Table of Contents

1. Modularity Lab

Goals

- Package and deploy a simple `HelloBean` in a JAR file
- Consume the `HelloBean` through OSGi Blueprint
- Develop an OSGi service and use it using an OSGi *client* bundle
- Consume the OSGi service from within an Apache Camel route
- Design integration tests using OSGi testing frameworks

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

1.1. Prepare Your Lab Environment for These Exercises

To use the code in these exercises, you must first do the following:

- Connect to the `demo` application on OpenShift
- Clone the Git project that contains the exercises to your local Git repository environment
- Use Maven to compile the application
- Use Maven to deploy the application into a JBoss Fuse server on OpenShift
- Set up some JBoss Fuse containers and deploy the lab projects into these containers

To complete these tasks, follow these steps:

1. Use SSH to connect to your OpenShift account as described here.

   Once you complete this initial setup, you can log in to OpenShift from JBoss Developer Studio or your DOS/Unix terminal.

2. Clone the project and build the code:

   a. 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
```

b. Edit the `settings.xml` file to add `<servers><server></server></servers>` configuration tags that contain the credentials you use to log into your JBoss Fuse Fabric Server running on OpenShift as well as the server id.

```
    <server>
      <id>fabric</id>
      <username>username</username>
      <password>password</password>
    </server>
```

c. Clone the `camel-labs` project:

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

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

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

3. Use the following Maven command to build the application:

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

4. Deploy the exercises:

In the module 1 lab exercises, you created a JBoss Fuse server called `demo`, and you deployed a `camel twitter` application to this server. If this OpenShift gear is still available, reuse as part of this lab exercise. Deploy the compiled code to the local Maven repository. This code is used by JBoss Fuse to provision and deploy the artifacts into multiple containers. To do this, you need your username, your 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 code artifacts into the Maven repository of JBoss Fuse.

a. 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/uploa
d -DskipTests=true -s $OPENSHIFT_DATA_DIR/settings.xml
```

b. Before deploying new Maven artifacts, (for example, after pushing new changes to the Git repository), remove all content of this directory and then restart the `demo` application.

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

1.2. Create a Plain JAR

In this exercise, you create a `HelloBean` class containing a field/property message that passes a text message whenever the bean is initialized or destroyed. You will not directly instantiate the bean, but it will be instantiated by the Inversion of Control (IoC) framework (in Blueprint).

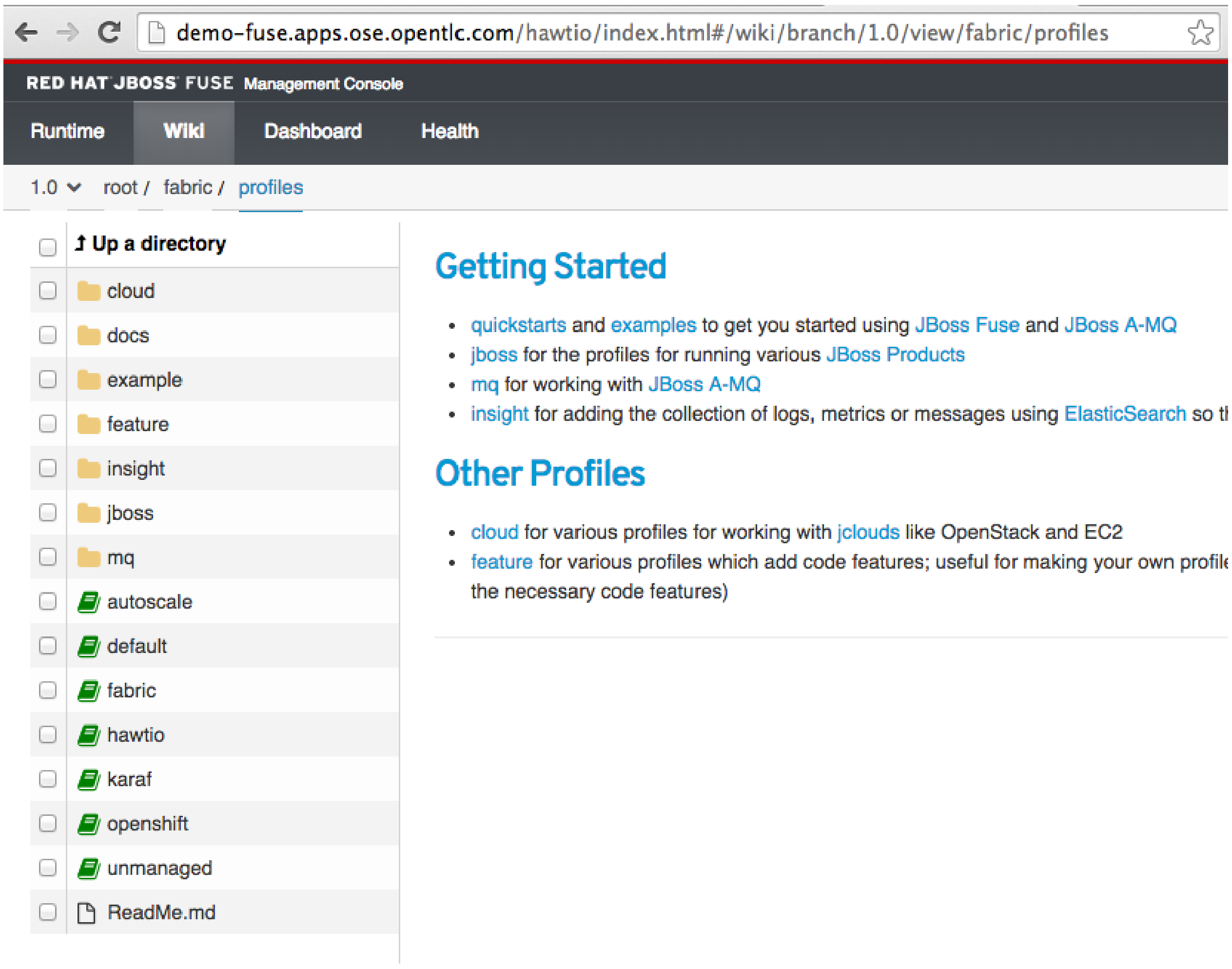
> As an alternative, you can complete this exercise using the Spring IoC framework instead of a Blueprint framework.

1. In **JBoss Developer Studio**, use **Project Explorer** to open the `osgi-plain-jar` project that you imported from the `lab_assets` directory.
2. Review the code skeleton and notice the `HelloBean` class, the Spring XML file and the Blueprint XML file.
3. Modify the `HelloBean` class defined within the `com.redhat.gpe.training.bean` package:
   a. Add the private string field `message`.
   b. Add its `setter` method. No `getter` is required.
   c. Create the following public methods which do not return any objects:
      - `init()`
      - `destroy()`
   d. In each of your new methods, add a `LOG.info()` method that generates the following text message when the bean is initialized or destroyed: `"Initializing/Destroying hello bean with message = '" + message + "'"`.

   > You are done designing the `HelloBean` class. Next you design the Blueprint XML file, which is used by the OSGi container to set up containers for singleton instances of `HelloBean`

4. Modify the Blueprint XML file:
   a. Open the `blueprint-context.xml` file within the `deploy` directory.
   b. Add a `<bean>` tag with the attribute id `helloBean`.
   c. Define the attributes `init` and `destroy`, which refer to your two new methods, for the `HelloBean` class.
   d. Append a `<property>` tag to the `message` field and set the value to: `"Hello Students and welcome to OSGi World"`.
   e. Close the `</bean>` tag.

5. Use the `mvn clean install` command to compile the project and produce a JAR file.
6. Create a JBoss Fuse container and deploy the project into it:
   a. Connect to the server at `https://broker00.ose.opentlc.com` using the log in credentials you defined in module 1 of this course.
   b. Open the **JBoss Fuse Management Console** for the `demo` JBoss Fuse server.
   c. Click the **Wiki** tab.



Getting Started

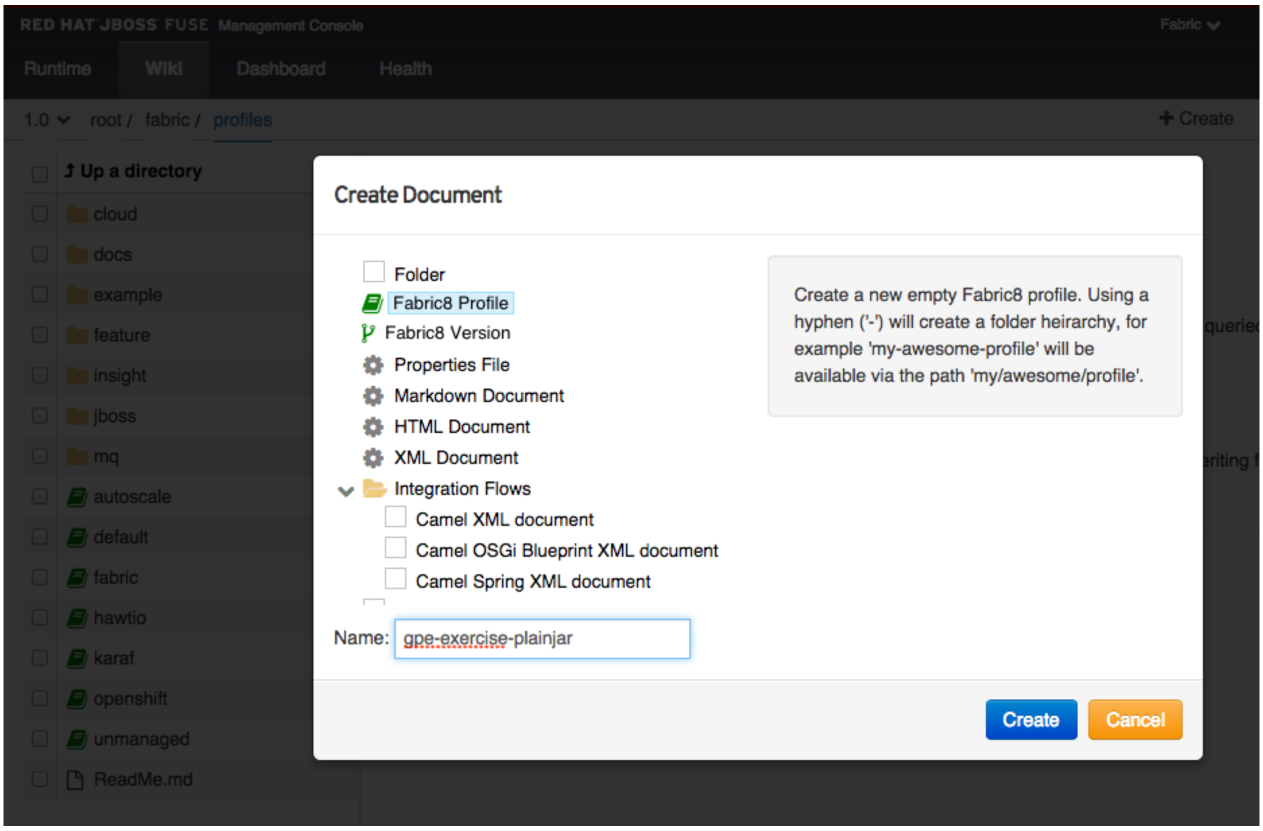
- quickstarts and examples to get you started using JBoss Fuse and JBoss A-MQ
- jboss for the profiles for running various JBoss Products
- mq for working with JBoss A-MQ
- insight for adding the collection of logs, metrics or messages using ElasticSearch so tl

Other Profiles

- cloud for various profiles for working with jclouds like OpenStack and EC2
- feature for various profiles which add code features; useful for making your own profile the necessary code features)

Figure 1. JBoss Fuse Management Console - Wiki Tab

   d. Click **Create**, and then select **Fabric8 profile**.
   e. In the **Name** field, enter `gpe-exercise-plainjar`.
   f. Click **Create**.



Create a new empty Fabric8 profile. Using a hyphen ('-') will create a folder heirarchy, for example 'my-awesome-profile' will be available via the path 'my/awesome/profile'.

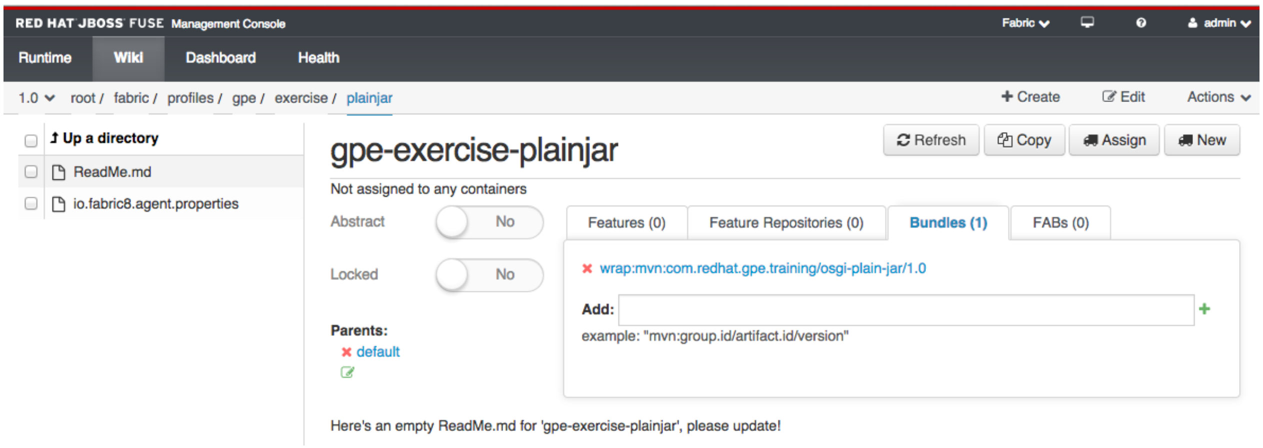
Name: gpe-exercise-plainjar

Figure 2. Create Plainjar Profile

g. After the profile is created, click the **Bundles 0** tab.

h. Click the `+` symbol and add `wrap:mvn:com.redhat.gpe.training/osgi-plain-jar/1.0` as the JAR location (in Maven-friendly syntax).

In this instruction, the Maven `wrap` plug-in converts the JAR into an OSGi bundle.



gpe-exercise-plainjar

Not assigned to any containers

Abstract No

Locked No

Parents: default

wrap:mvn:com.redhat.gpe.training/osgi-plain-jar/1.0

Add:

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

Here's an empty ReadMe.md for 'gpe-exercise-plainjar', please update!

Figure 3. Wrap Plain JAR

This Fabric8 profile will be used to provision and deploy the OSGi bundle into a Fuse Container.

7. Create a new container:

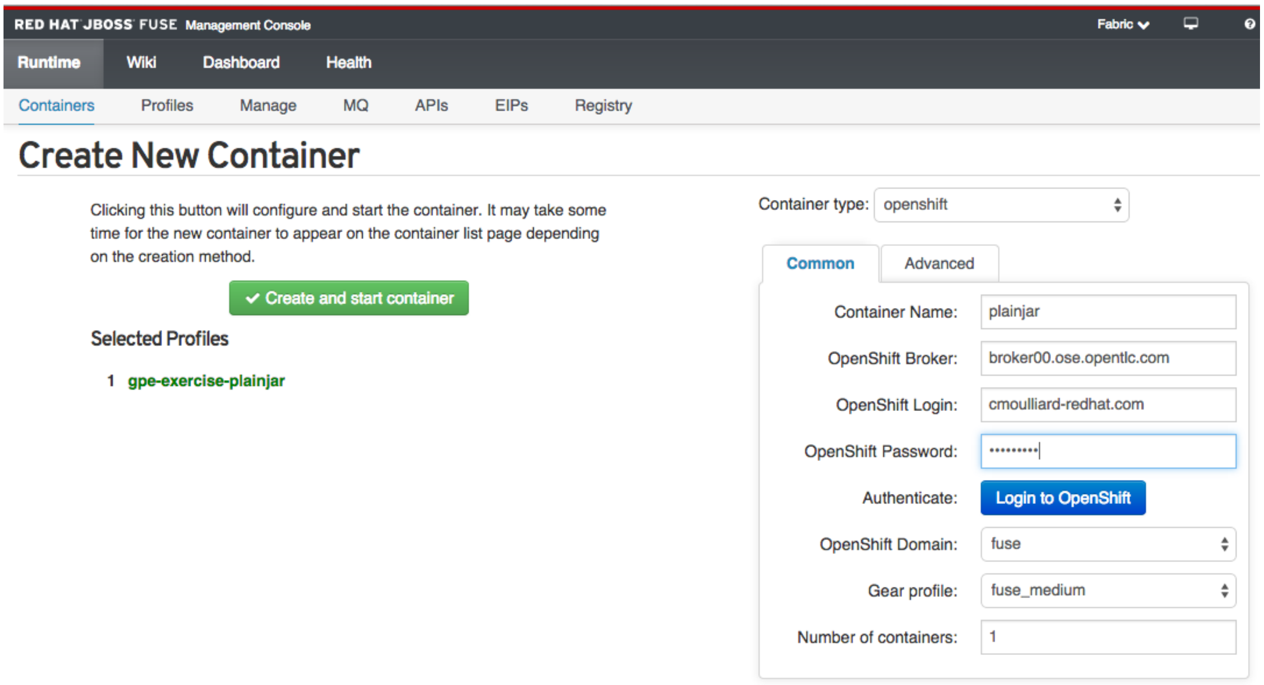
a. In the artifact editor, click **New**.

b. For the **Container Name**, enter `plainjar`.

c. Enter your OpenShift account credentials.

d. Select `Fuse` for the **OpenShift Domain**, and `fuse_medium` for the **Gear Profile**.

e. Click **Create and start container**.



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 gpe-exercise-plainjar

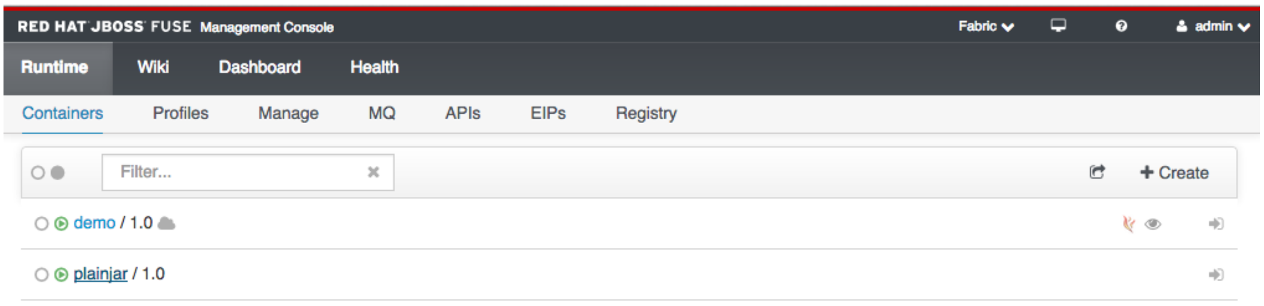
Container type: openshift

Container Name: plainjar
OpenShift Broker: broker00.ose.opentlc.com
OpenShift Login: cmoulliard-redhat.com
OpenShift Password: ••••••••
Authenticate:
OpenShift Domain: fuse
Gear profile: fuse_medium
Number of containers: 1

Figure 4. Create New Container

8. Verify your work:

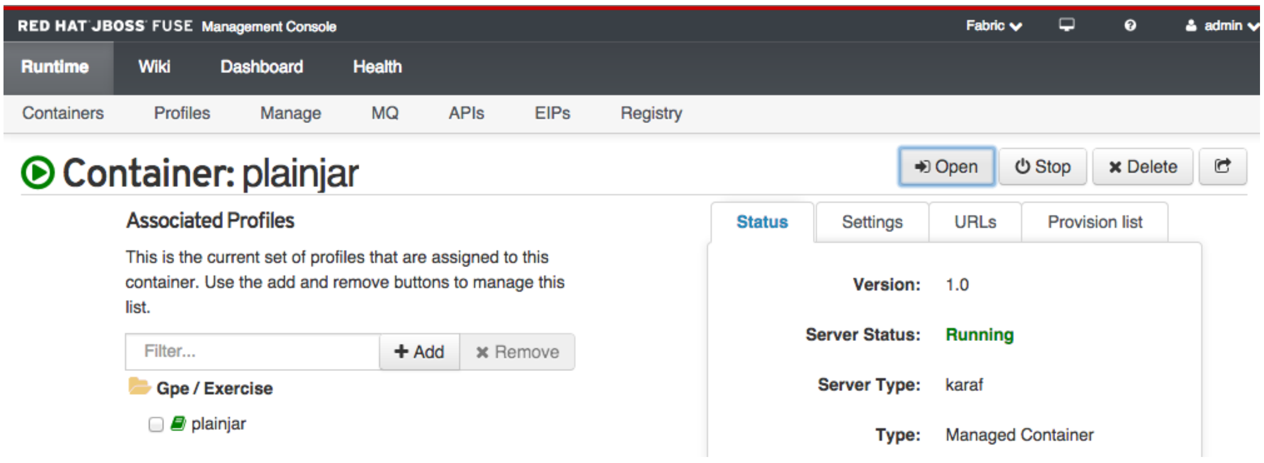
a. From the **Runtime** tab **Containers** view, click the `plainjar` container.



demo / 1.0
plainjar / 1.0

Figure 5. Connect Plainjar Container

b. Click **Open** to connect to the Fuse container.



Container: plainjar

Associated Profiles

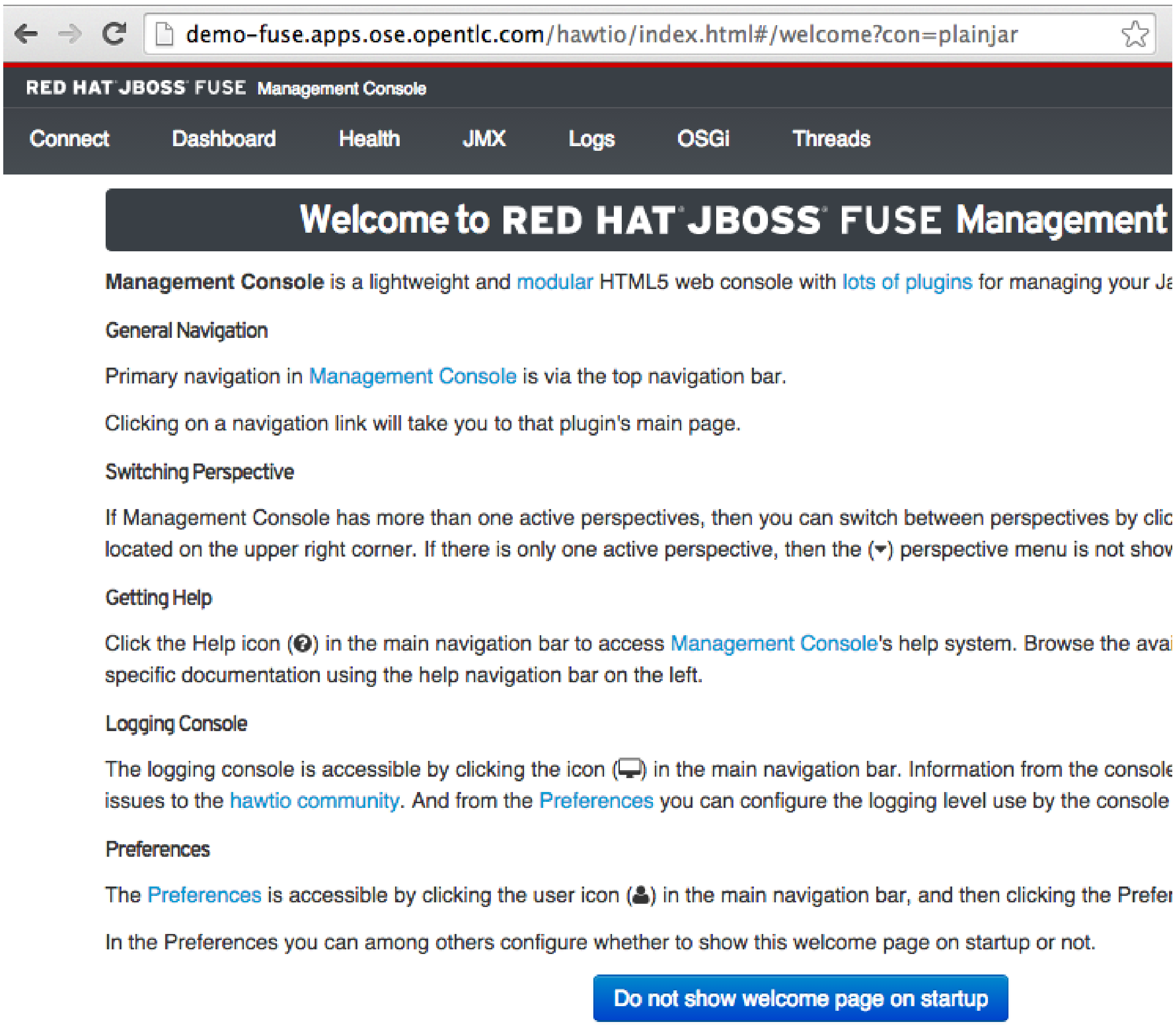
This is the current set of profiles that are assigned to this container. Use the add and remove buttons to manage this list.

Gpe / Exercise
plainjar

Version: 1.0
Server Status: Running
Server Type: karaf
Type: Managed Container

Figure 6. Plainjar Container

c. Confirm that a new web page displays JBoss Fuse Management Console with tabs that include **OSGi**, **Logs**, **Threads**, **JMX**.



demo-fuse.apps.ose.opentlc.com/hawtio/index.html#/welcome?con=plainjar

RED HAT JBOSS FUSE Management Console

Connect Dashboard Health JMX Logs OSGi Threads

Welcome to RED HAT JBOSS FUSE Management

Management Console is a lightweight and modular HTML5 web console with lots of plugins for managing your Ja

General Navigation

Primary navigation in Management Console is via the top navigation bar.

Clicking on a navigation link will take you to that plugin's main page.

Switching Perspective

If Management Console has more than one active perspectives, then you can switch between perspectives by clic
located on the upper right corner. If there is only one active perspective, then the (▾) perspective menu is not show

Getting Help

Click the Help icon (?) in the main navigation bar to access Management Console's help system. Browse the avai
specific documentation using the help navigation bar on the left.

Logging Console

The logging console is accessible by clicking the icon (🖵) in the main navigation bar. Information from the console
issues to the hawtio community. And from the Preferences you can configure the logging level use by the console

Preferences

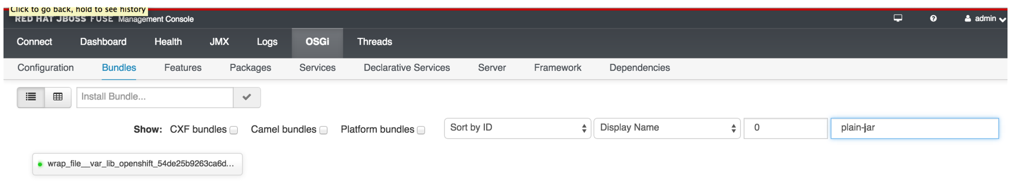
The Preferences is accessible by clicking the user icon (👤) in the main navigation bar, and then clicking the Prefer

In the Preferences you can among others configure whether to show this welcome page on startup or not.

Do not show welcome page on startup

Figure 7. JBoss Fuse Management Console - Tabs

d. Select **OSGi** → **Bundles**.



Click to go back, hold to see history

RED HAT JBOSS FUSE Management Console — admin

Connect Dashboard Health JMX Logs OSGi Threads

Configuration Bundles Features Packages Services Declarative Services Server Framework Dependencies

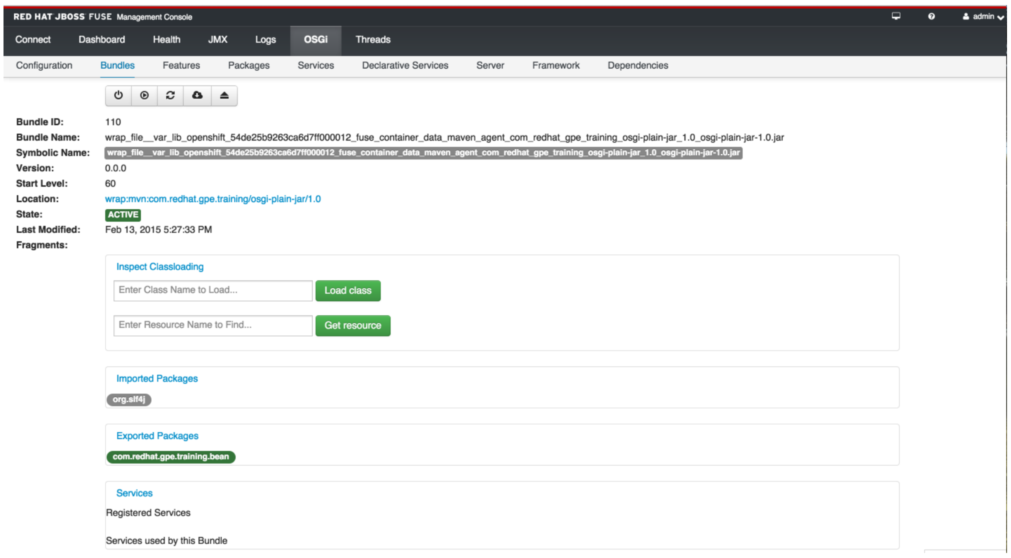
Install Bundle...

Show: CXF bundles ☐ Camel bundles ☐ Platform bundles ☐ — Sort by ID — Display Name — 0 — plain-jar

● wrap_file__var_lib_openshift_54de25b9263ca6d...

Figure 8. OSGi Tab, Installed Bundles, view 1

e. Confirm that the JAR/bundle was installed in the OSGi container.



Bundle ID:	110
Bundle Name:	wrap_file__var_lib_openshift_54de25b9263ca6d7ff000012_fuse_container_data_maven_agent_com_redhat_gpe_training_osgi-plain-jar_1.0_osgi-plain-jar-1.0.jar
Symbolic Name:	wrap_file__var_lib_openshift_54de25b9263ca6d7ff000012_fuse_container_data_maven_agent_com_redhat_gpe_training_osgi-plain-jar_1.0_osgi-plain-jar-1.0.jar
Version:	0.0.0
Start Level:	60
Location:	wrap:mvn:com.redhat.gpe.training/osgi-plain-jar/1.0
State:	ACTIVE
Last Modified:	Feb 13, 2015 5:27:33 PM
Fragments:	

Inspect Classloading

Enter Class Name to Load...

Enter Resource Name to Find...

Imported Packages

org.slf4j

Exported Packages

com.redhat.gpe.training.bean

Services

Registered Services

Services used by this Bundle

Figure 9. OSGi Tab, Installed Bundles, view 2

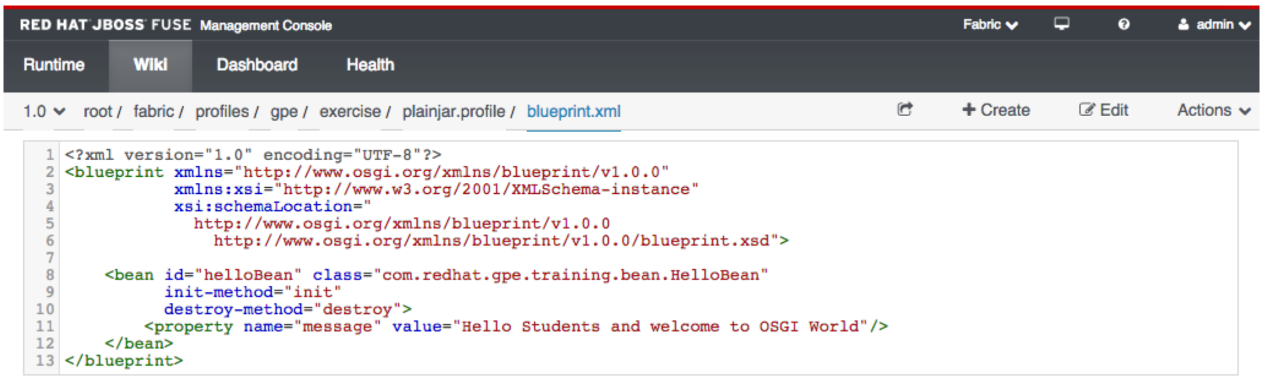
f. Click the **Logs** tab and confirm that the bean was not instantiated.

> Because you have only deployed the JAR containing the `HelloBean` class and no main class exists, `HelloBean` cannot be instantiated. In the next exercise, you create and deploy a Blueprint XML file that instantiates `HelloBean`.

1.3. Use Hot Deploy With Aries Blueprint

In this exercise, you deploy the Blueprint XML file into the OSGi container, which means a new bundle containing the file must be created. Whenever a bundle is deployed, the deployment tool generates an event that Fuse intercepts, and Fuse detects the technology that is being used, such as Blueprint, Spring, or CDI. Apache Karaf then uses the bundle’s classloader to create the appropriate runtime container for the bundled artifacts. In this exercise, you instantiate a Blueprint container which serves as the runtime environment for `HelloBean`.

1. Open the **JBoss Fuse Management Console** for the `demo` JBoss Fuse server.
2. Add an XML file to +plainjar.profile`:
3. On the **Wiki** tab, edit `plainjar.profile`.
   a. Click **Create**, and then select `XML document` from the list of artifacts.
   b. Name this file `blueprint.xml`.
   c. Copy and paste the content developed in the first exercise into this file.
   d. Save the file.

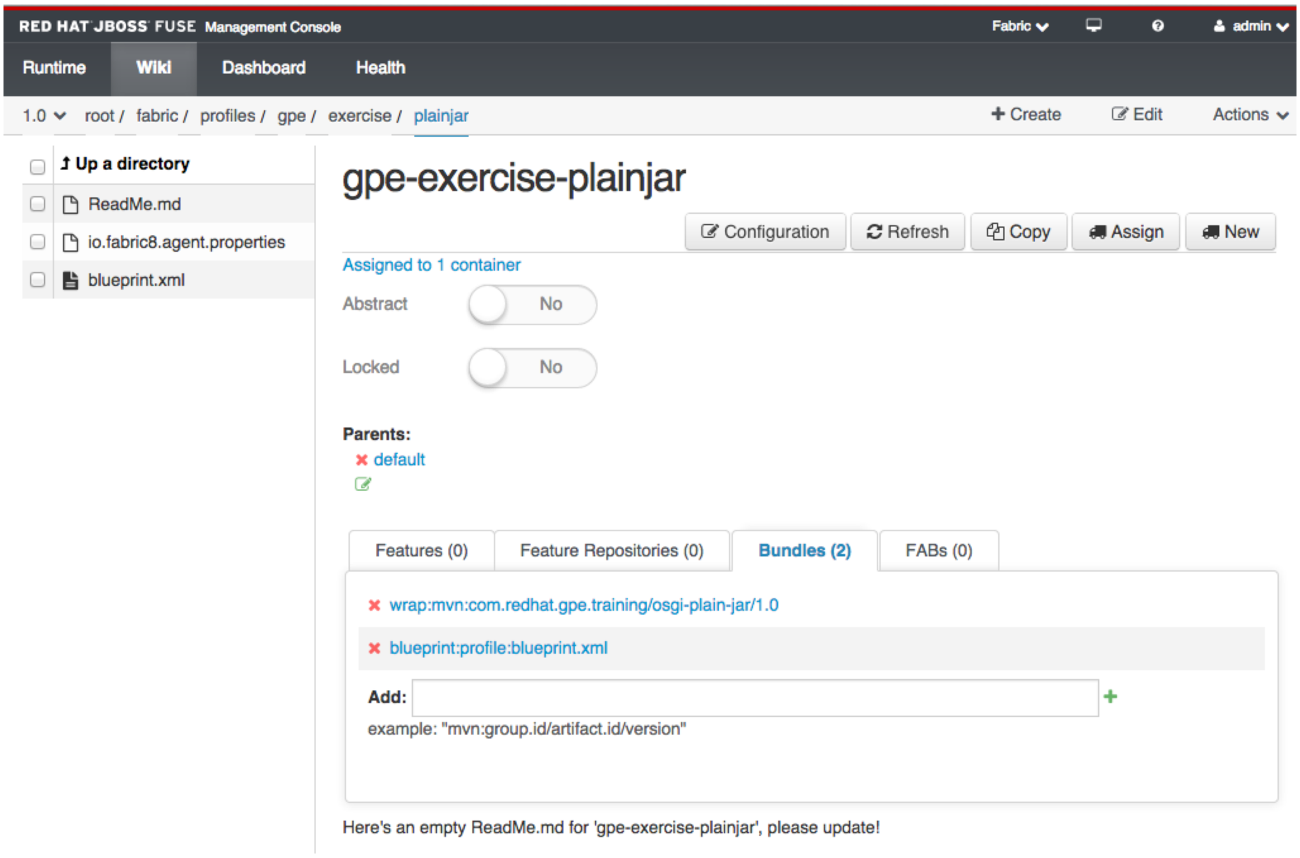

RED HAT JBOSS FUSE Management Console
Fabric
admin
Runtime
Wiki
Dashboard
Health
1.0
root / fabric / profiles / gpe / exercise / plainjar.profile / blueprint.xml
Create
Edit
Actions

```
<?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"
           xsi:schemaLocation="
             http://www.osgi.org/xmlns/blueprint/v1.0.0
               http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd">

    <bean id="helloBean" class="com.redhat.gpe.training.bean.HelloBean"
          init-method="init"
          destroy-method="destroy">
        <property name="message" value="Hello Students and welcome to OSGI World"/>
    </bean>
</blueprint>
```

Figure 10. Add Blueprint XML file

4. Click the **Bundles** tab, and enter `blueprint:profile:blueprint.xml` in the **Add** box to define the location of the blueprint file and associate it with this profile.



gpe-exercise-plainjar

Assigned to 1 container

Abstract No

Locked No

Parents:
default

wrap:mvn:com.redhat.gpe.training/osgi-plain-jar/1.0
blueprint:profile:blueprint.xml

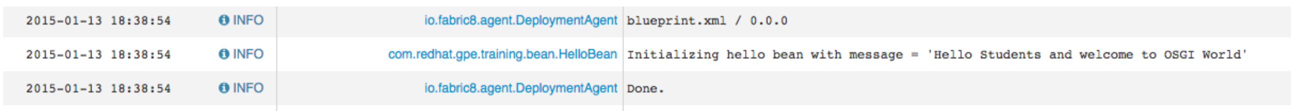
Add:
example: "mvn:group.id/artifact.id/version"

Here's an empty ReadMe.md for 'gpe-exercise-plainjar', please update!

Figure 11. Blueprint XML file

Whenever a profile is modified, JBoss Fuse determines whether the modification must be propagated to any of the containers associated with the profile, it defines the scope of these modifications (install, update, remove bundles, and so on), and then initiates the required modifications.

5. Return to your `plainjar` container and inspect the **Logs** tab to confirm that the bean was instantiated.



```
2015-01-13 18:38:54  INFO  io.fabric8.agent.DeploymentAgent     blueprint.xml / 0.0.0
2015-01-13 18:38:54  INFO  com.redhat.gpe.training.bean.HelloBean  Initializing hello bean with message = 'Hello Students and welcome to OSGI World'
2015-01-13 18:38:54  INFO  io.fabric8.agent.DeploymentAgent     Done.
```

Figure 12. Log - Plain Jar Container

1.4. Use the OSGi Service and a Client

The goal of this exercise is to create an OSGi Service using a `Greeter` Java interface as well as an OSGi bundle that serves as a client. The service will be registered with the OSGi Service registry, and the client will look up this service the registry. The object retrieved is a Java proxy object that the client bundle then uses to invoke the `GreeterImpl` service.

You will also configure the message used by the client and service to communicate with each other. The mechanism used is based on OSGi Config Admin and manages this information as a collection of key-value pairs. Each key-value is defined in a specific configuration file that you deploy in the OSGi container.

Here is the code implementation of the `GreeterImpl` service and the client that uses it:

GreeterImpl Service

```
@Component(name = GreeterImpl.SERVICE_PID)
@Service(Greeter.class)
public class GreeterImpl implements Greeter {

    // Logger
    private static final Logger LOGGER = LoggerFactory.getLogger(GreeterImpl.class);
    protected static final String SERVICE_PID = "greeterservice"; // This corresponds to the key
registered within the OSGi Config Admin registry

    // default response message
    @Property(name = "DEFAULT_RESPONSE", label="Default Response message", value = "Hi there from
the OSGi greeter service...")
    private String defaultResponse;

    @Activate // CALLED WHEN THE BUNDLE STARTS
    public void init(Map<String, ?> conf) throws Exception {
        defaultResponse = (String)conf.get("DEFAULT_RESPONSE"); // IF NO VALUE is RETRIEVED, THE
DEFAULT RESPONSE IS USED
        LOGGER.info("Greeter Service initialized with default response : \"" + defaultResponse +
"\"");
    }

    /**
     * Checks the incoming request message and returns an appropriate response message.
     *
     **/
    public String sayHello(String message) {
        ...
    }

    @Deactivate  // CALLED WHEN THE BUNDLE STOPS
    public void destroy() throws Exception {
        LOGGER.info("Shutting down yhe Greeter Service ...");
    }

}
```

GreeterClient

```java
@Component(name = GreeterClient.SERVICE_PID,immediate = true)
public class GreeterClient {

    private static final Logger LOGGER = LoggerFactory.getLogger(GreeterClient.class);
    protected static final String SERVICE_PID = "greeterclient";

    // OSGi Greeter service reference
    @Reference
    private Greeter greeterService;

    // default request message
    @Property(name="DEFAULT_REQUEST", label ="Default Request message", value = "I'm the Greeter
Client")
    private String defaultRequest;

     @Activate
    public void init(Map<String, ?> conf) throws Exception {
        defaultRequest = (String)conf.get("DEFAULT_REQUEST");
        LOGGER.info("Greeter Client Initialized with a default request : \"" + defaultRequest +
"\"");
        ...
    }

    /**
     * Utility method for invoking the OSGi Greeter service, with small delays for easier
demonstration.
     */
    private void invokeGreeterService(String message) throws Exception {
        ...
    }

    @Deactivate
    public void destroy() throws Exception {
        LOGGER.info("Shutting down the GreeterClient...");
    }

}
```

This exercise uses the Service Component Runtime (SCR) and its annotations as the framework to register and retrieve the OSGi service.

You do not have to use Spring or Blueprint to instantiate the bean. SCR automatically instantiates the bean via the `@Activate` or `@Deactivate` annotations, which are called when the bundle is started, stopped, or deployed. SCR also registers services, performs lookups, and retrieves key-value pairs from the OSGi config admin registry, which simplifies coding activity.

1. In **JBoss Developer Studio**, use **Project Explorer** to open the `osgi-service` and `osgi-client` projects.
2. Review the skeletal code and configuration files of the project:
   - `Greeter`

- `GreeterImpl`
- `GreeterClient`
- `osgi-client/etc/greeterclient.cfg`
- `osgi-service/etc/greeterservice.cfg`

> You can change the service message by editing each of these configuration files.

3. Deploy the project into a new JBoss Fuse container:
   a. Connect to the OpenShift Enterprise server at `https://broker00.ose.opentlc.com` using the credentials defined in Module 1.
   b. Open the **JBoss Fuse Management Console** for the `demo` JBoss Fuse server.
   c. Click the **Runtime** tab and select the **Containers** view.
   d. Click **Create** and enter the following.
      - **Container Name**: `clientservice`.
      - Your OpenShift credentials
   e. Click **Create And Start Container** to create the container.

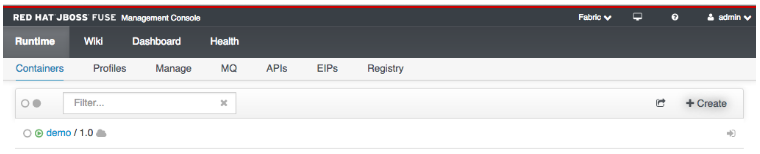
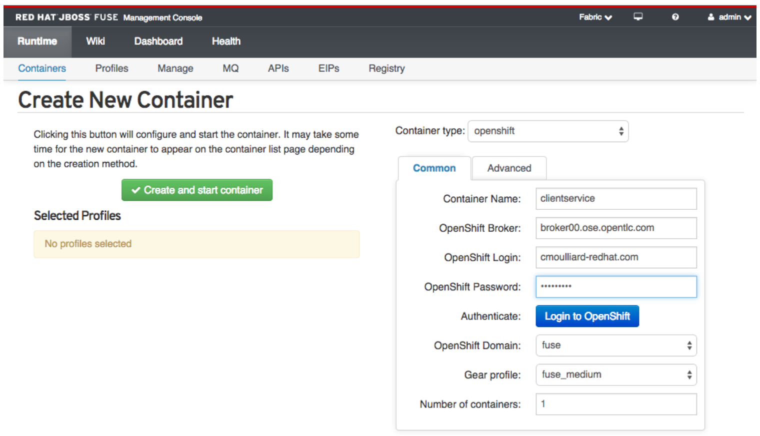


Figure 13. Create Client Service - 1



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

No profiles selected

Container type: openshift

Container Name: clientservice
OpenShift Broker: broker00.ose.opentlc.com
OpenShift Login: cmoulliard-redhat.com
OpenShift Password: •••••••••
Authenticate: Login to OpenShift
OpenShift Domain: fuse
Gear profile: fuse_medium
Number of containers: 1

Figure 14. Create Client Service - 2

4. Create a profile for the project:
   a. Click the **Wiki** tab.

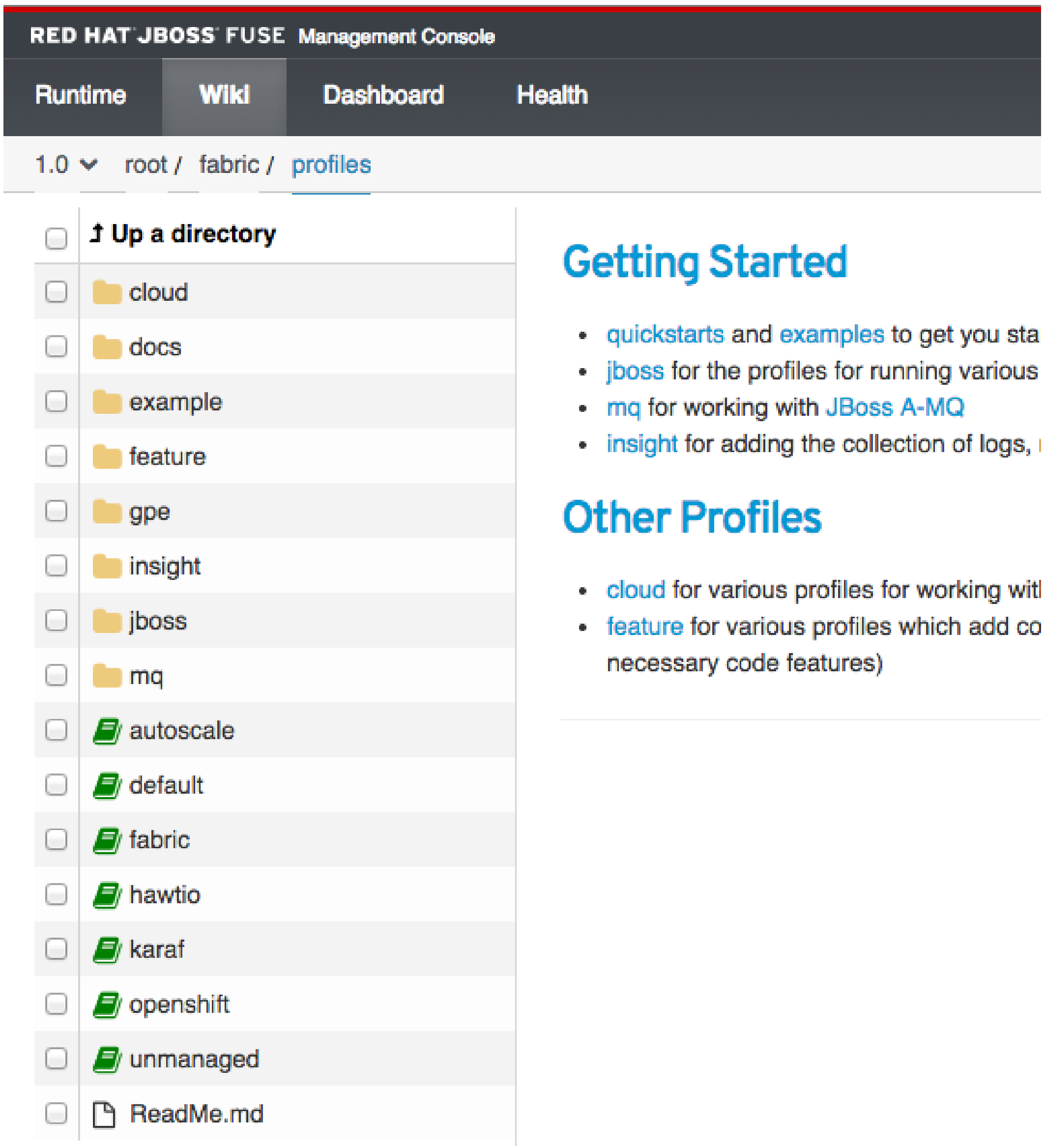

RED HAT JBOSS FUSE Management Console
Runtime
Wiki
Dashboard
Health
1.0
root / fabric / profiles
Up a directory
cloud
docs
example
feature
gpe
insight
jboss
mq
autoscale
default
fabric
hawtio
karaf
openshift
unmanaged
ReadMe.md
Getting Started
quickstarts and examples to get you sta
jboss for the profiles for running various
mq for working with JBoss A-MQ
insight for adding the collection of logs,
Other Profiles
cloud for various profiles for working wit
feature for various profiles which add co
necessary code features)

Figure 15. Wiki - Tree

b. Expand the tree to `gpe/exercise`.

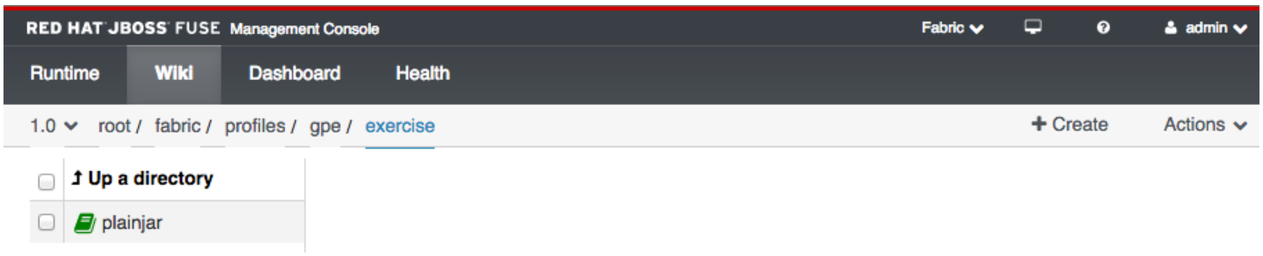
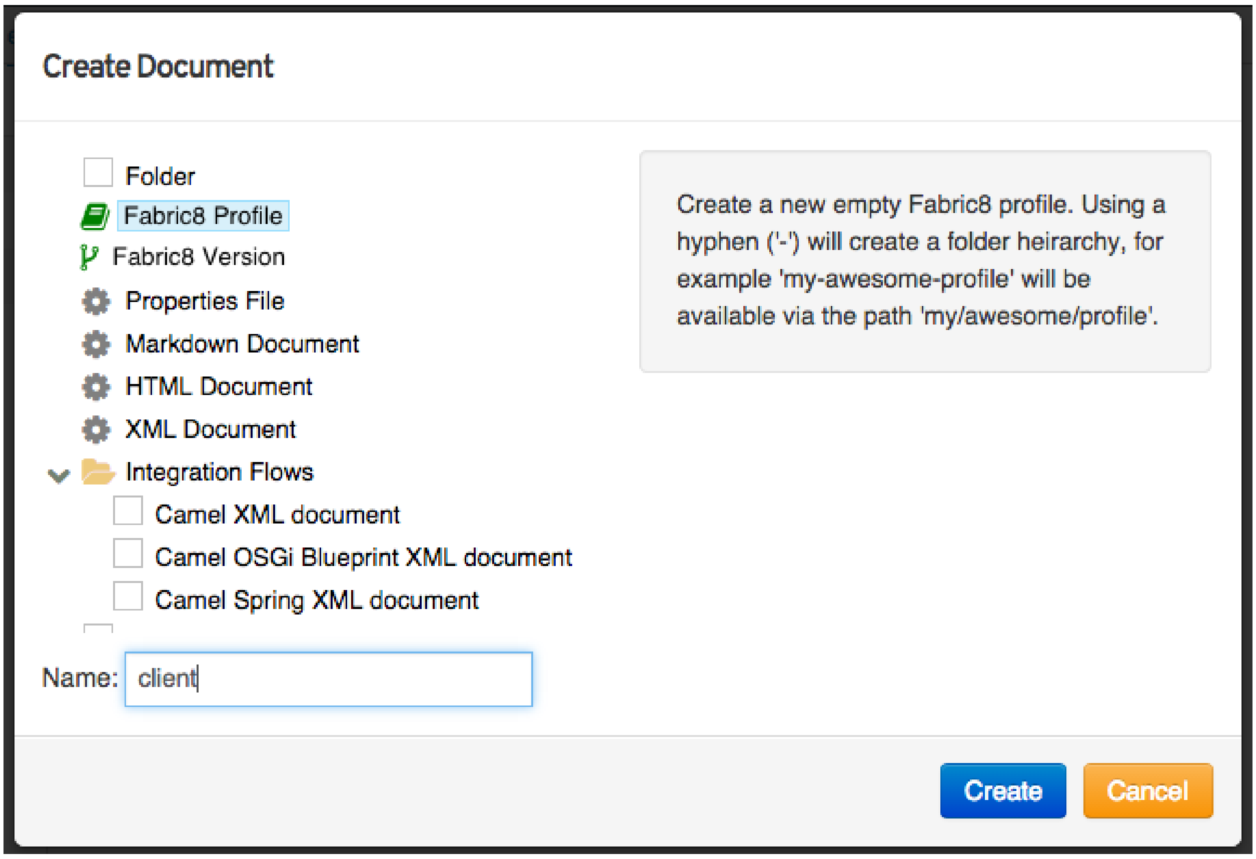

RED HAT JBOSS FUSE Management Console
Fabric
admin
Runtime
Wiki
Dashboard
Health
1.0
root / fabric / profiles / gpe / exercise
+ Create
Actions
Up a directory
plainjar

Figure 16. Project Tree

c. Click **Create** in the menu bar.

d. On the **Create Document** screen, select `Fabric8 profile`.

e. Enter `client` for the new profile **Name**, and then click **Create**.


Create Document

Folder
Fabric8 Profile
Fabric8 Version
Properties File
Markdown Document
HTML Document
XML Document
Integration Flows
Camel XML document
Camel OSGi Blueprint XML document
Camel Spring XML document

Create a new empty Fabric8 profile. Using a hyphen ('-') will create a folder heirarchy, for example 'my-awesome-profile' will be available via the path 'my/awesome/profile'.

Name: client

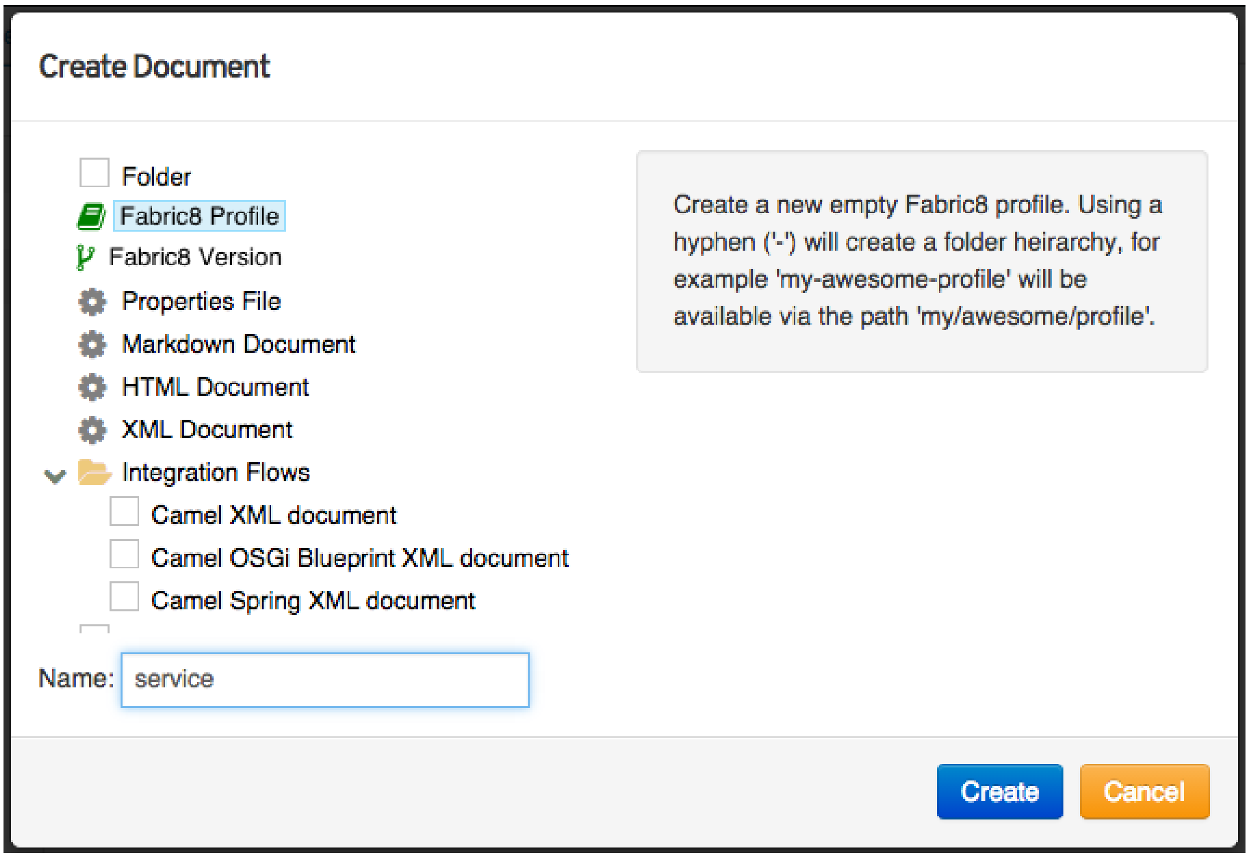
Figure 17. Create Client Profile

f. Click the **Bundles 0** tab and set the location of the `client` bundle file to the following:

```
mvn:com.redhat.gpe.training/osgi-client/1.0
```

g. Click the **+** symbol save these configuration changes.

h. Repeat the steps above to create the `service` profile.


Create Document

Folder
Fabric8 Profile
Fabric8 Version
Properties File
Markdown Document
HTML Document
XML Document
Integration Flows
Camel XML document
Camel OSGi Blueprint XML document
Camel Spring XML document

Create a new empty Fabric8 profile. Using a hyphen ('-') will create a folder heirarchy, for example 'my-awesome-profile' will be available via the path 'my/awesome/profile'.

Name: service

Figure 18. Service Profile

i. Click the **Bundles 0** tab and set the location of the `service` bundle file to the following:

```
mvn:com.redhat.gpe.training/osgi-service/1.0
```

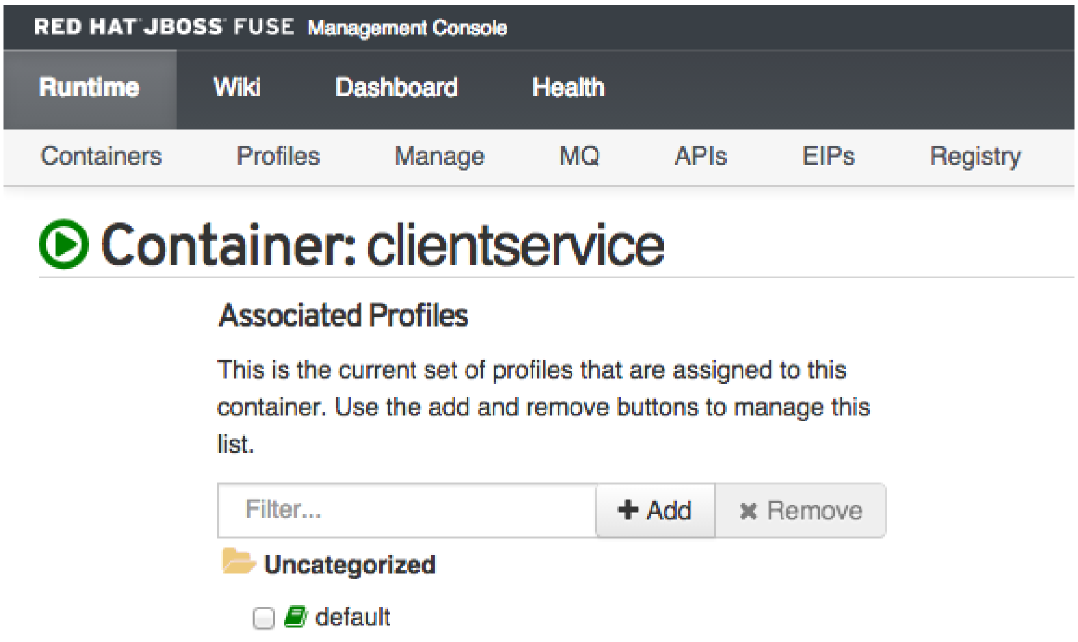
j. Click the **+** symbol save these configuration changes.

5. Deploy your two new profiles to the `clientservice` container:

   a. On the **Runtime** tab, click the `clientservice` icon.

   b. Inspect the contents of the `clientservice` container.

   c. Click **Add** and then select the `client` and `service` profiles to associate them with the container. After selecting each profile, click **Add** again.

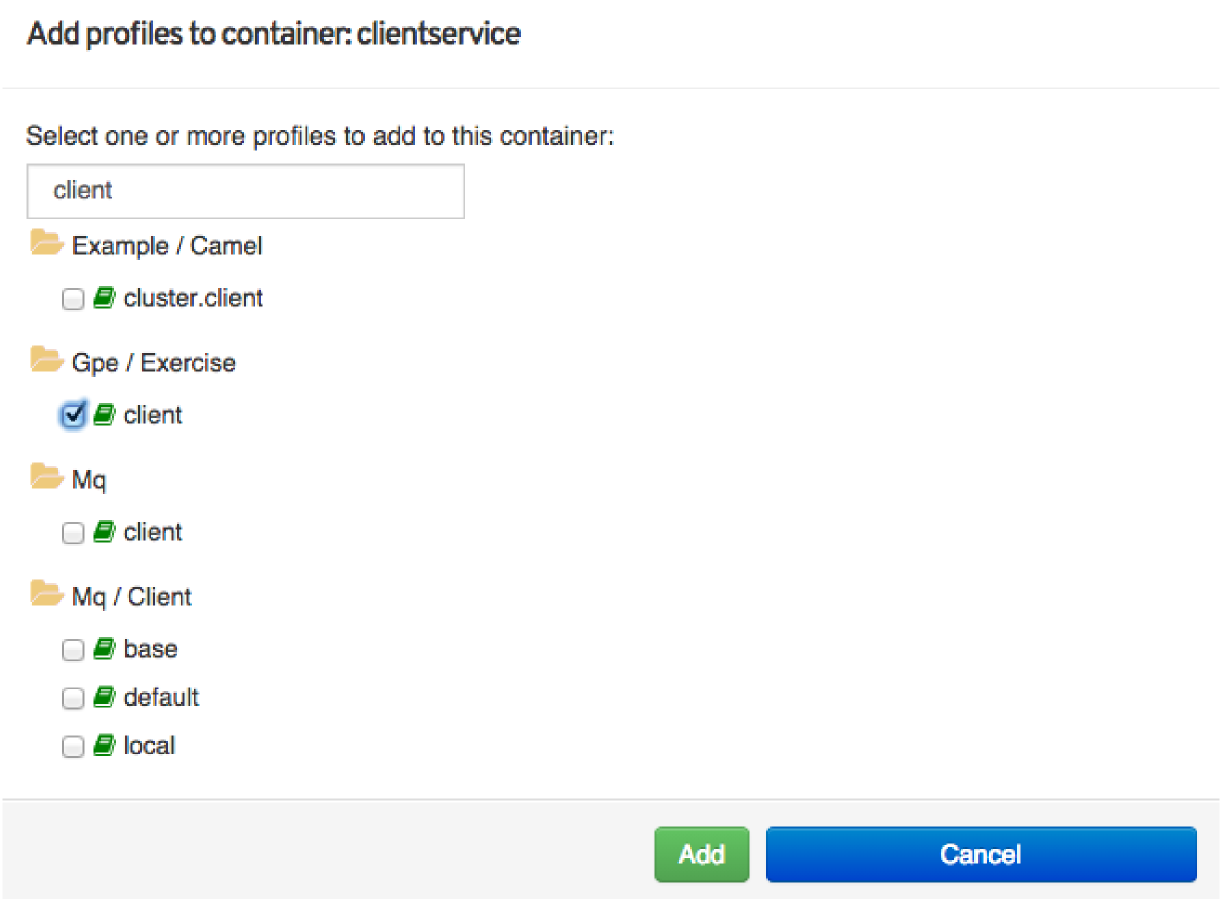


Container: clientservice

Associated Profiles

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

Figure 19. Clientservice Profile


Add profiles to container: clientservice

Select one or more profiles to add to this container:

client

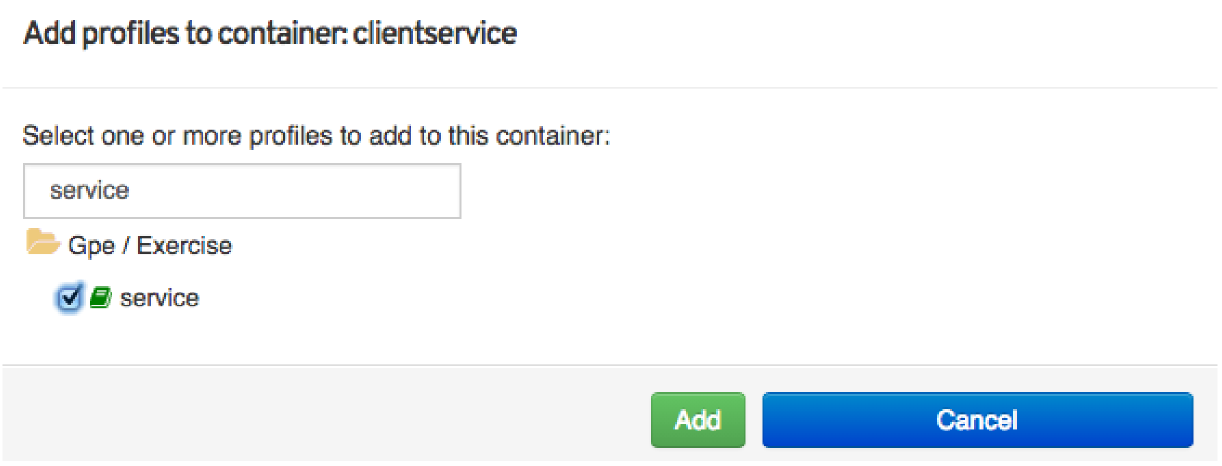
Example / Camel
- [ ] cluster.client

Gpe / Exercise
- [✓] client

Mq
- [ ] client

Mq / Client
- [ ] base
- [ ] default
- [ ] local

Figure 20. Add Client Profile


Add profiles to container: clientservice

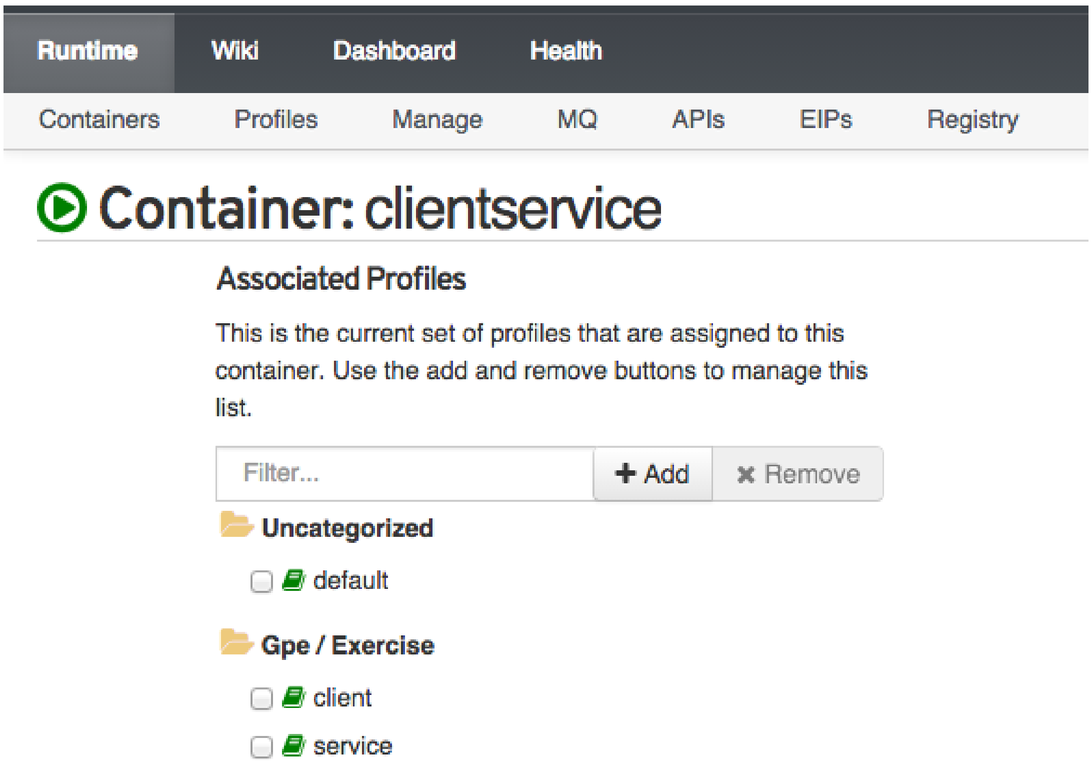
Select one or more profiles to add to this container:

service

Gpe / Exercise
- [✓] service

Figure 21. Add Service Profile

d. On the **Runtime** tab, look for a green icon next to the `Container:clientservice` title; this icon indicates successful deployment.



Container: clientservice

Associated Profiles

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

Uncategorized
- default

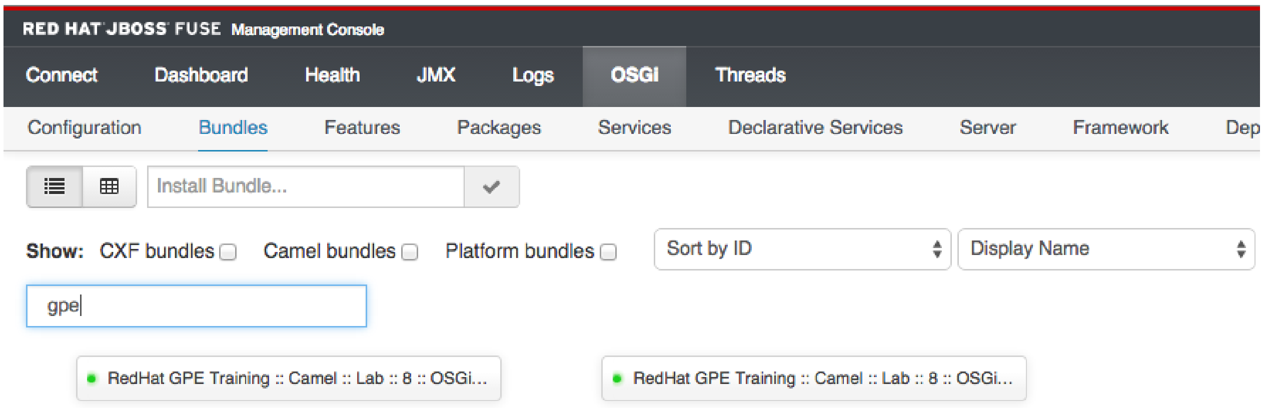
Gpe / Exercise
- client
- service

Figure 22. Verify Profile Deployment

6. Connect to the JBoss Fuse container:

   a. Click **Open** to open a new web page that displays the **JBoss Fuse Management Console** with the standard tabs.

   b. Click the **OSGi** tab and enter `gpe` in the search filter.



gpe

RedHat GPE Training :: Camel :: Lab :: 8 :: OSGi...

RedHat GPE Training :: Camel :: Lab :: 8 :: OSGi...

Figure 23. OSGi Tab Bundle Example

   c. Click the `client` bundle icon and review the following details of this bundle:

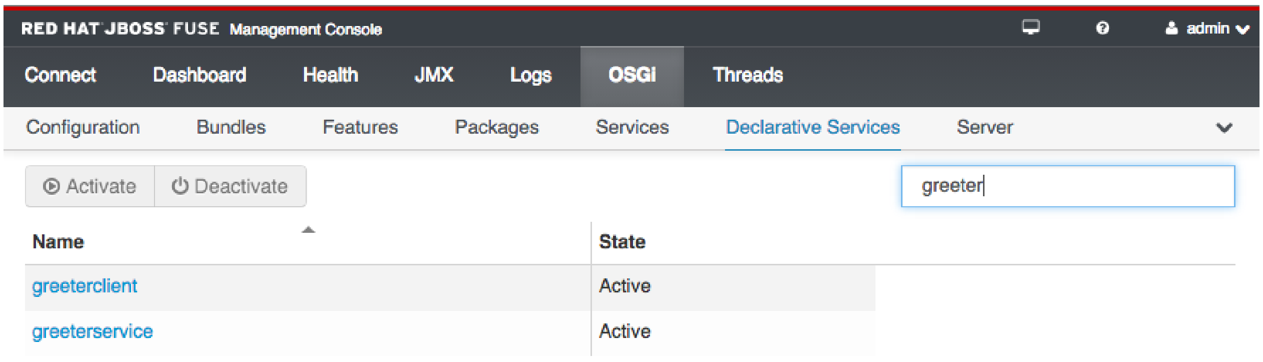
      - The `imported` and `exported` packages
      - Bundle **State**
      - Click **Services** to see the services used by the selected bundle.

        OSGi Tab - Details image::images/lab8/osgi_detail_clientservice.png[]

7. Verify the OSGi Service deployment:

   a. On the **OSGi** tab, select the **Declarative Services** view

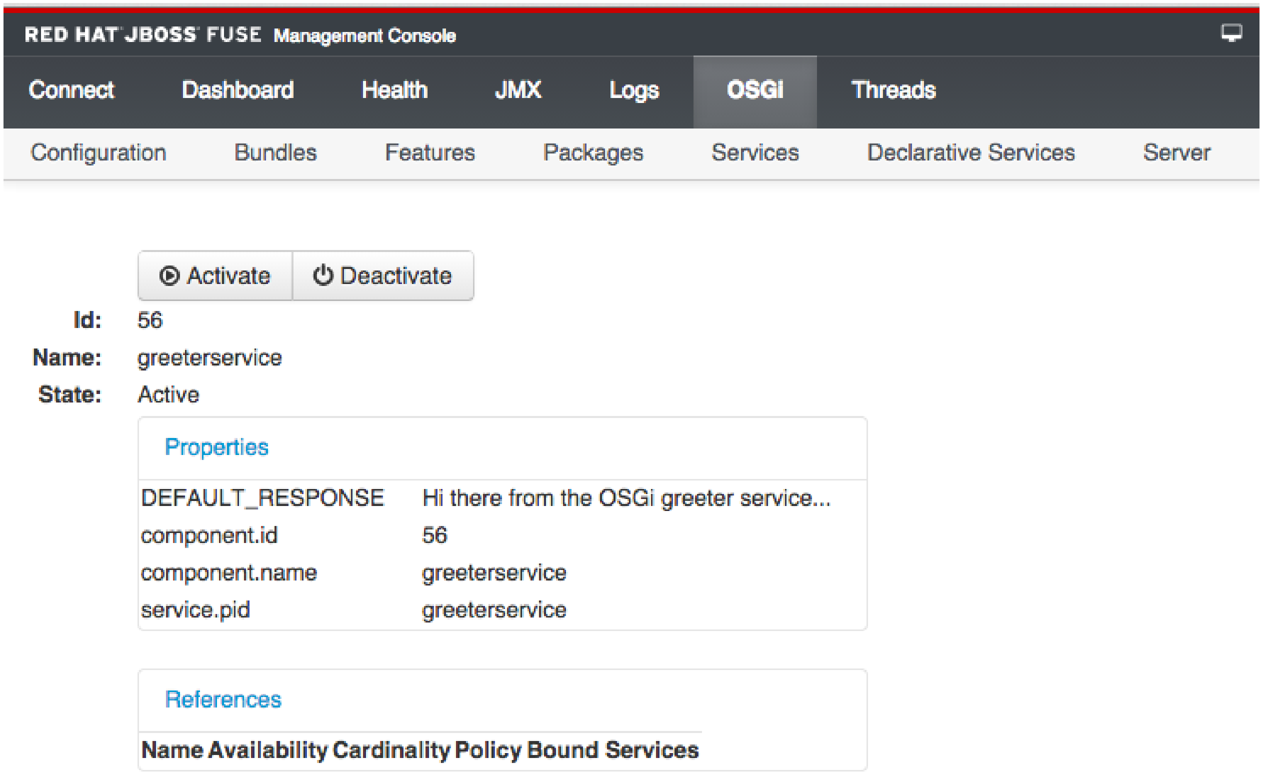
   b. Filter the list of services using `greeter` as the keyword.



Name	State
greeterclient	Active
greeterservice	Active

Figure 24. OSGi Tab - Key Filter

c. Click the `greeterservice` link to view the details of the service, such as the service **Id**, **Name**, **State**, and **Properties**.



Id: 56
Name: greeterservice
State: Active

Properties

DEFAULT_RESPONSE	Hi there from the OSGi greeter service...
component.id	56
component.name	greeterservice
service.pid	greeterservice

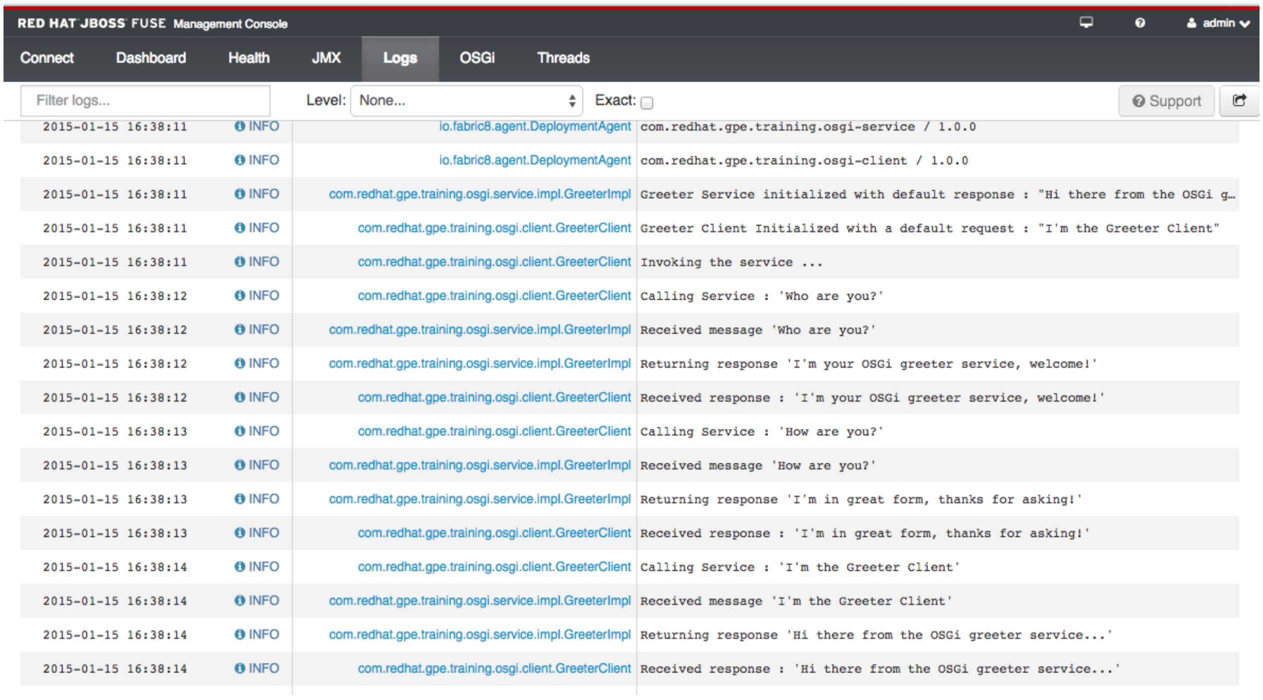
References

Name	Availability	Cardinality	Policy	Bound Services

Figure 25. OSGi Tab - Key Filter Details

d. Click the **Logs** tab and confirm that the `client` and `service` beans were instantiated and that they have exchanged messages.

Because you have not yet deployed the configuration files, the default request and response messages appear in the log files.



```
2015-01-15 16:38:11  INFO  io.fabric8.agent.DeploymentAgent                      com.redhat.gpe.training.osgi-service / 1.0.0
2015-01-15 16:38:11  INFO  io.fabric8.agent.DeploymentAgent                      com.redhat.gpe.training.osgi-client / 1.0.0
2015-01-15 16:38:11  INFO  com.redhat.gpe.training.osgi.service.impl.GreeterImpl Greeter Service initialized with default response : "Hi there from the OSGi g…
2015-01-15 16:38:11  INFO  com.redhat.gpe.training.osgi.client.GreeterClient     Greeter Client Initialized with a default request : "I'm the Greeter Client"
2015-01-15 16:38:11  INFO  com.redhat.gpe.training.osgi.client.GreeterClient     Invoking the service ...
2015-01-15 16:38:12  INFO  com.redhat.gpe.training.osgi.client.GreeterClient     Calling Service : 'Who are you?'
2015-01-15 16:38:12  INFO  com.redhat.gpe.training.osgi.service.impl.GreeterImpl Received message 'Who are you?'
2015-01-15 16:38:12  INFO  com.redhat.gpe.training.osgi.service.impl.GreeterImpl Returning response 'I'm your OSGi greeter service, welcome!'
2015-01-15 16:38:12  INFO  com.redhat.gpe.training.osgi.client.GreeterClient     Received response : 'I'm your OSGi greeter service, welcome!'
2015-01-15 16:38:13  INFO  com.redhat.gpe.training.osgi.client.GreeterClient     Calling Service : 'How are you?'
2015-01-15 16:38:13  INFO  com.redhat.gpe.training.osgi.service.impl.GreeterImpl Received message 'How are you?'
2015-01-15 16:38:13  INFO  com.redhat.gpe.training.osgi.service.impl.GreeterImpl Returning response 'I'm in great form, thanks for asking!'
2015-01-15 16:38:13  INFO  com.redhat.gpe.training.osgi.client.GreeterClient     Received response : 'I'm in great form, thanks for asking!'
2015-01-15 16:38:14  INFO  com.redhat.gpe.training.osgi.client.GreeterClient     Calling Service : 'I'm the Greeter Client'
2015-01-15 16:38:14  INFO  com.redhat.gpe.training.osgi.service.impl.GreeterImpl Received message 'I'm the Greeter Client'
2015-01-15 16:38:14  INFO  com.redhat.gpe.training.osgi.service.impl.GreeterImpl Returning response 'Hi there from the OSGi greeter service...'
2015-01-15 16:38:14  INFO  com.redhat.gpe.training.osgi.client.GreeterClient     Received response : 'Hi there from the OSGi greeter service...'
```

Figure 26. Logs Tab - Default Messages

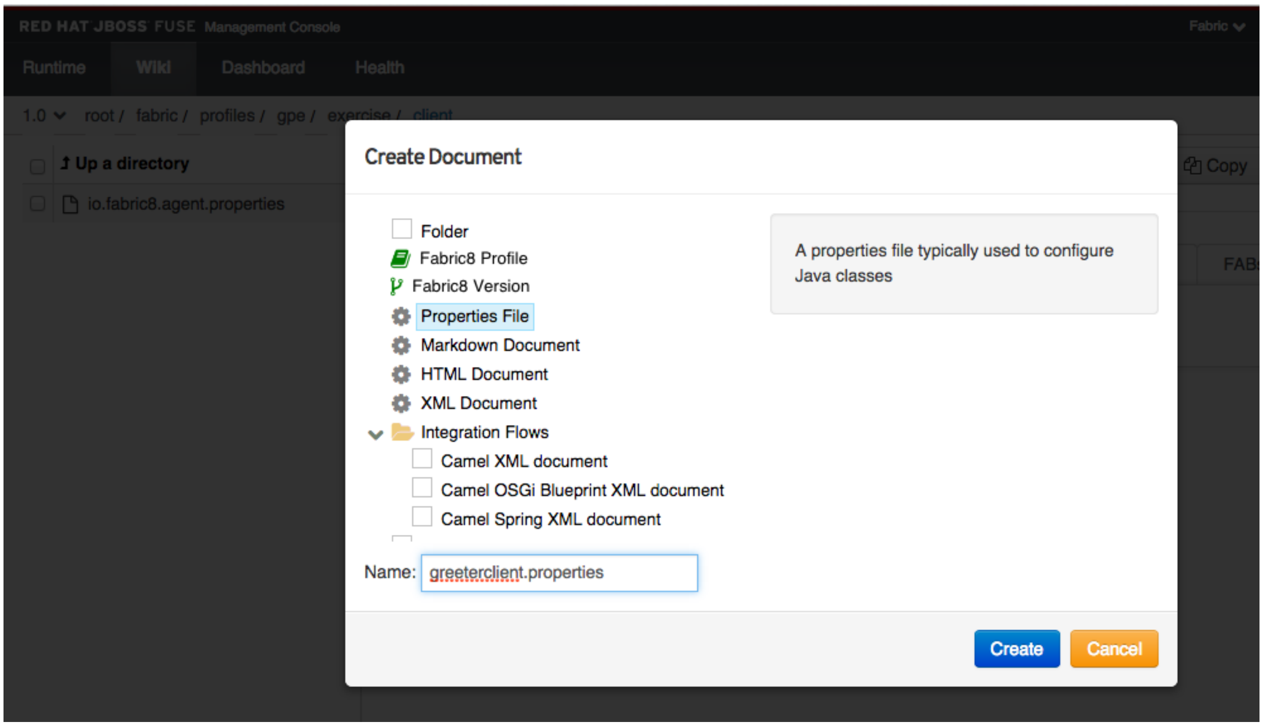
8. Add configuration files to the profiles:

   a. Return to the **JBoss Fuse Management Console** for the `demo` container.

   b. Click the **Wiki** tab.

   c. Navigate the profiles tree to `gpe/exercise` and select the `client` profile.

   d. Create a properties file called `greeterclient.properties`.



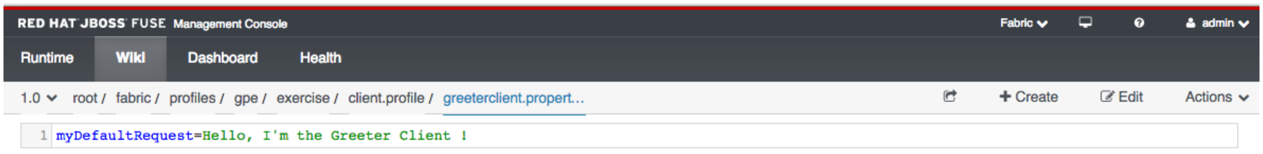
A properties file typically used to configure Java classes

Name: greeterclient.properties

Figure 27. Create Property

   e. Paste the content of the `osgi-client/etc/greeterclient.cfg` into the file you just created.

   f. Save the file.

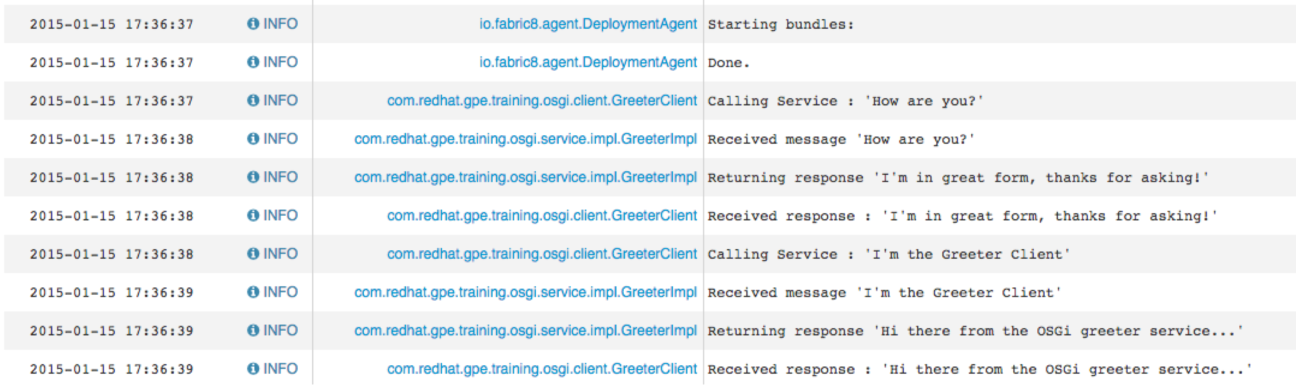

RED HAT JBOSS FUSE Management Console
Fabric
admin
Runtime
Wiki
Dashboard
Health
1.0 root / fabric / profiles / gpe / exercise / client.profile / greeterclient.propert...
Create
Edit
Actions

```
myDefaultRequest=Hello, I'm the Greeter Client !
```

Figure 28. Add Properties

g. Repeat the steps above to add the `service` profile, and select the `osgi-service/etc/greeterservice.cfg` file instead.

9. Click the **Logs** tab and confirm that the `client` and `service` beans were re-created and that they have exchanged messages using the information in the OSGi configuration files rather than the default messages. .



```
2015-01-15 17:36:37  INFO  io.fabric8.agent.DeploymentAgent                       Starting bundles:
2015-01-15 17:36:37  INFO  io.fabric8.agent.DeploymentAgent                       Done.
2015-01-15 17:36:37  INFO  com.redhat.gpe.training.osgi.client.GreeterClient      Calling Service : 'How are you?'
2015-01-15 17:36:38  INFO  com.redhat.gpe.training.osgi.service.impl.GreeterImpl  Received message 'How are you?'
2015-01-15 17:36:38  INFO  com.redhat.gpe.training.osgi.service.impl.GreeterImpl  Returning response 'I'm in great form, thanks for asking!'
2015-01-15 17:36:38  INFO  com.redhat.gpe.training.osgi.client.GreeterClient      Received response : 'I'm in great form, thanks for asking!'
2015-01-15 17:36:38  INFO  com.redhat.gpe.training.osgi.client.GreeterClient      Calling Service : 'I'm the Greeter Client'
2015-01-15 17:36:39  INFO  com.redhat.gpe.training.osgi.service.impl.GreeterImpl  Received message 'I'm the Greeter Client'
2015-01-15 17:36:39  INFO  com.redhat.gpe.training.osgi.service.impl.GreeterImpl  Returning response 'Hi there from the OSGi greeter service...'
2015-01-15 17:36:39  INFO  com.redhat.gpe.training.osgi.client.GreeterClient      Received response : 'Hi there from the OSGi greeter service...'
```

Figure 29. Logs Tab - Exchanged Messages

1.5. Access the OSGi Service From an Apache Camel Route

In this exercise, you learn how an OSGi Service is accessed by an Apache Camel route, from a separate bundle. You reuse the same service that you developed previously, but you use the https://access.redhat.com/documentation/en-US/Red_Hat_JBoss_Fuse/6.1/html/Apache_Camel_Component_Reference/IDU-Bean.html[Bean component} of Apache Camel to consume it. When Apache Camel performs a lookup to retrieve a bean, it must use available registries to complete the lookup. For OSGi deployment, it first uses OSGi/Blueprint to retrieve the bean according to the interface name.

The code implementation for the route is as follows:

```
from_DIRECT
.log_message("We will call the service using this message : ${body}")
.call_the_bean("greeterService")
.log_response_message("Response received from the Greeter Service : ${body}")
```

1. In **JBoss Developer Studio**, use **Project Explorer** to open the `osgi-camel-blueprint` project in the `lab_assets` directory.
2. Review the code of the various classes.
3. Open the `src/main/OSGI-INF/blueprint/camel-context.xml` file in the **Fuse Integration Editor**.
4. Design a Camel route that polls the file(s) from a local `exercises` directory and uses `file://exercise?moveFailed=failed&delete=false` as the endpoint definition:

   a. Convert the message body to a String.
   b. Log a message with the name of the file.
   c. Send the String to the `direct://training` producer.
5. Design a second route which starts from the `direct://training` endpoint:
   a. Log this message: `We will call the service using this message : ${body}`.
   b. Call the bean using the reference ID `greeterService`.
   c. The method of the service is `sayHello`, and the body must specify http://camel.apache.org/bean-binding.html as a parameter.
   d. Log this message using the response from the `greeterService`: `Response received from the Greeter Service : ${body}`
   e. Remember to add bean references using the `greeterService` ID and the `com.redhat.gpe.training.osgi.service.Greeter` interface.

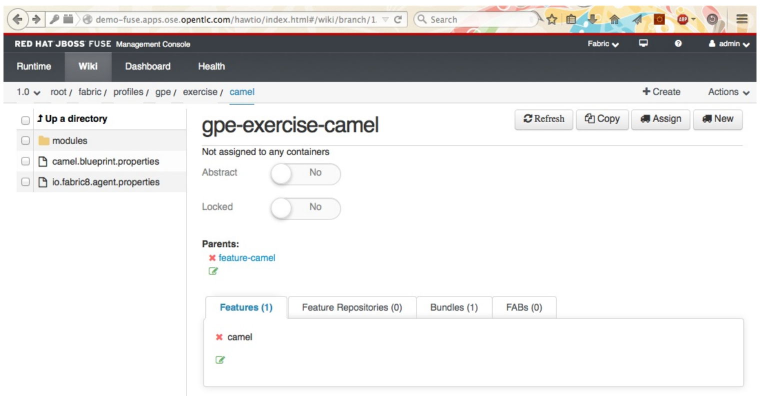
      The endpoints should be externalized using the Apache Camel **Properties** component and OSGi Service Platform Compendium `Property` annotation. The key-value pairs are defined in the file `src/main/fabric8/camel.blueprint.properties`

6. Deploy the project into a new JBoss Fuse container named `clientservice`.
   a. Verify that the `demo` Fuse Server is running.
   b. Connect to the OpenShift Enterprise server at this address `https://broker00.ose.opentlc.com` using the login credentials you defined in Module 1 of this course.
   c. Open the **JBoss Fuse Management Console** for the `demo` JBoss Fuse server.
   d. On the OpenShift Enterprise console, check that the container, which is also the Fabric server, is running.
      - If the Fabric server is running, continue the exercise.
      - If the Fabric server is not running, restart the server.
   e. In a console terminal (either of JBoss Developer Studio or your local machine), run the `mvn fabric8:deploy` Maven command.

      This is the terminal where you will use the `osgi-camel-blueprint` Maven plug-in to create a profile and deploy it into the Git repository of the `demo` Openshift Fabric8 server. Follow the instructions defined here to use Maven to set up UserID/password pair for accessing the JBoss Fuse Fabric8 container running on OpenShift Enterprise.

      You must modify the `<jolokiaURL>` defined in the `<properties>` tag of the Maven `settings.xml` file so that it contains the address of the Fabric8 server.

f. Check the **Wiki** tab to see whether the deployment was successful. If deployment succeeded, a new `gpe/exercise/camel` profile appears on this tab.



demo-fuse.apps.ose.opentlc.com/hawtio/index.html#/wiki/branch/1.

RED HAT JBOSS FUSE Management Console

Runtime | Wiki | Dashboard | Health

1.0 root / fabric / profiles / gpe / exercise / camel

+ Create | Actions

Up a directory
modules
camel.blueprint.properties
io.fabric8.agent.properties

gpe-exercise-camel

Refresh | Copy | Assign | New

Not assigned to any containers

Abstract: No

Locked: No

Parents:
feature-camel

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

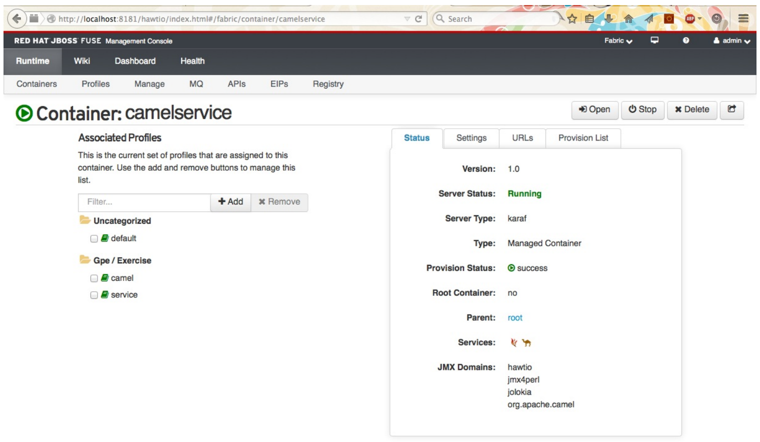
camel

Figure 30. Confirm New Profile

7. Provision a new `camelservice` container with the newly created `gpe/exercise/camel` profile and the existing `service`:

   a. From the **JBoss Fuse Management Console**, create a new `camelservice` container.

   b. Assign both the `gpe/exercise/service` and `gpe/exercise/camel` profiles to the new container.



http://localhost:8181/hawtio/index.html#/fabric/container/camelservice

RED HAT JBOSS FUSE Management Console

Runtime | Wiki | Dashboard | Health

Containers | Profiles | Manage | MQ | APIs | EIPs | Registry

Container: camelservice

Open | Stop | Delete

Associated Profiles

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

Filter... | + Add | Remove

Uncategorized
- default

Gpe / Exercise
- camel
- service

Status | Settings | URLs | Provision List

Version:	1.0
Server Status:	Running
Server Type:	karaf
Type:	Managed Container
Provision Status:	success
Root Container:	no
Parent:	root
Services:	
JMX Domains:	hawtio jmx4perl jolokia org.apache.camel

Figure 31. Assign Camelservice

   c. Open the `camelservice` Fuse container and click the `camel` menu tab.

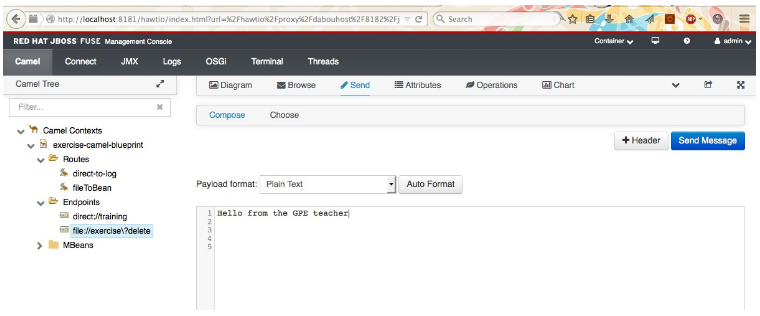
   d. Expand the `direct-to-log` and `fileToBean` routes to verify that they were created and started.

   e. Expand the endpoints and verify that they were also created and started.

   f. Select the `direct://training` endpoint.

   g. Click **Send** → **Compose**.

   h. Copy and paste the contents of the `src/data/greeter.txt` file into the message editor.



```
Hello from the GPE teacher
```

Figure 32. Send Message

8. Verify results:

    a. Return to the `direct-to-log` Apache Camel route and observe that an Exchange object was generated.

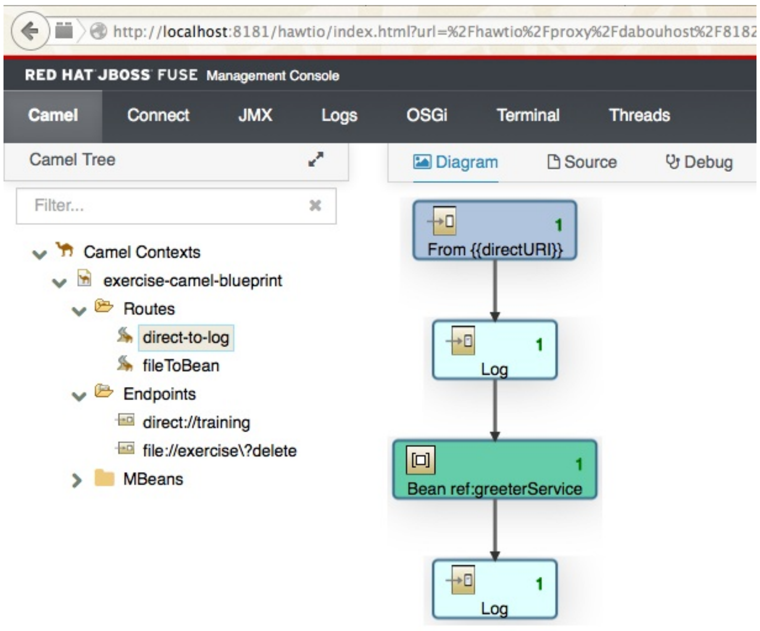
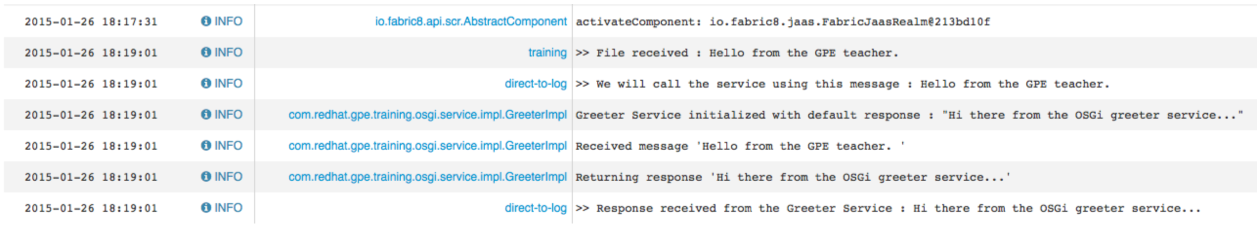


Figure 33. Exchange Fired

    b. Click the `Logs` tab.

    c. Observe the different messages that were logged and that the `Greeter` service was called.



```
2015-01-26 18:17:31  INFO             io.fabric8.api.scr.AbstractComponent activateComponent: io.fabric8.jaas.FabricJaasRealm@213bd10f
2015-01-26 18:19:01  INFO                                         training >> File received : Hello from the GPE teacher.
2015-01-26 18:19:01  INFO                                    direct-to-log >> We will call the service using this message : Hello from the GPE teacher.
2015-01-26 18:19:01  INFO com.redhat.gpe.training.osgi.service.impl.GreeterImpl Greeter Service initialized with default response : "Hi there from the OSGi greeter service..."
2015-01-26 18:19:01  INFO com.redhat.gpe.training.osgi.service.impl.GreeterImpl Received message 'Hello from the GPE teacher. '
2015-01-26 18:19:01  INFO com.redhat.gpe.training.osgi.service.impl.GreeterImpl Returning response 'Hi there from the OSGi greeter service...'
2015-01-26 18:19:01  INFO                                    direct-to-log >> Response received from the Greeter Service : Hi there from the OSGi greeter service...
```

Figure 34. Camel Logs Greeter Service

1.6. Design Integration Tests Using OSGi Testing Frameworks

In this exercise, you learn two different OSGi testing approaches to design a OSGi Unit test. One approach is based on the OPS4J Pax-Exam framework, which locally starts a JBoss Fuse container. The other approach is based on the PoJoSR framework, which is currently used by Apache Camel.

Both frameworks extend the Junit framework. Therefore, the methods you design within the test classes use the static methods of JUnit as well as the @Test annotation. The difference between the two frameworks is that PoJoSR uses a classloader to load the bundles and the test, but PAX-Exam initializes an OSGi container and then required bundles are deployed in that container using their own classloader.

1.6.1. Use the Pax-Exam Framework

1. In **JBoss Developer Studio**, use **Project Explorer** to open the `osgi-camel-pax-testing` project.
2. Expand the `src/main/java` tree and note the following:
   - The `OSGiIntegrationTestSupport` class contains the code required to configure the OSGi container. You will reuse it in your own tests.
   - The `camel` and `spring` features to be deployed into the container are defined by default.
   - For container setup and parameter definitions, such as the Karaf runtime version and installation location, you will use the `getDefaultCamelKarafOptions` method.
   - This class will be extended by the `BindyDataFormatCsvTest` test case. This test case contains the `@Configuration` and `@BeforeClassTest` annotations that are used to instantiate and initialize the container. When this is complete, the different `@Test` annotations are scanned to begin the tests.
3. Open the `BindyDataFormatCsvTest` class and do the following: ..Expand the `testMarshal` method and create an Apache Camel test in this method.
   a. Read the content of the class and observe the route definition and the data used, such as `Employees`.
   b. Within the body of the method, create a `MockEndpoint` with `mock:bindy-marshal` as the endpoint address.
   c. Use the `FIXED_DATA` string to set what you expect to receive as body.
   d. Use the template to send a list of `Employees` to the `direct:marshal` endpoint using the `sendBody` method.
   e. At the mock endpoint, make a call to the `assertSatisfied` method.
4. Verify the test:
   a. Open a console terminal window and move to the `osgi-camel-pax-testing` directory.
   b. Run the `mvn test` Maven command to execute the test. If the test executes successfully, you will see this message:

```
2015-01-28 11:42:37,863 [main           ] INFO  ReactorManager                 - suite
finished
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 12.044 sec - in
com.redhat.gpe.training.osgi.test.BindyDataFormatCsvTest

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 16.227 s
[INFO] Finished at: 2015-01-28T11:42:38+01:00
[INFO] Final Memory: 36M/451M
[INFO] ------------------------------------------------------------------------
```

1.6.2. Use the PoJoSR Framework

In this exercise, you explore the PoJoSR framework used by `CamelBlueprintTestSupport` class and you design the Apache Camel route that the unit test uses.

For more information about the binding used in this framework, see: https://access.redhat.com/documentation/en-US/Red_Hat_JBoss_Fuse/6.1/html/Apache_Camel_Component_Reference/IDU-Bean.html.

1. In **JBoss Developer Studio**, use **Project Explorer** to open the `osgi-camel-pojosr-testing` project.
2. Expand the `src/main/java` and `src/test/java` trees and explore the Java classes.
3. Use the **Fuse Integration Editor** to open the `src/main/resources/OSGI-INF/blueprint-camel-test.xml` file, and do the following:
    a. Create an Apache Camel route starting with the `from("direct:start")` endpoint.
    b. Introduce a bean processor that accepts the `beanType` parameter `com.redhat.gpe.training.osgi.impl.HelloWorldSvcImpl` and that calls the `sayHello()` method. The parameter for the `sayHello` method is the Exchange body.
    c. Complete the route with the `mock:result` endpoint.
    d. Review your route.
4. Open a console terminal and move to the `osgi-camel-pojosr-testing` directory.
5. Run the `mvn test` Maven command to execute the test. If the test executes successfully, you will see this message:

```
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.595 sec - in
com.redhat.gpe.training.osgi.test.BlueprintOSGiServiceTest

Results :


Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

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