Red Hat JBoss Fuse 6.3 Migration Guide

Migrating to Red Hat JBoss Fuse 6.3

JBoss A-MQ Docs
Team

Red Hat JBoss Fuse 6.3 Migration Guide

Migrating to Red Hat JBoss Fuse 6.3

JBoss A-MQ Docs Team
Content Services
fuse-docs-support@redhat.com

Legal Notice

Abstract

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

Table of Contents

CHAPTER 1. WHAT'S NEW

Abstract

This section describes the main features and changes in version 6.3.

1.1. NEW FEATURES

The main new features in version 6.3 are:

- JBDSIS tooling is now available in a standalone installer package, which combines the base JBDS 9.1.0.GA development environment with the JBDSIS 9.0.2.GA plug-ins for JBoss Fuse development. See chapter "Install Red Hat JBoss Development Tools" in "Installation on Apache Karaf" for details.
- JBDSIS tooling has undergone a major revision for this release. It is now based on JBDS 9.1 and the look and feel of the Camel route editor has been completely revised.
- The Camel CDI component has been rewritten to provide better integration with the JSR 299: Contexts and Dependency Injection (CDI) programming model. See chapter "Camel CDI" in "Deploying into Apache Karaf" for more details.
- New Camel Braintree component (**`camel-braintree`**) provides integration with various payment systems, including PayPal. See chapter "Braintree" in "Apache Camel Component Reference" for more details.
- New Camel ServiceNow component (**`camel-servicenow`**) provides integration with the ServiceNow REST API. See chapter "ServiceNow" in "Apache Camel Component Reference" for more details.

1.2. TECHNOLOGY PREVIEW FEATURES

The following features are provided as technical previews only in version 6.3, and are not suitable for production environments. For details on what *technical preview* means, see https://access.redhat.com/support/offerings/techpreview/.

- RH-SSO adapter for JBoss Fuse. See the section JBoss Fuse Adapter from the RH-SSO 7.0 *Securing Applications and Services Guide* for more details.

1.3. IMPORTANT NOTES

New Maven repository

Since JBoss Fuse 6.3, the JBoss Fuse Maven artifacts are available *only* from the following Maven repositories:

- https://maven.repository.redhat.com/ga
- https://maven.repository.redhat.com/earlyaccess/all

Hence for JBoss Fuse 6.3, you need to edit your Maven **`settings.xml`** file, replacing the old **`repo.fusesource.com`** repository URLs (at https://repo.fusesource.com/nexus/content/repositories/releases and https://repo.fusesource.com/nexus/content/groups/ea) with the new Maven repository URLs.

Note

Older versions of JBoss Fuse (prior to 6.3) continue to use the old Maven repositories.

ActiveMQ runtime can now be wired to JMS 2.0 API bundle in OSGi

In JBoss Fuse 6.3, the ActiveMQ runtime has been modified so that it is compatible with and can be wired to the JMS 2.0 API bundle. This does *not* imply that ActiveMQ supports JMS 2.0. In fact, ActiveMQ still supports JMS 1.1 only. This change does mean, however, that it is now possible to deploy an ActiveMQ broker (which is a JMS 1.1 application) alongside a JMS 2.0 compliant application in the same OSGi container. This can be useful, for example, if you want to deploy two different messaging products in the same Apache Karaf container.

Upgraded Jetty from 8.1.x to 9

In the Apache Karaf container, Jetty (which provides the default HTTP servlet container for Karaf) has been upgraded from Jetty 8.1.x to Jetty 9. This has a significant impact on the Jetty container configuration, affecting settings in the **`etc/org.ops4j.pax.web.cfg`** file, in the **`etc/jetty.xml`** file, and in the Camel Jetty endpoint. For more details, see chapter "Securing the Jetty HTTP Server" in "Security Guide" and chapter "Securing the Camel Jetty Component" in "Security Guide".

Apache Karaf package name changed from jboss-fuse-full to jboss-fuse-karaf

The package name for the Apache Karaf distribution of JBoss Fuse has changed from **`jboss-fuse-full-`*`ProductVersion`*`.zip`** to **`jboss-fuse-karaf-`*`ProductVersion`*`.zip`** in this release.

CXF security changes

Note the following important changes to CXF security in this release:

- The STS (Security Token Service) now issues tokens using the RSA-SHA256 signature algorithm by default (previously RSA-SHA1), and the SHA-256 digest algorithm (previously SHA-1).
- The SAML/XACML functionality previously available in the **`cxf-rt-security`** module is now in the **`cxf-rt-security-saml`** module.

New interceptor required for transactional RFC SAP endpoints

A new interceptor object is provided in JBoss Fuse 6.3, which is needed to configure transactional RFC destinations properly for the Camel SAP component. For details, see section "Interceptor for tRFC and qRFC destinations" in "Apache Camel Component Reference".

Make Quickstart Examples Available

In previous releases, profiles for quickstart example were available by default. The new default behavior is that profiles for quickstart examples are not available in a new fabric. To create a fabric in which you can run the quickstart examples, edit the **$FUSE_HOME/fabric/io.fabric8.import.profiles.properties** file by uncommenting the line that starts with the following:

```
# importProfileURLs =
```

If you create a fabric without doing this and you want to run the quickstart examples, follow these steps to make them available:

1. Edit the **$FUSE_HOME/quickstarts/pom.xml** file to add a fabric I/O plugin, for example:

   ```
   <plugin>
        <groupId>io.fabric8</groupId>
        <artifactId>fabric8-maven-plugin</artifactId>
        <version>1.2.0.redhat-630187</version>
   </plugin>
   ```

2. In the **$FUSE_HOME/quickstarts** directory, change to the directory for the quickstart example you want to run, for example:

   ```
   cd beginner
   ```

3. In that directory, execute the following command:

   ```
   mvn fabric8:deploy
   ```

   You would need to run this command in each directory that contains a quickstart example that you want to run.

CHAPTER 2. DEPRECATED AND REMOVED FEATURES

SUPPORT FOR FABRIC8 1.X WILL BE DROPPED IN THE NEXT MAJOR RELEASE OF JBOSS FUSE

In the next major release of JBoss Fuse (planned as JBoss Fuse 7.0), Fabric8 version 1 will be dropped, to be replaced by Fabric8 version 2. The Fabric8 version 2 upgrade is a new generation of distributed container technology that is entirely cloud-based and leverages the OpenShift technology stack to provide the foundation for containerized JBoss Fuse applications.

BIN/DELETEFABRIC8 SCRIPT HAS BEEN REMOVED

The **`bin/deletefabric8`** script has been removed in this release.

CAMEL COMPONENTS FOR GOOGLE APP ENGINE ARE DEPRECATED

The GAE components for Apache Camel are deprecated in JBoss Fuse 6.3 and will be removed from a future release of JBoss Fuse.

CAMEL NETTY COMPONENT IS DEPRECATED

The Camel Netty component and the SwitchYard Camel Netty component are both deprecated in JBoss Fuse 6.3 and will be removed in a future release of JBoss Fuse. It is recommended that you use the Camel Netty4 component instead.

CAMEL JBPM COMPONENT IS DEPRECATED

The Camel jBPM component (**`camel-jbpm`**) is deprecated in JBoss Fuse 6.3 and will be removed in a future release of JBoss Fuse.

CAMEL LEVELDB COMPONENT IS DEPRECATED ON ALL OPERATING SYSTEMS EXCEPT FOR LINUX

Since JBoss Fuse 6.3, the Camel LevelDB (**`camel-leveldb`**) component is deprecated on all operating systems except for Red Hat Enterprise Linux. In future, the Camel LevelDB component will be supported only on Red Hat Enterprise Linux.

TANUKI BASED WRAPPER FOR INSTALLING JBOSS FUSE AS A SERVICE IS DEPRECATED

The Tanuki based wrapper scripts—generated using the **`wrapper:install`** Karaf console command—for installing JBoss Fuse as a service are deprecated since JBoss Fuse 6.3 and will be removed in a future release of JBoss Fuse. To install the Apache Karaf container as a service, it is

recommended that you use the new **`karaf-service-*.sh`** scripts from the **`bin/contrib`** directory instead.

SMOOKS IS DEPRECATED

Since JBoss Fuse 6.3, the Smooks component for SwitchYard is deprecated and will be removed in a future release of JBoss Fuse.

BPEL IS DEPRECATED

BPEL (based on the Riftsaw project) is no longer being actively developed and will be removed from a future release of JBoss Fuse. If you are currently using BPEL, it is recommended that you consider migrating to the Red Hat JBoss BPM Suite (which is supported through the JBoss Fuse Integration Package).

DESIGN TIME GOVERNANCE IS DEPRECATED

The Design Time Governance component is deprecated since 6.2.1 and not included with the product.

RUNTIME GOVERNANCE IS DEPRECATED.

Since JBoss Fuse 6.3, the Runtime Governance (RTGov) component is deprecated and will be removed in a future release of JBoss Fuse.

S-RAMP IS DEPRECATED

The SOA Repository Artifact Model and Protocol (S-RAMP) component is deprecated since 6.2.1 and will be removed from a future release.

BIN/PATCH SCRIPT IS DEPRECATED

The **`bin/patch`** script (**`bin\patch.bat`** on Windows O/S) is deprecated and will be removed in a future release.

SPRING DYNAMIC MODULES (SPRING-DM) IS DEPRECATED

Spring-DM (which integrates Spring XML with the OSGi service layer) is deprecated since 6.2.1 and you should use the Blueprint framework instead. Using Blueprint does not prevent you from using the Java libraries from the Spring framework: the latest version of Spring is compatible with Blueprint.

APACHE OPENJPA IS DEPRECATED

The Apache OpenJPA implementation of the Java Persistence API (JPA) is deprecated since 6.2.1. It is recommended that you use the Hibernate implementation instead.

CHAPTER 3. MIGRATION PATHS FOR JBOSS FUSE 6.3

MIGRATION PATH FOR JBOSS FUSE 6.3 ON KARAF

There is no automated migration path for JBoss Fuse 6.3. A new installation must be performed, with configuration and other modified files copied across manually.

MIGRATION PATH FOR JBOSS FUSE 6.3 ON EAP

There is no automated migration path to JBoss Fuse 6.3 on EAP from previous version of JBoss Fuse on EAP. To migrate to JBoss Fuse 6.3 you will need to make a new installation of JBoss Fuse 6.3 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.

CHAPTER 4. MIGRATE MAVEN PROJECTS

OVERVIEW

JBoss Fuse has a JBoss Fuse BOM (Bill of Materials), which defines the versions of all the JBoss Fuse Maven artifacts. You can use the BOM to simplify migration of your Maven POM files. Instead of updating the **`version`** elements on each Maven dependency, all you need to do is to import the latest JBoss Fuse BOM, which defines default versions for all of the dependencies provided by JBoss Fuse.

JBOSS FUSE BOM

The JBoss Fuse BOM is a parent POM that defines the versions for all of the Maven artifacts provided by JBoss Fuse. The JBoss Fuse BOM exploits Maven's *dependency management* mechanism to specify default versions for the Maven artifacts, so that it is no longer necessary to specify the artifact versions explicitly in your POM.

CURRENT VERSION OF THE JBOSS FUSE BOM

The easiest way to discover the current version of the JBoss Fuse BOM is to examine one of the sample **`pom.xml`** files from the **`quickstarts`** examples. For example, in the ***`InstallDir`*`/quickstarts/eip/pom.xml`** file, you can find a line that defines the JBoss Fuse BOM version, as follows:

```
<project ...>
    ...
    <properties>
        ...
        <!-- the version of the JBoss Fuse BOM, defining all the
dependency versions -->
        <jboss.fuse.bom.version>6.3.0.redhat-187</jboss.fuse.bom.version>
    </properties>
    ...
</project>
```

HOW TO MIGRATE MAVEN DEPENDENCIES USING THE JBOSS FUSE BOM

To migrate Maven dependencies using the JBoss Fuse BOM, open the Maven **`pom.xml`** file for your project and edit it as follows:

1. Define the JBoss Fuse BOM version as a property in your **`pom.xml`** file, using the current BOM version. For example:

   ```
   <project ...>
       ...
       <properties>
           ...
   ```

```
        <jboss.fuse.bom.version>6.3.0.redhat-
187</jboss.fuse.bom.version>
    </properties>
    ...
</project>
```

2. Reference the JBoss Fuse BOM parent POM in a **dependencyManagement** element, so that it defines default versions for the artifacts provided by JBoss Fuse. Add the following **dependencyManagement** element to your **pom.xml** file:

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

3. Now delete all of the **version** elements in your JBoss Fuse dependencies. All of the versions defined in the JBoss Fuse BOM will be applied automatically to your dependencies (through the Maven dependency management mechanism). For example, if you already had some Apache Camel dependencies, as follows:

```
 <dependencies>
   <dependency>
     <groupId>org.apache.camel</groupId>
     <artifactId>camel-core</artifactId>
<version>2.17.0.redhat-630187</version>
   </dependency>
   <dependency>
     <groupId>org.apache.camel</groupId>
     <artifactId>camel-blueprint</artifactId>
<version>2.17.0.redhat-630187</version>
   </dependency>
   <dependency>
     <groupId>org.apache.camel</groupId>
     <artifactId>camel-jetty</artifactId>
<version>2.17.0.redhat-630187</version>
   </dependency>
   ...
 </dependencies>
```

You would delete the version elements, so that the dependencies are specified as follows:

```
 <dependencies>
```

```
    <dependency>
      <groupId>org.apache.camel</groupId>
      <artifactId>camel-core</artifactId>
    </dependency>
    <dependency>
      <groupId>org.apache.camel</groupId>
      <artifactId>camel-blueprint</artifactId>
    </dependency>
    <dependency>
      <groupId>org.apache.camel</groupId>
      <artifactId>camel-jetty</artifactId>
    </dependency>
    ...
  </dependencies>
```

4. In future, when you migrate to a later version of JBoss Fuse, all that you need to do to upgrade your **`pom.xml`** file is to edit the **`jboss.fuse.bom.version`** property, so that it references the new version of the JBoss Fuse BOM.

CHAPTER 5. MIGRATE DATA STORE

OVERVIEW

JBoss Fuse on Apache Karaf uses a KahaDB data store. There is an automatic migration facility that enables the KahaDB data store to be migrated to the new JBoss Fuse version.

The Aries transaction module must be installed and enabled before it can be used. See Fuse Transaction Guide for more details. Ignore the Aries transaction files instructions below if you do not have Aries installed.

MIGRATE THE KAHADB DATA STORE

Note

When migrating or patching JBoss Fuse, always back up the KahaDB files and Aries transaction files.

1. Backup the KhahaDB files and Aries transaction files from the old container. The files can be found at:
   - KahaDB files - ***InstallDir*/data/amq/kahadb/*.***
   - Aries transaction files - ***InstallDir*/data/txlog/*.***
2. Manually copy all of the KahaDB files from the old container to the same location in the new container.
3. Manually copy all Aries transaction log files from the same location in the sold container to the new container.

Auto-migration will take place when the new container is started.

CHAPTER 6. CAMEL MIGRATION ISSUES

CAMEL 2.17 MIGRATION ISSUES FOR JBOSS FUSE 6.3

JBoss Fuse 6.3 uses Camel 2.17. There are a number of changes in Camel 2.17 that have to be considered before upgrading to JBoss Fuse 6.3.

As part of the Camel CDI component refactoring:

- The **`@ContextName`** qualifier no longer has a default empty value.
- The **`CdiPropertiesComponent`** class has been removed, the standard **`PropertiesComponent`** can be used instead

There are changes to Camel 2.17 that must be considered before upgrading:

- Removed camel-hbase as Karaf feature as it did not work well in OSGi
- The Kafka component has been migrated to use the Java Kafka client instead of Scala. A consequence of this is that there may be migration efforts or code changes required when upgrading.
- Fixed using statement.xxx options on the JDBC consumer would only be used in first poll.
- Fixed HTTP and HTTP4 to keep trailing slash if provided in uri when calling remote HTTP service.
- Fixed **`OnCompletion`** to keep any caught exception stored as property on the Exchange which allows to access that information to know if there was an exception during routing.
- Fixed an issue with Bean component or Simple language with OGNL method call, would pick method with java.lang.Object type over a better suited method, when the method is overloaded.
- When installing the camel feature in Apache Karaf, camel-spring is no longer installed by default. You need to install the camel-spring feature if you are using spring-dm on Karaf.
- Removed camel-docker from karaf features as it does not work in OSGi
- Some changes have been made in Rest DSL to adjust naming and types to the Swagger Spec 2.0
- Any custom component that supports suspension in doSuspend/doResume should implement the new **`Suspendable`** marker interface, so Camel knows there is custom logic for suspension in the component.

- Exchange and Message only output id in their toString method to avoid outputting any message details such as sensitive details from message bodies.
- Upgraded camel-hbase to Hadoop 2.x and HBase 1.1.x
- camel-mustache requires Java 8.
- camel-jetty8 is deprecated and being removed in next release.
- Moved some Camel tooling related dependencies (such as maven/plexus) from the Camel Parent BOM to the tooling BOM (to have them separated).
- camel-amqp does not support 0.9 anymore.
- camel-spring-integration feature has been removed from the Camel karaf.
- The Mail component now requires to configure to, cc, and bcc using lower case keys, eg to=foo@bar.com, instead of To=foo@bar.com as previously.
- The File consumer no longer probe the file content by default. See the option probeContentType for more details.
- If using Bean or Class component and specifying additional parameters in the endpoint uri to configure on the bean, then these options should now be prefixed with bean., eg foo=123 is now bean.foo=123.
- The Twitter delay option is changed from seconds to milli seconds by default, eg 10 should be 10000 to indicate 10 seconds. This is aligned how other components with delay option behaves.
- The options **`attributeNames`** and **`messageAttributeNames`** on AWS-SQS is changed to a string type where you can separate multiple values using comma. Before the type was a Collection which was much harder to configure in the Camel uris.
- Rest DSL is exposing the REST services using all local IP address (eg 0.0.0.0) by default, instead of the local IP address of the host.
- The hbase component now require row mapping from the endpoint uri to be prefixed with row. as prefix.Before:

  ```
  family=info&qualifier=firstName&family2=birthdate&qualifier2=year.
  ```

  After:

  ```
  row.family=info&row.qualifier=firstName&row.family2=birthdate&row.quali
  fier2=year.
  ```

- As part of the Camel CDI component refactoring, **`DeltaSpike`** is not used anymore for the

sourcing of the configuration properties. This new version of the component is agnostic to any configuration sourcing mechanism and delegates that concern the application so that it can declare a custom **`PropertiesComponent`** bean whose sourcing is tailored to its need. **`DeltaSpike`** can still be used by the application by declaring a **`PropertiesComponent`** bean configured with a **`PropertiesParser`** relying on DeltaSpike. See the camel-example-cdi-properties example for more details.

- The Kafka component has been migrated to use the Java Kafka client instead of Scala. As such there may be migration efforts or code changes that can affect users upgrading.
- Improved Rest DSL when CORS enabled to process and return the configured CORS headers in the rest-dsl in all the supported Rest DSL components.
- The options verb in the Rest DSL has been deprecated and are not in use if CORS is enabled.
- camel-gae is deprecated and will be removed from Camel 2.18 onwards.

A known issue is that **`camel-guice`** cannot install in Apache Karaf.

CAMEL 2.16 MIGRATION ISSUES FOR JBOSS FUSE 6.3

There are also a number of updates to Camel 2.16 that may have an impact on upgrading to JBoss Fuse 6.3.

- The **`dumpRoutesAsXml`** operation now preserves the property placeholder used in the route models.
- **`setFaultBody`** and **`setFaultHeader`** behave similarly to **`setBody`** or **`setHeader`** respectively, by preserving existing headers/attachments by setting on existing IN or OUT message.
- If using concurrent consumer on JMS endpoints for request/reply over JMS then you must use the new **`replyToConcurrentConsumers`**, **`replyToMaxConcurrentConsumers`** options to configure the values.
- When the Aggregator2 is force completed the exchange property **`Exchange.AGGREGATED_COMPLETED_BY`** value has been changed from **`forceCompletion`** to **`force`** so its named like the other completion triggers.
- Removed unsupported modules **`camel-web`** and **`camel-web-standalone`**.
- Removed unsupported **`camel:dot`** functionality from **`camel:run`** plugin.
- Removed unsupported **`camel-archetype-scala-component`** from maven archetypes.

- The Maven coordinate for linkedin and olingo2 components has been changed to have **`groupId`** as just org.apache.camel.
- If using MongoDB component, the option **`invokeGetLastError`** has been removed. If this funcitonality is needed, use an acknowledged WriteConcern when executing the write operation and then verify the correctness of the operation with the method **`wasAcknowledged()`** of WriteResult.
- The Jing component now uses **`jing`** as schema name in uris, instead of **`rng`** or **`rnc`**. **`rng`** or **`rnc`** have been removed.
- Swagger module now supports getting api-docs from multiple camel contexts in the JVM. The entry point at **`/api-docs`** now lists the contexts detected, and you need to append the context id in the path, eg **`/api-docs/myCamel`**
- If using <contextScan> with Spring or Blueprint to filter RouteBuilder classes, then Camel will now by default only look for singleton beans. You can turn on the old behavior to include prototype scoped with the new option **`includeNonSingletons`**.
- camel-vertx has been upgraded to vertx 3.0 which requires Java 8 at runtime.
- camel-cdi is now using CDI 1.1 api - support for 1.0 has been dropped.
- Content Enricher with enrich and pollEnrich now supports dynamic endpoint uris. Dynamic endpoint uris are computed using an expression that allows the use of values from the current Exchange. The Java DSL stays backwards compatible.
- WireTap now supports dynamic endpoint uris. Dynamic endpoint uris are computed using an expression that allows the use of values from the current Exchange. The Java DSL stays backwards compatible. The Java DSL stays backwards compatible.
- If you have explicitly configured the JMX statistics level to **`All`** then that option is now called **`Default`**.
- The HTTP based consumers no longer include Camel headers in the responses by default.
- Bindy is requires to be configured using class names instead of package names, as it now supports having multiple model classes in the same java packages.
- Using CamelProxy now binds the method parameters to the message body/header using Camel annotations to define the binding rules. If no annotations are defined, the parameter is assumed to be the message body. You can turn this off to have the old behavior.
- SJMS component has been aligned to bind between Camel Messages and JMS Messages in the same way as the JMS component does. This has caused some APIs and behavior to change.
- **`DefaultExchangeHolder`** now only keeps primitive/String type headers/exchange properties (like JMS component) and filter out other types such as java instances (caught exception on

exchange property is kept as well).

- The Scala based Swagger (camel-swagger) is deprecated in favor of the new **`camel-swagger-java`** component.

APACHE CAMEL 2.16 API BREAKING CHANGES

- **`org.apache.camel.mode.LoadBalancerDefinition`** no longer implements **`org.apache.camel.processor.loadbalancer.LoadBalancer`** which is the runtime processor (this was never intended).

- The ref attribute on <loadBalance> has been removed, as it has been deprecated for a long time.You should use a <customLoadBalancer> to refer to a custom load balancer.

- The copy method on **`StreamCache`** now takes an Exchange as parameter.

- Various APIs in camel-jms has been adjusted to support including the JMS session parameter - **`javax.jms.Session`**. These API changes are mostly internal facing and are not expected to affect end users.

- The **`resourceUri`** and **`resourceRef`** attributes on <enrich> and <pollEnrich> have been removed as they now support a dynamic uris computed from an Expression.

- Various APIs from **`camel-http`** in the package **`org.apache.camel.component.http`** have been moved to the **`camel-http-common`** module in the package **`org.apache.camel.http.common`**, which means that you may need to change the imports.

- Renamed All **`enum`** on **`org.apache.camel.ManagementStatisticsLevel`** to **`Default`**.

- Added new boolean parameter to method on **`org.apache.camel.spi.ShutdownPrepared`**

- Added **`configure`** method to allow configuring CamelContext on **`org.apache.camel.main.MainListener`**.

- Renamed **`org.apache.camel.component.sjms.jms.KeyFormatStrategy`** to **`org.apache.camel.component.sjms.jms.JmsKeyFormatStrategy`**

CHAPTER 7. APACHE CXF ISSUES

7.1. APACHE CXF 3.1 MIGRATION

Overview

JBoss Fuse 6.3 uses Apache CXF 3.1. This introduces some issues that you sould be aware of before migrating.

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.

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.

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_PROPERTY,
address);
        port.addNumbers3(-1, 2);
    }
```

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.