Red Hat Fuse 7.7

Getting Started

Get started quickly with Red Hat Fuse!

Last Updated: 2023-07-24

Red Hat Fuse 7.7 Getting Started

Get started quickly with Red Hat Fuse!

Legal Notice

Abstract

Get started with Fuse on Spring Boot, Fuse on Apache Karaf, and Fuse on JBoss Enterprise Application Platform.

Table of Contents

PREFACE

To get started with Fuse, you need to download and install the files for your desired container, whether that is Spring Boot, JBoss EAP, or Apache Karaf. The information and instructions here guide you in installing, developing, and building your first Fuse application for each of those containers.

CHAPTER 1. GETTING STARTED WITH FUSE ON SPRING BOOT

To develop Fuse applications on Spring Boot, get started by generating and building a Fuse sample booster project that runs on Spring Boot. The following topics provide details:

- Section 1.1, “About Fuse on Spring Boot”
- Section 1.2, “Generating your booster project”
- Section 1.3, “Building your booster project”

1.1. ABOUT FUSE ON SPRING BOOT

Spring Boot is an evolution of the well-known Spring container. A distinctive quality of the Spring Boot container is that container functionality is divided up into small chunks, which can be deployed independently. This enables you to deploy a container with a small footprint, specialized for a particular kind of service, and this happens to be exactly what you need to fit the paradigm of a *microservices architecture*.

Distinctive features of this container technology are:

- Particularly suited to running on a scalable cloud platform (Kubernetes and OpenShift).
- Small footprint (ideal for microservices architecture).
- Optimized for *convention over configuration*.
- No application server required. You can run a Spring Boot application Jar directly in a JVM.

1.2. GENERATING YOUR BOOSTER PROJECT

Fuse booster projects exist to help developers get started with running standalone applications. The instructions provided here guide you through generating one of those booster projects, the Circuit Breaker booster. This exercise demonstrates useful components of the Fuse on Spring Boot.

The Netflix/Hystrix circuit breaker enables distributed applications to handle interruptions to network connectivity and temporary unavailability of backend services. The basic idea of the circuit breaker pattern is that the loss of a dependent service is detected automatically and an alternative behavior can be programmed, in case the backend service is temporarily unavailable.

The Fuse circuit breaker booster consists of two related services:

- A **name** service, the backend service that returns a name to greet.
- A **greetings** service, the frontend service that invokes the **name** service to get a name and then returns the string, **Hello, NAME**.

In this booster demonstration, the Hystrix circuit breaker is inserted between the **greetings** service and the **name** service. If the backend **name** service becomes unavailable, the **greetings** service can fall back to an alternative behavior and respond to the client immediately, instead of being blocked while it waits for the **name** service to restart.

Prerequisites

- You must have access to the Red Hat Developer Platform.
- You must have a supported version of the Java Developer Kit (JDK). See the Supported Configurations page for details.
- You must have Apache Maven 3.3.x or later.

Procedure

1. Navigate to https://developers.redhat.com/launch.
2. Click **START**.
   The launcher wizard prompts you to log in to your Red Hat account.
3. Click the **Log in or register** button and then log in.
4. On the **Launcher** page, click the **Deploy an Example Application** button.
5. On the **Create Example Application** page, type the name, **fuse-circuit-breaker**, in the **Create Example Application as** field.
6. Click **Select an Example**.
7. In the **Example** dialog, select the **Circuit Breaker** option. A **Select a Runtime** dropdown menu appears.
   a. From the **Select a Runtime** dropdown, select **Fuse**.
   b. From the version dropdown menu, select **7.7 (Red Hat Fuse)** (do not select the **2.21.2 (Community)** version).
   c. Click **Save**.
8. On the **Create Example Application** page, click **Download**.
9. When you see the **Your Application is Ready** dialog, click **Download.zip**. Your browser downloads the generated booster project (packaged as a ZIP file).
10. Use an archive utility to extract the generated project to a convenient location on your local file system.

1.3. BUILDING YOUR BOOSTER PROJECT

These instructions guide you through building the Circuir Breaker booster with Fuse on Spring Boot.

Prerequisites

- You must have generated and downloaded your booster project via the Red Hat Developer Portal.
- You must have a supported version of the Java Developer Kit (JDK). See the Supported Configurations page for details.
- You must have Apache Maven 3.3.x or later.

Procedure

1. Open a shell prompt and build the project from the command line, using Maven:

```
cd fuse-circuit-breaker
```

```
mvn clean package
```

After Maven builds the project, it displays a **Build Success** message.

2. Open a new shell prompt and start the name service, as follows:

```
cd name-service
```

```
mvn spring-boot:run -DskipTests -Dserver.port=8081
```

As Spring Boot starts up, you should see output similar to the following:

```
...
2019-05-06 20:19:59.401  INFO 9553 --- [           main] o.a.camel.spring.SpringCamelContext
: Route: route1 started and consuming from: servlet:/name?httpMethodRestrict=GET
2019-05-06 20:19:59.402  INFO 9553 --- [           main] o.a.camel.spring.SpringCamelContext
: Total 1 routes, of which 1 are started
2019-05-06 20:19:59.403  INFO 9553 --- [           main] o.a.camel.spring.SpringCamelContext
: Apache Camel 2.21.0.fuse-730078-redhat-00001 (CamelContext: camel-1) started in 0.287
seconds
2019-05-06 20:19:59.406  INFO 9553 --- [           main] o.a.c.c.s.CamelHttpTransportServlet
: Initialized CamelHttpTransportServlet[name=CamelServlet, contextPath=]
2019-05-06 20:19:59.473  INFO 9553 --- [           main]
b.c.e.u.UndertowEmbeddedServletContainer : Undertow started on port(s) 8081 (http)
2019-05-06 20:19:59.479  INFO 9553 --- [           main]
com.redhat.fuse.boosters.cb.Application  : Started Application in 5.485 seconds (JVM
running for 9.841)
```

3. Open a new shell prompt and start the greetings service, as follows:

```
cd greetings-service
```

```
mvn spring-boot:run -DskipTests
```

As Spring Boot starts up, you should see output similar to the following:

```
...
2019-05-06 20:22:19.051  INFO 9729 --- [           main] o.a.c.c.s.CamelHttpTransportServlet
: Initialized CamelHttpTransportServlet[name=CamelServlet, contextPath=]
2019-05-06 20:22:19.115  INFO 9729 --- [           main]
b.c.e.u.UndertowEmbeddedServletContainer : Undertow started on port(s) 8080 (http)
2019-05-06 20:22:19.123  INFO 9729 --- [           main]
com.redhat.fuse.boosters.cb.Application  : Started Application in 7.68 seconds (JVM running
for 12.66)
```

The greetings service exposes a REST endpoint at the **http://localhost:8080/camel/greetings** URL.

4. Invoke the REST endpoint by either opening the URL in a web browser or by opening another

4. Invoke the REST endpoint by either opening the URL in a web browser or by opening another shell prompt and typing the following **curl** command:

```
curl http://localhost:8080/camel/greetings
```

Here is the response:

```
{"greetings":"Hello, Jacopo"}
```

5. To demonstrate the circuit breaker functionality provided by Camel Hystrix, kill the backend name service by typing **Ctrl-C** in the shell prompt window where the name service is running. Now that the name service is unavailable, the circuit breaker kicks in to prevent the greetings service from hanging when it is invoked.

6. Invoke the greetings REST endpoint by either opening **http://localhost:8080/camel/greetings** in a web browser or by typing the following **curl** command in another shell prompt window:

```
curl http://localhost:8080/camel/greetings
```

Here is the response:

```
{"greetings":"Hello, default fallback"}
```

In the window where the greetings service is running, the log shows the following sequence of messages:

```
2019-05-06 20:24:16.952  INFO 9729 --- [-CamelHystrix-2] route2                                  : Try
to call name Service
2019-05-06 20:24:16.956  INFO 9729 --- [-CamelHystrix-2]
o.a.c.httpclient.HttpMethodDirector      : I/O exception (java.net.ConnectException) caught
when processing request: Connection refused (Connection refused)
2019-05-06 20:24:16.956  INFO 9729 --- [-CamelHystrix-2]
o.a.c.httpclient.HttpMethodDirector      : Retrying request
2019-05-06 20:24:16.957  INFO 9729 --- [-CamelHystrix-2]
o.a.c.httpclient.HttpMethodDirector      : I/O exception (java.net.ConnectException) caught
when processing request: Connection refused (Connection refused)
2019-05-06 20:24:16.957  INFO 9729 --- [-CamelHystrix-2]
o.a.c.httpclient.HttpMethodDirector      : Retrying request
2019-05-06 20:24:16.957  INFO 9729 --- [-CamelHystrix-2]
o.a.c.httpclient.HttpMethodDirector      : I/O exception (java.net.ConnectException) caught
when processing request: Connection refused (Connection refused)
2019-05-06 20:24:16.957  INFO 9729 --- [-CamelHystrix-2]
o.a.c.httpclient.HttpMethodDirector      : Retrying request
2019-05-06 20:24:16.964  INFO 9729 --- [-CamelHystrix-2] route2                                  : We
are falling back!!!!
```

7. For more information about this example, open the **Circuit Breaker - Red Hat Fuse**page at http://localhost:8080/ (while the **greetings-service** is running). This page includes a link to the Hystrix dashboard that monitors the state of the circuit breaker.

CHAPTER 2. GETTING STARTED WITH FUSE ON KARAF

To learn about Fuse on Karaf as well as install, develop, and build your first Fuse application on a Karaf container, the information and instructions here assist you with this. See the following topics for details:

- Section 2.1, “About Fuse on Karaf”
- Section 2.2, “Installing Fuse on Karaf”
- Section 2.3, “Building your first Fuse application on Karaf”

2.1. ABOUT FUSE ON KARAF

Apache Karaf is based on the OSGi standard from the OSGi Alliance. OSGi originated in the telecommunications industry, where it was used to develop gateway servers that could be upgraded on the fly, without needing to shut down the server (a feature known as *hot code swapping*). Subsequently, OSGi container technology has found a variety of other uses and is popular for modularised applications (for example, the Eclipse IDE).

Distinctive features of this container technology are:

- Particularly suited to running in standalone mode.
- Strong support for modularisation (OSGi bundles), with sophisticated class-loading support.
- Multiple versions of a dependency can be deployed side by side in a container (but this requires some care in practice).
- Hot code swapping, enabling you to upgrade or replace a module without shutting down the container. This is a unique feature, but requires significant effort to make it work properly.

Note: Spring Dynamic Modules (Spring-DM) (which integrates Spring XML with the OSGi service layer in Apache Karaf) is not supported. Instead, you should use the Blueprint framework. Using Blueprint XML does not prevent you from using the Java libraries from the Spring framework: the latest version of Spring is compatible with Blueprint.

2.2. INSTALLING FUSE ON KARAF

The standard installation package for Fuse 7.7 on Karaf is available for download from the Red Hat Customer Portal. It installs the standard assembly of the Karaf container, and provides the full Fuse technology stack.

Prerequisites

- You need a full-subscription account on the Red Hat Customer Portal.
- You must be logged into the customer portal.
- You must have downloaded the CodeReady Studio installer.
- You must have downloaded the Fuse on Karaf installer.

Procedure

1. Unpack the downloaded **.zip** archive file for Fuse on Apache Karaf to a convenient location on your file system, **FUSE_INSTALL**.
2. Add an administrator user to the Fuse runtime.
   a. Open the **FUSE_INSTALL/etc/users.properties** file in a text editor.
   b. Delete the **#** character at the start of the line that starts with **#admin = admin**.
   c. Delete the **#** character at the start of the line that starts with **#_g_\:admingroup**.
   d. Customize the username, **USERNAME**, and password, **PASSWORD**, of the user entry, so that you have a user entry and an admin group entry like the following (on consecutive lines):

```
USERNAME = PASSWORD,_g_:admingroup
_g_\:admingroup = group,admin,manager,viewer,systembundles,ssh
```

   e. Save the **etc/users.properties** file.
3. Run the CodeReady Studio installer as follows:

```
java -jar DOWNLOAD_LOCATION/codereadystudio-12.21.3.GA-installer-standalone.jar
```

4. During installation:
   a. Accept the terms and conditions.
   b. Choose your preferred installation path.
   c. Select the Java 8 JVM.
   d. At the **Select Platforms and Servers** step, configure the Fuse on Karaf runtime by clicking **Add** and browsing to the location of the **FUSE_INSTALL** directory.
   e. At the **Select Additional Features to Install** step, select **Red Hat Fuse Tooling**.
5. CodeReady Studio starts up. When the **Searching for runtimes** dialog appears, click **OK** to create the Fuse on Karaf runtime.
6. *(Optional)* In order to use Apache Maven from the command line, you need to install and configure Maven.

NOTE

If you are using CodeReady Studio exclusively, it is not strictly necessary to install Maven, because CodeReady Studio has Maven pre-installed and configured for you. However, if you plan to invoke Maven from the command line, it is necessary to perform this step.

2.3. BUILDING YOUR FIRST FUSE APPLICATION ON KARAF

This set of instructions assists you in building your first Fuse application on Karaf.

Prerequisites

- You need a full-subscription account on the Red Hat Customer Portal .
- You must be logged into the customer portal.
- You must have downloaded the CodeReady Studio installer.
- You must have downloaded and successfully installed Fuse on Karaf.

Procedure

1. In CodeReady Studio, create a new project, as follows:
   a. Select **File→New→Fuse Integration Project**.
   b. Enter **fuse-camel-cbr** in the **Project Name** field.
   c. Click **Next**.
   d. In the **Select a Target Environment** pane, choose the following settings:
      - Select **Standalone** as the deployment platform.
      - Select **Karaf/Fuse on Karaf** as the runtime environment and use the **Runtime (optional)** dropdown menu to select the **fuse-karaf-7.11.1.fuse-7_11_1-00013-redhat-00003 Runtime** server as the target runtime.
   e. After selecting the target runtime, the **Camel Version** is automatically selected for you and the field is grayed out.
   f. Click **Next**.
   g. In the **Advanced Project Setup** pane, select the **Beginner→Content Based Router - Blueprint DSL** template.
   h. Click **Finish**.
   i. If prompted to open the associated Fuse Integration perspective, click **Yes**.
   j. Wait while CodeReady Studio downloads required artifacts and builds the project in the background.

IMPORTANT

If this is the first time you are building a Fuse project in CodeReady Studio, it will take *several minutes* for the wizard to finish generating the project, as it downloads dependencies from remote Maven repositories. Do not attempt to interrupt the wizard or close CodeReady Studio while the project is building in the background.

2. Deploy the project to the server, as follows:
   a. In the **Servers** view (bottom left corner of the Fuse Integration perspective), if the server is not already started, select the **fuse-karaf-7.11.1.fuse-7_11_1-00013-redhat-00003 Runtime Server** server and click the green arrow to start it.

NOTE

If you see the dialog, **Warning: The authenticity of host 'localhost' can't be established.**, click **Yes** to connect to the server and access the Karaf console.

b. Wait until you see a message like the following in the **Console** view:

```
Karaf started in 1s. Bundle stats: 12 active, 12 total
```

c. After the server has started, switch back to the **Servers** view, right-click on the server and select **Add and Remove** from the context menu.

d. In the **Add and Remove** dialog, select the **fuse-camel-cbr** project and click the **Add >** button.

e. Click **Finish**.

f. You can check whether the project's OSGi bundle has started up by going to the **Terminal** view and entering **bundle:list | tail**. You should see some output like the following:

```
...
228 │ Active │  80 │ 1.0.0.201505202023      │ org.osgi:org.osgi.service.j
232 │ Active │  80 │ 1.0.0.SNAPSHOT          │ Fuse CBR Quickstart
```

NOTE

As soon as the Camel route starts up, it will create a directory, **work/cbr/input**, in your Fuse installation (*not* in the **fuse-camel-cbr** project).

3. Copy the files you find in the project's **src/main/data** directory to the **FUSE_INSTALL/work/cbr/input** directory. You can do this in your system file browser (outside of Eclipse).

4. Wait a few moments and then look in the **FUSE_INSTALL/work/cbr/output** directory to see the same files organized by country:

   a. **order1.xml** in **work/cbr/output/others**

   b. **order2.xml** and **order4.xml** in **work/cbr/output/uk**

   c. **order3.xml** and **order5.xml** in **work/cbr/output/us**

5. Undeploy the project, as follows:

   a. In the **Servers** view, select the **Red Hat Fuse 7+ Runtime Server** server.

   b. Right-click on the server and select **Add and Remove** from the context menu.

   c. In the **Add and Remove** dialog, select your **fuse-camel-cbr** project and click the **< Remove** button.

   d. Click **Finish**.

CHAPTER 3. GETTING STARTED WITH FUSE ON JBOSS EAP

This chapter introduces Fuse on JBoss EAP, and explains how to install, develop, and build your first Fuse application on a JBoss EAP container.

See the following topics for details:

- Section 3.1, "About Fuse on JBoss EAP"
- Section 3.2, "Installing Fuse on JBoss EAP"
- Section 3.3, "Building your first Fuse application on JBoss EAP"

3.1. ABOUT FUSE ON JBOSS EAP

JBoss Enterprise Application Platform (EAP), based on Jakarta EE technology (previously, Java EE) from the Eclipse Foundation, was originally created to address use cases for developing enterprise applications. JBoss EAP is characterized by well-defined patterns for implementing services and standardized Java APIs (for example, for persistence, messaging, security, and so on). In recent years, this technology has evolved to be more lightweight, with the introduction of CDI for dependency injection and simplified annotations for enterprise Java beans.

Distinctive features of this container technology are:

- Particularly suited to running in standalone mode.
- Many standard services (for example, persistence, messaging, security, and so on) pre-configured and provided out-of the-box.
- Application WARs typically small and lightweight (because many dependencies are pre-installed in the container).
- Standardized, backward-compatible Java APIs.

3.2. INSTALLING FUSE ON JBOSS EAP

The standard installation package for Fuse 7.7 on JBoss EAP is available for download from the Red Hat Customer Portal. It installs the standard assembly of the JBoss EAP container, and provides the full Fuse technology stack.

Prerequisites

- You must have a full-subscription account on the Red Hat Customer Portal .
- You must be logged into the customer portal.
- You must have downloaded JBoss EAP and JBoss EAP 7.2 Update 05 .
- You must have downloaded Fuse on JBoss EAP.
- You must have downloaded the CodeReady Studio installer.

Procedure

1. Run the JBoss EAP installer from a shell prompt, as follows:

```
java -jar DOWNLOAD_LOCATION/jboss-eap-7.4.5-installer.jar
```

2. During installation:
   a. Accept the terms and conditions.
   b. Choose your preferred installation path, **EAP_INSTALL**, for the JBoss EAP runtime.
   c. Create an administrative user and make a careful note of these administrative user credentials for later.
   d. You can accept the default settings on the remaining screens.
3. Open a shell prompt and change directory to **EAP_INSTALL**.
4. From the **EAP_INSTALL** directory, apply JBoss EAP 7.2 Update 05. For example:

```
bin/jboss-cli.sh "patch apply jboss-eap-7.2.x-patch.zip"
```

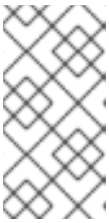
5. From the **EAP_INSTALL** directory, run the Fuse on EAP installer, as follows:

```
java -jar DOWNLOAD_LOCATION/fuse-eap-installer-7.11.1.jar
```

6. Run the CodeReady Studio installer, as follows:

```
java -jar DOWNLOAD_LOCATION/codereadystudio-12.21.3.GA-installer-standalone.jar
```

7. During installation:
   a. Accept the terms and conditions.
   b. Choose your preferred installation path.
   c. Select the Java 8 JVM.
   d. At the **Select Platforms and Servers** step, configure the JBoss EAP runtime by clicking **Add** and browsing to the location of the **EAP_INSTALL** directory.
   e. At the **Select Additional Features to Install** step, select **Red Hat Fuse Tooling**.
8. CodeReady Studio starts up. When the **Searching for runtimes** dialog appears, click **OK** to create the JBoss EAP runtime.
9. *(Optional)* In order to use Apache Maven from the command line, you need to install and configure Maven.



NOTE

If you are using CodeReady Studio exclusively, it is not strictly necessary to install Maven, because CodeReady Studio has Maven pre-installed and configured. However, if you plan to invoke Maven from the command line, you must perform this step.

3.3. BUILDING YOUR FIRST FUSE APPLICATION ON JBOSS EAP

This set of instructions assists you in building your first Fuse application on JBoss EAP.

Prerequisites

- You need a full-subscription account on the Red Hat Customer Portal .
- You must be logged into the customer portal.
- You must have downloaded and successfully installed Fuse on JBoss EAP.
- You must have downloaded and successfully installed the CodeReady Studio installer.

Procedure

1. In CodeReady Studio, create a new project, as follows:
   a. Select **File→New→Fuse Integration Project**.
   b. In the **Project Name** field, enter **eap-camel**.
   c. Click **Next**.
   d. In the **Select a Target Environment** pane, choose the following settings:
      - Select **Standalone** as the deployment platform.
      - Select **Wildfly/Fuse on EAP** as the runtime environment and use the **Runtime (optional)** dropdown menu to select the **JBoss EAP 7.x Runtime** server as the target runtime.
   e. After selecting the target runtime, the **Camel Version** is automatically selected for you and the field is grayed out.
   f. Click **Next**.
   g. In the **Advanced Project Setup** pane, select the **Spring Bean - Spring DSL** template.
   h. Click **Finish**.

IMPORTANT

If this is the first time you are building a Fuse project in CodeReady Studio, it will take *several minutes* for the wizard to finish generating the project. This is because it downloads dependencies from remote Maven repositories. Do not interrupt the wizard or close CodeReady Studio while the project is building in the background.

   i. If prompted to open the associated Fuse Integration perspective, click **Yes**.
   j. Wait while CodeReady Studio downloads required artifacts and builds the project in the background.

2. Deploy the project to the server, as follows:

a. In the **Servers** view (bottom right corner of the Fuse Integration perspective), if the server is not already started, select the **Red Hat JBoss EAP 7.2 Runtime** server and click the green arrow to start it.

b. Wait until you see a message like the following in the **Console** view:

```
14:47:07,283 INFO  [org.jboss.as] (Controller Boot Thread) WFLYSRV0025: JBoss EAP
7.2.0.GA (WildFly Core 6.0.11.Final-redhat-00001) started in 13948ms - Started 495 of
680 services (326 services are lazy, passive or on-demand)
```

c. After the server has started, switch back to the **Servers** view, right-click the server and select **Add and Remove** from the context menu.

d. In the **Add and Remove** dialog, select the **eap-camel** project and click **Add >**.

e. Click **Finish**.

3. Verify that the project is working, as follows:

a. Browse to the following URL to access the service running in the **eap-camel** project: http://localhost:8080/camel-test-spring?name=Kermit

b. The browser window should show the response **Hello Kermit**.

4. Undeploy the project, as follows:

a. In the **Servers** view, select the **Red Hat JBoss EAP 7.2 Runtime** server.

b. Right-click the server and select **Add and Remove** from the context menu.

c. In the **Add and Remove** dialog, select your **eap-camel** project and click **< Remove**.

d. Click **Finish**.

CHAPTER 4. SETTING UP MAVEN LOCALLY

Typical Fuse application development uses Maven to build and manage projects.

The following topics describe how to set up Maven locally:

- Section 4.1, “Preparing to set up Maven”
- Section 4.2, “Adding Red Hat repositories to Maven”
- Section 4.3, “Using local Maven repositories”
- Section 4.4, “Setting Maven mirror using environmental variables or system properties”
- Section 4.5, “About Maven artifacts and coordinates”

4.1. PREPARING TO SET UP MAVEN

Maven is a free, open source, build tool from Apache. Typically, you use Maven to build Fuse applications.

Procedure

1. Download the latest version of Maven from the Maven download page .
2. Ensure that your system is connected to the Internet.
   While building a project, the default behavior is that Maven searches external repositories and downloads the required artifacts. Maven looks for repositories that are accessible over the Internet.

   You can change this behavior so that Maven searches only repositories that are on a local network. That is, Maven can run in an offline mode. In offline mode, Maven looks for artifacts in its local repository. See Section 4.3, “Using local Maven repositories” .

4.2. ADDING RED HAT REPOSITORIES TO MAVEN

To access artifacts that are in Red Hat Maven repositories, you need to add those repositories to Maven’s **settings.xml** file. Maven looks for the **settings.xml** file in the **.m2** directory of the user’s home directory. If there is not a user specified **settings.xml** file, Maven uses the system-level **settings.xml** file at **M2_HOME/conf/settings.xml**.

Prerequisite

You know the location of the **settings.xml** file in which you want to add the Red Hat repositories.

Procedure

In the **settings.xml** file, add **repository** elements for the Red Hat repositories as shown in this example:

```
<?xml version="1.0"?>
<settings>

  <profiles>
    <profile>
      <id>extra-repos</id>
```

```
<activation>
  <activeByDefault>true</activeByDefault>
</activation>
<repositories>
 <repository>
     <id>redhat-ga-repository</id>
     <url>https://maven.repository.redhat.com/ga</url>
     <releases>
        <enabled>true</enabled>
     </releases>
     <snapshots>
        <enabled>false</enabled>
     </snapshots>
  </repository>
  <repository>
     <id>redhat-ea-repository</id>
     <url>https://maven.repository.redhat.com/earlyaccess/all</url>
     <releases>
        <enabled>true</enabled>
     </releases>
     <snapshots>
        <enabled>false</enabled>
     </snapshots>
  </repository>
  <repository>
    <id>jboss-public</id>
    <name>JBoss Public Repository Group</name>
    <url>https://repository.jboss.org/nexus/content/groups/public/</url>
  </repository>
</repositories>
<pluginRepositories>
  <pluginRepository>
     <id>redhat-ga-repository</id>
     <url>https://maven.repository.redhat.com/ga</url>
     <releases>
        <enabled>true</enabled>
     </releases>
     <snapshots>
        <enabled>false</enabled>
     </snapshots>
  </pluginRepository>
  <pluginRepository>
     <id>redhat-ea-repository</id>
     <url>https://maven.repository.redhat.com/earlyaccess/all</url>
     <releases>
        <enabled>true</enabled>
     </releases>
     <snapshots>
        <enabled>false</enabled>
     </snapshots>
  </pluginRepository>
  <pluginRepository>
    <id>jboss-public</id>
    <name>JBoss Public Repository Group</name>
    <url>https://repository.jboss.org/nexus/content/groups/public</url>
  </pluginRepository>
```

```
    </pluginRepositories>
  </profile>
</profiles>

<activeProfiles>
  <activeProfile>extra-repos</activeProfile>
</activeProfiles>

</settings>
```

4.3. USING LOCAL MAVEN REPOSITORIES

If you are running the Apache Karaf container without an Internet connection, and you need to deploy an application that has dependencies that are not available offline, you can use the Maven dependency plug-in to download the application's dependencies into a Maven offline repository. You can then distribute this customized Maven offline repository to machines that do not have an Internet connection.

Procedure

1. In the project directory that contains the **pom.xml** file, download a repository for a Maven project by running a command such as the following:

   ```
   mvn org.apache.maven.plugins:maven-dependency-plugin:3.1.0:go-offline -
   Dmaven.repo.local=/tmp/my-project
   ```

   In this example, Maven dependencies and plug-ins that are required to build the project are downloaded to the **/tmp/my-project** directory.

2. Edit the **etc/org.ops4j.pax.url.mvn.cfg** file to set **org.ops4j.pax.url.mvn.offline** to true. This enables offline mode:

   ```
   ##
   # If set to true, no remote repository will be accessed when resolving artifacts
   #
   org.ops4j.pax.url.mvn.offline = true
   ```

3. Distribute this customized Maven offline repository internally to any machines that do not have an Internet connection.

4.4. SETTING MAVEN MIRROR USING ENVIRONMENTAL VARIABLES OR SYSTEM PROPERTIES

When running the applications you need access to the artifacts that are in the Red Hat Maven repositories. These repositories are added to Maven's **settings.xml** file. Maven checks the following locations for **settings.xml** file:

- looks for the specified url
- if not found looks for **${user.home}/.m2/settings.xml**
- if not found looks for **${maven.home}/conf/settings.xml**
- if not found looks for **${M2_HOME}/conf/settings.xml**

- if no location is found, empty **org.apache.maven.settings.Settings** instance is created.

4.4.1. About Maven mirror

Maven uses a set of remote repositories to access the artifacts, which are currently not available in local repository. The list of repositories almost always contains Maven Central repository, but for Red Hat Fuse, it also contains Maven Red Hat repositories. In some cases where it is not possible or allowed to access different remote repositories, you can use a mechanism of Maven mirrors. A mirror replaces a particular repository URL with a different one, so all HTTP traffic when remote artifacts are being searched for can be directed to a single URL.

4.4.2. Adding Maven mirror to settings.xml

To set the Maven mirror, add the following section to Maven's **settings.xml**:

```
<mirror>
    <id>all</id>
    <mirrorOf>*</mirrorOf>
    <url>http://host:port/path</url>
</mirror>
```

No mirror is used if the above section is not found in the **settings.xml** file. To specify a global mirror without providing the XML configuration, you can use either system property or environmental variables.

4.4.3. Setting Maven mirror using environmental variable or system property

To set the Maven mirror using either environmental variable or system property, you can add:

- Environmental variable called **MAVEN_MIRROR_URL** to **bin/setenv** file
- System property called **mavenMirrorUrl** to **etc/system.properties** file

4.4.4. Using Maven options to specify Maven mirror url

To use an alternate Maven mirror url, other than the one specified by environmental variables or system property, use the following maven options when running the application:

- **-DmavenMirrorUrl=mirrorId::mirrorUrl**
  for example, **-DmavenMirrorUrl=my-mirror::http://mirror.net/repository**
- **-DmavenMirrorUrl=mirrorUrl**
  for example, **-DmavenMirrorUrl=http://mirror.net/repository**. In this example, the <id> of the <mirror> is just a mirror.

4.5. ABOUT MAVEN ARTIFACTS AND COORDINATES

In the Maven build system, the basic building block is an *artifact*. After a build, the output of an artifact is typically an archive, such as a JAR or WAR file.

A key aspect of Maven is the ability to locate artifacts and manage the dependencies between them. A *Maven coordinate* is a set of values that identifies the location of a particular artifact. A basic coordinate has three values in the following form:

groupId:artifactId:version

Sometimes Maven augments a basic coordinate with a *packaging* value or with both a *packaging* value and a *classifier* value. A Maven coordinate can have any one of the following forms:

```
groupId:artifactId:version
groupId:artifactId:packaging:version
groupId:artifactId:packaging:classifier:version
```

Here are descriptions of the values:

groupdId

Defines a scope for the name of the artifact. You would typically use all or part of a package name as a group ID. For example, **org.fusesource.example**.

artifactId

Defines the artifact name relative to the group ID.

version

Specifies the artifact's version. A version number can have up to four parts: **n.n.n.n**, where the last part of the version number can contain non-numeric characters. For example, the last part of **1.0-SNAPSHOT** is the alphanumeric substring, **0-SNAPSHOT**.

packaging

Defines the packaged entity that is produced when you build the project. For OSGi projects, the packaging is **bundle**. The default value is **jar**.

classifier

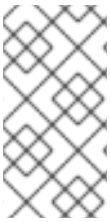
Enables you to distinguish between artifacts that were built from the same POM, but have different content.

Elements in an artifact's POM file define the artifact's group ID, artifact ID, packaging, and version, as shown here:

```
<project ... >
  ...
  <groupId>org.fusesource.example</groupId>
  <artifactId>bundle-demo</artifactId>
  <packaging>bundle</packaging>
  <version>1.0-SNAPSHOT</version>
  ...
</project>
```

To define a dependency on the preceding artifact, you would add the following **dependency** element to a POM file:

```
<project ... >
  ...
  <dependencies>
    <dependency>
      <groupId>org.fusesource.example</groupId>
      <artifactId>bundle-demo</artifactId>
      <version>1.0-SNAPSHOT</version>
    </dependency>
  </dependencies>
  ...
</project>
```



NOTE

It is not necessary to specify the **bundle** package type in the preceding dependency, because a bundle is just a particular kind of JAR file and **jar** is the default Maven package type. If you do need to specify the packaging type explicitly in a dependency, however, you can use the **type** element.