# Red Hat Fuse 7.7

## Installing on Apache Karaf

Installing Red Hat Fuse on the Apache Karaf container

Last Updated: 2020-10-28

# Red Hat Fuse 7.7 Installing on Apache Karaf

Installing Red Hat Fuse on the Apache Karaf container

## Legal Notice



## Abstract

It is easy to install Red Hat Fuse and tailor the installation to a particular environment.

# Table of Contents

# PREFACE

Red Hat Fuse is a lightweight, flexible integration platform that enables rapid integration across the extended enterprise—on-premise or in the cloud.

Based on Apache Camel, Fuse leverages pattern-based integration, a rich connector catalog, and extensive data transformation capabilities to enables users to integrate anything.

# CHAPTER 1. INSTALL FUSE FOR DEVELOPMENT ON APACHE KARAF

To develop Fuse applications that run on Karaf, install Fuse locally as described in the following topics:

- Section 1.1, “Prepare to install Fuse on Apache Karaf”
- Section 1.2, “Install Fuse on Apache Karaf”
- Section 1.3, “About running Fuse on Karaf offline”
- Section 1.4, “Optionally use standalone Apache distributions”

## 1.1. PREPARE TO INSTALL FUSE ON APACHE KARAF

To prepare to install Fuse on Apache Karaf, check your system to ensure that it meets hardware requirements, is a supported platform, and has a supported Java runtime. Also, confirm that you plan to use supported standard software for web services, APIs, and transport protocols.

**Procedure**

1. On the system on which you plan to install Fuse, confirm that it has:
   - 250 MB of free disk space
   - 2 GB of RAM

   This hardware requirement is for a full installation of Fuse on Apache Karaf. In addition, a system that is running Fuse requires space for caching, persistent message stores, and other functions. The actual requirement is dependent on what your Fuse applications do.
2. Confirm that the system on which you plan to install Fuse is running a supported platform. Red Hat tests and supports Fuse products on the platforms that are listed in Red Hat Fuse Supported Configurations.
3. Confirm that your system is running a Java runtime that is supported by Fuse on Apache Karaf. See Supported Java Versions in Red Hat Fuse Supported Configurations .
4. Ensure that your Java runtime is not installed under a directory path that includes whitespace. For example, **C:\Program Files\Java\jdk8** is not an acceptable path. Whitespace in paths causes unpredictable errors in Fuse on Apache Karaf at run time.
5. Check the list of Red Hat Fuse Supported Standards to confirm that you are using supported standard software.

## 1.2. INSTALL FUSE ON APACHE 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 Apache Karaf container, and provides the full Fuse technology stack.

It is possible to create your own custom assembly of Fuse 7.7, which contains a customized subset of the Fuse features and bundles. The **custom** quickstart demonstrates how to use Maven to create a custom assembly of Red Hat Fuse. You can install all of the quickstarts from a downloadable file available on the Fuse Software Downloads page.

### Prerequisite

The system on which you plan to install Fuse meets the hardware and software requirements described in Section 1.1, “Prepare to install Fuse on Apache Karaf” .

### Procedure

1. In a browser, go to the Fuse Software Downloads page.
   If you are not already logged in to the Red Hat customer portal, there is a prompt to log in and then the download page appears (your account must be associated with a Red Hat Fuse subscription).

2. In the Fuse **Software Downloads** page, to the right of **Red Hat Fuse 7.7 on Karaf Installer**, click **Download** to obtain a local zip file.

3. Extract the contents of the zip file into a directory for which you have all permissions.
   **Do not** unpack the zip file into a directory that has spaces or any of the following special characters in its path name: **#**, **%**, **^**, **"**. For example, do not unpack into **C:\Documents and Settings\Greco#Roman\Desktop\fuse**.

4. If you are using the IBM JDK, perform the following additional steps:

   a. In your Fuse installation directory, in the **/lib/endorsed** directory, remove the **saaj-api .jar** file, for example:

   ```
   rm lib/endorsed/org.apache.servicemix.specs.saaj-api-1.3-2.9.0.jar
   ```

   b. Set the **JAVA_OPTS** environment variable as follows:

   ```
   JAVA_OPTS=-Xshareclasses:none
   ```

   You must set the **JAVA_OPTS** environment variable before you start the Karaf container.

5. Add an administrator user to enable remote access to the Fuse on Karaf container and to access the Fuse Console.
   By default, no users are defined for the container. You can run the container in the foreground in this case, but you cannot access the container remotely and you cannot run it in the background. It is recommended that you create at least one user with the **admin** role by following these steps:

   a. In a text editor, open the **etc/users.properties** file, which is in your Fuse installation directory.

   b. Locate the following lines:

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

   c. For each line, remove the leading **#** character to uncomment the line.

   d. In the first line, change the first instance of **admin** to a username that you choose, for example **user1**.

   e. In the same line, change the second instance of **admin** to a password for that user, for example **passw0rd**.
   For example:

```
user1 = passw0rd,_g_:admingroup
_g_\:admingroup = group,admin,manager,viewer,systembundles,ssh
```

f. Save and close the file.

6. To start Fuse, run **bin/fuse** on Linux/Unix or **bin\fuse.bat** on Windows.

7. Optionally, to access the Fuse Console, open the provided URL in a web browser and login with the username and password that you set in the **etc/users.properties** file. For more information about using the Fuse Console see Managing Fuse.

## 1.3. ABOUT RUNNING FUSE ON KARAF OFFLINE

You can run the Apache Karaf container in offline mode, that is, without an Internet connection. However, if you are planning to deploy custom applications to the container, it might be necessary to download additional dependencies to a local Maven repository before you can run the container in offline mode with these applications.

To run the Apache Karaf container in offline mode, it is necessary to distinguish between the following kinds of dependencies:

- **Runtime dependencies** are dependencies that are required to run the Apache Karaf container in its default configuration.

- **Build-time dependencies** are dependencies that are required to build a custom application, which might include third-party libraries.

Here is a summary of what can be done in offline mode and what needs to be done in online mode (with an Internet connection):

- **Running the Apache Karaf container in its default configuration** is supported in offline mode. The default configuration of the Apache Karaf container is specified by the **featuresBoot** property in the **etc/org.apache.karaf.features.cfg** file. The required dependencies are provided in the **system/** sub-directory of the installation.

- **Installing additional features** is, in general, **not** supported in offline mode. In principle, you can use the **features:install** command to install any of the features from the standard feature repositories (as specified by the **featuresRepositories** property in the **etc/org.apache.karaf.features.cfg** file), but the majority of these features must be downloaded from the Internet and are thus not supported in offline mode.

- **Deploying custom applications** is, in general, **not** supported in offline mode. There may be some cases where an application with a minimal set of build-time dependencies is deployable offline. However, custom applications typically have third-party dependencies that require an Internet connection so that JAR files can be downloaded by Apache Maven.

### Additional resource

Section 3.3, “Using local Maven repositories”

## 1.4. OPTIONALLY USE STANDALONE APACHE DISTRIBUTIONS

Red Hat Fuse provides an additional package to download, which contains the standard distributions of Apache Camel and Apache CXF. If you want to use a standard, upstream distribution of Apache Camel or Apache CXF (without the OSGi container) use the archived versions in the downloaded **extras**

package.

**Procedure**

1. Log in to the Red Hat customer portal.
2. Go to the Red Hat Customer Portal→Downloads→Red Hat Fuse→Downloads page.
3. Select **7.7.0** from the **Version** drop-down list on the **Software Downloads** page.
4. Download the Red Hat Fuse 7.7.0 Extras archive.
   The extras archive file contains the following archive files nested inside it:
   - **apache-camel-2.21.0.fuse-770013-redhat-00001.zip**
   - **apache-cxf-3.2.7.fuse-770017-redhat-00001.zip**
5. Copy these files to the desired location and decompress them using the appropriate utility for your platform.

**WARNING**

Do not unpack an archive file into a folder that has spaces in its path name. For example, do not unpack into **C:\Documents and Settings\Greco Roman\Desktop\fuse**.

# CHAPTER 2. APPLYING HOTFIX PATCH TO FUSE ON APACHE KARAF

## 2.1. PATCHING FEATURES AND BUNDLES

Patches are ZIP archives that contain the updated versions of files present in Fuse on Apache Karaf installation. These include:

- Bundles: These are the most common and in the simplest case, hotfix patch may include single bundle.
- Configuration files and scripts that are present respectively in **$FUSE_HOME/etc** and **$FUSE_HOME/bin** directories.
- Libraries that are not ordinary bundles and reside in **$FUSE_HOME/lib** directory.
- Feature definition changes: Normally Karaf features are included in descriptors available in the **$FUSE_HOME/system** directory, but hotfix patches do not change these files. Instead, hot fix patch may alter feature override file which is **$FUSE_HOME/etc/org.apache.karaf.features.xml**. This allows you to alter feature definitions in hotfix manner by upgrading given feature's bundles or even make given feature use additional bundle.

**Difference between Upgrading and Hotfix Patches**

- Hotfix Patch: A hotfix patch contains fixes for only one or more critical bugs. These are intended to be applied on top of your current Red Hat Fuse distribution. Its main purpose is to update some of the bundles and librabies in an existing distribution.
- Upgrading: The Fuse on Apache Karaf upgrade mechanism enables you apply fixes to an Apache Karaf container without needing to reinstall an updated version of Fuse on Karaf. It also allows you to roll back the upgrade, if the upgrade causes problems with your deployed applications. The Fuse on Apache Karaf upgrade process updates any files, including bundle JARs, configuration files, and any static files.

For Fuse on Apache Karaf Standalone you can apply the patch using commands from the Karaf console's patch shell. This approach is non-destructive and reversible. Following procedure can also be used for upgrading Red Hat Fuse on Apache Karaf. For more information of upgrading see Upgrading Fuse on Apache Karaf.

## 2.2. APPLYING A HOTFIX PATCH TO RED HAT FUSE ON APACHE KARAF

You can use the hotfix mechanism to update the available feature definitions and bundles at the same time. The procedure to apply a hotfix patch to the Fuse on Apache Karaf installation is as follows.

**Procedure**

1. Download the required patch from the Customer Portal.
2. Make a full backup of your Fuse on Apache Karaf installation before upgrading.
3. Open the terminal and start Fuse on Apache karaf server.
   - 

```
[user@FUSE_HOME/bin ~] $ ./fuse
```

4. Add the patch to the container's environment by entering the **patch:add** command. For example, to add the patch-xxx.zip patch file, enter:

```
karaf@root()> patch:add 'file:///Downloads/patch-xxx.zip'
[name]              [installed] [rollup]   [description]
my-patch-x          false       false      my-patch-x
```

5. Simulate installing the patch by entering the **patch:simulate** command.
This generates a log of the changes that will be made to the container when the patch is installed, but will not make any actual changes to the container. Review the simulation log to understand these changes.

6. Enter **patch:list** command to view a list of added patches. In this list, the entries under the [name] heading are patch IDs.

```
patch:list
[name]                  [installed]   [description]
my-patch-x                false
```

7. Apply a patch to the container by entering the **patch:install** command and specifying the patch ID for the patch that you want to apply.

```
patch:install my-patch-x
```

## 2.3. ROLLING BACK A PATCH

You can rollback the installed hotfix patch and restore it to pre-patch behavior using the **patch:rollback** command, as follows:

### Procedure

1. Enter the **patch:list** command to obtain the patch ID of the most recently installed patch.

2. To rollback the updated bundle enter the following command:

```
karaf@root()> patch:rollback my-patch-x
INFO : org.jboss.fuse.modules.patch.patch-management (2): Rolling back non-rollup patch
"my-patch-x"
removing overriden feature: hawtio-rbac/2.0.0.fuse-000117
refreshing features
Enter feature:info command to view the information about the feature.
```

```
karaf@root()> feature:info hawtio-rbac
Feature hawtio-rbac 2.0.0.fuse-000117
Details:
    Installs the hawtio RBAC enabler bundle(s)
Feature has no configuration
Feature has no configuration files
Feature has no dependencies.
```

```
Feature contains followed bundles:
    mvn:io.hawt/hawtio-osgi-jmx/2.0.0.fuse-000117
Feature has no conditionals.
```

# CHAPTER 3. 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 3.1, “Preparing to set up Maven”
- Section 3.2, “Adding Red Hat repositories to Maven”
- Section 3.3, “Using local Maven repositories”
- Section 3.4, “Setting Maven mirror using environmental variables or system properties”
- Section 3.5, “About Maven artifacts and coordinates”

## 3.1. 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 3.3, “Using local Maven repositories” .

## 3.2. 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>
```

## 3.3. 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.

## 3.4. 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.

### 3.4.1. 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.

### 3.4.2. 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.

### 3.4.3. 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

### 3.4.4. 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.

## 3.5. 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***

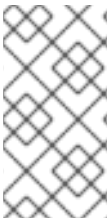
Enables you to distinguish between artifacts that were built from the same POM, but have different content.

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.