Racing Camel with JBoss BPM & Red Hat JBoss Fuse : Lab Guide

| Information	
Technology/Product	Red Hat JBoss BPM Suite 6.1 & Red Hat JBoss Fuse 6.1.1
Difficulty	3
Time	2 hrs
Prerequisites	Basic Java knowledge, ability to read English instructions.

Before we begin,

Let's install and create the fuse fabric.
- To start up JBoss Fuse, go to [project-root-directory]/summit-racing-camel-with-jboss-bpm-fuse/target/jboss-fuse-6.1.1.redhat-412/bin and run
  - ./fuse
- In console create Fuse Fabric by typing
  - `fabric:create --wait-for-provisioning`
- Stop your fuse by typing `exit` in console

Situation

We are an operating travel agency, our company has been ask to create an web service for hotel enquiry and hotel bookings to allow other applications, such as a web client, mobile application and most importantly to provide service to a customer's employee online travel booking process service.
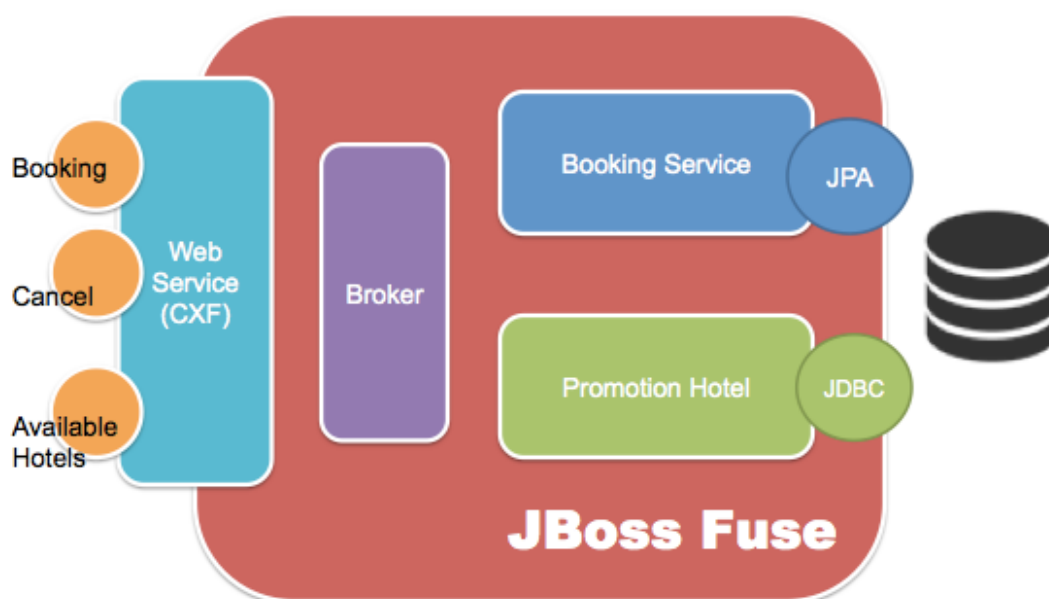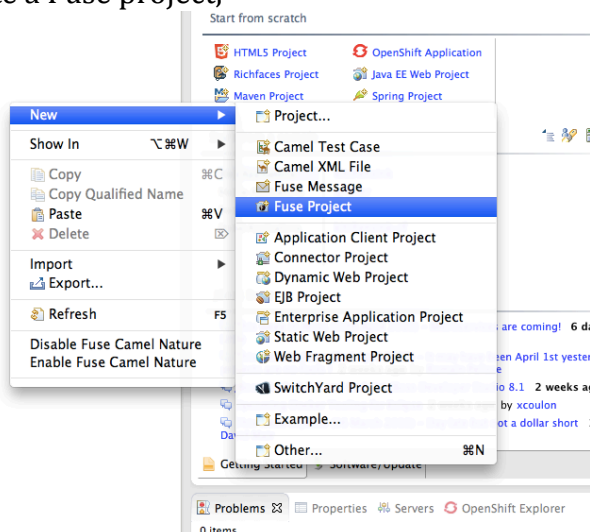
Promotion Hotels recommend - Retrieving Data from Database

This application lists all the available hotels to the client with condition like the location of the hotels. After retrieving all the possible hotel data from database, it'll either randomly pick one of them, or create a default hotel.
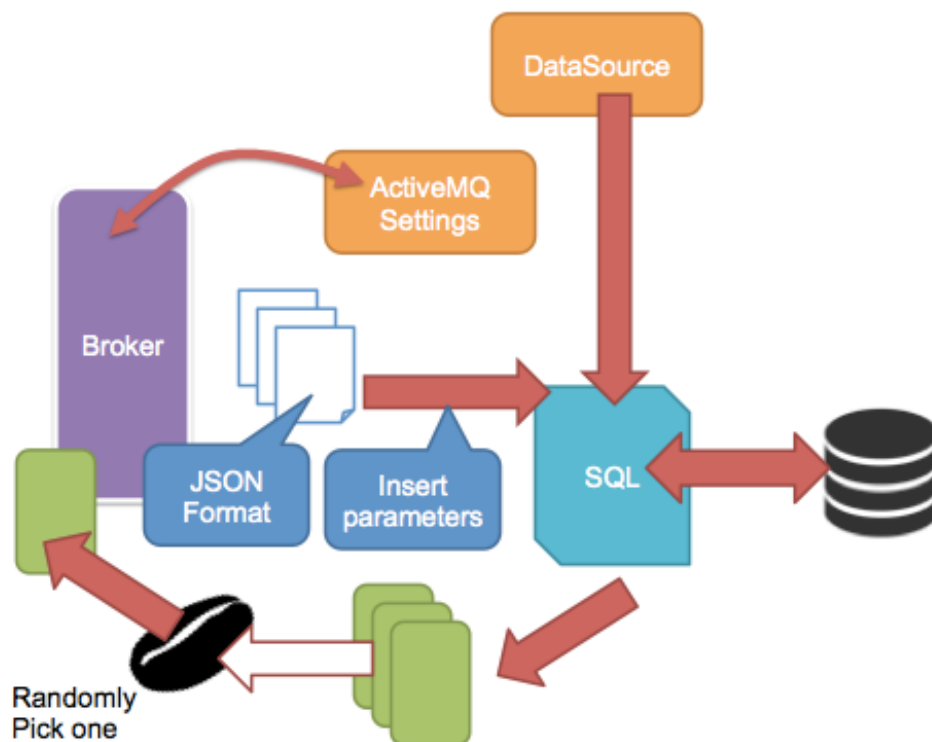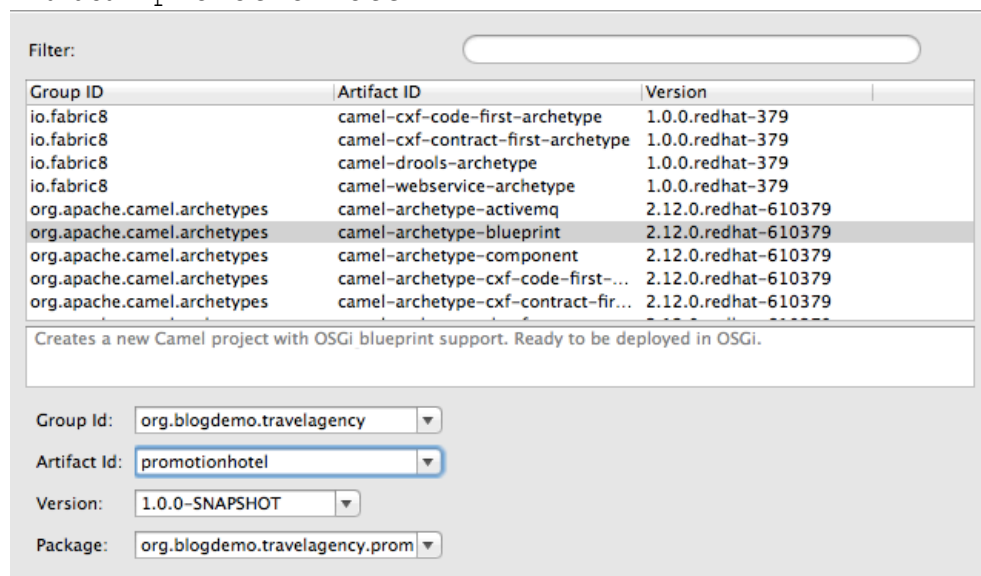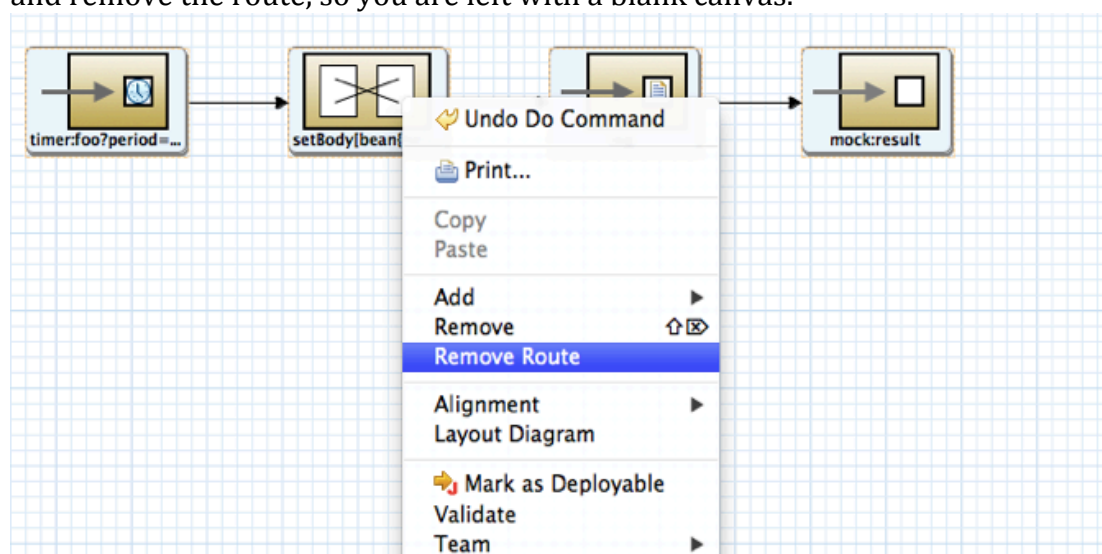

Instructions

Create a Fuse project,

Choose the blueprint archetype,
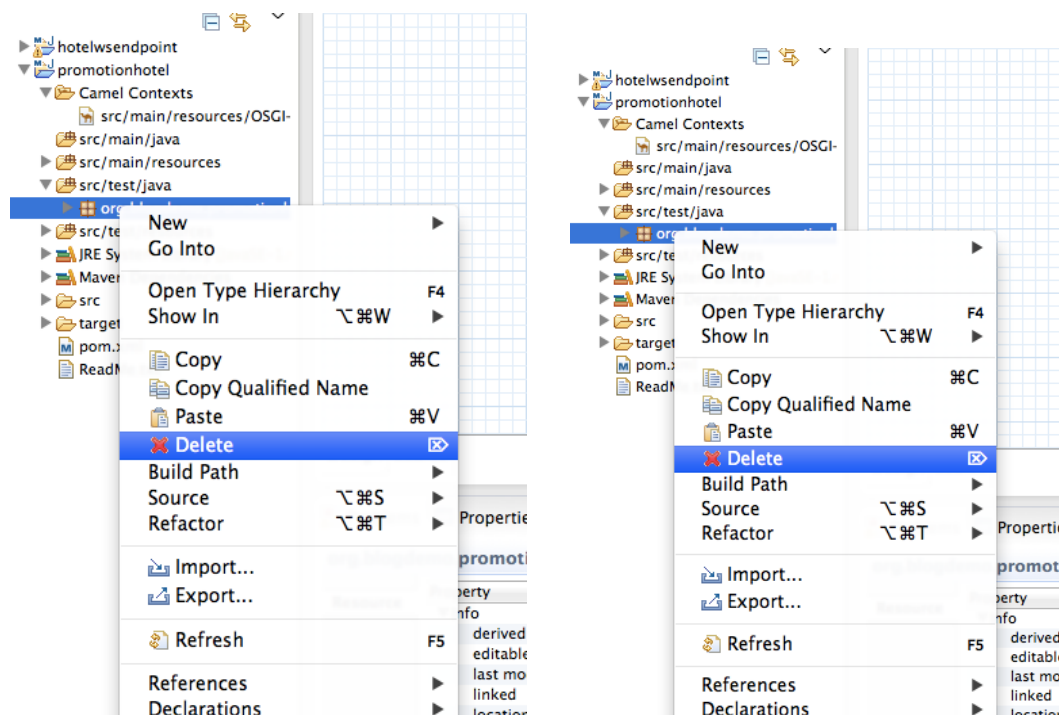GroupID: `org.blogdemo.travelagency`
ArtifactID:`promotionhotel`

Open up `/example/src/main/resources/OSGI-INF/blueprint/blueprint.xml`
and remove the route, so you are left with a blank canvas.

Remove the hellobean registry in the xml file. And delete all the java files.

```
<bean id="helloBean" class="com.mycompany.camel.blueprint.HelloBean">
    <property name="say" value="Hi from Camel"/>
</bean>
```

Remove everything under `src/java/*` and `src/test/*`

Now we have a brand new project to work with. First we want to make sure we have all the dependency needed in pom.xml. Add the following dependency in.


```xml
<!-- Database -->
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jdbc</artifactId>
  <version>2.12.0.redhat-610379</version>
</dependency>
<dependency>
  <groupId>commons-dbcp</groupId>
  <artifactId>commons-dbcp</artifactId>
  <version>1.4</version>
</dependency>
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <version>1.4.181</version>
</dependency>

<!--Messaging from and to AMQ -->
<dependency>
  <groupId>org.apache.activemq</groupId>
  <artifactId>activemq-camel</artifactId>
  <version>5.9.0.redhat-610379</version>
</dependency>

<!-- Message type conversion -->
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jackson</artifactId>
  <version>2.12.0.redhat-610379</version>
</dependency>
```
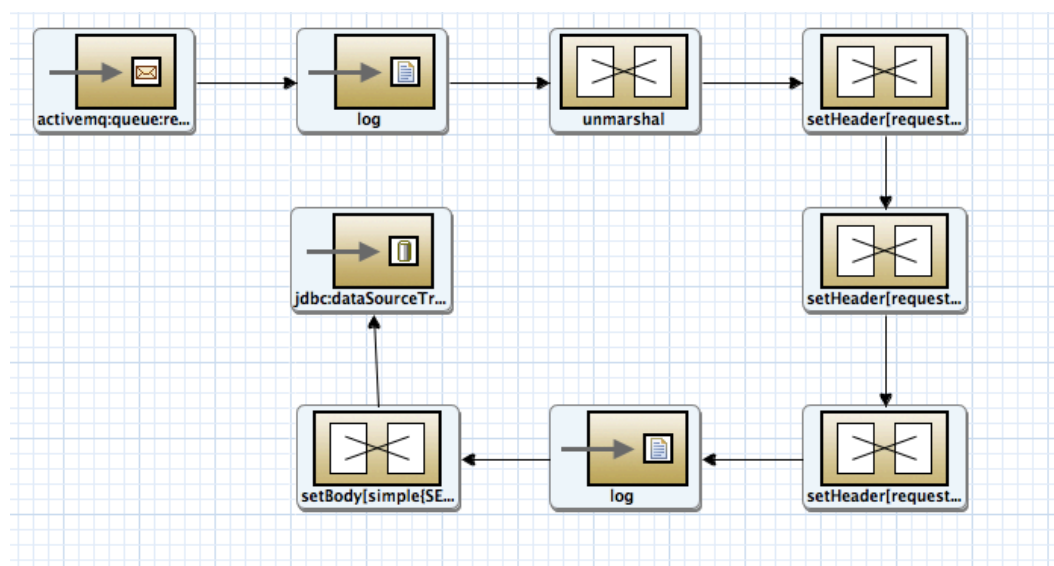
Go back Camel route file, `blueprint.xml`, add datasource setting

```xml
<bean id="dataSourceTravelAgency" class="org.apache.commons.dbcp.BasicDataSource">
  <property name="driverClassName" value="org.h2.Driver"/>
  <property name="url" value="jdbc:h2:file:~/h2/travelagency;AUTO_SERVER=TRUE" />
  <property name="username" value="sa"/>
  <property name="password" value=""/>
</bean>
```

Add activemq setting to connect to our messaging queue.

```xml
<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>
```

Create route to
1. Read request data from queue
2. Convert data from Json to a Map object
3. Set data to header later will use these in JDBC query
4. Add SQL query and call JDBC endpoint to retrieve data.

(When you save the route, it'll become a straight line)

- AMQ Endpoint
    - Uri: activemq:queue:requestHotel
- Log Endpoint
    - Uri: ${body}
- Unmarshal
    - Tab: json
    - Library: Jackson
    - Unmarshal Type Name: java.util.HashMap
- setHeader
    - HeaderName: requestCity
    - Language: simple
    - Expression: ${body[targetCity]}

- setHeader
  - HeaderName: requestStartDate
  - Language: simple
  - Expression ${body[startDate]}

- setHeader
  - HeaderName: requestEndDate
  - Language: simple
  - Expression: ${body[endDate]}
- setBody
  - Language: simple
  - Expression: SELECT * from avaliablehotels where hotelcity='${header.requestCity}
- Log Endpoint
  - Expression: requestCity ${headers.requestCity} requestStartDate ${headers.requestStartDate}
- JDBC Endpoint
  - Uri: jdbc:dataSourceTravelAgency

We have the data returned in a Hashmap format, but it's better to map them into a POJO, so create a custom bean to do this.
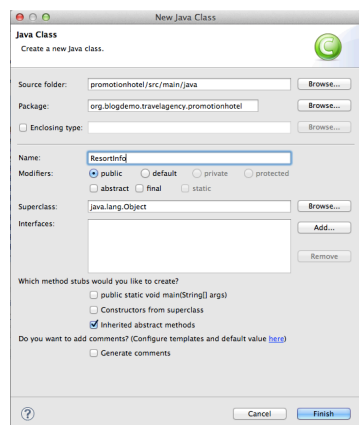Create the POJO java class
Package: org.blogdemo.travelagency.promotionhotel
Class name: ResortInfo

```java
package org.blogdemo.travelagency.promotionhotel;

import java.io.Serializable;
import java.math.BigDecimal;

public class ResortInfo implements Serializable{

  private static final long serialVersionUID =
    6313854994010205L;
  int hotelId;
  String hotelName;
  BigDecimal ratePerPerson;
  String hotelCity;
  String availableFrom;
  String availableTo;
  public int getHotelId() {return hotelId; }
  public void setHotelId(int hotelId) {
    this.hotelId = hotelId;
  }
  public String getHotelName() {return hotelName;}
  public void setHotelName(String hotelName) {
    this.hotelName = hotelName;
  }
  public BigDecimal getRatePerPerson() {
    return   ratePerPerson;
  }
  public void setRatePerPerson(BigDecimal ratePerPerson){
    this.ratePerPerson = ratePerPerson;
  }
  public String getHotelCity() {return hotelCity;}
  public void setHotelCity(String hotelCity) {
    this.hotelCity = hotelCity;
  }
  public String getAvailableFrom() {return availableFrom;}
  public void setAvailableFrom(String availableFrom){
    this.availableFrom = availableFrom;
  }
  public String getAvailableTo() {return availableTo;}
  public void setAvailableTo(String availableTo){
    this.availableTo = availableTo;
  }
```

Create a java class
Package: org.blogdemo.travelagency.promotionhotel
Class name: ResortDataConvertor

```java
package org.blogdemo.travelagency.promotionhotel;

import java.math.BigDecimal;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

public class ResortDataConvertor {

  public List<ResortInfo> processResultSet(List<Map<String, Object>> resultset){
            List<ResortInfo> avaliableResorts = new ArrayList<ResortInfo>();
            System.out.println("resultset:["+resultset.size()+"]");
    for(Map<String, Object> obj:resultset){
      ResortInfo resortInfo = new ResortInfo();
      resortInfo.setHotelId((Integer)obj.get("HOTELID"));
      resortInfo.setHotelName((String)obj.get("HOTELNAME"));
      resortInfo.setHotelCity((String)obj.get("HOTELCITY"));
      resortInfo.setAvailableFrom(new SimpleDateFormat("MM-dd-yyyy
HH:mm:ss").format(obj.get("AVAILABLEFROM")));
      resortInfo.setAvailableTo(new SimpleDateFormat("MM-dd-yyyy
HH:mm:ss").format(obj.get("AVAILABLETO")));
      resortInfo.setRatePerPerson((BigDecimal)obj.get("RATEPERPERSON"));
      avaliableResorts.add(resortInfo);
    }
   return avaliableResorts;
  }
}
```

Because of the restriction of other systems, we can only return one hotel at a
time, create the java class that randomly choose one hotel, and also create a
default one if no criteria is matched in database.
Package: org.blogdemo.travelagency.promotionhotel
Class name: ResortDataConvertor

```java
package org.blogdemo.travelagency.promotionhotel;

import java.math.BigDecimal;
import java.util.List;
import java.util.Random;
import org.apache.camel.Exchange;
import org.apache.camel.Processor;

public class RandomHotelBean implements Processor{
  @Override
  public void process(Exchange exchange) throws Exception {
    List<ResortInfo> resorts = (List<ResortInfo>)exchange.getIn().getBody();
    ResortInfo resort;
    if(resorts == null || resorts.size() == 0){
      resort = new ResortInfo();
      resort.setHotelId(201);
      resort.setHotelName("The Grand Default Hotel");
      resort.setHotelCity((String)exchange.getIn().getHeader("requestCity"));
      resort.setAvailableFrom((String)exchange.getIn().getHeader("requestStartDate"));
      resort.setAvailableTo((String)exchange.getIn().getHeader("requestEndDate"));
      resort.setRatePerPerson(new BigDecimal("109.99"));
    }else{
      Random rand = new Random();
      int  n = rand.nextInt(resorts.size());
      resort = resorts.get(n);
    }
    exchange.getOut().setBody(resort);
    }
}
```

Registry the two beans we created in camel context (in your blueprint file)

```
        <property name="userName" value="admin"/>
        <property name="password" value="admin"/>
    </bean>

    <bean id="dataProcessor" class="org.blogdemo.travelagency.promotionhotel.ResortDataConvertor" />
    <bean id="randomHotel" class="org.blogdemo.travelagency.promotionhotel.RandomHotelBean" />

  <camelContext trace="false" id="blueprintContext" xmlns="http://camel.apache.org/schema/blueprint">
    <route id="promtionHotelRoute">
        <from uri="activemq:queue:requestHotel"/>
        <log message="${body}"/>
```
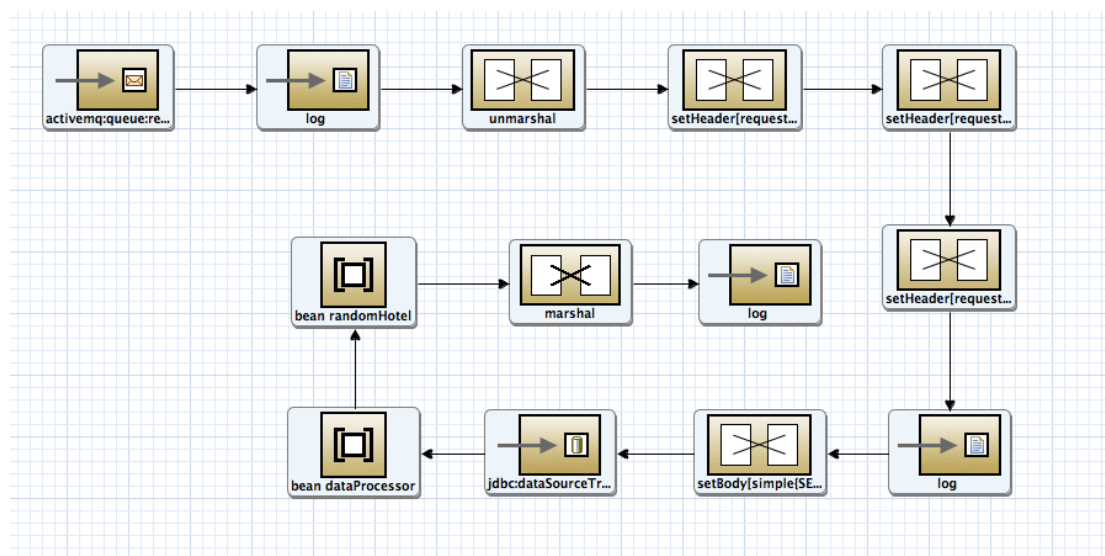
```xml
<bean id="dataProcessor"
class="org.blogdemo.travelagency.promotionhotel.ResortDataConvertor" />
<bean id="randomHotel"
class="org.blogdemo.travelagency.promotionhotel.RandomHotelBean" />
```

And we can start adding the processes that deal with the JDBC result returned. Add the last three steps in the end of your route.

- Bean
  - Bean Name: dataProcessor
  - Method: processResultSet
- Bean:
  - Bean Name: randomHotel
- Marshal
  - Tab: json
  - Library: Jackson
- Log Endpoint
  - Message: ${body}

Deploy project

Start up JBoss Fuse by going to [project-root-directory]/summit-racing-camel-with-jboss-bpm-fuse/target/jboss-fuse-6.1.1.redhat-412/bin and run,
./fuse

Go back to your JBDS. Add the fabric8 plugin in pom.xml

```xml
<!-- For Fabric8 deployment -->
  <plugin>
    <groupId>io.fabric8</groupId>
    <artifactId>fabric8-maven-plugin</artifactId>
    <version>1.0.0.redhat-412</version>
  </plugin>
```

Also the deployment configurations in pom.xml

```
Cd dbcp/1.4</fabric8.bundles>
<fabric8.features>camel-jdbc activemq-camel camel-jackson</fabric8.features>
```

```xml
M promotionhotel/pom.xml ⊠
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>org.blogdemo</groupId>
    <artifactId>promotionhotel</artifactId>
    <packaging>bundle</packaging>
    <version>1.0-SNAPSHOT</version>

    <name>Travel Agency Promotion Hotel service (JDBC)</name>
    <url>http://www.myorganization.org</url>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <fabric8.parentProfiles>feature-camel mq-client</fabric8.parentProfiles>
        <fabric8.profile>demo-travelagency-promotionhotel</fabric8.profile>
        <fabric8.bundles>wrap:mvn:com.h2database/h2/1.4.181 wrap:mvn:commons-dbcp/commons-dbcp/1.4</fabric8.bundles>
        <fabric8.features>camel-jdbc activemq-camel camel-jackson</fabric8.features>
    </properties>
```

Add dynamic import to bundle maven configuration

```xml
<DynamicImport-Package>
    org.apache.commons.dbcp, *
</DynamicImport-Package>
```

```
</plugin>

<!-- to generate the MANIFEST-FILE of the bundle -->
<plugin>
    <groupId>org.apache.felix</groupId>
    <artifactId>maven-bundle-plugin</artifactId>
    <version>2.3.7</version>
    <extensions>true</extensions>
    <configuration>
        <instructions>
            <Bundle-SymbolicName>promotionhotel</Bundle-SymbolicName>
            <DynamicImport-Package>org.apache.commons.dbcp, *</DynamicImport-Package>
            <Private-Package>org.blogdemo.promotionhotel.*</Private-Package>
            <Import-Package>*</Import-Package>
        </instructions>
    </configuration>
</plugin>

<!-- to run the example using mvn camel:run -->
<plugin>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-maven-plugin</artifactId>
    <version>2.12.0.redhat-610379</version>
    <configuration>
        <useBlueprint>true</useBlueprint>
    </configuration>
</plugin>

<!-- For Fabric8 deployment -->
<plugin>
    <groupId>io.fabric8</groupId>
    <artifactId>fabric8-maven-plugin</artifactId>
    <version>${fabric8.version}</version>
</plugin>

</plugins>
```
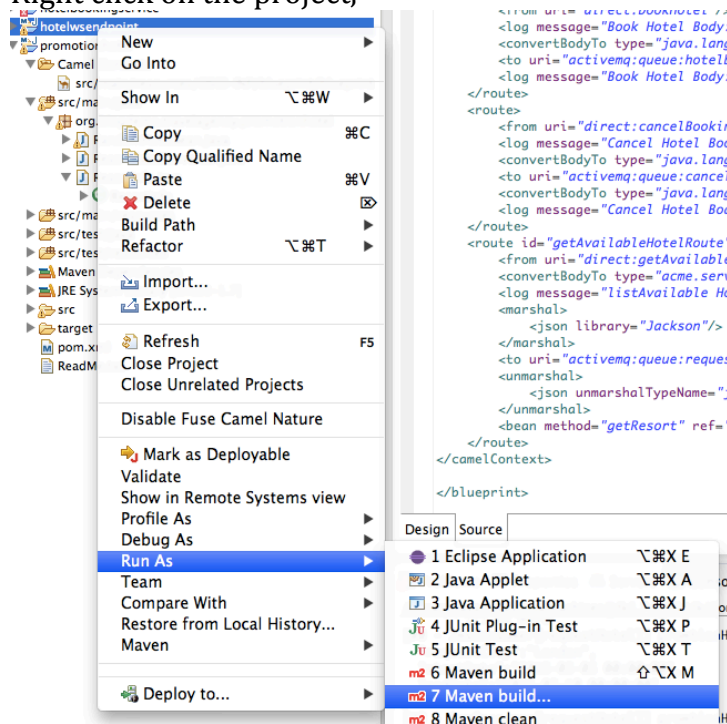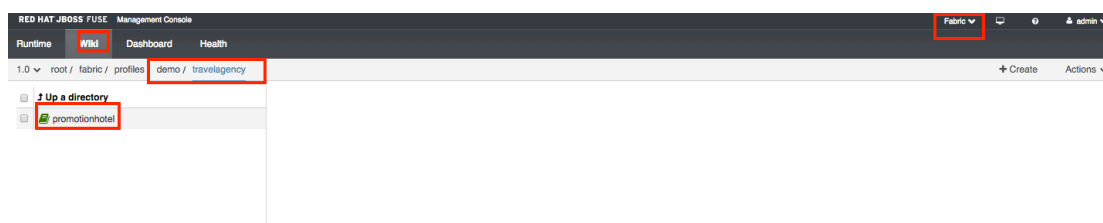
Right click on the project,

Type in "fabric8:deploy" as goal,
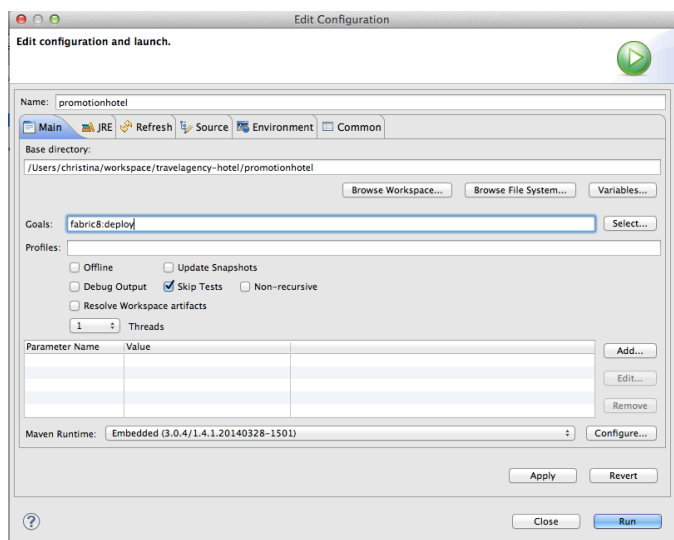
Wait a few seconds. You will be able to see it in your fabric console.

Hotel Service – Contract first Web Service (CXF)

Lastly, we are going to create web service endpoint for different clients, these clients services are already up and running for a while. And here is how it works, after receiving hotel request from client, the web service will be dispatched to messaging queues. And because the result needs to go back to the client, we need to make sure the request comes back, there for we need to set the JMS to in-out to make sure it gets the return message back.
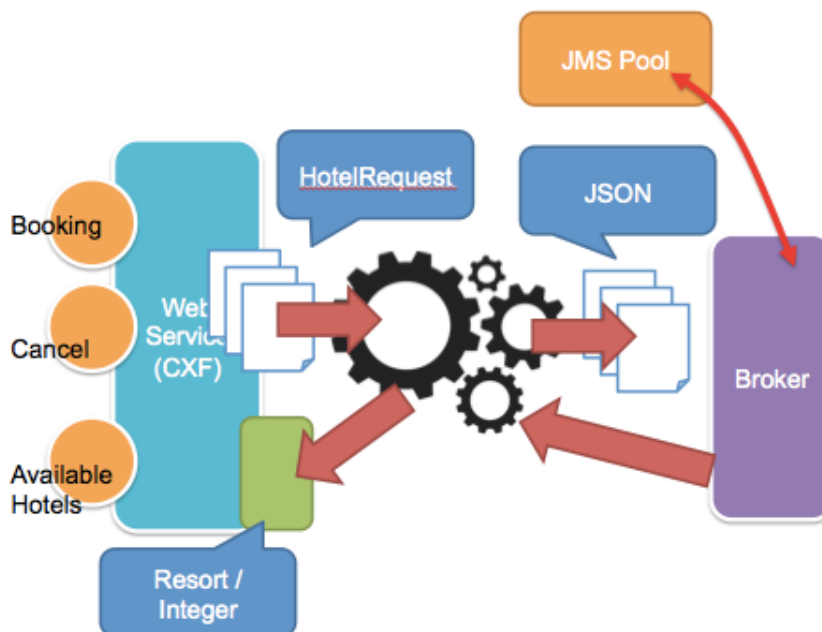
Instructions

Create another blueprint project like the first one
GroupID: `org.blogdemo.travelagency`
ArtifactID: `hotelwsendpoint`
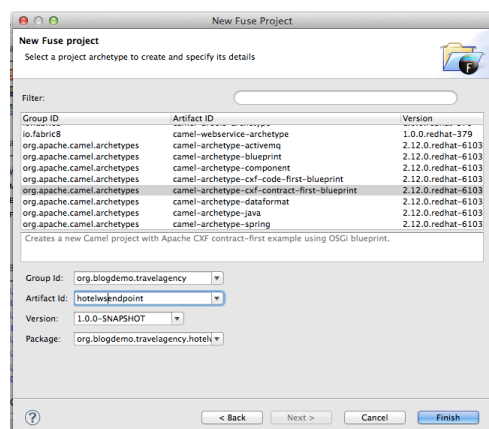
Delete the route, hello bean registry in blueprint.xml and delete everything under `src/java/*` and `src/test/*` (Go back to beginning of the first promotion hotel project if you are not sure what to do)

Setup the dependencies in pom.xml

```xml
<!-- For Web Service with CXF -->
<dependency>
   <groupId>org.apache.camel</groupId>
   <artifactId>camel-cxf</artifactId>
   <version>2.12.0.redhat-610379</version>
   <exclusions>
     <exclusion>
        <groupId>asm</groupId>
        <artifactId>asm</artifactId>
     </exclusion>
   </exclusions>
</dependency>

    <dependency>
        <groupId>org.apache.cxf</groupId>
        <artifactId>cxf-rt-frontend-jaxrs</artifactId>
        <version>2.7.0.redhat-610379</version>
    </dependency>

    <dependency>
        <groupId>org.apache.cxf</groupId>
        <artifactId>cxf-rt-transports-http-jetty</artifactId>
        <version>2.7.0.redhat-610379</version>
    </dependency>

    <!-- Data transformation -->
    <dependency>
      <groupId>org.apache.camel</groupId>
      <artifactId>camel-jackson</artifactId>
      <version>2.12.0.redhat-610379</version>
    </dependency>

    <!-- Messging with AMQ and JMS -->
    <dependency>
          <groupId>org.apache.activemq</groupId>
          <artifactId>activemq-camel</artifactId>
          <version>5.9.0.redhat-610379</version>
    </dependency>
    <dependency>
      <groupId>org.apache.camel</groupId>
      <artifactId>camel-jms</artifactId>
      <version>2.12.0.redhat-610379</version>
    </dependency>
```

And because we already have an webservice wsdl, we are going to use it to generate all the classes we need, so add the plugin to your pom.xml too.

```xml
<!-- to generate source code from the wsdl file -->
<plugin>
   <groupId>org.apache.cxf</groupId>
   <artifactId>cxf-codegen-plugin</artifactId>
   <version>2.7.0.redhat-610379</version>
   <executions>
     <execution>
        <id>generate-sources</id>
        <phase>generate-sources</phase>
        <configuration>
          <sourceRoot>${basedir}/target/generated/src/main/java</sourceRoot>
          <wsdlOptions>
```
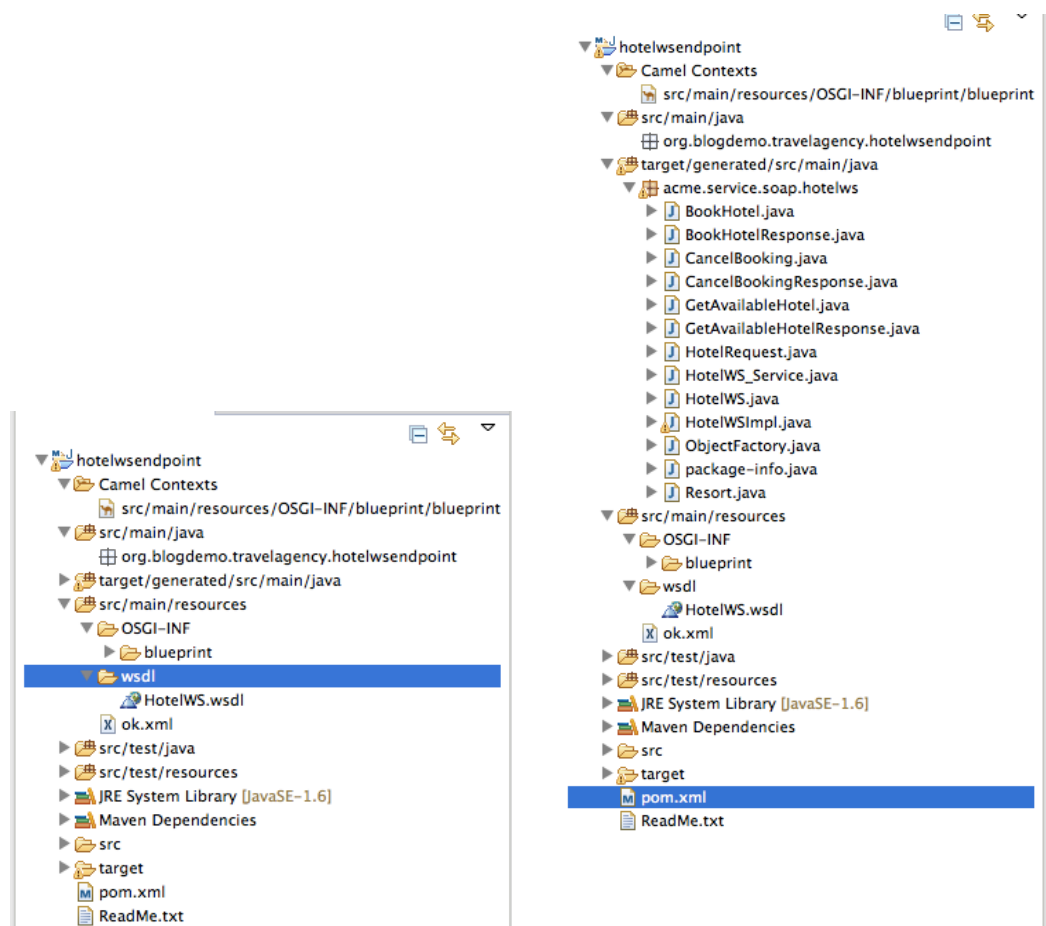
```
            <wsdlOption>
               <wsdl>${basedir}/src/main/resources/wsdl/HotelWS.wsdl</wsdl>
               <extraargs>
                  <extraarg>-impl</extraarg>
               </extraargs>
            </wsdlOption>
         </wsdlOptions>
      </configuration>
      <goals>
         <goal>wsdl2java</goal>
      </goals>
   </execution>
  </executions>
</plugin>
```

As you can see, we need to point the plugin to a wsdl, so place our Hotews.wsdl in src/main/resources/wsdl and click save, you should see all the classes generated.

Next up, all we have to do, is tell our Camel CXF component where the service address is going to locate, it's service class and the WSDL file location.

```
<cxf:cxfEndpoint id="hotelEndpoint"
                 address="/acme-hotel-service-2.0"
                 serviceClass="acme.service.soap.hotelws.HotelWS"
                 wsdlURL="wsdl/HotelWS.wsdl"/>
```

```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:cxf="http://camel.apache.org/schema/blueprint/cxf"
           xsi:schemaLocation="
              http://www.osgi.org/xmlns/blueprint/v1.0.0 http://www.osgi.org/xmlns/blueprint/v1.0.0
              http://camel.apache.org/schema/blueprint/cxf http://camel.apache.org/schema/blueprint
              http://camel.apache.org/schema/blueprint http://camel.apache.org/schema/blueprint/cam


  <cxf:cxfEndpoint id="hotelEndpoint"
                   address="/acme-hotel-service-2.0/"
                   serviceClass="acme.service.soap.hotelws.HotelWS"
                   wsdlURL="wsdl/HotelWS.wsdl"/>

  <camelContext xmlns="http://camel.apache.org/schema/blueprint">
     <route/>
  </camelContext>

</blueprint>
```

Rather then just a simple connection to the messaging queue, this small application needs interact with messaging a lot, so we are going to create a pool to the broker

```xml
<bean id="jmsConnectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">
       <property name="brokerURL" value="tcp://localhost:61616"/>
       <property name="userName" value="admin"/>
       <property name="password" value="admin"/>
</bean>
<bean id="pooledConnectionFactory"
class="org.apache.activemq.pool.PooledConnectionFactory" init-method="start" destroy-
method="stop">
       <property name="maxConnections" value="2" />
       <property name="connectionFactory" ref="jmsConnectionFactory" />
</bean>
<bean id="jmsConfig" class="org.apache.camel.component.jms.JmsConfiguration">
       <property name="connectionFactory" ref="pooledConnectionFactory"/>
       <property name="concurrentConsumers" value="2"/>
</bean>
<bean id="activemq" class="org.apache.activemq.camel.component.ActiveMQComponent">
       <property name="configuration" ref="jmsConfig"/>
</bean>
```

Take a look at the WSDL, here you see it has three services, with different inputs, so we are going to implement these services.
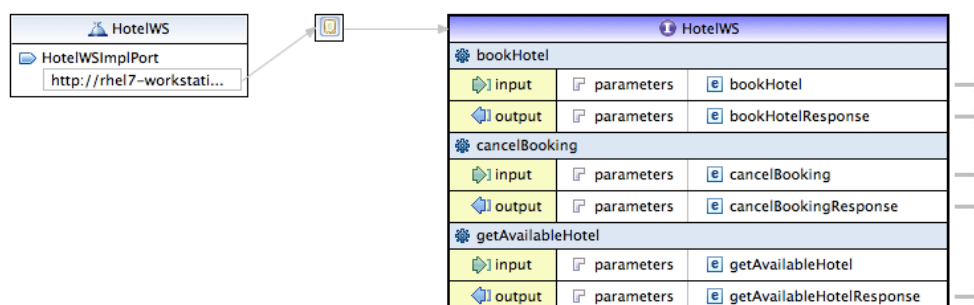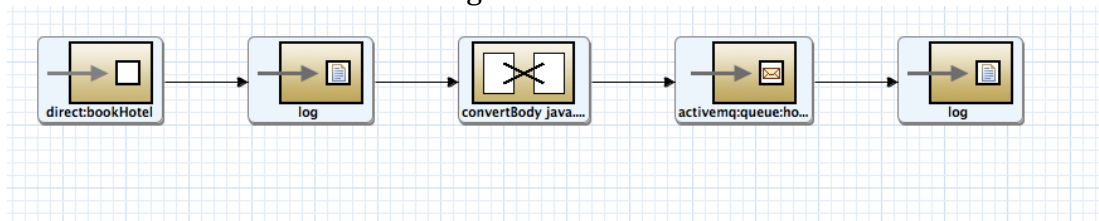
The first route is the universal endpoint, where it will take in the entire request, and dispatch them to the routes that actually implements the web service.
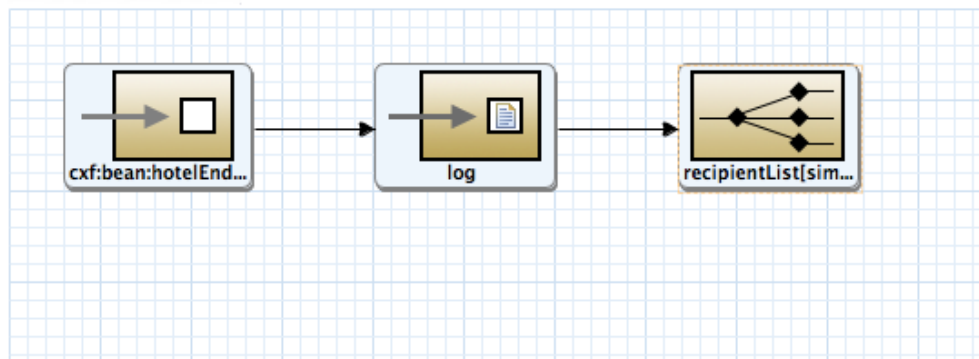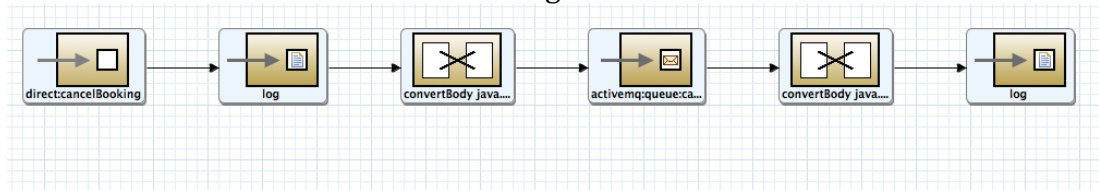

- CXF endpoint
  - o Uri: cxf:bean:hotelEndpoint
- Log Endpoint
  - o Message: ${header.operationName}
- RecipientList
  - o Expression: direct:${header.operationName}
  - o Language: simple

Create the route to handle booking

- Direct endpoint
  - o Uri: direct:bookHotel
- Log Endpoint
  - o Message: Book Hotel Body:[${body}]
- ConvertBodyTo
  - o Type: java.lang.String
- ActiveMQ Endpoint
  - o Uri: activemq:queue:hotelbooking
  - o Pattrn:InOut
- Log Endpoint
  - o Message: Book Hotel Body:[${body}]

Create the route to handle canel booking

- Direct endpoint
  - o Uri: direct:cancelBooking

- Log Endpoint
  - Message: Cancel Hotel Booking Body:[${body}]
- ConvertBodyTo
  - Type: java.lang.String
- ActiveMQ Endpoint
  - Uri: activemq:queue:cancelhotelbooking
  - Pattrn:InOut
- ConvertBodyTo
  - Type: java.lang.Integer
- Log Endpoint
  - Message: Cancel Hotel Body:[${body}]

The last service requires us to return a specific type of object, so we need to process with our custom code.

Create a java class
Package: org.blogdemo.travelagency.hotelwsendpoint
Class name: ListHotelBean
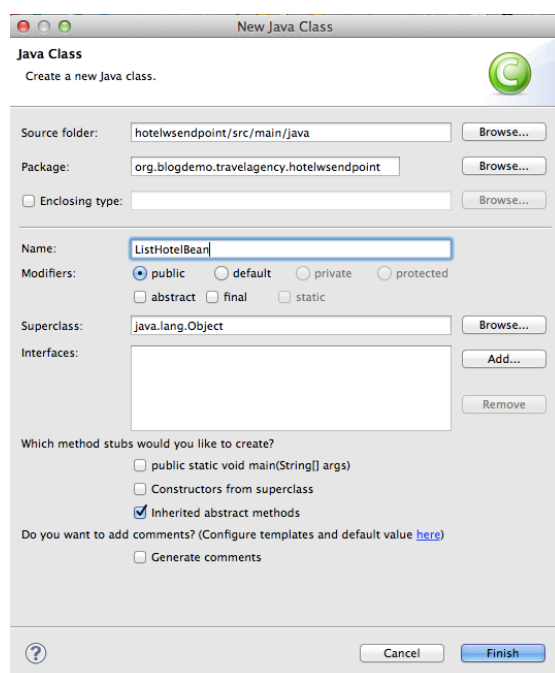

```java
package org.blogdemo.travelagency.hotelwsendpoint;

import java.util.Map;

import acme.service.soap.hotelws.Resort;

public class ListHotelBean {

    public Resort getResort(Map<String, Object> promotedResorts){
        Resort resort = new Resort();
        resort.setHotelId((Integer)promotedResorts.get("hotelId"));
        resort.setHotelName((String)promotedResorts.get("hotelName"));
        resort.setHotelCity((String)promotedResorts.get("hotelCity"));
        resort.setAvailableFrom((String)promotedResorts.get("availableFrom"));
```

```java
                resort.setAvailableTo((String)promotedResorts.get("availableTo"));
                resort.setRatePerPerson(((Double)promotedResorts.get("ratePerPerson")));

                return resort;
            }
    }
```
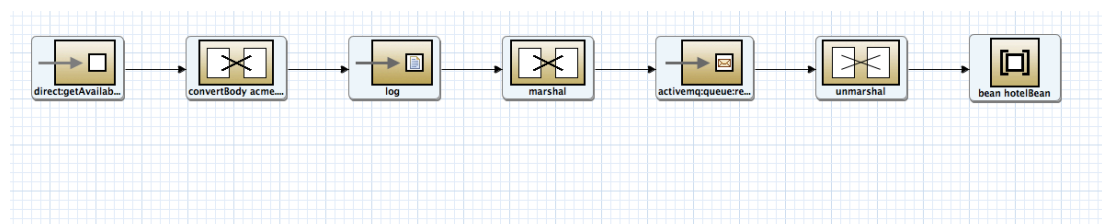
Register the bean in camel context:

```xml
<bean id="hotelBean"
      class="org.blogdemo.travelagency.hotelwsendpoint.ListHotelBean" />
```


- Direct Endpoint
    - o Uri: direct:getAvailableHotel
- ConvertBodyTo
    - o Type: acme.service.soap.hotelws.HotelRequest
- Log
    - o Message: listAvailable Hotels targetcity:${body.targetCity}
      startDate:${body.startDate} endDate{$body.endDate}
- Marshal
    - o Tab: json
    - o Library: Jackson
- ActiveMQ Endpoint
    - o Uri: activemq:queue:requestHotel
    - o Pattrn:InOut
- Unmarshal
    - o Tab: json
    - o Library: Jackson
    - o Unmarshal Type Name: java.util.HashMap
- Bean
    - o Bean Name: hotelBean
    - o Method: getResort

Deploy project

Add the fabric8 plugin in pom.xml

```xml
<!-- For Fabric8 deployment -->
  <plugin>
    <groupId>io.fabric8</groupId>
    <artifactId>fabric8-maven-plugin</artifactId>
    <version>1.0.0.redhat-412</version>
  </plugin>
```

Also the deployment configurations in pom.xml

```
<fabric8.parentProfiles>feature-camel mq-client</fabric8.parentProfiles>
<fabric8.profile>demo-travelagency-hotelwsendpoint</fabric8.profile>
<fabric8.features>camel-cxf activemq-camel camel-jackson</fabric8.features>
```
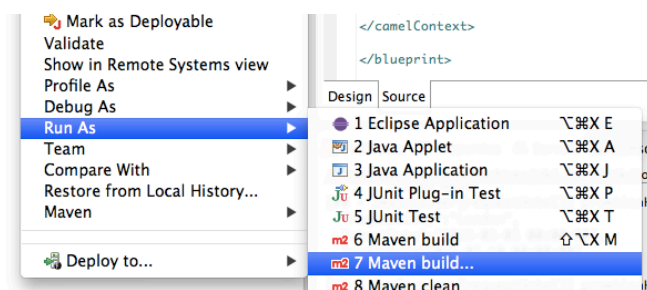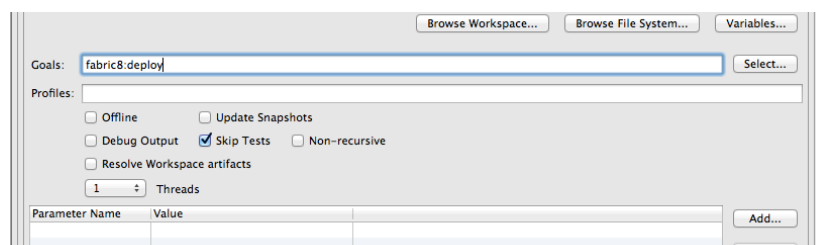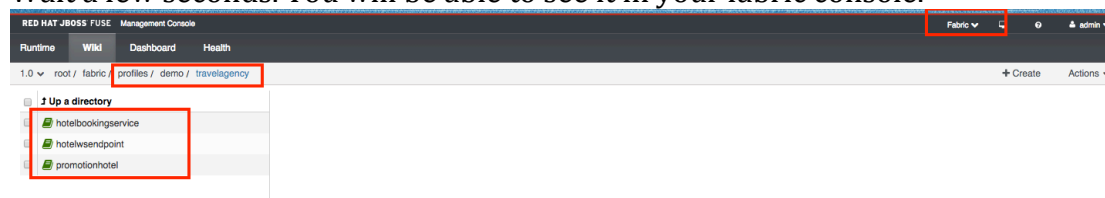
```
<packaging>bundle</packaging>
<version>1.0.0-SNAPSHOT</version>

<name>Travel Agency Hotel Contract first Web Endpoint Service (CXF)</name>
<url>http://www.myorganization.org</url>

<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <fabric8.parentProfiles>feature-camel mq-client</fabric8.parentProfiles>
    <fabric8.profile>demo-travelagency-hotelwsendpoint</fabric8.profile>
    <fabric8.features>camel-cxf activemq-camel camel-jackson</fabric8.features>
</properties>
```
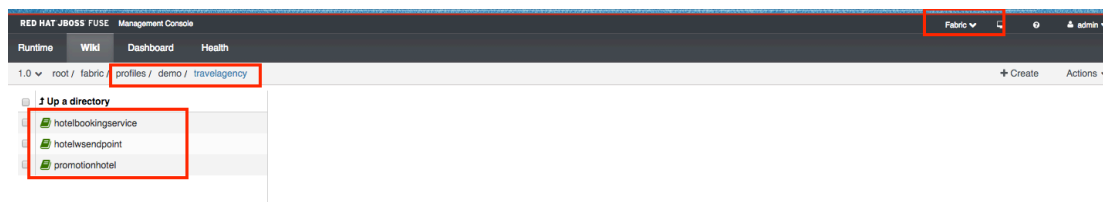
Right click on the project,

Type in "fabric8:deploy" as goal,

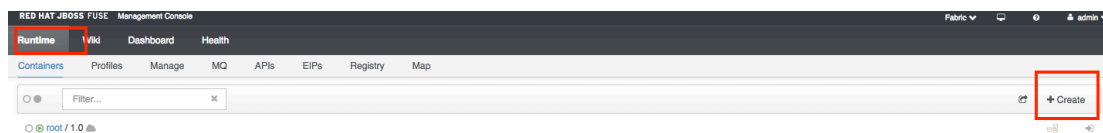Wait a few seconds. You will be able to see it in your fabric console.
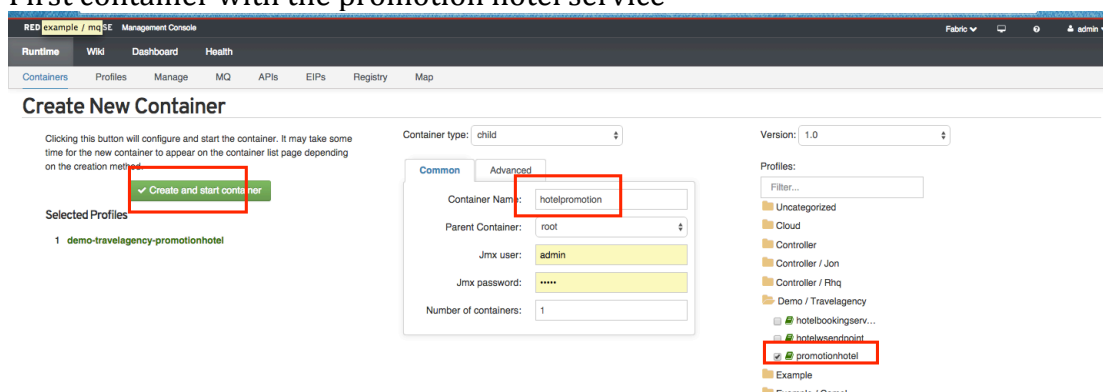
Running Application in Fuse Fabric
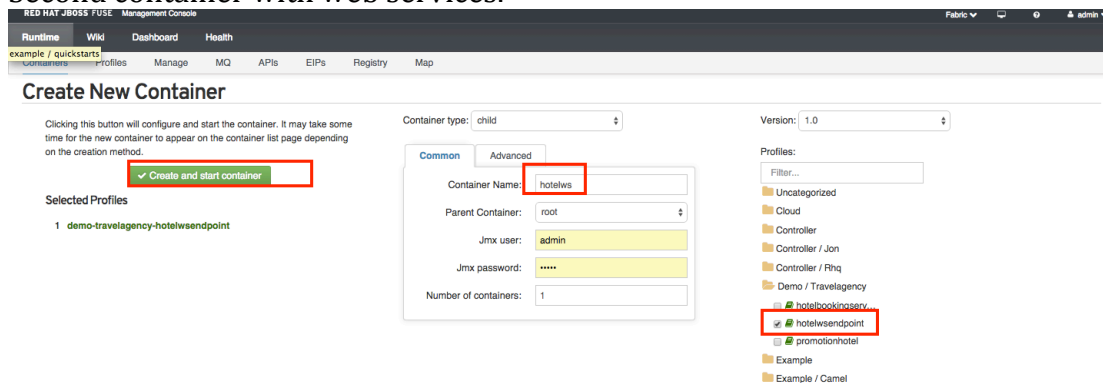By this time, you will have two profiles in your application.

Create two containers, each deploy the profile into it. Click on Runtime and on the right hand side, click on create.
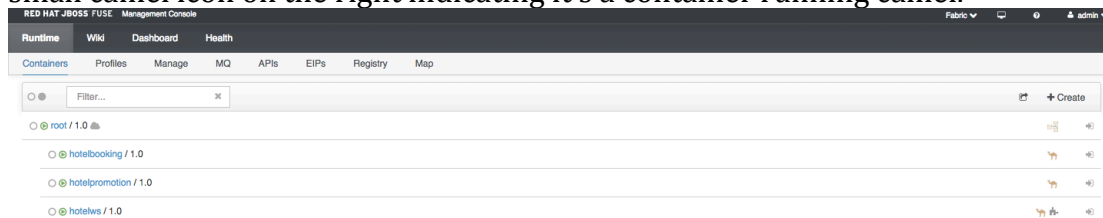

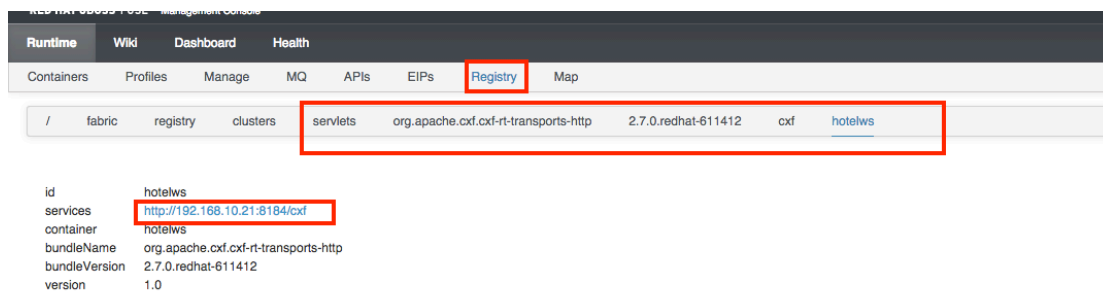First container with the promotion hotel service

Second container with web services.

After few minutes, you should see all of the containers are up and running, with a small camel icon on the right indicating it's a container running camel.

To see where the web service is registered, go to "Registry", and drill down the links `servlets/org.apache.cxf.cxf-rt-transports-http/2.7.0.redhat-611412/cxf/hotelws`

Now, let's install and start up the rest of the application, if you have shutdown JBoss Fuse previously, please start them again by going to the bin directory and run ./fuse

And install and start up the rest of the microservices by running the fusemicroservices.sh in the [project-root-directory]/summit-racing-camel-with-jboss-bpm-fuse directory.

That's all ☺