Red Hat Summit Labs

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.

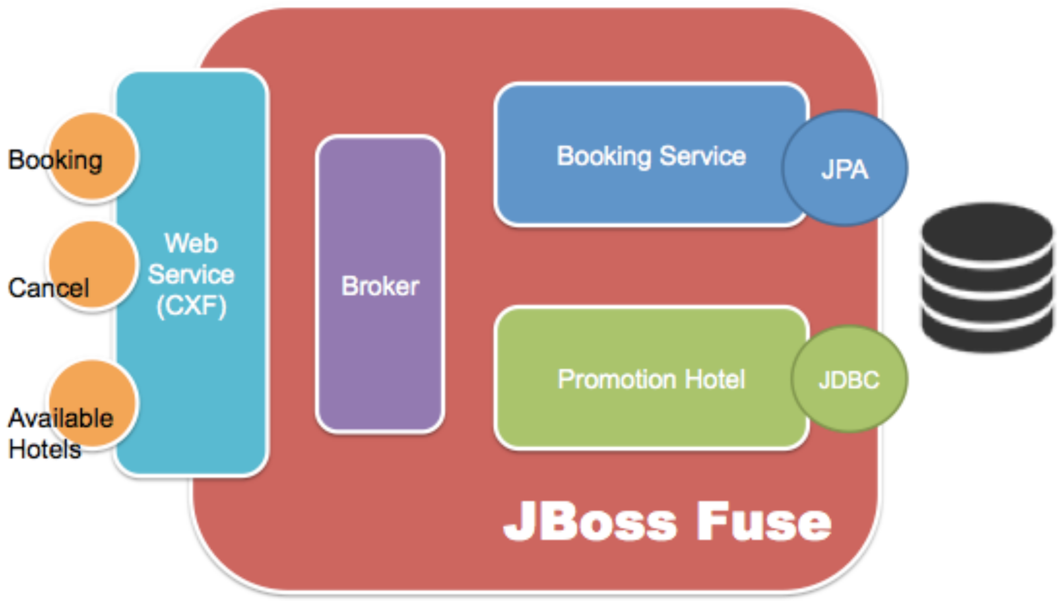
Before we begin,

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

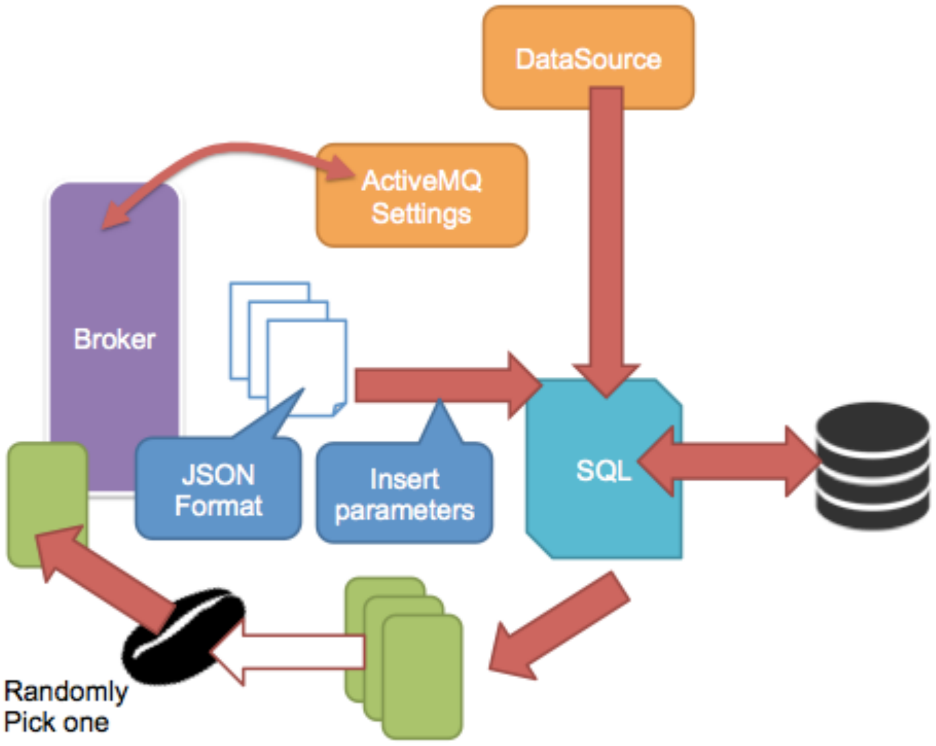
Situation

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.



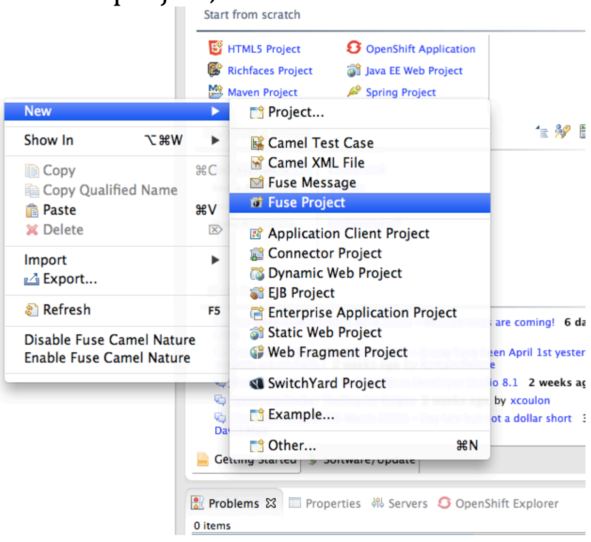
Promotion Hotels recommend - Retrieving Data from Database

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.

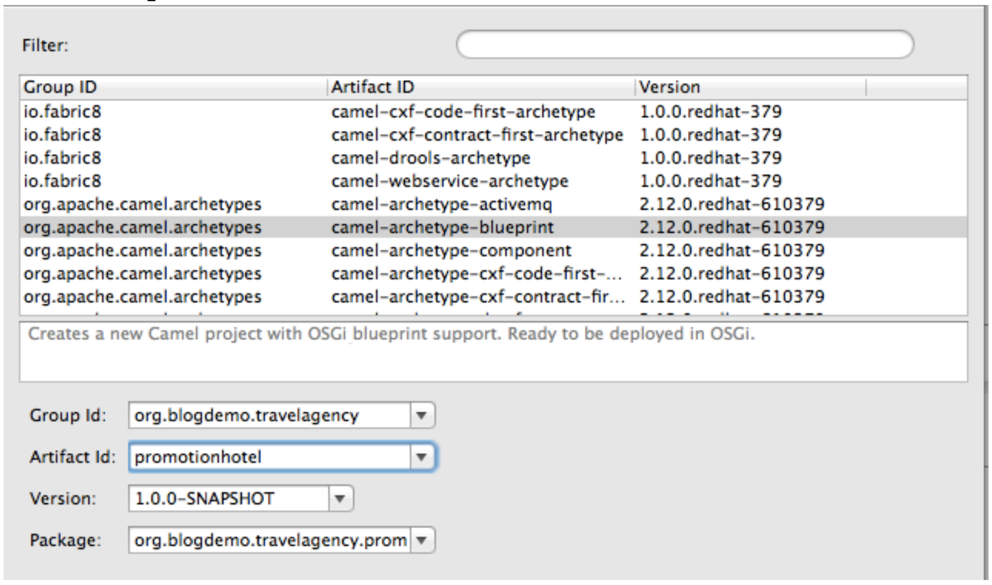


Instructions

Create a Fuse project,



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



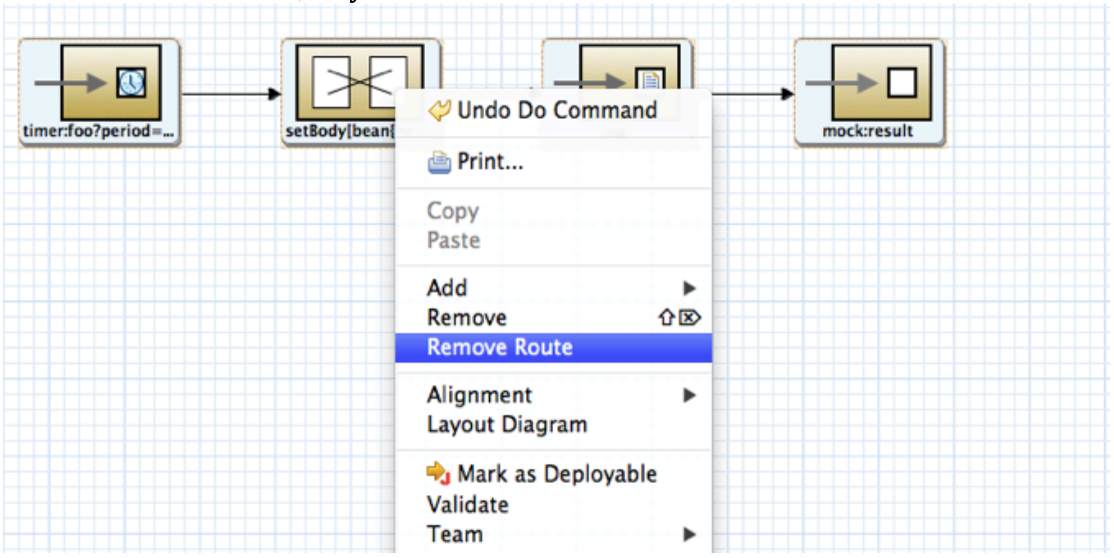
Filter:

Group ID	Artifact ID	Version
io.fabric8	camel-cxf-code-first-archetype	1.0.0.redhat-379
io.fabric8	camel-cxf-contract-first-archetype	1.0.0.redhat-379
io.fabric8	camel-drools-archetype	1.0.0.redhat-379
io.fabric8	camel-webservice-archetype	1.0.0.redhat-379
org.apache.camel.archetypes	camel-archetype-activemq	2.12.0.redhat-610379
org.apache.camel.archetypes	camel-archetype-blueprint	2.12.0.redhat-610379
org.apache.camel.archetypes	camel-archetype-component	2.12.0.redhat-610379
org.apache.camel.archetypes	camel-archetype-cxf-code-first-...	2.12.0.redhat-610379
org.apache.camel.archetypes	camel-archetype-cxf-contract-fir...	2.12.0.redhat-610379

Creates a new Camel project with OSGi blueprint support. Ready to be deployed in OSGi.

Group Id: org.blogdemo.travelagency
Artifact Id: promotionhotel
Version: 1.0.0-SNAPSHOT
Package: org.blogdemo.travelagency.prom

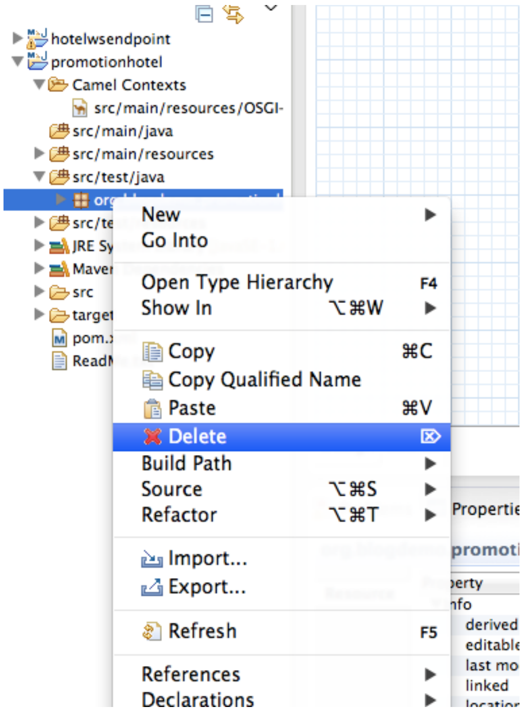
Open up `/example/src/main/resources/OSGI-INF/blueprint/blueprint.xml` and remove the route, so you are left with a blank canvas.

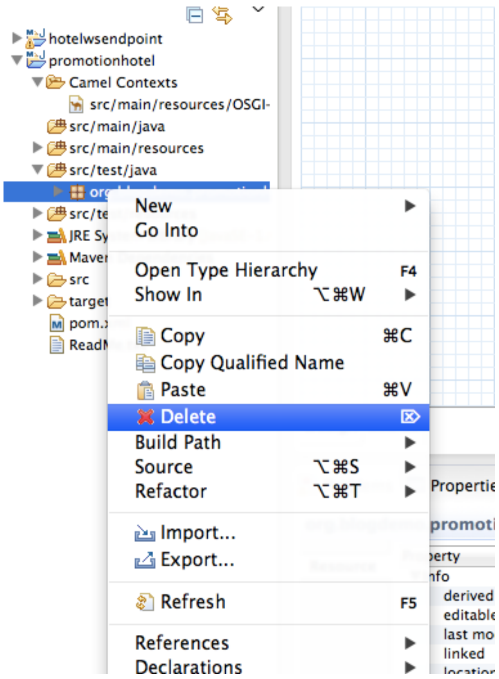


Remove the hellobean registry in the xml file. And delete all the java files.

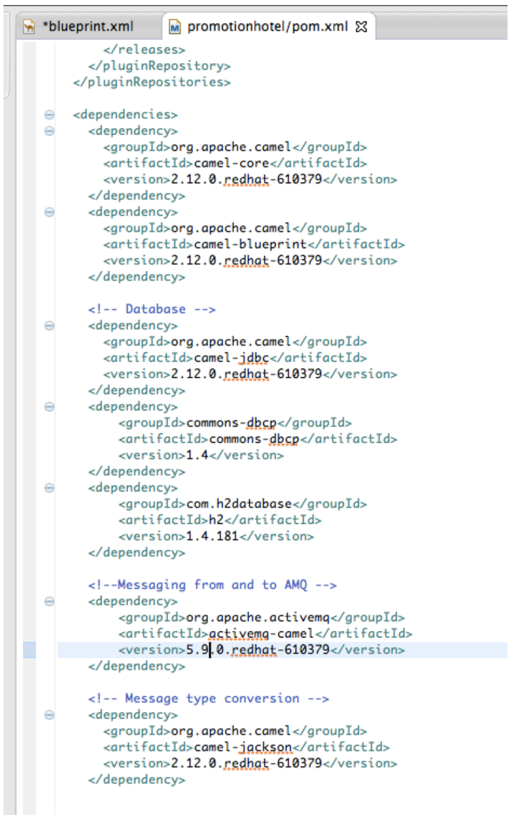
```
~~<bean id="helloBean" class="com.mycompany.camel.blueprint.HelloBean">~~
      ~~<property name="say" value="Hi from Camel"/>~~
~~</bean>~~
```

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.



```
        </releases>
      </pluginRepository>
    </pluginRepositories>

    <dependencies>
      <dependency>
        <groupId>org.apache.camel</groupId>
        <artifactId>camel-core</artifactId>
        <version>2.12.0.redhat-610379</version>
      </dependency>
      <dependency>
        <groupId>org.apache.camel</groupId>
        <artifactId>camel-blueprint</artifactId>
        <version>2.12.0.redhat-610379</version>
      </dependency>

      <!-- 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>
```

```
<!-- 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

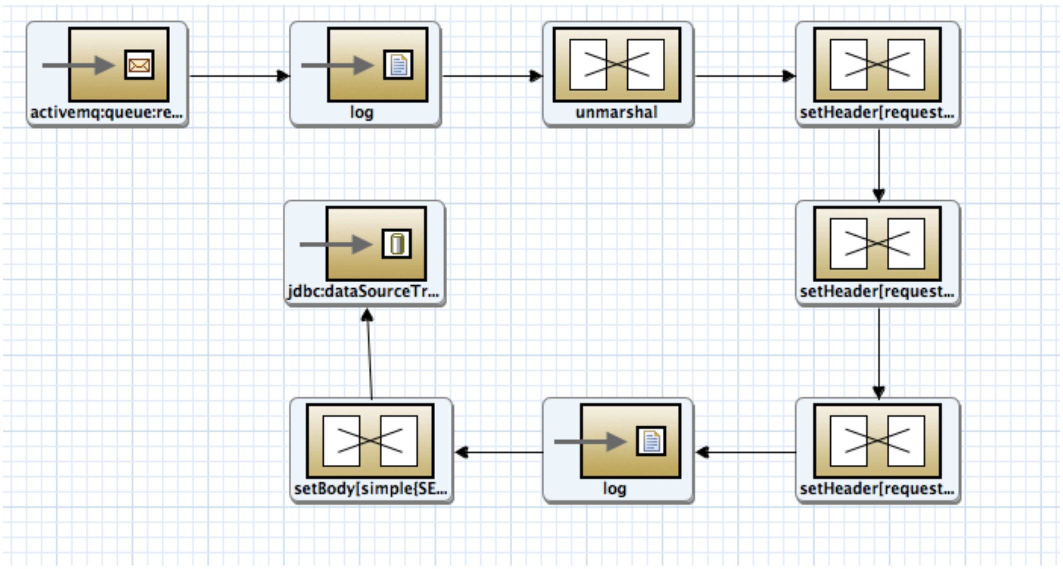
```
<bean id="dataSourceTravelAgency" class="org.apache.commons.dbcp.BasicDataSource">
  <property name="driverClassName" value="org.h2.Driver"/>
  <property name="url" value="jdbc:h2:file:~/h2/travelagency;AUTO_SERVER=TRUE" />
  <property name="username" value="sa"/>
  <property name="password" value=""/>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

Add activemq setting to connect to our messaging queue.

```
<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.



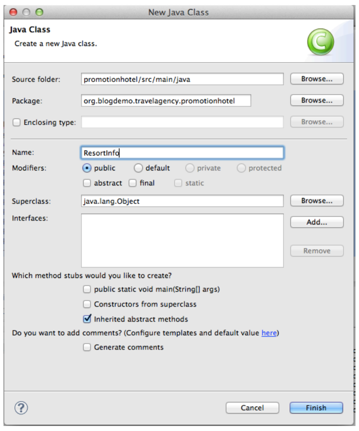
(When you save the route, it'll become a straight line)

- 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



New Java Class

Java Class
Create a new Java class.

Source folder: promotionhotel/src/main/java
Package: org.blogdemo.travelagency.promotionhotel
Enclosing type: [ ]
Name: ResortInfo
Modifiers: [✓ public] default private protected
[ ] abstract [ ] final [ ] static
Superclass: java.lang.Object
Interfaces:
Which method stubs would you like to create?
[ ] public static void main(String[] args)
[ ] Constructors from superclass
[✓] Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
[ ] Generate comments

```
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;
  }
}
```

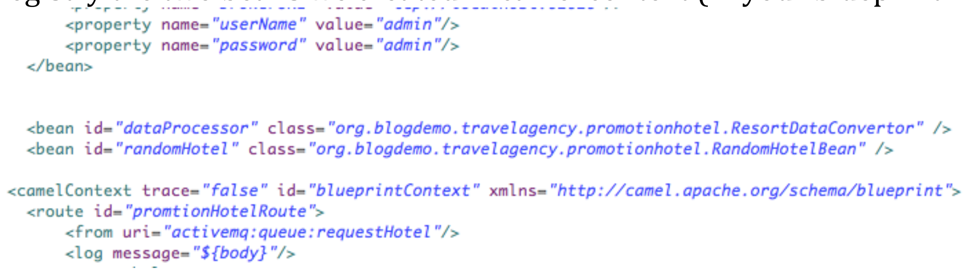
Create a java class
Package: org.blogdemo.travelagency.promotionhotel
Class name: ResortDataConvertor

```
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

```
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);
  }

}
```

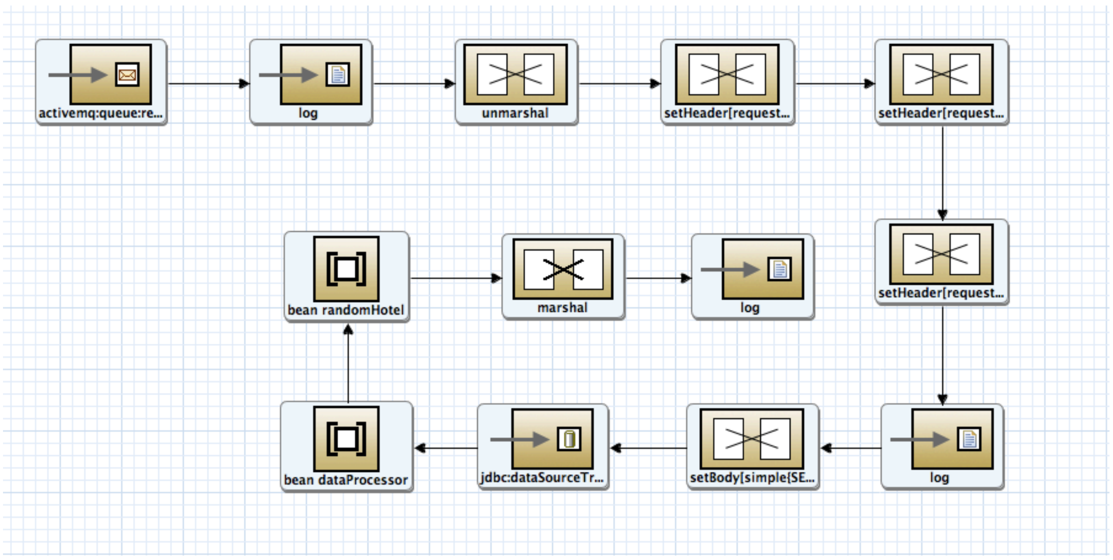
Registry the two beans we created in camel context (in your blueprint file)



```
        <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}"/>
```

```
<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

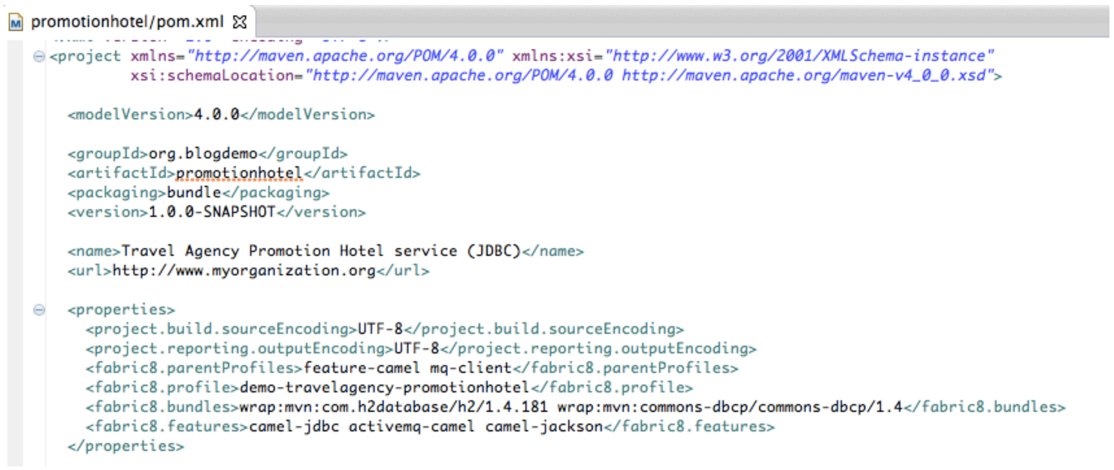
Start up JBoss Fuse by going to [project-root-directory]/summit-racing-camel-with-jboss-bpm-fuse/target/jboss-fuse-6.1.1.redhat-412/bin and run,
./fuse

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

```
    <!-- 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>
```

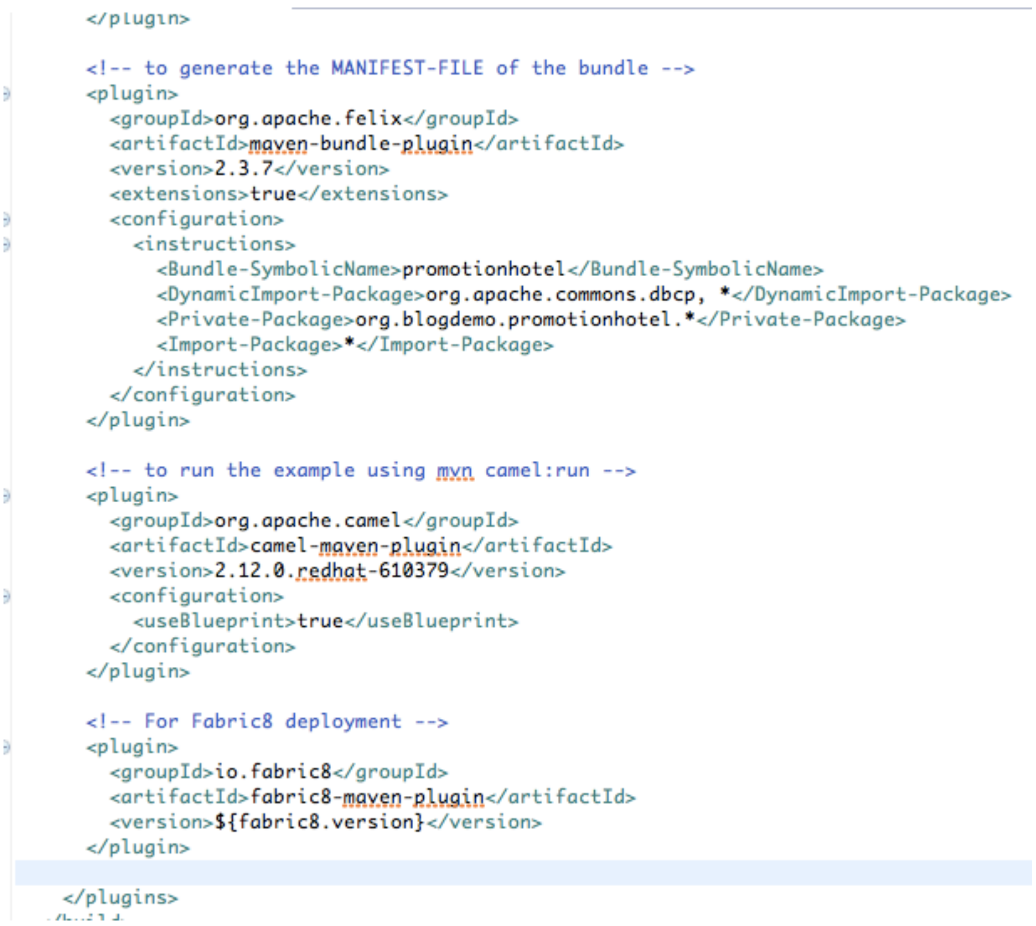


promotionhotel/pom.xml

```
<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>1.0.0-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>
```

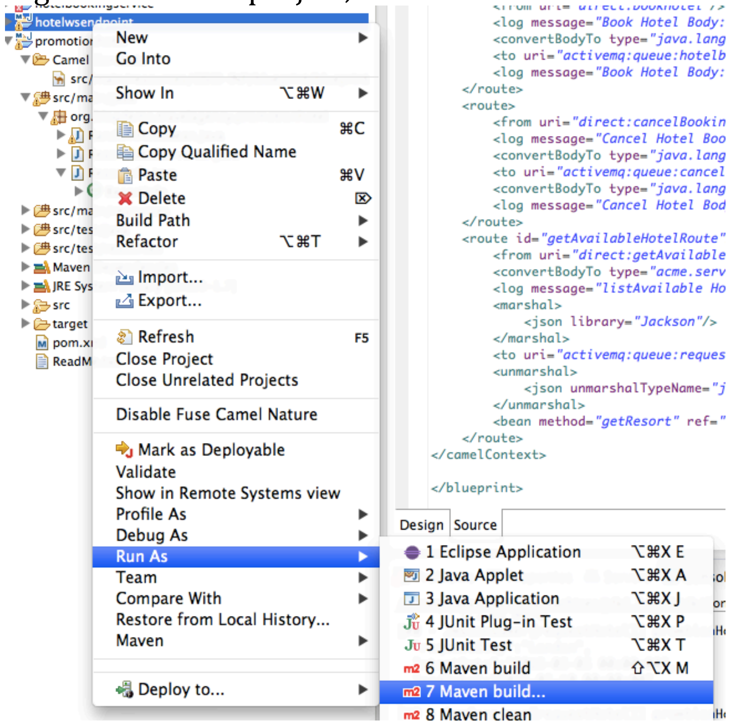
Add dynamic import to bundle maven configuration

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



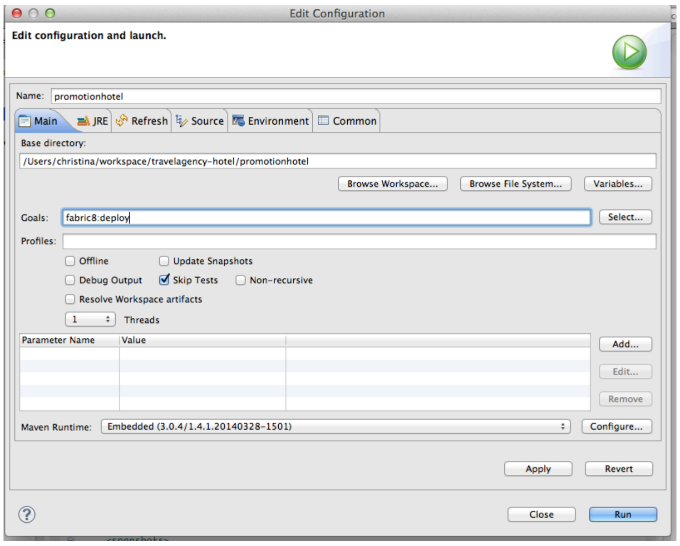
```
    </plugin>

    <!-- 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,



```
            <log message="Book Hotel Body:
            <convertBodyTo type="java.lang
            <to uri="activemq:queue:hotelb
            <log message="Book Hotel Body:
        </route>
        <route>
            <from uri="direct:cancelBookin
            <log message="Cancel Hotel Boo
            <convertBodyTo type="java.lang
            <to uri="activemq:queue:cancel
            <convertBodyTo type="java.lang
            <log message="Cancel Hotel Bod
        </route>
        <route id="getAvailableHotelRoute"
            <from uri="direct:getAvailable
            <convertBodyTo type="acme.serv
            <log message="listAvailable Ho
            <marshal>
                <json library="Jackson"/>
            </marshal>
            <to uri="activemq:queue:reques
            <unmarshal>
                <json unmarshalTypeName="j
            </unmarshal>
            <bean method="getResort" ref="
        </route>
    </camelContext>

</blueprint>
```

Type in “fabric8:deploy” as goal,


Edit Configuration

Edit configuration and launch.

Name: promotionhotel

Base directory:
/Users/christina/workspace/travelagency-hotel/promotionhotel

Goals: fabric8:deploy

Profiles:

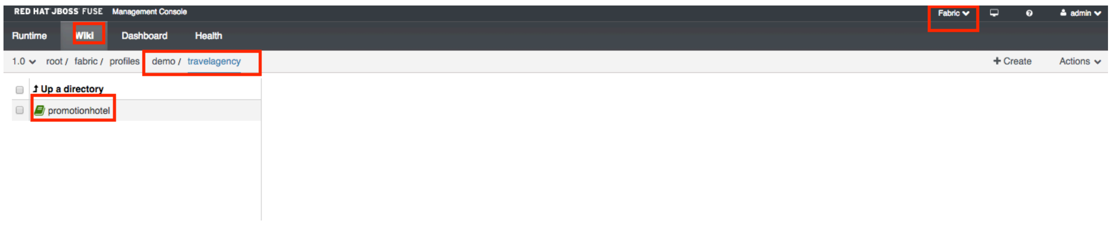
- [ ] Offline
- [ ] Update Snapshots
- [ ] Debug Output
- [x] Skip Tests
- [ ] Non-recursive
- [ ] Resolve Workspace artifacts

1 Threads

Parameter Name	Value

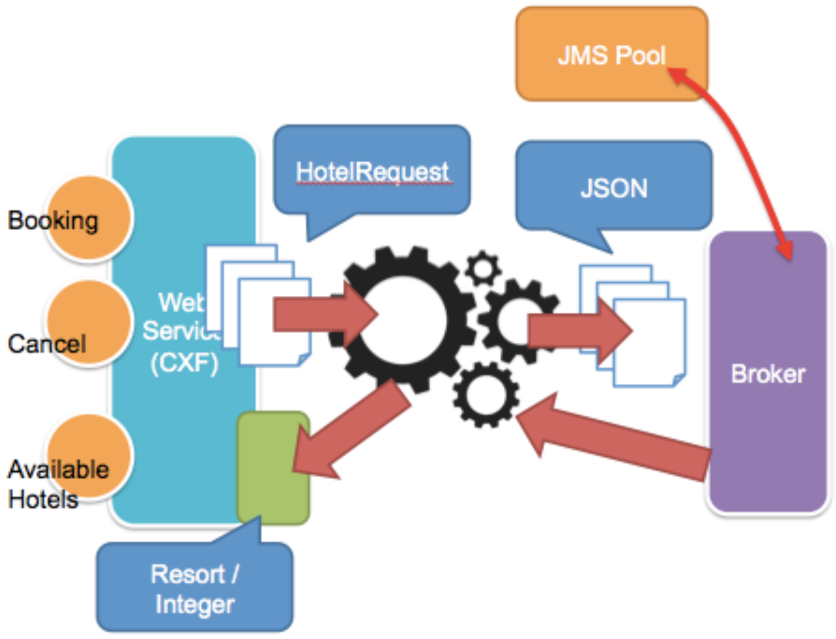
Maven Runtime: Embedded (3.0.4/1.4.1.20140328-1501)

Wait a few seconds. You will be able to see it in your fabric console.


RED HAT JBOSS FUSE Management Console
Fabric
admin
Runtime
Wiki
Dashboard
Health
1.0
root / fabric / profiles demo / travelagency
Create
Actions
Up a directory
promotionhotel

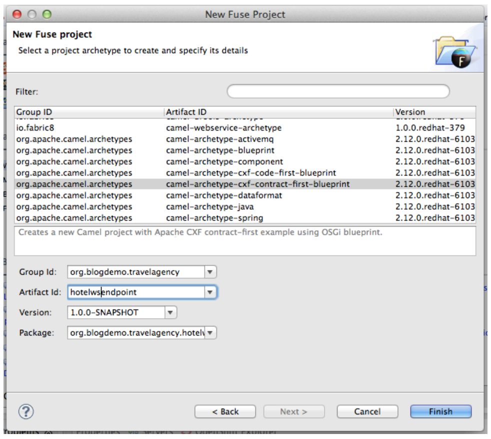
Hotel Service – Contract first Web Service (CXF)

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`



New Fuse project

Select a project archetype to create and specify its details

Filter:

Group ID	Artifact ID	Version
io.fabric8	camel-webservice-archetype	1.0.0.redhat-379
org.apache.camel.archetypes	camel-archetype-activemq	2.12.0.redhat-6103
org.apache.camel.archetypes	camel-archetype-blueprint	2.12.0.redhat-6103
org.apache.camel.archetypes	camel-archetype-component	2.12.0.redhat-6103
org.apache.camel.archetypes	camel-archetype-cxf-code-first-blueprint	2.12.0.redhat-6103
org.apache.camel.archetypes	camel-archetype-cxf-contract-first-blueprint	2.12.0.redhat-6103
org.apache.camel.archetypes	camel-archetype-dataformat	2.12.0.redhat-6103
org.apache.camel.archetypes	camel-archetype-java	2.12.0.redhat-6103
org.apache.camel.archetypes	camel-archetype-spring	2.12.0.redhat-6103

Creates a new Camel project with Apache CXF contract-first example using OSGi blueprint.

Group Id: org.blogdemo.travelagency
Artifact Id: hotelwsendpoint
Version: 1.0.0-SNAPSHOT
Package: org.blogdemo.travelagency.hotelv

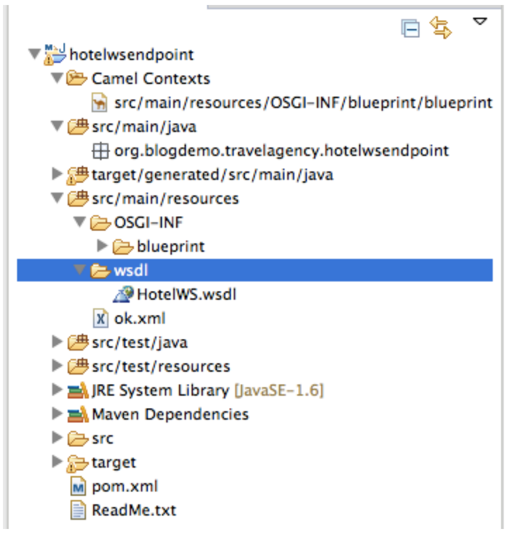
Delete the route, hello bean registry in blueprint.xml and delete everything under `src/java/*` and `src/test/*`(Go back to beginning of the first promotion hotel project if you are not sure what to do)
Setup the dependencies in pom.xml

```
    <!-- 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>
```

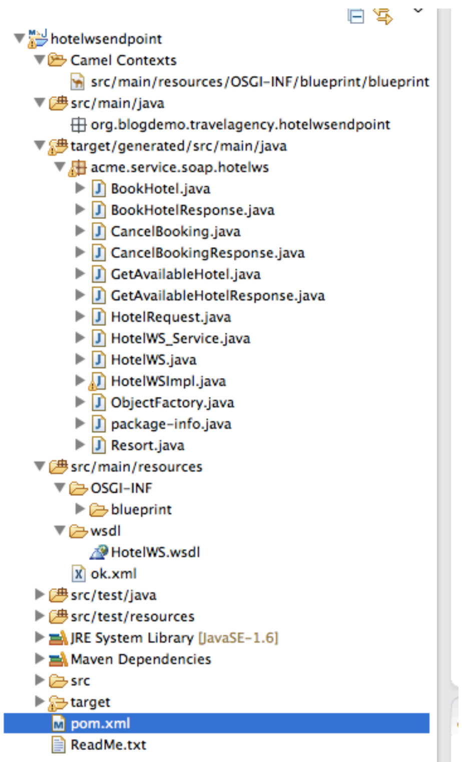
And because we already have an webservice wsdl, we are going to use it to generate all the classes we need, so add the plugin to your pom.xml too.

```
<!-- 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.



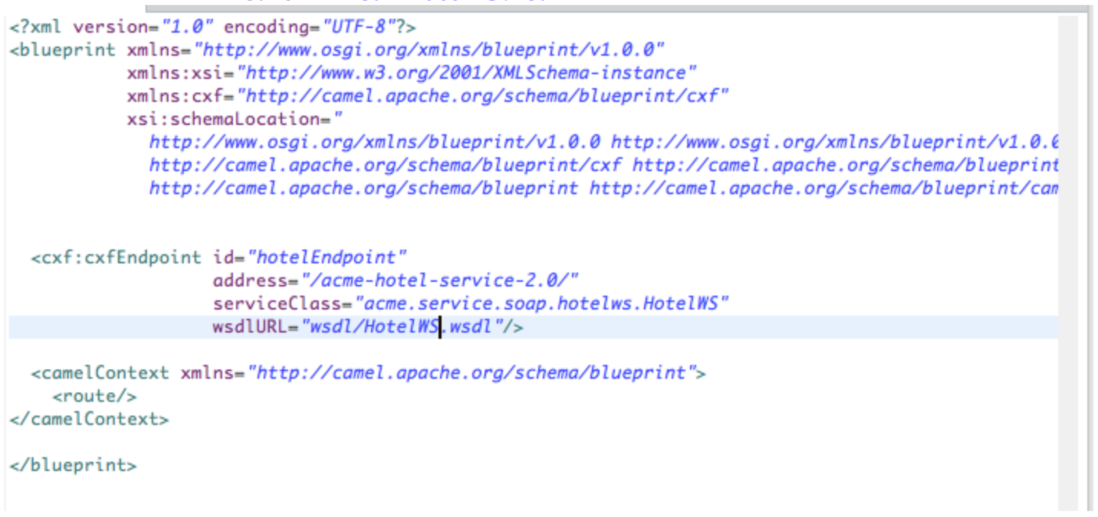
```
hotelwsendpoint
  Camel Contexts
    src/main/resources/OSGI-INF/blueprint/blueprint
  src/main/java
    org.blogdemo.travelagency.hotelwsendpoint
  target/generated/src/main/java
  src/main/resources
    OSGI-INF
      blueprint
    wsdl
      HotelWS.wsdl
    ok.xml
  src/test/java
  src/test/resources
  JRE System Library [JavaSE-1.6]
  Maven Dependencies
  src
  target
  pom.xml
  ReadMe.txt
```



```
hotelwsendpoint
  Camel Contexts
    src/main/resources/OSGI-INF/blueprint/blueprint
  src/main/java
    org.blogdemo.travelagency.hotelwsendpoint
  target/generated/src/main/java
    acme.service.soap.hotelws
      BookHotel.java
      BookHotelResponse.java
      CancelBooking.java
      CancelBookingResponse.java
      GetAvailableHotel.java
      GetAvailableHotelResponse.java
      HotelRequest.java
      HotelWS_Service.java
      HotelWS.java
      HotelWSImpl.java
      ObjectFactory.java
      package-info.java
      Resort.java
  src/main/resources
    OSGI-INF
      blueprint
    wsdl
      HotelWS.wsdl
    ok.xml
  src/test/java
  src/test/resources
  JRE System Library [JavaSE-1.6]
  Maven Dependencies
  src
  target
  pom.xml
  ReadMe.txt
```

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"/>
```

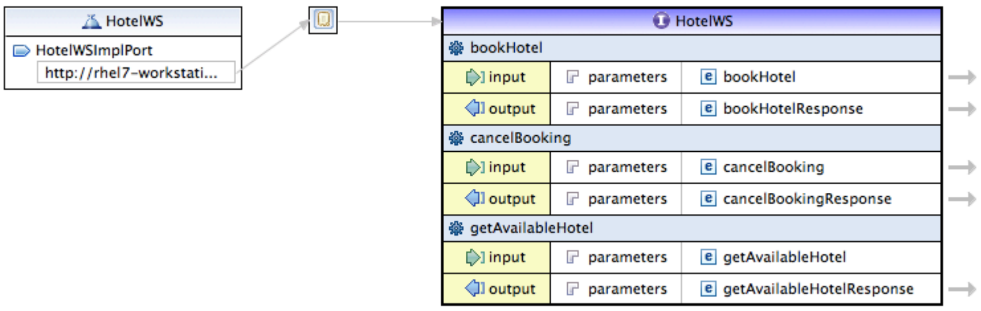


```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:cxf="http://camel.apache.org/schema/blueprint/cxf"
           xsi:schemaLocation="
             http://www.osgi.org/xmlns/blueprint/v1.0.0 http://www.osgi.org/xmlns/blueprint/v1.0.
             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

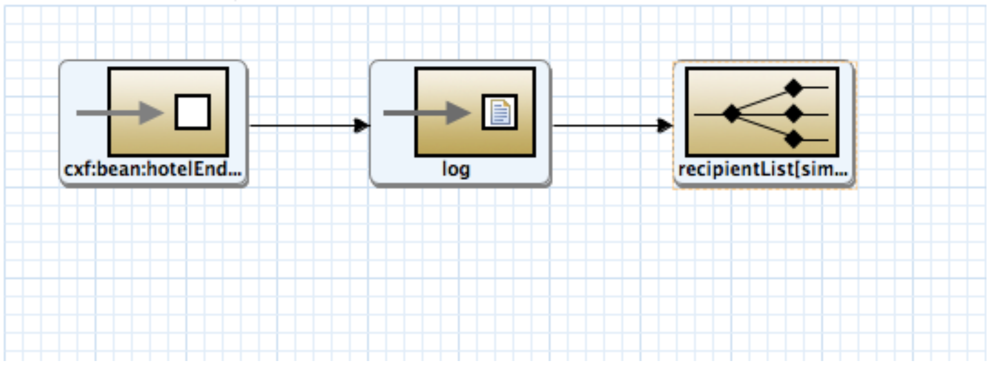
```
<bean id="jmsConnectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">
        <property name="brokerURL" value="tcp://localhost:61616"/>
        <property name="userName" value="admin"/>
        <property name="password" value="admin"/>
</bean>
<bean id="pooledConnectionFactory"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



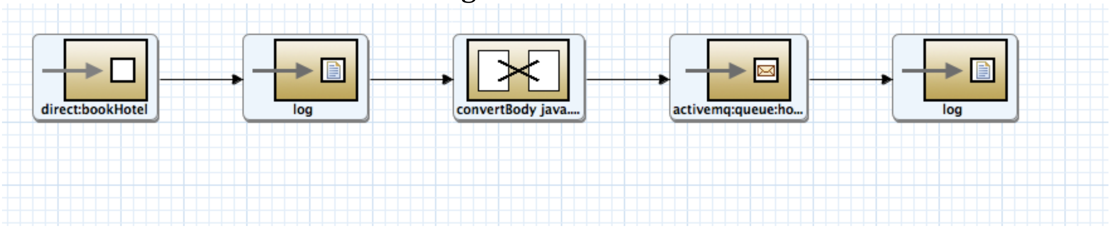
bookHotel		
input	parameters	bookHotel
output	parameters	bookHotelResponse
cancelBooking		
input	parameters	cancelBooking
output	parameters	cancelBookingResponse
getAvailableHotel		
input	parameters	getAvailableHotel
output	parameters	getAvailableHotelResponse

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

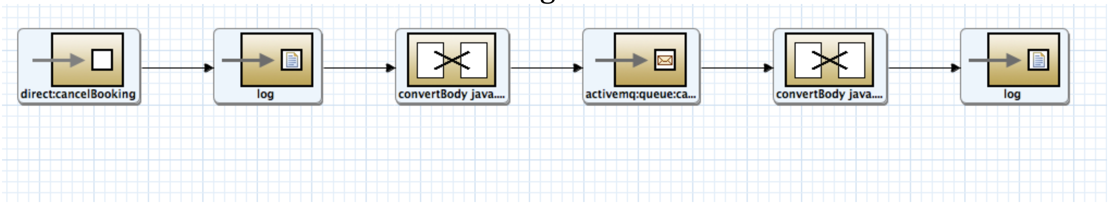
Create the route to handle booking



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

- Log Endpoint
    - Message: Book Hotel Body:[${body}]

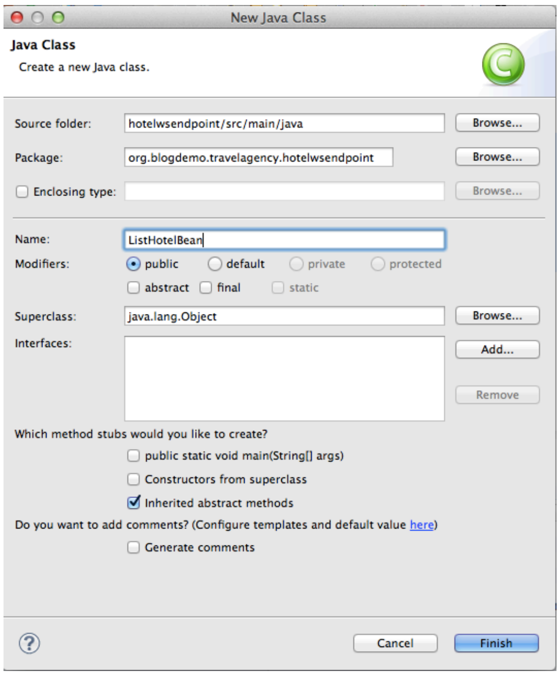
Create the route to handle canel booking



- Direct endpoint
    - 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


New Java Class

Java Class
Create a new Java class.

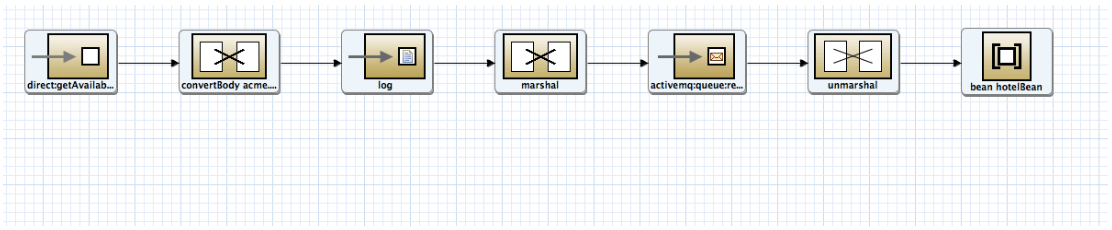
Source folder: hotelwsendpoint/src/main/java
Package: org.blogdemo.travelagency.hotelwsendpoint
Enclosing type:
Name: ListHotelBean
Modifiers: [✓ public] default private protected
abstract final static
Superclass: java.lang.Object
Interfaces:

Which method stubs would you like to create?
public static void main(String[] args)
Constructors from superclass
[✓ Inherited abstract methods]
Do you want to add comments? (Configure templates and default value here)
Generate comments

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

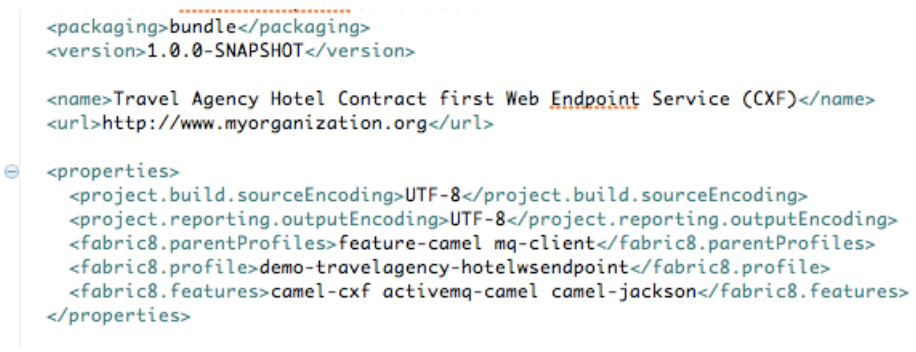
Deploy project

Add the fabric8 plugin in pom.xml

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

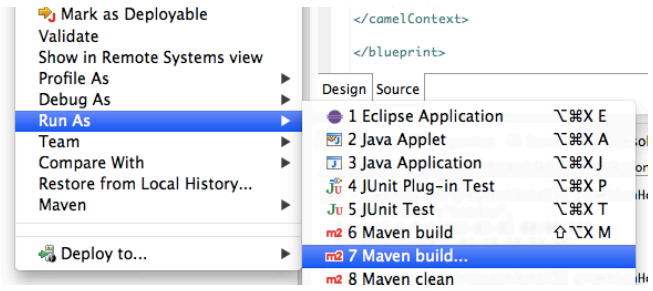
Also the deployment configurations in pom.xml

```
<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>
```

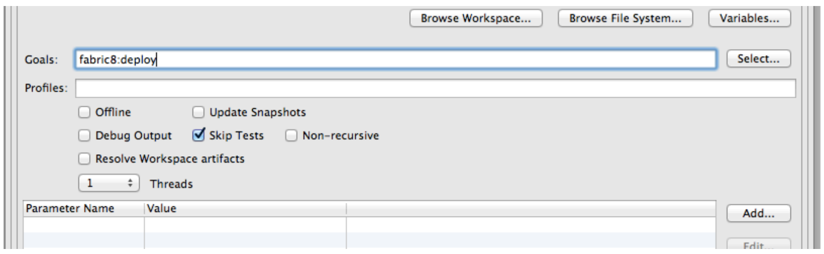


```
<packaging>bundle</packaging>
<version>1.0.0-SNAPSHOT</version>

<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,



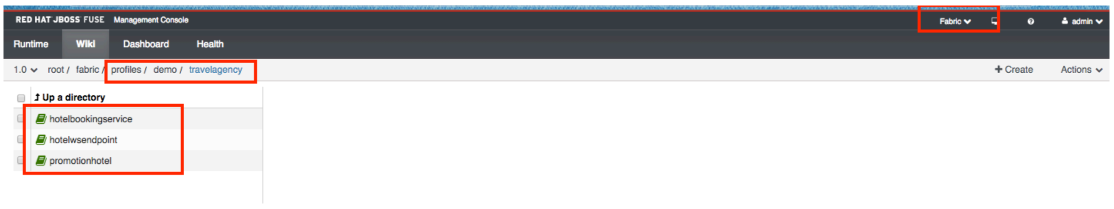
Goals: fabric8:deploy

Profiles:

- [ ] Offline
- [ ] Update Snapshots
- [ ] Debug Output
- [x] Skip Tests
- [ ] Non-recursive
- [ ] Resolve Workspace artifacts

Threads: 1

Wait a few seconds. You will be able to see it in your fabric console.

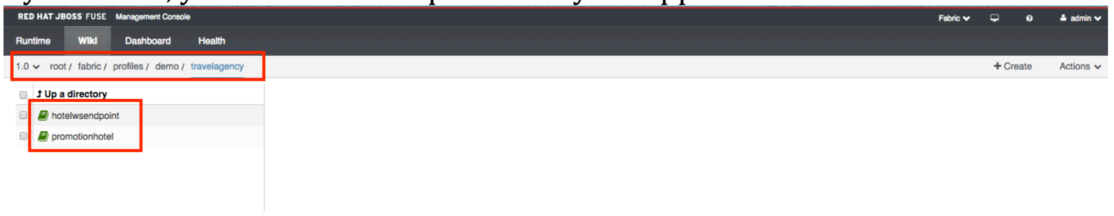


1.0 root / fabric / profiles / demo / travelagency

Up a directory
- hotelbookingservice
- hotelwsendpoint
- promotionhotel

Running Application in Fuse Fabric

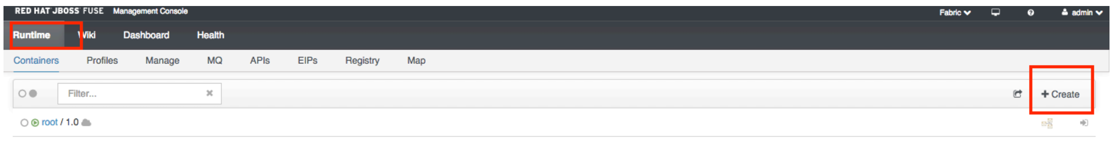
By this time, you will have two profiles in your application.



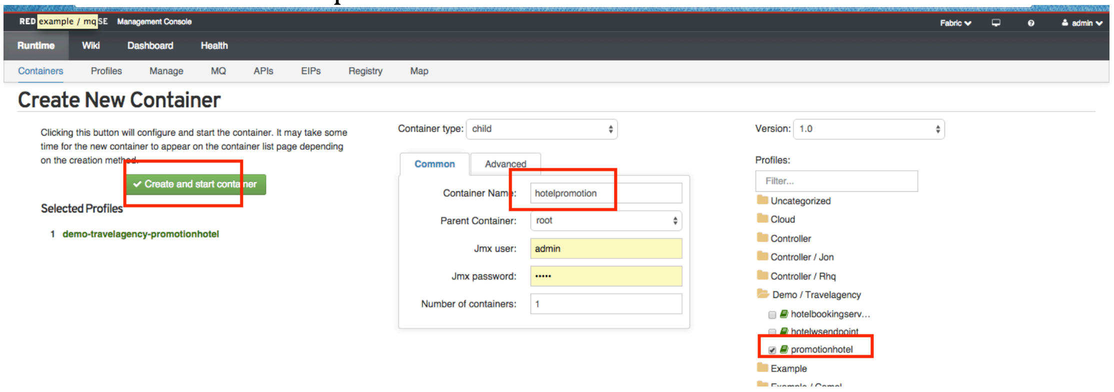
1.0 root / fabric / profiles / demo / travelagency

Up a directory
- hotelwsendpoint
- promotionhotel

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



Create New Container

Clicking this button will configure and start the container. It may take some time for the new container to appear on the container list page depending on the creation method.

Selected Profiles

1 demo-travelagency-promotionhotel

Container type: child

Container Name: hotelpromotion

Parent Container: root

Jmx user: admin

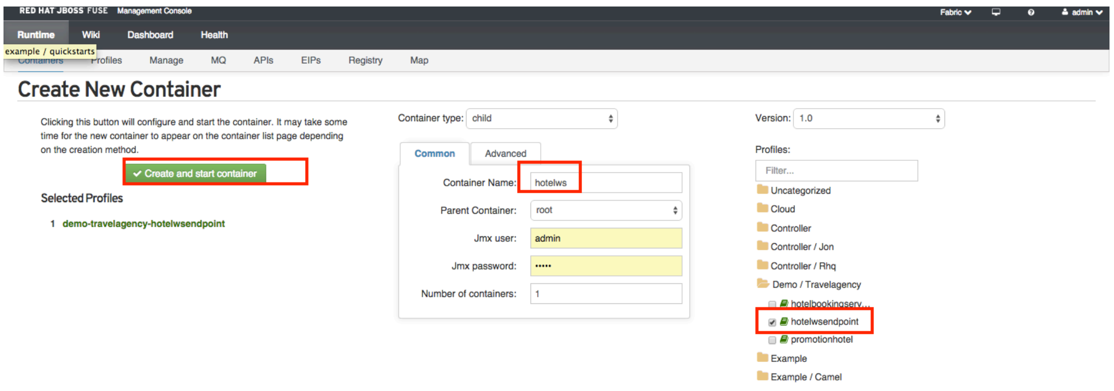
Jmx password: •••••

Number of containers: 1

Version: 1.0

Profiles:
- Uncategorized
- Cloud
- Controller
- Controller / Jon
- Controller / Rhq
- Demo / Travelagency
  - [ ] hotelbookingserv...
  - [ ] hotelwsendpoint
  - [✓] promotionhotel
- Example
- Example / Camel

Second container with web services.



Create New Container

Clicking this button will configure and start the container. It may take some time for the new container to appear on the container list page depending on the creation method.

Selected Profiles

1 demo-travelagency-hotelwsendpoint

Container type: child

Container Name: hotelws

Parent Container: root

Jmx user: admin

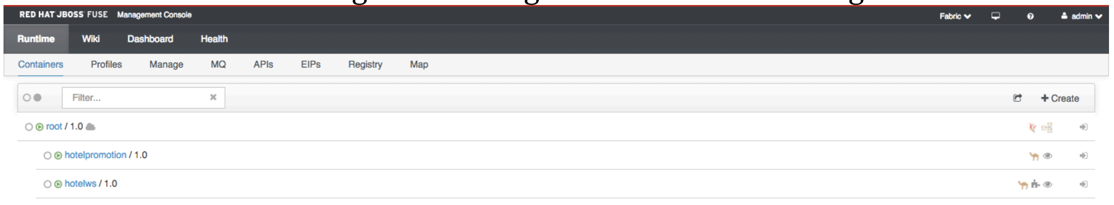
Jmx password: •••••

Number of containers: 1

Version: 1.0

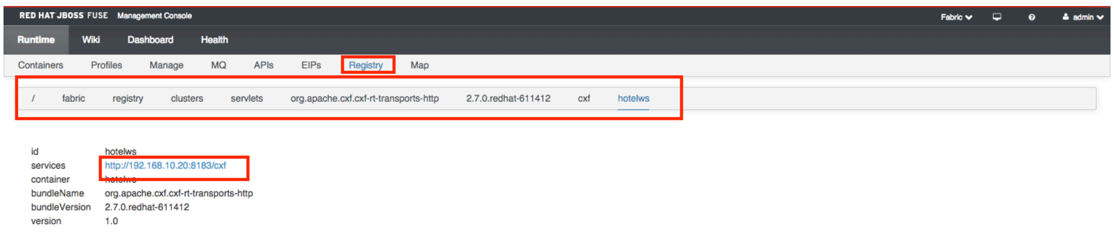
Profiles:
- Uncategorized
- Cloud
- Controller
- Controller / Jon
- Controller / Rhq
- Demo / Travelagency
  - [ ] hotelbookingserv...
  - [✓] hotelwsendpoint
  - [ ] promotionhotel
- Example
- Example / Camel

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

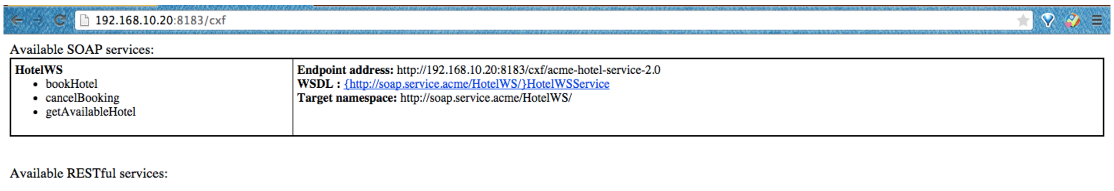


root / 1.0
hotelpromotion / 1.0
hotelws / 1.0

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`



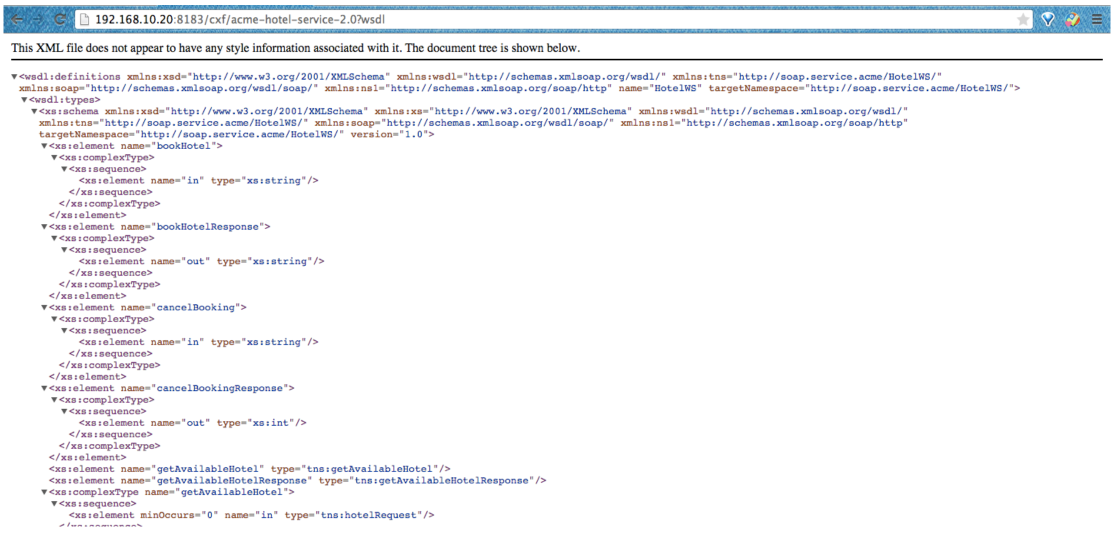
id	hotelws
services	http://192.168.10.20:8183/cxf
container	hotelws
bundleName	org.apache.cxf.cxf-rt-transports-http
bundleVersion	2.7.0.redhat-611412
version	1.0



Available SOAP services:

HotelWS • bookHotel • cancelBooking • getAvailableHotel	**Endpoint address:** http://192.168.10.20:8183/cxf/acme-hotel-service-2.0 **WSDL :** {http://soap.service.acme/HotelWS/}HotelWSService **Target namespace:** http://soap.service.acme/HotelWS/

Available RESTful services:



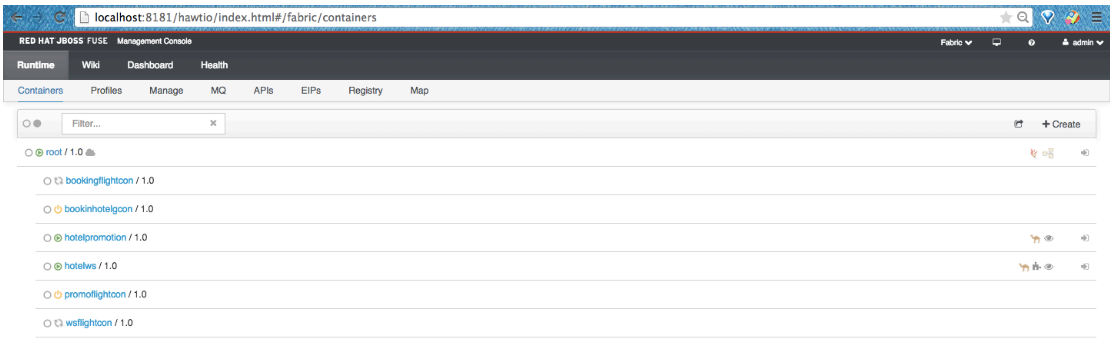
This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://soap.service.acme/HotelWS/"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:ns1="http://schemas.xmlsoap.org/soap/http" name="HotelWS" targetNamespace="http://soap.service.acme/HotelWS/">
 ▼<wsdl:types>
  ▼<xs:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://soap.service.acme/HotelWS/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:ns1="http://schemas.xmlsoap.org/soap/http"
    targetNamespace="http://soap.service.acme/HotelWS/" version="1.0">
   ▼<xs:element name="bookHotel">
    ▼<xs:complexType>
     ▼<xs:sequence>
        <xs:element name="in" type="xs:string"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   ▼<xs:element name="bookHotelResponse">
    ▼<xs:complexType>
     ▼<xs:sequence>
        <xs:element name="out" type="xs:string"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   ▼<xs:element name="cancelBooking">
    ▼<xs:complexType>
     ▼<xs:sequence>
        <xs:element name="in" type="xs:string"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   ▼<xs:element name="cancelBookingResponse">
    ▼<xs:complexType>
     ▼<xs:sequence>
        <xs:element name="out" type="xs:int"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="getAvailableHotel" type="tns:getAvailableHotel"/>
    <xs:element name="getAvailableHotelResponse" type="tns:getAvailableHotelResponse"/>
   ▼<xs:complexType name="getAvailableHotel">
    ▼<xs:sequence>
       <xs:element minOccurs="0" name="in" type="tns:hotelRequest"/>
```

To install and start up the rest of the microservices run

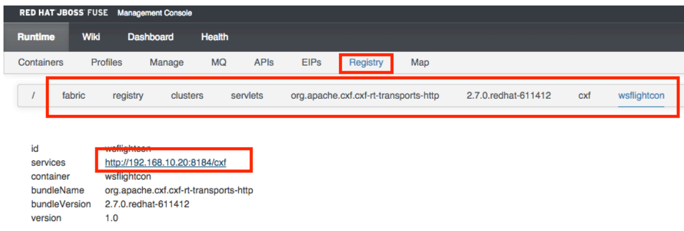
./fusemicroservices.sh

under [project-root-directory]/summit-racing-camel-with-jboss-bpm-fuse directory. It'll take a few minute to start,



```
root / 1.0
    bookingflightcon / 1.0
    bookinhotelgcon / 1.0
    hotelpromotion / 1.0
    hotelws / 1.0
    promoflightcon / 1.0
    wsflightcon / 1.0
```

And under Registry, you will see the port for flight web service.



```
/   fabric   registry   clusters   servlets   org.apache.cxf.cxf-rt-transports-http   2.7.0.redhat-611412   cxf   wsflightcon

id              wsflightcon
services        http://192.168.10.20:8184/cxf
container       wsflightcon
bundleName      org.apache.cxf.cxf-rt-transports-http
bundleVersion   2.7.0.redhat-611412
version         1.0
```

That's all ☺