



JB OSS

FUSE

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com



<https://www.facebook.com/tutorialspointindia>



<https://twitter.com/tutorialspoint>

About the Tutorial

JBoss Fuse is an enterprise edition of Apache Servicemix Community Project. Fuse is one of the finest and low-memory footprint based open source ESB. Fuse is emerging as one of the key factors in SOA technologies.

Audience

This tutorial has been prepared for professionals aspiring to make a career in Enterprise integration and ESB. This tutorial will give you enough understanding on creating and deploying Camel Routes and CXF Web Services with basic understanding of JBoss Fuse.

Prerequisites

Before proceeding with this tutorial, you must have a good understanding of Core Java and Maven.

Copyright & Disclaimer

© Copyright 2016 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com.

Table of Contents

About the Tutorial.....	1
Audience.....	1
Prerequisites.....	1
Copyright & Disclaimer.....	1
Table of Contents	2
1. INTRODUCTION TO ESB	5
What is ESB?	5
The Integration Problem	5
Why ESB?.....	6
SOA & ESB?	7
2. WHAT IS FUSE?	8
Integration to Fuse	8
Architecture	8
Components.....	8
Installing Fuse	9
Basic Configuration	10
Configuring Maven	12
HAWTIO	13
3. APACHE KARAF	14
The JVM Problem	14
OSGi.....	14
Bundles Vs Features	14
Why another Container?.....	15
4. APACHE CAMEL	17
What is Apache Camel?	17

Basic Example	17
Install Project into Fuse	20
5. CAMEL CONCEPTS	22
EIP	23
Splitter	24
Recipient List	26
Exception Handling in Camel	27
Deploying Bundle in Fuse	28
6. APACHE CXF	30
What is Apache CXF?	30
SOAP	30
SOAP Development Using CXF	31
7. REST WEB SERVICES	33
REST Development using CXF	33
Create Service Class	35
Create Blueprint.xml	36
8. APACHE AMQ	38
What is AMQ?	38
Types of Messaging	38
Creating Queue and Topics	39
Browsing /Deleting Contents of the Queue	41
9. AMQ WITH CAMEL	42
Configuring to ActiveMQ Component	42
10. FABRIC	48
What is Fabric?	48
Why Fabric?	48

Fabric Setup	49
Profiles.....	49
Deploying a Bundle	52
Un-deploying a Bundle	53
10. CHILD CONTAINER	55
Creating a Child container	55
Managing a Child Container	56
11. FUSE – ISSUES AND SOLUTIONS.....	58
Code Changes are not Reflected	58
Bundle not Being Downloaded	58
Not Able to Login into FMC (Browser based GUI)	59
HAWTIO Port is Different	59
AMQ Broker is not working	60

1. Introduction to ESB

In this chapter, we will start with the essentials of Enterprise Service Bus. Given below is a detailed explanation about ESB along with its advantages, disadvantages and a couple of diagrams for easier understanding.

What is ESB?

ESB stands for Enterprise Service Bus. ESB in its simplest form is a middleware which acts as an information highway aiding multiple applications to communicate.

In the enterprise world, we develop solutions for many things. These solutions may use different technologies and different data formats. It becomes cumbersome to use these solutions together due to compatibility variance of communication or data format in these technologies. Therefore we need a technology that will allow **loosely coupled integration** between these different solutions.

ESB aims to simplify this problem of integration by becoming a 'HUB' that sits in the middle of all your applications and facilitates message routing between them. ESB serves as a mediator, acting as information highway, taking care of data transformation routing, leaving the Coder or the Developer to focus on his own application logic.

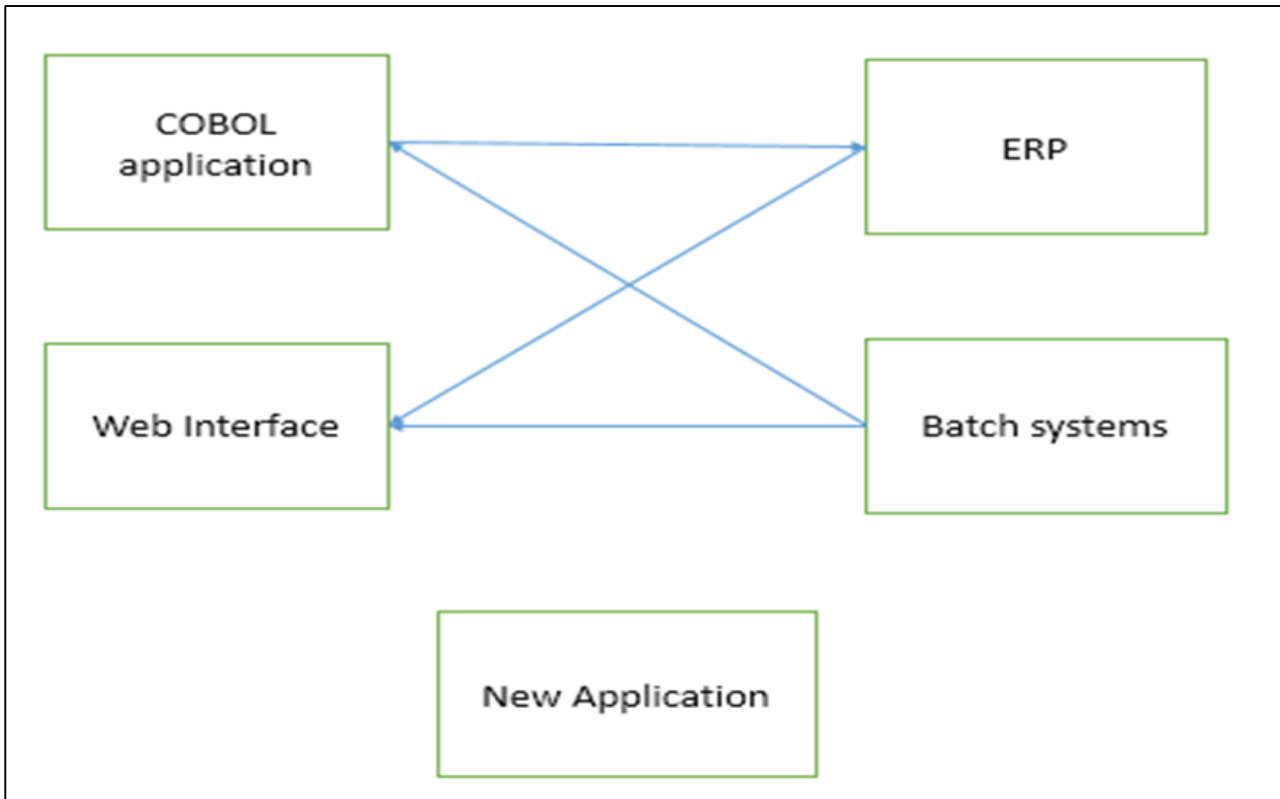
Understanding ESB becomes very simple when we understand the problem for which it was especially designed and the solution becomes easy. One should have a clear understanding of how to enable many disparate systems, written in different languages and running on different machines using different data formats to share information and form an integrated business platform.

The Integration Problem

In the enterprise platform, it is common for multiple applications to collaborate and provide business functionality as a whole, but integration of these applications is the most recurring problem. It becomes even difficult with time as applications grow.

Each application may input and output data in their own format. This approach works well if the number of applications is less, but as the number of applications grows, the integration wheels also need to be churned with a better approach. For instance, if a particular application for a business needs to be changed, its output or input data format for all the applications having dependency on that Master application are affected.

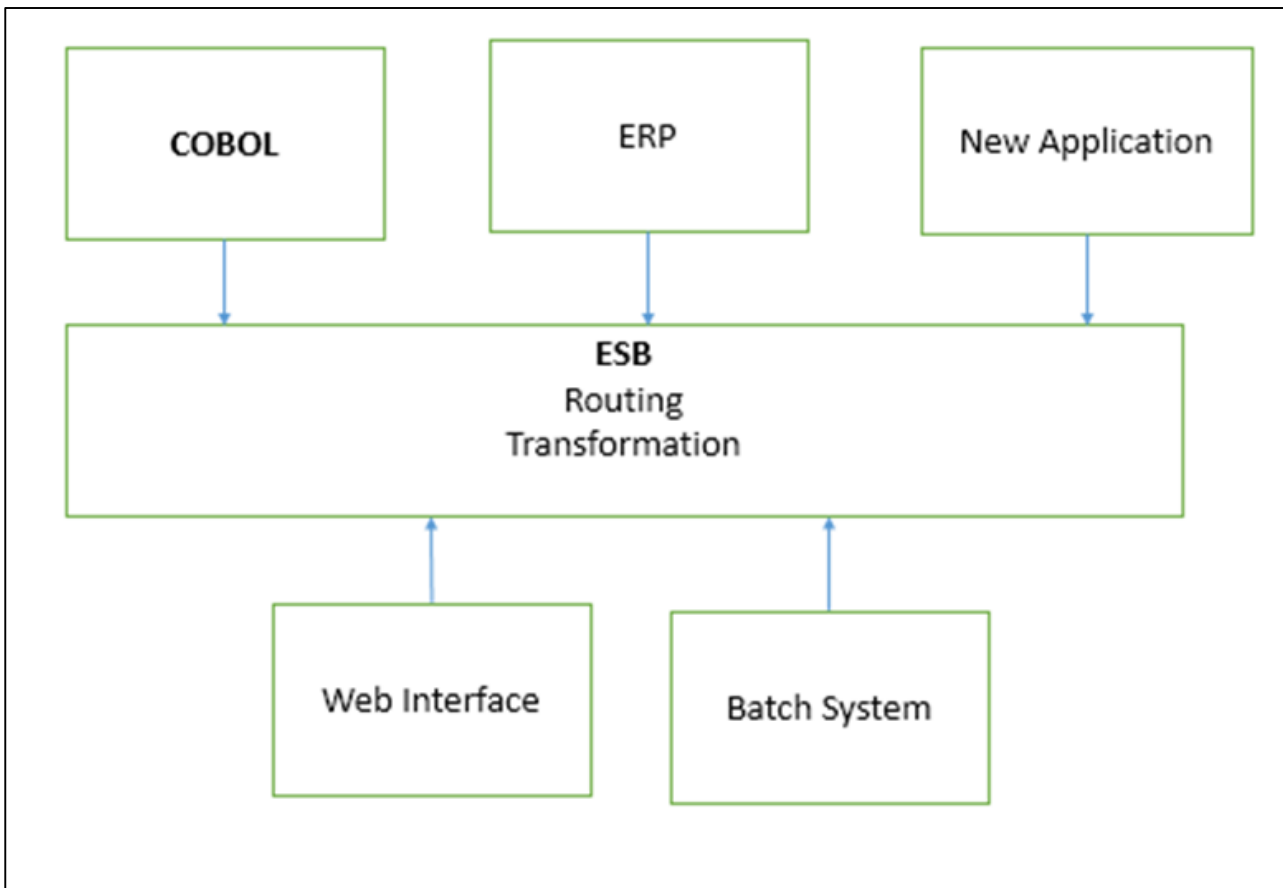
Such an approach serves as the biggest hurdle for the Integration which expects a tightly coupled architecture. This is where ESB comes into the picture. Each application need not communicate directly with other application; instead, all the applications communicate with the ESB and the ESB handles the routing of information and internal data format conversion.



Why ESB?

Following are a few points which explain why Enterprise Service Bus is essential.

- ESB aims to simplify the problem of integration with variant compatible applications.
- It acts as a Middleware, which serves as a mediator of all your applications and facilitates message routing between them.
- Instead of every application interfacing with every other application directly, each application now just has one interface to the ESB.
- The ESB is responsible for translating messages to/from a common format and routing them to their destinations.
- The major saving in this approach comes as a boon if you have to replace any of your existing applications. Instead of writing a whole bunch of new interfaces, you now only have one interface to be concerned about (between your application and the ESB).



SOA & ESB?

SOA and ESB are commonly used interchangeably, but they are completely different.

SOA is a design pattern which allows application to expose its functionalities as a service over network via communication protocols, whereas ESB is a model which facilitates communication between disparate systems, but ESB can be used as a backbone while implementing SOA.

2. What is Fuse?

JBoss Fuse is an Open source ESB solution by Redhat. It is an enterprise solution based on community project, Apache Servicemix.

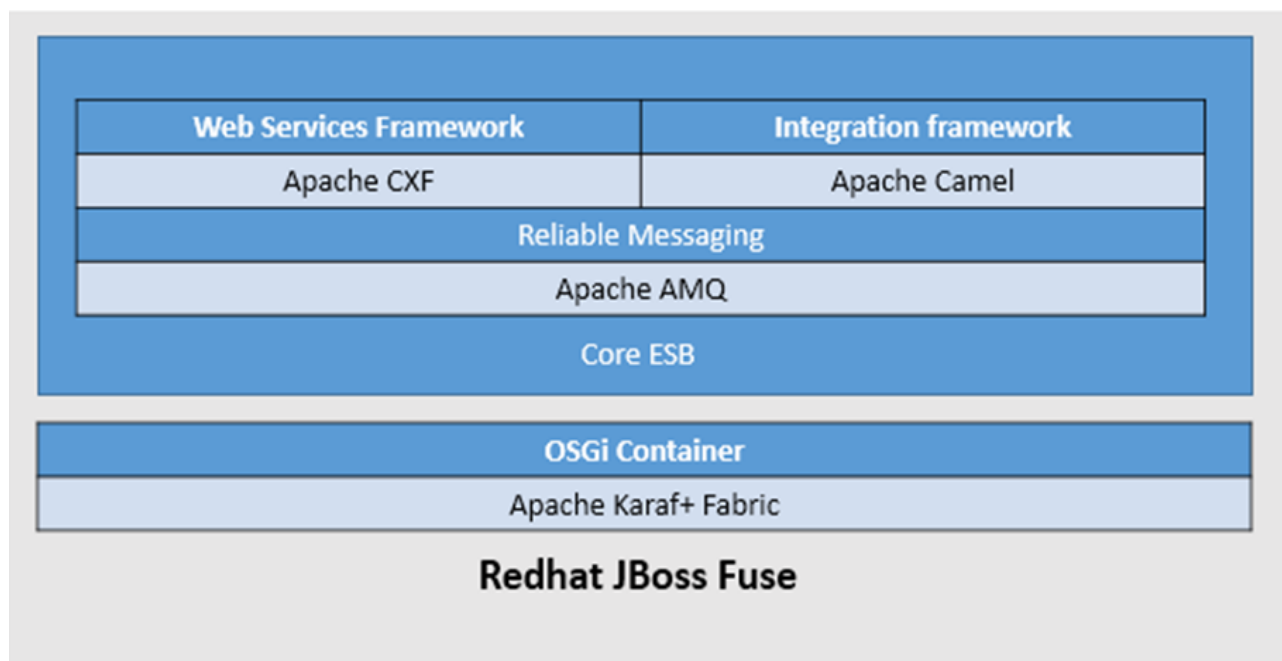
Integration to Fuse

JBoss Fuse is a lightweight and flexible integration platform which allows rapid integration of enterprise applications.

Fuse was initially developed by Progressive software Inc. which was acquired by Redhat in 2012. JBoss Fuse 6.1.0.redhat-379 GA is a stable version of Fuse which can be downloaded from their official website.

Architecture

Fuse combines various technologies together as a single product.



Components

Apache CXF

Apache CXF is an open source web services development framework which also supports development of SOAP & Rest web services.

Apache Camel

Apache Camel is a EIP based integration framework. EIP or Enterprise Integration patterns are identified solutions to the recurring problems in Enterprise Integration. Complete integration solution can be achieved meteorically with combinations of these pre-defined out of the box patterns.

It allows to write routing logic in several Domain Specific Languages like Java, Spring DSL, and Scala etc.

Apache AMQ

Apache AMQ is a JMS which provides reliable messaging system as per JMS standards. It not only support JMS specification but also provides some exciting and useful features which are not included in JMS specifications.

Apache Karaf

Apache Karaf is lightweight OSGi container which acts as runtime for the artifacts. Apache Karaf is more dynamic in nature as compared to JVM. It allows to install or uninstall modules at runtime. All the artifacts in Fuse are deployed in Karaf.

Fabric

Fabric provides easy way to manage deployments of artifacts in a large and distributed environment. It provides centralized management for all multiple fuse instances.

Installing Fuse

Installing Fuse is quite simple. Like other JBoss products, Fuse comes as a zip file that can be extracted and after some minor configuration changes it can directly be started.

Installing Fuse is a four step process –

Download

Download Fuse 6.1.0 GA from the following link.

http://www.jboss.org/download-manager/file/jboss-fuse-6.1.0.GA-full_zip.zip

Unzip

Like all the other JBoss products, Fuse is also a platform independent zip.

Unzip the downloaded file into the destination directory you want to be the Fuse installation directory. Choose this directory wisely as this should remain same over the lifetime of Fuse instance.

Note: *Even though Fuse unzips and starts like other JBoss products, it is not recommended to move Fuse installation from one location to another location after installation is complete.*

Configure

After you unzip Fuse, you will find the following directories inside the extracted Directory:

- bin
- etc
- deploy
- lib
- licenses
- extras
- quickstarts

Out of which we are going to use only two directories **bin** & **etc**.

Virtually after extracting Fuse, we should be able to start fuse directly, but this will start Fuse with all the default configurations which is not advisable for production environment. It is strongly recommended to do the following changes before starting Fuse.

Set Environment variables

- Set the following Environment variables – **JAVA_HOME**
- The variable should point to the java installation directory – **M2_HOME**
- The variable should point to Maven installation directory – **PATH**
- Set the path variable to include Java & Maven executables.

Windows

On windows, settings can be done by following the below given instructions:

Start → My Computer → Right Click → Properties → Advanced System settings → Environment variables.

UNIX & Clones

For each user there is a bash profile in the ***nix** operating systems. We can add or edit the existing system variable by changing this file.

```
$ vi ~/.bash_profile
```

Note: Any changes in this file are permanent. It is highly recommended to take a backup of the existing file before changing the original.

Basic Configuration

We will discuss about the basic configuration of JBoss Fuse and for that we have to start with the following command **Edit \$FUSE_INSTALLATION_DIR/etc/**

- In **user.properties**
 - #admin=admin,admin

```

Please wait while JBoss Fuse is loading...
100% [-----]
JBoss Fuse
JBoss Fuse (6.1.0.redhat-379)
http://www.redhat.com/products/jbossenterprise middleware/fuse/
Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Open a browser to http://localhost:8181 to access the management console
Create a new Fabric via 'fabric:create'
or join an existing Fabric via 'fabric:join [someUrls]'
Hit '<ctrl-d>' or 'osgi:shutdown' to shutdown JBoss Fuse.
JBossFuse:karaf@root> █

```

- This needs to be changed according to the first admin with username we want, second admin with password, third one might be kept as it is because it indicates a role and don't forget to remove #
 - For example – FuseAdmin=FusePass,admin
- In **System.properties**
 - karafName=root
 - This indicates the name you want to give to Karaf instance.
 - We can name it anything we want like Cont1.
 - Make sure this name you give is unique name and not already being used by another instance of Fuse.
 - In **org.ops4j.pax.web.cfg**
 - Org.osgi.service.http.port=8181
 - This property indicates the port that is to be used for accessing browser-based interface HAWTIO provided by Fuse
 - HAWTIO is an in-built browser interface to Fuse which is available from 6.0 onwards
 - In **org.ops4j.pax.url.mvn.cfg**
 - org.ops4j.pax.url.mvn.localRepository=D:/repository

- This property indicates the path to localRepository of our Maven from where Fuse will install its artifacts.
- `org.ops4j.pax.url.mvn.settings=D:/Maven/conf/settings.xml`
- This property indicates settings.xml which Fuse should use to get artifacts from Maven.

Configuring Maven

Maven is a prerequisite for installing Fuse. If you don't know what maven is please refer to <http://www.tutorialspoint.com/maven/>

Maven is a built tool used for building Fuse artifacts. Fuse first searches in Maven local repository for artifacts when we issue command to install artifact. So we must let Fuse know where Maven is installed and the path of Maven's local repository.

Edit `$FUSE_INSTALLATION_DIR/etc/org.ops4j.paxurl.mvn.cfg`

Update the following two properties:

- `org.ops4j.pax.url.mvn.settings=$M2_HOME/conf /settings.xml`
- `org.ops4j.pax.url.mvn.localRepository=$local_repo`

Note: Please change `$local_repo` with the actual path of your local repository mentioned in Mavens settings.xml.

Run

After doing basic configuration changes, we can now start Fuse. All the binary files to work with Fuse are located in `$FUSE_INSTALLATION_DIR`.

There are two ways to start Fuse:

- Using `./fuse`
 - This will allow you to see all the progress and logs on the same window in which you started Fuse.
 - It will give you Karaf console in the same terminal as shown below.

Note: *This will start fuse in console mode which means Fuse process will also be stopped when user logs out from session or closes Terminal which is not desirable in production or development scenario. This script should be used only for debugging Fuse.*

- Using `./start`
 - This won't show any logs on screen not even the progress but this will start Fuse in background and Fuse service won't be stopped when user exits session or closes terminal.
 - In the real world Application, this type of behavior is desired. Fuse should be running in the background even if we close the terminal.

- If you want to connect to Fuse running in the background, you can use **client** script which is located in the same folder.
- You should get the display as shown in the following screenshot.
- Exiting from client script won't stop Fuse service. It will just close the Fuse console.

```

Please wait while JBoss Fuse is loading...
100% [-----]

JBoss Fuse

JBoss Fuse (6.1.0.redhat-379)
http://www.redhat.com/products/jbossenterprisemiddleware/fuse/

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.

Open a browser to http://localhost:8181 to access the management console

Create a new Fabric via 'fabric:create'
or join an existing Fabric via 'fabric:join [someUrls]'

Hit '<ctrl-d>' or 'osgi:shutdown' to shutdown JBoss Fuse.

JBossFuse:karaf@root> █

```

HAWTIO

Fuse also provides complete GUI access to it using FMC (Fuse management console). You can find GUI on below URL <http://localhost:8181>.

State	Context	Route	Completed #	Failed #	Inflight #	Mean Time	Min Time	Max Time
●	camel-1	route1	2	0	0	267	55	480

Everything we did by executing commands can also be done by accessing this browser-based GUI. It becomes extremely helpful when we have more than one container and we are running in a Fabric environment.

3. Apache Karaf

In this chapter, we will discuss about Apache Karaf and why it is called as a lightweight OSGi Container along with its benefits and other important features.

The JVM Problem

JVM or Java virtual Machine does not act as an actual virtual machine. A machine which will allow you to stop, start or restart components running inside it on the fly. It may sometimes allow hot deployments at class level but there is no way you could deploy or undeploy a component of your application in your virtual machine without restarting it.

To solve this problem and allow modularity in Java application, Fuse uses an OSGi based runtime known as Apache Karaf.

OSGi

The OSGi technology is a set of specifications that define a dynamic component system for java. These specifications allow a development model where applications are (dynamically) composed of many different (reusable) components.

Benefits of OSGi

- **Reduced Complexity** – Application is built as collaborating components which hide their implementation details from each other resulting in reduced complexity.
- **Reusability** – Many components can leverage same component deployed in a container.
- **Deployment** – OSGi provides support for start, stop and update of components on the fly with its lifecycle management APIs without container restart.

Bundles Vs Features

Following is the comparison between Bundles and Features.

Bundles

Bundles are equivalent to OSGi what jars are to JVM. Bundles are artifacts which are deployable in an OSGi container. The bundles are components which work together or independently to form an application.

These bundles can be installed, uninstalled, updated, started or stopped at runtime without restarting the container.

Features

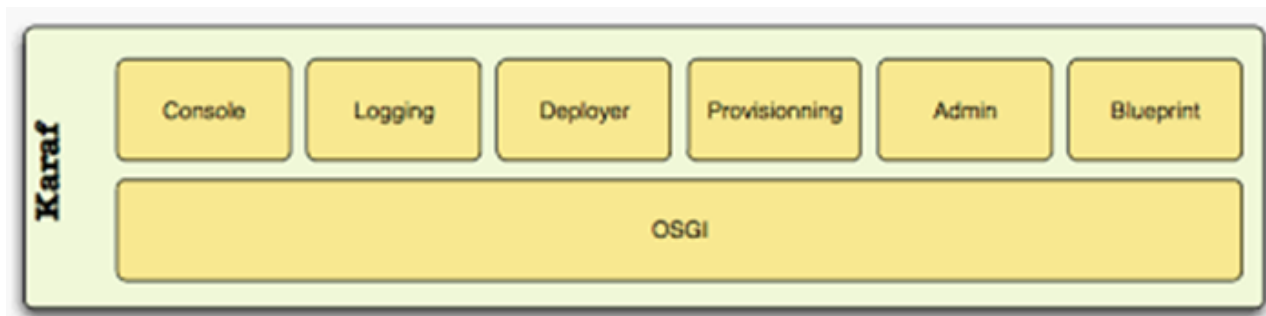
Features are a way of deploying multiple bundles together. Sometimes it makes more sense to deploy bundles in group. Features allow us to deploy a group of bundles with just one command.

Why another Container?

Apache Karaf is an OSGi based runtime, it is where our Application bundles run. Fuse uses Apache Karaf as its runtime in which bundles run and collaborate to provide business functionality.

Karaf is built on Felix and equinox which are OSGi Frameworks.

Karaf Architecture



Apache Karaf adds the following additional functionalities to basic OSGi runtime.

Hot Deployment

Karaf supports hot deployment. It contains a hot deploy directory. Anything that is placed in this directory is automatically deployed and installed in Karaf as a bundle.

Logging

Karaf provides centralized logging by generating logs for all bundles in **\$Fuse_home/data/log**. We can edit logger configuration in **org.ops4j.pax.logging.cfg** in **\$Fuse_home/etc directory**.

Admin console

Karaf provides a sophisticated and lucid Admin console to interact with running instance of fuse. It also provides a pre-installed set of commands which can be used to manage and monitor components (Bundle) at runtime. This console is extensible so it allows us to add new commands to the console by adding new bundles to console.


```
Please wait while JBoss Fuse is loading...
100% [-----]

JBoss Fuse

JBoss Fuse (6.1.0.redhat-379)
http://www.redhat.com/products/jbossenterprisemiddleware/fuse/

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.

Open a browser to http://localhost:8181 to access the management console.

Create a new Fabric via 'fabric:create'
or join an existing Fabric via 'fabric:join [someUrls]'

Hit '<ctrl-d>' or 'osgi:shutdown' to shutdown JBoss Fuse.

JBossFuse:karaf@root> █
```

SSH Access

Karaf allows remote access to this Admin console with SSH. Anyone with valid credentials can connect to karaf admin console over SSH terminal.

4. Apache Camel

In this chapter, we will discuss what Apache Camel is and how it effectively routes data between endpoints, along with a few examples.

What is Apache Camel?

Apache Camel is an open source integration framework which was started in early 2007.

It is an EIP (Enterprise Integration Pattern) based approach which provides several out of the box patterns implementations that can be used to solve enterprise integration problems. EIP are nothing but proven solutions to the well documented and recurring problems in enterprise integration.

Camel is also known as routing and mediation engine as it effectively routes data between endpoints, while taking heavy load like transformation of data formats, endpoint connectivity and many more.

Basic Example

The prerequisites to use Apache Camel are:

- Java
- Maven
- Redhat JBoss Fuse 6.1-GA-379

Create basic skeleton of Application

```
mvn:archetype generate -DgroupId=com.tutorialpoint.app -DartifactId=camel-first-app -DarchetypeGroupId=org.apache.camel.archetypes -DarchetypeArtifactId=camel-archetype-spring -DinteractiveMode=false -X
```

This should generate the following directory structure.



This is a basic skeleton of our Camel application being generated.

Edit camel-context.xml

Edit **camel-first-app** → **src** → **main** → **resources** → **META-INF\spring\camel-context.xml** to match as below

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Configures the Camel Context-->
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://camel.apache.org/schema/spring
http://camel.apache.org/schema/spring/camel-spring.xsd">

  <camelContext xmlns="http://camel.apache.org/schema/spring">
    <!-- here is a sample which processes the input files
         (leaving them in place - see the 'noop' flag)
         then performs content based routing on the message using XPath -->
    <route>
      <from uri="file:///d:/src/data?noop=false"/>
      <choice>
        <when>
          <xpath>/person/city = 'London'</xpath>

```

```

<log message="UK message"/>
    <to uri="file:///d:/target/messages/uk"/>
</when>
<otherwise>
    <log message="Other message"/>
    <to uri="file:///d:/target/messages/others"/>
</otherwise>
</choice>
</route>
</camelContext>
</beans>

```

Edit pom.xml

Add the following code inside <plugins></plugins>

```

<plugin>
    <groupId>org.apache.felix</groupId>
    <artifactId>maven-bundle-plugin</artifactId>
    <version>2.3.4</version>
    <extensions>>true</extensions>
    <configuration>
        <instructions>
            <Bundle-SymbolicName>${project.artifactId}</Bundle-SymbolicName>
            <Import-Package>*</Import-Package>
        </instructions>
    </configuration>
</plugin>

```

Change packaging type from **jar** → **bundle**.

```
<packaging>bundle</packaging>
```

Build the project using the following command:

```
mvn clean install
```

Install Project into Fuse

Start Fuse using **Fuse.bat/start.bat**. If you start Fuse using **start.bat**, use **client.bat** to connect to Fuse. You should get the UI as shown in the following screenshot.

```
Please wait while JBoss Fuse is loading...
100% [=====]

JBoss Fuse
JBoss Fuse (6.1.0.redhat-379)
http://www.redhat.com/products/jbossenterprisemiddleware/fuse/

Hit '<tab>' for a list of available commands
and '<cmd> --help' for help on a specific command.

Open a browser to http://localhost:8181 to access the management console

Create a new Fabric via 'fabric:create'
or join an existing Fabric via 'fabric:join [someUrls]'

Hit '<ctrl-d>' or 'osgi:shutdown' to shutdown JBoss Fuse.

JBossFuse:karaf@root>
```

This is the CLI for accessing Karaf and Fuse commands.

```
install -s mvn:com.tutorialpoint.app/camel-firt-app/1.0-SNAPSHOT
```

Test if your Project is Running

Now your application should be installed in Fuse. Copy data directory inside **camel-first-app** and place it in **D:/src/** and it should copy message having city=London into **D:/target/messages/uk**.

Place the input file in **D:/src/data**

Input

Message1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<person user="james">
  <firstName>James</firstName>
  <lastName>Strachan</lastName>
  <city>London</city>
</person>
```

Message2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<person user="hiram">
  <firstName>Hiram</firstName>
  <lastName>Chirino</lastName>
  <city>Tampa</city>
</person>
```

Output

In D:/target/messages/uk

```
<?xml version="1.0" encoding="UTF-8"?>
<person user="james">
  <firstName>James</firstName>
  <lastName>Strachan</lastName>
  <city>London</city>
</person>
```

In D:/target/messages/others

```
<?xml version="1.0" encoding="UTF-8"?>
<person user="hiram">
  <firstName>Hiram</firstName>
  <lastName>Chirino</lastName>
  <city>Tampa</city>
</person>
```

5. Camel Concepts

In this chapter, we will understand the different Camel Concepts. Let us start by taking a basic example to understand core concepts to begin with.

CamelContext

Every camel application will have at least one CamelContext. This is the place where we add camel routes. It is similar to **ApplicationContext** of Spring.

Camel context can be thought as a container which keeps all things together. One camel context can have multiple routes inside it.

Routes

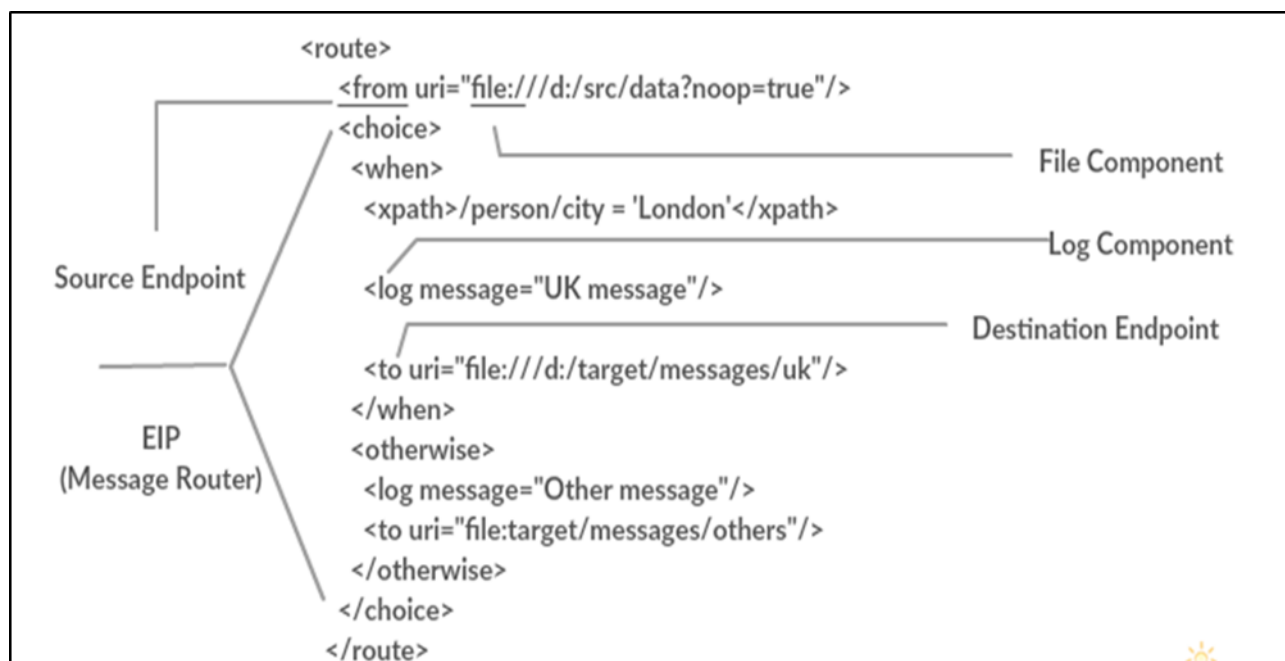
CamelContext may contain one or more routes. Routes are the integration logic which defines how data will flow in camel context from one endpoint to another.

Endpoint

Endpoint is end of channel through which system can send or receive messages. This is what we call as destination or source in communication language.

Components

Components are point of extension in Camel. Components can be an interface to technology, data format, transformers, etc. They may also act as a factory for endpoints.

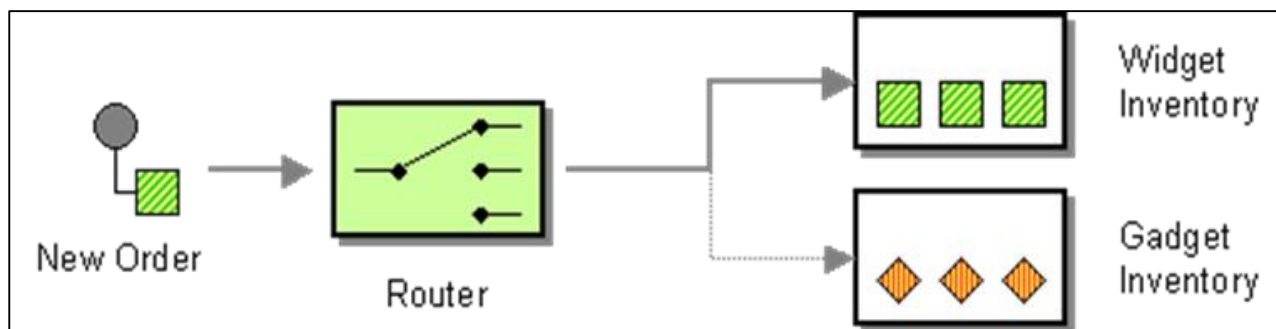


EIP

EIP stands for Enterprise Integration Pattern. These are identified and well-known solutions to a recurring problem. Camel supports most of the Enterprise Integration Patterns.

Content Based Router

CBR patterns allow us to route data as per the content of the input file.



This pattern is used when we have to route values depending on the contents of the body of input.

The following example will read data from **D:/data/input** directory. After reading, it will check for value tag inside the data tag. If the value tag contains **value1**, it will be sent to **D:/value1**, If it contains value2, it will be sent to **D:/value2** and if none of these both, then it will be sent to others.

```
<CamelContext xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="file:///D:/data/input"/>
    <choice>
      <when>
        <xpath>/data/value = 'value1'</xpath>
        <to uri="file:///D:/value1"/>
      </when>
      <when>
        <xpath>/data/value = 'value2'</xpath>
        <to uri="file:///D:/value2"/>
      </when>
      <otherwise>
        <to uri="file:///D:/others "/>
      </otherwise>
    </choice>
  </route>
</CamelContext>
```



```

    </route>
</camelContext>

```

Input

D:/data/input/message1.xml

```

<data>
  <value>value1</value>
</data>

```

D:/data/input/message2.xml

```

<data>
  <value>value2</value>
</data>

```

Output

D:/value1/

```

<data>
  <value>value1</value>
</data>

```

D:/Value2/

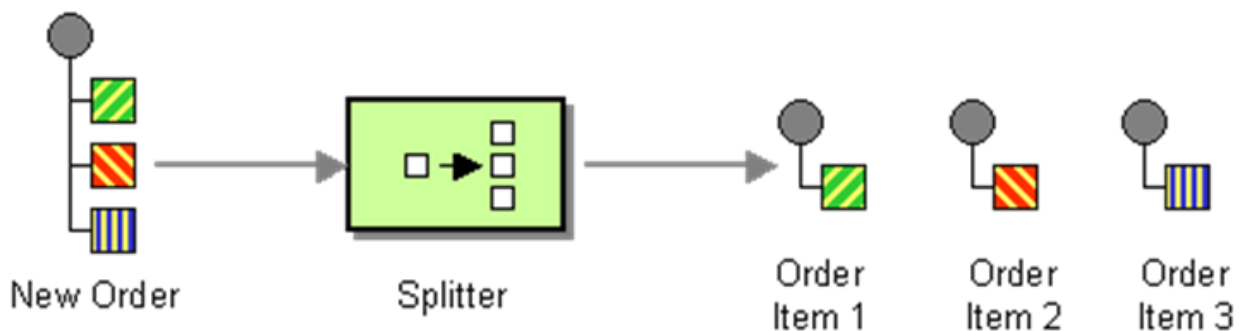
```

<data>
  <value>value2</value>
</data>

```

Splitter

A splitter pattern is used to split input data into smaller chunks.



This pattern is used most of the times with huge data input which requires to be split in chunks, so it becomes process-able. It breaks down input into smaller fragments based on input token string.

```
<CamelContext xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="file:///D:/inbox"/>
    <split streaming="true">
      <tokenize token="order" xml="true"/>
      <to uri="activemq:queue:order"/>
    </split>
  </route>
</CamelContext>
```

Input

D:/inbox/message.xml

```
<order>
  <data>
    <value>value1</value>
  </data>
</order>
<order>
  <data>
    <value>value2</value>
  </data>
</order>
<order>
  <data>
    <value>value3</value>
  </data>
</order>
```

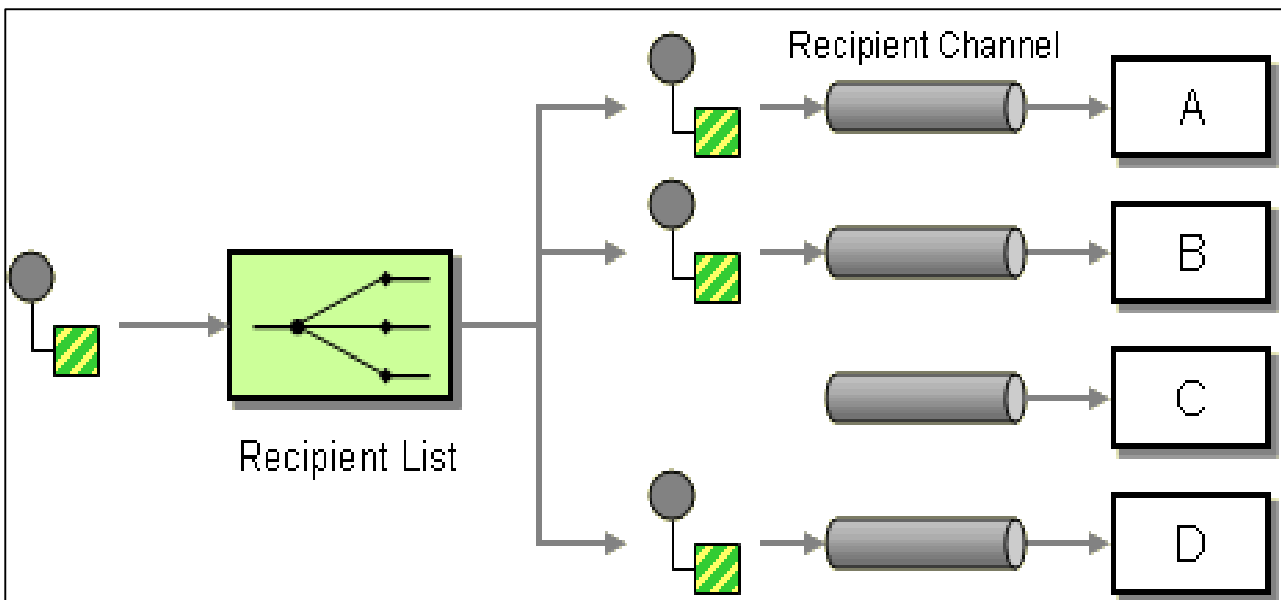
Output

If you check AMQ you will find 3 messages posted.

```
<order>
  <data>
    <value>value4</value>
  </data>
</order>
```

Recipient List

A recipient list pattern is used when a list of recipient needs to be retrieved from the message body itself.



In the following example, a message will be sent to all the recipients who are listed in the customer tag as comma separated list of strings.

```
<CamelContext xmlns="http://camel.apache.org/schema/spring">
<route>
  <from uri="jms:xmlOrders" />
  <recipientList>
    <xpath>/order/customer</xpath>
  </recipientList>
</route>
</camelContext>
```

Other EIPs

Camel provides support to almost all the EIPs identified. Some of commonly used EIP are as mentioned below.

- **Log** - To log complete message or part of it
- **Message Filter** - Filtering contents of messages
- **Re-Sequencer** – To get all tokens in sequence
- **Wiretap** - To inspect travelling messages

The complete list of EIP and their usage can be found at Camel's official documentation <http://camel.apache.org/eip.html>

Exception Handling in Camel

Using Error Handler – This is the easiest way to handle exceptions in camel.

To use this, we have to configure Error handler class bean and provide it as reference to **CamelContext errorHandlerRef** attribute.

```
<bean id="loggingErrorHandler" class="org.apache.camel.builder.LoggingErrorHandler">
  <property name="logName" value="mylogger.name"/>
  <property name="level" value="DEBUG"/>
</bean>
<camelContext errorHandlerRef=" loggingErrorHandler" >
...
</camelContext>
```

Using Try Catch Finally

Camel also supports Java style **Try Catch Finally block** for error handling.

Just like Java, it has the following three blocks:

- **doTry** block contains code that may generate exception.
- **doCatch** block contains code that needs to be executed in case of exception.
- **doFinally** block has code that must be executed irrespective of exception. It will always be executed no matter if exception was raised or not.

Note: *Mock is testing component and not recommended for other purposes. It is the component in camel used for testing just like jMock component in Test driven development.*

```
<route>
  <from uri="direct:start"/>
  <doTry>
```

```

        <process ref="someProcessorThatmayFail"/>
    <to uri="mock:result"/>

    <doCatch>
        <exception>java.io.IOException</exception>
        <exception>java.lang.IllegalStateException</exception>
        <to uri="mock:catch"/>
    </doCatch>
    <doFinally>

    <to uri="mock:finally"/>
    </doFinally>
</doTry>
</route>

```

In the above example, we can give a list of exceptions that need to be handled by the catch block.

Deploying Bundle in Fuse

Start Fuse using **Fuse.bat/start.bat**.

If you start Fuse using start.bat, use client.bat to connect to Fuse. You should get the UI as shown in the following screenshot.

```

Please wait while JBoss Fuse is loading...
100% [=====]

JBoss Fuse
JBoss Fuse (6.1.0.redhat-379)
http://www.redhat.com/products/jbossenterprise middleware/fuse/

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.

Open a browser to http://localhost:8181 to access the management console

Create a new Fabric via 'fabric:create'
or join an existing Fabric via 'fabric:join [someUrls]'

Hit '<ctrl-d>' or 'osgi:shutdown' to shutdown JBoss Fuse.

JBossFuse:karaf@root>

```

This is the CLI for accessing Karaf and Fuse commands.

```
install -s mvn:group.id /artifact.id/version
```

```
e.g. install -s mvn:com.tutorialpoint.app/camel-firt-app/1.0-SNAPSHOT
```

6. Apache CXF

In this chapter, let us discuss about what Apache CXF is and how it can be helpful in developing SOAP and Rest Web Services.

What is Apache CXF?

Apache CXF is a web service development framework that can be utilized to develop SOAP and Rest web services. CXF is fully compliant with **JAX-RS and JAX-WS** standard.

It is most widely used web service development framework now. CXF has learned and improved over Axis2 which is now gradually being replaced by CXF.

CXF vs Axis2

	CXF	Axis2
Improvements	CXF is most used framework as of now. It has lot <i>improvements</i> over Axis2	Axis2 is gradually being replaced by CXF. It requires more code as compared to CXF
Code required	CXF requires less code as compared to Axis2	Axis2 requires more code comparatively
Standard Compliance	CSF is fully compliant with JAX-RS and JAX-WS	Axis2 is not fully compliant with JAX-RS and JAX-WS
Compatible with Spring	Yes	No
Separation of front-ends	Clean separation of front-end from JAX-WS code	No clean separation is provided

SOAP

SOAP stands for Simple Object Access Protocol. It is a protocol for exchanging structured information over web services between two systems. It mostly relies on XML for structuring data and uses HTTP or SMTP for message negotiation and transmission.

There are two approaches to develop SOAP web services:

- **Code first** – In this approach, WSDL is generated from code.
- **Contract first** – In contract first, code is generated from WSDL.

SOAP Development Using CXF

Configure Maven

Add the following profile to your settings.xml of Maven.

```
<profiles>
  <profile>
    <id>Jboss-Fuse</id>
    <activation>
      <activeByDefault>>true</activeByDefault>
    </activation>
    <repositories>
      <repository>
        <id>fusesource</id>
        <url>http://repo.fusesource.com/nexus/content/groups/public/</url>
        <snapshots>
          <enabled>>false</enabled>
        </snapshots>
        <releases>
          <enabled>>true</enabled>
        </releases>
      </repository>
    </repositories>
  </profile>
</profiles>
```

Create Skeleton

```
mvn archetype:generate -DarchetypeGroupId=org.apache.servicemix.tooling -
DarchetypeArtifactId=servicemix-cxf-code-first-osgi-bundle -
DarchetypeVersion=2012.01.0.redhat-60024 -DgroupId=org.fusesource.example -
DartifactId=cxf-basic -Dversion=1.0-SNAPSHOT
```


Build Web Service Project

```
mvn clean install
```

Install web-service into Fuse using the following command

```
JBossFuse:karaf@root>install -s mvn:org.fusesource.example/cxf-basic/1.0-SNAPSHOT
```

Check if bundle has registered SOQP web-service

Open URL <http://localhost:8181/cxf>



The web-service should be listed as follows.

Testing Web-Service

```
mvn -Pclient
```

INFO: Creating Service {<http://ws.tutorials.com/>}PersonService from class com.to

```
torials.ws.Person
Invoking getPerson...
getPerson._getPerson_personId=Guillaume
getPerson._getPerson_ssn=000-000-0000
getPerson._getPerson_name=Guillaume
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 30.668 s
[INFO] Finished at: 2016-02-15T21:01:20+05:30
[INFO] Final Memory: 10M/37M
[INFO] -----
```

7. REST Web Services

To begin with, REST stands for Representational State Transfer. It is a way of developing web services based on state-less, cacheable, client-server protocol, which is HTTP in most cases.

REST web services use HTTP requests to post, get, delete data from network.

REST Development using CXF

Create a simple Maven quick-start project

```
mvn archetype:generate -DgroupId=com.tuts.abhinav -DartifactId=rest-service
-DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

Add dependencies

```
<dependency>
  <groupId>org.apache.servicemix.specs</groupId>
  <artifactId>org.apache.servicemix.specs.jsr311-api-1.1.1</artifactId>
  <version>1.9.0</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.apache.servicemix</groupId>
  <artifactId>servicemix-http</artifactId>
  <version>2013.01</version>
</dependency>
<dependency>
  <groupId>log4j</groupId>
  <artifactId>log4j</artifactId>
  <version>1.2.16</version>
</dependency>
```

Add Build Instruction

```
<build>
  <defaultGoal>install</defaultGoal>
</build>
```

```

    <plugins>
      <plugin>
        <groupId>org.apache.felix</groupId>
        <artifactId>maven-bundle-plugin</artifactId>
        <version>2.3.4</version>
        <extensions>>true</extensions>
        <configuration>
          <instructions>
            <Bundle-SymbolicName>rest-example-database-
post-method</Bundle-SymbolicName>
            <Import-Package>* </Import-Package>
          </instructions>
        </configuration>
      </plugin>
    </plugins>
  </build>

```

Add Fuse Plugin Repositories

```

<pluginRepositories>
  <pluginRepository>
    <id>fusesource.m2</id>
    <name>FuseSource Community Release Repository</name>
    <url>http://repo.fusesource.com/nexus/content/repositories/releases</url>
    <snapshots>
      <enabled>>false</enabled>
    </snapshots>
    <releases>
      <enabled>>true</enabled>
    </releases>
  </pluginRepository>

```

Add Repositories

```

<repositories>
  <repository>

```

```

        <id>fusesource.m2</id>
        <name>FuseSource Community Release Repository</name>
        <url>http://repo.fusesource.com/nexus/content/repositories/releases</url>
        <snapshots>
            <enabled>>false</enabled>
        </snapshots>
        <releases>
            <enabled>>true</enabled>
        </releases>
    </repository>
    <repository>
        <id>fusesource.ea</id>
        <name>FuseSource Community Early Access Release Repository</name>
        <url>http://repo.fusesource.com/nexus/content/groups/ea</url>
        <snapshots>
            <enabled>>false</enabled>
        </snapshots>
        <releases>
            <enabled>>true</enabled>
        </releases>
    </repository>
</repositories>

```

Create Service Class

Create class UserService.java under com/tuts/

```

package com.tuts;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/UserService_1")
public class UserService
{

```

```

@GET
@Path("/get_data")
@Produces(MediaType.APPLICATION_JSON)
public String getUser() {

    String reponse = "This is standard response from REST";

    return reponse;
}
}

```

Create Blueprint.xml

Create blueprint.xml under/src/main/resources/OSGI-INF/blueprint blueprint.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:jaxrs="http://cxf.apache.org/blueprint/jaxrs"
    xsi:schemaLocation="
        http://www.osgi.org/xmlns/blueprint/v1.0.0
        http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
        http://cxf.apache.org/blueprint/jaxrs
        http://cxf.apache.org/schemas/blueprint/jaxrs.xsd">
    <jaxrs:server id="service" address="/users">
        <jaxrs:serviceBeans>
            <ref component-id="userService" />
        </jaxrs:serviceBeans>
    </jaxrs:server>

    <bean id="userService" class="com.tuts.UserService" />
</blueprint>

```

Install Rest service in Fuse

```
install -s mvn:com.tuts.abhinav/rest-service/1.0-SNAPSHOT
```

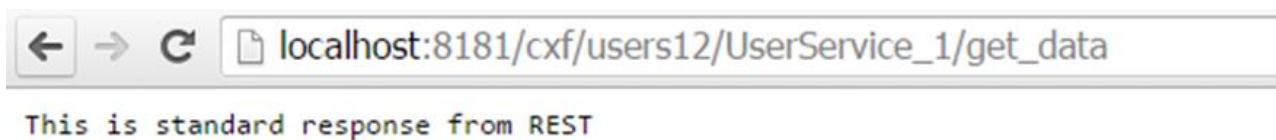
Check if Bundle has a Registered Web-Service

Open URL <http://localhost:8181/cxf>



Test Web Service

Open URL http://localhost:8181/cxf/users12/UserService_1/get_data



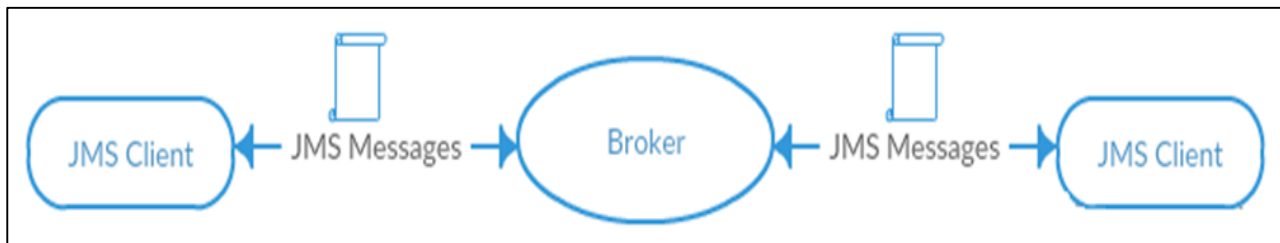
8. Apache AMQ

In this chapter, we will get to know about ActiveMQ and how it acts as a broker of messages to allow applications to communicate with each other.

What is AMQ?

ActiveMQ is an open source message broker written in Java. It's fully compliant with JMS 1.1 standards.

JMS is a specification that allows development of message based system. ActiveMQ acts as a broker of messages which sits in between applications and allows them to communicate in asynchronous and reliable way.



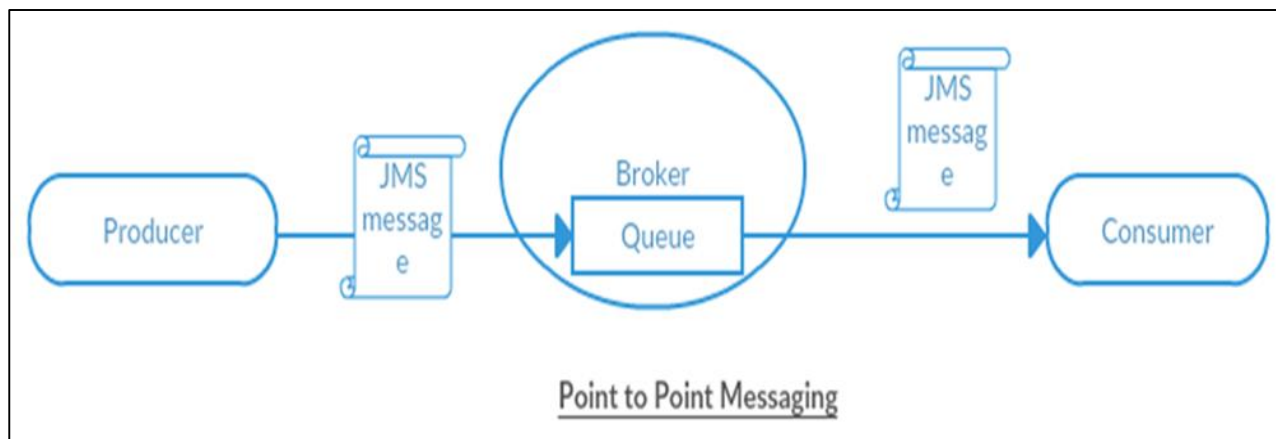
Types of Messaging

There are two types of messaging options explained below for better understanding.

Point to Point

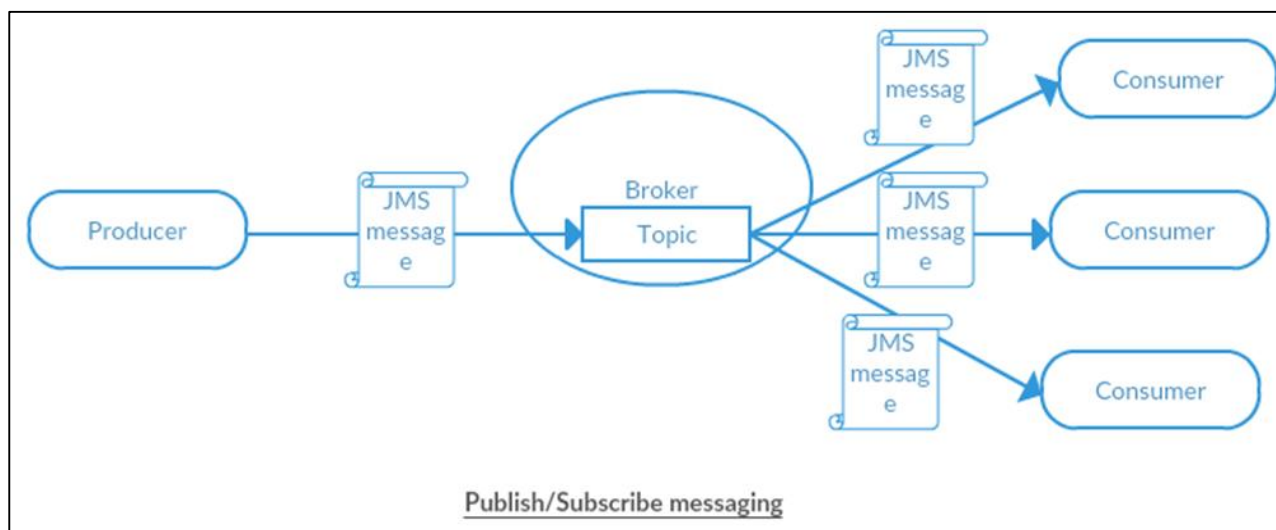
In this type of communication, the broker sends messages to only one consumer, while the other consumers will wait till they get the messages from the broker. No consumer will get the same message.

If there are no consumers, the Broker will hold the messages till it gets a consumer. This type of communication is also called as **Queue based communication** where the Producer sends messages to a queue and only one consumer gets one message from the queue. If there is more than one consumer, they may get the next message but they won't get the same message as the other consumer.



Publish/Subscribe

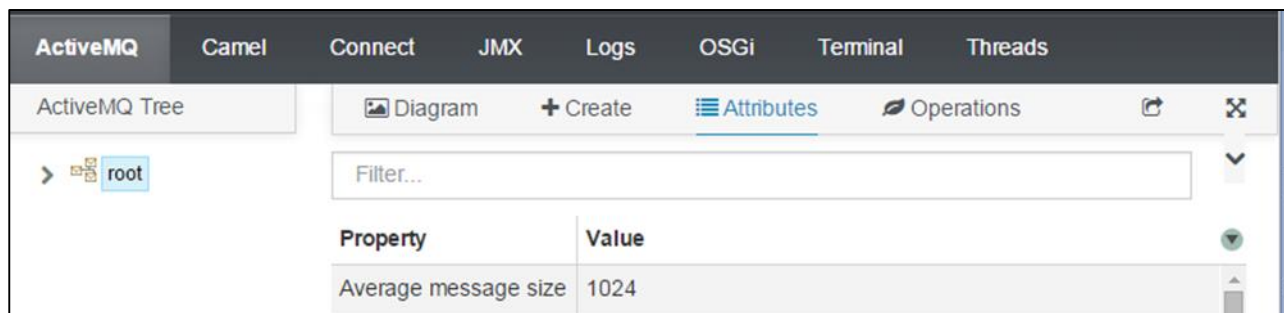
In this type of communication, the Broker sends same copy of messages to all the active consumers. This type of communication is also known as **Topic based communication** where broker sends same message to all active consumer who has subscribed for particular Topic. This model supports one-way communication where no verification of transmitted messages is expected.



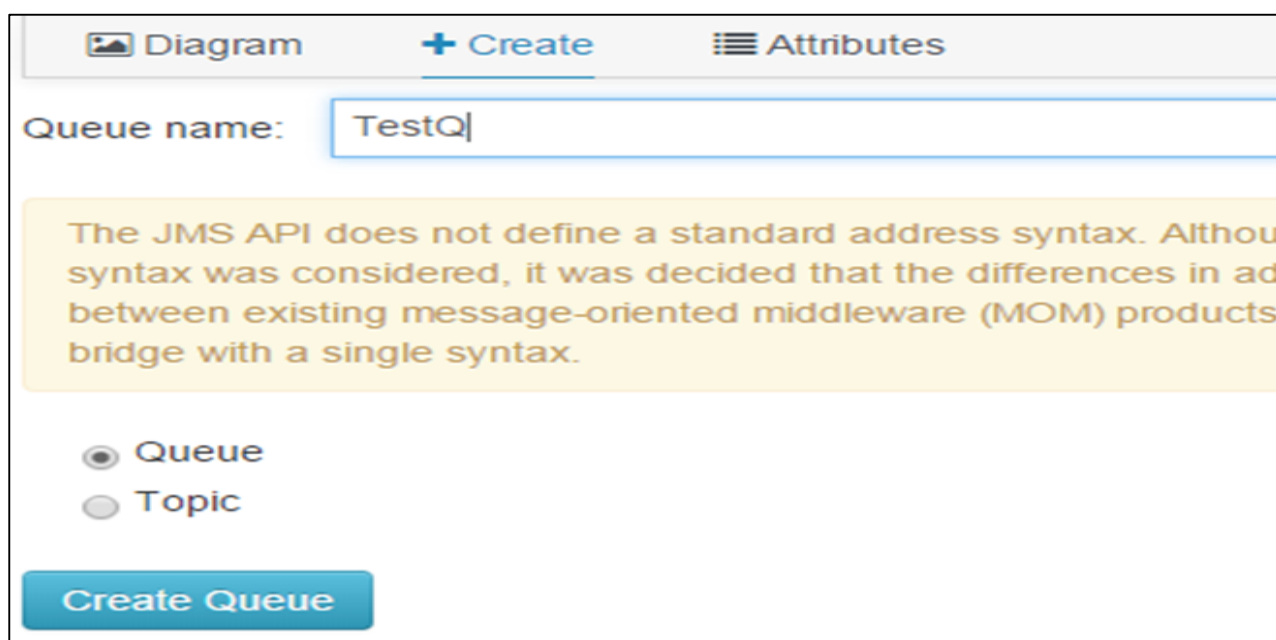
Creating Queue and Topics

Fuse comes bundled with ActiveMQ. We can access ActiveMQ using FMC console (the browser based interface to work with AMQ).

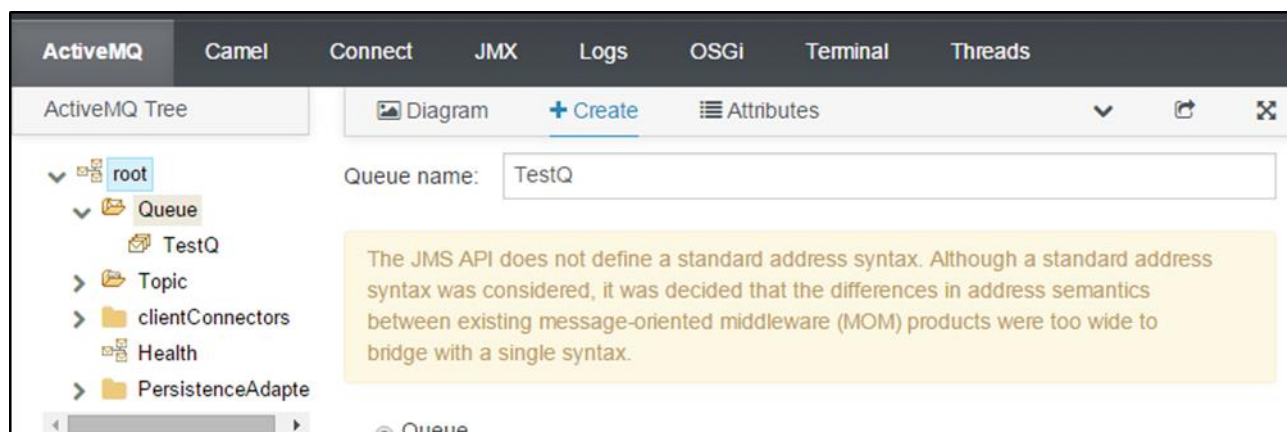
Login to FMC using **localhost:8181** and select **ActiveMQ** tab.



- Click on +Create
- Enter Queue/Topic name
- Select Queue/Topic from radio button
- Click on Create Queue/Create topic



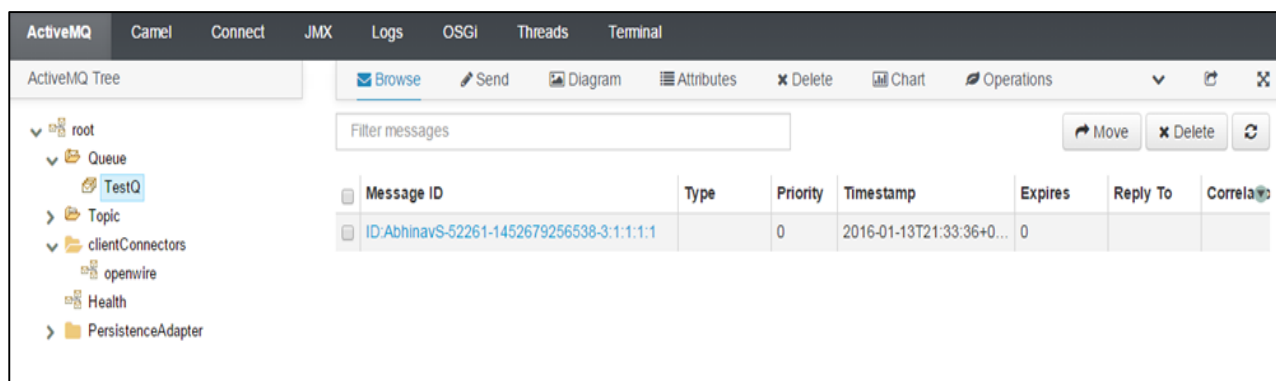
Now you should be able to see the **TestQ** created under root → Queue →



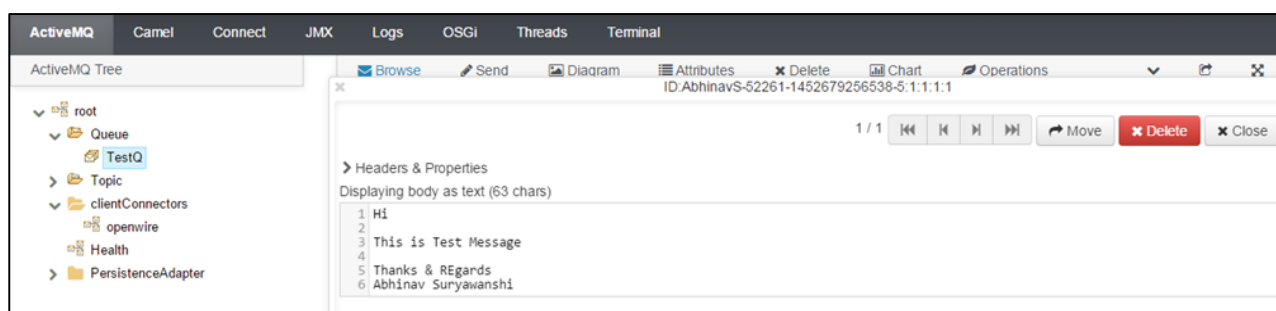
To check the topic created follow root→Topic.

Browsing /Deleting Contents of the Queue

- Login to FMC using localhost:8181
- Select ActiveMQ tab
- Root → Queue → TestQ <select queue that you want to browse> → Browse



- To check contents of this message, click on that particular message.



- You can delete a particular message by clicking on the Delete button shown on the top right corner

9. AMQ with Camel

In this chapter, we will learn the basics of how ActiveMQ works with Camel.

Configuring to ActiveMQ Component

Before we can use ActiveMQ queue or topic in our code we have to configure ActiveMQComponent. Minimal configuration of ActiveMQComponent can be done as shown in the following program –

```
<bean id="activemq" class="org.apache.activemq.camel.component.ActiveMQComponent">
  <property name="brokerURL" value="tcp://localhost:61616"/>
  <property name="userName" value="admin"/>
  <property name="password" value="admin"/>
</bean>
```

- **brokerURL**- Specifies host and port for AMQ Broker.
- **username** - Specifies username to use for connecting to AMQ Broker.
- **password** - specifies password for connecting to AMQ Broker.

Connecting to Queue

Now that we have configured ActiveMQComponent, we can use it in our CamelContext as endpoint.

We will use AMQ endpoint in the following format –

```
Activemq:[queue|topic]:[queueName|topicName]
```

Writing Messages to AMQ

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Configures the Camel Context-->
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://camel.apache.org/schema/spring
    http://camel.apache.org/schema/spring/camel-spring.xsd">
```

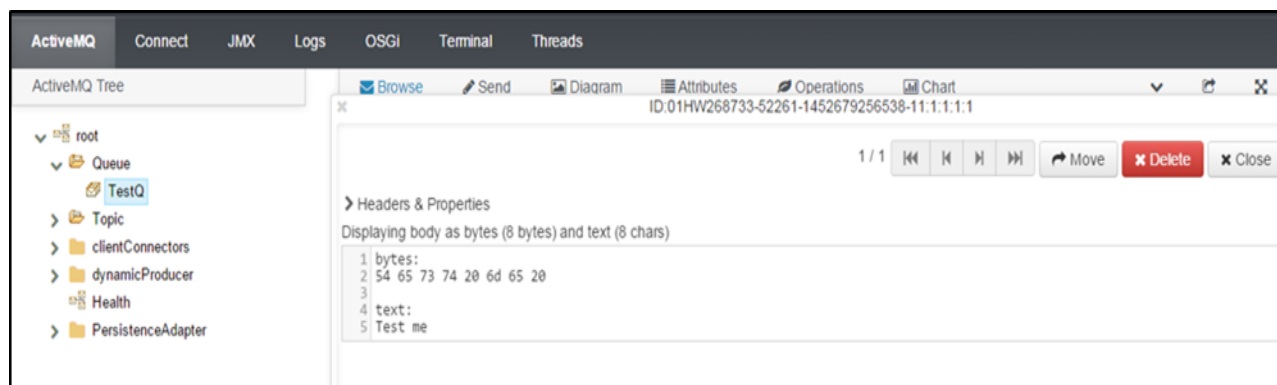
After deploying this bundle in Fuse container, you should be able to see messages posted to AMQ which were placed as files in **D:/src/data**.

Input

D:/src/data/input.txt

Test me

Output



Reading from AMQ

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Configures the Camel Context-->

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd
           http://camel.apache.org/schema/spring
           http://camel.apache.org/schema/spring/camel-spring.xsd">

  <camelContext xmlns="http://camel.apache.org/schema/spring">
    <!-- here is a sample which processes the input files
         (leaving them in place - see the 'noop' flag)
         then performs content based routing on the message using XPath -->
    <route>
      <from uri="activemq:queue:TestQ"/>
      <to uri="file:///d:/src"/>
    </route>
  </camelContext>
</beans>
```

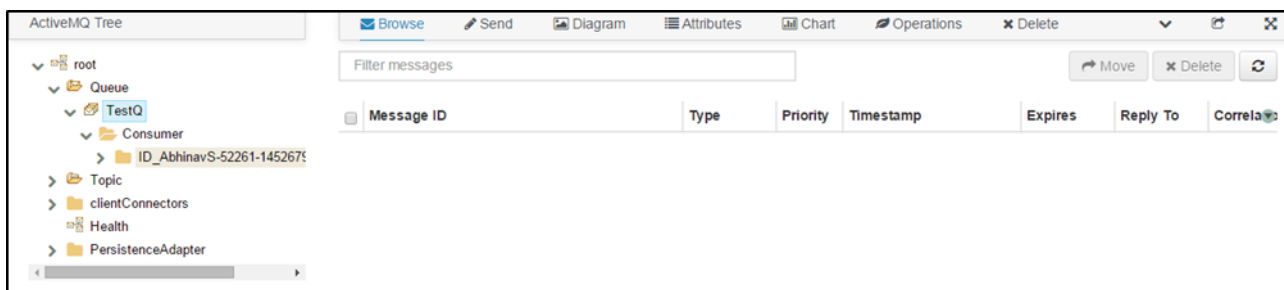
```

    </route>
</camelContext>
<bean id="activemq" class="org.apache.activemq.camel.component.ActiveMQComponent">
  <property name="brokerURL" value="tcp://localhost:61616"/>
  <property name="userName" value="admin"/>
  <property name="password" value="admin"/>
</bean>
</beans>

```

Input

After deploying this bundle, you should see a file being generated in D:/src and messages are consumed. Also Consumer should be shown for that Queue.



Output

D:/src

Test me

Writing to Topic

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- Configures the Camel Context-->

<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://camel.apache.org/schema/spring
    http://camel.apache.org/schema/spring/camel-spring.xsd">

```

```

<camelContext xmlns="http://camel.apache.org/schema/spring">
  <!-- here is a sample which processes the input files
        (leaving them in place - see the 'noop' flag)
        then performs content based routing on the message using XPath -->
  <route>
    <from uri="file:///d:/src"/>
    <to uri="activemq:topic:TestTopic" />
  </route>
</camelContext>
<bean id="activemq" class="org.apache.activemq.camel.component.ActiveMQComponent">
  <property name="brokerURL" value="tcp://localhost:61616"/>
  <property name="userName" value="admin"/>
  <property name="password" value="admin"/>
</bean>
</beans>

```

Reading from Topic

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- Configures the Camel Context-->

<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
    http://camel.apache.org/schema/spring
http://camel.apache.org/schema/spring/camel-spring.xsd">

  <camelContext xmlns="http://camel.apache.org/schema/spring">
    <!-- here is a sample which processes the input files
          (leaving them in place - see the 'noop' flag)
          then performs content based routing on the message using XPath -->
    <route>
      <from uri="activemq:topic:TestTopic"/>

```

```

    <to uri="file:///d:/src2"/>
  </route>
</camelContext>
<bean id="activemq" class="org.apache.activemq.camel.component.ActiveMQComponent">
  <property name="brokerURL" value="tcp://localhost:61616"/>
  <property name="userName" value="admin"/>
  <property name="password" value="admin"/>
</bean>
</beans>

```

Input

D:/src/file1.xml

```

<order>
  <data>
    <value>value1</value>
  </data>
</order>
<order>
  <data>
    <value>value2</value>
  </data>
</order>
<order>
  <data>
    <value>value3</value>
  </data>
</order>

```

Output

D:/src/

```

<order>
  <data>
    <value>value1</value>
  </data>

```

```
</order>  
<order>  
  <data>  
    <value>value2</value>  
  </data>  
</order>  
<order>  
  <data>  
    <value>value3</value>  
  </data>  
</order>
```


10. Fabric

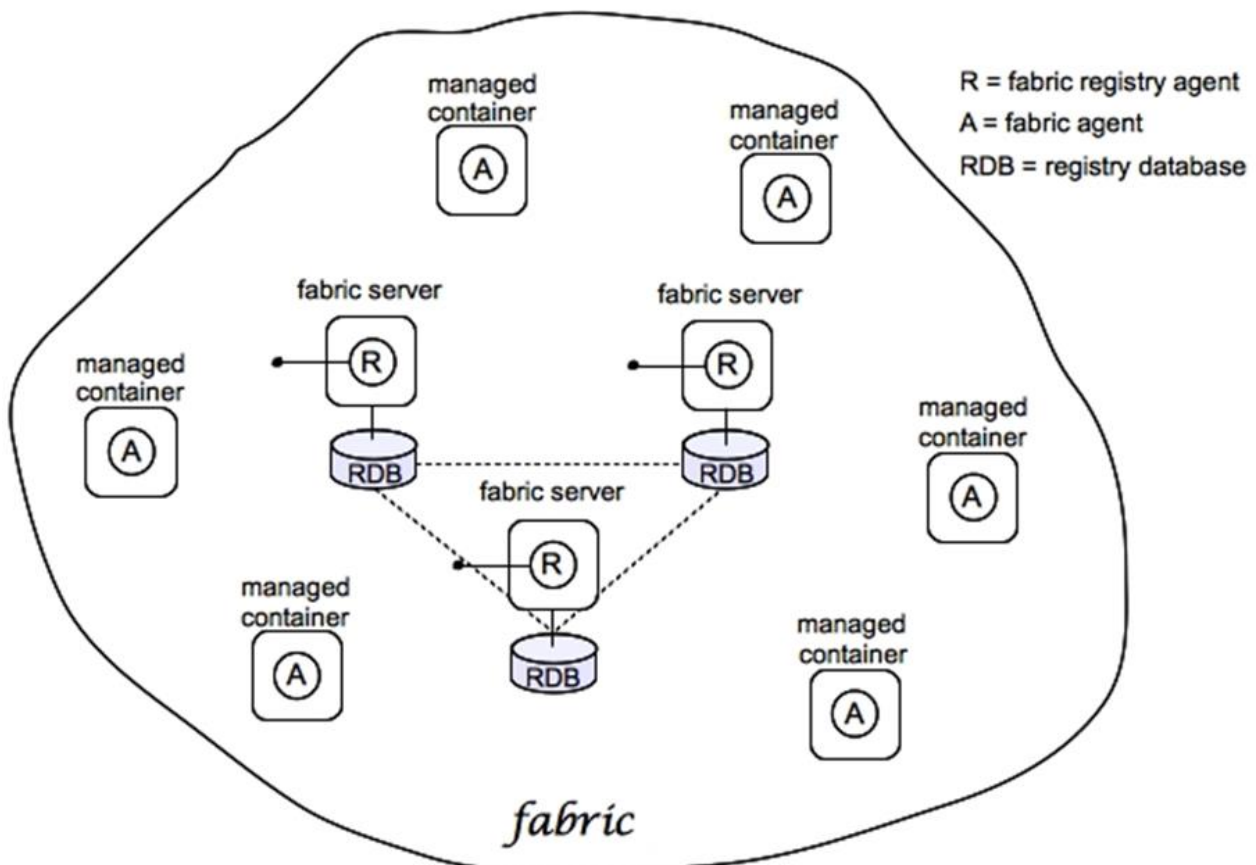
What is Fabric?

Fabric provides management and orchestration capabilities for multiple Fuse instances. Fabric allows us to control all Fuse instances connected to it from a single point. A normal Fuse container can be converted to act as a Fabric. Fabric has fabric registry in it which serves as data store that contains all information regarding containers, it manages.

Why Fabric?

Fabric has the following special abilities which makes it an ideal candidate for use in distributed environments.

- Monitoring the state of all containers in the fabric.
- Starting and stopping remote containers.
- Provisions remote container to run a particular application.
- Upgrading applications and rolling out patches in the live system.
- Starting and provisioning with new containers quickly for example to cope with increased load on the system.



Fabric Setup

Creating Fabric

Normal Fuse container can be converted to Fabric by using the following command

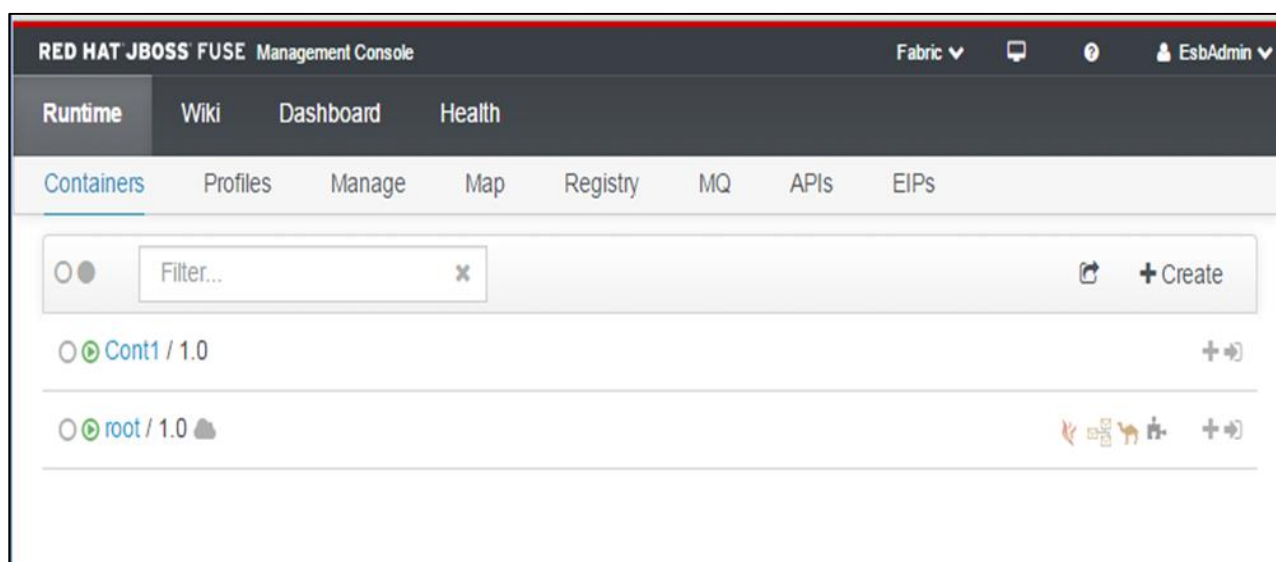
```
fabric: create --clean --zookeeper-password myZooPass
```

Connecting other container to Fabric:

```
fabric:join --zookeeper-password myZooPass <fabric_host>:2181 Cont1
```

Note: Please replace <fabric_host> with actual host name on which fabric is running.

When you login to the Fuse Management Console from your browser using **localhost:8181**, you should be able to see two containers as shown in the following screenshot. The Fabric container is indicated by a small cloud symbol in front of it.



Profiles

A Profile contains the following information –

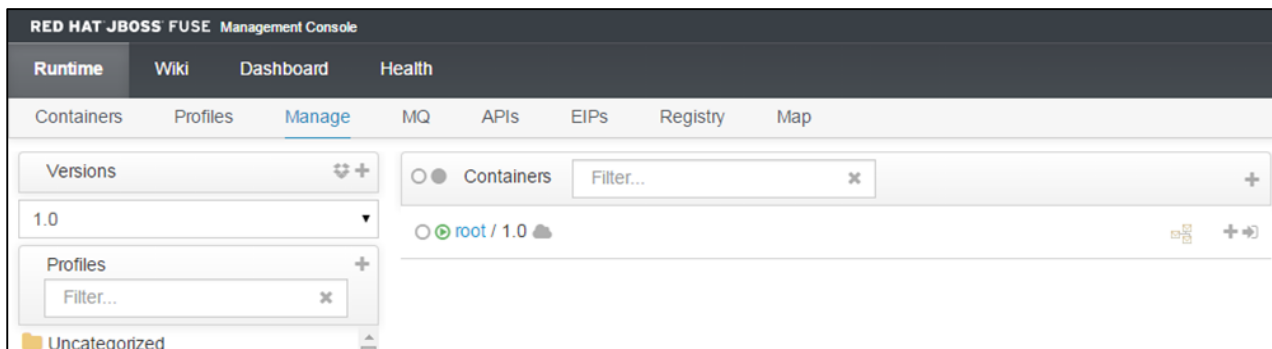
- Bundles to be installed
- Features to be installed
- Configurations to be applied

A Profile provides a way in fabric environment to install same set of bundles, features and configuration on multiple servers.

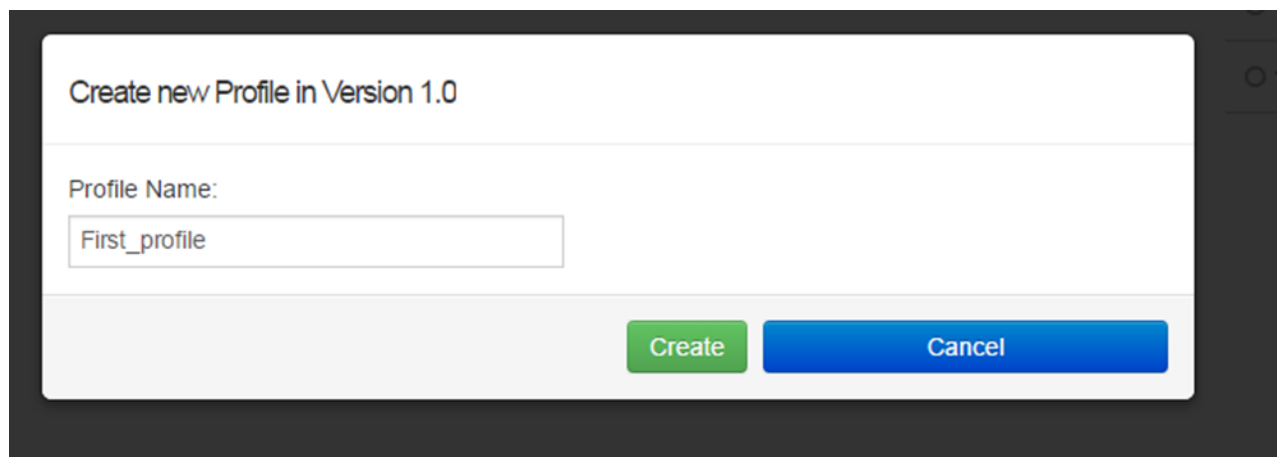
If the same profile is applied to multiple containers and we do changes to that profile from any container similar changes will be deployed automatically to the remaining containers to which it is applied.

Creating Profiles

- Login in to FMC localhost:8181
- Runtime → Manage
- In the left hand side under Profile menu click on +



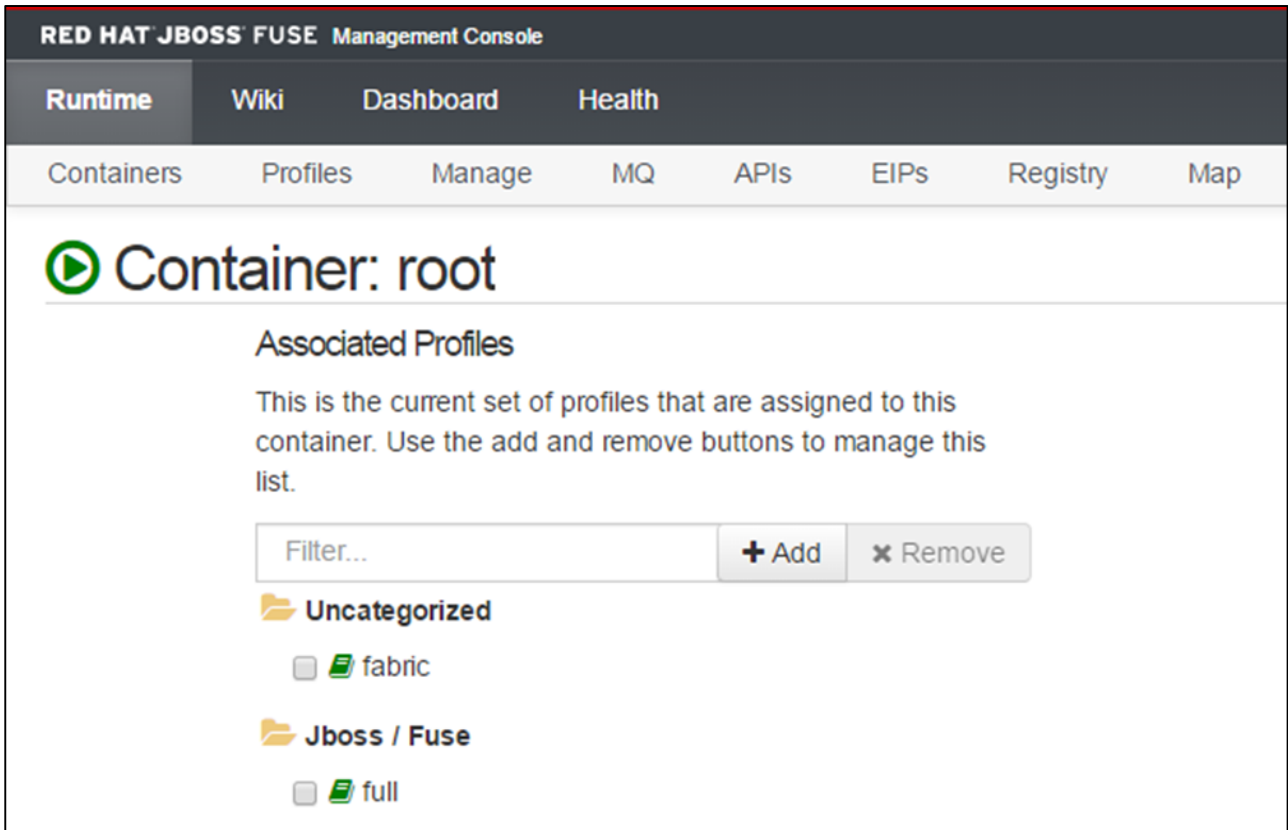
Enter the name you want to give to the profile and click create.



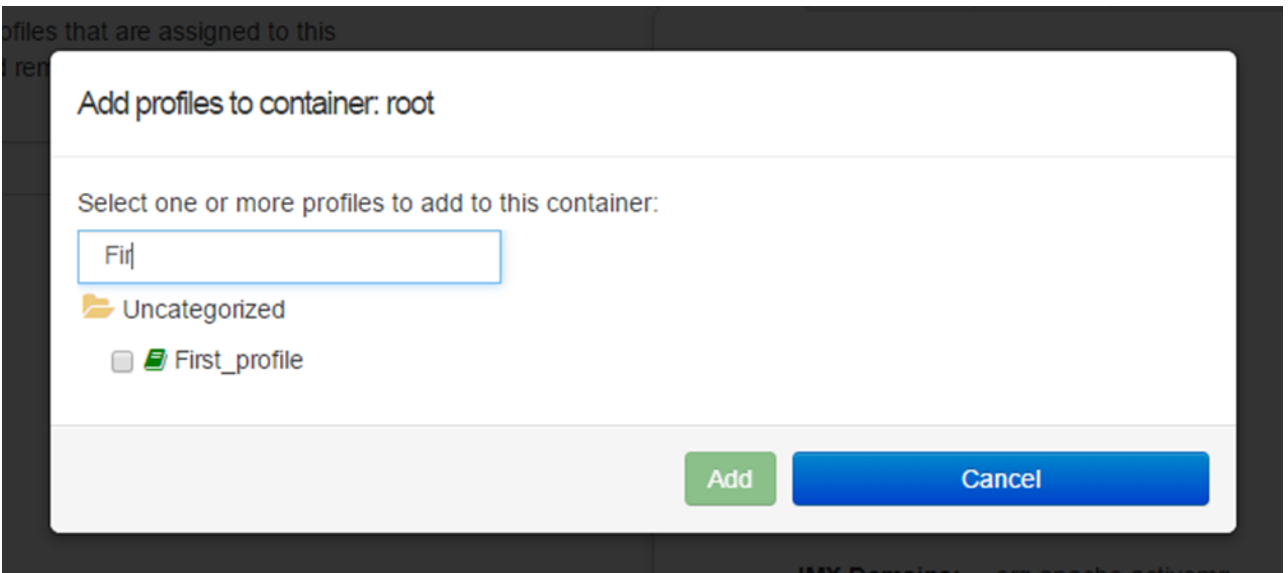
After this, the profile should be created.

Applying Profile to Container

Runtime → Containers → root (select the container you want)



Click **Add** which will lead to a pop-up box. Search for the profile you want and then again click **Add**.



The profile should be shown in the list as shown in the following screenshot.

RED HAT JBOSS FUSE Management Console

Runtime Wiki Dashboard Health

Containers Profiles Manage MQ APIs EIPs Registry Map

Container: root

Associated Profiles

This is the current set of profiles that are assigned to this container. Use the add and remove buttons to manage this list.

Filter... **+ Add** **x Remove**

- Uncategorized**
 - fabric
 - First_profile
- Jboss / Fuse**
 - full

Deploying a Bundle

To deploy a bundle, use the following path:

Runtime → Containers → root (select the container you want) → First_profile (select profile)

- Uncategorized**
 - fabric
 - First_profile
- Jboss / Fuse**
 - full

Click the Bundles tab. Set the bundle path in the following format and then click **+**.

mvn:group.id/artifact.id/version

For example: **mvn:com.tutorialpoint.app/camel-firt-app/1.0-SNAPSHOT**

A bundle will be added to the profile and will be deployed on all the containers to which the profile is assigned.

Un-deploying a Bundle

To un-deploy a bundle, use the following path:

Runtime → Containers → root (select container you want) → First_profile (select profile)

Click the Bundles tab and search for the bundle that you want to delete and then click on **X**. The Bundle will be deleted from all the containers to which the profile is applied.

Features (0)	Feature Repositories (0)	Bundles (1)	FABs (0)
--------------	--------------------------	--------------------	----------

✘ mvn:com.tutorialpoint.app/camel-firt-app/1.0-SNAPSHOT

Add: +

example: "mvn:group.id/artifact.id/version"

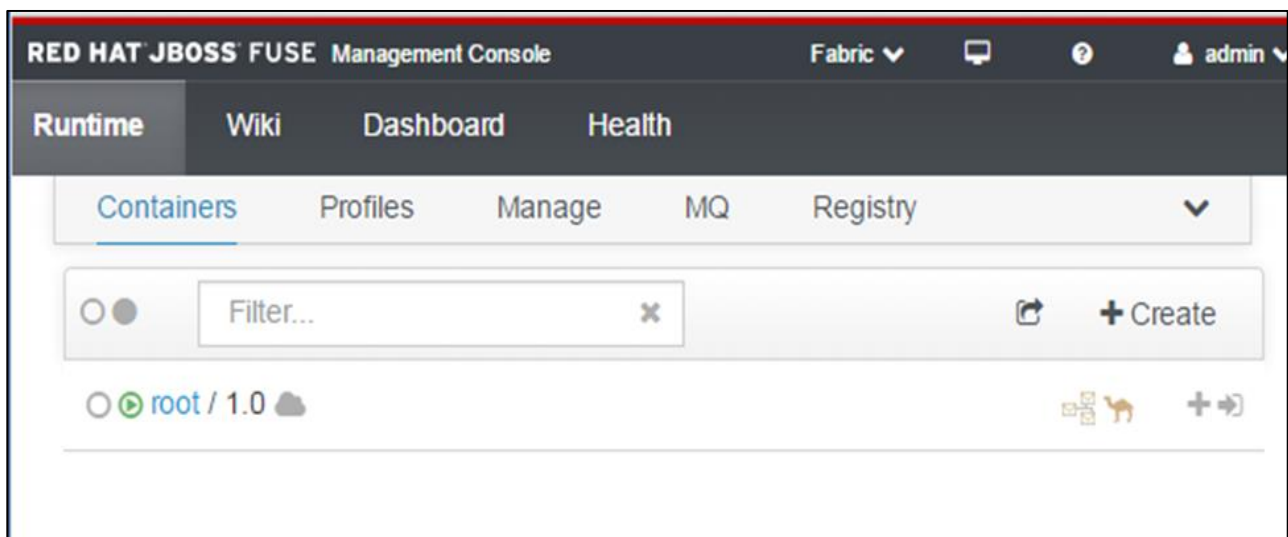
11. Child Container

A Child Container provides the easiest way to manage the increasing load. When the system is experiencing sudden load in traffic and a single container is not able to cope up with the load, we can easily create a set of child containers and distribute the load among them, rather than creating a complete new container.

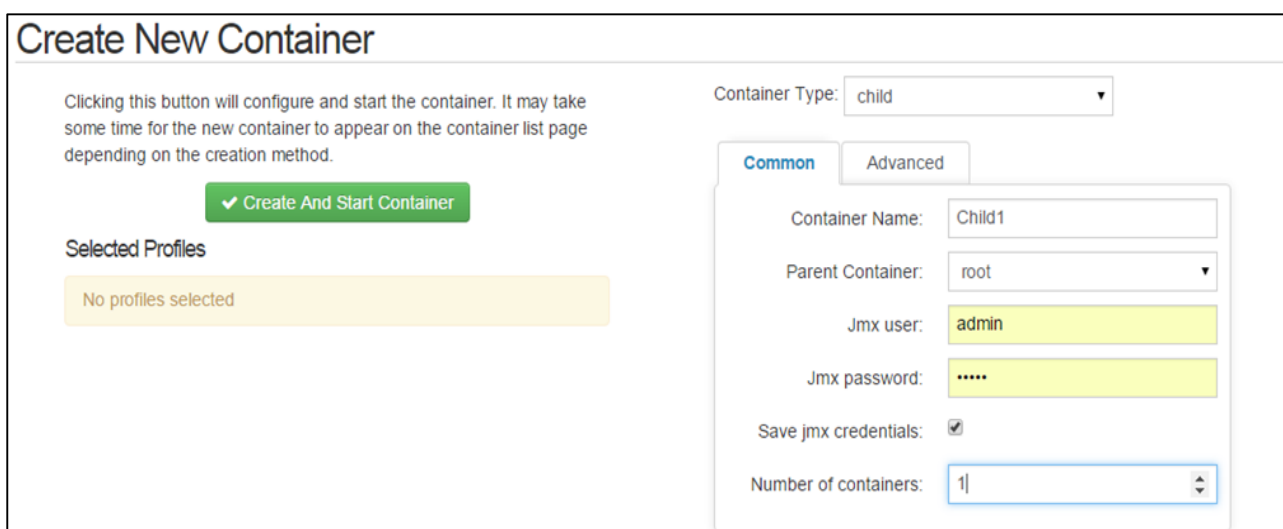
Creating a Child Container

Login to FMC using **localhost:8181**

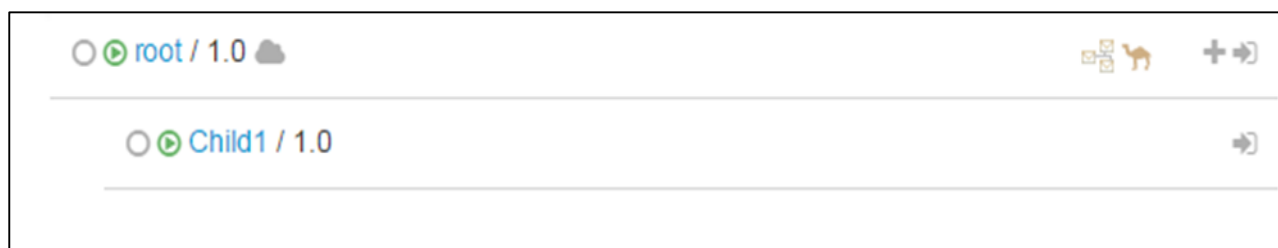
Now, follow the path: Runtime → container → +Create (button on right hand side)



Enter details like child name, parent container Number of instances etc.

The screenshot shows the "Create New Container" form. On the left, there's a green button "Create And Start Container" and a section for "Selected Profiles" which currently shows "No profiles selected". On the right, the "Container Type" is set to "child". The form is divided into "Common" and "Advanced" tabs. Under "Common", the following fields are visible: "Container Name" (Child1), "Parent Container" (root), "Jmx user" (admin), "Jmx password" (masked with dots), "Save jmx credentials" (checked), and "Number of containers" (1).

Click **Create And Start Container**



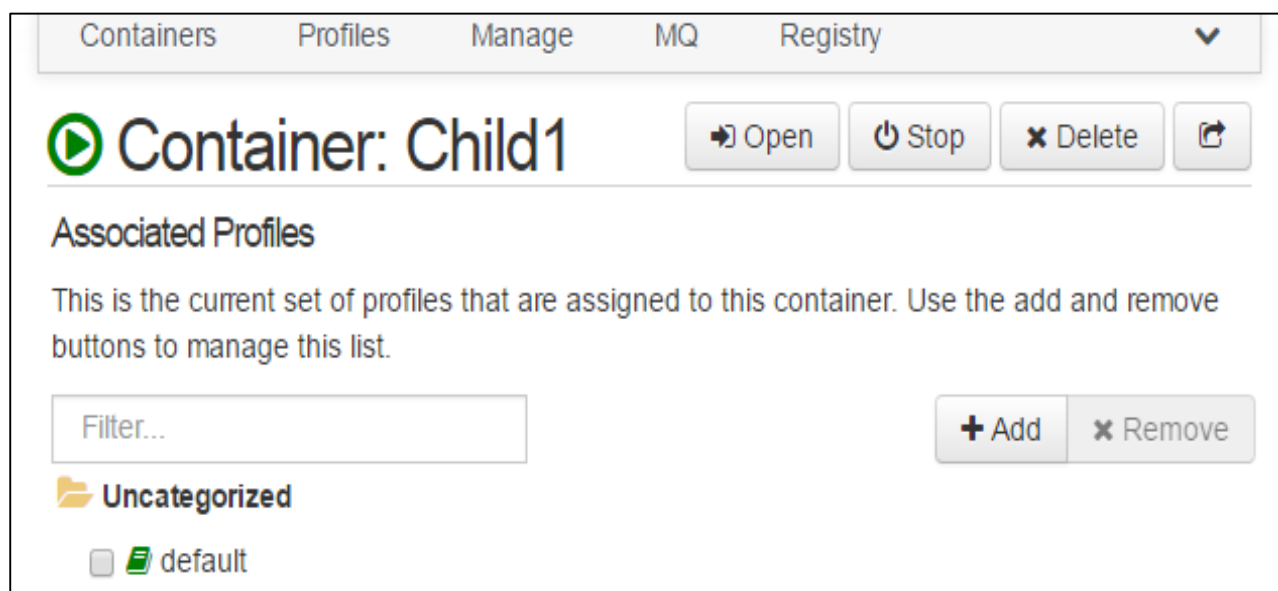
Managing a Child Container

A Child container acts as a normal container only.

Stopping a Child Container

To stop a child container, follow the path: Runtime → Container → Child1

Click Stop to stop the Child Container.



Starting a Child Container

To start a child container, follow the path: Runtime → Container → Child1

Click Start to start the child container.

Runtime Wiki Dashboard Health

Containers Profiles Manage MQ Registry Map APIs

Container: Child1

Start Delete

Associated Profiles

This is the current set of profiles that are assigned to this container. Use the add and remove buttons to manage this list.

Filter... + Add x Remove

Uncategorized

- default

12. Fuse – Issues and Solutions

In this chapter, we will discuss a few known issues that you might encounter while working with Fuse. We will also discuss how you can get over these issues.

Code Changes are not Reflected

Connect to Fuse instance using a client script. Search the bundle for which you are facing an issue, using the following command.

```
JBossFuse:karaf@root> list|grep <Bundle Description>
For Example:
JBossFuse:karaf@root> list|grep Camel
[ 255] [Active    ] [          ] [        ] [ 60] Fabric8 :: Camel Component
(1.0.0.redhat-379)
[ 266] [Active    ] [          ] [Started] [ 60] A Camel Spring Route
(1.0.0.SNAPSHOT)
```

Note: Bundle ID for the bundle from output of above command and use below command.

```
JBossFuse:karaf@root>update <bundle id>
JBossFuse:karaf@root>update 266
```

Bundle not Being Downloaded

It may happen because of the following two reasons –

- Maven repository not specified
- Bundle not present in repository

Maven Repository not Specified

Maven is a built tool used for building Fuse artifacts. Fuse first searches in Maven local repository for artifacts, when we issue command to install artifact. So we must let Fuse know where Maven is installed and path of Mavens local repository.

Edit \$FUSE_INSTALLATION_DIR/etc/**org.ops4j.paxurl.mvn.cfg**

Update the following two properties –

- org.ops4j.pax.url.mvn.settings=\$M2_HOME/conf /settings.xml
- org.ops4j.pax.url.mvn.localRepository=\$local_repo

Note: Please change \$local_repo with actual path of your local repository mentioned in Mavens settings.xml

Bundle not Present in Repository

If Maven settings are in place but still if you face issues while downloading the bundle, make sure bundles **JAR** is present at the correct location in Maven Repository.

For Example, if the following bundle is throwing errors while downloading –

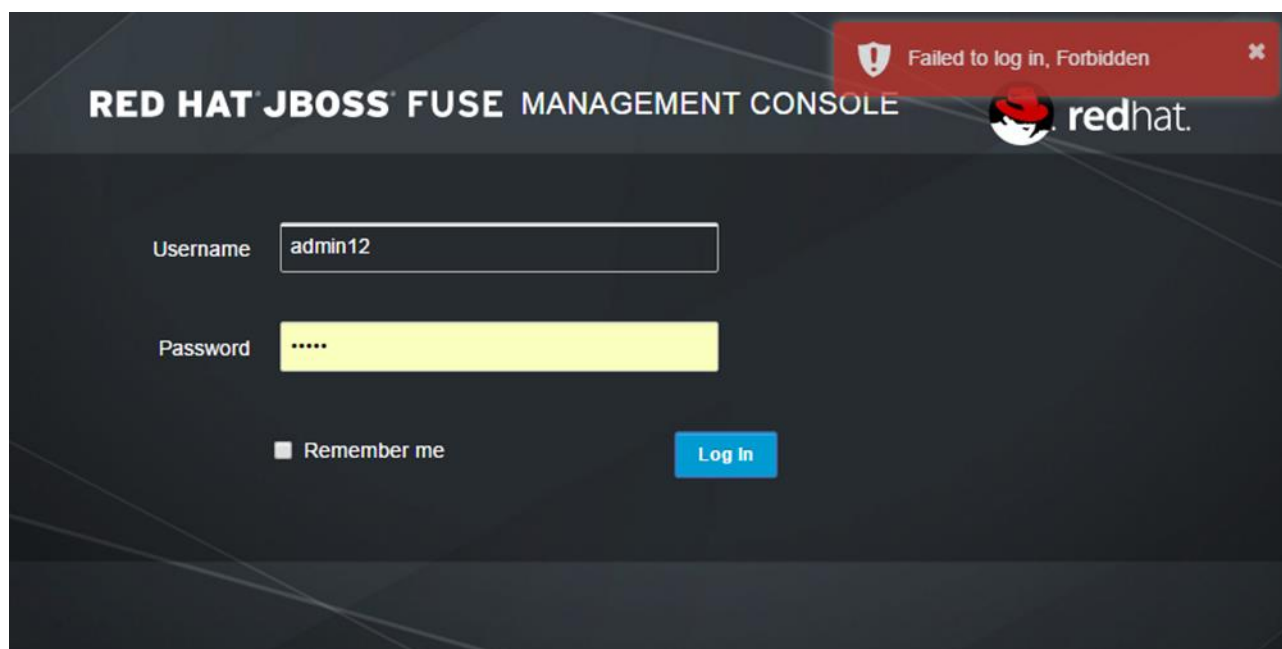
mvn:com.tutorialpoint.app/camel-first-app/1.0-SNAPSHOT

We have to check in `$M2_REPO/com/tutorialpoint/app/camel-first-app/1.0-SNAPSHOT` if actual JAR is present.

Note: `$M2_REPO` needs to be replaced with actual path of Maven repository we have Fuse configured to use.

Not Able to Login into FMC (Browser based GUI)

Users not Created – If you are getting the following UI but not able to login with a message saying “Failed to log in, Forbidden”.



Check whether you have added users in

`$FUSE_INSTALLATION_HOME/etc/users.properties`

The correct format to add users is –

```
Username=Password,Role
```

HAWTIO Port is Different

If you are not even able to get the UI at `localhost:8181` in the browser, check if you have mentioned correct port in URL.

`$FUSE_INSTALLATION_HOME/etc/org.ops4j.pax.web.cfg`

Edit the following property in the file to use port you want to access.

```
org.osgi.service.http.port=8181
```

AMQ Broker is not working

Make sure that the 61616 port is open and not being currently used by another port. If you want to change the default 61616 port for the same, you can edit it in **\$FUSE_INSTALLATION_HOME/etc/System.properties**

```
activemq.port = 61616
```