Table of Contents

---

1. Fabric Lab

Goals

- Design an Apache Camel Route using the Wiki editor and provision a local container
- Develop a Camel project using JBoss Developer Studio and deploy it into the Fabric server using the Fabric8 Maven plug-in
- Migrate the project to a new profile version and roll it back
- Virtualize Camel endpoints and load balance requests

Lab Assets

The lab exercises and solutions are available in the following zip archives:

- https://github.com/gpe-mw-training/camel-labs/archive/v0.3-exercise.zip
- https://github.com/gpe-mw-training/camel-labs/archive/v0.3-solution.zip

To use the code in these exercises, you must connect to the OpenShift gear `demo` application to clone the Git exercises project locally, to compile it using Maven, and to deploy it into a JBoss Fuse server.

Next, you set up some JBoss Fuse containers and deploy the exercises.

1.1. Use SSH to Connect to Your Online Lab Account

Your lab instructions include the steps required to access your OpenShift gear via SSH here.

After you connect to your OpenShift gear, you can log on from JBoss Developer Studio or within your DOS/Unix terminal.

1.2. Clone the Project and Build the Code

1. After you are connected, use the following commands to set up a local Maven repository:

```
    cd $OPENSHIFT_DATA_DIR
    echo -e  "<settings>\n
<localRepository>$OPENSHIFT_DATA_DIR/.m2/repository</localRepository>\n</settings>\
n" > settings.xml
```

2. Clone the `camel-labs` project:

```
    git clone https://github.com/gpe-mw-training/camel-labs.git
```

3. Change to the `camel-labs` directory and checkout the `solution` branch:

```
    cd camel-labs/
    git checkout solution
```

4. Use the following command to build the code:

```
    mvn clean install -DskipTests=true -s $OPENSHIFT_DATA_DIR/settings.xml
```

1.3. Deploy it

In the module 1 lab exercises, you created a `demo` JBoss Fuse server and you deployed a Camel Twitter` application to explore the Fuse technology. If you have not deleted this OpenShift gear, you will reuse it here to deploy the compiled code to the local Maven Proxy. This code is used by JBoss Fuse to provision and deploy the artifacts into the various JBoss Fuse containers.

The information that you need is the username, the password, and the name of your OpenShift gear machine (e.g: `demo-fuse.apps.ose.opentlc.com`). You will use this information, and the Maven deploy command, to copy and paste your artifacts into the Maven Proxy directory of JBoss Fuse.

Execute the following commands, and replace the `username`, `password`, and `gear_machine` variables with the values that correspond to your setup:

```
    mvn clean deploy -
DaltDeploymentRepository=fabric::default::http://username:password@gear_machine/maven/u
pload -DskipTests=true -s $OPENSHIFT_DATA_DIR/settings.xml
```

Before deploying new Maven artifacts, (for example, after pushing new changes to the Git repository), remove what was previously deployed under this directory and restart the `demo` application.

```
cd $OPENSHIFT_DATA_DIR
rm rf -f ../../fuse/container/data/maven/proxy/tmp
```

1.4. Design a Camel Route Using the Wiki Editor

In this exercise, you create a new Apache Camel route using the Wiki editor in the JBoss Fuse Management Console. Next, you include your new route in a profile, and then deploy it in a new container on your OpenShift machine.

To design your new Apache Camel route, you will use the Blueprint DSL language and edit a Camel OSGI Blueprint XML document.

1. From your `demo` container, use your web browser to open the **JBoss Fuse Management Console**.
2. Click the **Wiki** tab and expand the tree to locate the `gpe/exercise` folder.
3. Create a profile:
   a. Click **Fabric8 Profile**, enter `cameldemo` as the **Name**, and then click **Create**.
   b. On the screen that appears, select **Fabric profile** as the document type, and enter `cameldemo` as the name of the document you want to create.

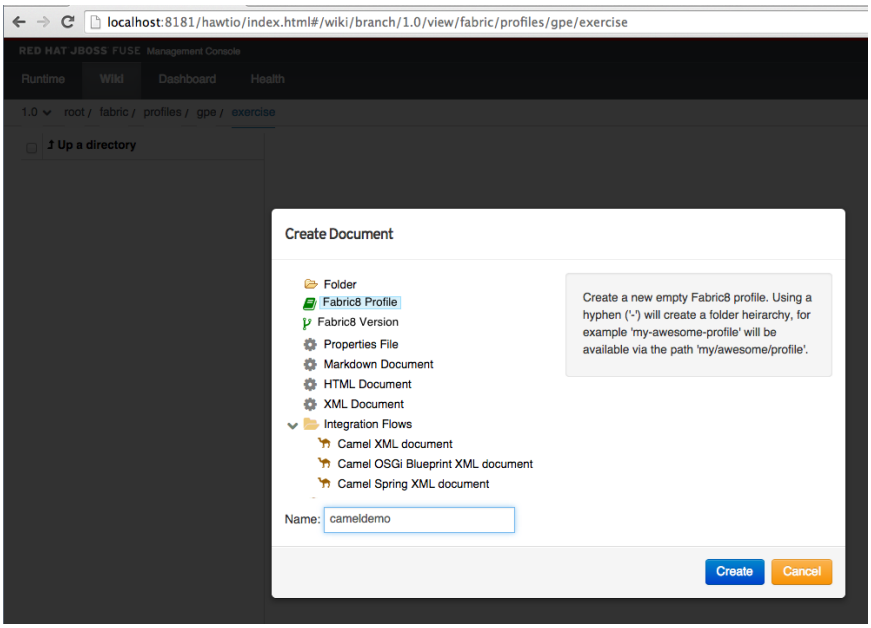

Figure 1. Create Document

4. After the profile is created, use the Wiki editor to design an Apache Camel route:
   a. Click **Create** again.
   b. In the popup window, select **Camel OSGI Blueprint XML document** as the document type and **camel-blueprint.xml** as the file name.

      The Wiki editor refreshes and a dummy route appears on the screen.

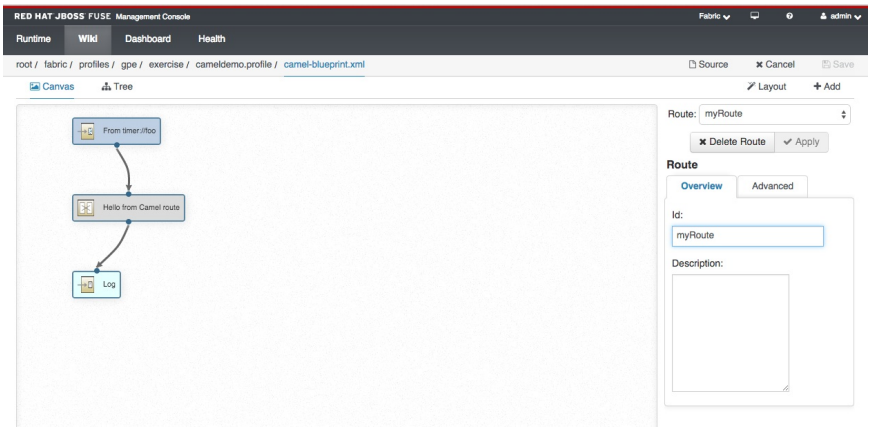

Figure 2. Wiki Editor

c. Select **Log** as the last endpoint of the route.

d. Click **Add** to add a new node.

e. In the **Add Node** window, under the **Transformation** category, select **Set Header**, and then click **Add**.

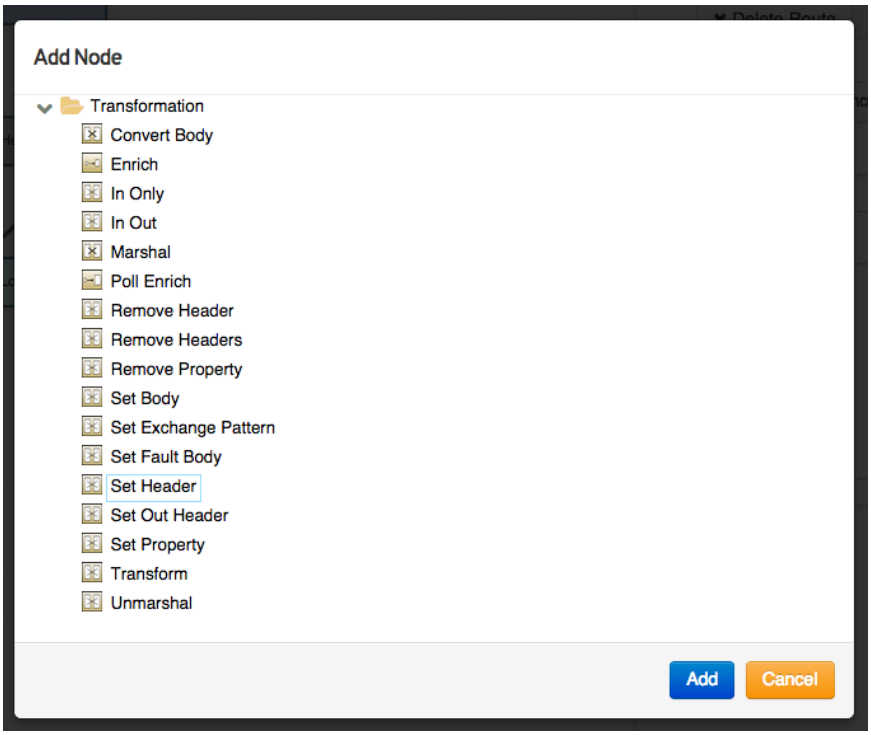

Figure 3. Add Node Window

The route refreshes and a new endpoint displays at the end of the route.

f. Reselect the **Set Header** endpoint and define the following:

- **Expression**: `Hello from GPE Team members`
- **Language**: `simple`
- **Header Name**: `simple`

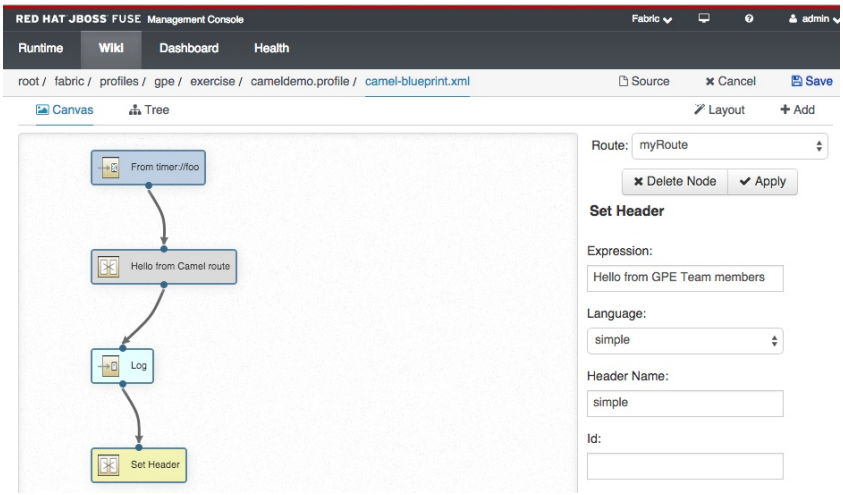

Figure 4. Define Set Header

g. Click **Apply**, and then click **Save** to refresh the editor.

5. Reselect the last processor of the route and add a **Log** processor node:

a. Select the last endpoint, and click **Add**.

b. In the popup window, under the **Endpoints** category, select the **Log** processor.

c. Click **Add**.

d. Under **Message**, enter `Header Simple : ${header.simple}`.

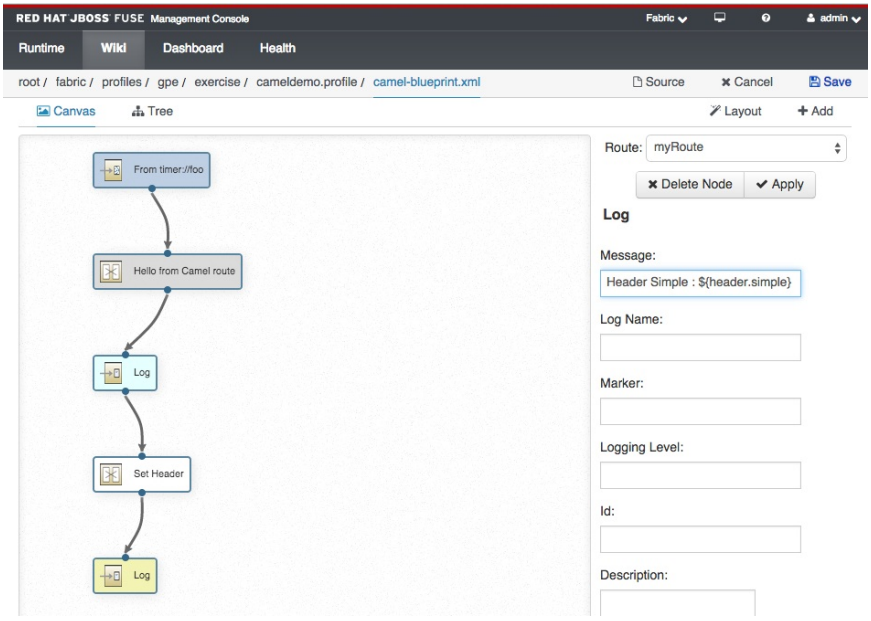

Figure 5. Define Log

e. Click **Apply**, and then click **Save** to finalize the route.

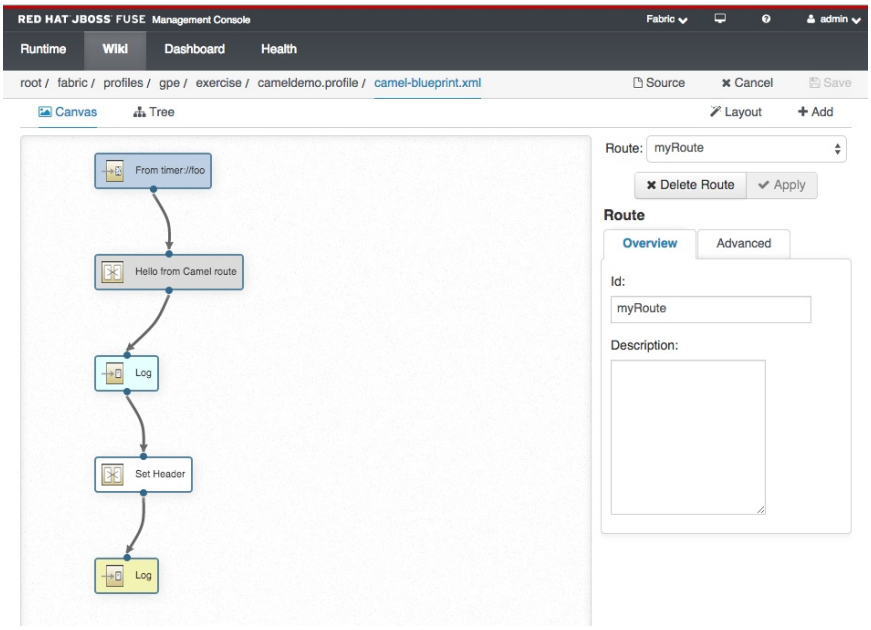

Figure 6. Complete Route

f. Click **View Source** to review the generated DSL.

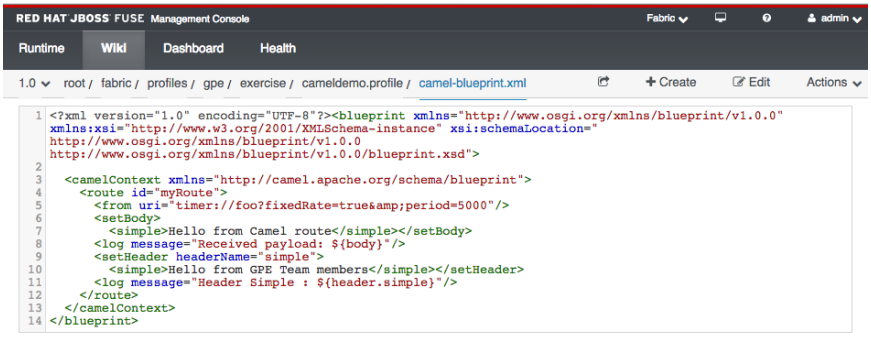

Figure 7. `camel-blueprint.xml` Source

6. Finalize the definition of the project/profile so you can deploy this route in a Fabric managed container:

a. Select the **cameldemo.profile** link from the Wiki tree.

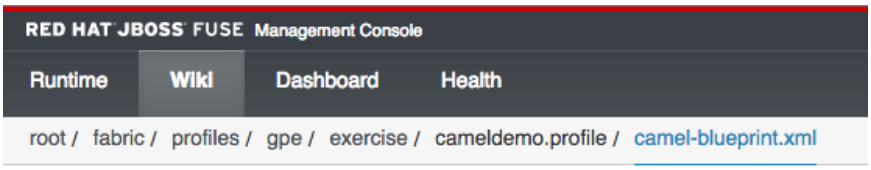

Figure 8. Wiki Tree

b. After the screen refreshes, click the small green editor icon under **Parents**.

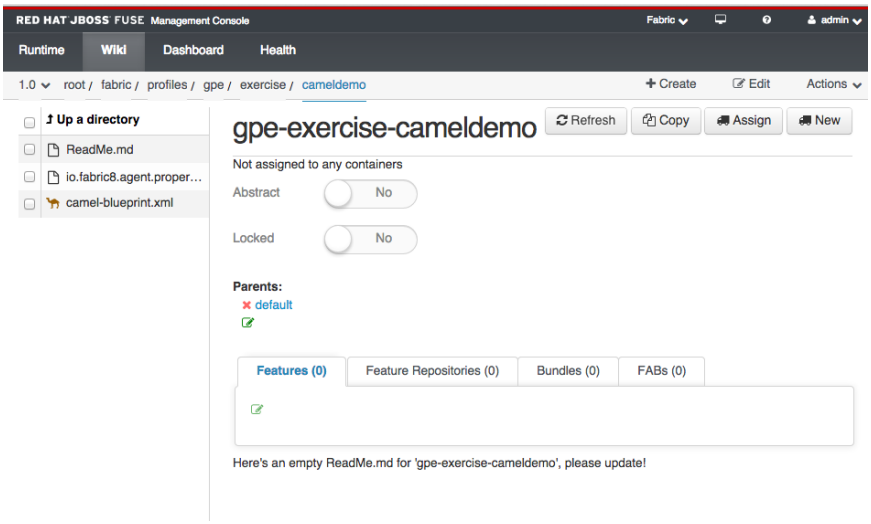

Figure 9. `cameldemo` Wiki

c. Use **Feature** as the filter token for the profile selections.

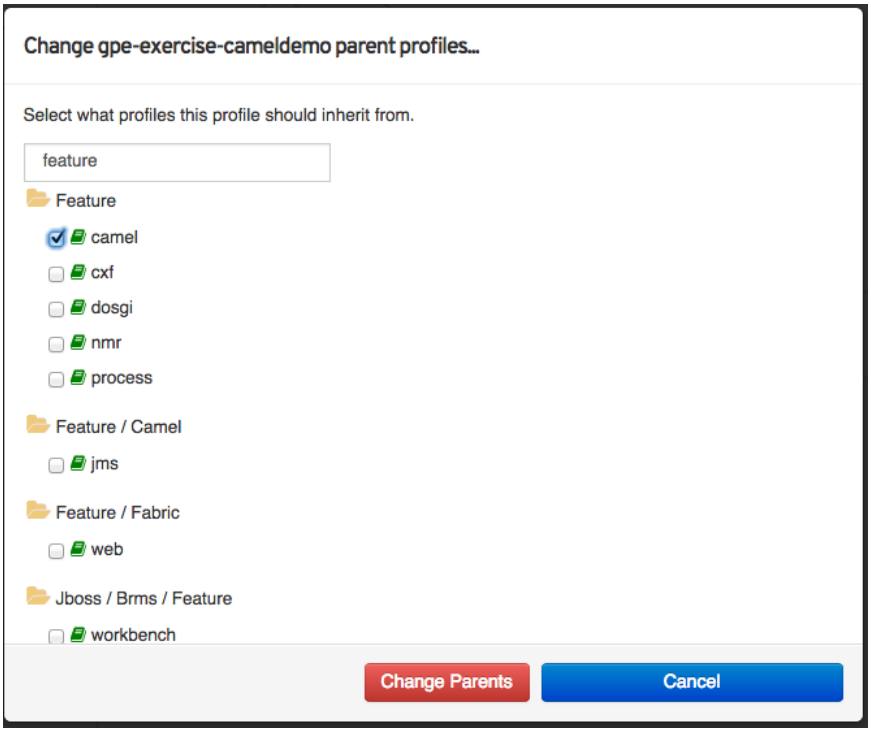

Figure 10. Filter Profile Selections

d. Click **Change Parents** and wait for the editor to refresh.

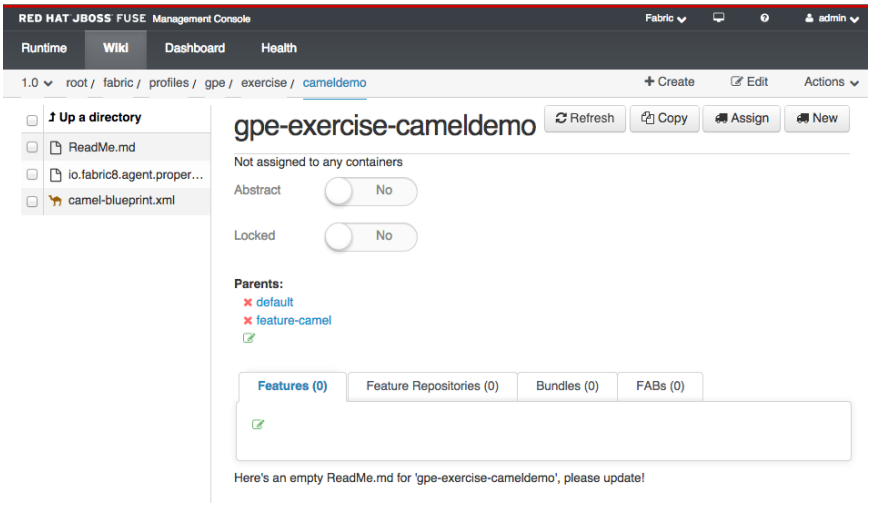

Figure 11. Edit Parents

e. Add the Camel route by clicking the **Bundles** tab and adding the location of the `blueprint:profile:camel-blueprint.xml` file.

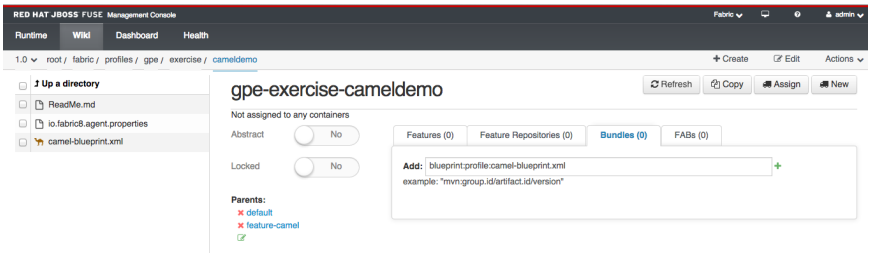

Figure 12. Add Bundle - Part 1

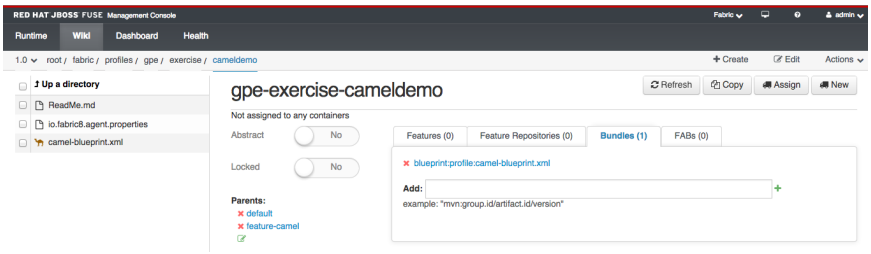

Figure 13. Add Bundle - Part 2

The `Blueprint` protocol is used to create a bundle, which is a JAR file that contains your XML route definition. This file is located under the `OSGI-INF/blueprint/camel-blueprint.xml` directory . When you provision your Fabric container, this is the bundle you will deploy.

7. Deploy this profile/project into a new Fuse container:

a. Click **New** and enter `cameldemo` as the **Container Name**.

b. Enter your OpenShift platform credentials to register with the server, so you can create a new Fuse container in your OpenShift gear.

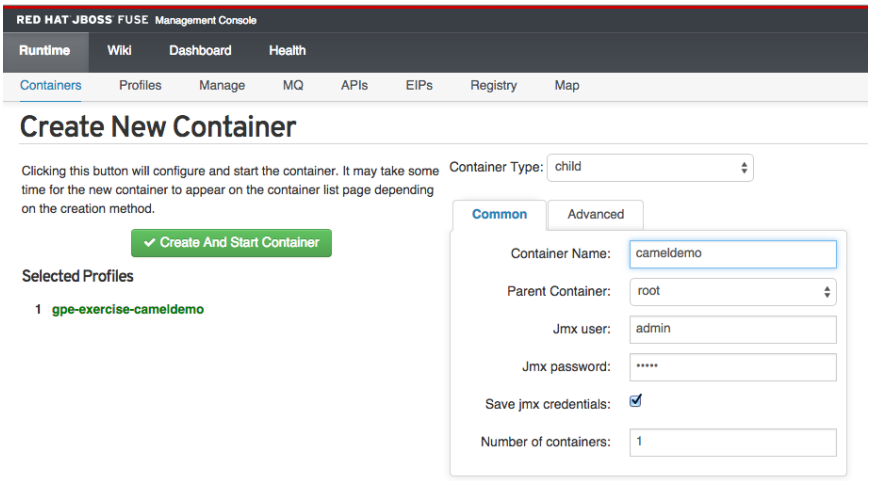

Figure 14. Create New Container

c. Click **Create And Start Container**.

d. Wait until you are redirected to the **Container** view on the **Runtime** tab. This is where you can select the container and open it.

e. Click the name of the container to discover its information: associated profiles, status, settings, urls.

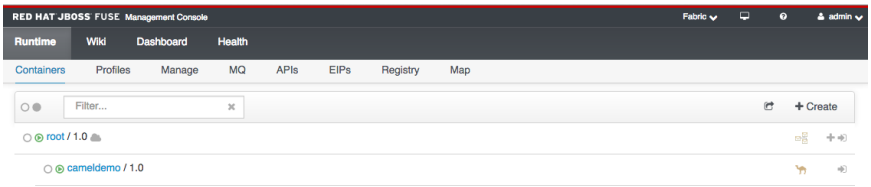

Figure 15. Select Container

f. Click **Open** to access the JBoss Fuse Management Console for this container.

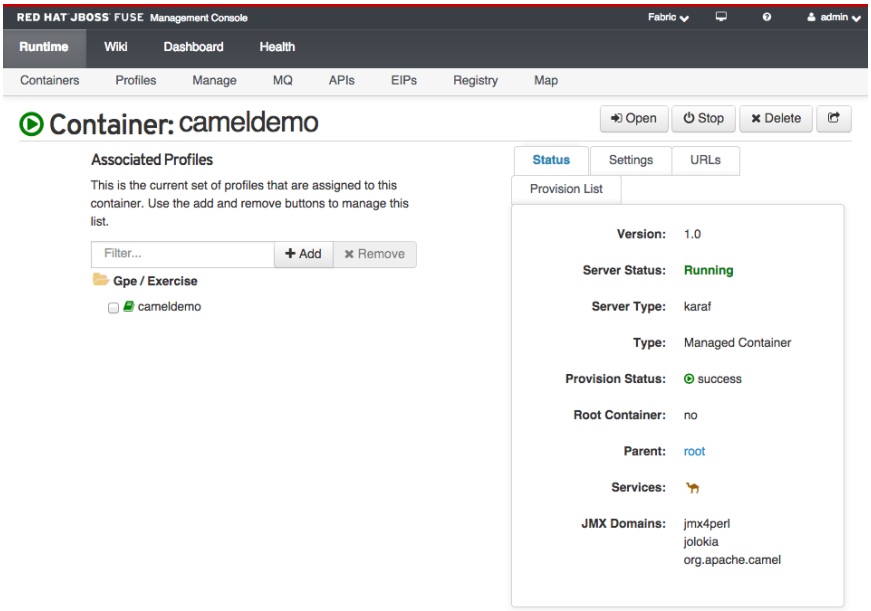

Figure 16. Container Details

g. Click the **Logs** tab to review the **Filter logs** content.

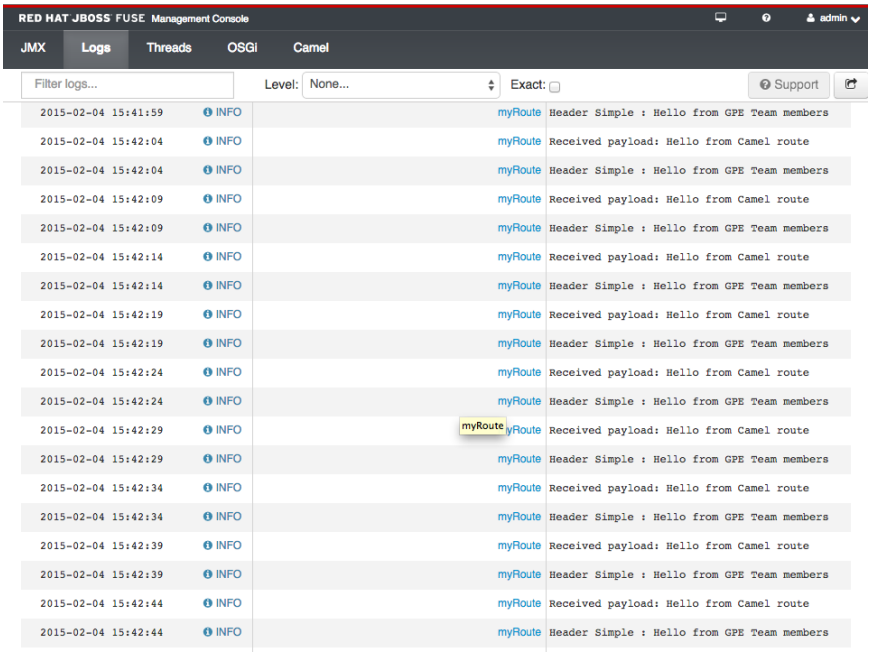

Figure 17. Logs

h. Click the `Camel` tab to review the deployed route, endpoints, metrics, and so on.

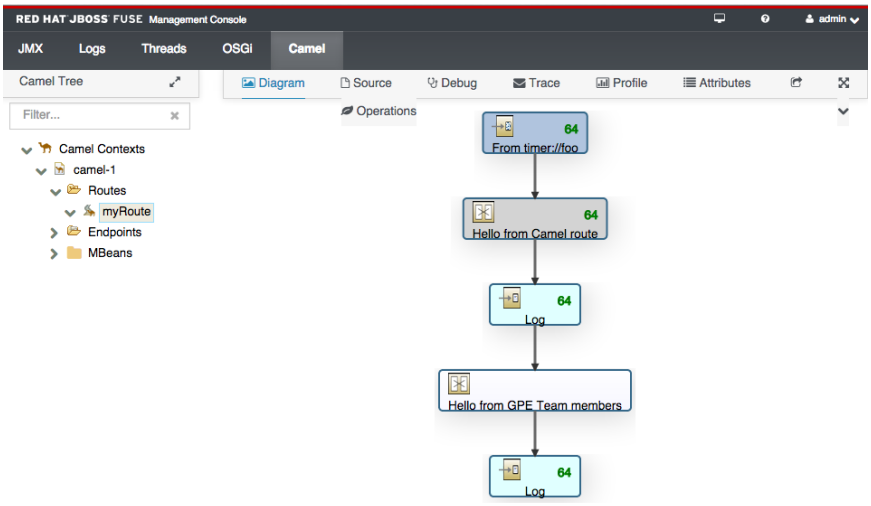

Figure 18. Camel Tab - Diagram View

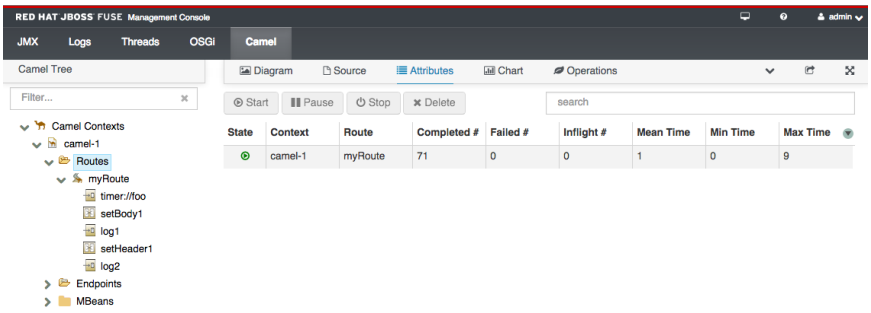

Figure 19. Camel Tab - Attributes View

1.5. Develop a Camel Project Using JBoss Developer Studio

In this exercise, you develop a Camel project using JBoss Developer Studio. This project uses the `camel-weather` component to collect weather information from different locations and then publishes the results in the log. You will use the Camel Blueprint DSL language to design the route.

1. Open **JBoss Developer Studio** and create a new workspace named `cameldemo`.
2. Create a new JBoss Fuse project:
   a. Select **File** → **New** → **Fuse Project** from the menu.
   b. Skip the first screen, **New Fuse project** / **Select project location**, and click **Next**.
   c. Set the **project archetype** to `camel-archetype-blueprint`.
   d. Set the **Group Id** to `com.redhat.gpe`.
   e. Set the **Artifact Id** to `fabric-forecasting-application`.
   f. Set the **Version** to `1.0`.
   g. Click **Finish**.

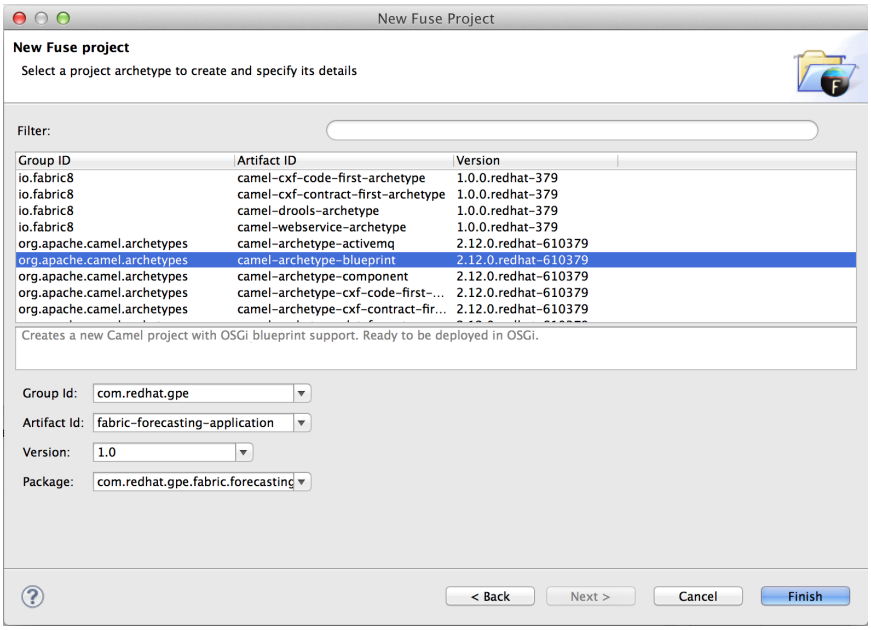

Figure 20. New Fuse Project

The project is created and should now appear in **Project Explorer**.

3. In the `pom.xml` file, add two new dependencies:
   a. Copy and paste the `camel-core` dependencies twice.
   b. Replace the first instance of `camel-core` with `camel-groovy`.
   c. Replace the second instance of `camel-core` with `camel-weather`.

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

Figure 21. Dependencies in `pom.xml`

4. Prepare to design the Camel route:
   a. In **Project Explorer**, expand `src/main/resources/OSGI-INF/blueprint`.
   b. Open the `blueprint.xml` file. The default route displays.
   c. Click the **Source** tab and remove the `<bean/>` and `route` content.

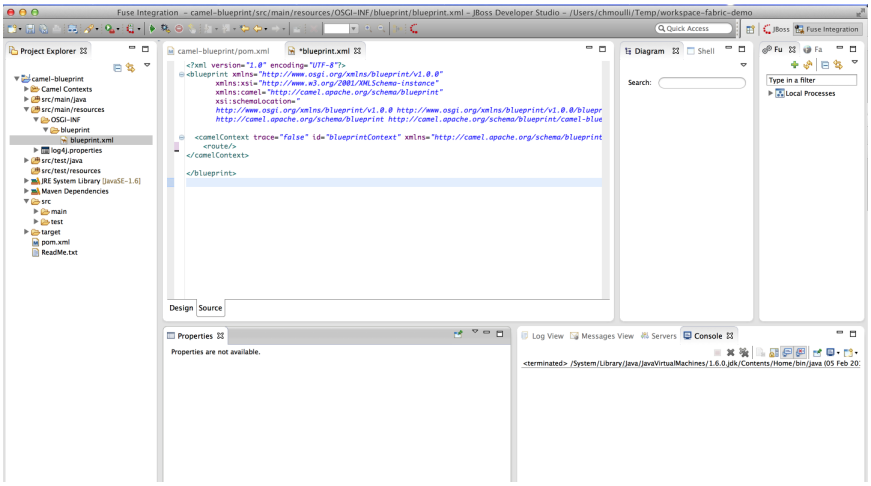

Figure 22. Route With Sections Removed

5. Create your first route:
   a. Click the **Design** tab.
   b. Drag the **Generic** from the palette and drop it on the workspace.
   c. Click the endpoint you just created.
   d. Under **Properties**, set the **Endpoint Uri** to `timer:foo?period=10000`.
   e. Set the event to fire every `10s`.
   f. Select **File** → **Save**.

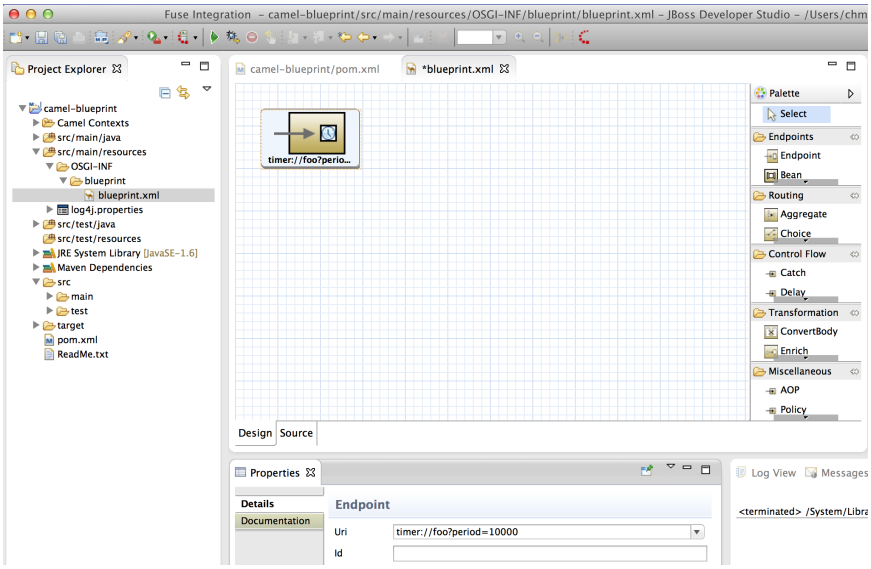

Figure 23. From Endpoint

6. Add a **Log** processor:
   a. Drag the **Log** processor from the palette and drop it on the workspace.
   b. Link the two endpoints using the arrow of the **From** endpoint.
   c. Set the **Message** to `Collecting weather info`.
   d. Select **File** → **Save**.

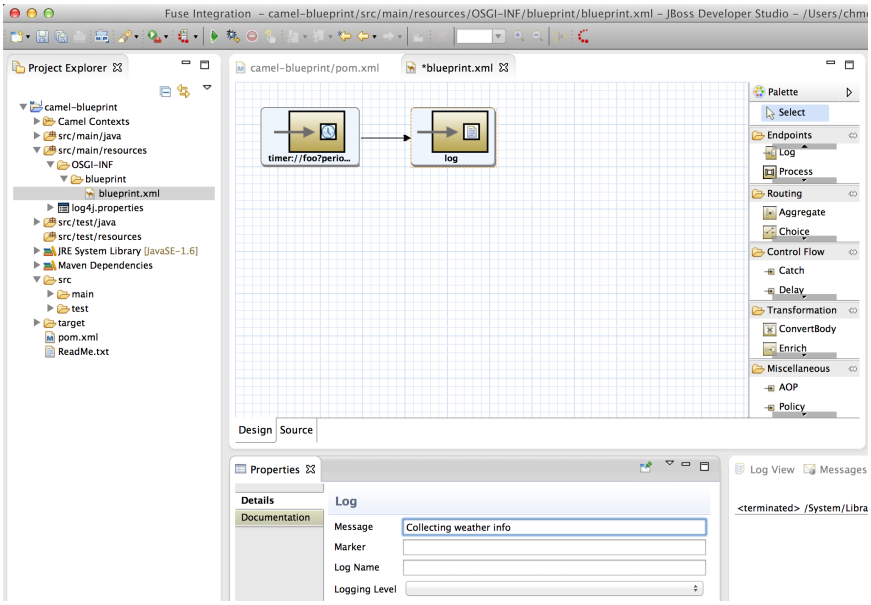

Figure 24. Log Processor

7. Define cities for weather data collection:
   a. Drag the **SetBody** processor from the palette (under **Transformation**), and drop it on the workspace.
   b. Set the **Language** to `simple`.
   c. In **Expression**, enter `Paris,London,Brussels` as the cities.
   d. Link this processor to the **Log** endpoint.
   e. Select **File** → **Save**.

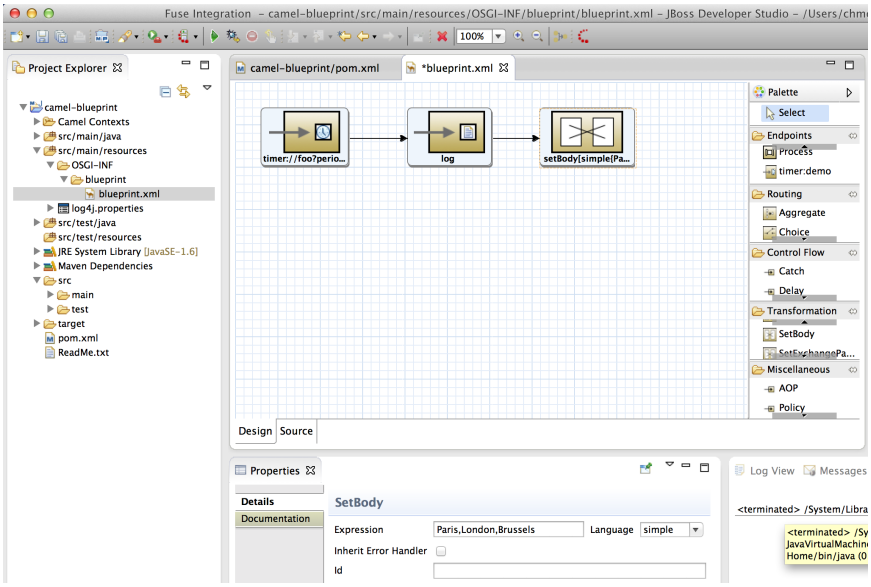

Figure 25. SetBody Processor

8. Split the body content:
   a. Drag the **Split** EIP pattern from the palette (under **Routing**), and drop it on the workspace.
   b. Set the **Language** to `groovy`.

c. In **Expression**, enter `request.body.split(',')` to split the list of cities with the `,` separator.

d. Select **File** → **Save**.

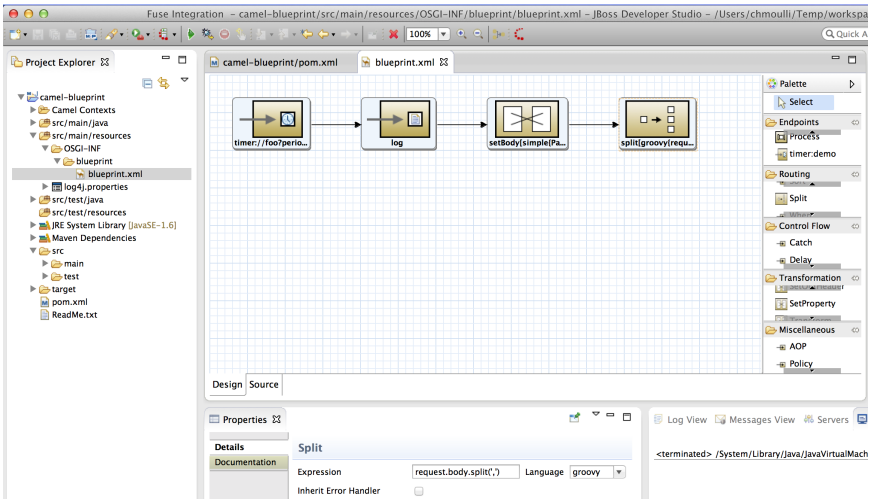

Figure 26. Split Route

9. Set a header that uses the city name:

a. Drag **SetHeader** from the palette (under **Transformation**), and drop it on the workspace.

b. Set the **Header Name** to `CamelWeatherLocation`.

c. Set the **Language** to `simple`.

d. In **Expression**, enter `${body}`.

e. Select **File** → **Save**.

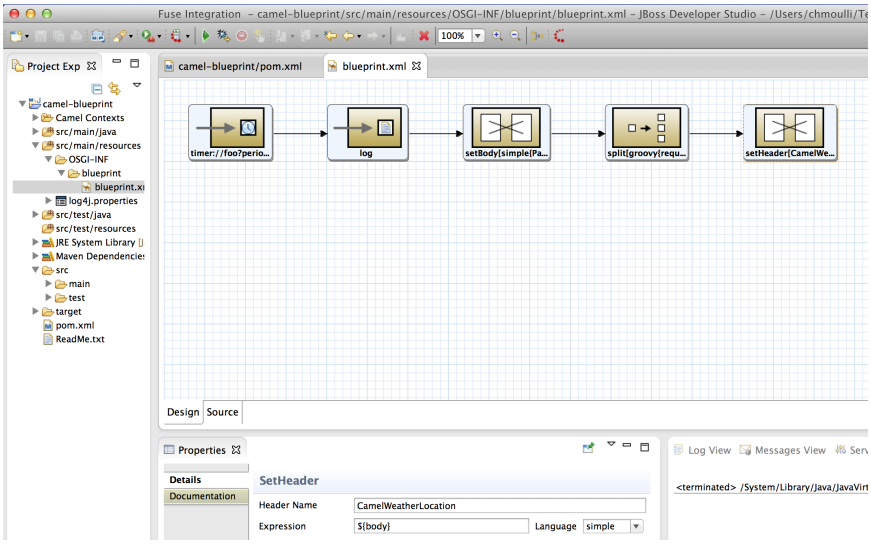

Figure 27. SetHeader

10. Add two more components:

a. Drag and drop a new **Generic** and set the **Uri** of the weather component to `weather:foo?period=1d&units=METRIC&mode=JSON`.

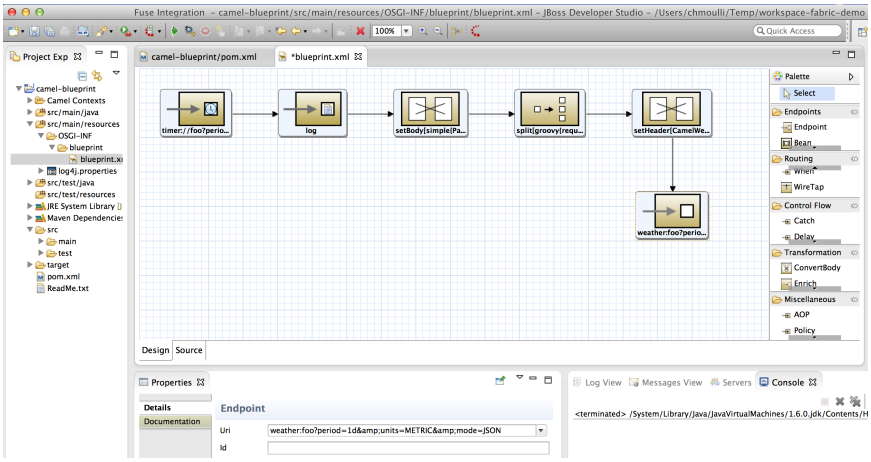

Figure 28. Set Endpoint Uri

b. Add a **Log** endpoint to log the information received by the weather station.

c. Improve the log output by using a **SetBody** property with these values:

- **Language**: `groovy`
- **Expression**: `groovy.json.JsonOutput.prettyPrint(request.body)`

d. Select **File** → **Save**.

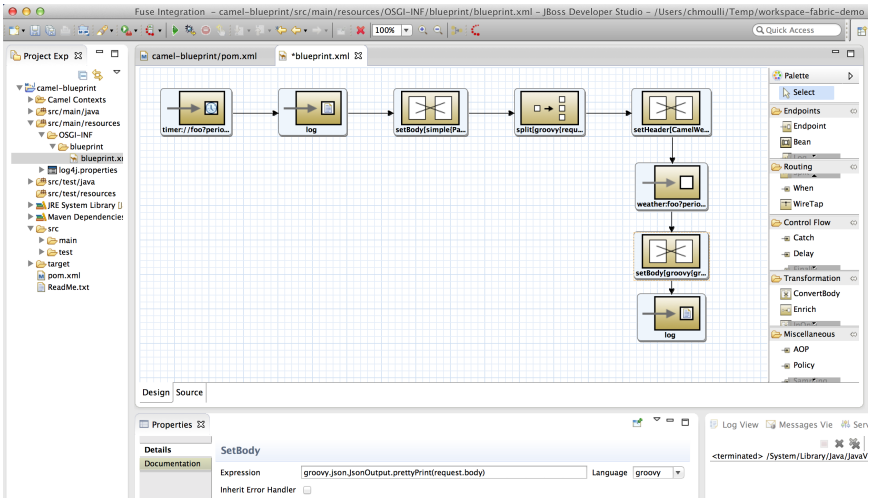

Figure 29. Enhance Log Output

11. Before you deploy the project in the Fuse Fabric environment, test the project locally in **JBoss Developer Studio**:

a. In **Project Explorer**, expand `src/main/resources/OSGI-INF/blueprint`.

b. Right-click the `blueprint.xml` file.

c. Select **Run as** → **Local Camel Context**.

The **Console** tab appears and displays the execution log for the Camel Maven plug-in.

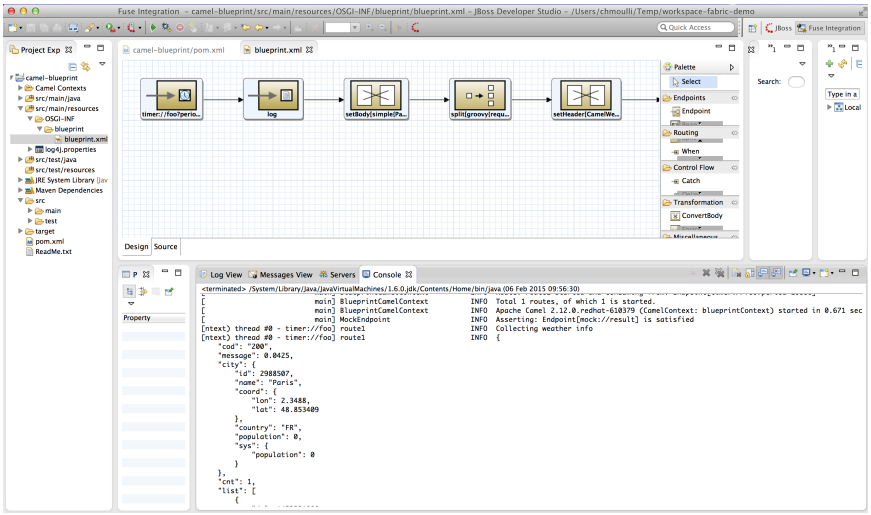

Figure 30. Console Tab With Execution Log

1.6. Deploy Your New Camel Project

In this exercise, you deploy the Camel project you just created into the Fabric server using the Fabric8 Maven plug-in.

First, you will make some modifications to deploy your project into a new profile and to document what you did for this application by adding a `Readme.md` file to the existing project. Next, you will include the Fabric Maven plug-in and define the properties for the profile, its parent, the features to deploy, and the Fabric server where it should be deployed.

1. Create a new folder under `src/main`:
   a. Right-click the `main` folder and select **New** → **Folder**.
   b. Name the folder `fabric8`.
   c. Click **Finish**.
2. Create the `Readme.md` file:
   a. Right-click the new `fabric8` folder and select **New** → **File**.
   b. Name the file `Readme.md`.
   c. Click **Finish**.
   d. Open the `Readme.md` file by selecting **Open with** → **Text editor**.
   e. Enter the following text to document your project:

```
# RedHat GPE - Weather Forecasting Application

This is a _Weather Forecasting Application_ designed using the integration
framework : Apache Camel with the component _camel-weather_ performing REST
calls against the weather provider _http://openweathermap.org/_ to collect
forecasting information about some cities.
```

   f. Save the file.

3. Edit the configuration file:
   a. Open the `pom.xml` file.
   b. In the `<plugins/>` section, add the following Fabric8 Maven plug-in information after the `camel-maven-plugin`:

```
<plugin>
   <groupId>io.fabric8</groupId>
   <artifactId>fabric8-maven-plugin</artifactId>
   <version>1.1.0.CR5</version>
</plugin>
```

   c. Add the following `<configuration>` section to specify the profile's name, parent, and features to deploy:

```
<configuration>
 <profile>gpe-exercise-forecastingapplication</profile>
 <parentProfiles>feature-camel</parentProfiles>
 <features>camel-groovy camel-weather</features>
</configuration>
```

   d. Change the name of the project from `<name>A Camel Blueprint Route</name>` to `<name>GPE :: Exercise :: Forecasting Application</name>`.
   e. Modify the OSGI metadata for the `maven-bundle-plugin` by modifying the code below. This will accomplish the following:
      - Import the packages needed to process the Groovy expression.
      - Rename the `Bundle-SymbolicName` to use your `camelweather-demo` project.
      - Use the `<DynamicImport-Package>` option so that Groovy can generate new classes on the fly using the `script2233445566` convention, and then allow the bundle's classloader to import them.

```
<instructions>
 <Bundle-SymbolicName>gpe-exercise-forecasting-application</Bundle-
SymbolicName>
 <Import-Package>groovy.lang,*</Import-Package>
 <DynamicImport-Package>*</DynamicImport-Package>
</instructions>
```

   f. Provide the server's URL to the Fabric8 Maven plug-in by adding a new property to the `<configuration/>` tag for `fabric8-maven-plugin`. This tag must point to your OpenShift application. Here is an example, but you must replace the

hostname shown here with your hostname:

```
<configuration>
  ...
  <jolokiaUrl>http://demo-fuse.apps.ose.opentlc.com/jolokia</jolokiaUrl>
</configuration>
```

g. Save the file.

4. Validate the hostname URL by checking the response returned by the **jolokia** servlet:

Figure 31. Jolokia Servlet

5. Deploy the profile:
   a. Declare the login/password used to access your Fuse Fabric server by entering a `<server>` tag in the Maven settings configuration as described in setup the login / password.
   b. Make sure the **Fuse Fabric** server is started and the **Fabric Ensemble** server is running.
   c. Open a DOS/Unix terminal within this project, and deploy the profile to the Fuse Fabric server with the `mvn fabric8:deploy` command.
6. Deploy the project:
   a. Switch to the **JBoss Fuse Management Console**.
   b. On the **Wiki** tab, select the Wiki that the profile created.

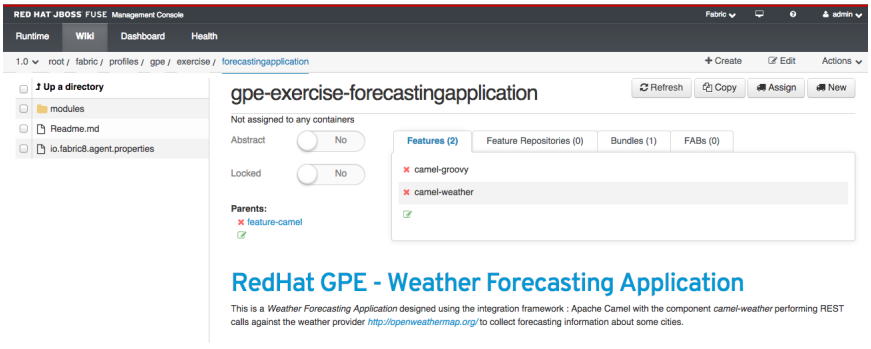

Figure 32. Wiki Created by Profile

c. On the **Runtime** tab, deploy the project into a new `forecasting` container.

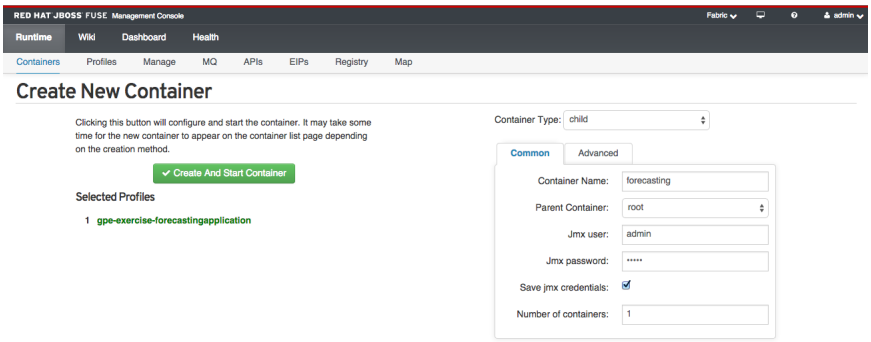

Figure 33. Runtime Tab - Containers View

7. Verify deployment:

a. On the **Camel** tab, open the `forecasting` container and confirm that the Camel route is deployed and collecting weather forecasts.

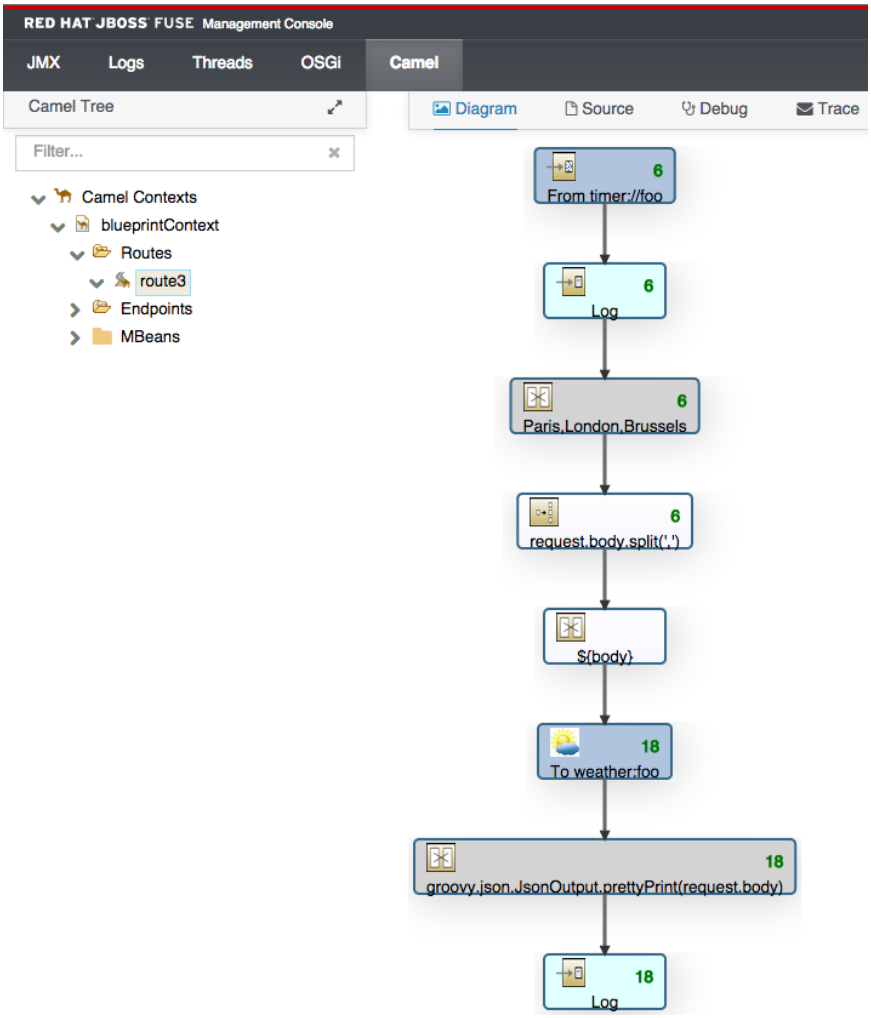

Figure 34. Camel Tab - Diagram View

b. On the **Log** tab, review the weather information collected for `Paris, London, ...`.

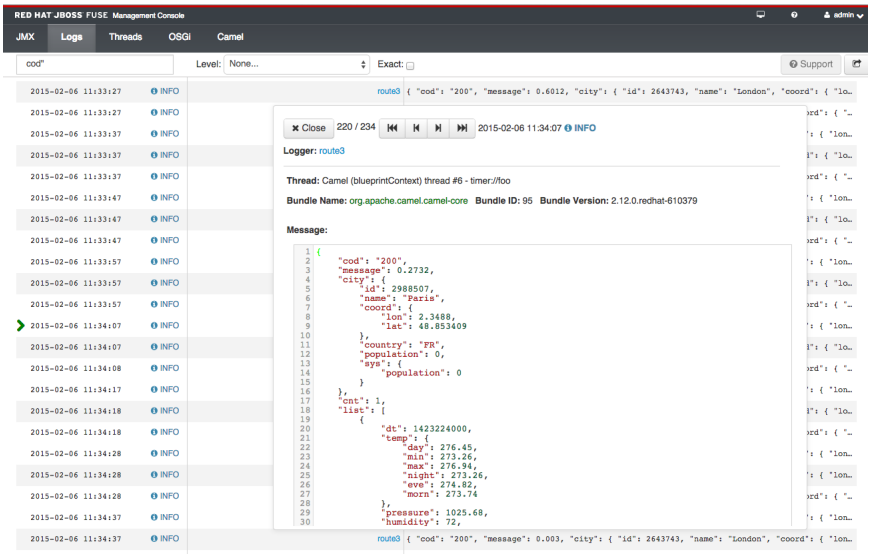

Figure 35. Log Tab

1.7. Migrate and Rollback a Camel Project

In this exercise, you modify the project you created in JBoss Developer Studio and redeploy it to the same container, but to achieve this goal, you use the rolling upgrade and rollback feature of the Fuse Fabric technology. This means that you create a new 1.1 version of your profile and upgrade your container to that new version.

When you create a new profile version, a new Git branch is instantiated by the Fabric server, the profiles are cloned, and you can modify the content (features, bundles, properties, and so on). This new branch enables you to migrate or rollback to an existing application. In other words, if after making a set of changes you are not happy with the modifications, you can rollback to the previous profile version.

1. Edit the existing route:
   a. Open **JBoss Developer Studio** and the workspace you created earlier in this lab.
   b. In **Project Explorer**, open the `fabric-forecasting-application` project.
   c. Expand the `src/main/resources/OSGI-INF/` directory and edit the Apache Camel `blueprint` route.
   d. When the **Fuse Integration** editor appears, select the `setBody` node on the diagram.
   e. In the **Expression** field, add `New-York,Tokyo,Moscow` to the list of cities.
   f. Save the route.
2. Edit the configuration file:
   a. Edit the `pom.xml` file to change the `<version>` number for this artifact to `1.1`.

```
<?xml version="1.0" encoding="UTF-8"?>
<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">
    ...
 <groupId>com.redhat.gpe</groupId>
 <artifactId>fabric-forecasting-application</artifactId>
 <packaging>bundle</packaging>
 <version>1.1</version>
```

b. Because you want to migrate your project to a newer version of your Fuse Fabric profile, you also need to add a `<version>1.1</version>` tag to the `<configuration>` section for `fabric8-maven-plugin` that specifies the version you want to use:

```
<plugin>
 <groupId>io.fabric8</groupId>
 <artifactId>fabric8-maven-plugin</artifactId>
 <version>1.1.0.CR5</version>
 <configuration>
  <profile>gpe-exercise-forecastingapplication</profile>
  <parentProfiles>feature-camel</parentProfiles>
  <features>camel-groovy camel-weather</features>
  <version>1.1</version>
  ...
```

When you use the `<version/>` tag, the Fabric8 Maven plug-in calls the Fuse Fabric server and requests that a new version number be created (if the version number does not match the current version number). Behind the scenes, a new Git branch is created, and the existing profiles are cloned to that new branch—in this case, 1.1.

c. Save the `pom.xml` file.

3. Redeploy the profile:

a. Deploy the profile using the `mvn fabric8:deploy` command.

b. Review the Maven log and observe that `version: 1.1` was used to populate this new profile.

```
[INFO] About to invoke mbean io.fabric8:type=ProjectDeployer on jolokia URL:
http://localhost:8181/jolokia with user: admin
[INFO] Result: DeployResults{profileUrl='null', profileId='gpe-exercise-
forecastingapplication', versionId='1.1'}
[INFO] Uploading file Readme.md to invoke mbean io.fabric8:type=Fabric on
jolokia URL: http://localhost:8181/jolokia with user: admin
```

```
[INFO] Performing profile refresh on mbean: io.fabric8:type=Fabric version: 1.1
profile: gpe-exercise-forecastingapplication
```

4. Migrate the profile in JBoss Developer Studio:
   a. Open the **JBoss Fuse Management Console** for your **Fuse Fabric** server.
   b. On the **Runtime** tab, select the **Manage** view.
   c. On the left part of the panel, use the **Filter** field to select your `forecast` profile.

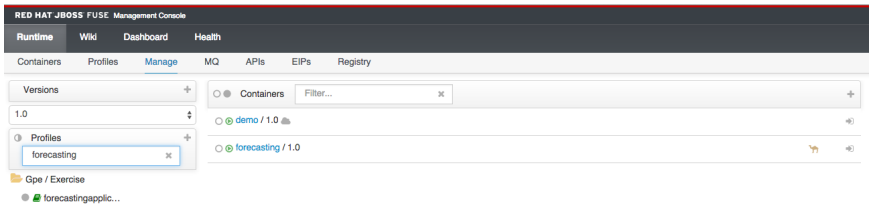

Figure 36. Runtime Tab - Manage View

   d. Use the **Versions** dropdown to switch from profile `1.0` to `1.1`.

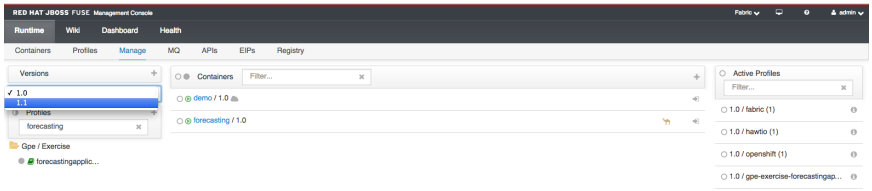

Figure 37. Version Dropdown

   e. Select `forecasting` as the target container, and then click the green `>` button to apply the `1.1` profile to that container.

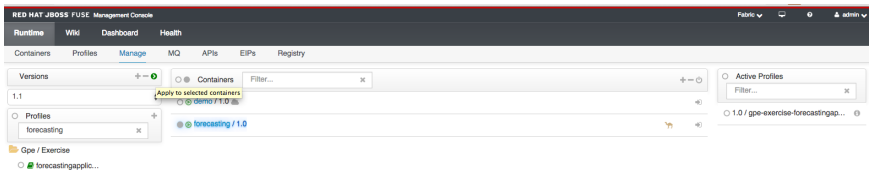

Figure 38. Target Container

   f. Wait a few moments for the container to migrate.

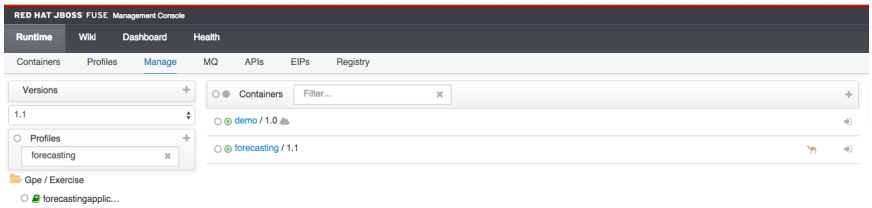

Figure 39. Migrated Container

5. Verify migration:
   a. Open the `forecasting` container.
   b. Examine the log and confirm that the new cities you added, such as `Tokyo`, appear in the forecast.

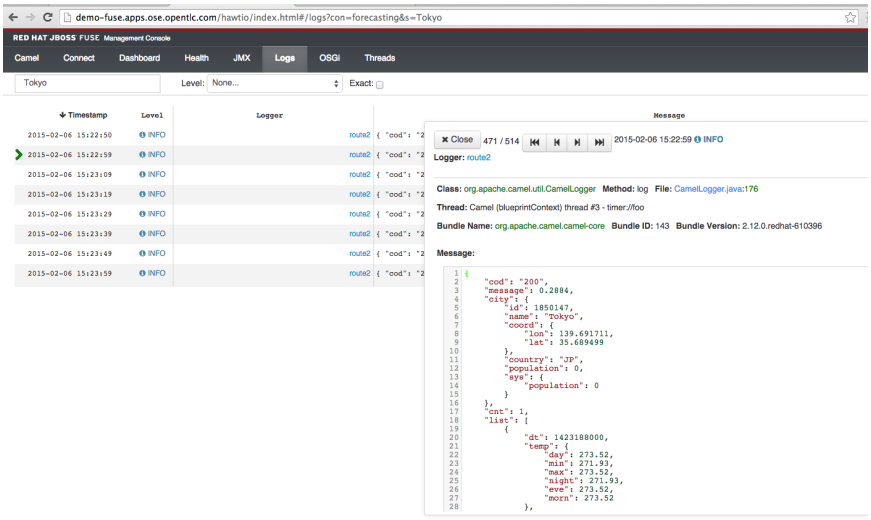

Figure 40. Log With New Cities

6. Do one of the following to rollback the profile to version 1.0:
   - On the **Runtime** tab, select the **Manage** view, and repeat the same steps that you used to migrate the profile, but select version `1.0` instead of version `1.1`.
   - On the **Runtime** tab, select the **Container** view, click **Migrate to**, and then select version `1.0`.

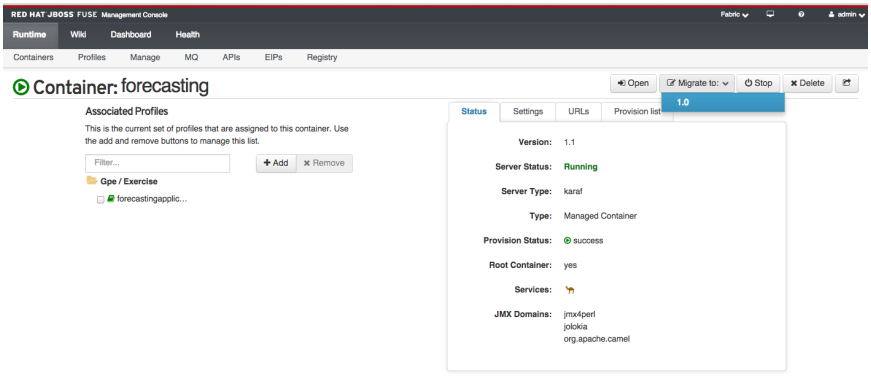

Figure 41. Runtime Tab - Container View

1.8. Virtualize Camel Endpoints and Load Balance Requests

In this final exercise, you use the virtualization and load balancing capabilities of the Fuse Fabric technology to make a Camel endpoint `transparent` between the `producer` and the `consumer`. In other words, the hostname and port number for the endpoint are registered into the Fabric repository (also known as the Apache Zookeeper server) under a logical group name. This group name is also used by the client to query the registry for the hostname and port number of the server to be connected (HTTP Web container). If there are multiple HTTP endpoints registered by Apache Camel in the Fuse Fabric registry, then the client is load balanced between the different servers.

1. Open the **JBoss Fuse Management Console** for your Fuse Fabric server.
2. On the **Wiki** tab, navigate to the `example/camel` profile.
3. Select the `cluster.server` profile.

4. Review the definition of the Apache Camel route and observe the following:
    - There are different beans defined.
    - The bean definitions are important because they allow you to register your endpoint with Fabric.
    - The bean definitions also externalize the port number using the OSGI `propertyPlaceholder` mechanism.
5. Review the following explanation of the different entries in the `blueprint.xml` file.

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:cm="http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.0.0"
           xsi:schemaLocation="
           http://www.osgi.org/xmlns/blueprint/v1.0.0
http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd">

    <!-- osgi blueprint property placeholder -->
    <!-- 1. The property placeholder will help us to lookup about the properties :
bind.address, port into the OSGI Config Admin service.
           These properties are externalized and not hardcoded within the route
definition -->
    <cm:property-placeholder id="myConfig" persistent-
id="io.fabric8.examples.camel.loadbalancing.server"/>

    <!-- 2. Lookup into the OSGI Service registry to retrieve the OSGI Curator
Service which is a Zookeeper client calling the Zookeeper server running into the
Fuse Fabric Server -->
    <reference id="curator"
interface="org.apache.curator.framework.CuratorFramework"/>

    <!-- 3. Instantiate the Fabric Camel Component with the reference of the
curator client -->
    <bean id="fabric-camel" class="io.fabric8.camel.FabricComponent">
        <property name="curator" ref="curator"/>
    </bean>

    <!-- 4. Route exposing a Jetty Web Container and returning as response to the
client a message
           with a header - karaf.name which is the karaf container name
    -->
    <camelContext id="camel" trace="false"
xmlns="http://camel.apache.org/schema/blueprint">
        <!-- using Camel properties component and refer to the blueprint property
placeholder by its id -->
        <propertyPlaceholder id="properties" location="blueprint:myConfig"
                             prefixToken="[[" suffixToken="]]"/>

        <route id="fabric-server">
            <from uri="fabric-camel:cluster:jetty:http://[[bind.address]]:
[[port]]/fabric"/> // HERE IS WHERE THE JETTY WEB CONTAINER ENDPOINT WILL BE
REGISTERED WITH THE GROUPNAME CLUSTER
```

```
            <log message="Request received : ${body}"/>
            <setHeader headerName="karaf.name">
                <simple>${sys.karaf.name}</simple>
            </setHeader>
            <transform>
                <simple>Response from Fabric Container</simple>
            </transform>
        </route>
    </camelContext>

    <!-- 5. Lookup into the OSGI Service registry to retrieve the Apache Camel
Component Resolver service used to do a lookup in order to load the
            JettyComponent from the classloader of the bundle exposing it -->
    <reference interface="org.apache.camel.spi.ComponentResolver" filter="
(component=jetty)"/>

</blueprint>
```

6. Review the `cluster.client` profile and the definition of the Apache Camel route:

```
<!-- 1. Global error handler managing any kind of exceptions returned by a
processor to the component issuing the creation of the exchange.
        The property redeliveryPolicyRef is defined with the next bean definition -
->
<errorHandler id="errorHandler" redeliveryPolicyRef="redeliveryPolicy"/>

<!-- 2. This is the delivery policy that the error handler & camel will use to
retry to call the
        processor where the exception occurs. After -->
<redeliveryPolicyProfile id="redeliveryPolicy" maximumRedeliveries="3"
redeliveryDelay="5000"
                         retryAttemptedLogLevel="WARN"/>

<!-- 3. The fabric camel component is used to do a lookup into the Zk registry to
find the hostname/port number to be used to call the HTTP Server -->
<route id="fabric-client" errorHandlerRef="errorHandler">
    <from uri="timer://foo?fixedRate=true&amp;period=1000"/>
    <setBody>
        <simple>Hello from Fabric Client to group "Cluster"</simple>
    </setBody>
    <to uri="fabric-camel:cluster"/>
    <log message=">>> ${body} : ${header.karaf.name}"/>
</route>
```

7. Deploy the `cluster.server` profile into two containers managed by Fuse:
   a. Create two containers: `cluster1` and `cluster2`.
   b. Assign the `cluster.server` profile to the new containers.
   c. On the **Runtime** tab, select the **Container** view.

d. Confirm that the new containers were created and provisioned.

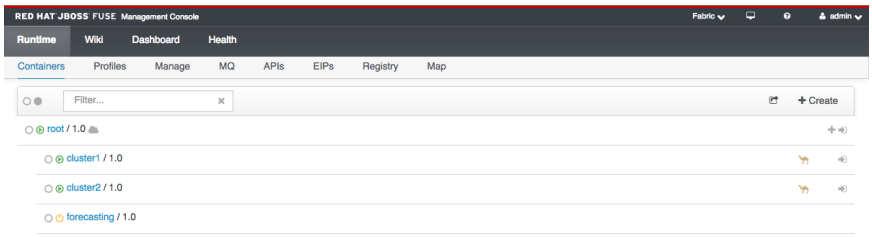

Figure 42. Runtime Tab With New Containers

8. Confirm that the endpoints of the routes are registered in the Fuse Fabric registry:
   a. On the **Runtime** tab, select the **Registry** view.
   b. Expand the `/fabric/registry/camel/endpoints/cluster` tree.
   c. Confirm that you have two endpoints registered with these addresses:
      - `http://machine:8282/fabric`
      - `http://machine:8383/fabric`

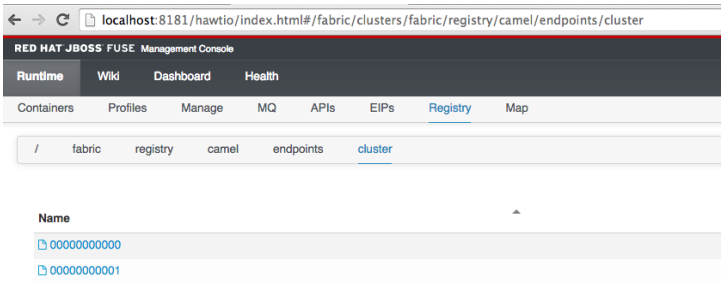

Figure 43. Registry Cluster View

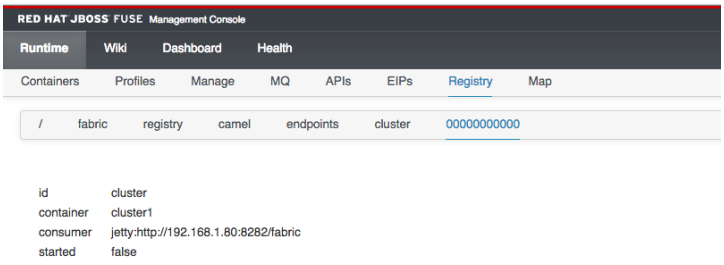

Figure 44. Registry - Machine 8382

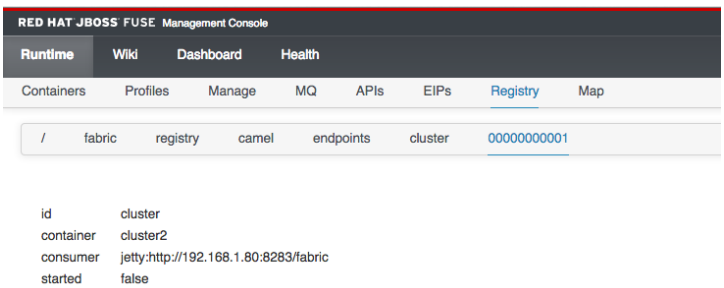

Figure 45. Registry - Machine 8382

9. With the servers running, use the `client` profile to create the container managed by Fuse.

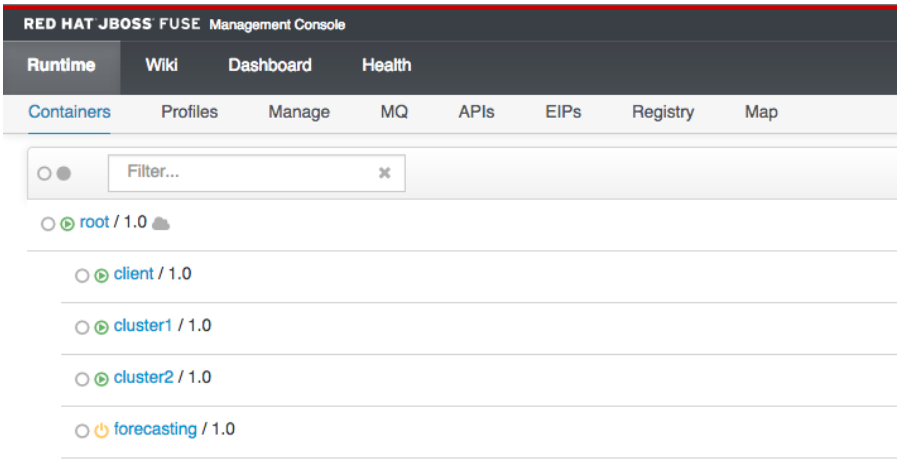

Figure 46. Create Container From Client Profile

10. Verify load balancing:
    a. Open the `client` container.
    b. Click the **Log** tab.
    c. Verify that the client received requests from the two servers.

    The requests are now load balanced and are received randomly from one of the two servers.

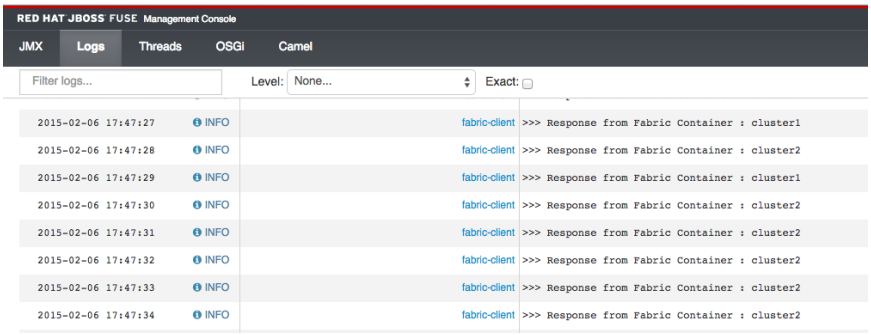

Figure 47. Log for Load-Balanced Servers

Last updated 2015-11-12 12:04:12 EST