Red Hat Fuse 7.3

Migration Guide

Migrating to Red Hat Fuse 7.3

Last Updated: 2019-06-21

Red Hat Fuse 7.3 Migration Guide

Migrating to Red Hat Fuse 7.3

Legal Notice

Abstract

Use this guide to help you when when upgrading to the latest version of Red Hat Fuse.

Table of Contents

CHAPTER 1. MIGRATION PATHS FOR FUSE 7.3

1.1. MIGRATION PATH FOR FUSE 7.3 ON KARAF

Fuse provides automated tooling for upgrading standalone Fuse on Apache Karaf installations. This new upgrade mechanism is based on an enhanced version of the pre-existing Fuse patch mechanism, making it possible to upgrade to a new version of Fuse while preserving the configuration customizations you have already made. The Fuse 7.3.0 on Karaf installation archive can thus be used as an upgrade installer to upgrade from version 7.2.0 to version 7.3.0.

1.2. MIGRATION PATH FOR FUSE 7.3 ON EAP

There is no automated migration path to Fuse 7.3 on EAP from a previous version of Fuse on EAP. To migrate to Fuse 7.3 you will need to make a new installation of Fuse 7.3 on JBoss EAP. After a successful installation, any existing deployments will need to be re-deployed to the new system.

1.3. DEPRECATED AND REMOVED FEATURES

For the list of features that have been deprecated or removed in Fuse 7.3, see Release Notes.

CHAPTER 2. UPGRADING FUSE ON APACHE KARAF

2.1. UPGRADING OVERVIEW

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.

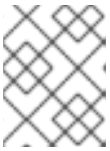
The upgrade installer file is the very same file that you would use to make a fresh installation of Fuse on Apache Karaf. To obtain the upgrade installer file, go to the **Downloads** page of the Red Hat customer portal and download the latest version of the installation archive for Fuse on Apache Karaf (for example, **fuse-karaf-7.3.0.fuse-730079-redhat-00001.zip**).

2.2. UPGRADE PROCEDURES FOR FUSE ON KARAF

2.2.1. Impact of upgrading

The upgrade mechanism can make updates to **any** installation files including **bundle JARs** and **static files** (including, for example, configuration files under the **etc/** directory). The Fuse on Apache Karaf upgrade process:

- Updates any files, including bundle JARs, configuration files, and any static files.
- Patches both the current container instance (and its runtime storage under the **data/** directory) and the underlying installation. Hence, patches are preserved after deleting a container instance.
- Updates all of the files related to Karaf features, including the features repository files and the features themselves. Hence, any features installed after the rollup patch will reference the correct patched dependencies.
- If necessary, updates configuration files (for example, files under **etc/**), automatically merging any configuration changes you have made with the configuration changes made by the patch. If merge conflicts occur, see the patch log for details of how they are handled.
- Most of the merge conflicts are resolved automatically. For example, the patch mechanism detects conflicts at property level for the property files. It detects whether it was a user or patch that changed any property. The change is preserved, if only one side changed the property.
- Tracks **all** of the changes made to the installation (including to static files), so that it is possible to roll back the patch.



NOTE

The rollup patching mechanism uses an internal git repository (located under **patches/.management/history**) to track the changes made.

2.2.2. Upgrading the Karaf container

To upgrade a standalone Apache Karaf container:

1. Make a full backup of your Fuse on Apache Karaf installation before upgrading.

2. Start the container, if it is not already running. If the container is running in the background (or remotely), connect to the container using the SSH console client, **bin/client**.

3. Add the upgrade installer file to the container's environment by invoking the **patch:add** command. For example, to add the **fuse-karaf-7.3.0.fuse-730079-redhat-00001.zip** upgrade installer file:

```
patch:add file:///path/to/fuse-karaf-7.3.0.fuse-730079-redhat-00001.zip
```

4. Run the **patch:update** command. There is no need to restart the container.

```
karaf@root()> patch:update
Current patch mechanism version: 7.1.0.fuse-710023-redhat-00001
New patch mechanism version detected: 7.2.0.fuse-720035-redhat-00001
Uninstalling patch features in version 7.1.0.fuse-710023-redhat-00001
Installing patch features in version 7.2.0.fuse-720035-redhat-00001
```

5. Invoke the **patch:list** command to display a list of upgrade installers. In this list, the entries under the **[name]** heading are upgrade IDs. For example:

```
karaf@root()> patch:list
[name]                                  [installed] [rollup] [description]
fuse-karaf-7.2.0.fuse-720035-redhat-00001 false      true     fuse-karaf-7.2.0.fuse-720035-
redhat-00001
```

6. Simulate the upgrade by invoking the **patch:simulate** command and specifying the upgrade ID for the upgrade that you want to apply, as follows:

```
karaf@root()> patch:simulate fuse-karaf-7.2.0.fuse-720035-redhat-00001
INFO : org.jboss.fuse.modules.patch.patch-management (226): Installing rollup patch "fuse-
karaf-7.2.0.fuse-720035-redhat-00001"
========== Repositories to remove (9):
 - mvn:io.hawt/hawtio-karaf/2.0.0.fuse-710018-redhat-00002/xml/features
...
========== Repositories to add (9):
 - mvn:io.hawt/hawtio-karaf/2.0.0.fuse-720044-redhat-00001/xml/features
...
========== Repositories to keep (10):
 - mvn:org.apache.activemq/artemis-features/2.4.0.amq-711002-redhat-1/xml/features
...
========== Features to update (100):
[name]                  [version]                  [new version]
aries-blueprint            4.2.0.fuse-710024-redhat-00002    4.2.0.fuse-720061-redhat-00001
...
========== Bundles to update as part of features or core bundles (100):
[symbolic name]                                  [version]                  [new location]
io.hawt.hawtio-log                               2.0.0.fuse-710018-redhat-00002
mvn:io.hawt/hawtio-log/2.0.0.fuse-720044-redhat-00001
...
========== Bundles to reinstall as part of features or core bundles (123):
[symbolic name]                                  [version]                  [location]
com.fasterxml.jackson.core.jackson-annotations             2.8.11
mvn:com.fasterxml.jackson.core/jackson-annotations/2.8.11
```

```
...
Simulation only - no files and runtime data will be modified.
karaf@root()>
```

This generates a log of the changes that will be made to the container when the upgrade is performed, but will not make any actual changes to the container. Review the simulation log to understand the changes that will be made to the container.

7. Upgrade the container by invoking the **patch:install** command and specifying the upgrade ID for the upgrade that you want to apply. For example:

```
karaf@root()> patch:install fuse-karaf-7.3.0.fuse-730079-redhat-00001
```

8. Validate the upgrade, by searching for one of the upgrade artifacts. For example, if you had just upgraded Fuse 7.1.0 to Fuse 7.2.0, you could search for bundles with the build number, 730079, as follows:

```
karaf@root()> bundle:list -l | grep 730079
 22 │ Active │  80 │ 7.3.0.fuse-730079-redhat-00001  │ mvn:org.jboss.fuse.modules/fuse-
pax-transx-tm-narayana/7.3.0.fuse-730079-redhat-00001
188 │ Active │  80 │ 7.3.0.fuse-730079-redhat-00001  │
mvn:org.jboss.fuse.modules.patch/patch-commands/7.3.0.fuse-730079-redhat-00001
```

After upgrading, you also see the new version and build number in the Welcome banner when you restart the container.

2.2.3. Rolling back an upgrade

Occasionally an upgrade might not work or might introduce new issues to a container. In these cases, you can easily roll back the upgrade and restore your system to its previous state using the **patch:rollback** command, as follows:

1. Invoke the **patch:list** command to obtain the upgrade ID, **UPGRADE_ID**, of the most recently installed patch.

2. Invoke the **patch:rollback** command, as follows:

```
patch:rollback UPGRADE_ID
```

In some cases the container needs to restart to roll back the upgrade. In these cases, the container restarts automatically. Due to the highly dynamic nature of the OSGi runtime, during the restart you might see some occasional errors related to incompatible classes. These errors are related to OSGi services that have just started or stopped and can be safely ignored.

CHAPTER 3. UPGRADING FUSE ON JBOSS EAP

There is no automated migration path to Fuse 7.2 on EAP from a previous version of Fuse on EAP. To migrate to Fuse 7.2 you will need to make a new installation of Fuse 7.2 on JBoss EAP. After a successful installation, any existing deployments will need to be re-deployed to the new system.

3.1. UPDATING EXISTING FUSE ON EAP APPLICATIONS

The recommended approach to upgrading existing Fuse on EAP applications is to modify the Maven bill-of-materials (BOM) file in your projects' **pom.xml** file. The BOM file encapsulates all of the versions for the productised Maven artifacts from Fuse. By exploiting the BOM file for Fuse on EAP, you can be confident that your Fuse applications are using a supported combination of Fuse artifacts. For more details, see Chapter 4, *Migrate Maven Projects*.

CHAPTER 4. MIGRATE MAVEN PROJECTS

To simplify migration of Maven projects, Fuse provides several Maven Bill of Materials (BOM) files. A common parent BOM file defines mutual dependencies. There is also a dedicated BOM file for each container that Fuse runs in:

- Apache Karaf
- JBoss EAP
- Spring Boot

Each BOM file is a set of Maven dependency versions that work well together. This removes the need to define the version individually for each Maven artifact.

You can find these BOM files here: https://github.com/jboss-fuse/redhat-fuse. The following sections provide details for using the BOM files to migrate your Maven projects.

4.1. BOM FILE FOR APACHE KARAF

The purpose of a Maven Bill of Materials (BOM) file is to provide a curated set of Maven dependency versions that work well together, saving you from having to define versions individually for every Maven artifact.

The Fuse BOM for Apache Karaf offers the following advantages:

- Defines versions for Maven dependencies, so that you do not need to specify the version when you add a dependency to your POM.
- Defines a set of curated dependencies that are fully tested and supported for a specific version of Fuse.
- Simplifies upgrades of Fuse.

IMPORTANT

Only the set of dependencies defined by a Fuse BOM are supported by Red Hat.

To incorporate a Maven BOM file into your Maven project, specify a **dependencyManagement** element in your project's **pom.xml** file (or, possibly, in a parent POM file), as shown in the following example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project ...>
  ...
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

    <!-- configure the versions you want to use here -->
    <fuse.version>7.3.0.fuse-730058-redhat-00001</fuse.version>

  </properties>

  <dependencyManagement>
    <dependencies>
      <dependency>
```

```
      <groupId>org.jboss.redhat-fuse</groupId>
      <artifactId>fuse-karaf-bom</artifactId>
      <version>${fuse.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
 </dependencyManagement>
 ...
</project>
```

NOTE

The **org.jboss.redhat-fuse** BOM is new in Fuse 7 and has been designed to simplify BOM versioning. The Fuse quickstarts and Maven archetypes still use the old style of BOM, however, as they have not yet been refactored to use the new one. Both BOMs are correct and you can use either one in your Maven projects. In an upcoming Fuse release, the quickstarts and Maven archetypes will be refactored to use the new BOM.

After specifying the BOM using the dependency management mechanism, it becomes possible to add Maven dependencies to your POM *without* specifying the version of the artifact. For example, to add a dependency for the **camel-velocity** component, you would add the following XML fragment to the **dependencies** element in your POM:

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-velocity</artifactId>
</dependency>
```

Note how the **version** element is omitted from this dependency definition.

4.2. BOM FILE FOR JBOSS EAP

The purpose of a Maven Bill of Materials (BOM) file is to provide a curated set of Maven dependency versions that work well together, saving you from having to define versions individually for every Maven artifact.

The Fuse BOM for JBoss EAP offers the following advantages:

- Defines versions for Maven dependencies, so that you do not need to specify the version when you add a dependency to your POM.
- Defines a set of curated dependencies that are fully tested and supported for a specific version of Fuse.
- Simplifies upgrades of Fuse.

IMPORTANT

Only the set of dependencies defined by a Fuse BOM are supported by Red Hat.

To incorporate a BOM file into your Maven project, specify a **dependencyManagement** element in your project's **pom.xml** file (or, possibly, in a parent POM file), as shown in the following example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project ...>
  ...
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

    <!-- configure the versions you want to use here -->
    <fuse.version>7.3.0.fuse-730058-redhat-00001</fuse.version>

  </properties>

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.jboss.redhat-fuse</groupId>
        <artifactId>fuse-eap-bom</artifactId>
        <version>${fuse.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
  ...
</project>
```

After specifying the BOM using the dependency management mechanism, it becomes possible to add Maven dependencies to your POM *without* specifying the version of the artifact. For example, to add a dependency for the **camel-velocity** component, you would add the following XML fragment to the **dependencies** element in your POM:

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-velocity</artifactId>
  <scope>provided</scope>
</dependency>
```

Note how the **version** element is omitted from this dependency definition.

4.3. BOM FILE FOR SPRING BOOT

The purpose of a Maven Bill of Materials (BOM) file is to provide a curated set of Maven dependency versions that work well together, saving you from having to define versions individually for every Maven artifact.

The Fuse BOM for Spring Boot offers the following advantages:

- Defines versions for Maven dependencies, so that you do not need to specify the version when you add a dependency to your POM.
- Defines a set of curated dependencies that are fully tested and supported for a specific version of Fuse.
- Simplifies upgrades of Fuse.

IMPORTANT

Only the set of dependencies defined by a Fuse BOM are supported by Red Hat.

To incorporate a BOM file into your Maven project, specify a **dependencyManagement** element in your project's **pom.xml** file (or, possibly, in a parent POM file), as shown in the following example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project ...>
  ...
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

    <!-- configure the versions you want to use here -->
    <fuse.version>7.3.0.fuse-730058-redhat-00001</fuse.version>
    <spring-boot.version>1.5.17.RELEASE</spring-boot.version>
  </properties>

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.jboss.redhat-fuse</groupId>
        <artifactId>fuse-springboot-bom</artifactId>
        <version>${fuse.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
  ...
</project>
```

After specifying the BOM using the dependency management mechanism, it becomes possible to add Maven dependencies to your POM *without* specifying the version of the artifact. For example, to add a dependency for the **camel-hystrix** component, you would add the following XML fragment to the **dependencies** element in your POM:

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-hystrix-starter</artifactId>
</dependency>
```

Note how the Camel artifact ID is specified with the **-starter** suffix – that is, you specify the Camel Hystrix component as **camel-hystrix-starter**, not as **camel-hystrix**. The Camel starter components are packaged in a way that is optimized for the Spring Boot environment.