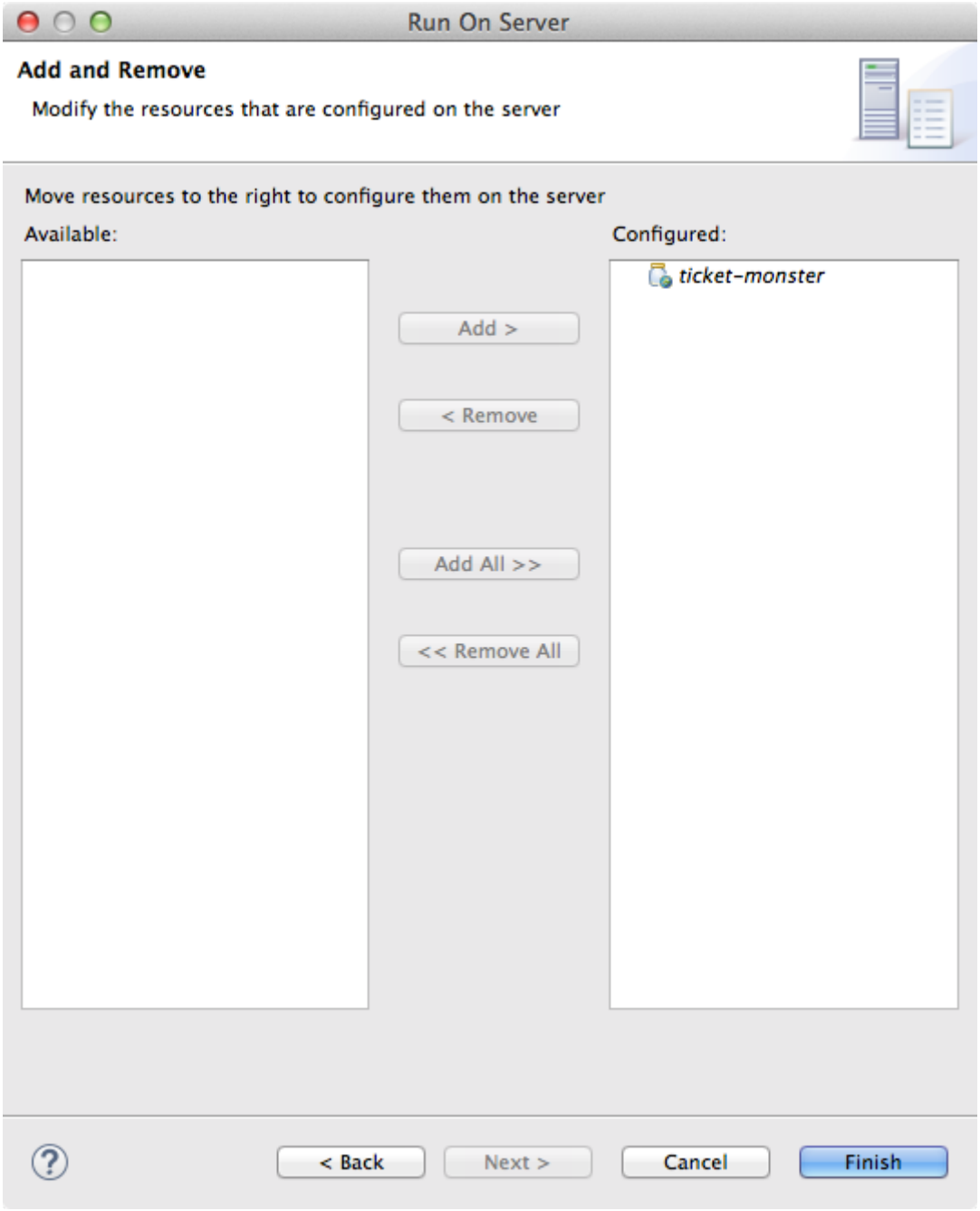

Figure 3. ticket-monster Listed in the Configured Column

Deploying to JBoss EAP using the command-line

Start JBoss Enterprise Application Platform 6.3.

1. Open a command line and navigate to the root of the JBoss server directory.
2. The following shows the command line to start the server with the web profile:

```
For Linux:   JBOSS_HOME/bin/standalone.sh
For Windows: JBOSS_HOME\bin\standalone.bat
```

Then, *deploy TicketMonster*.

1. Make sure you have started the JBoss Server as described above.
2. Type this command to build and deploy the archive into a running server instance.

   ```
   mvn clean package jboss-as:deploy
   ```

   (You can use the `arq-jbossas-remote` profile for running tests as well)

   If you have not configured the Maven settings, to use the Red Hat Enterprise Maven repositories:

   ```
   mvn clean package jboss-as:deploy -s TICKETMONSTER_MAVEN_PROJECT_ROOT/settings.xml
   ```

3. This will deploy `target/ticket-monster.war` to the running instance of the server.
4. Now you can see the application running at http://localhost:8080/ticket-monster.

Using MySQL as the database

You can deploy TicketMonster to JBoss EAP, making use of a 'real' database like MySQL, instead of the default in-memory H2 database. You can follow the procedure outlined as follows:

1. Install the MySQL JBDC driver as a new JBoss module.
   a. Define a new JBoss module named `com.mysql` under the `modules` directory of the JBoss EAP installation. Under the `modules/system/layers/base` directory structure, create a directory named `com`, containing sub-directory named `mysql`, containing a sub-directory named `main`. Place the MySQL JBDC driver in the `main` directory. Finally, define the module via a `module.xml` file with the following contents:

      EAP_HOME/system/layers/base/com/mysql/main/module.xml

      ```
      <module xmlns="urn:jboss:module:1.0" name="com.mysql">
         <resources>
           <resource-root path="mysql-connector-java-5.1.34-bin.jar"/>

         </resources>

         <dependencies>
            <module name="javax.api"/>
            <module name="javax.transaction.api"/>
          </dependencies>
      </module>
      ```

      This module declares the MySQL JDBC driver as a resource (from which to load classes) for the module. It also declares a dependency on the `javax.api` and `javax.transaction.api`

modules, since the JDBC driver depends on classes from these modules. Remember to make corrections to the JBDC driver resource path, if you are using a driver JAR with a different name. The JBoss EAP directory structure should now look like this:

```
modules
+---system
¦   +---layers
¦   ¦    +---base
¦   ¦    ¦    +---com
¦   ¦    ¦    ¦    +---mysql
¦   ¦    ¦    ¦    ¦    +---main
¦   ¦    ¦    ¦    ¦    ¦    +-------module.xml
¦   ¦    ¦    ¦    ¦    ¦    +-------mysql-connector-java-5.1.34-bin.jar
```

2. Register the MySQL datasource used by the application. Edit the server configuration file (`standalone.xml`), to add the datasource definition:

```
<datasources>
   <datasource jndi-name="java:jboss/datasources/TicketMonsterMySQLDS" pool-
name="MySQLDS">
      <connection-url>jdbc:mysql://localhost:3306/ticketmonster</connection-url>
      <driver>com.mysql</driver>
      <transaction-isolation>TRANSACTION_READ_COMMITTED</transaction-isolation>
      <pool>
        <min-pool-size>10</min-pool-size>
        <max-pool-size>100</max-pool-size>
        <prefill>true</prefill>
      </pool>
      <security>
        <user-name>test</user-name>
        <password>test</password>
      </security>
      <statement>
        <prepared-statement-cache-size>32</prepared-statement-cache-size>
        <share-prepared-statements/>
      </statement>
    </datasource>
    <drivers>
      <driver name="com.mysql" module="com.mysql">
        <driver-class>com.mysql.jdbc.Driver</driver-class>
        <xa-datasource-class>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource</xa-
datasource-class>
      </driver>
    </drivers>
</datasources>
```

Replace the values for the `connection-url`, `user-name` and `password` with the correct ones for your environment.

3. Build and deploy the application, using the `mysql` profile defined in the project POM :

   a. In JBoss Developer Studio, you can do this by opening the project's context menu: right-click on the project, click **Maven → Select Maven Profiles...**, and activate the `mysql` profile by selecting it's checkbox. Once you have activated the profile, you can publish the project to a JBoss EAP instance from JBoss Developer Studio in the same manner described previously.

   b. If you are building and deploying from the command-line, activate the `mysql` profile, by specifying it during the build command like so:

   ```
   mvn clean package jboss-as:deploy -Pmysql
   ```

   c. If you have not configured the Maven settings, to use the Red Hat Enterprise Maven repositories:

   ```
   mvn clean package jboss-as:deploy -Pmysql -s
   TICKETMONSTER_MAVEN_PROJECT_ROOT/settings.xml
   ```

Using PostgreSQL as the database

Just like MySQL, you can deploy TicketMonster to JBoss EAP, making use of a 'real' database like PostgreSQL, instead of the default in-memory H2 database. You can follow the procedure outlined as follows:

1. Install the PostgreSQL JBDC driver as a new JBoss module.

   a. Define a new JBoss module named `com.mysql` under the `modules` directory of the JBoss EAP installation. Under the `modules/system/layers/base` directory structure, create a directory named `org`, containing sub-directory named `postgresql`, containing a sub-directory named `main`. Place the PostgreSQL JBDC driver in the `main` directory. Finally, define the module via a `module.xml` file with the following contents:

   EAP_HOME/system/layers/base/com/mysql/main/module.xml

   ```
   <module xmlns="urn:jboss:module:1.0" name="org.postgresql">
      <resources>
        <resource-root path="postgresql-9.3-1102.jdbc4.jar"/>
      </resources>

      <dependencies>
         <module name="javax.api"/>
         <module name="javax.transaction.api"/>
       </dependencies>
   </module>
   ```

   This module declares the PostgreSQL JDBC driver as a resource (from which to load classes) for the module. It also declares a dependency on the `javax.api` and `javax.transaction.api` modules, since the JDBC driver depends on classes from these modules. Remember to make

corrections to the JBDC driver resource path, if you are using a driver JAR with a different name. The JBoss EAP directory structure should now look like this:

```
modules
+---system
¦   +---layers
¦   ¦    +---base
¦   ¦    ¦    +---org
¦   ¦    ¦    ¦    +---postgresql
¦   ¦    ¦    ¦    ¦    +---main
¦   ¦    ¦    ¦    ¦    ¦    +-------module.xml
¦   ¦    ¦    ¦    ¦    ¦    +-------postgresql-9.3-1102.jdbc4.jar
```

2. Register the PostgreSQL datasource used by the application. Edit the server configuration file (`standalone.xml`), to add the datasource definition:

```
<datasources>
   <datasource jndi-name="java:jboss/datasources/TicketMonsterPostgreSQLDS" pool-
name="PostgreSQLDS">
      <connection-url>jdbc:postgresql://localhost:5432/ticketmonster</connection-
url>
      <driver>org.postgresql</driver>
      <transaction-isolation>TRANSACTION_READ_COMMITTED</transaction-isolation>
      <pool>
        <min-pool-size>10</min-pool-size>
        <max-pool-size>100</max-pool-size>
        <prefill>true</prefill>
      </pool>
      <security>
        <user-name>test</user-name>
        <password>test</password>
      </security>
      <statement>
        <prepared-statement-cache-size>32</prepared-statement-cache-size>
        <share-prepared-statements/>
      </statement>
    </datasource>
    <drivers>
      <driver name="org.postgresql" module="org.postgresql">
        <xa-datasource-class>org.postgresql.xa.PGXADataSource</xa-datasource-class>
      </driver>
    </drivers>
</datasources>
```

Replace the values for the `connection-url`, `user-name` and `password` with the correct ones for your environment.

3. Build and deploy the application, using the `postgresql` profile defined in the project POM :

a. In JBoss Developer Studio, you can do this by opening the project's context menu: right-click on the project, click **Maven → Select Maven Profiles...**, and activate the `postgresql` profile by selecting it's checkbox. Once you have activated the profile, you can publish the project to a JBoss EAP instance from JBoss Developer Studio in the same manner described previously.

b. If you are building and deploying from the command-line, activate the `postgresql` profile, by specifying it during the build command like so:

```
mvn clean package jboss-as:deploy -Ppostgresql
```

c. If you have not configured the Maven settings, to use the Red Hat Enterprise Maven repositories:

```
mvn clean package jboss-as:deploy -Ppostgresql -s
TICKETMONSTER_MAVEN_PROJECT_ROOT/settings.xml
```