



# Red Hat Fuse 7.9

## Migration Guide

Migrate to Red Hat Fuse 7.9



# Red Hat Fuse 7.9 Migration Guide

---

Migrate to Red Hat Fuse 7.9

## Legal Notice

Copyright © 2021 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

XFS<sup>®</sup> is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js<sup>®</sup> is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack<sup>®</sup> Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## Abstract

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

---

## Table of Contents

<b>PREFACE</b> .....	<b>3</b>
<b>MAKING OPEN SOURCE MORE INCLUSIVE</b> .....	<b>4</b>
<b>CHAPTER 1. UPGRADING FUSE ONLINE</b> .....	<b>5</b>
1.1. UPGRADING FUSE ONLINE BY USING THE OPERATORHUB (OCP 4.X)	5
1.2. UPGRADING FUSE ONLINE BY USING THE INSTALL SCRIPT (OCP 3.11)	6
1.3. UPGRADING FUSE ONLINE INTEGRATIONS	8
1.3.1. Camel migration considerations	8
<b>CHAPTER 2. UPGRADE TO SPRING BOOT 2</b> .....	<b>11</b>
2.1. BEFORE YOU BEGIN	11
2.2. UPGRADE FROM SPRING BOOT 1 TO SPRING BOOT 2	11
<b>CHAPTER 3. UPGRADING FUSE APPLICATIONS ON SPRING BOOT STANDALONE</b> .....	<b>13</b>
3.1. CAMEL MIGRATION CONSIDERATIONS	13
3.2. ABOUT MAVEN DEPENDENCIES	15
3.3. UPDATING YOUR FUSE PROJECT'S MAVEN DEPENDENCIES	16
<b>CHAPTER 4. UPGRADING FUSE APPLICATIONS ON JBOSS EAP STANDALONE</b> .....	<b>18</b>
4.1. CAMEL MIGRATION CONSIDERATIONS	18
4.2. ABOUT MAVEN DEPENDENCIES	20
4.3. UPDATING YOUR FUSE PROJECT'S MAVEN DEPENDENCIES	21
4.4. UPGRADING YOUR JAVA EE DEPENDENCIES	22
4.5. UPGRADING AN EXISTING FUSE ON JBOSS EAP INSTALLATION	22
4.6. UPGRADING FUSE AND JBOSS EAP SIMULTANEOUSLY	23
<b>CHAPTER 5. UPGRADING FUSE APPLICATIONS ON KARAF STANDALONE</b> .....	<b>24</b>
5.1. CAMEL MIGRATION CONSIDERATIONS	24
5.2. ABOUT MAVEN DEPENDENCIES	26
5.3. UPDATING YOUR FUSE PROJECT'S MAVEN DEPENDENCIES	27
<b>CHAPTER 6. UPGRADING FUSE STANDALONE ON KARAF</b> .....	<b>28</b>
6.1. IMPACT OF UPGRADING FUSE ON KARAF	28
6.2. UPGRADING FUSE STANDALONE ON KARAF	28
6.3. ROLLING BACK AN UPGRADE FOR FUSE ON KARAF	30



---

## PREFACE

This guide provides information on updating Red Hat Fuse and Fuse applications:



### NOTE

If you want to migrate from Fuse 6 to the latest Fuse 7 release, before you follow the instructions in this guide, you should follow the instructions in the [Red Hat Fuse 7.0 Migration Guide](#).

*Chapter 1, Upgrading Fuse Online*

*Chapter 3, Upgrading Fuse applications on Spring Boot standalone*

*Chapter 4, Upgrading Fuse applications on JBoss EAP standalone*

*Chapter 5, Upgrading Fuse applications on Karaf standalone*

*Chapter 6, Upgrading Fuse Standalone on Karaf*

## 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. UPGRADING FUSE ONLINE

For OCP 4.x, use the OpenShift OperatorHub to upgrade from Fuse 7.8 to 7.9, regardless of how you installed Fuse 7.8 (by using the install script or the OperatorHub).

For OCP 3.11, use the install script to upgrade from Fuse 7.8 to 7.9.

You should determine whether upgrading to Fuse 7.9 requires you to make changes to your existing integrations. Even if no changes are required, you must republish any running integrations.

## 1.1. UPGRADING FUSE ONLINE BY USING THE OPERATORHUB (OCP 4.X)

Use the OpenShift OperatorHub to upgrade from Fuse Online 7.8 to 7.9, regardless of how you installed Fuse Online 7.8 (by using the install script or the OperatorHub).

The procedure for upgrading to Fuse Online 7.9 depends on how you installed your existing Fuse Online:

- You installed Fuse Online 7.8 or an earlier 7.9 version by using the OperatorHub
- You installed Fuse Online 7.8 by using the install script.

**Note:** Fuse Online 7.9 requires OpenShift Container Platform (OCP) 4.6 or later. If you are using OCP 4.5 or earlier, you must upgrade to OCP 4.6 or later, if you want to upgrade to Fuse Online 7.9.

### Upgrading from a previous Fuse Online OperatorHub installation

The upgrade process from a Fuse Online 7.8 or an earlier 7.9 version to a newer Fuse Online 7.9 version depends on the **Approval Strategy** that you selected when you installed Fuse Online:

- For **Automatic** updates, when a new version of the Fuse Online operator is available, the OpenShift Operator Lifecycle Manager (OLM) automatically upgrades the running instance of the Fuse Online without human intervention.
- For **Manual** updates, when a newer version of an Operator is available, the OLM creates an update request. As a cluster administrator, you must then manually approve that update request to have the Fuse Online operator updated to the new version as described in the [Manually approving a pending Operator upgrade](#) section of the OpenShift documentation.

During and after an infrastructure upgrade, existing integrations continue to run with the *older* versions of Fuse Online libraries and dependencies.

To have existing integrations run with the updated Fuse Online version, you must republish the integrations.

### Upgrading from a Fuse Online 7.8 install script installation

If you installed Fuse Online 7.8 by using the install script, follow the steps in this procedure to use the OperatorHub to upgrade to Fuse Online 7.9.

#### Prerequisite

- Your existing Fuse Online 7.8 install script installation includes an operator, operands, and a syndesis custom resource.

#### Procedure

To upgrade by using the Fuse Online operator:

1. In the OpenShift web console, click **Operators > OperatorHub**.
2. Select the Fuse Online 7.9 operator and then click **Install**.
3. Select the namespace that contains the Fuse Online 7.8 installation and then click **Install**.
4. After a successful upgrade, you must republish all existing integrations.

## 1.2. UPGRADING FUSE ONLINE BY USING THE INSTALL SCRIPT (OCP 3.11)

For OCP 4.x, if you installed Fuse Online 7.8 by using the install script, upgrade to version 7.9 by using the OperatorHub as described in [Upgrading Fuse Online by using the OperatorHub](#) .

For OCP 3.11, the install script procedure for the following upgrades is the same:

- From Fuse Online 7.8 to Fuse Online 7.9
- From a Fuse Online 7.9 version to a newer Fuse Online 7.9 version

### Prerequisites

- You installed and are running version 7.8 of Fuse Online on OCP 3.11 on-site **OR** you installed and are running a version of 7.9 of Fuse Online on OCP 3.11 and you want to upgrade to fresh application images.

For earlier versions:

- If you are running version 7.7 of Fuse Online on OCP, then you must [upgrade to 7.8](#) and then you can upgrade to 7.9.
- If you are running version 7.6 of Fuse Online on OCP, then you must [upgrade to 7.7](#) and then you can upgrade to 7.8.
- If you are running version 7.5 of Fuse Online on OCP, then you must [upgrade to 7.6](#) and then you can upgrade to 7.7.
- If you are running version 7.4 of Fuse Online on OCP, then you must [upgrade to 7.5](#) and then you can upgrade to 7.6.
- If you are running version 7.3 of Fuse Online on OCP, then you must [upgrade to 7.4](#) and then you can upgrade to 7.5.
- If you are running version 7.2 of Fuse Online on OCP, then you must [upgrade to 7.3](#).
- If you are running version 7.1 of Fuse Online on OCP, then you must [upgrade to 7.2](#).
- You installed the **oc** client tool and it is connected to the OCP cluster in which Fuse Online is installed.
- You have cluster administration permissions, which are required for the first two steps in this procedure.

### Procedure

1. To avoid encountering a possible Docker limit error, a cluster administrator sets up access to Docker images as described in [Access Docker images before an upgrade](#).
2. A cluster administrator downloads the Fuse Online package and grants permission for a user to upgrade Fuse Online in a particular project:
  - a. Download the package containing the Fuse Online installation scripts from the following location:  
<https://github.com/syndesio/fuse-online-install/releases/tag/1.12>  
  
 Unpack the downloaded archive at a convenient location on your file system. The **fuse-online-install-1.12** directory contains the scripts and supporting files for upgrading Fuse Online.
  - b. Change to the directory that contains the extracted archive. For example:  
**cd fuse-online-install-1.12**
  - c. Log in to OpenShift with a cluster administration account, for example:  
**oc login -u admin -p admin**
  - d. Change to the OpenShift project in which Fuse Online needs to be upgraded, for example:  
**oc project fuse-online-project**
  - e. Update the Fuse Online custom resource definition:  
**bash install\_ocp.sh --setup**
  - f. Grant permission for upgrading Fuse Online in just this project. For example, the following command grants permission for upgrading Fuse Online to the **developer** user. After the cluster administrator runs this command, the **developer** user can upgrade Fuse Online in only this project, which is **fuse-online-project**, in this example:  
**bash install\_ocp.sh --grant developer**
3. The user who was granted permission to upgrade Fuse Online performs the upgrade:
  - a. Log in to OpenShift, for example:  
**oc login -u developer**
  - b. Switch to the project in which you want to upgrade Fuse Online, for example:  
**oc project fuse-online-project**
  - c. To check which version you are about to upgrade to, run the update script with the **--version** option, as follows:  
**bash update\_ocp.sh --version**
  - d. Invoke the update script as follows:  
**bash update\_ocp.sh**  
  
 To learn more about the script, invoke **bash update\_ocp.sh --help**.  
  
 During and after an infrastructure upgrade, existing integrations continue to run with the *older* versions of Fuse Online libraries and dependencies.
4. Upgrade Fuse Online integrations that are running as follows:
  - a. In Fuse Online, select the integration that you want to upgrade.
  - b. Select **Edit**.

- c. Select **Publish** to republish the integration.

Republishing the integration forces a rebuild that uses the latest Fuse Online dependencies.

## 1.3. UPGRADING FUSE ONLINE INTEGRATIONS


When you upgrade to Fuse Online 7.9, you should determine whether you need to make changes to your existing integrations.

Review the Apache Camel updates described in [Camel Migration Considerations](#).

Even if your integrations do not require changes, you must republish any running integrations because during and after an infrastructure upgrade, existing integrations continue to run with the older versions of Fuse Online libraries and dependencies. To have them run with the updated versions, you must republish them.

### Procedure

To republish your integrations, in your Fuse Online environment:

1. In the Fuse Online left navigation panel, click **Integrations**.
2. For each integration:
  - a. To the right of the integration entry, click  and select **Edit**.
  - b. When Fuse Online displays the integration for editing, in the upper right, click **Publish**.

Publishing forces a rebuild that uses the latest Fuse Online dependencies.



### NOTE

The Fuse Online user interface shows a warning if any element of an integration has a newer dependency that you need to update.

### 1.3.1. Camel migration considerations

Red Hat Fuse uses Apache Camel 2.23. You should consider the following updates to Camel 2.22 and 2.23 when you upgrade to Fuse 7.8.

#### Camel 2.22 updates

- Camel has upgraded from Spring Boot v1 to v2 and therefore v1 is no longer supported.
- Upgraded to Spring Framework 5. Camel should work with Spring 4.3.x as well, but going forward Spring 5.x will be the minimum Spring version in future releases.
- Upgraded to Karaf 4.2. You may run Camel on Karaf 4.1 but we only officially support Karaf 4.2 in this release.
- Optimized using toD DSL to reuse endpoints and producers for components where it is possible. For example, HTTP based components will now reuse producer (HTTP clients) with dynamic URIs sending to the same host.
- The File2 consumer with read-lock idempotent/idempotent-changed can now be configured to

delay the release tasks to expand the window when a file is regarded as in-process, which is usable in active/active cluster settings with a shared idempotent repository to ensure other nodes don't too quickly see a processed file as a file they can process (only needed if you have `readLockRemoveOnCommit=true`).

- Allow to plugin a custom request/reply correlation id manager implementation on Netty4 producer in request/reply mode. The Twitter component now uses extended mode by default to support tweets greater than 140 characters
- Rest DSL producer now supports being configured in REST configuration by using `endpointProperties`.
- The Kafka component now supports `HeaderFilterStrategy` to plugin custom implementations for controlling header mappings between Camel and Kafka messages.
- REST DSL now supports client request validation to validate that `Content-Type/Accept` headers are possible for the REST service.
- Camel now has a Service Registry SPI which allows you to register routes to a service registry (such as consul, etcd, or zookeeper) by using a Camel implementation or Spring Cloud.
- The SEDA component now has a default queue size of 1000 instead of unlimited.
- The following noteworthy issues have been fixed:
  - Fixed a CXF continuation timeout issue with camel-cxf consumer that could cause the consumer to return a response with data instead of triggering a timeout to the calling SOAP client.
  - Fixed camel-cxf consumer doesn't release UoW when using a robust one-way operation.
  - Fixed using `AdviceWith` and using `weave` methods on `onException` etc. not working.
  - Fixed Splitter in parallel processing and streaming mode may block, while iterating message body when the iterator throws an exception in the first invoked `next()` method call.
  - Fixed Kafka consumer to not auto commit if `autoCommitEnable=false`.
  - Fixed file consumer was using `markerFile` as read-lock by default, which should have been none.
  - Fixed using manual commit with Kafka to provide the current record offset and not the previous (and -1 for first).
  - Fixed Content Based Router in Java DSL may not resolve property placeholders in when predicates.

### Camel 2.23 updates

- Upgraded to Spring Boot 2.1.
- Additional component-level options can now be configured by using spring-boot auto-configuration. These options are included in the spring-boot component metadata JSON file descriptor for tooling assistance.
- Added a documentation section that includes all the Spring Boot auto configuration options for all the components, data-formats, and languages.

- All the Camel Spring Boot starter JARs now include **META-INF/spring-autoconfigure-metadata.properties** file in their JARs to optimize Spring Boot auto-configuration.
- The Throttler now supports correlation groups based on dynamic expression so that you can group messages into different throttled sets.
- The Hystrix EIP now allows inheritance for Camel's error handler so that you can retry the entire Hystrix EIP block again if you have enabled error handling with redeliveries.
- SQL and EISql consumers now support dynamic query parameters in route form. Note that this feature is limited to calling beans by using simple expressions.
- The swagger-restdsl maven plugin now supports generating DTO model classes from the Swagger specification file.
- The following noteworthy issues have been fixed:
  - The Aggregator2 has been fixed to not propagate control headers for forcing completion of all groups, so it will not happen again if another aggregator EIP is in use later during routing.
  - Fixed Tracer not working if redelivery was activated in the error handler.
  - The built-in type converter for XML Documents may output parsing errors to stdout, which has now been fixed to output by using the logging API.
  - Fixed SFTP writing files by using the charset option would not work if the message body was streaming-based.
  - Fixed Zipkin root id to not be reused when routing over multiple routes to group them together into a single parent span.
  - Fixed optimized toD when using HTTP endpoints had a bug when the hostname contains an IP address with digits.
  - Fixed issue with RabbitMQ with request/reply over temporary queues and using manual acknowledge mode. It would not acknowledge the temporary queue (which is needed to make request/reply possible).
  - Fixed various HTTP consumer components that may not return all allowed HTTP verbs in Allow header for OPTIONS requests (such as when using rest-dsl).
  - Fixed the thread-safety issue with FluentProducerTemplate.

## CHAPTER 2. UPGRADE TO SPRING BOOT 2

This chapter explains how to upgrade your applications to Spring Boot 2.0 from Spring Boot 1.

### 2.1. BEFORE YOU BEGIN

Before you start the migration to Spring Boot 2, you must review the system requirements and dependencies.

- Upgrade to the latest 1.5.x version
  - Before you start, upgrade to the latest 1.5.x available version. This is to ensure that you are building against the most recent dependencies of that line.
- Review dependencies
  - The migration to Spring Boot 2 will result in upgrading a number of dependencies. Review the dependency management for 1.5.x with the dependency management for 2.0.x to assess how your project is affected.
  - Identify the compatible versions for the dependencies that are not managed by Spring Boot and then define the explicit versions for these.
- Review custom configuration
  - Any custom configuration that your project defines might need to be reviewed on upgrade. If this can be replaced by the use of standard auto-configuration, do it so before upgrading.
- Review system requirements
  - Spring Boot 2.0 requires Java 8 or later.
  - It also requires Spring Framework 5.0.
  - Java 6 and 7 are no longer supported.

### 2.2. UPGRADE FROM SPRING BOOT 1 TO SPRING BOOT 2

Once you have reviewed the state of your project and its dependencies, upgrade to the latest maintenance release of Spring Boot 2.x. It is recommended to upgrade in the phases. For example, first upgrade from Spring Boot 1.5 to Spring Boot 2.0 and then upgrade to 2.1 and then to the latest maintenance release of Spring Boot 2.

#### Migrating configuration properties

With Spring Boot 2.0, many configuration properties were renamed or removed. Hence you need to update the **application.properties/application.yml** accordingly. You can achieve that with the help of a new **spring-boot-properties-migrator** module. Once added as a dependency to your project, this will not only analyze your application's environment and print diagnostics at startup, but also temporarily migrate properties at runtime for you.

#### Procedure

1. Add **spring-boot-properties-migrator** module to dependency section of your project's **pom.xml**.

```
<dependency>  
<groupId>org.springframework.boot</groupId>  
<artifactId>spring-boot-properties-migrator</artifactId>  
<scope>runtime</scope>  
</dependency>
```

```
runtime("org.springframework.boot:spring-boot-properties-migrator")
```

**NOTE**

Once you're done with the migration, please make sure to remove this module from your project's dependencies.



## CHAPTER 3. UPGRADING FUSE APPLICATIONS ON SPRING BOOT STANDALONE

To upgrade your Fuse applications on Spring Boot:

- You should consider Apache Camel updates as described in [Section 3.1, “Camel migration considerations”](#).
- You must update your Fuse project’s Maven dependencies to ensure that you are using the correct version of Fuse.

Typically, you use Maven to build Fuse applications. Maven is a free and open source build tool from Apache. Maven configuration is defined in a Fuse application project’s **pom.xml** file. While building a Fuse project, the default behavior is that Maven searches external repositories and downloads the required artifacts. You add a dependency for the Fuse Bill of Materials (BOM) to the **pom.xml** file so that the Maven build process picks up the correct set of Fuse supported artifacts.

The following sections provide information on Maven dependencies and how to update them in your Fuse projects.

- [Section 3.2, “About Maven dependencies”](#)
- [Section 3.3, “Updating your Fuse project’s Maven dependencies”](#)

### 3.1. CAMEL MIGRATION CONSIDERATIONS

Red Hat Fuse uses Apache Camel 2.23. You should consider the following updates to Camel 2.22 and 2.23 when you upgrade to Fuse 7.8.

#### Camel 2.22 updates

- Camel has upgraded from Spring Boot v1 to v2 and therefore v1 is no longer supported.
- Upgraded to Spring Framework 5. Camel should work with Spring 4.3.x as well, but going forward Spring 5.x will be the minimum Spring version in future releases.
- Upgraded to Karaf 4.2. You may run Camel on Karaf 4.1 but we only officially support Karaf 4.2 in this release.
- Optimized using toD DSL to reuse endpoints and producers for components where it is possible. For example, HTTP based components will now reuse producer (HTTP clients) with dynamic URIs sending to the same host.
- The File2 consumer with read-lock idempotent/idempotent-changed can now be configured to delay the release tasks to expand the window when a file is regarded as in-process, which is usable in active/active cluster settings with a shared idempotent repository to ensure other nodes don’t too quickly see a processed file as a file they can process (only needed if you have readLockRemoveOnCommit=true).
- Allow to plugin a custom request/reply correlation id manager implementation on Netty4 producer in request/reply mode. The Twitter component now uses extended mode by default to support tweets greater than 140 characters
- Rest DSL producer now supports being configured in REST configuration by using endpointProperties.

- The Kafka component now supports HeaderFilterStrategy to plugin custom implementations for controlling header mappings between Camel and Kafka messages.
- REST DSL now supports client request validation to validate that Content-Type/Accept headers are possible for the REST service.
- Camel now has a Service Registry SPI which allows you to register routes to a service registry (such as consul, etcd, or zookeeper) by using a Camel implementation or Spring Cloud.
- The SEDA component now has a default queue size of 1000 instead of unlimited.
- The following noteworthy issues have been fixed:
  - Fixed a CXF continuation timeout issue with camel-cxf consumer that could cause the consumer to return a response with data instead of triggering a timeout to the calling SOAP client.
  - Fixed camel-cxf consumer doesn't release UoW when using a robust one-way operation.
  - Fixed using AdviceWith and using weave methods on onException etc. not working.
  - Fixed Splitter in parallel processing and streaming mode may block, while iterating message body when the iterator throws an exception in the first invoked next() method call.
  - Fixed Kafka consumer to not auto commit if autoCommitEnable=false.
  - Fixed file consumer was using markerFile as read-lock by default, which should have been none.
  - Fixed using manual commit with Kafka to provide the current record offset and not the previous (and -1 for first).
  - Fixed Content Based Router in Java DSL may not resolve property placeholders in when predicates.

### Camel 2.23 updates

- Upgraded to Spring Boot 2.1.
- Additional component-level options can now be configured by using spring-boot auto-configuration. These options are included in the spring-boot component metadata JSON file descriptor for tooling assistance.
- Added a documentation section that includes all the Spring Boot auto configuration options for all the components, data-formats, and languages.
- All the Camel Spring Boot starter JARs now include **META-INF/spring-autoconfigure-metadata.properties** file in their JARs to optimize Spring Boot auto-configuration.
- The Throttler now supports correlation groups based on dynamic expression so that you can group messages into different throttled sets.
- The Hystrix EIP now allows inheritance for Camel's error handler so that you can retry the entire Hystrix EIP block again if you have enabled error handling with redeliveries.
- SQL and EISql consumers now support dynamic query parameters in route form. Note that this feature is limited to calling beans by using simple expressions.

- The swagger-restdsl maven plugin now supports generating DTO model classes from the Swagger specification file.
- The following noteworthy issues have been fixed:
  - The Aggregator2 has been fixed to not propagate control headers for forcing completion of all groups, so it will not happen again if another aggregator EIP is in use later during routing.
  - Fixed Tracer not working if redelivery was activated in the error handler.
  - The built-in type converter for XML Documents may output parsing errors to stdout, which has now been fixed to output by using the logging API.
  - Fixed SFTP writing files by using the charset option would not work if the message body was streaming-based.
  - Fixed Zipkin root id to not be reused when routing over multiple routes to group them together into a single parent span.
  - Fixed optimized toD when using HTTP endpoints had a bug when the hostname contains an IP address with digits.
  - Fixed issue with RabbitMQ with request/reply over temporary queues and using manual acknowledge mode. It would not acknowledge the temporary queue (which is needed to make request/reply possible).
  - Fixed various HTTP consumer components that may not return all allowed HTTP verbs in Allow header for OPTIONS requests (such as when using rest-dsl).
  - Fixed the thread-safety issue with FluentProducerTemplate.

## 3.2. ABOUT MAVEN DEPENDENCIES

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.

There is a dedicated BOM file for each container in which Fuse runs.

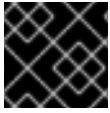


### NOTE

You can find these BOM files here: <https://github.com/jboss-fuse/redhat-fuse>. Alternatively, go to the [latest Release Notes](#) for information on BOM file updates.

The Fuse BOM 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.xml** file.
- 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.

### 3.3. UPDATING YOUR FUSE PROJECT'S MAVEN DEPENDENCIES

To upgrade your Fuse application for Spring Boot, update your project's Maven dependencies.

**Procedure**

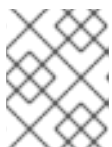
1. Open your project's **pom.xml** file.
2. Add a **dependencyManagement** element in your project's **pom.xml** file (or, possibly, in a parent **pom.xml** file), as shown in the following example:

```
<?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>7.9.0.fuse-sb2-790065-redhat-00001</fuse.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>
```

**NOTE**

Ensure you update your Spring Boot version as well. This is typically found under the Fuse version in the **pom.xml** file:

```
<properties>
  <!-- configure the versions you want to use here -->
  <fuse.version>7.9.0.fuse-sb2-790065-redhat-00001</fuse.version>
  <spring-boot.version>2.3.9.RELEASE</spring-boot.version>
</properties>
```

3. Save your **pom.xml** file.

After you specify the BOM as a dependency in your **pom.xml** file, it becomes possible to add Maven

dependencies to your **pom.xml** file *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.xml** file:

```
<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.

## CHAPTER 4. UPGRADING FUSE APPLICATIONS ON JBOSS EAP STANDALONE

To upgrade your Fuse applications on JBoss EAP:

- You should consider Apache Camel updates as described in [Section 4.1, “Camel migration considerations”](#).
- You must update your Fuse project’s Maven dependencies to ensure that you are using the correct version of Fuse.  
Typically, you use Maven to build Fuse applications. Maven is a free and open source build tool from Apache. Maven configuration is defined in a Fuse application project’s **pom.xml** file. While building a Fuse project, the default behavior is that Maven searches external repositories and downloads the required artifacts. You add a dependency for the Fuse Bill of Materials (BOM) to the **pom.xml** file so that the Maven build process picks up the correct set of Fuse supported artifacts.

The following sections provide information on Maven dependencies and how to update them in your Fuse projects.

- [Section 4.2, “About Maven dependencies”](#)
- [Section 4.3, “Updating your Fuse project’s Maven dependencies”](#)
- You must update your Fuse project’s Maven dependencies to ensure that you are using the upgraded versions of the Java EE dependencies as described in [Section 4.4, “Upgrading your Java EE dependencies”](#).

### 4.1. CAMEL MIGRATION CONSIDERATIONS

Red Hat Fuse uses Apache Camel 2.23. You should consider the following updates to Camel 2.22 and 2.23 when you upgrade to Fuse 7.8.

#### Camel 2.22 updates

- Camel has upgraded from Spring Boot v1 to v2 and therefore v1 is no longer supported.
- Upgraded to Spring Framework 5. Camel should work with Spring 4.3.x as well, but going forward Spring 5.x will be the minimum Spring version in future releases.
- Upgraded to Karaf 4.2. You may run Camel on Karaf 4.1 but we only officially support Karaf 4.2 in this release.
- Optimized using toD DSL to reuse endpoints and producers for components where it is possible. For example, HTTP based components will now reuse producer (HTTP clients) with dynamic URIs sending to the same host.
- The File2 consumer with read-lock idempotent/idempotent-changed can now be configured to delay the release tasks to expand the window when a file is regarded as in-process, which is usable in active/active cluster settings with a shared idempotent repository to ensure other nodes don’t too quickly see a processed file as a file they can process (only needed if you have `readLockRemoveOnCommit=true`).

- Allow to plugin a custom request/reply correlation id manager implementation on Netty4 producer in request/reply mode. The Twitter component now uses extended mode by default to support tweets greater than 140 characters
- Rest DSL producer now supports being configured in REST configuration by using `endpointProperties`.
- The Kafka component now supports `HeaderFilterStrategy` to plugin custom implementations for controlling header mappings between Camel and Kafka messages.
- REST DSL now supports client request validation to validate that `Content-Type/Accept` headers are possible for the REST service.
- Camel now has a Service Registry SPI which allows you to register routes to a service registry (such as consul, etcd, or zookeeper) by using a Camel implementation or Spring Cloud.
- The SEDA component now has a default queue size of 1000 instead of unlimited.
- The following noteworthy issues have been fixed:
  - Fixed a CXF continuation timeout issue with camel-cxf consumer that could cause the consumer to return a response with data instead of triggering a timeout to the calling SOAP client.
  - Fixed camel-cxf consumer doesn't release UoW when using a robust one-way operation.
  - Fixed using `AdviceWith` and using `weave` methods on `onException` etc. not working.
  - Fixed Splitter in parallel processing and streaming mode may block, while iterating message body when the iterator throws an exception in the first invoked `next()` method call.
  - Fixed Kafka consumer to not auto commit if `autoCommitEnable=false`.
  - Fixed file consumer was using `markerFile` as read-lock by default, which should have been none.
  - Fixed using manual commit with Kafka to provide the current record offset and not the previous (and -1 for first).
  - Fixed Content Based Router in Java DSL may not resolve property placeholders in when predicates.

### Camel 2.23 updates

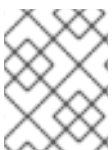
- Upgraded to Spring Boot 2.1.
- Additional component-level options can now be configured by using spring-boot auto-configuration. These options are included in the spring-boot component metadata JSON file descriptor for tooling assistance.
- Added a documentation section that includes all the Spring Boot auto configuration options for all the components, data-formats, and languages.
- All the Camel Spring Boot starter JARs now include **META-INF/spring-autoconfigure-metadata.properties** file in their JARs to optimize Spring Boot auto-configuration.

- The Throttler now supports correlation groups based on dynamic expression so that you can group messages into different throttled sets.
- The Hystrix EIP now allows inheritance for Camel's error handler so that you can retry the entire Hystrix EIP block again if you have enabled error handling with redeliveries.
- SQL and EISql consumers now support dynamic query parameters in route form. Note that this feature is limited to calling beans by using simple expressions.
- The swagger-restdsl maven plugin now supports generating DTO model classes from the Swagger specification file.
- The following noteworthy issues have been fixed:
  - The Aggregator2 has been fixed to not propagate control headers for forcing completion of all groups, so it will not happen again if another aggregator EIP is in use later during routing.
  - Fixed Tracer not working if redelivery was activated in the error handler.
  - The built-in type converter for XML Documents may output parsing errors to stdout, which has now been fixed to output by using the logging API.
  - Fixed SFTP writing files by using the charset option would not work if the message body was streaming-based.
  - Fixed Zipkin root id to not be reused when routing over multiple routes to group them together into a single parent span.
  - Fixed optimized toD when using HTTP endpoints had a bug when the hostname contains an IP address with digits.
  - Fixed issue with RabbitMQ with request/reply over temporary queues and using manual acknowledge mode. It would not acknowledge the temporary queue (which is needed to make request/reply possible).
  - Fixed various HTTP consumer components that may not return all allowed HTTP verbs in Allow header for OPTIONS requests (such as when using rest-dsl).
  - Fixed the thread-safety issue with FluentProducerTemplate.

## 4.2. ABOUT MAVEN DEPENDENCIES

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.

There is a dedicated BOM file for each container in which Fuse runs.



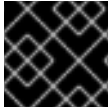
### NOTE

You can find these BOM files here: <https://github.com/jboss-fuse/redhat-fuse>. Alternatively, go to the [latest Release Notes](#) for information on BOM file updates.

The Fuse BOM 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.xml** file.
- 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.

## 4.3. UPDATING YOUR FUSE PROJECT'S MAVEN DEPENDENCIES

To upgrade your Fuse application for JBoss EAP, update your project's Maven dependencies.

### Procedure

1. Open your project's **pom.xml** file.
2. Add a **dependencyManagement** element in your project's **pom.xml** file (or, possibly, in a parent **pom.xml** 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>7.9.0.fuse-sb2-790065-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>
```

3. Save your **pom.xml** file.

After you specify the BOM as a dependency in your **pom.xml** file, it becomes possible to add Maven dependencies to your **pom.xml** file *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.xml** file:

```
<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.

## 4.4. UPGRADING YOUR JAVA EE DEPENDENCIES

In Fuse 7.8, some managed dependencies in the BOM file have updated **groupId** or **artifactId** properties, therefore you must update your project's **pom.xml** file accordingly.

### Procedure

1. Open your project's **pom.xml** file.
2. To change the **org.jboss.spec.javaee.transaction** version from 1.2 to 1.3 and the **org.jboss.spec.javaee.servlet** version from 3.1 to 4.0, update the dependencies as shown in the following example:

```
<dependency>
  <groupId>org.jboss.spec.javaee.transaction</groupId>
  <artifactId>jboss-transaction-api_1.3_spec</artifactId>
</dependency>

<dependency>
  <groupId>org.jboss.spec.javaee.servlet</groupId>
  <artifactId>jboss-servlet-api_4.0_spec</artifactId>
</dependency>
```

3. To migrate from Java EE API to Jakarta EE, replace **javax.\*** with **jakarta.\*** for each **groupId** and modify the **artifactId** for individual dependencies as shown in the following example:

```
<dependency>
  <groupId>jakarta.validation</groupId>
  <artifactId>jakarta.validation-api</artifactId>
</dependency>

<dependency>
  <groupId>jakarta.enterprise</groupId>
  <artifactId>jakarta.enterprise.cdi-api</artifactId>
</dependency>

<dependency>
  <groupId>jakarta.inject</groupId>
  <artifactId>jakarta.inject-api</artifactId>
</dependency>
```

## 4.5. UPGRADING AN EXISTING FUSE ON JBOSS EAP INSTALLATION

The following procedure describes how to upgrade an existing Fuse on JBoss EAP installation.

## Procedure

1. To upgrade from one JBoss EAP minor release to another, you should follow the instructions in the [JBoss EAP Patching and Upgrading Guide](#) guide.
2. To update Fuse, you must run the Fuse on JBoss EAP installer as described in the [Installing on JBoss EAP](#) guide.



### NOTE

You should not need to recompile or redploy your Fuse application.

## 4.6. UPGRADING FUSE AND JBOSS EAP SIMULTANEOUSLY

The following procedure describes how to upgrade a Fuse installation and the JBoss EAP runtime simultaneously, for example, if you are migrating from Fuse 7.7 on JBoss EAP 7.2 to Fuse 7.8 on JBoss EAP 7.3.



### WARNING

When upgrading **both** Fuse and the JBoss EAP runtime, Red Hat recommends that you perform a fresh installation of both Fuse and the JBoss EAP runtime.

## Procedure

1. To perform a new installation of the JBoss EAP runtime, follow the instructions in the [Installing on JBoss EAP](#) guide.
2. To perform a new installation of Fuse, run the Fuse on JBoss EAP installer as described in the [Installing on JBoss EAP](#) guide.

## CHAPTER 5. UPGRADING FUSE APPLICATIONS ON KARAF STANDALONE

To upgrade your Fuse applications on Karaf:

- You should consider Apache Camel updates as described in [Section 5.1, “Camel migration considerations”](#).
- You must update your Fuse project’s Maven dependencies to ensure that you are using the correct version of Fuse.

Typically, you use Maven to build Fuse applications. Maven is a free and open source build tool from Apache. Maven configuration is defined in a Fuse application project’s **pom.xml** file. While building a Fuse project, the default behavior is that Maven searches external repositories and downloads the required artifacts. You add a dependency for the Fuse Bill of Materials (BOM) to the **pom.xml** file so that the Maven build process picks up the correct set of Fuse supported artifacts.

The following sections provide information on Maven dependencies and how to update them in your Fuse projects.

- [Section 5.2, “About Maven dependencies”](#)
- [Section 5.3, “Updating your Fuse project’s Maven dependencies”](#)

### 5.1. CAMEL MIGRATION CONSIDERATIONS

Red Hat Fuse uses Apache Camel 2.23. You should consider the following updates to Camel 2.22 and 2.23 when you upgrade to Fuse 7.8.

#### Camel 2.22 updates

- Camel has upgraded from Spring Boot v1 to v2 and therefore v1 is no longer supported.
- Upgraded to Spring Framework 5. Camel should work with Spring 4.3.x as well, but going forward Spring 5.x will be the minimum Spring version in future releases.
- Upgraded to Karaf 4.2. You may run Camel on Karaf 4.1 but we only officially support Karaf 4.2 in this release.
- Optimized using toD DSL to reuse endpoints and producers for components where it is possible. For example, HTTP based components will now reuse producer (HTTP clients) with dynamic URIs sending to the same host.
- The File2 consumer with read-lock idempotent/idempotent-changed can now be configured to delay the release tasks to expand the window when a file is regarded as in-process, which is usable in active/active cluster settings with a shared idempotent repository to ensure other nodes don’t too quickly see a processed file as a file they can process (only needed if you have readLockRemoveOnCommit=true).
- Allow to plugin a custom request/reply correlation id manager implementation on Netty4 producer in request/reply mode. The Twitter component now uses extended mode by default to support tweets greater than 140 characters
- Rest DSL producer now supports being configured in REST configuration by using endpointProperties.

- The Kafka component now supports HeaderFilterStrategy to plugin custom implementations for controlling header mappings between Camel and Kafka messages.
- REST DSL now supports client request validation to validate that Content-Type/Accept headers are possible for the REST service.
- Camel now has a Service Registry SPI which allows you to register routes to a service registry (such as consul, etcd, or zookeeper) by using a Camel implementation or Spring Cloud.
- The SEDA component now has a default queue size of 1000 instead of unlimited.
- The following noteworthy issues have been fixed:
  - Fixed a CXF continuation timeout issue with camel-cxf consumer that could cause the consumer to return a response with data instead of triggering a timeout to the calling SOAP client.
  - Fixed camel-cxf consumer doesn't release UoW when using a robust one-way operation.
  - Fixed using AdviceWith and using weave methods on onException etc. not working.
  - Fixed Splitter in parallel processing and streaming mode may block, while iterating message body when the iterator throws an exception in the first invoked next() method call.
  - Fixed Kafka consumer to not auto commit if autoCommitEnable=false.
  - Fixed file consumer was using markerFile as read-lock by default, which should have been none.
  - Fixed using manual commit with Kafka to provide the current record offset and not the previous (and -1 for first).
  - Fixed Content Based Router in Java DSL may not resolve property placeholders in when predicates.

### Camel 2.23 updates

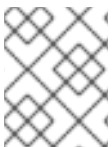
- Upgraded to Spring Boot 2.1.
- Additional component-level options can now be configured by using spring-boot auto-configuration. These options are included in the spring-boot component metadata JSON file descriptor for tooling assistance.
- Added a documentation section that includes all the Spring Boot auto configuration options for all the components, data-formats, and languages.
- All the Camel Spring Boot starter JARs now include **META-INF/spring-autoconfigure-metadata.properties** file in their JARs to optimize Spring Boot auto-configuration.
- The Throttler now supports correlation groups based on dynamic expression so that you can group messages into different throttled sets.
- The Hystrix EIP now allows inheritance for Camel's error handler so that you can retry the entire Hystrix EIP block again if you have enabled error handling with redeliveries.
- SQL and EISql consumers now support dynamic query parameters in route form. Note that this feature is limited to calling beans by using simple expressions.

- The swagger-restdsl maven plugin now supports generating DTO model classes from the Swagger specification file.
- The following noteworthy issues have been fixed:
  - The Aggregator2 has been fixed to not propagate control headers for forcing completion of all groups, so it will not happen again if another aggregator EIP is in use later during routing.
  - Fixed Tracer not working if redelivery was activated in the error handler.
  - The built-in type converter for XML Documents may output parsing errors to stdout, which has now been fixed to output by using the logging API.
  - Fixed SFTP writing files by using the charset option would not work if the message body was streaming-based.
  - Fixed Zipkin root id to not be reused when routing over multiple routes to group them together into a single parent span.
  - Fixed optimized toD when using HTTP endpoints had a bug when the hostname contains an IP address with digits.
  - Fixed issue with RabbitMQ with request/reply over temporary queues and using manual acknowledge mode. It would not acknowledge the temporary queue (which is needed to make request/reply possible).
  - Fixed various HTTP consumer components that may not return all allowed HTTP verbs in Allow header for OPTIONS requests (such as when using rest-dsl).
  - Fixed the thread-safety issue with FluentProducerTemplate.

## 5.2. ABOUT MAVEN DEPENDENCIES

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.

There is a dedicated BOM file for each container in which Fuse runs.

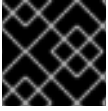


### NOTE

You can find these BOM files here: <https://github.com/jboss-fuse/redhat-fuse>.  
Alternatively, go to the [latest Release Notes](#) for information on BOM file updates.

The Fuse BOM 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.xml** file.
- 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.

### 5.3. UPDATING YOUR FUSE PROJECT'S MAVEN DEPENDENCIES

To upgrade your Fuse application for Karaf, update your project's Maven dependencies.

#### Procedure

1. Open your project's **pom.xml** file.
2. Add a **dependencyManagement** element in your project's **pom.xml** file (or, possibly, in a parent **pom.xml** 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>7.9.0.fuse-sb2-790065-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>
```

3. Save your **pom.xml** file.

After you specify the BOM as a dependency in your **pom.xml** file, it becomes possible to add Maven dependencies to your **pom.xml** file *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.xml** file:

```
<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.

## CHAPTER 6. UPGRADING FUSE STANDALONE ON KARAF

The Fuse on Apache Karaf upgrade mechanism enables you to 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 *same file* that you use to install Fuse on Apache Karaf.



### NOTE

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-7.9.0.fuse-790071-redhat-00001.zip**).

- [Section 6.1, “Impact of upgrading Fuse on Karaf”](#)
- [Section 6.2, “Upgrading Fuse Standalone on Karaf”](#)
- [Section 6.3, “Rolling back an upgrade for Fuse on Karaf”](#)

### 6.1. IMPACT OF UPGRADING FUSE ON KARAF

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.

### 6.2. UPGRADING FUSE STANDALONE ON KARAF



The following instructions guide you through upgrading Fuse on Apache Karaf. Ensure all prerequisites are completed before commencing the upgrade procedure.

## Prerequisites

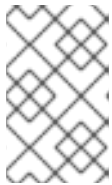
- Ensure you have a full backup of your Fuse on Apache Karaf installation before upgrading.
- Start the container, if it is not already running.

## TIP

If the container is running in the background (or remotely), connect to the container using the SSH console client, **bin/client**.

- Add the upgrade installer file to the container's environment by invoking the **patch:add** command. For example, to add the **fuse-karaf-7.9.0.fuse-790071-redhat-00001.zip** upgrade installer file:

```
patch:add file:///path/to/fuse-karaf-7.9.0.fuse-790071-redhat-00001.zip
```



## NOTE

The **patch:find** command can only be used to find and add the latest hot fix patches to the container's environment; it cannot be used to apply full upgrade patches.

## Procedure

1. 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
```

2. 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
```

3. 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.

- 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-7.9.0.fuse-790071-redhat-00001
```

- 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, 790071, as follows:

```

karaf@root(>) bundle:list -l | grep 790071
 22 | Active | 80 | 7.9.0.fuse-790071-redhat-00001 |
mvn:org.jboss.fuse.modules/fuse-pax-transx-tm-narayana/7.9.0.fuse-790071-redhat-00001
188 | Active | 80 | 7.9.0.fuse-790071-redhat-00001 |
mvn:org.jboss.fuse.modules.patch/patch-commands/7.9.0.fuse-790071-redhat-00001

```



#### NOTE

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

## 6.3. ROLLING BACK AN UPGRADE FOR FUSE ON KARAF

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. This set of instructions guides you through this procedure.

## Prerequisites

- You have recently upgraded Fuse on Karaf.
- You want to rollback the upgrade.

## Procedure

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
```



### NOTE

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.