JBoss Fuse Integration Days - Hands on Lab Session

Table of Contents

Session 1 : Fuse Development - 90min

Prerequisites

- If not yet done, download and install the following software

- Apache Maven / Apache Maven 3.0.5,
- Java JDK 7 / JDK 7,
- SOAPui / SOAPUi 5.0,
- JBoss Developer Studio (JBDS) / JBDS
- JBoss Fuse & JBoss AMQ / JBoss Fuse 6.1 & JBoss A-MQ 6.1

Remark : - The software required can be installed from one of the USB key available

Post-Installation

- After installing JBDS on your machine, the SOA/Integration tools must be deployed as it contains the Camel Editor like also BRMS/BPMS tools when you would like to work with the Rules engine or Business process engine
- Follow these steps defined hereafter

```
Start JBDS and select from the menu
Help > Install New Software...
Add...
Insert the following Location: https://devstudio.jboss.com/updates/7.0/integration-stack/
```

- Optional : SOAPUI can be also installed within JBDS. Follow these steps

```
Start JBDS and select from the menu
1. Select Help > Install New Software...
2. In the Work with field, type http://www.soapui.org/eclipse/update and click Add...
```

- To avoid that every participant download same maven artefacts from internet, a maven proxy server has been configured. To use it, you must change your maven settings.xml
- Edit the ~/.m2/settings.xml and add the following lines to use the maven mirror server

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">
<mirrors>
  <mirror>
    <id>archiva.default</id>
    <url>http://192.168.137.1:8080/repository/all/</url>
    <mirrorOf>external:*</mirrorOf>
  </mirror>
</mirrors>
</settings>
```

Offline installation

An installation script can also be used to install JBDS including also the *integration tools*, JBoss Fuse on your Unix machine from a local FTP server. This procedure will also update your maven settings.xml file (located under ~/.m2/settings.xml) to use a Proxy HTTP server running on the instructor machine. This installation procedure could be achieved for you without internet connection ("offline mode").

Here are the steps to follow :

- Open a unix terminal
- Create under your temp directory a lab/scripts directory

- Run the following command

The address of the FTP server will be provided by the instructor (e.g : 192.168.137.1)

```
cd ~/Temp/lab/scripts
wget ftp://192.168.137.1/pub/scripts/all_commands.sh
chmod +x *.sh
./all_commands.sh ~/Temp/lab/ 192.168.137.1
```

Installation of the code for the demos

- Clone the `hands on lab` git project and move to the directory `hands-on-lab` containing the code of the demos. Demos is available in the folder *demo-camel-ws*

```
git clone https://github.com/cmoulliard/fuse-days-2014.git
```

or use the code available on the

- USB key
  - Copy file demos.zip on your machine
  - Unzip the file

Demos

Demo 1 - Using Camel Archetype with JBDS

- Start JBDS 7.1. GA
- Create in your home directory a workspace wih the name `workspace-lab`
- Select `New Fuse Project`
- Skip first screen `Select project location`
- Next, select the archetype `camel-archetype-activemq`
- Define the following values for :

```
groupid : my
archetypeid : cool-demo
version : 1.0
```

- Click on `next` button
- When the project appears, switch to perspective `Fuse integration`
- Open the camel-context.xml file located under src/main/resources/META-INF/sporing directory to navigate between the different camel routes (see menu - `routes`) and review them
- Start the camelcontext using `Run as Local Camel Context`
- Open JMX Browser and navigate to discover the MBeans of ActiveMQ browser and Camel
- Rick click on button "Refresh" if the MBeans didn’t appear in the list
- Review the MBeans (routes, endpoints, …)
- Enable tracing

- Drag and drop a message to an endpoint
- Review statistics under Message log
- When everything is done, shutdown the camel process

Demo 2 - Import project - Demo Camel CXF with JBDS

- Start JBDS 7.1. GA
- Select the `workspace-lab`
- Import the `demo-camel-ws` project (as a Existing Maven project)
- Compile it using `mvn install` goal
- Change project properties (select in the menu bar → Project → Properties)
- Select Java Build Path, Click on Source and add folder `target/generated/src/main/java` to include code created by cxf-codegen-plugin
- Open Fuse integration perspective, review the camel route
- Start the camelContext using `Run as Local Camel Context`
- Open the view `soapUI Navigator` from the menu bar (Show view → Other → soapUI → soapUI Navigator) OR
- Start SOAPUI
- Select New project and add the URL point to the WSDL file of the Web Service exposed by the Camel route `http://localhost:9090/training/WebService?wsdl`
- When the project has been imported, select the service `getCustomersByName` and open `Request 1`. Replace the question mark symbol by `RedHat` to search about it.
- You will get a response

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getCustomerByNameResponse xmlns="http://example.fuse.redhat.com/">
      <return>
        <name>RedHat</name>
        <address>RedHat Office</address>
        <numOrders>78</numOrders>
        <revenue>5580.0</revenue>
        <test>100.0</test>
        <type>BUSINESS</type>
      </return>
    </getCustomerByNameResponse>
  </soap:Body>
</soap:Envelope>
```

- Test the service `getAllCustomers` and `saveCustomer`
- A request1 example to save a new customer is available under `src/data/request-save` directory

you can also use **curl** to send request and get the result

curl -X POST -H "SOAPAction: http://example.fuse.redhat.com/getCustomerByName" -T hands-on-lab/demo-camel-ws-after/src/main/resources/data/request.xml http://localhost:9090/training/WebService | xmllint --format -

curl -X POST -H "SOAPAction: http://example.fuse.redhat.com/saveCustomerRequest" -T hands-on-lab/demo-camel-ws-after/src/main/resources/data/request-save.xml http://localhost:9090/training/WebService | xmllint --format -

curl -X POST -H "SOAPAction: http://example.fuse.redhat.com/getAllCustomers" -T hands-on-lab/demo-camel-ws-after/src/main/resources/data/request-all.xml http://localhost:9090/training/WebService | xmllint --format -

Demo 3 - Unit Test with JBDS

- Use project created for my cool demo (first demo)
- To create the unit test, select `New` and `Camel Test Case` after selecting the project `cool-demo`
- Accept the by default value and verify that the camel-context.xml file of your project is well selected
- Package name should be `my.cool.demo`
- Browse to select the camel xml file located under your src/main/resources/META-INF/spring` directory
- Keep by default endpoints selected and click on `finish` button
- Run the unit test using `Run As` and `Junit Test` options available under the java class of the unit test case created
- Check result

Session 2 : Packaging and deployment in JBoss Fuse - 90min

Presentation of JBoss Fuse & AMQ 6.1

Demos

Demo 1 - Tailor your JBoss Fuse server

- If this is not yet done, edit this file `etc/org.ops4j.pax.url.mvn.cfg` part of the JBoss Fuse or AMQ distribution and verify that you use the proxy server of the lab to download maven artifacts

```
org.ops4j.pax.url.mvn.repositories= \
  http://192.168.137.1:8080/repository/all
```

- To tailor your Fuse server, we will edit the features file before to start the first time the server and

just deploy what we want

- Edit the file `etc/org.apache.karaf.features.cfg` located under /etc directory of the JBoss Fuse/AMQ distribution
- Comment the line starting with the key `featuresBoot` and add the following features

```
featuresBoot=jasypt-encryption,config,management,fabric,fabric-bundle,fabric-maven-proxy,patch,war,hawtio
```

- Start `fuse` with the command `/fuse` or fuse.bat` in Unix/Dos terminal
- Check on the console that the number of bundles deployed corresponds to `128`

```
...
[ 126] [Active     ] [            ] [   60] JLine (2.11.0)
[ 127] [Active     ] [Created     ] [   60] hawtio :: Karaf terminal plugin (1.2.0.redhat-379)
[ 128] [Active     ] [Created     ] [   60] hawtio :: hawtio-maven-indexer (1.2.0.redhat-379)
```

- Deploy new `features/modules` camel and camel-cxf using these commands

```
features:install camel
features:install camel-cxf
```

- Verify the bundles deployed but also new commands, OSGI services

```
[ 199] [Active     ] [            ] [       ] [   50] Apache CXF Compatibility Bundle Jar (2.7.0.redhat-610379)
[ 200] [Active     ] [Created     ] [       ] [   50] camel-cxf-transport (2.12.0.redhat-610379)
[ 201] [Active     ] [Created     ] [       ] [   50] camel-cxf (2.12.0.redhat-610379)
```

A new Karaf command has been added `cxf`

cxf:list-busses cxf:list-endpoints cxf:start-endpoint cxf:stop-endpoint

Demo 2 - Review JBoss Fuse commands

- Type on the keyboard `tab` key → Display all 311 possibilities? (y or n)
- Answer `yes`
- Type `log:` tab to see the commands belonging to `log` family
- Repeat with `OSGI`, `features`, `admin`, `log` and dicuss with instructor purpose of the commands

Demo 3 - Transform project into a bundle

- Return to your JBDS and open the demo-camel-ws project
- Change the packaging of the pom.xml file of the camel cxf demo project to bundle

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

- Add Apache Felix Bundle plugin to generate META-DATA required by OSGI runtime

```
<!-- to generate the MANIFEST-FILE required by the bundle ->
<plugin>
   <groupId>org.apache.felix</groupId>
   <artifactId>maven-bundle-plugin</artifactId>
   <version>${maven.bundle.plugin.version}</version>
   <extensions>true</extensions> <!-- Used in combination with bundle packaging to extend generate process ->
   <configuration>
      <instructions>
         <Bundle-SymbolicName>${project.artifactId}</Bundle-SymbolicName>
         <Import-Package>
            !com.redhat.fuse.example.camel,
```

```
            javax.jws,
            javax.jws.soap,
            javax.xml.bind,
            javax.xml.bind.annotation,
            javax.xml.datatype,
            javax.xml.namespace,
            javax.xml.ws,
            org.apache.camel;version="[2.10,3)",
            org.apache.camel.builder;version="[2.10,3)",
            org.apache.camel.model;version="[2.10,3)",
            org.apache.cxf.interceptor;version="[2.6,3)",
            org.apache.cxf.message;version="[2.6,3)",
            org.slf4j;version="[1.6,2)"
        </Import-Package>
    </instructions>
  </configuration>
</plugin>
```

- Build the project using the command `mvn clean install` (Right click on the pom.xml file and select `Run as --> mvn install`)
- Deploy the project on JBoss Fuse using the OSGI install command

```
osgi:install -s mvn:com.redhat.fuse/demo-camel-ws/1.0
```

```
Remark : Fuse supports different protocols (mvn, file, ftp, http, ...) to deploy your project (jar, bundle, war)
```

- Verify that the bundle is deployed and status equals to `start`

```
osgi:list | grep -i demo
```

- Verify also that we have a CXF WS endpoint running (web browser → `http://localhost:9090/training/WebService?wsdl`) but also using cxf command

```
cxf:list-endpoints
Name                    State     Address                                  BusID
[CustomerServicePort    ] [Started ] [http://localhost:9090/training/WebService          ] [demo-camel-ws-cxf1991170407        ]
```

- Using `soapui`, send a SOAPUI XML request by selecting the `getAllCustomers` Request generated and click on the send button
- Look to the log file `log:display` or `ld` which is a shortcut to verify that SOAP request/response have been traced
- Shutdown JBoss Fuse Server (`ctrl-D or osgi:shutdown)

Demo 4 - Deploy to a Fuse Container using JBDS

- Before to run this exercise, remove the directory `data` of the JBoss Fuse / AMQ distribution. It contains the log files but also the bundles previously deployed.
- Switch to JBDS in order to configure it to use the JBoss Fuse 6.1 server and deploy projects into the `deploy` folder (which is a folder scanned every 1s by Karaf to hot deploy jar, war, bundles, … on the platform
- Select `DeployTo` and `Deploy Folder configuration` available after right clicking on the demo camel ws project
- When the screen `Deploy Folders` appear, add the name `jbossfuse6.1`, point to your `deploy` folder

```
~/../jboss-fuse-6.1.0.redhat-379/deploy
```

- Description should be the same as `name`
- Add this config and click on `ok` button
- Select from the pom.xml file the option deploy to → `jbossfuse6.1`
- mvn build will take place and deploy the jar file created under the `deploy` directory of your server
- Next, select the servers view and add JBoss Fuse 6.1
- When this is done, click on the start button to launch the server
- Select the `shell view`, open it and use the karaf shell

Remarks : - When the ssh connection will be established, verify that the port number of the karaf instance to be connected is `8101`, the hostname is `0.0.0.0` and login/password `admin/admin` - The first time you will connect, you will get a pop up windows which will ask you if you accept the digital signature created to connect to karaf server. Reply by yes

- List the bundles deployed on the console and consult the log file
- Check if camel and camel-cf modules are well deployed. If this is not the case, deploy them
- Deploy the web service using the command `Deploy to --> jbossfuse6.1` (Right click on the project of the camel demo ws)
- Verify if the web service is working.
- Remove the jar/bundle deployed

```
rm -rf ../deploy/demo-camel-ws-1.0.jar
```

Demo 5 - Package the project using a features file

- Before to run this exercise, remove the directory `data` of the JBoss Fuse / AMQ distribution. It contains the log files but also the bundles previously deployed.
- Switch to JBDS
- To achieve this goal, we will create a feature xml file containing our bundle
- Create the file features.xml under src/main/resources/respository directory

```
<?xml version="1.0" encoding="UTF-8"?>
<features name="my-cool-demo-${project.version}"
        xmlns="http://karaf.apache.org/xmlns/features/v1.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://karaf.apache.org/xmlns/features/v1.0.0 http://karaf.apache.org/xmlns/features/v1.0.0">
```

```
  <feature name="cool-camel-cxf-demo" version="${project.version}">
    <bundle>mvn:com.redhat.fuse/demo-camel-ws/${project.version}</bundle>
  </feature>
```

```
  </features>
```

- Next we will do some maven modifications to add this file during the build process into the maven repository

```
  <resources>
```

```
    <resource>
      <directory>${project.basedir}/src/main/resources</directory>
      <filtering>true</filtering>
    </resource>
  </resources>
```

```
<plugins>
  <plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>build-helper-maven-plugin</artifactId>
    <version>1.8</version>
    <executions>
      <execution>
        <id>attach-artifacts</id>
        <phase>package</phase>
        <goals>
          <goal>attach-artifact</goal>
        </goals>
        <configuration>
          <artifacts>
            <artifact>
              <file>${project.build.outputDirectory}/repository/features.xml</file>
              <type>xml</type>
              <classifier>features</classifier>
            </artifact>
          </artifacts>
        </configuration>
      </execution>
    </executions>
  </plugin>
```

- Build the project `mvn clean install`
- Shutdown JBoss Fuse `ctrl-d` and restart it by doing a cleanup `./fuse clean`
- install the features xml file

```
features:addurl mvn:com.redhat.fuse/demo-camel-ws/1.0/xml/features
```

- Check that the file is well installed

```
listurl
```

- Grep the `demo` feature

```
features:list | grep -i my-cool
[uninstalled] [1.0                ] cool-camel-cxf-demo             my-cool-demo-1.0
```

- Install the feature

```
features:install cool-camel-cxf-demo
```

- Verify that the webservice is well working
- Analyze the camel statistics (command line)

```
camel:endpoint-list
camel:context-list
camel:route-list
```

```
camel:route-profile route1
```

- Execute some curl or soapui request, chck the statistics

```
camel:route-profile route1
```

```
Profile
    Camel Context: camel-1
```

Id	Count	Last (ms)	Delta (ms)	Mean (ms)	Min (ms)	Max (ms)	Total (ms)	Self (ms)
route1	14	1	-1	2	1	11	33	0
setExchangePattern1	14	0	0	0	0	0	46	0
choice1	14	1	-1	2	1	9	46	29
log1	0	0	0	0	0	0	17	0
bean1	0	0	0	0	0	0	17	0
log2	14	0	-1	0	0	1	17	5
bean2	14	1	0	0	0	2	12	12
log3	0	0	0	0	0	0	0	0
bean3	0	0	0	0	0	0	0	0

Session 3 : Management & Monitoring - 60min

Presentation of Fuse Management Console - Hawtio

Demos

Demo 1 - Use Fuse Management Console / Hawtio

- Open your web browser and point to the following URL address `http://localhost:8181/`

> The login/password `admin/admin` should be commented in the file etc/users.properties prior to start JBoss Fuse

- Add you login/password - admin/admin
- Verify that `camel` plugin is there
- Check content of the different plugins `log`, `osgi`, `threads`, `dashboard`
- Discover the camel plugin : route, context, attributes, source, diagram, operations, chart

Demo 2 - Trace and Debug camel routes

- Open the camelContext, next the route-1 and click on diagram

```
http://localhost:8181/hawtio/index.html#/camel/routes?tab=camel&nid=root-org.apache.camel-demo-camel-ws-routes-%22route1%22
```

- Send a SOAP Message `saveCustomerRequest, getAllCustomers` and check the counters changing within the route diagram
- Visualize the code source

```
http://localhost:8181/hawtio/index.html#/camel/source?tab=camel&nid=root-org.apache.camel-demo-camel-ws-routes-%22route1%22
```

- Next we will enable the tracing to get more info in the FMC console
- Click on `trace` tab and click on the button `Start tracing`
- Send a new SOAP Request - getAllCustomers
- 3 entries have been created in the table of the tracing. they belong to the same exchangeID
- Click on the the first and next on `header` button, you should see the camel headers, SOAPAction
- Navigate between the different entries, you should see in the diagram that the color change between the different processor indicating where you stand
- Stop the tracing (click on the button - stop tracing)
- Now we will use the debug option

- Select `debug` tab and click on the button `start debugging`

```
http://localhost:8181/hawtio/index.html#/camel/debugRoute?tab=camel&nid=root-org.apache.camel-demo-camel-ws-routes-%22route1%22
```

- Click on the choice processor. The color of the processor turns to `yellow` as a breakpoint has been added and is suspended
- Send a SOAPMessage using curl. You should see that the response has not been received

```
curl -X POST -H "SOAPAction: http://example.fuse.redhat.com/getAllCustomers" -T hands-on-lab/demo-camel-ws/src/main/resources/data/request-all.xml
http://localhost:9090/training/WebService | xmllint --format -
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
```

- Return to your browser and look to the table (headers)
- Next click to the button `step into the next node`
- You will move between the different processors
- To finish to process a message, click on the resume button
- To leave the debugger, click on `stop` button

Demo 3 - Create a dashboard for your project to monitor specifically a camel route

- Select the `dashboard` plugin and click on the `manage` button
- Click on the `+ create` button to add your camel dashboard, name it `my-camel` in the dashboard table
- Return to your camel plugin
- Select the camel route1

```
http://localhost:8181/hawtio/index.html#/camel/profileRoute?tab=camel&nid=root-org.apache.camel-demo-camel-ws-routes-%22route1%22
```

- Click on the `profile` tab that we have for the `route 1`, you will see the statistics of your camel route
- To use this information and add it to your personal dashboard, click on the button `add this view to a dashboard`. This button is located on the panel of your tabs (right part)
- We switch to the dashboard view,
- Select your `my-camel` dashboard by clicking on the checkbox and next Click on `Add view to dashboard`
- When this is done you should be redirected to the dashboard - `my-camel` and a small widget representing the profile of your camel is displayed
- Increase the size of the widget
- Send new requests and check the values

Session 5 : Fabric & Insight

Presentation of Fuse Fabric

Demos

Demo 1 - Deploy project (my-cool-camel-cxf) as a profile

- Using the JBoss Fuse command line, create the fabric server

```
fabric:create
```

- Add maven plugin `fabric8:deploy` to the pom.xml file the project
- And define the properties required to create the profile

```
<fabric8.profile>my-cool-profile</fabric8.profile>
<fabric8.features>camel camel-cxf</fabric8.features>
<fabric8.parentProfiles>feature-camel</fabric8.parentProfiles>
```

- Add a new `maven` configuration to JBDS and name it `fabric8`.
- Select the `demo-camel-ws` project. Right-click on the pom.xml file and select `Run as --> maven build ...`
- In the pop-up window displayed, name your maven build as `fabric8` and add the goal `fabric8:deploy`.
- Click on apply and button close

the admin/password login/password must be added to your settings.xml file of your maven config in order to run the plugin.

```
[ERROR] Please add the following to your ~/.m2/settings.xml file (using the correct user/password values):
[ERROR]
[ERROR] <servers>
[ERROR] <server>
[ERROR] <id>fabric8.upload.repo</id>
[ERROR] <username>admin</username>
[ERROR] <password>admin</password>
[ERROR] </server>
[ERROR] </servers>
```

Example of settings.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">
 <servers>
   <server>
      <id>fabric8.upload.repo</id>
      <username>admin</username>
      <password>admin</password>
   </server>
</servers>
</settings>
```

- Launch `maven fabric8` and wait till the build succeeds

```
Uploading: http://192.168.137.1:8080/maven/upload/com/redhat/fuse/demo-camel-ws/1.0/demo-camel-ws-1.0.pom
[INFO] About to invoke mbean io.fabric8:type=ProjectDeployer on jolokia URL: http://localhost:8181/jolokia with user: admin
[INFO] Got result: {"profileId":"my-cool-profile","versionId":"1.0"}
```

- Open the Fuse Management Console. You should see that a new perspective has been added `fabric`, select it

- Review concepts introduced by the instructor
- Select wiki and navigate to the profile `my-cool-demo` to verify that your bundle is well attached to the profile and that the profile is linked to the profile (camel & cxf)

```
http://localhost:8181/hawtio/index.html#/wiki/branch/1.0/view/fabric/profiles/my/cool/profile.profile
```

- Next click on the button `new` to add a new child container with this profile
- Call the container `my-cool-container`
- Verify that the profile is `my-cool-profile`.
- Click on `create and start container`
- You will see that the container will created and the bundles currently deployed

Demo 2 - Metrics

Objective : Install Insight + ElasticSearch to collect statistics/metrics of the camel routes. Play with Full Text Search of ElasticSearch (Lucene) to query some Camel routes (using Message ID)

- Download fabric8 - http://fabric8.io/#/site/book/doc/index.md?chapter=getStarted_md
- Select in the `fabric` perspective, the container `root` runtime and add the profile `insight-core` and `insight-kibana`
- After a few moments, you should be able to verify that the `insight` perspective has been added to the menu
- Create a new child container `my-cool-insight` and add the profile `insight-camel` & `insight-core`
- Now we can install a camel project. Add the profile *all* of **camel example loanbroker**
- Wait a few moments that the container is provisioned
- Open next the perspective `insight`, click on `camel events` tab and check the camel exchanges created
- Navigate between the columns of the table of ES
- Add or remove some columns
- Check the detail of a camel exchange
- Search about some exchanges using the Lucene FT Search

Session 4 : HA & Load Balancing

Presentation of High-Availability topologies and Loadbalancing

Demos

Demo 1 - HA and Network of brokers

Objective : Setup a Network of Brokers Topology and show how messages are persisted/processed

(to support HA scenario)

- We will follow instructions of the FuseByExample/projects to setup 2 master/slave ActiveMQ brokers in a network of brokers (west to east)
  - https://github.com/FuseByExample/external-mq-fabric-client
  - https://github.com/FuseByExample/external-mq-fabric-client/blob/master/docs/fabric-ha-setup-master-slave.md

Demo 2 - Loadbalancing

Objective : Turn on your Fabric/Fuse container into a load balancing platform to distribute workload

- Review the configuration of the cluster server (http://localhost:8181/hawtio/index.html#/wiki/branch/1.0/view/fabric/profiles/example/camel) and the camel route http://localhost:8181/hawtio/index.html#/wiki/branch/1.0/view/fabric/profiles/example/camel/cluster.serve
- Do the same but for the client - http://localhost:8181/hawtio/index.html#/wiki/branch/1.0/view/fabric/profiles/example/camel/cluster.clien
- Create 2 child containers using the `cluster server` profile

change 1 with 2 in the field `Number of containers` and name the containers `camel-cluster`

- Click on `Create and start containers`
- Verify that the 2 camel child containers are up and running
- Add now a new child container which is a client of the cluster
- Create a child container using the profile `cluster.client` and name the container `camel-client`
- Next connect to the child container and check the log to see that you get response from one the of the two servers

```
2014-05-05 17:13:52 INFOfabric-client >>> Response from Fabric Container : camel-cluster1
2014-05-05 17:13:53 INFOfabric-client >>> Response from Fabric Container : camel-cluster1
2014-05-05 17:13:54 INFOfabric-client >>> Response from Fabric Container : camel-cluster2
2014-05-05 17:13:55 INFOfabric-client >>> Response from Fabric Container : camel-cluster1
```

Session 6 : Complex Business Processing - 90 min

Presentation of Drools Rule Language and Drools plugin in JBoss Developer Studio

Demos

Demo 1 - Drools & Camel Project

Objective : Discover Drools, Kie and Camel technologies

- Creation of a knowledge project with Rules using JBoss Developer Studio

- Import project *demo-camel-drools*
- Review the code and build it `mvn install`
- Setup a JBoss Fuse - Fabric server and deploy the Drools project using a profile and fabric8:deploy plugin in a child container `my-cool-drools-container`
- Open the container created and check the log file where you should see messages

```
2014-05-06 09:49:51,657 | INFO  | imer://testRoute | Home                                | rg.apache.camel.util.CamelLogger  176 | 230 - org.apache.camel.camel-core -
2.12.0.redhat-610379 | Person Young Person is staying home
2014-05-06 09:50:01,659 | INFO  | imer://testRoute | Bar                                 | rg.apache.camel.util.CamelLogger  176 | 230 - org.apache.camel.camel-core -
2.12.0.redhat-610379 | Person Old Person can go to the bar
```

Demo 2 - Add new Rule and extend the camel route

- Next we will add a new rule and modify the camel route
- After selecting the *drl* directory, Select from the menu `New -> Other -> Drools -> Rule Resource`
- file name should be `favoriteCheese-rule.drl`
- Type of resource → `New rule`
- Define the `Rule package name` to `org.drools.camel.example`
- Click on `next` button
- Add 2 new rules to define favoriteCheese and import the Person object

```
package org.drools.camel.example
```

```
import org.drools.camel.example.Person;
```

```
rule "Cheddar cheese for old person"
    when
      p : Person( age >= 21 )
    then
      p.setFavoriteCheese("cheddar");
end
```

```
rule "Gouda cheese for young person"
    when
      p : Person( age < 21 )
    then
      p.setFavoriteCheese("gouda");
end
```

- Modify the camel route to control the `favorite cheese` selected

```
    <choice>
      <when id="CanDrink">
        <simple>${body.canDrink}</simple>
        <log logName="Bar" message="${body.name} can go to the bar"/>
        <to uri="direct://cheese"/>
      </when>
      <otherwise>
        <log logName="Home" message="${body.name} is staying home"/>
        <to uri="direct://cheese"/>
      </otherwise>
    </choice>
</route>
```

```
<route>
    <from uri="direct://cheese"/>
    <when>
      <simple>${body.favoriteCheese} == 'gouda'</simple>
      <log logName="Gouda" message="${body.name} prefers gouda cheese"/>
```

```
    </when>
    <when>
      <simple>${body.favoriteCheese} == 'cheddar'</simple>
      <log logName="Cheese" message="${body.name} prefers cheddar cheese"/>
    </when>
  </route>
```

- Rebuild the project `mvn clean install`
- Update the bundle within the `my-cool-drools-container`
- Verify in the log the messages displayed

```
2014-05-06 09:57:36,960 | INFO  | imer://testRoute | Bar                 | rg.apache.camel.util.CamelLogger  176 | 230 - org.apache.camel.camel-core -
2.12.0.redhat-610379 | Old Person can go to the bar
2014-05-06 09:57:36,961 | INFO  | imer://testRoute | Cheese              | rg.apache.camel.util.CamelLogger  176 | 230 - org.apache.camel.camel-core -
2.12.0.redhat-610379 | Old Person prefers cheddar cheese
```

```
2014-05-06 09:57:56,963 | INFO  | imer://testRoute | Home                | rg.apache.camel.util.CamelLogger  176 | 230 - org.apache.camel.camel-core -
2.12.0.redhat-610379 | Young Person is staying home
2014-05-06 09:57:56,964 | INFO  | imer://testRoute | Gouda               | rg.apache.camel.util.CamelLogger  176 | 230 - org.apache.camel.camel-core -
2.12.0.redhat-610379 | Young Person prefers gouda cheese
```

This is the end.

We hope that you have enjoyed and appreciated !