Red Hat Fuse 7.9

Getting Started with Fuse on JBoss EAP

Get started quickly with Red Hat Fuse on EAP

Last Updated: 2021-08-12

Red Hat Fuse 7.9 Getting Started with Fuse on JBoss EAP

Get started quickly with Red Hat Fuse on EAP

Legal Notice

Abstract

Get started building an application with 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 JBoss EAP container. The information and instructions here guide you in installing, developing, and building your first Fuse application.

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.

CHAPTER 1. 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 1.1, “About Fuse on JBoss EAP”
- Section 1.2, “Installing Fuse on JBoss EAP”
- Section 1.3, “Building your first Fuse application on JBoss EAP”

1.1. 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.

1.2. INSTALLING FUSE ON JBOSS EAP

The standard installation package for Fuse 7.9 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.
- 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-7.4.0-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, run the Fuse on EAP installer, as follows:

```
java -jar DOWNLOAD_LOCATION/fuse-eap-installer-7.9.0.jar
```

5. Run the CodeReady Studio installer, as follows:

```
java -jar DOWNLOAD_LOCATION/codereadystudio-12.19.1.GA-installer-standalone.jar
```

6. 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**.
7. CodeReady Studio starts up. When the **Searching for runtimes** dialog appears, click **OK** to create the JBoss EAP runtime.
8. *(Optional)* In order to use Apache Maven from the command line, you need to install and configure Maven as described in as described in Setting up Maven locally).

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.

1.3. BUILDING YOUR FIRST FUSE APPLICATION ON JBOSS EAP

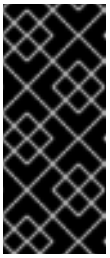
This set of instructions assists you in building your first Fuse application on JBoss EAP.

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 7.3 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
7.3.2.GA (WildFly Core 10.1.11.Final-redhat-00001) started in 3301ms - Started 314 of
576 services (369 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 7.3 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 2. 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 2.1, “Preparing to set up Maven”
- Section 2.2, “Adding Red Hat repositories to Maven”
- Section 2.3, “Using local Maven repositories”
- Section 2.4, “Setting Maven mirror using environmental variables or system properties”
- Section 2.5, “About Maven artifacts and coordinates”

2.1. 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 2.3, “Using local Maven repositories” .

2.2. 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>
```

2.3. USING LOCAL MAVEN REPOSITORIES

If you are running a 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.

2.4. 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.

2.4.1. 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.

2.4.2. 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.

2.4.3. 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

2.4.4. 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.

2.5. 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

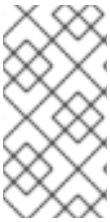
Enables you to distinguish between artifacts that were built from the same POM, but have different content.

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.