Red Hat Fuse 7.1

Migration Guide

Migrating to Red Hat Fuse 7.1

Last Updated: 2019-01-02

Red Hat Fuse 7.1 Migration Guide

Migrating to Red Hat Fuse 7.1

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 7.0

1.1. MIGRATION PATH FOR FUSE 7.0 ON KARAF

There is no automated migration path for Fuse 7.0. A new installation must be performed, with configuration and other modified files copied across manually. Applications will need to be recompiled to align with the new versions provided. Use the Maven Bill of Materials (BOM) file to migrate Maven dependencies to the new versions and see also Component Details.

1.2. MIGRATION PATH FOR FUSE 7.0 ON EAP

There is no automated migration path to Fuse 7.0 on EAP from previous version of Fuse on EAP. To migrate to Fuse 7.0 you will need to make a new installation of Fuse 7.0 on JBoss EAP. After a successful installation, any existing deployments will need to be re-deployed to the new system. For installation information please see Installation on JBoss EAP and for deployment information see Deployment in the Management Console.

1.3. DEPRECATED AND REMOVED FEATURES

For the list of features that have been deprecated or removed in Fuse 7.0, see Release Notes.

CHAPTER 2. APACHE ACTIVEMQ MIGRATION

In Fuse 7.0, the Apache ActiveMQ is no longer provided as an embedded broker in Apache Karaf. Instead of embedding the broker, Fuse 7.0 provides a variety of messaging clients which you can use to connect to an *external broker* (such as Red Hat AMQ 7 or JBoss A-MQ 6.3).

For more details, see Deploying into Apache Karaf.

CHAPTER 3. UPGRADING FUSE ON APACHE KARAF

3.1. 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.

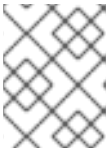
The upgrade installer file is the very same file that you would use to make a fresh installation of Fuse on Apache Karaf. To obtain the upgrade installer file, go to the **Downloads** page of the Red Hat customer portal and download the latest version of the installation archive for Fuse on Apache Karaf (for example, **`fuse-karaf-7.1.0.fuse-710023-redhat-00001.zip`**).

3.2. UPGRADE PROCEDURES FOR FUSE ON KARAF

3.2.1. 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.

3.2.2. 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-7.1.0.fuse-710023-redhat-00001.zip`** upgrade installer file:

```
patch:add file:///path/to/fuse-karaf-7.1.0.fuse-710023-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: 7.0.0.fuse-000191-redhat-1
New patch mechanism version detected: 7.1.0.fuse-710023-redhat-00001
Uninstalling patch features in version 7.0.0.fuse-000191-redhat-1
Installing patch features in version 7.1.0.fuse-710023-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-7.1.0.fuse-710023-redhat-00001 false       true
fuse-karaf-7.1.0.fuse-710023-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-7.1.0.fuse-710023-redhat-
00001
INFO : org.jboss.fuse.modules.patch.patch-management (226):
Installing rollup patch "fuse-karaf-7.1.0.fuse-710023-redhat-00001"
========== Repositories to remove (10):
 - mvn:io.hawt/hawtio-karaf/2.0.0.fuse-000172-redhat-1/xml/features
...
========== Repositories to add (10):
 - mvn:io.hawt/hawtio-karaf/2.0.0.fuse-710018-redhat-
00002/xml/features
...
========== Repositories to keep (7):
 - 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-000237-redhat-1
4.2.0.fuse-710024-redhat-00002
...
========== Bundles to update as part of features or core bundles
(113):
[symbolic name]
```

```
[version]                          [new location]
io.hawt.hawtio-log
2.0.0.fuse-000172-redhat-1         mvn:io.hawt/hawtio-log/2.0.0.fuse-
710018-redhat-00002
...
========== Bundles to reinstall as part of features or core bundles
(110):
[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.
```

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-7.1.0.fuse-710023-redhat-
00001
```

8. Validate the upgrade, by searching for one of the upgrade artifacts. For example, if you had just upgraded Fuse 7.0.0 to Fuse 7.1.0, you could search for bundles with the build number, 710023, as follows:

```
karaf@root()> bundle:list -l | grep 710023
 22 | Active |  80 | 7.1.0.fuse-710023-redhat-00001  |
mvn:org.jboss.fuse.modules/fuse-pax-transx-tm-narayana/7.1.0.fuse-
710023-redhat-00001
188 | Active |  80 | 7.1.0.fuse-710023-redhat-00001  |
mvn:org.jboss.fuse.modules.patch/patch-commands/7.1.0.fuse-710023-
redhat-00001
```

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

3.2.3. 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 4. EAP 7.X MIGRATION

This section covers the changes in the EAP 7.x related to Messaging, WildFly Management Port, CXF consumers, and other components that are used in Fuse 7.0 .

4.1. MESSAGING

The messaging subsystem on EAP 7.x uses Artemis instead of HornetQ. You need to migrate custom EAP CLI scripts that reference the old EAP 6.x messaging subsystem to Artemis. See, EAP migration guide [https://access.redhat.com/documentation/en-us/red_hat_jboss_enterprise_application_platform/7.1/html-single/migration_guide/index] for details.

4.2. WILDFLY MANAGEMENT PORT

The WildFly management port is changed to 9990. The old port number 9999 is no longer in use. For configurations that use the **`wildly-maven-plugin`** in the pom files, you must remove references to port 9999 as the plugin defaults to 9990.

4.3. COMPONENT `CAMEL-RESTLET`

The **`camel-restlet`** component has been removed from Fuse on EAP. The **`camel-restlet`** producers are supported, but the consumers working on the old EAP JBoss Web stack never worked. Considering that we support a number of alternative HTTP components, the **`camel-restlet`** component was removed from Fuse 7.

You should switch to an alternate HTTP consumer component such as **`undertow`**, **`http4`**, **`netty-http4`**, and so on.

4.4. WORKAROUNDS FOR CXF CONSUMERS

The **`camel-cxf`** consumers are supported in Fuse 7.x. You can migrate to 'skinny' WAR deployments instead of deploying 'fat' camel WAR deployments or other workarounds for using CXF consumers in Fuse EAP 6.x.

4.5. MAVEN POM VERSION UPDATES

You need to update the Maven POMs to reference to the latest BOM & Fuse and EAP artifact versions.

CHAPTER 5. MIGRATE FABRIC PROFILES

This section covers the migration of Fabric8 1.x profiles manually in Fuse 7.1.

5.1. OVERVIEW

- Fabric8 V1 monolithic application deployments may need to be migrated to micro-service applications or migrated to a monolith container in OpenShift that exposes several services (not optimal).
- Re-factor network of Broker architectures to JBoss AMQ-7. The re-factoring affects how user applications connect to Broker and are deployed in OpenShift.
- Containers can be mapped one to one from Fabric8 V1 deployments if the Fabric8 V1 architecture was a micro services style deployment. Container meta data such as host name, port, and so on need to be mapped to OpenShift resources and concepts such as Nodes, Pods and Services.
- Features and bundles that were only available in Fabric8 V1 need to be mapped to either OpenShift resources/features or to an alternative solution.

5.2. HIGH LEVEL CONCERNS

- Fabric8 V1 deployments may be monoliths that are connected using Fabric, or they may be a large number of small Fabric containers running ActiveMQ Brokers and, or Camel routes.
- Monolith deployments will have to be refactored into several services under OpenShift.
- Applications developed to run in Karaf could potentially be affected in the migration from version 2 to version 4. You have a choice between redeploying existing applications to a Karaf image on OpenShift or (optionally) refactoring applications to run in a Spring Boot container instead.
- Use OpenShift's **EFK** (ElasticSearch+Fluentd+Kibana) stack instead of Fabric8 V1 Insight for log monitoring.
- Use OpenShift application services and routes instead of Fabric8 V1 Gateway.
- Monitoring in OpenShift is supported using Fuse 7 HawtIO console and Prometheus monitoring service. Users have to configure and deploy their own Prometheus servers as the server requirements are unique to the applications being monitored.

5.3. IMPLEMENTATION DETAILS

- Fabric8 V1 containers refactored to run small services will map to Fuse on OpenShift based projects packaged as OpenShift pod based services.
- Fuse on OpenShift projects could be deployed using S2I templates to build the source in OpenShift builder pods, or built externally using Fabric8 maven plugin and deployed using S2I binary deployment. External builds may be more efficient and can be performed using external CI/CD infrastructure such as Jenkins.
- Fabric8 V1 container versions can be migrated to ImageStream tags for versioning in OpenShift.
- OpenShift Deployment Configuration supports liveness probes and scaling of services.

- A new **`fabric8-karaf-cm`** feature bridges OpenShift **`ConfigMaps`** and Karaf **`ConfigAdmin`** service to provide dynamic configuration updates in OpenShift Karaf applications. See Fuse on OpenShift Guide for more details.

CHAPTER 6. 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.

6.1. 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>7.1.0.fuse-710019-redhat-00002</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:

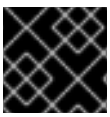
```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-velocity</artifactId>
</dependency>
```

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

fuseversion = 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>7.1.0.fuse-710019-redhat-00002</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>
```

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>
  <scope>provided</scope>
</dependency>
```

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

6.2. 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

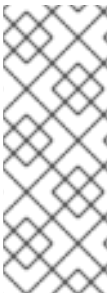
Only the set of dependencies defined by a Fuse BOM are supported by Red Hat.

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>7.1.0.fuse-710019-redhat-00002</fuse.version>
    <spring-boot.version>1.5.13.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>
```



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-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.

CHAPTER 7. CAMEL MIGRATION ISSUES

7.1. CAMEL 2.21 MIGRATION ISSUES

Fuse 7.0 uses Camel 2.21. This section covers the changes in Camel 2.21 that are to be considered before upgrading to Fuse 7.0.

The changes to Camel 2.21 that must be considered before upgrading:

- Jetty has been upgraded to version 9.4 by default and **`camel-jetty`** needs version 9.3 or 9.4 to run in OSGi.
- The component **`camel-saxon`** is used to create the **`SaxonXpathFactory`** class is from Saxon. In absence of **`camel-saxon`** the factory method is created as per the old way.
- The **`camel-json-validator`** component uses the **`NetworkNT JSon`** Schema validator library instead of **`Everit`**. **`Everit`** had ASF license implications and will be removed from future Camel releases. The **`NetworkNT`** supports v4 draft of **`JSon`** Schema for validation so update your schemas to use the draft version.
- The **`FileIdempotentRepository`** is updated to use the internal in-memory cache for quick lookup of the most frequent file names, and for lookup from disk. See the class javadoc of the file for more details.
- The Karaf commands for routes are changed so the arguments for the camel context is placed first, and the route id is the second argument. This allows the route completer to use the selected camel context name to only show route ids from that camel context else it shows all the routes for every Camel application running in Karaf.
- The **`camel-spring-boot`** actuator endpoints for routes are now in read-only mode by default. The operations to **`start`**, **`stop`**, **`suspend, `resume routes`** is forbidden. You can turn off read-only mode by setting the spring boot configuration **`endpoints.camelroutes.read-only = false`**.

7.2. CAMEL 2.20 MIGRATION ISSUES

This section covers the changes in Camel 2.20 that are to be considered before upgrading to Fuse 7.0.

The changes to Camel 2.20 that must be considered before upgrading:

- The Maven version 3.3.3 or higher is required to build the project.
- The **`camel-dropbox`** is upgraded to v2 api. There can be backward compatibility issues becuase of the V2 upgrade.
- In the **`camel-infinispan`** the result is not set in the **`CamelInfinispanOperationResult`** header but in the in body. To change this behavior you can set the header **`CamelInfinispanOperationResultHeader`** with the name of the header that contains the result or with the **`resultHeader`** URI option.
- The **`camel-infinispan`** URI option command has been deprecated and replaced by operation for consistency purposes.

- In **camel-infinispan** commands are changed to use the short form such as PUT, GET. The old operation names **CamelInfinispanOperationPut** and **CamelInfinispanOperationGet** have been deprecated.
- In **camel-undertow** the **matchOnUriPrefix** option, the default value is set to FALSE to make it consistent with other components such as, Camel HTTP components.
- The Twitter components are split into four types, **directmessage**, **search**, **streaming** and **timeline** and has its own endpoint and scheme.
- The **RuntimeEndpointRegistry** is no longer in extended mode by default. To use extended mode, set the management statistics level to **Extended** explicitly.
- There is no **RuntimeEndpointRegistry** in use by default. You need to explicitly configure a registry to be used, or turn it on using the management agent, or set the statistics level to extended mode.
- Camel with Spring XML routes do not register endpoints in the Spring registry from Camel routes where <from> or <to> have endpoints assigned with an explicit id attribute. The option **registerEndpointIdsFromRoute** can be set to true on **<camelContext>** for backward compatibility. But this registration is deprecated and instead you should use **<endpoint>** to register Camel endpoints with id's in Spring registry.
- The **camel-spring-dm** has been removed. For XML DSL with OSGi use **camel-blueprint**.
- Copying streams in IOHelper from **came-core** now regard EOL of data if the first read byte is zero. This change is a work around for issues on application servers such as IBM WebSphere. The setting can be turned off by configuring JVM system property **"camel.zeroByteEOLEnabled=false"**.
- The **camel-jms** component is based on the JMS 2.0 API (geronimo-jms_2.0_spec) instead of JMS 1.1 API (geronimo-jms_1.1_spec). But **camel-jms** works at runtime with both JMS 1.1 or 2.0.
- The **camel-kura** is upgraded to newer OSGi API version.
- The **camel-stomp** uses the destination without replacing all slash characters with colon.
- The **camel-ignite** is updated to use Ignite version 2.2.x .
- The **camel-dozer** has been upgraded from Dozer v5 to v6 which requires migration. See, Dozer migration guides https://dozermapper.github.io/gitbook/migration/v5-to-v6.html and https://dozermapper.github.io/gitbook/migration/v6-to-v61.html

7.3. CAMEL 2.19 MIGRATION ISSUES

There are a number of changes in Camel 2.19 that have to be considered before upgrading to Fuse 7.0.

There are known issues that can break the API.

- The groovy DSL from **camel-groovy** has been moved to **camel-groovy-dsl** module. The camel-groovy contains only the Camel Groovy Language.
- The **Camel-spring-LDAP** uses **java.util.function.BiFunction<L, Q, S>** instead of **org.apache.camel.component.springldap.LdapOperationsFunction<Q, S>**.

- The deprecated APIs from **camel-spring-boot** has been removed to upgrade and support Spring Boot 1.5.x .
- The **camel-mongodb-gridf** schema is renamed to **mongodb-gridfs**.
- The **commands-core** Catalog commands have been removed.
- The **org.apache.camel.spring.boot.FatJarRouter** is removed so you use the regular **RouteBuilder** classes in Spring Boot applications.
- The Kafka endpoint option **seekToBeginning=true** should be migrated to **seekTo=beginning**.
- The Kafka endpoint option **bridgeEndpoint** has moved from endpoint to the **KafkaConfiguration** class.
- The Kafka component is now easier to configure and use. There is a backwards incompatible change so users need to migrate. The kafka URI is changed from **kafka:brokers** to **kafka:topic**. So you need to specify the topic name in the **context-path** and the brokers as parameters, for example, the old syntax was **kafka:myserver?topic=sometopic** which is changed to **kafka:sometopic?brokers=myserver**.
- The Infinispan URI syntax has changed from **infinispan:hostName?options** to **infinispan:cacheName?options**.

There are changes to Camel 2.19 that must be considered before upgrading:

- The **camel-spring-dm** has been disabled from the Karaf features file so users cannot install it out of the box, it is also deprecated and users are encouraged to use OSGi Blueprint instead. The JAR is still shipped and can be installed manually but it there is no support avaiable. The JAR will be removed completed in a future release.
- The **Groovy DSL** and **Scala DSL** is deprecated and will be moved to **Camel Extra** and not distributed out of the box in the future.
- Camel now uses Karaf 4.x API and therefore not possible to run on older Karaf versions.
- The **camel-blueprint** changed startup behavior to start on **Blueprint.CREATED** event which is more appropriate way of startup instead of **Blueprint.REGISTERED** as was used previously.
- The **camel-spring-boot** does not include prototype scoped beans when auto scanning for RouteBuilder instances, which is how **camel-spring** works. You can revert back using the **includeNonSingletons** option.
- The **camel-spring-javaconfig** removed from Karaf features as it was not supported in OSGi/Karaf.
- The **camel spring-boot** shell commands have been removed as **spring-boot** shell has been deprecated in **spring-boot**.
- The **camel-box** has been migrated to use box v2 api so there may be some migration needed as the old **camel-box** component was using box v1 api.

- The **JSon** schema from **camel-catalog** have changed to use boolean, integer and numeric values when applicable instead of using string values.
- The **camel-catalog** Karaf commands has been removed.

CHAPTER 8. APACHE CXF ISSUES

8.1. APACHE CXF 3.1 MIGRATION

Fuse 7.0 uses Apache CXF 3.1. This introduces some issues that you sould be aware of before migrating.

8.1.1. Main Changes

- The JAX-WS/Simple frontend ServerFactoryBean will automatically call reset at the end of the **`create()`** call. This allows resources to be cleaned up and garbage collected sooner. However, it also prevents multiple calls to **`create()`** from sharing the same ServerInfo/EndpointInfo objects, as they would in older versions. That sharing has caused many problems in the past due to sharing of properties, such as token caches, that are stored on those objects. The new behavior is more correct, but it is different from previous versions so care must be taken when upgrading.

- The Karaf **`features.xml`** file for CXF 3.1 will no longer install spring or spring-dm when installing the **`cxf`** feature. If you require spring/spring-dm, you will need to install those features prior to installing the CXF feature.

8.1.2. Security changes

- The STS (Security Token Service) now issues tokens using the RSA-SHA256 signature algorithm by default, and the SHA-256 digest algorithm . Previously it used RSA-SHA1 and SHA-1 respectively.

- Some security configuration tags have been renamed from **`ws-security.*`** to **`security.\*`**, as they are now shared with some of the JAX-RS stack. The old tags will continue to work as before however without any change. See the Security Configuration page for more information.

- The SAML/XACML functionality previously available in the cxf-rt-security module is now in the cxf-rt-security-saml module. If you are explicitly specifying the SAML version in a SAML CallbackHandler, then this is changed in CXF 3.1 due to the migration to use OpenSAML 3.1. The version is now set on the SAMLCallback using a org.apache.wss4j.common.saml.bean.Version class. Previously there was a dependency on OpenSAML's SAMLVersion class.

- It is now possible to plug in custom **`WS-SecurityPolicy`** validators if you wish to change the default validation logic for a particular policy.

8.1.3. New Features

- The CXF JAX-WS code generator has a new option, **`seiSuper`**, that can be used to specify additional super interfaces for the SEI. This makes the code nonportable to other JAX-WS containers. The primary use would be to add AutoCloseable to the interface to allow use of the clients in Java7 try with resource blocks.

- New Metrics feature for collecting metrics about a CXF services. Codahale/DropWizard based collector included.

- New Throttling feature for easily throttling CXF services. Sample included that uses the Metrics component to help make the throttling decisions.

- New Logging feature for more advanced logging than the logging available in cxf-core
- New Metadata service for SAML SSO to allow you to publish SAML SSO metadata for your service provider.
- The **`cxf`** frontend to the JAX-WS code generator, **`-fe cxf`** now generates code that is more Java7-friendly as the return type of the **`getPort(…)`** calls is a sub-interface of the SEI that also implements AutoCloseable, BindingProvider, and Client. Code that used to look like:

```
(AddNumbersPortType port = service.getAddNumbersPort();
    ((BindingProvider)port).getRequestContext()
            .put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
address);
    port.addNumbers3(-1, 2);
    ((Closeable)port).close();
```

can be replaced with:

```
try (AddNumbersPortTypeProxy port = service.getAddNumbersPort()) {

port.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPER
TY, address);
        port.addNumbers3(-1, 2);
    }
```

8.1.4. Major Dependency Changes

- The Jetty based HTTP transport has been updated to support Jetty 9 as well as Jetty 8. However, support for Jetty 7 has been dropped.
- Due to the Jetty upgrade, support for running Jetty based endpoints in Karaf 2.3.x has been dropped.
- Support for using JAX-WS 2.1 based API jars has been removed. Java 7 (now required) includes JAX-WS 2.2 so this should not be an issue.
- WSS4J 2.1 is included, which in turn includes OpenSAML 3.0.