Red Hat JBoss Enterprise Application Platform 8.0

Getting started with developing applications for JBoss EAP deployment

Get started creating applications for JBoss EAP deployment.

Last Updated: 2024-02-21

Red Hat JBoss Enterprise Application Platform 8.0 Getting started with developing applications for JBoss EAP deployment

Get started creating applications for JBoss EAP deployment.

Abstract

Get started creating applications for JBoss EAP deployment by using Maven as the project management tool.Deploy your applications to JBoss EAP runing on bare metal or on OpenShift Container Platform.

Table of Contents

PROVIDING FEEDBACK ON JBOSS EAP DOCUMENTATION

To report an error or to improve our documentation, log in to your Red Hat Jira account and submit an issue. If you do not have a Red Hat Jira account, then you will be prompted to create an account.

Procedure

1. Click the following link to **create a ticket**.
2. Enter a brief description of the issue in the **Summary**.
3. Provide a detailed description of the issue or enhancement in the **Description**. Include a URL to where the issue occurs in the documentation.
4. Clicking **Submit** creates and routes the issue to the appropriate documentation team.

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message .

The best way to become familiar with a new programming language or a technology is to create a "Hello World" application. You can create a "Hello World" application for JBoss EAP by using Maven as the project management tool.

To create a Hello World application, deploy it and test the deployment, follow these procedures:

Bare metal deployment

- Creating a Maven project for a hello world application
- Creating a hello world servlet
- Deploying an application to a bare metal installation
- Adding the Maven dependencies and profile required for integration tests
- Testing an application deployed on JBoss EAP that is running on bare metal

OpenShift Container Platform deployment

- Creating a Maven project for a hello world application
- Creating a hello world servlet
- Deploying an application to OpenShift Container Platform
- Adding the Maven dependencies and profile required for integration tests
- Testing an application deployed to JBoss EAP on OpenShift Container Platform

CHAPTER 1. CREATING A MAVEN PROJECT FOR A HELLO WORLD APPLICATION

A Maven project contains a **pom.xml** configuration file and has the directory structure required for creating an application. You can configure the **pom.xml** configuration file to add dependencies for your application.

To create a Maven project for a hello world application, follow these procedures:

- Creating a Maven project with **maven-archetype-webapp**
- Defining properties in a Maven project
- Defining the repositories in a Maven project
- Importing the JBoss EAP BOMs as dependency management in a Maven project
- Adding plug-in management in a Maven project
- Verifying a maven project

1.1. CREATING A MAVEN PROJECT WITH MAVEN-ARCHETYPE-WEBAPP

Use the **maven-archetype-webapp** archetype to create a Maven project for building applications for JBoss EAP deployment. Maven provides different archetypes for creating projects based on templates specific to project types. The **maven-archetype-webapp** creates a project with the structure required to develop simple web-applications.

Prerequisites

- You have installed Maven. For more information, see Downloading Apache Maven.

Procedure

1. Set up a Maven project by using the **mvn** command. The command creates the directory structure for the project and the **pom.xml** configuration file.

```
$ mvn archetype:generate                        \
-DgroupId=org.jboss.as.quickstarts              \ (1)
-DartifactId=helloworld                    \ (2)
-DarchetypeGroupId=org.apache.maven.archetypes  \ (3)
-DarchetypeArtifactId=maven-archetype-webapp  \ (4)
-DinteractiveMode=false                   (5)
```

(1) **groupID** uniquely identifies the project.

(2) **artifactID** is the name for the generated **jar** archive.

(3) The **groupID** of **maven-archetype-webapp**.

(4) The **artifactID** of **maven-archetype-webapp**.

(5) Tell Maven to use the supplied parameters rather than starting interactive mode.

2. Navigate to the generated directory.

   ```
   $ cd helloworld
   ```

3. Open the generated **pom.xml** configuration file in a text editor.

4. Remove the content inside the **<project>** section of **pom.xml** configuration file after the **<name>helloworld Maven Webapp</name>** line.
   Ensure that the file looks like this:

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

   <project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
     <modelVersion>4.0.0</modelVersion>

     <groupId>org.jboss.as.quickstarts</groupId>
     <artifactId>helloworld</artifactId>
     <version>1.0-SNAPSHOT</version>
     <packaging>war</packaging>
     <name>helloworld Maven Webapp</name>

   </project>
   ```

   The content was removed because it is not required for the application.

Next steps

- Defining properties in Maven project for a JBoss EAP hello world application .

1.2. DEFINING PROPERTIES IN A MAVEN PROJECT

You can define properties in a Maven **pom.xml** configuration file as place holders for values. Define the value for JBoss EAP server as a property to use the value consistently in the configuration.

Prerequisites

- You have initialized a Maven project.
  For more informartion, see Initializing a Maven project for a JBoss EAP hello world application .

Procedure

- Define a property **<version.server>** as the JBoss EAP version on which you will deploy the configured application.

  ```
  <project>
    ...
    <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
      <maven.compiler.source>11</maven.compiler.source>
      <maven.compiler.target>11</maven.compiler.target>
  ```

```
    <version.server>8.0.0.GA-redhat-00009</version.server>
  </properties>
</project>
```

Next steps

- Defining the repositories in a Maven project .

1.3. DEFINING THE REPOSITORIES IN A MAVEN PROJECT

Define the artifact and plug-in repositories in which Maven looks for artifacts and plug-ins to download.

Prerequisites

- You have initialized a Maven project.
  For more informartion, see Initializing a Maven project for a JBoss EAP hello world application .

Procedure

1. Define the artifacts repository.

```
<project>
  ...
  <repositories>
    <repository>                                    1
      <id>jboss-public-maven-repository</id>
      <name>JBoss Public Maven Repository</name>
      <url>https://repository.jboss.org/nexus/content/groups/public/</url>
      <releases>
        <enabled>true</enabled>
        <updatePolicy>never</updatePolicy>
      </releases>
      <snapshots>
        <enabled>true</enabled>
        <updatePolicy>never</updatePolicy>
      </snapshots>
      <layout>default</layout>
    </repository>
    <repository>                                    2
      <id>redhat-ga-maven-repository</id>
      <name>Red Hat GA Maven Repository</name>
      <url>https://maven.repository.redhat.com/ga/</url>
      <releases>
        <enabled>true</enabled>
        <updatePolicy>never</updatePolicy>
      </releases>
      <snapshots>
        <enabled>true</enabled>
        <updatePolicy>never</updatePolicy>
      </snapshots>
      <layout>default</layout>
    </repository>
  </repositories>
</project>
```

1. The Red Hat GA Maven repository provides all the productized JBoss EAP and other Red Hat artifacts.

2. The JBoss Public Maven Repository provides artifacts such as WildFly Maven plug-ins

2. Define the plug-ins repository.

```
<project>
  ...
  <pluginRepositories>
    <pluginRepository>
      <id>jboss-public-maven-repository</id>
      <name>JBoss Public Maven Repository</name>
      <url>https://repository.jboss.org/nexus/content/groups/public/</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>true</enabled>
      </snapshots>
    </pluginRepository>
    <pluginRepository>
      <id>redhat-ga-maven-repository</id>
      <name>Red Hat GA Maven Repository</name>
      <url>https://maven.repository.redhat.com/ga/</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>true</enabled>
      </snapshots>
    </pluginRepository>
  </pluginRepositories>
</project>
```

Next steps

- Importing the JBoss EAP BOMs dependency management in Maven project .

1.4. IMPORTING THE JBOSS EAP BOMS AS DEPENDENCY MANAGEMENT IN A MAVEN PROJECT

Import the JBoss EAP EE With Tools Bill of materials (BOM) to control the versions of runtime Maven dependencies. When you specify a BOM in the **<dependencyManagement>** section, you do not need to individually specify the versions of the Maven dependencies defined in the **provided** scope.

Prerequisites

- You have initialized a Maven project.
  For more informartion, see Initializing a Maven project for a JBoss EAP hello world application .

Procedure

1. Add a property for the BOM version in the properties section of the **pom.xml** configuration file.

```
<properties>
    ....
    <version.bom.ee>${version.server}</version.bom.ee>
</properties>
```

   The value defined in the property **<version.server>** is used as the value for BOM version.

2. Import the JBoss EAP BOMs dependency management.

```
<project>
    ...
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.jboss.bom</groupId> (1)
                <artifactId>jboss-eap-ee-with-tools</artifactId> (2)
                <version>${version.bom.ee}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>
</project>
```

(1) GroupID of the JBoss EAP-provided BOM.

(2) ArtifactID of the JBoss EAP-provided BOM that provides supported JBoss EAP Java EE APIs plus additional JBoss EAP API JARs and client BOMs, and development tools such as Arquillian.

Next steps

- Adding plug-in management in a Maven project

1.5. ADDING PLUG-IN MANAGEMENT IN A MAVEN PROJECT

Add Maven plug-in management section to the **pom.xml** configuration file to get plug-ins required for Maven CLI commands.

Prerequisites

- You have initialized a Maven project.
  For more informartion, see Initializing a Maven project for a JBoss EAP hello world application .

Procedure

1. Define the versions for **wildfly-maven-plugin** and **maven-war-plugin**, in the **<properties>** section.

```
<properties>
    ...
    <version.plugin.wildfly>4.1.1.Final</version.plugin.wildfly>
    <version.plugin.war>3.3.2</version.plugin.war>
</properties>
```

2. Add **<pluginManagement>** in **<build>** section inside the **<project>** section.

```
<project>
    ...
    <build>
        <pluginManagement>
            <plugins>
                <plugin>    1
                    <groupId>org.wildfly.plugins</groupId>
                    <artifactId>wildfly-maven-plugin</artifactId>
                    <version>${version.plugin.wildfly}</version>
                </plugin>
                <plugin>                  2
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-war-plugin</artifactId>
                    <version>${version.plugin.war}</version>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>
</project>
```

1 You can use the **wildfly-maven-plugin** to deploy an application to JBoss EAP using the **wildfly:deploy** command.

2 You need to manage the war plugin version to ensure compatibility with JDK17+.

Next steps

- Verifying a maven project

1.6. VERIFYING A MAVEN PROJECT

Verify that the Maven project you configured builds.

Prerequisites

- You have defined Maven properties.
  For more information, see Defining properties in a Maven project .
- You have defined Maven repositories.
  For more information, see Defining the repositories in a Maven project .
- You have imported the JBoss EAP Bill of materials (BOMs) as dependency management.
  For more information, see Importing the JBoss EAP BOMs as dependency management in a Maven project.
- You have added plug-in management.

For more information, see Adding plugin management in Maven project for a server hello world application.

Procedure

- Install the Maven dependencies added in the **pom.xml** locally.

  ```
  $ mvn package
  ```

  You get an output similar to the following:

  ```
  ...
  [INFO] ------------------------------------------------------------------------
  [INFO] BUILD SUCCESS
  [INFO] ------------------------------------------------------------------------
  ...
  ```

Next steps

- Creating a hello world servlet

CHAPTER 2. CREATING A HELLO WORLD SERVLET

Create a servlet that returns "Hello world!" when accessed.

In this procedure, *<application_home>* refers to the directory that contains the **pom.xml** configuration file for the application.

Prerequisites

- You have created a Maven project.
  For more information, see Creating a Maven project for a Hello World application .

Procedure

1. Add the required dependency to **pom.xml** configuration file after the **<dependencyManagement>** section.

```
<project>
    ...
    <dependencies>
        <dependency>                                      (1)
            <groupId>jakarta.servlet</groupId>
            <artifactId>jakarta.servlet-api</artifactId>
            <scope>provided</scope>                          (2)
        </dependency>
    </dependencies>
```

(1) **jakarta.servlet-api** dependency provides Jakarta Servlet API.

(2) Define the scope as **provided** so that the dependency is not included in the application. The reason for not including the dependency in the application is that this dependency is managed by the **jboss-eap-ee-with-tools** BOM and such dependencies are are included with JBoss EAP.

NOTE

The dependency is defined without a version because **jboss-eap-ee-with-tools** BOM was imported in the **<dependencyManagement>** section.

2. Navigate to the *<application_home>* directory.

3. Create a directory to store the Java files.

```
$ mkdir -p src/main/java/org/jboss/as/quickstarts/helloworld
```

4. Navigate to the new directory.

```
$ cd src/main/java/org/jboss/as/quickstarts/helloworld
```

5. Create the Servlet **HelloWorldServlet.java** that returns "Hello World!".

```
package org.jboss.as.quickstarts.helloworld;
```

```
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

@WebServlet("/HelloWorld")              1
public class HelloWorldServlet extends HttpServlet {

    static String PAGE_HEADER = "<html><head><title>helloworld</title></head><body>";

    static String PAGE_FOOTER = "</body></html>";

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        resp.setContentType("text/html");
        PrintWriter writer = resp.getWriter();
        writer.println(PAGE_HEADER);
        writer.println("<h1> Hello World! </h1>");
        writer.println(PAGE_FOOTER);
        writer.close();
    }
}
```

1 The **@WebServlet("/HelloWorld")** annotation provides the following information to JBoss EAP:

- This class is a servlet.
- Make the servlet available at the URL "*<application_URL>*/HelloWorld".
  For example, if JBoss EAP is running on the localhost and is available at the default HTTP port, 8080, the URL is **http://localhost:8080/helloworld/HelloWorld**.

6. Navigate to the *<application_home>*/src/main/webapp directory.
   You find the file "index.jsp" that Maven created. This file prints "Hello World!" when you access the application.

7. Update the "index.jsp" file to redirect to the Hello World servlet by replacing its content with the following content:

```
<html>
    <head>
        <meta http-equiv="Refresh" content="0; URL=HelloWorld">
    </head>
</html>
```

8. Navigate to the *<application_home>* directory.

9. Compile and package the application as a web archive (WAR) with the following command:

```
$ mvn package
```

Example output

```
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
...
```

Next steps

- Deploying an application to the server

CHAPTER 3. DEPLOYING AN APPLICATION TO THE SERVER

You can deploy your application on a JBoss EAP server running on bare metal or on OpenShift Container Platform.

To deploy your application on a JBoss EAP server running on bare metal, follow this procedure:

- Deploying an application to a bare metal installation

To deploy your application on a JBoss EAP server running on OpenShift Container Platform, follow these procedures:

- Preparing an application for deployment on OpenShift Container Platform
- Deploying an application to JBoss EAP on OpenShift with Helm

3.1. DEPLOYING AN APPLICATION TO A BARE METAL INSTALLATION

You can deploy an application to JBoss EAP by using the JBoss EAP deploy plug-in.

Prerequisites

- You have created an application.
  For more information, see Creating a hello world servlet .
- JBoss EAP is running.

Procedure

1. Navigate to the application root directory.
   The application root directory contains the **pom.xml** configuration file.
2. Add the following build configuration to the **pom.xml** configuration file in the **<project>** section to define the application archive filename.

```
<build>
   ...
   <finalName>${project.artifactId}</finalName>    1
</build>
```

1 Set the name of the deployment to the project's artifact ID.

3. Build and deploy the application by using the JBoss EAP deploy plug-in.

```
$ mvn package wildfly:deploy
```

Verification

- Navigate to the address **http://localhost:8080/helloworld/** in a browser.
  You are redirected to http://localhost:8080/helloworld/HelloWorld and you get the following message:

```
Hello World!
```

–

Next steps

- Testing an application deployed on JBoss EAP

3.2. DEPLOYING AN APPLICATION TO OPENSHIFT CONTAINER PLATFORM

You can use the source-to-image (S2I) workflow to deploy your applications to JBoss EAP on OpenShift Container Platform. The S2I workflow takes source code from a Git repository and injects it into a container that's based on the language and framework you want to use. After the S2I workflow is completed, the **src** code is compiled, the application is packaged and is deployed to the JBoss EAP server.

3.2.1. Preparing an application for deployment on OpenShift Container Platform

OpenShift Container Platform uses application hosted on a Git repository. To deploy your application on OpenShift, you must first push your application to a Git repository. After that, you can use JBoss EAP helm chart to configure your application deployment.

Prerequisites

- You have created an application.
  For more information, see Creating a Hello World servlet.
- You have created a Git repository.

Procedure

1. Move the application to your local Git repository, if it already is not in it.

   ```
   $ mv -r helloworld/ <your_git_repo>
   ```

2. Define the following property in the **pom.xml** configuration file:

   ```
   <properties>
     ...
     <version.plugin.eap>1.0.0.Final-redhat-00013</version.plugin.eap> 1
   </properties>
   ```

   1 **<version.plugin.eap>** defines the version for JBoss EAP Maven plug-in.

3. Add the JBoss EAP maven plugin to **<pluginManagement>**, in **<build>** section inside the **<project>** section.

   ```
   <project>
     ...
     <build>
       <pluginManagement>
         <plugins>
           ...
           <plugin>
   ```

```
                    <groupId>org.jboss.eap.plugins</groupId>
                    <artifactId>eap-maven-plugin</artifactId>
                    <version>${version.plugin.eap}</version>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>
</project>
```

4. Create a profile "openshift" in the **pom.xml** configuration file.
   This profile defines the plug-ins, feature packs, and layers required for deployment on OpenShift Container Platform.

```
<profiles>
    <profile>
        <id>openshift</id>
        <build>
            <plugins>
                <plugin>
                    <groupId>org.jboss.eap.plugins</groupId>
                    <artifactId>eap-maven-plugin</artifactId>     1
                    <configuration>
                        <channels>
                            <channel>
                                <manifest>
                                    <groupId>org.jboss.eap.channels</groupId>
                                    <artifactId>eap-8.0</artifactId>
                                </manifest>
                            </channel>
                        </channels>
                        <feature-packs>
                            <feature-pack>                 2
                                <location>org.jboss.eap:wildfly-ee-galleon-pack</location>
                            </feature-pack>
                            <feature-pack>
                                <location>org.jboss.eap.cloud:eap-cloud-galleon-pack</location>
                            </feature-pack>
                        </feature-packs>
                        <layers>                        3
                            <layer>cloud-server</layer>
                        </layers>
                        <name>ROOT.war</name>                  4
                    </configuration>
                    <executions>
                        <execution>
                            <goals>
                                <goal>package</goal>
                            </goals>
                        </execution>
                    </executions>
                </plugin>
            </plugins>
        </build>
    </profile>
</profiles>
```

1 **wildfly-maven-plugin** is a JBoss EAP plug-in for provisioning a JBoss EAP instance, with the application deployed, on OpenShift Container Platform.

2 **feature-packs** defines the feature-packs (zipped files that contains features to dynamically provision a server). In this case we need the feature-packs **org.wildfly:wildfly-galleon-pack** and **org.wildfly.cloud:wildfly-cloud-galleon-pack**.

3 **layers** defines the layers (from the configured feature-packs) to include in the provisioned server. Each layer identifies one or more server capabilities that can be installed on its own, or in combination with other layers. In our case we opt for the **cloud-server** layer, which provisions just the basic features of JBoss EAP, well suited for a cloud server.

4 **<name>ROOT.war</name>**: Defines the resulting name of the application's web archive (WAR). If **ROOT.war** is specified then the application is deployed at the root path of the server, otherwise it is deployed at **<name/>** relative path.

5. Verify that the applications compiles.

   ```
   $ mvn package -Popenshift
   ```

6. Push the changes to your repository.

Next steps

- Deploying an application to JBoss EAP on OpenShift with Helm

3.2.2. Deploying an application to JBoss EAP on OpenShift with Helm

Use the JBoss EAP Helm chart to configure and deploy application to JBoss EAP on OpenShift with Helm.

Prerequisites

- You have prepared your application for deployment on OpenShift Container Platform.
  For more information, see Preparing an application for deployment on OpenShift Container Platform.
- You have created a project in OpenShift Container Platform.
  For more information see Working with projects.
- You have installed OpenShift CLI (**oc**)
  For more information, see Installing the OpenShift CLI.
- You are logged in to OpenShift Container Platform from your machine.
  For more information, see Logging in to the OpenShift CLI .
- You have installed helm.
  For more information, see Installing Helm.

Procedure

1. Create a directory called **charts** in the application root directoy and navigate to it. Application root directory is the one that contains **pom.xml** configuration file.

```
$ mkdir charts; cd charts
```

2. Create a file **helm.yaml** with the following content:

```
build:
  uri: https://github.com/<user>/<repository>.git    1
  ref: <branch_name>                                  2
  contextDir: helloworld                              3
deploy:
  replicas: 1                                         4
```

1. Specify the URL for your Git repository that contains the application to deploy on OpenShift Container Platform.
2. Specify the Git branch that contains your application.
3. Specify the directory containing the application.
4. Specify the number of pods to create.

3. Configure the JBoss EAP repository in Helm.
   - If you haven’t added the JBoss EAP repository to Helm before, add it.

     ```
     $ helm repo add jboss-eap https://jbossas.github.io/eap-charts/
     ```

   - If you already have added the JBoss EAP repository to Helm, update it.

     ```
     $ helm repo update jboss-eap
     ```

4. Deploy the application using helm.

   ```
   $ helm install helloworld -f helm.yaml jboss-eap/eap8
   ```

   The deployment can take a few minutes to complete.

Verification

1. Get the URL of the route to the deployment.

   ```
   $ APPLICATION_URL=https://$(oc get route helloworld --template='{{ .spec.host }}') &&
   echo "" &&
   echo "Application URL: $APPLICATION_URL"
   ```

2. Navigate to the "Application URL" in a browser.
   You are redirected to the servlet at path "/HelloWorld" and you get the following message:

   ```
   Hello World!
   ```

Next steps

- Testing an application deployed on JBoss EAP

CHAPTER 4. TESTING AN APPLICATION DEPLOYED ON JBOSS EAP

To ensure that the Hello World application deployed on JBoss EAP is working, you can add integration tests.

To add tests for an application deployed on a JBoss EAP server running on bare metal, follow these procedures:

- Adding the Maven dependencies and profile required for integration tests
- Creating a test class to test an application
- Testing an application deployed on JBoss EAP that is running on bare metal

To add tests for an application deployed on a JBoss EAP server running on OpenShift Container Platform, follow these procedures:

- Adding the Maven dependencies and profile required for integration tests
- Creating a test class to test an application
- Testing an application deployed to JBoss EAP on OpenShift Container Platform

4.1. ADDING THE MAVEN DEPENDENCIES AND PROFILE REQUIRED FOR INTEGRATION TESTS

To create integration tests for your applications, add the required Maven dependencies.

Prerequisites

- You have created a Maven project.
  For more information, see Creating a Maven project for a hello world application .

Procedure

1. Define the following properties in the **pom.xml** configuration file:

```
<properties>
  ...
  <version.plugin.failsafe>3.2.2</version.plugin.failsafe>
</properties>
```

2. Add the dependency required for tests.

```
<project>
  ...
  <dependencies>
    ...
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <scope>test</scope>
```

```
        </dependency>
    </dependencies>
</project>
```

3. Define a profile to add the plug-ins required for integration tests.

```
<project>
    ...
    <profiles>
    ...
        <profile>
            <id>integration-testing</id>
            <build>
                <plugins>
                    <plugin>
                        <groupId>org.apache.maven.plugins</groupId>
                        <artifactId>maven-failsafe-plugin</artifactId> 1
                        <version>${version.plugin.failsafe}</version>
                        <configuration>
                            <includes>
                                <include>**/HelloWorldServletIT</include>           2
                            </includes>
                        </configuration>
                        <executions>
                            <execution>
                                <goals>
                                    <goal>integration-test</goal>
                                    <goal>verify</goal>
                                </goals>
                            </execution>
                        </executions>
                    </plugin>
                </plugins>
            </build>
        </profile>
    </profiles>
</project>
```

1 Maven plug-in for running integration tests.

2 The name of the Java class that tests the application.

Next steps

- Creating a test class to test an application

4.2. CREATING A TEST CLASS TO TEST AN APPLICATION

Create an integration test that verifies that the application is deployed and running on JBoss EAP on OpenShift Container Platform, by checking that the HTTP GET of its web page returns 200 OK.

In this procedure, *<application_home>* refers to the directory that contains the **pom.xml** configuration file for the application.

Prerequisites

- You have deployed your application to JBoss EAP.
  For more information, see Building and deploying an application to the server .
- You have added the Maven dependencies required for JUnit tests.
  For more information, see Adding the Maven dependencies and profile required for integration tests.

Procedure

1. Navigate to the *<application_home>* directory.
2. Create a directory to store the test class.

   ```
   $ mkdir -p src/test/java/org/jboss/as/quickstarts/helloworld
   ```

3. Navigate to the new directory.

   ```
   $ cd src/test/java/org/jboss/as/quickstarts/helloworld
   ```

4. Create a Java class **HelloWorldServletIT.java** that tests the deployment.

   ```
   package org.jboss.as.quickstarts.helloworld;

   import org.junit.Test;
   import java.io.IOException;
   import java.net.URI;
   import java.net.URISyntaxException;
   import java.net.http.HttpClient;
   import java.net.http.HttpRequest;
   import java.net.http.HttpResponse;
   import java.time.Duration;
   import static org.junit.Assert.assertEquals;

   public class HelloWorldServletIT {

       private static final String DEFAULT_SERVER_HOST = "http://localhost:8080/helloworld";
   1

       @Test
       public void testHTTPEndpointIsAvailable() throws IOException, InterruptedException,
   URISyntaxException {
           String serverHost = System.getProperty("server.host");
           if (serverHost == null) {
               serverHost = DEFAULT_SERVER_HOST;
           }
           final HttpRequest request = HttpRequest.newBuilder()
                   .uri(new URI(serverHost+"/HelloWorld"))
                   .GET()
                   .build();                                                    2
           final HttpClient client = HttpClient.newBuilder()
                   .followRedirects(HttpClient.Redirect.ALWAYS)
                   .connectTimeout(Duration.ofMinutes(1))
                   .build();                                                    3
   ```

```
        final HttpResponse<String> response = client.send(request,
HttpResponse.BodyHandlers.ofString()); 4
        assertEquals(200, response.statusCode());                              5
    }
}
```

1. The URL at which the application is running. This value is used if **sever.host** is undefined.
2. Create an HttpRequest instance for the application URI.
3. Create an HttpClient to send requests to and receive response from the application.
4. Get response from the application.
5. Test that the response revieved from the application is "200" indicating that the application is rechable.

Next steps

- To test an application deployed on a JBoss EAP server running on bare metal, follow this procedure:
  - Testing an application deployed on JBoss EAP that is running on bare metal
- To test an application deployed on a JBoss EAP server running on OpenShift Container Platform, follow this procedure:
  - Testing an application deployed to JBoss EAP on OpenShift Container Platform

4.3. TESTING AN APPLICATION DEPLOYED ON JBOSS EAP THAT IS RUNNING ON BARE METAL

Test the application deployed on JBoss EAP that is running on bare metal.

Prerequisites

- You have created a test class.
  For more information, see Creating a test class to test an application
- The application to test is deployed on JBoss EAP.
- JBoss EAP is running.

Procedure

1. Navigate to the *<application_home>* directory.
2. Run the integration test by using the **verify** command with the **integration-testing** profile.

   ```
   $ mvn verify -Pintegration-testing
   ```

   Example output

```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- maven-failsafe-plugin:3.2.2:verify (default) @ helloworld ---
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  9.982 s
[INFO] Finished at: 2023-11-22T14:53:54+05:30
[INFO] ------------------------------------------------------------------------
```

4.4. TESTING AN APPLICATION DEPLOYED TO JBOSS EAP ON OPENSHIFT CONTAINER PLATFORM

Test the application deployed to JBoss EAP on OpenShift Container Platform.

Prerequisites

- You have created a test class.
  For more information, see Creating a test class to test an application

Procedure

1. Push the changes to your Git repository.
2. Navigate to the *<application_home>* directory.
3. Run the test by using the **verify** command, activating the **integration-testing** profile and specifying the URL to the application.

```
$ mvn verify -Pintegration-testing -Dserver.host=https://$(oc get route helloworld --
template='{{ .spec.host }}')
```

NOTE

The tests use SSL/TLS to connect to the deployed application. Therefore, you need the certificates to be trusted by the machine the tests are run from.

To trust the certificates, you must add it to a Java trust store.

Example

```
$ keytool -trustcacerts -keystore _<path-to-java-truststore>_ -storepass
_<trust-store-password>_ -importcert -alias _<alias-for-the-certificate>_ -file
_<path-to-certificate>_/_<certificate-name>_
```

Example output

```
[INFO] Running org.jboss.as.quickstarts.helloworld.HelloWorldServletIT
```

```
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.345 s -- in
org.jboss.as.quickstarts.helloworld.HelloWorldServletIT
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- maven-failsafe-plugin:3.2.2:verify (default) @ helloworld ---
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.984 s
[INFO] Finished at: 2023-11-30T15:51:22+05:30
[INFO] ------------------------------------------------------------------------
```

Revised on 2024-02-21 14:02:46 UTC