

**Achieving True Agility**

**Through Microservices,**

**Containers, and APIs**

Red Hat Workshop

November 2017

**Table of Contents**

[**Introduction**](#_lcwz9tpcivov) **3**

[**Lab 1 - Create your first SpringBoot Fuse project**](#_8qezorwuog6x) **5**

[**Lab 2 - Expose an API endpoint in Fuse**](#_2qhw8vhimspt) **13**

[**Lab 3 - Deploying the API endpoint on OpenShift**](#_umiq4li0ry63) **16**

[**Lab 4 - Deploy a responsive web app using Node.js**](#_g7cpgpb9dc6l) **26**

[**Lab 5 - CI / CD with OpenShift**](#_45nd09z4c1fm) **31**

[**Lab 6 - Integration with API Management**](#_nusj7iic96de) **38**

[**Appendix**](#_716wst4xy67l) **53**

# Introduction

This very simple lab will guide you to create your very first Fuse Spring Boot project running on OpenShift. There are 4 sections in the lab.

* Create a microservice that reads from a database
* Expose a restful API endpoint to access microservice
* Use API management to access the microservice API
* Deploy your application on OpenShift

**3scale SaaS Setup**

In order to complete the labs you will need to have your account in the Red Hat 3Scale SaaS. If you do not have a Red Hat 3Scale SaaS account, please register for a free trial account at: <https://www.3scale.net/signup/>.

**GitHub Setup**

You will also need a GitHub account to store and share your source code that you will need throughout the labs. You can register for free at: <https://github.com>

**Installing and Setup of OpenShift**

1. Start the VirtualBox VM application and start the workshop image named “Agile Integration”.
2. The Login details are:
   * User name: JBoss
   * Password: Workshop!23
3. In the Home directory of the image there are two files to start and stop the OpenShift environment:
   * startOpenshift.sh
   * stopOpenshift.sh
4. Open a Terminal (under Applications). Ensure you are in the jboss user home directory and start Openshift.

pwd   
/home/jboss

sh startOpenshift.sh

or

./startOpenshift.sh

1. Log into OpenShift on the command line using the system account

oc login -u system:admin

1. Install Fuse image stream on OpenShift and the MYSQL database template for this lab

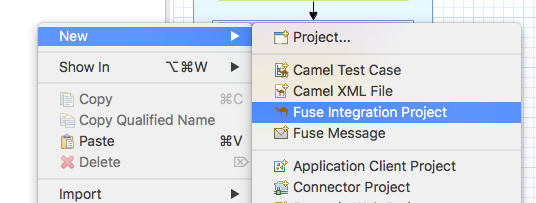
oc create -f https://raw.githubusercontent.com/jboss-fuse/application-templates/master/fis-image-streams.json -n openshift

oc create -f https://raw.githubusercontent.com/openshift/origin/master/examples/db-templates/mysql-ephemeral-template.json -n openshift

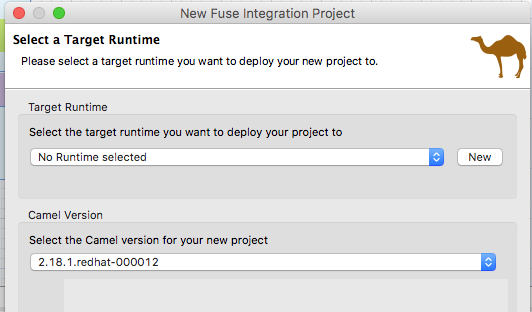
# Lab 1 - Create your first SpringBoot Fuse project

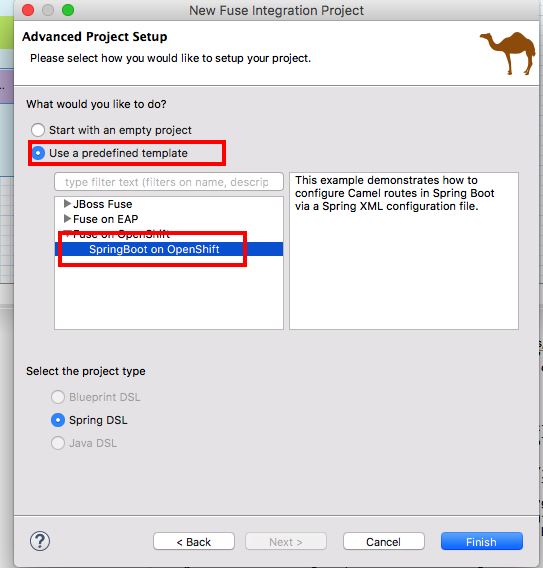
In this lab you will create a SpringBoot based application and deploy it inside your JBoss Developer Studio environment.

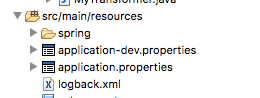
1. Start JBoss Developer Studio in the **Applications -> Programming** menu, and accept the default workspace.
2. In JBoss Developer Studio, create a new project by right click in the project explorer panel, select **New** -> **Fuse Integration Project**



1. Enter **myfuselab** as the project name, and click **Next**.   
   For **Select Target Runtime**, click **Next**



1. At **Advance Project Setup**, choose **Use a predefined template** and select **Fuse on OpenShift** -> **SpringBoot on OpenShift** and click **Finish**.   
     
     
     
   Creating the project will take some time since JBoss Developer Studio has to download dependencies from the Internet. This can take a while.
2. Under **Project Explorer -> myfuselab -> src/main/resources** duplicate **application.properties** and with name **application-dev.properties**, we are going to use this file for configuring our application properties.



1. Append at the end the following datasource configuration to file **application-dev.properties**.

# Database configuration

spring.datasource.url = jdbc:h2:mem:mydb;DB\_CLOSE\_DELAY=-1;DB\_CLOSE\_ON\_EXIT=FALSE

spring.datasource.username = sa

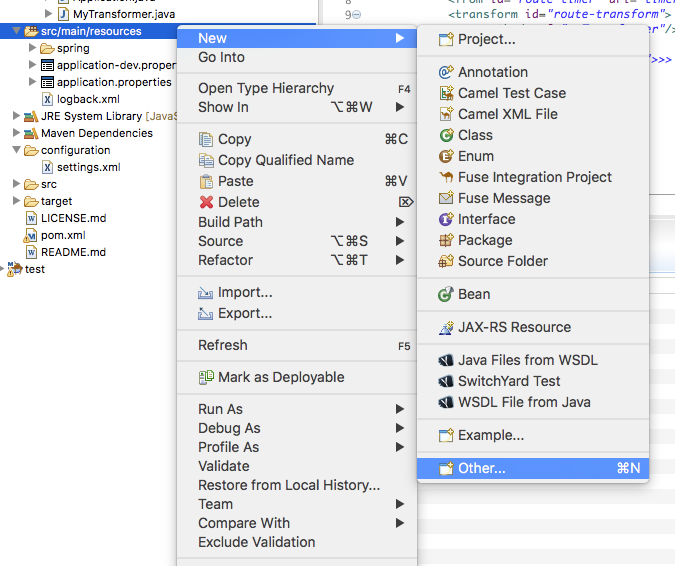
spring.datasource.password =

spring.datasource.driver-class-name = org.h2.Driver

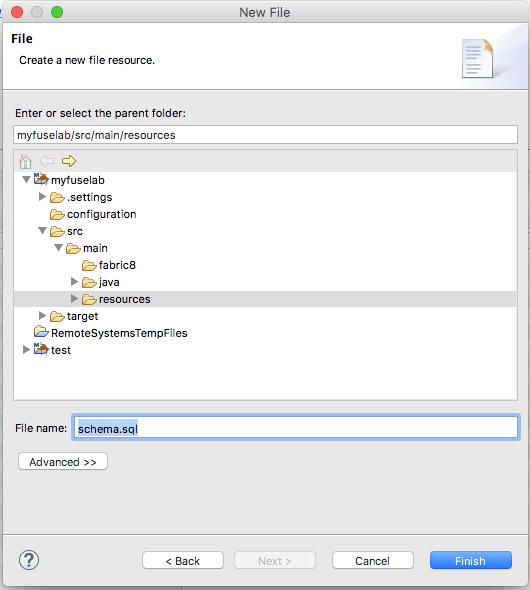
spring.datasource.platform = h2

*Note: we are using H2 in memory database for testing. And thanks to autowiring in Spring Boot, it is now automatically loaded and wired as the default datasource to the Camel context.*

1. Create new file under **src/main/resources** by right click on the folder itself in the project explorer panel, select **New** -> **Other**



1. In **Select a wizard**, choose **General -> File** and click **Next**.
2. In **File**, put **schema.sql** as the file name, and make sure it is located under  **myfuselab/src/main/resources** and select **Finish**.



1. Add the following SQL to **schema.sql** and save the changes.

DROP TABLE IF EXISTS customerdemo;

CREATE TABLE customerdemo (

customerID varchar(10) NOT NULL,

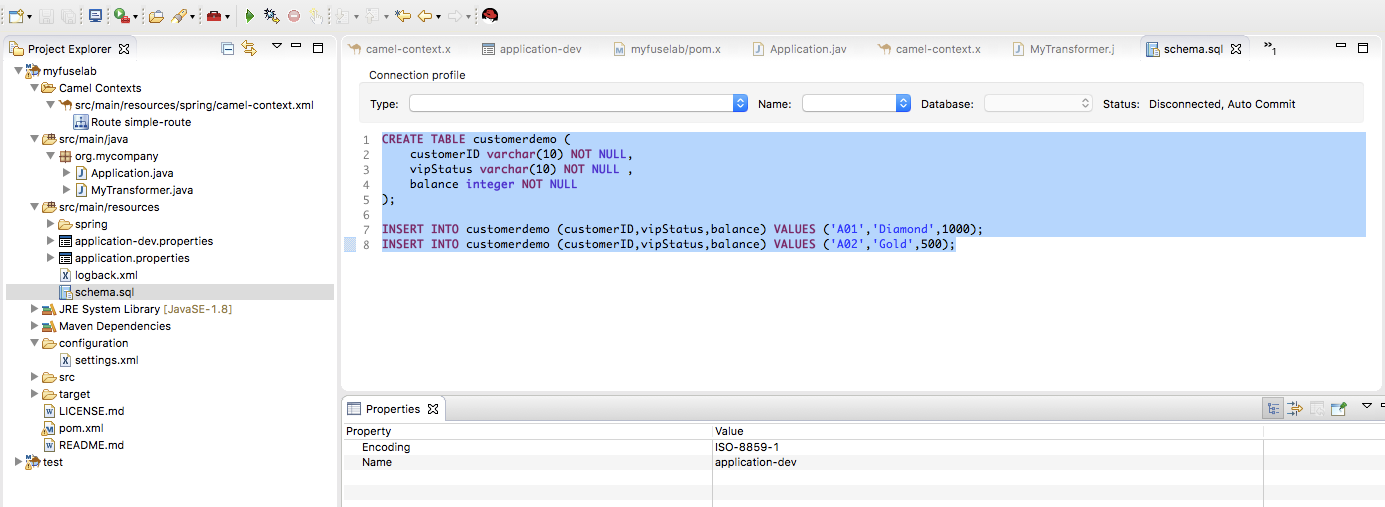
vipStatus varchar(10) NOT NULL ,

balance integer NOT NULL

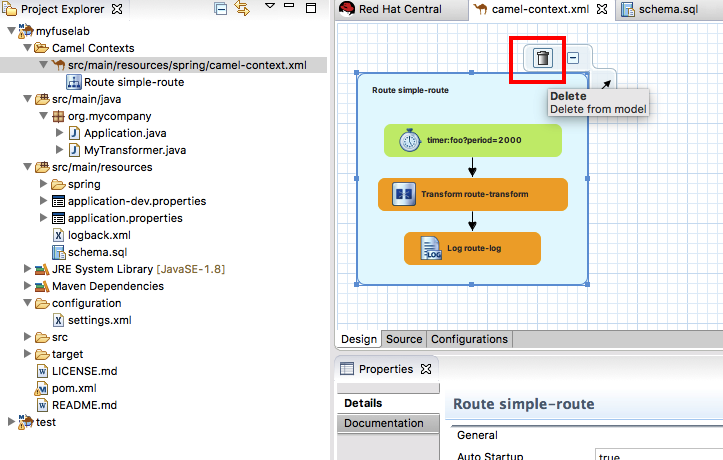
);

INSERT INTO customerdemo (customerID,vipStatus,balance) VALUES ('A01','Diamond',1000);

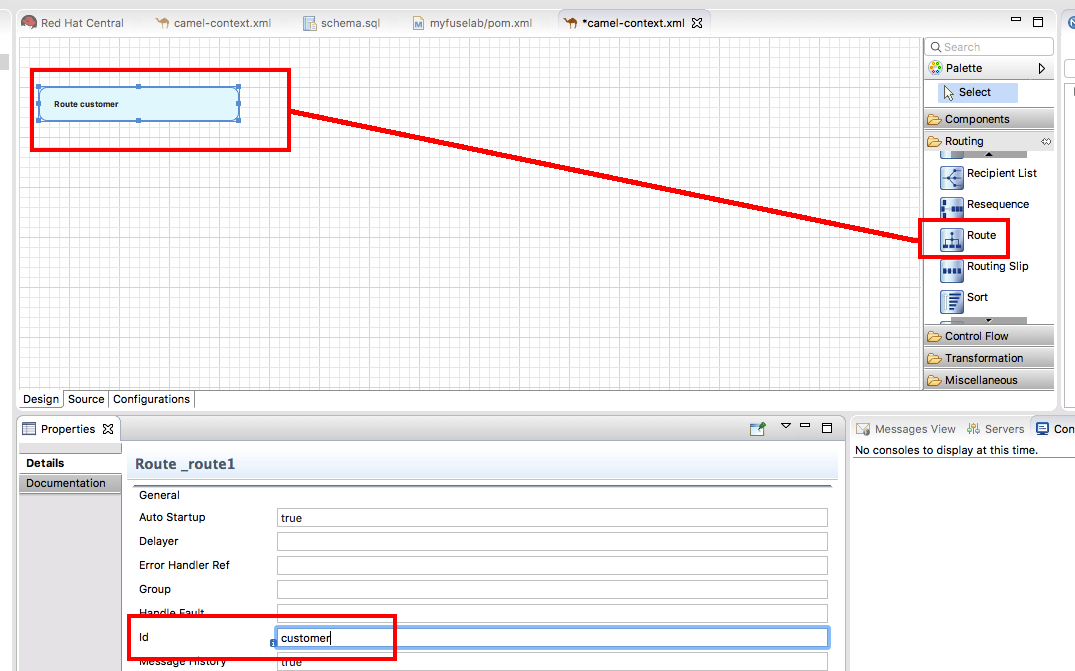
INSERT INTO customerdemo (customerID,vipStatus,balance) VALUES ('A02','Gold',500);



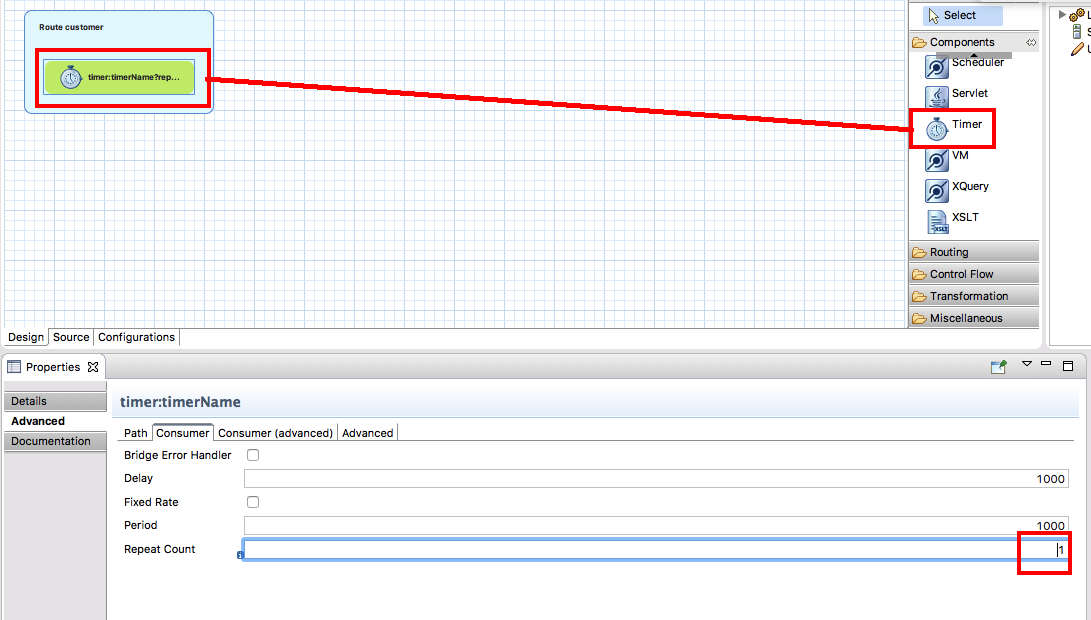
1. Double click on the **camel-context.xml** file under **Project Explorer ->** **Camel Contexts**, you will see the Camel route, delete the **simple-route** in the canvas.



1. Create a new route by dragging a **ROUTE** component from the **Routing** palette on the right. Name the route to **customer** by entering it in the **ID** textbox in the properties section.

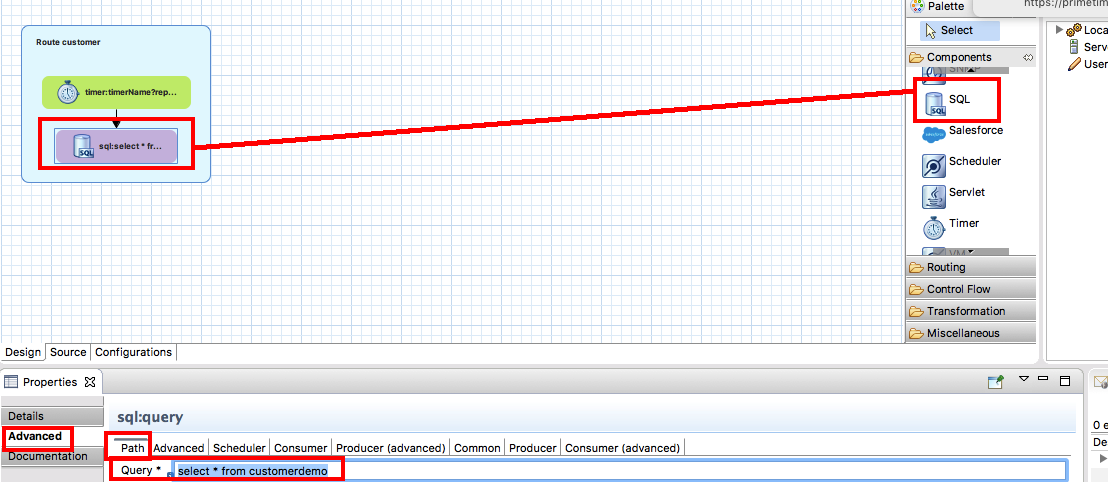


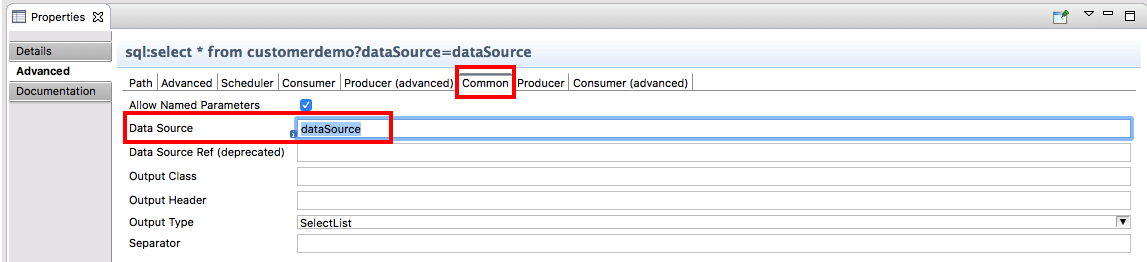
1. Use a **Timer** component to kick start the route by dragging the **TIMER** component from the **Components** palette on the right, and drag it to the route into the canvas. Select the **timer:timerName** element just added, and under **Properties-> Advanced -> Consumer** , set **Repeat Count** to **1**



Select **Save**.

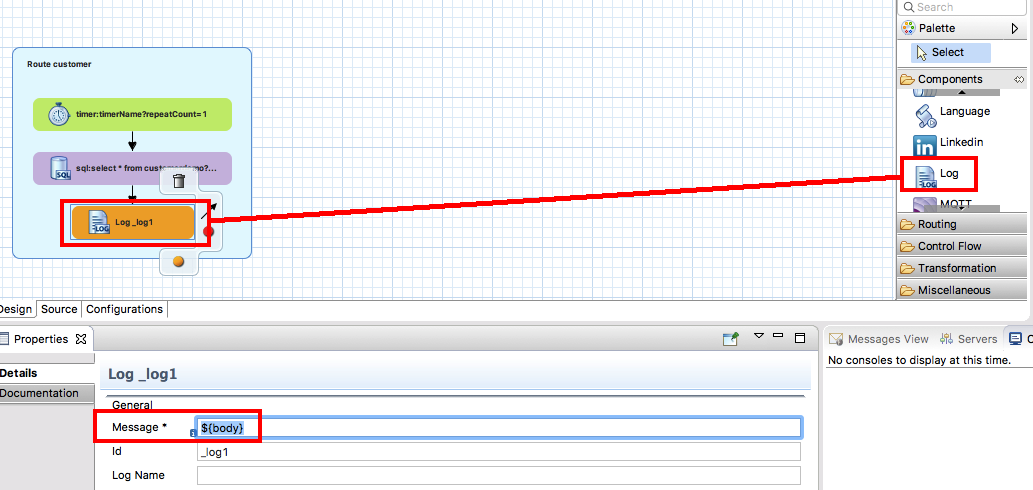
1. To read data from the data source, select a **SQL** component from the **Components** palette on the right to the route. Under **Properties-> Advanced -> Path**, set **Query** to  
     
    ***select \* from customerdemo***



1. In Common tab set Data Source to **dataSource** - ensure that the "d" is lower case

Select **Save**.

1. And lastly select a **LOG** component from the **Components** palette to the end of route. Under *Properties*-> *Detail* tab, set **Message** to **${body}** - these are curly brackets - and **Save.**



1. Before we start the application, add the database driver dependency in the **pom.xml** file (located in the **myfuselab** project's root directory) and save it.  
     
   Ensure you put the driver under the <dependencies> section in the root, and **not** under <dependencyManagement>.

...

<properties>

<run.profiles>dev</run.profiles>

...

</properties>

...

<dependencies>

...

<dependency>

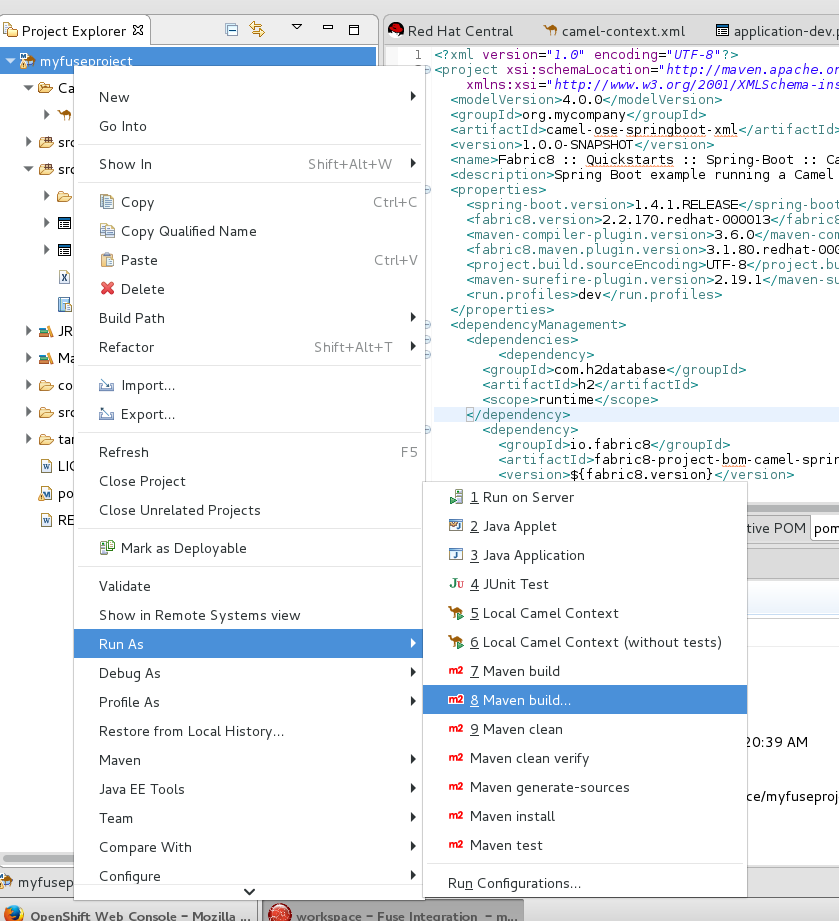
<groupId>com.h2database</groupId>

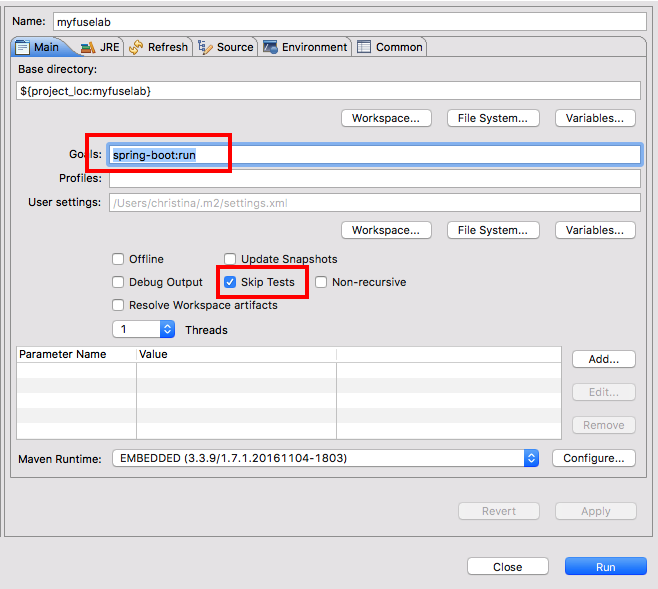
<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

</dependencies>

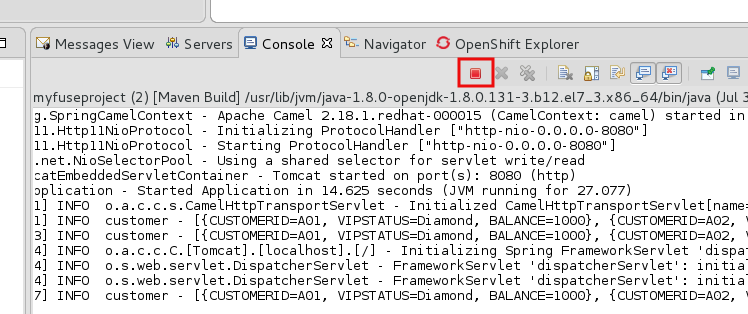
1. Right click on the **myfuselab** in the project explorer panel,   
   select **Run As..** -> **8 Maven build**  
     
     
   *Note: After you have done this the first time, subsequent builds can be done with the Run As -> Run Configurations, or Run As -> 7 Maven build options and choose this myfuselab build run that you create during this step.*

In the pop-up window enter **spring-boot:run** in **Goals** and select the **Skip Tests** checkbox, then select **Apply** followed by **Run**.

*Note: this may take a while the first time the project is built, as JBoss Developer Studio may have to download Maven dependencies from the Internet.*

1. In the **Console** log, verify that customer data is printed.

customer - [{CUSTOMERID=A01, VIPSTATUS=Diamond, BALANCE=1000}, {CUSTOMERID=A02, VIPSTATUS=Gold, BALANCE=500}]

1. **Important:** Terminate your application, by clicking the **Terminate** button in the console window.  
     
   
2. You have successfully completed this lab by creating a simple SpringBoot based microservice in a JBoss development environment.

# Lab 2 - Expose an API endpoint in Fuse

In this lab you will expose your SpringBoot application as a RESTful API.

1. Start JBoss Developer Studio and go to your **myfuselab** project.
2. To expose an HTTP API endpoint, we first have to inject a Servlet into the Camel context.   
     
   Go to the **camel-context.xml** file under **Camel Contexts**, open the **source** tab, add the following code snippet before the <camelContext..> tag.

...

<bean class="org.apache.camel.component.servlet.CamelHttpTransportServlet" id="camelHttpTransportServlet"/>

<bean class="org.springframework.boot.web.servlet.ServletRegistrationBean" id="servlet">

<property name="name" value="CamelServlet"/>

<property name="servlet" ref="camelHttpTransportServlet"/>

<property name="urlMappings" value="/myfuselab/\*"/>

</bean>

...

1. In the same file, under the <camelcontext..> and before the <route id="customer"> tags add the following code snippet to configure the REST endpoint. So that it is now using the Servlet we have injected from last step.

...

<restConfiguration apiContextPath="api-docs" bindingMode="json"

component="servlet" contextPath="/myfuselab">

<apiProperty key="cors" value="true"/>

<apiProperty key="api.title" value="My First Camel API Lab"/>

<apiProperty key="api.version" value="1.0.0"/>

</restConfiguration>

...

1. We are now going to expose a single API endpoint, right after the **restConfiguration** add and before <route id="customer">.

...

<rest path="/customer">

<get uri="all">

<description>Retrieve all customer data</description>

<to uri="direct:getallcustomer"/>

</get>

</rest>

...

1. Now, instead of triggering the database select query with a timer, we are going to trigger it by the API call. In **camel-context.xml** select the **Source** tab, replace the **Timer** with **Direct** component.

replace

<from id="\_from1" uri="timer:timerName?repeatCount=1"/>

with

<from id="direct1" uri="direct:getallcustomer"/>

Save your **camel-context.xml** file.

1. Next up, we are going to add all the dependencies needed to the maven **pom.xml** file and save it.

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.apache.camel</groupId>

<artifactId>camel-servlet-starter</artifactId>

</dependency>

<dependency>

<groupId>org.apache.camel</groupId>

<artifactId>camel-jackson-starter</artifactId>

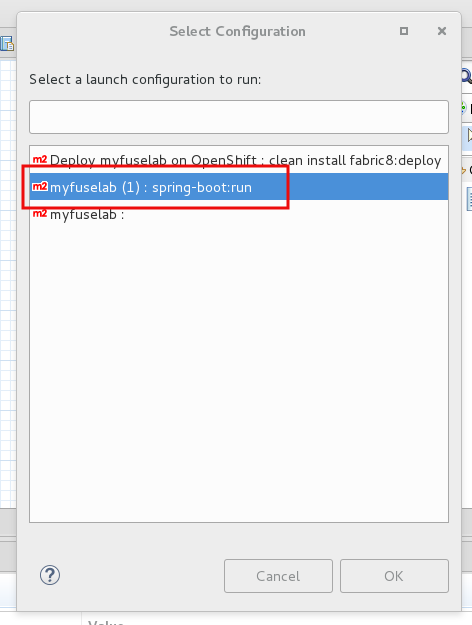
</dependency>

<dependency>

<groupId>org.apache.camel</groupId>

<artifactId>camel-swagger-java-starter</artifactId>

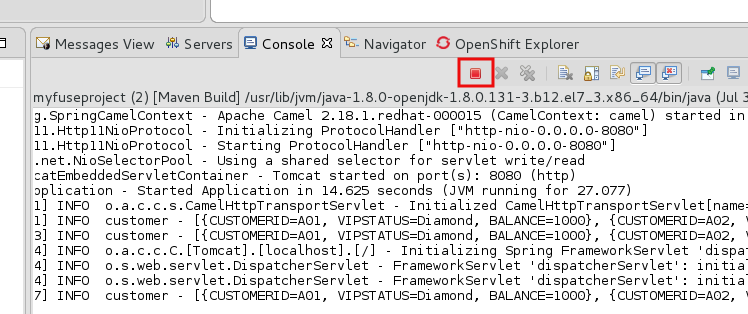
</dependency>

1. In the [Appendix](#_716wst4xy67l) of these instructions you can find a valid example of the [**camel-context.xml**](#_okaued48b9p9) file if you need to compare.
2. Make sure you have terminated any running versions of **myfuselab** else you will get a port conflict error.
3. Right click on the **myfuselab** in the project explorer panel, select **Run As..** -> **7.** **Maven build**, from the **Select Configuration** window choose the **myfuselab:spring-boot:run** to start up the Camel application again.   
     
   
4. Check the log output in the console and verify that the application is running.  
     
   Started Application in 13.332 seconds (JVM running for 29.401)
5. To test, run the following command in your terminal window on the command line.

curl -i http://localhost:8080/myfuselab/customer/all

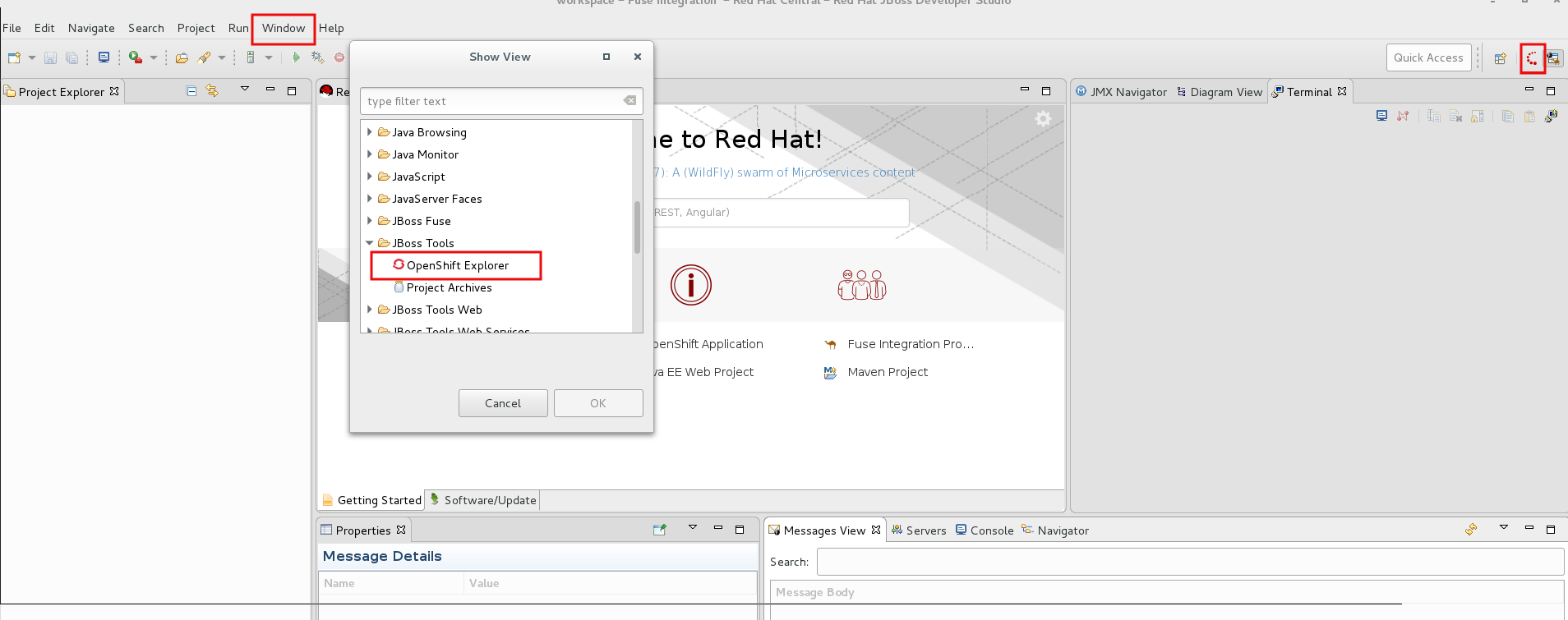
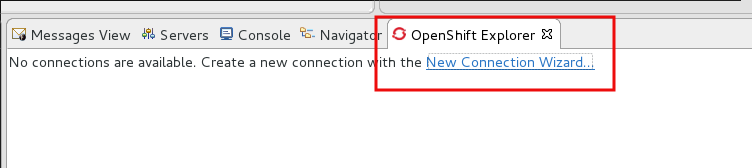
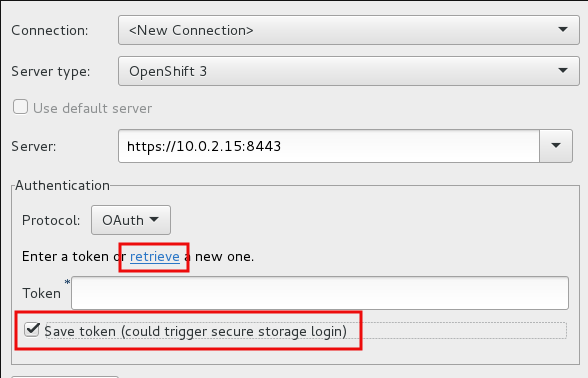
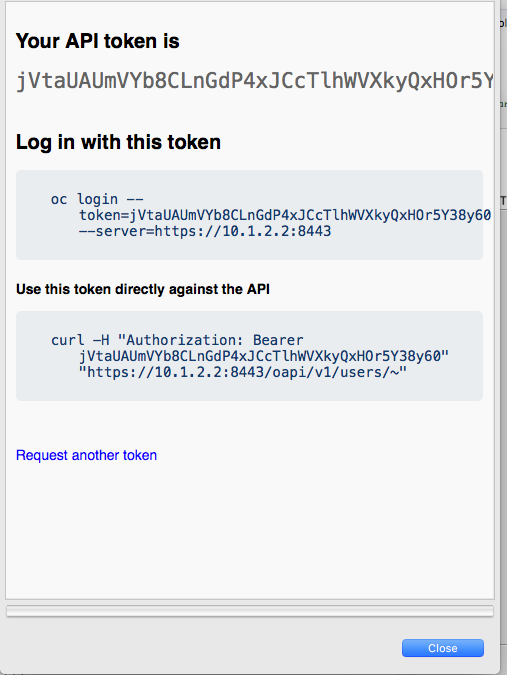
1. Verify that it is returning a list of customer data in JSON format

[{"CUSTOMERID":"A01","VIPSTATUS":"Diamond","BALANCE":1000},{"CUSTOMERID":"A02","VIPSTATUS":"Gold","BALANCE":500}]

1. Terminate your application, by clicking the **Terminate** button in the console.  
     
   
2. You have successfully completed this lab by exposing your microservice as a RESTful API.
3. Before you continue with the next lab, create a snapshot of your work in VirtualBox by going to **Machine -> Take Snapshot…** and naming it "Snapshot Lab 2".

# Lab 3 - Deploying the API endpoint on OpenShift

In this lab you will deploy your application to run on OpenShift. Ensure you have followed the steps for [**Install and Setup of OpenShift**](#_lcwz9tpcivov)found in the setup section at the start of this document.

1. Ensure that both OpenShift and JBoss Developer Studio are running.
2. In JBoss Developer Studio, open the OpenShift Explorer view (either by clicking on the JBoss Perspective near the top right-hand corner or selecting **Window -> Show View -> Other -> JBoss Tools -> OpenShift Explorer**).  
     
     
     
     
     
   Click on **New Connection Wizard..** to configure OpenShift setting Enter https://10.0.2.15:8443 as the **Server.**
3. In the Authentication section,select the option to **Save Token** and click on the **retrieve** link to access the token.
4. In the popup window, log in as Developer using the username and password combination **developer/developer**  
     
   
5. Select **Close** and then **Finish** to save the configuration.
6. In the "Untrusted SSL Certificate" window, click "Yes" and provide a Secure Storage Password of your choice ("*Workshop!23*" for example) in the next screen. Optionally you can provide a password hint.
7. We have been testing with the H2 Database in memory, now it's time to run it with a real database. It is possible and common to run a database external to OpenShift, however for this lab we will deploy a mysql database running in a container on OpenShift.  
     
   Add the following datasource setting under *src/main/resources* in **application.properties** and save the file.

# mysql specific

mysql.service.name=mysql

mysql.service.database=sampledb

mysql.service.username=dbuser

mysql.service.password=password

# Database configuration

spring.datasource.url = jdbc:mysql://${${mysql.service.name}.service.host}:${${mysql.service.name}.service.port}/${mysql.service.database}

spring.datasource.username = ${mysql.service.username}

spring.datasource.password = ${mysql.service.password}

1. Since we will be using MYSQL database, add the driver dependency in **pom.xml** and save it.

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

</dependency>

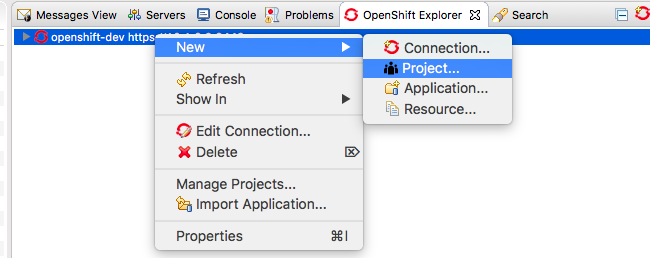
<dependency>

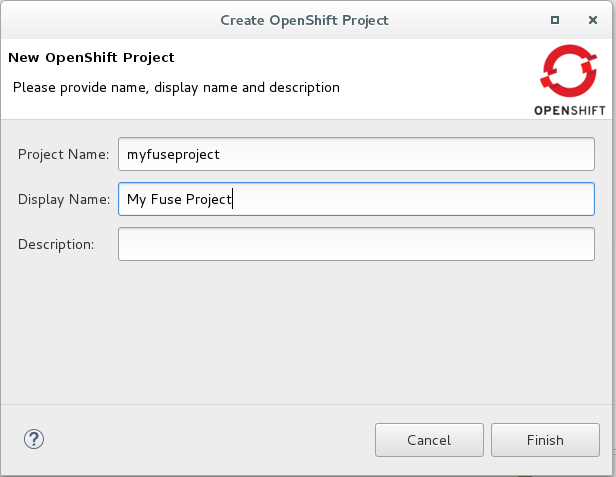
<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

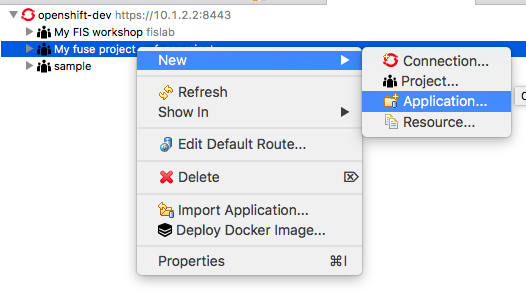
<scope>runtime</scope>

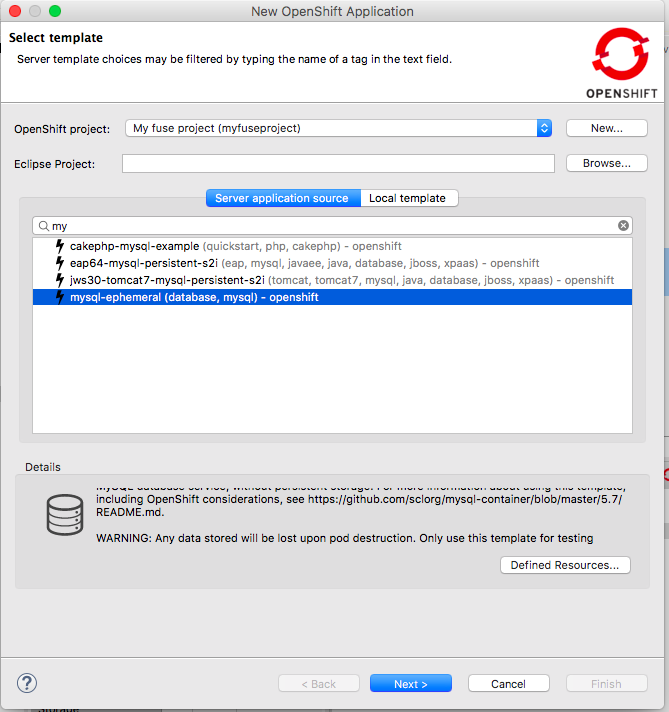
</dependency>

1. In the [Appendix](#_716wst4xy67l) of these instructions you can find a valid example of the [**pom.xml**](#_l25ynmfaih1c) file if you need to compare.
2. Open the OpenShift Explorer view to configure OpenShift through JBoss Developer Studio.
3. Right click on the connection that connects to current OpenShift, and create a new project. **New** -> **Project**
4. And create Project Name: **myfuseproject** with Display Name: **My Fuse Project.**Click **Finish.** The new project is now created on your OpenShift instance.



1. Inside the project we are going to first create a MYSQL database for our application, right click on the new project name **My Fuse Project** (**myfuseproject)** -> **New** -> **Application**

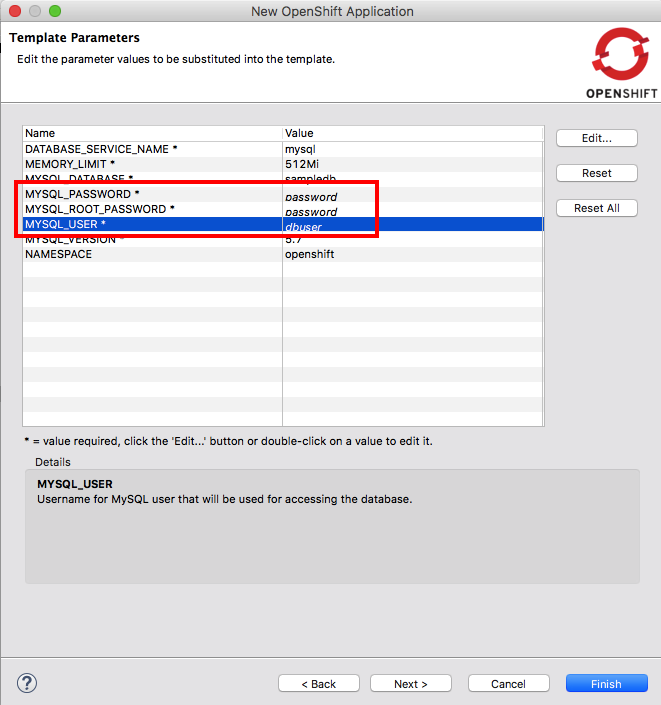


1. Under Server application source, select **mysql-ephemeral (database, mysql) - openshift** and click **Next**.
2. Make sure to configure the following parameters

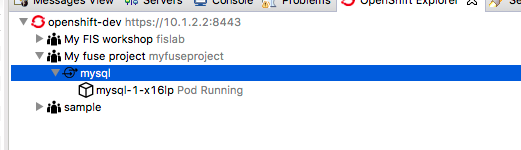
MYSQL\_PASSWORD = password

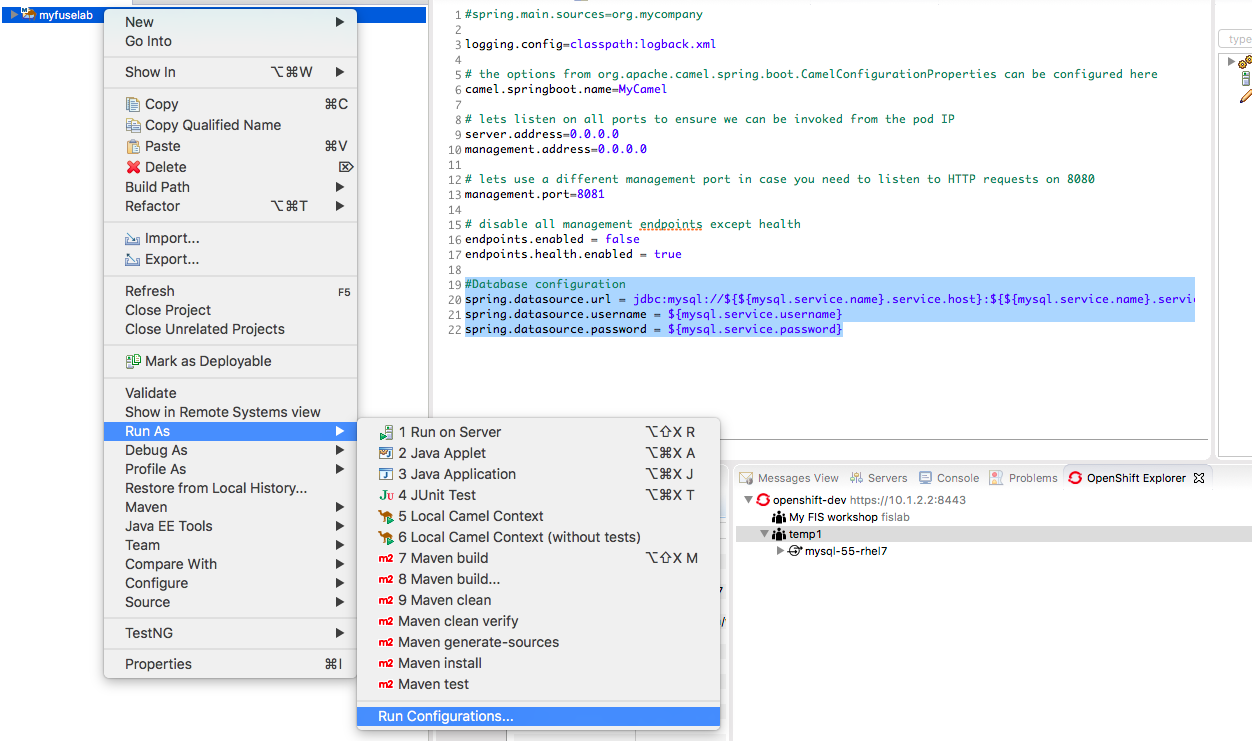
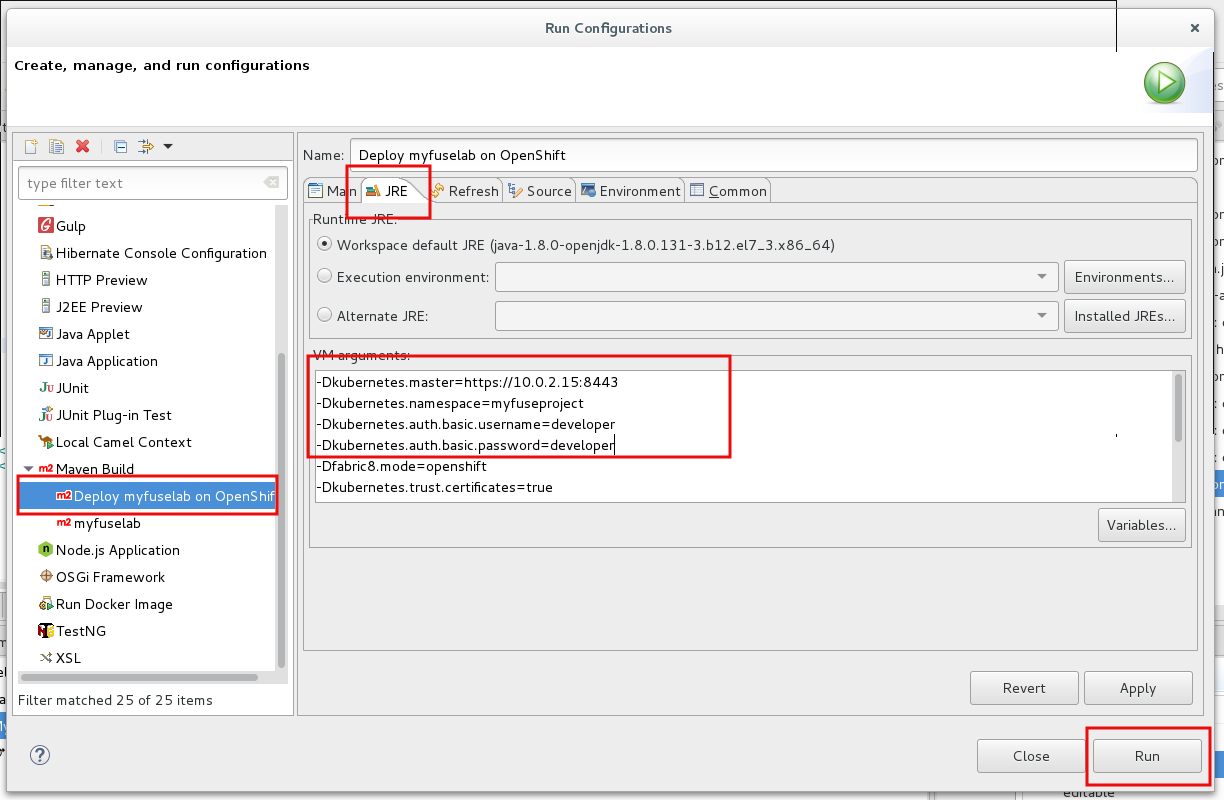
MYSQL\_ROOT\_PASSWORD = password

MYSQL\_USER = dbuser

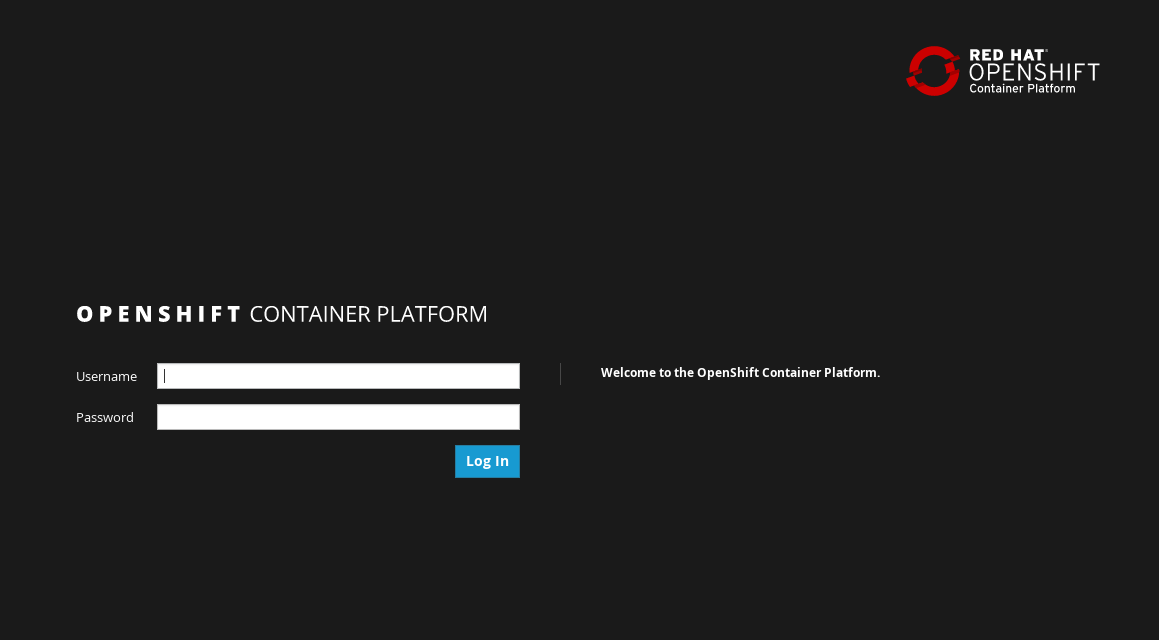


1. Click **Finish**, and confirm OK on the **Create Application Summary** screen.  
   You should see the mysql instance running in OpenShift Explorer.

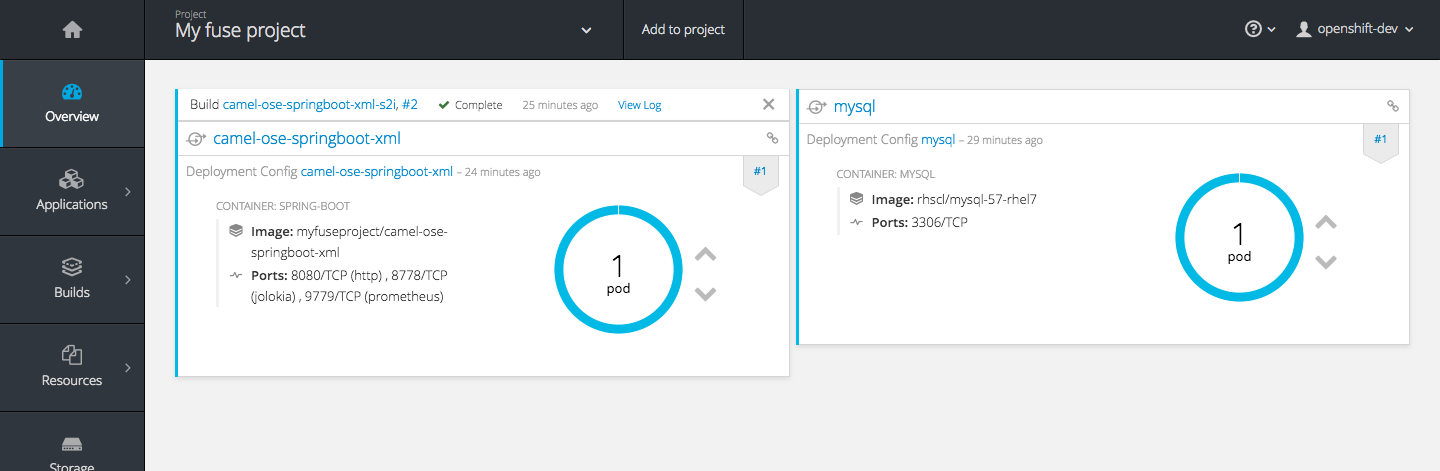
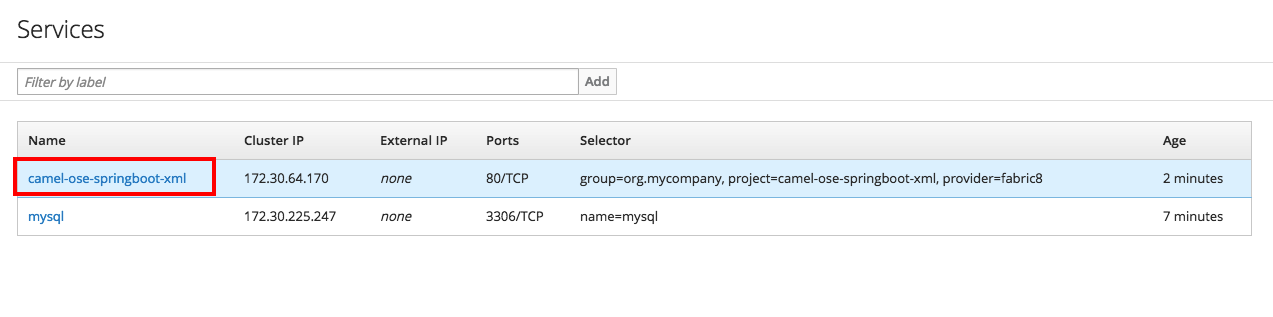
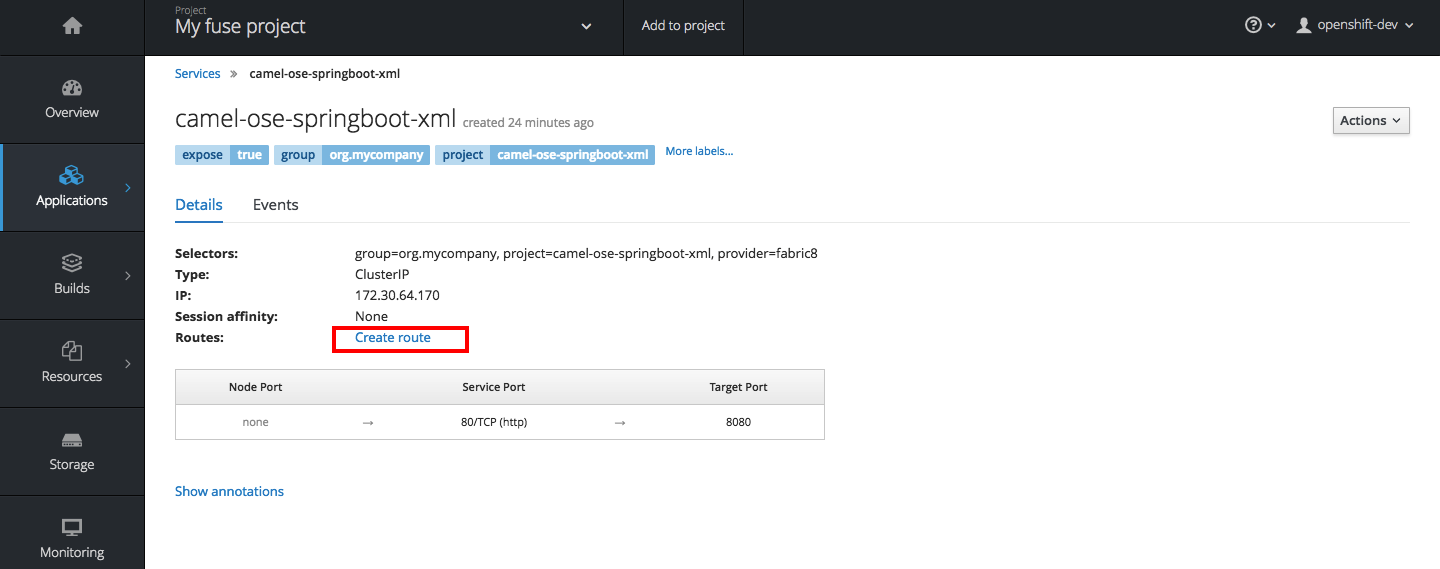


1. Now we can finally push our application from JBoss Developer Stuido to OpenShift by right clicking on your **myfuselab** in Project Explorer. Select **Run As**-> **Run Configurations…**
2. In the pop-up menu, select **Maven Build -> Deploy myfuselab on OpenShift** on the left panel. Go to **JRE** tab on the right, inside VM arguments  
     
    - update *kubernetes.master* to your OpenShift host name  
    (**https://10.0.2.15:8443**)  
    - update *kubernetes.namespace* to **myfuseproject**  
    - update *kubernetes,auth.basic.username* to **developer  
    -** update *kubernetes,auth.basic.password* to **developer**  
     
   Click on **Apply** and then on **Run**.  
     
   
3. If not already opened, open the web console for your OpenShift cluster in your browser: **https://10.0.2.15:8443/console/**

You should see the login screen:



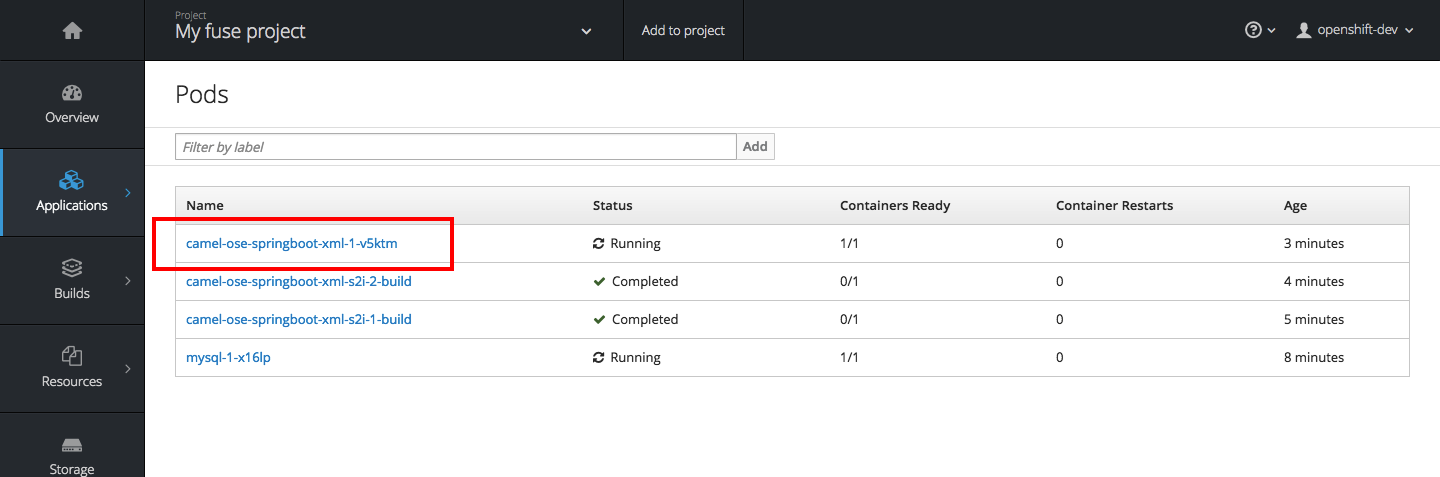
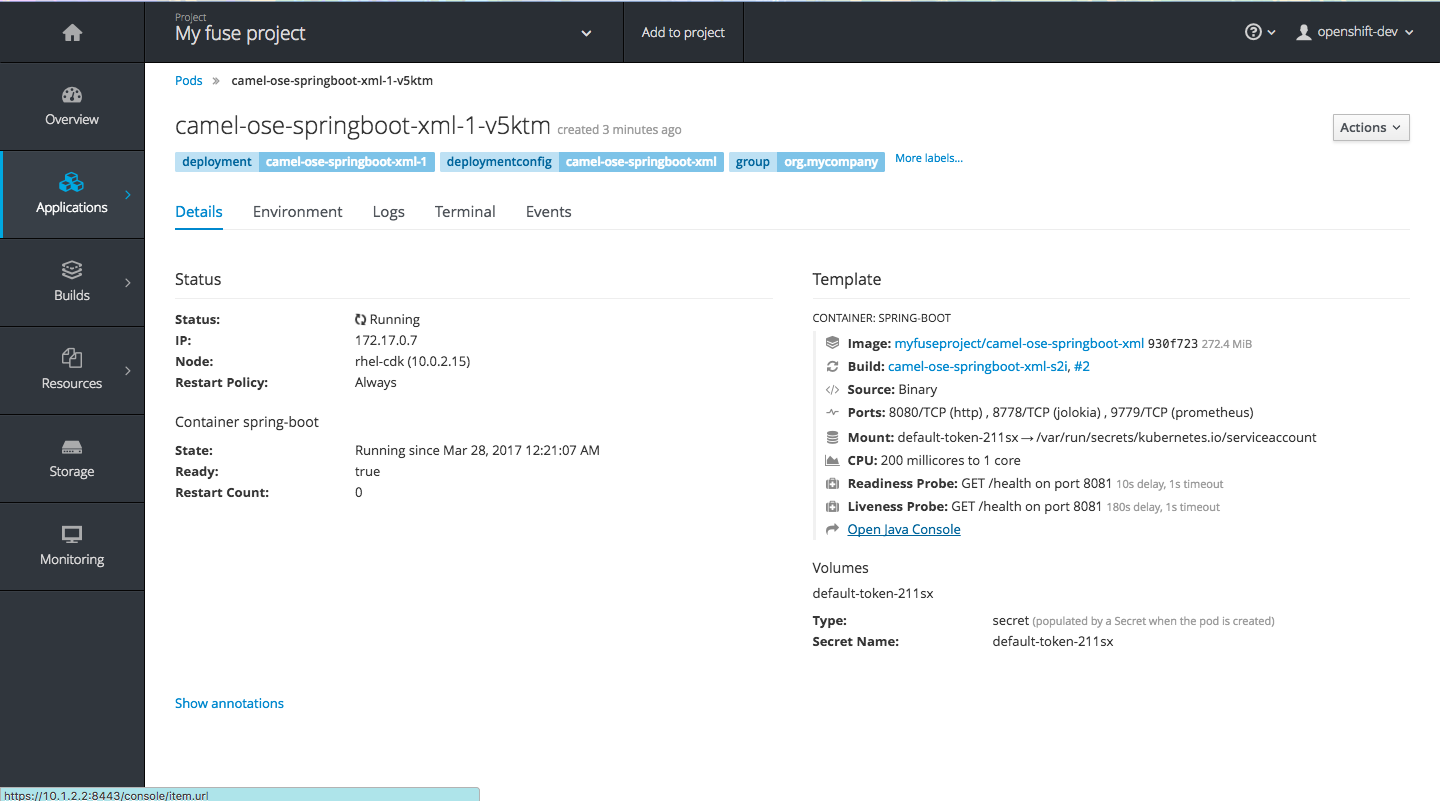
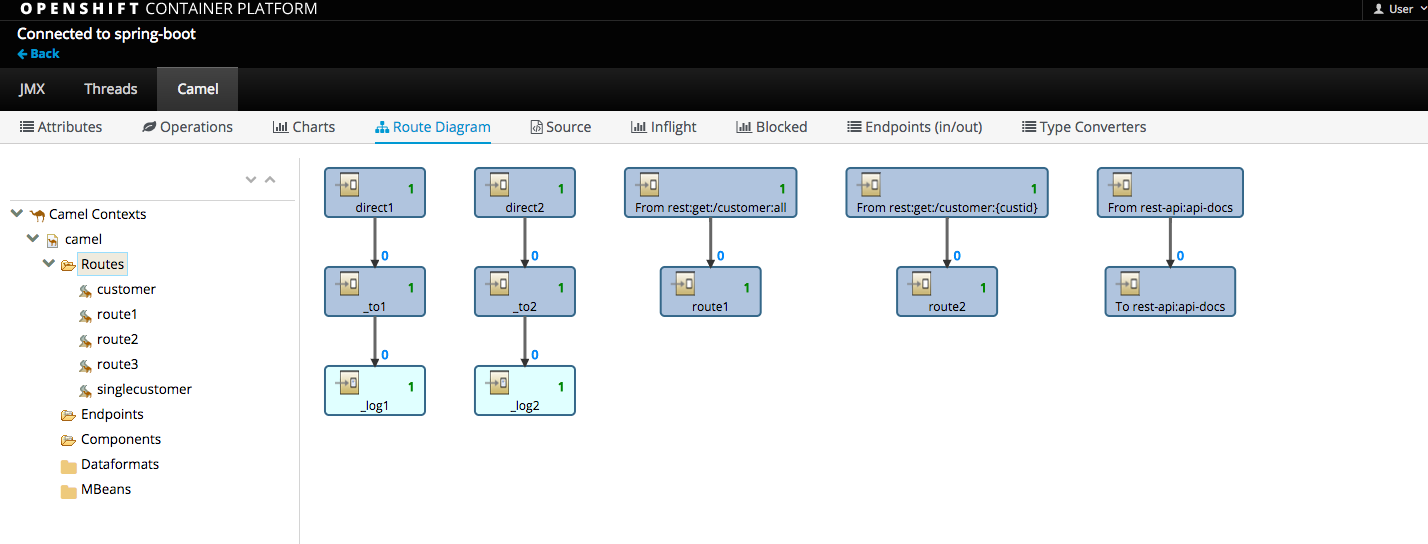
**Note:** You may receive a warning about an untrusted web-site. This is expected, as we are trying to access the web console through secure protocol, without having configured a valid certificate. While you should avoid this in production environment, for this test setup you can go ahead and create an exception for this address.

1. Login with the username **developer** and the password **developer**.
2. Select **My Fuse Project**. Once the deployment process from JBoss Developer Studio is finished, you will see both applications in the overview page.  
     
   
3. To access the service outside OpenShift, go to **Application** -> **Service** on the left menu, and click **camel-ose-springboot-xml** in the service page.
4. Click on **Create route**.
5. Don't change anything and hit Create.
6. Access the API endpoint by going to following URL of your route and invoking the API, for example

curl http://camel-ose-springboot-xml-myfuseproject.10.0.2.15.xip.io/myfuselab/customer/all

1. Verify that it is returning customer data in JSON format

[{"CUSTOMERID":"A01","VIPSTATUS":"Diamond","BALANCE":1000},{"CUSTOMERID":"A02","VIPSTATUS":"Gold","BALANCE":500}]

1. To see the Camel route in action, in your OpenShift console, go to **Applications** -> **Pods** and select the first **camel-ose-springboot-xml-1-xxxxx** pod.
2. Click **Open Java Console**, and it's going to take you to the individual console that show how your Camel route is doing.
3. Click on **Route Diagram** to inspect the route.
4. Click on the **<-back** link to return to the project view.
5. You have successfully completed this lab by deploying your microservice in your local OpenShift instance.
6. Before you continue with the next lab, create a snapshot of your work in VirtualBox by going to **Machine -> Take Snapshot…** and naming it "Snapshot Lab 3".
7. **Optional exercise: Viewing Inflight Count**

The Inflight count is shown by the small blue counter next to the route arrows above. This shows the number of routes that are in progress. If you would like to see this counter in action, we can force this by adding a delay into our route.

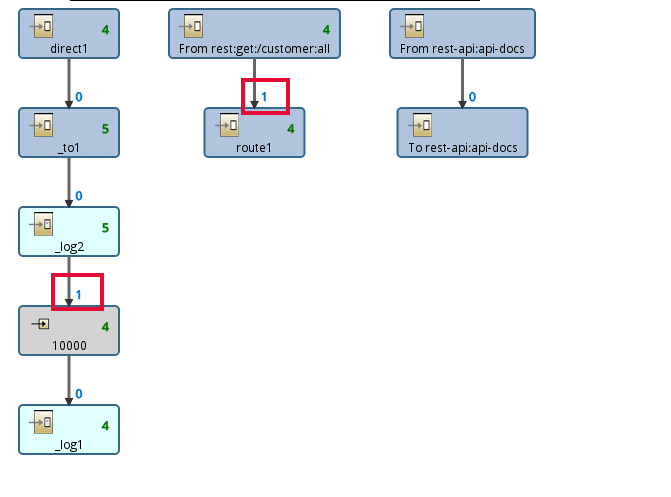
1. Go back to JBoss Developer Studio, select the camel-context.xml source.
2. Add the following text to add a delay component to your route before <log id=”\_log1” … /> tag.

<log id="\_log2" message="waiting for a delay"/>

<delay id="\_delay1">

<constant id="wait">10000</constant>

</delay>

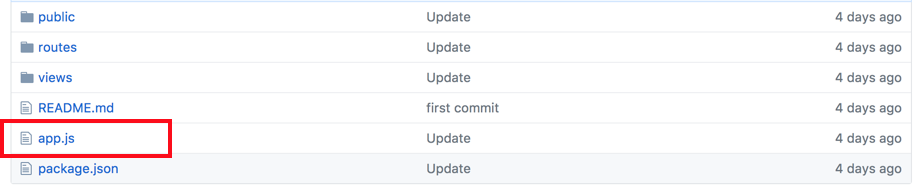
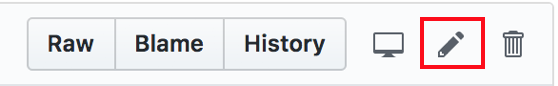
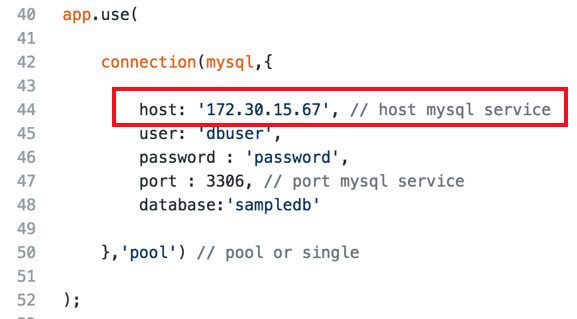
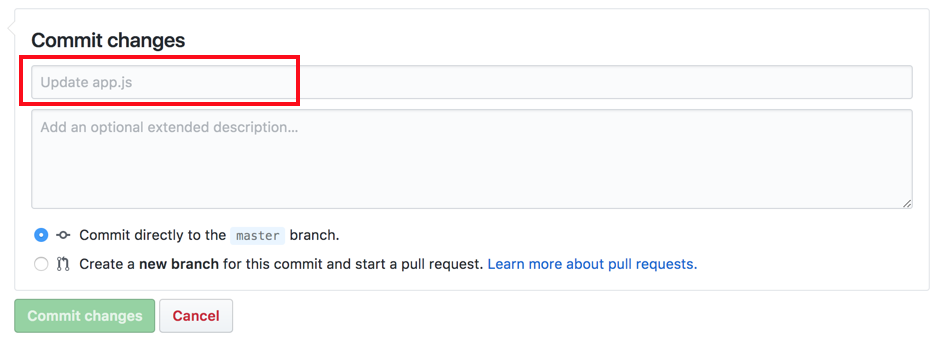
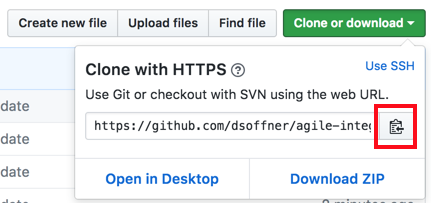
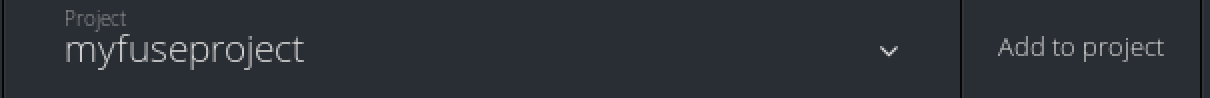
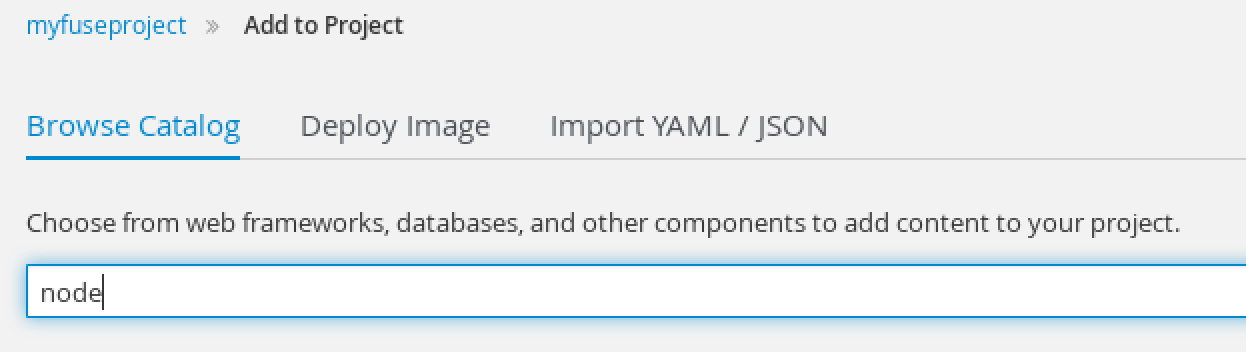
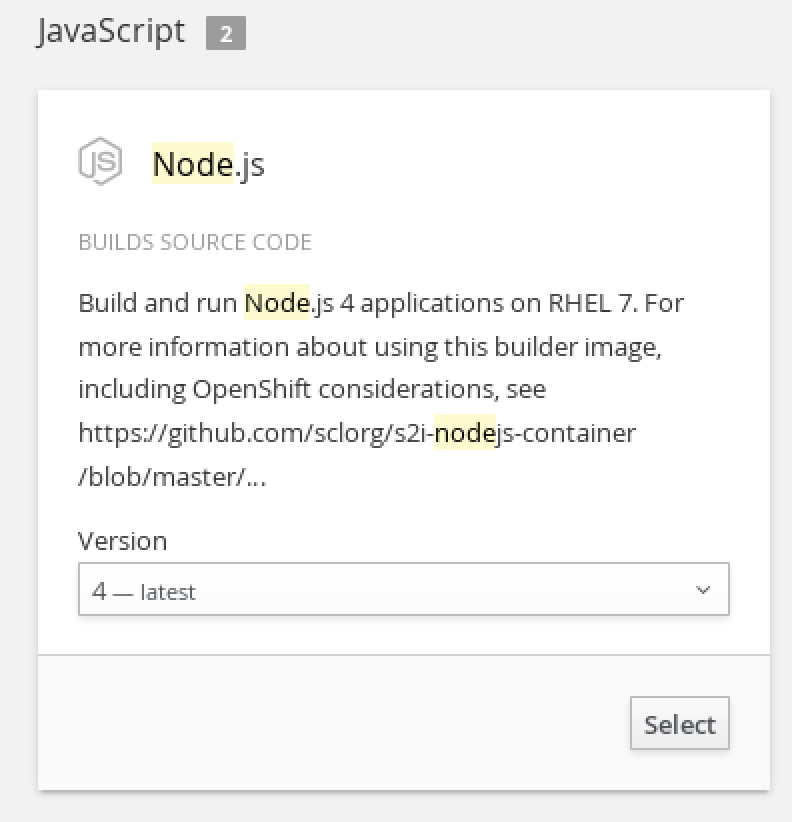
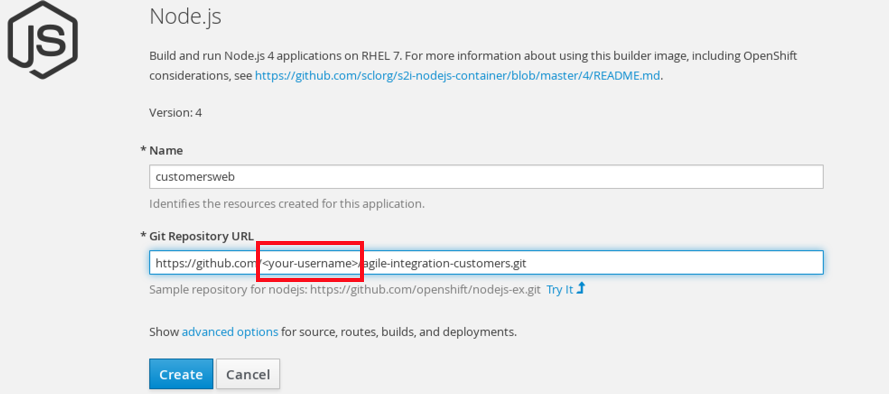
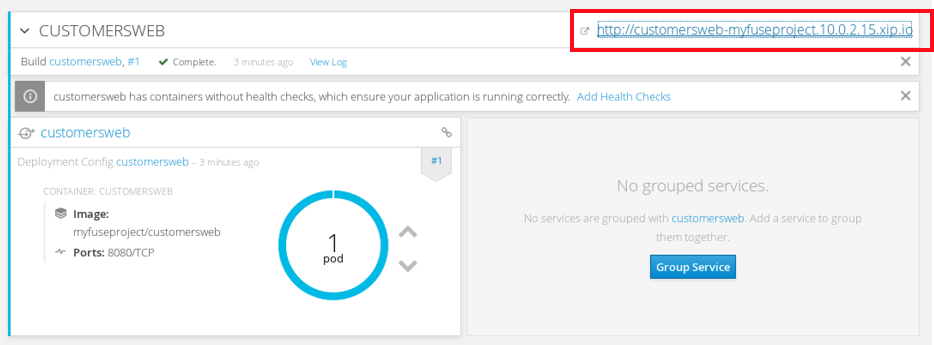
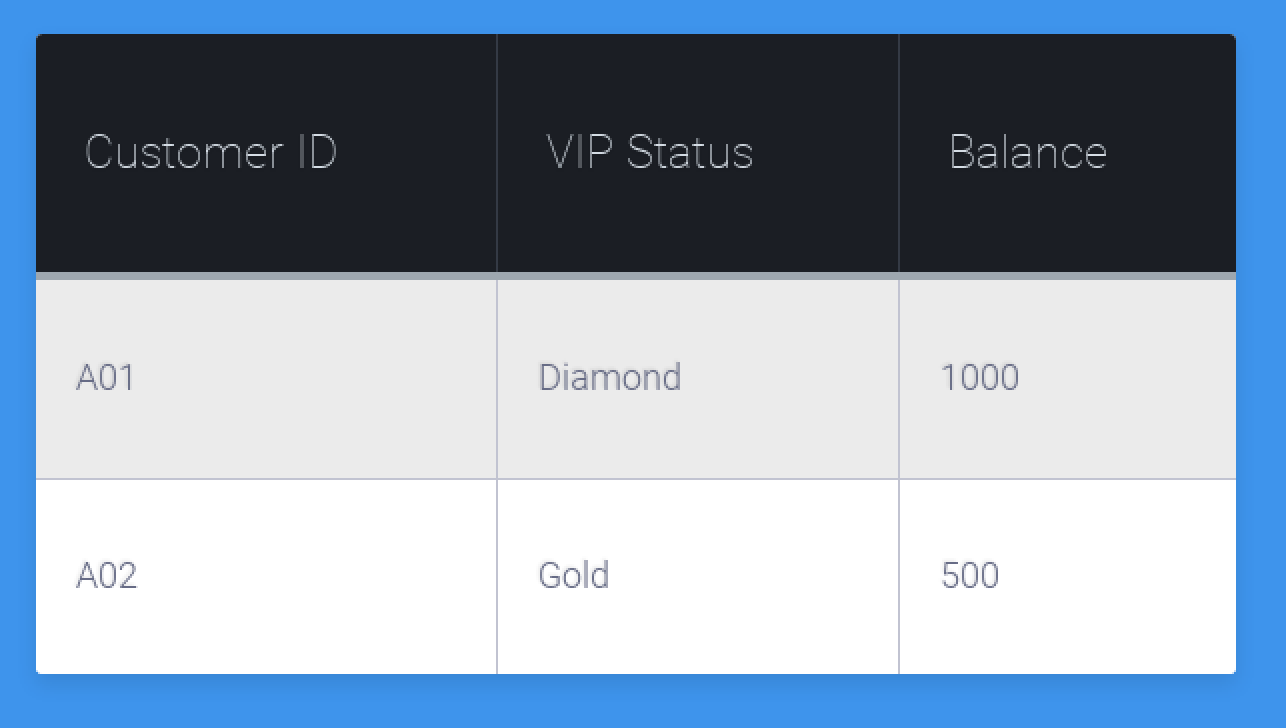
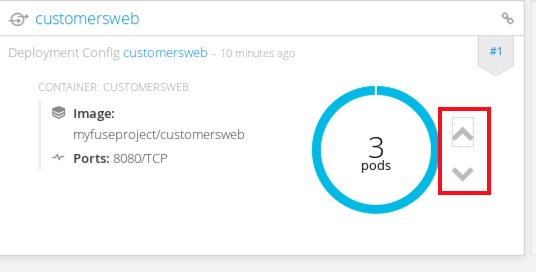
1. Save and rebuild this code by using the Run As configurations again. Once this is running, switch over to the OpenShift console and monitor the progress of the build in the Overview window.   
   Once the build is complete, OpenShift will bring up a new pod running your new code, once this is running it will bring down the old pod - this is a rolling deployment and is the default deployment method in OpenShift.
2. Return again the Camel Route window -> Route Diagram.
3. In another browser window refresh your application   
   <http://camel-ose-springboot-xml-myfuseproject.10.0.2.15.xip.io/myfuselab/customer/all>
4. Watch the Route Diagram and see how the inflight number increases as the delay is happening.  
   
5. You must now undo these changes by removing the added detail in the camel route and rerun the configurations to rebuild and redeploy the route.
6. End of the optional exercise.

# Lab 4 - Deploy a responsive web app using Node.js

In this lab, you will look at the "Source to Image" functionality and deploy a Node.js application that displays the data from the microservice you deployed earlier.

Specifically you will do the following

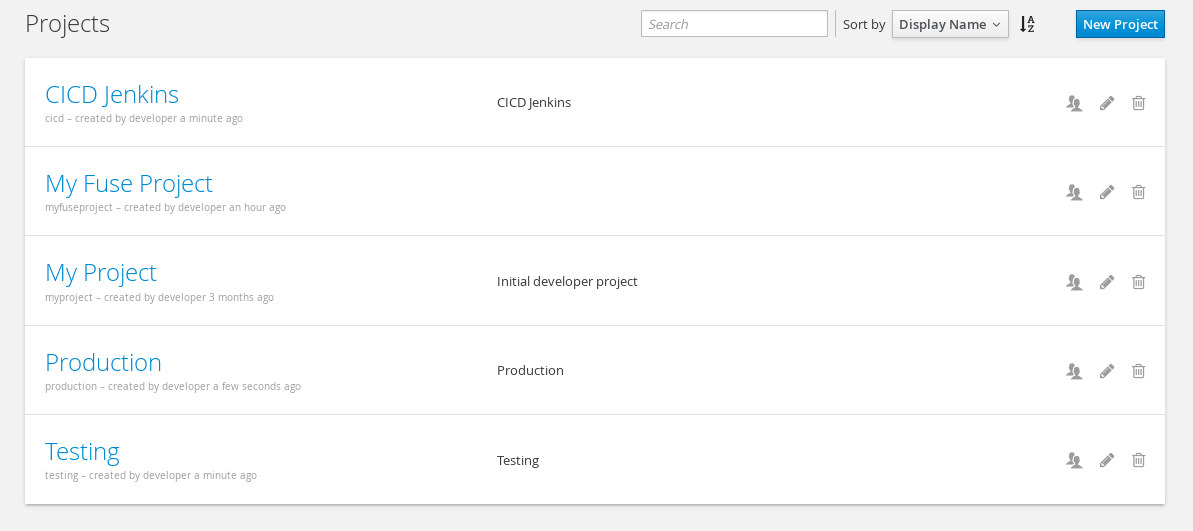
* Deploy a Node.js application on OpenShift using "Source to Image"
* Scaling up and down the number of pods for the Node.js application.

1. In the first step, you need to fork the Customers Node.js application into your GitHub account. Go to <https://github.com/> and log in with your username and password.
2. Go to <https://github.com/dsoffner/agile-integration-customers> and click on the "Fork" button in the upper right corner.  
     
     
     
   The project is forked and will now appear under your username:  
     
    **https://github.com/<your-username>/agile-integration-customers**  
   Use that link from now on to work on your local copy of the code and to deploy it onto OpenShift.
3. Ensure that OpenShift is running,
4. Before you can deploy the code on OpenShift, you need to make a minor change to reflect your environment.
5. In OpenShift note the IP address of the *mysql* service in your **myfuseproject** by going to **Applications -> Services**.  
     
   The IP address is listed under Cluster IP for the *mysql* service, for example "172.30.114.117". Copy this address.
6. In the **agile-integration-customers** project in your GitHub account, click on the link to the **app.js** file  
     
   
7. Click on the "Edit" symbol in the upper right corner to edit the file.  
     
   
8. Scroll down to line 44 of the source code and replace the IP address in the "host"entry with the IP address you noted down in step 4.  
     
   
9. Scroll to the end of the page and add a commit comment into the field. Then click on the green button "Commit changes".  
     
   
10. Your forked version of the Customers web application is now ready to be deployed on OpenShift using the "Source to Image" command.
11. In GitHub click on the "Code" tab to go back to the overview page.  
      
    Click on the green "Clone or download" button on the right side and then on the "Copy to clipboard" symbol.  
      
    
12. In OpenShift, go to the Overview page of your **myfuseproject** and click on "Add to project".  
      
    
13. You are now in OpenShift's image registry where you can pick the image to be used in deploying the new application.  
      
    In the free text field type in "node" and inspect the results.  
      
    
14. Pick the "Node.js" image by clicking on "Select" while leaving the preselected version "4 - latest" selected.  
      
    
15. In the next screen name your application "**customersweb**" and copy the Git Repository URL that you extracted in step 10 and that is pointing to your repository (that includes your GitHub username).  
      
    
16. Click on "Create" and then on "Continue to overview."  
      
    OpenShift is now invoking the "Source to Image" process to deploy the new application and executes the following steps:  
      
    - Retrieve the Node.js image from the internal registry  
    - Download the Node.js source code from your GitHub repository  
    - Deploy the application in a container on OpenShift  
      
    The process should take 3-5 minutes (depending on your Internet connection).
17. In the **myfuseproject** Overview page you should now see a new pod with the web application. Click on the link for the exposed route.  
      
    
18. The result is a Node.js web application that accesses the MySQL database (located in a different container) and displays the same data that we saw earlier using a curl command.  
      
    
19. Now, you can test OpenShift's scaling functionality by going back to the OpenShift administrative console and scaling up the number of pods from 1 to 3 using the arrows.  
      
    
20. Now scale the number of pods down to 1 again.
21. You have successfully completed this lab by deploying a web interface to your application using OpenShift's "Source to Image" functionality.
22. Before you continue with the next lab, create a snapshot of your work in VirtualBox by going to **Machine -> Take Snapshot…** and naming it "Snapshot Lab 4".

# Lab 5 - CI / CD with OpenShift

In this lab you will do the following:

* Deploy CI/CD tooling (Jenkins) on your OpenShift instance
* Create two additional projects that simulate a Test and Production environment
* Create and execute a pipeline to deploy the Customers Web application from the preceding lab across different environments.

1. Ensure that OpenShift is running,
2. In the OpenShift console, create 3 new projects for Jenkins, the Test environment and the Production environment using the "New Project" button.   
     
   Do not add any images to the projects yet - just create the projects. Use the following parameters:
   1. Name: "cicd"  
      Display Name: "CICD Jenkins"  
      Description: "CICD Jenkins"
   2. Name: "testing"  
      Display Name: "Testing"  
      Description: "Testing"
   3. Name: "production"  
      Display Name: "Production"  
      Description: "Production"  
        
        
      
3. Login into OpenShift as the developer on the command line using your Terminal.  
   Use the oc login command. The default login credentials are *username = "developer"* and *password = "developer"*:

oc login <https://10.0.2.15:8443>

Username: developer

Password: developer

You should see **Login successful** in the output.

1. In the command line add in role based access to our projects to allow the different service accounts to build, promote, and tag images.  
     
   First we will allow the cicd project’s Jenkins service account edit access to all of our projects:

oc policy add-role-to-user edit system:serviceaccount:cicd:jenkins -n myfuseproject

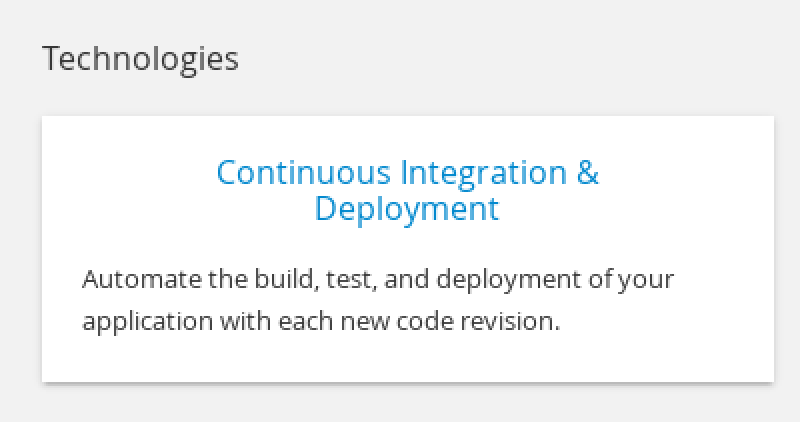
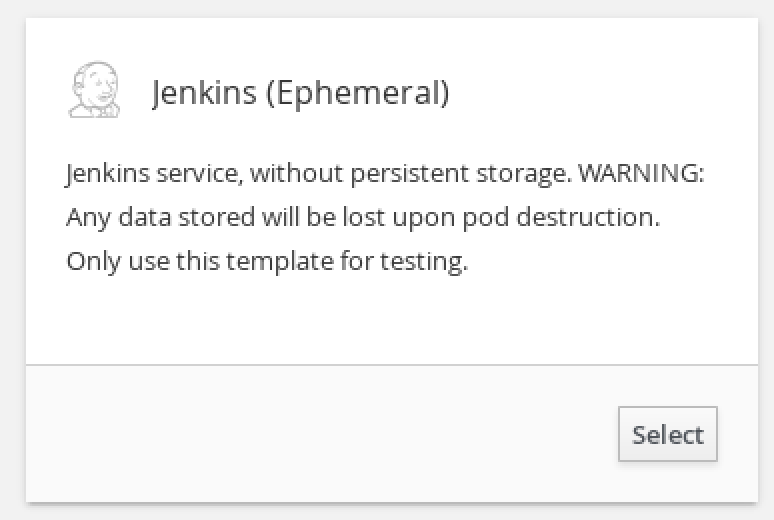
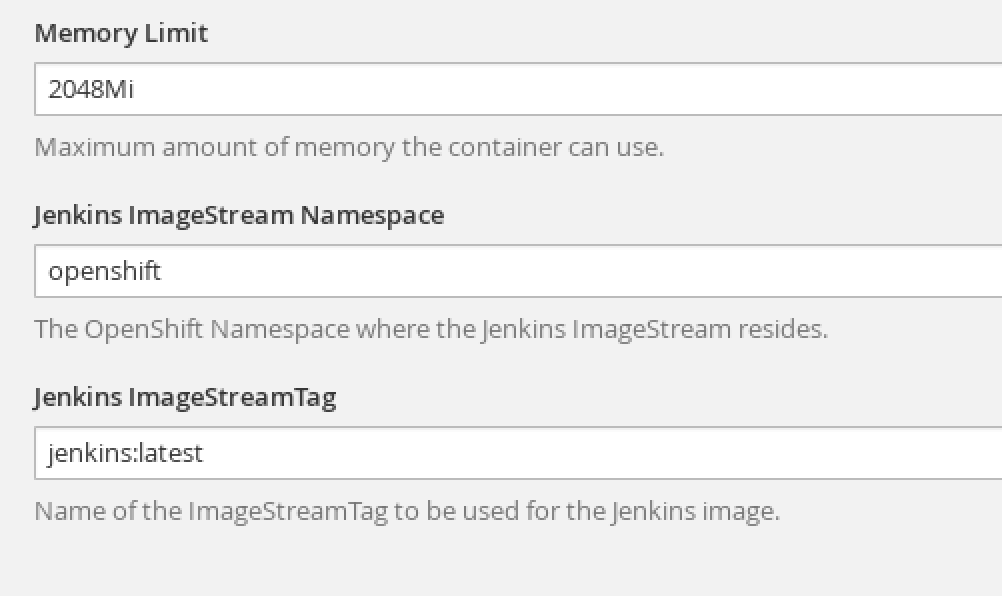
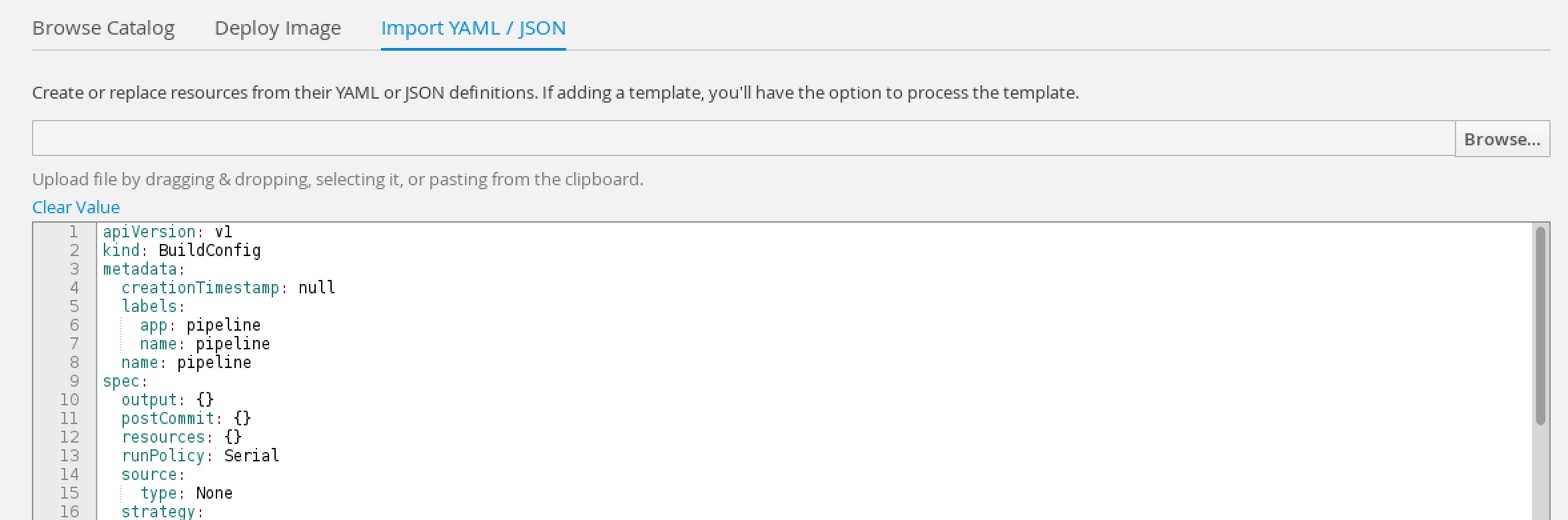
oc policy add-role-to-user edit system:serviceaccount:cicd:jenkins -n testing

oc policy add-role-to-user edit system:serviceaccount:cicd:jenkins -n production

1. Now we want to allow our testing and production service accounts the ability to pull images from the development project:

oc policy add-role-to-group system:image-puller system:serviceaccounts:testing -n myfuseproject

oc policy add-role-to-group system:image-puller system:serviceaccounts:production -n myfuseproject

1. Now it is time to deploy Jenkins, enable OAuth integration and a create a pipeline for our deployment between the different environments.
2. Select the "CICD Jenkins" project and and click on "Add to Project".
3. Scroll down to "Technologies" and click on the section displaying the different options for "Continuous Integration & Deployment".  
     
   
4. Select the "Jenkins (Ephemeral) image.  
     
   
5. In the configuration of the Jenkins template, leave all default values, but increase the value for "Memory Limit" to "2048Mi". Then click on "Create".  
     
   
6. Next we import a pre configured pipeline to deploy our application from Dev to Test to Prod. In the "CICD Jenkins" project click on "Add to Project" again.
7. Click on "Import YAML / JSON".
8. Access the YAML file with the pipeline definition that is located at this link:  
   <https://raw.githubusercontent.com/dsoffner/agile-integration-openshift/master/pipeline.yaml>
9. Copy the content of the YAML file and paste it into the large text field. Then click on "Create".  
     
   
10. You have configured Jenkins to run inside your OpenShift instance to deploy the Customers Web application between different instances.
11. The next step is to create different OpenShift artefacts for our Testing and Production environments, among them deployment configurations, services and routes. You will be using the command line for this again.  
      
    We are using two separate (arbitrary) image tags: **promoteQA** for testing promotion and **promotePRD** for production promotion.
12. Let's start with the Testing environment. In a Command Line tool issue the following commands sequentially:

oc project testing

oc create dc customersweb --image=172.30.1.1:5000/myfuseproject/customersweb:promoteQA

oc rollout cancel dc/customersweb

oc patch dc/customersweb -p '{"spec":{"template":{"spec":{"containers":[{"name":"default-container","imagePullPolicy":"Always"}]}}}}'

oc rollout cancel dc/customersweb

oc expose dc customersweb --port=8080

oc expose service customersweb --name=customersweb --hostname=customersweb-myfuseproject-testing.10.0.2.15.xip.io

1. Next do the same with Production:

oc project production

oc create dc customersweb --image=172.30.1.1:5000/myfuseproject/customersweb:promotePRD

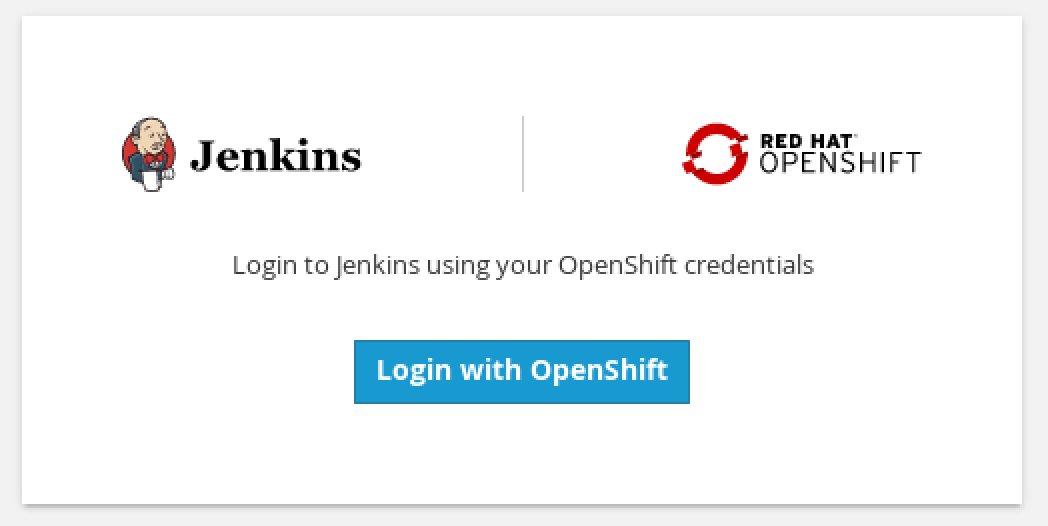
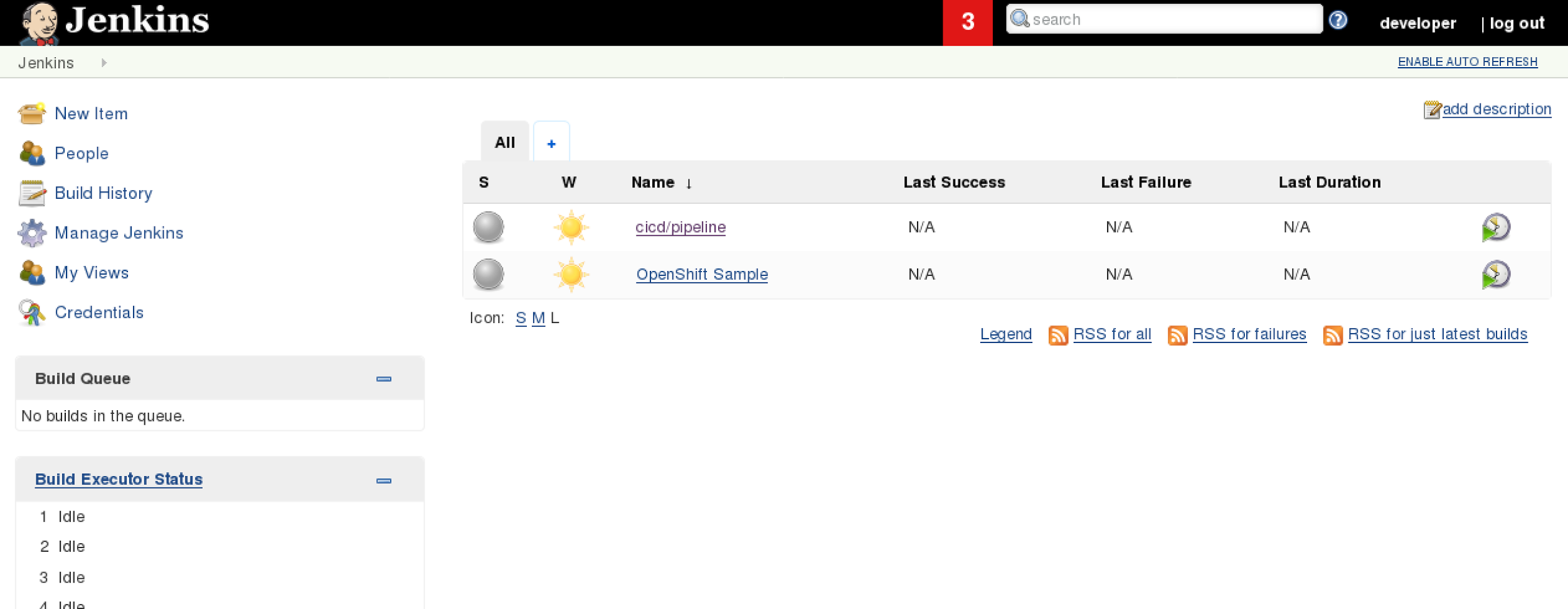
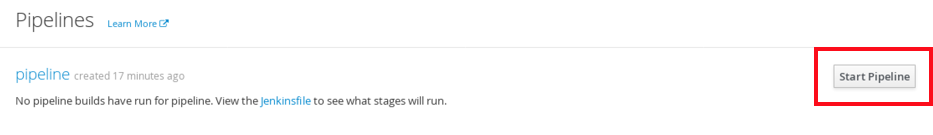
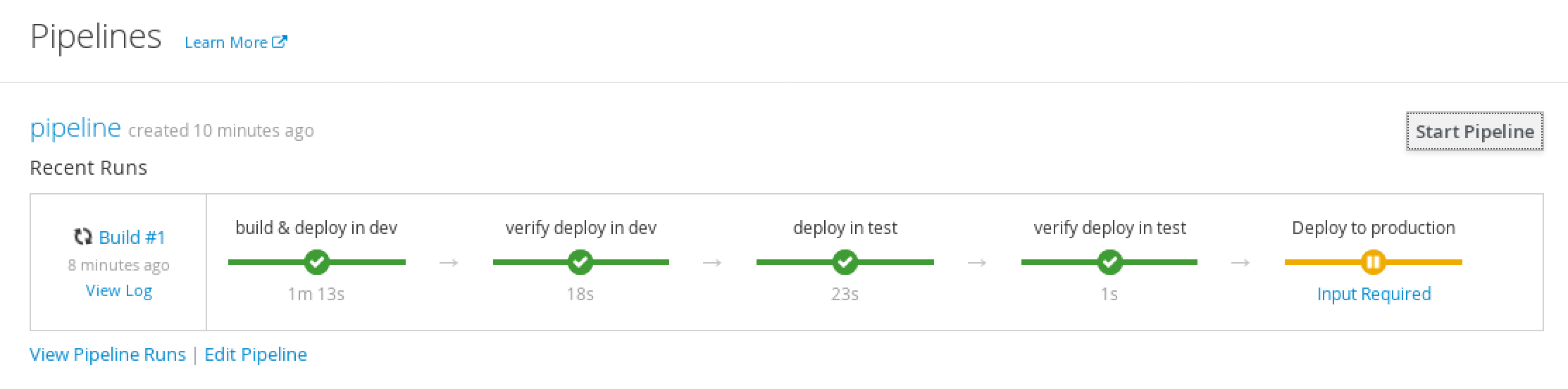
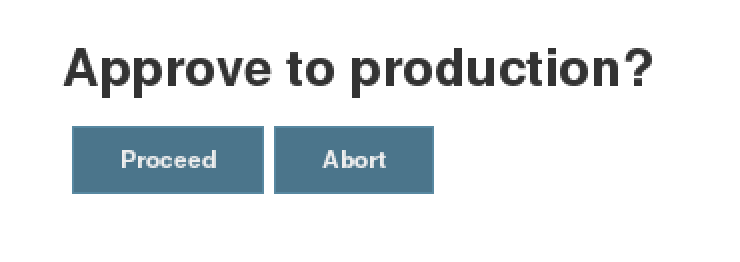
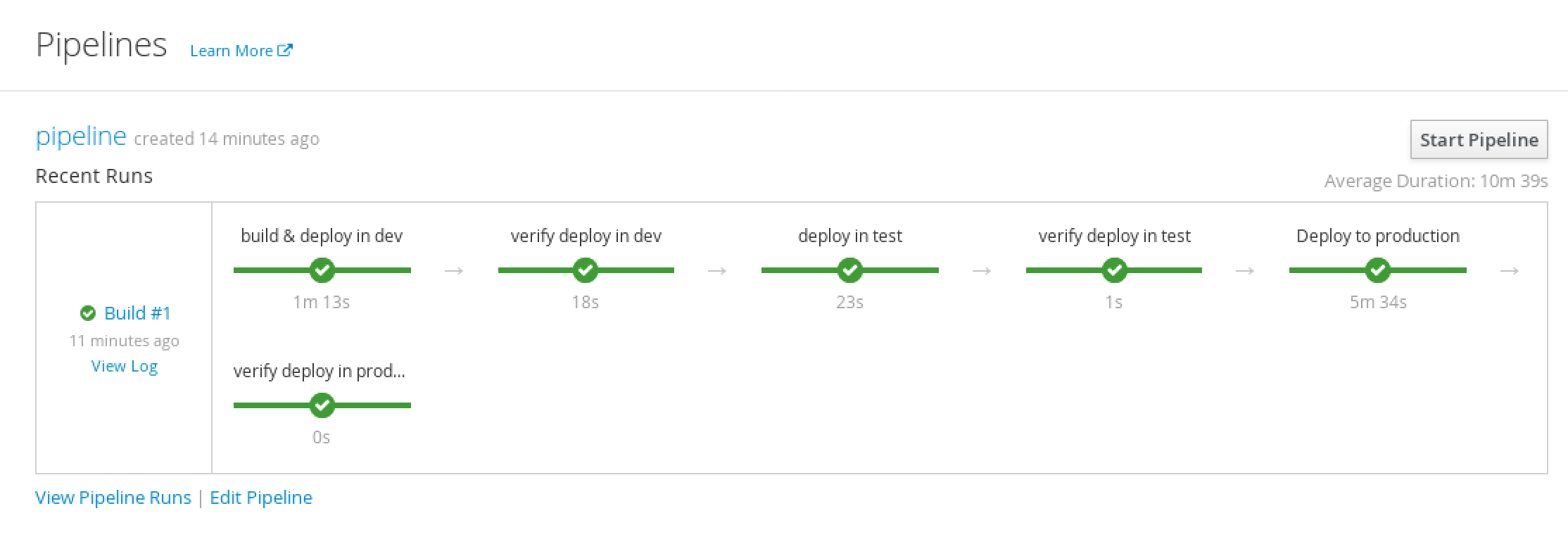
oc rollout cancel dc/customersweb

oc patch dc/customersweb -p '{"spec":{"template":{"spec":{"containers":[{"name":"default-container","imagePullPolicy":"Always"}]}}}}'

oc rollout cancel dc/customersweb

oc expose dc customersweb --port=8080

oc expose service customersweb --name=customersweb --hostname=customersweb-myfuseproject-production.10.0.2.15.xip.io

1. Now we are ready to run our pipeline deployment from the "CICD Jenkins" project. Before we kick it off, make sure the Jenkins Ephemeral pod is running and open the Jenkins instance in a separate Browser tab through the following URL (or by selecting the route in the CICD project):  
     
   <https://jenkins-cicd.10.0.2.15.xip.io>  
     
   Add a Security Exception your browser if required.  
     
   Since you configured OAuth earlier you can login with your OpenShift credentials. In the Login screen, click on "Login with OpenShift" and use your login credentials (**developer / developer**).  
     
   
2. Authorize access by having both **user:info** and **user:check-access** selected and click on "Allow selected permissions".
3. You are now seeing the Jenkins Dashboard running on OpenShift.  
     
   
4. Switch back to the OpenShift UI, select the "CICD Jenkins" project and select **Builds -> Pipelines**.  
     
   Then click on "Start Pipeline".  
     
   
5. Over the next few minutes you can see the pipeline build progressing in the Pipeline UI or alternatively in the Jenkins UI.  
     
   
6. You can also see that the pipeline is paused waiting for user input to deploy to the Production environment.  
     
   Select "Input Required" and you will be taken to the running Jenkins (you may have to log in if you haven’t already).  
     
   Select Proceed to allow the pipeline to continue to deploy to production.  
     
   
7. Once approved, the application is deployed and verified in production.  
   You can inspect the pipeline by switching back to the OpenShift console.  
     
   
8. You can test the application in both the Testing and Production environments by using the exposed routes:  
     
   For Testing: <http://customersweb-myfuseproject-testing.10.0.2.15.xip.io>  
     
   For Production: <http://customersweb-myfuseproject-production.10.0.2.15.xip.io>
9. You have successfully completed this lab by deploying Jenkins in your OpenShift environment and promoting your Customers web application through three different environments.
10. Before you continue with the next lab, create a snapshot of your work in VirtualBox by going to **Machine -> Take Snapshot…** and naming it "Snapshot Lab 5".

# 

# Lab 6 - Integration with API Management

Now it is time to expose your RESTful API in Red Hat's enterprise grade API Management solution 3scale.

In this lab you will do the following

* Deploy the 3Scale Gateway on OpenShift
* Configure the 3Scale Manager service to connect to OpenShift 3Scale Gateway
* Access your 3scale Admin Portal and set up your first plans and metrics and your first API keys.
* Configure API access policy and application plans.
* Integrate your API with 3scale using the API gateway

1. Ensure that OpenShift is running,
2. Login into OpenShift as the developer on the command line using your Terminal.  
   Use the oc login command. The default login credentials are *username = "developer"* and *password = "developer"*:

oc login <https://10.0.2.15:8443>

Username: developer

Password: developer

You should see **Login successful** in the output.

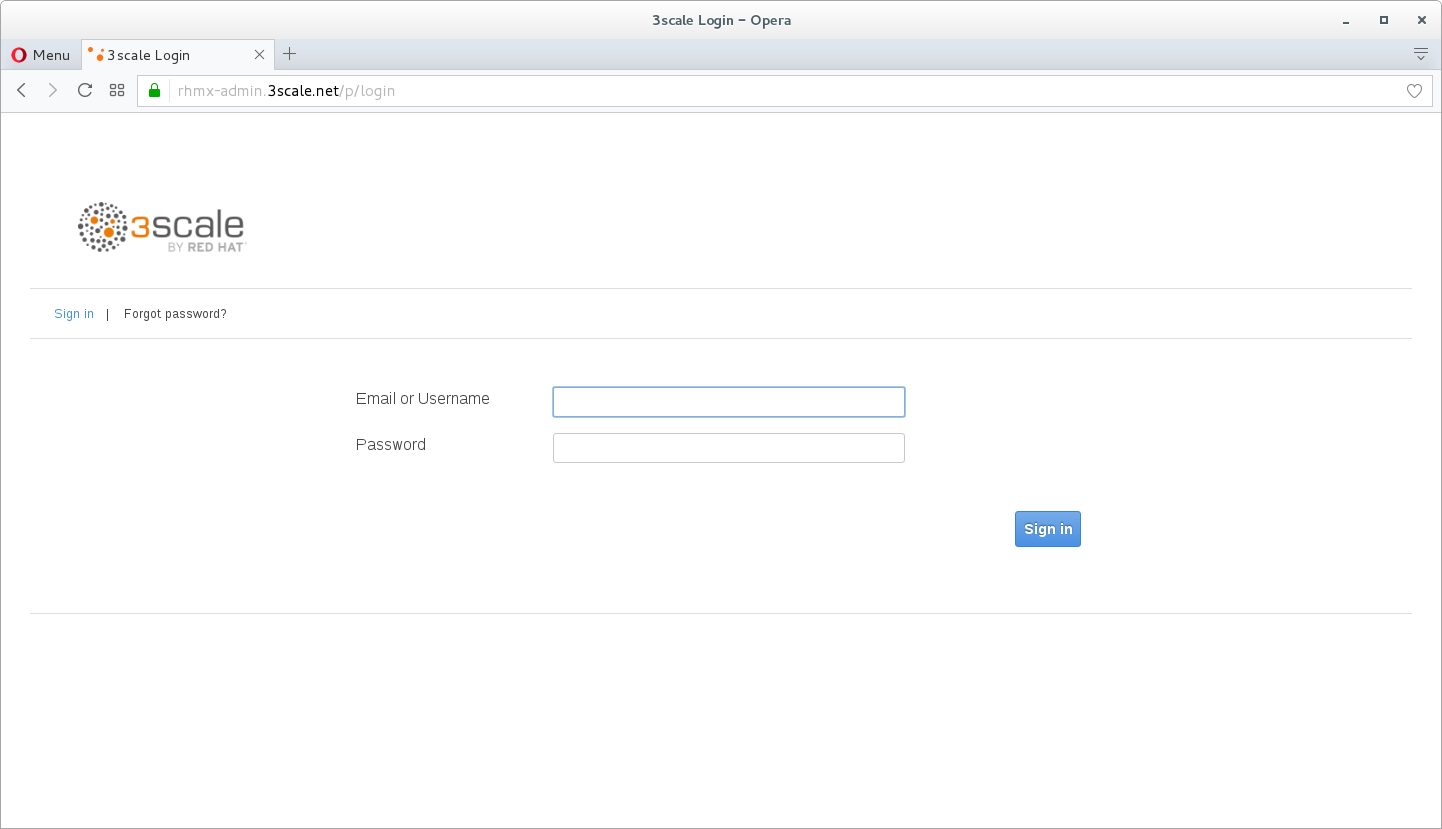
1. Create your project. This example sets the display name as **gateway**

oc new-project "3scalegateway" --display-name="gateway" --description="3scale Gateway"

The response should look like this:

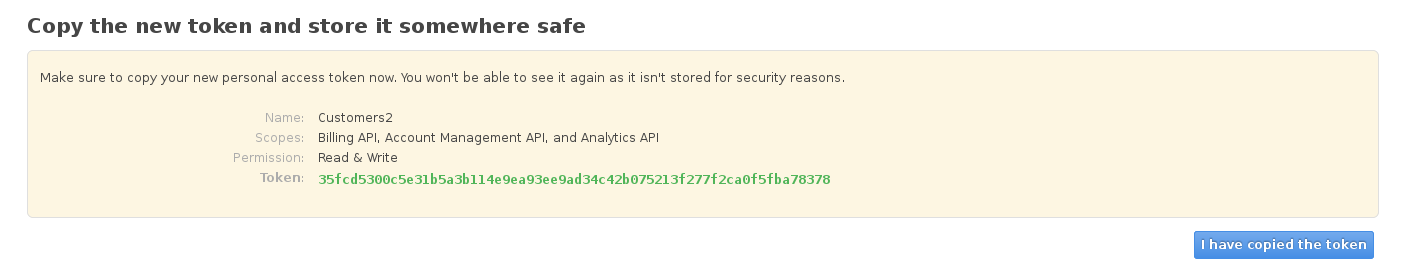
Now using project "3scalegateway" on server "https://10.0.2.15:8443".

Ignore the suggested next steps in the text output at the command prompt and proceed to the next step below.

1. Open a browser window and login into your 3scale Admin Portal at <DOMAIN>-admin.3scale.net where <DOMAIN> is your specific domain provided when you set up the 3scale online service. 
2. Click on the gear symbol in the upper right corner. Then click on **Personal Settings** and **Tokens**.

Under **Access Tokens** click on **Add Access Token**. Provide it with a name, click all three scopes and select "**Read & Write**" permission.

Click on **Create Access Token** and note down or copy the generated token. This is very important, as you will need it in the next step.



1. In OpenShift, create a new secret to reference your project by replacing and with yours. Use the following command on your command line

oc secret new-basicauth apicast-configuration-url-secret --password=https://<ACCESS\_TOKEN>@<DOMAIN>-admin.3scale.net

Here <ACCESS\_TOKEN> is the Access Token for the 3scale Account Management API generated in the previous step, and <DOMAIN>-admin.3scale.net is the URL of your 3scale Admin Portal.

The response should look like this:

secret/apicast-configuration-url-secret

1. Create an application for your APIcast Gateway from the template, and start the deployment:

oc new-app -f https://raw.githubusercontent.com/3scale/3scale-amp-openshift-templates/2.0.0.GA-redhat-2/apicast-gateway/apicast.yml

You should see the following messages at the bottom of the output:

--> Creating resources ...

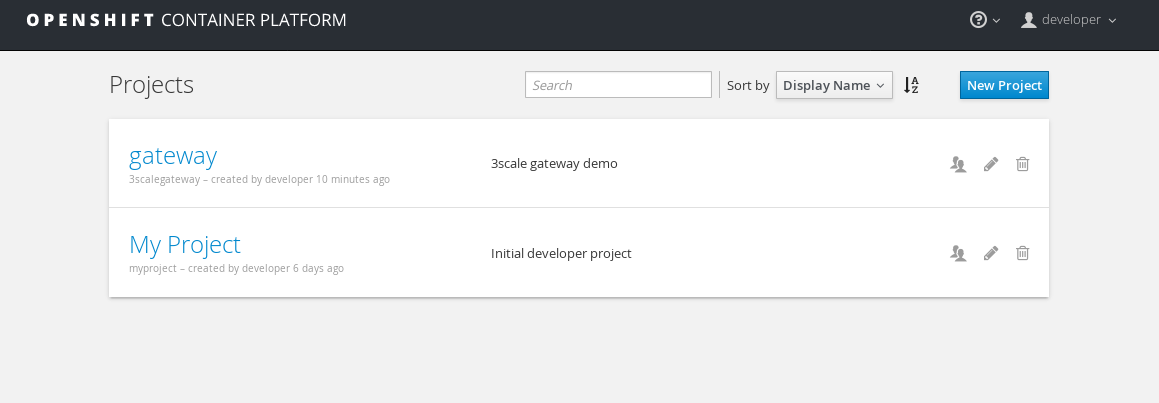
deploymentconfig "apicast" created

service "apicast" created

--> Success

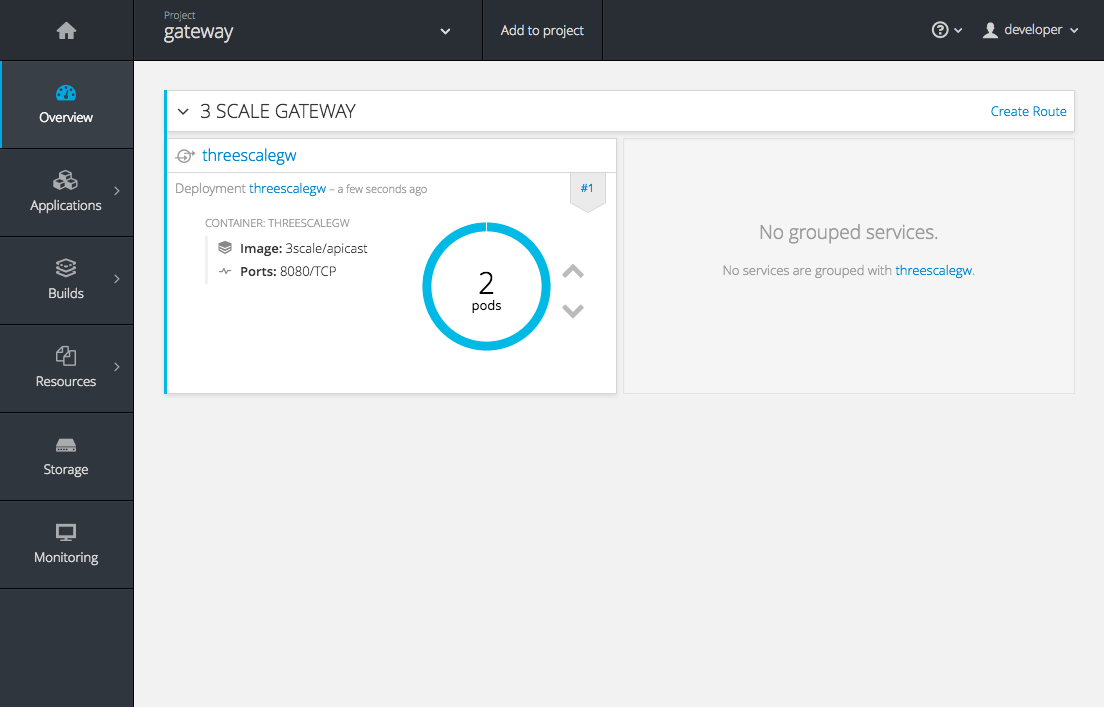
Run 'oc status' to view your app.

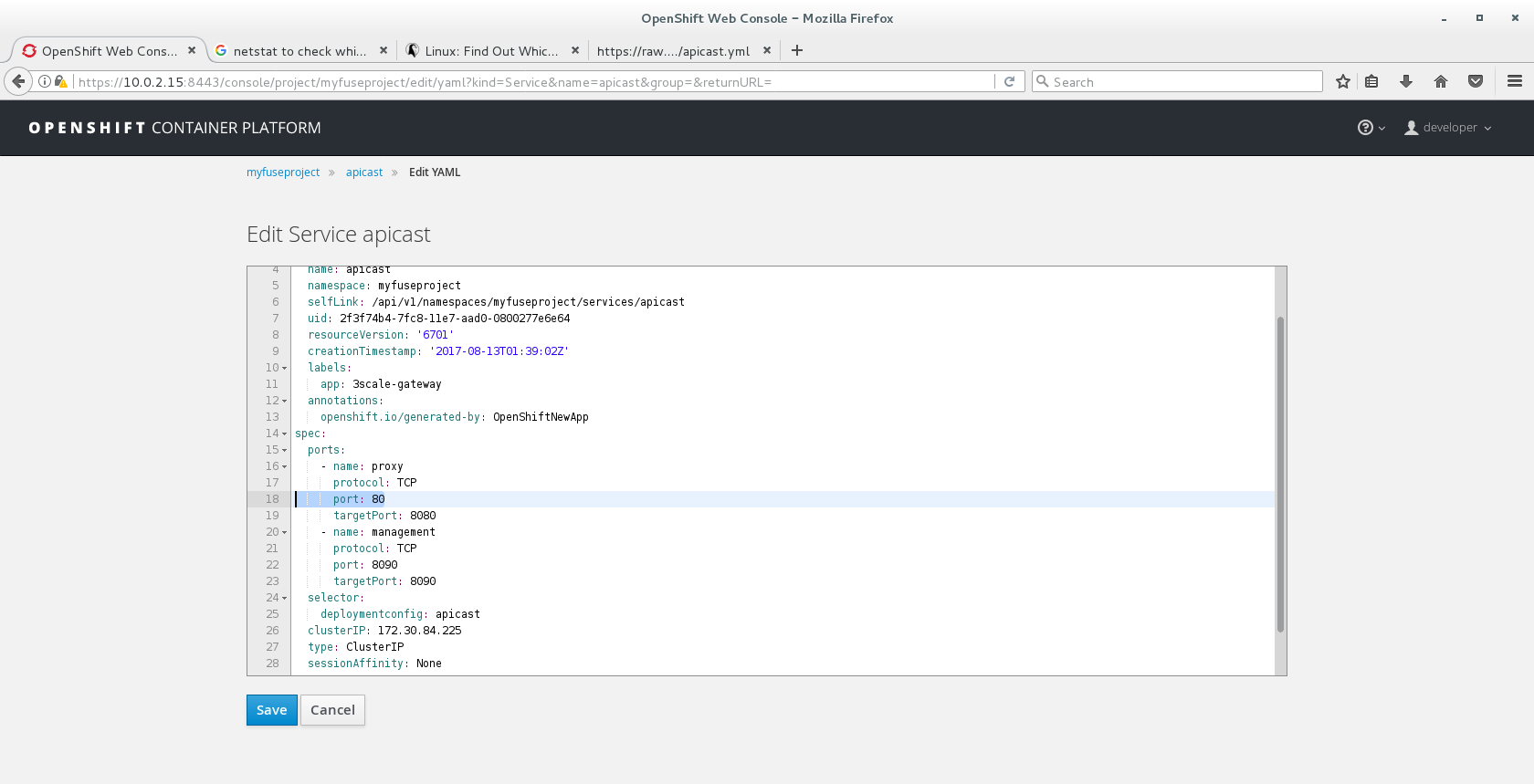
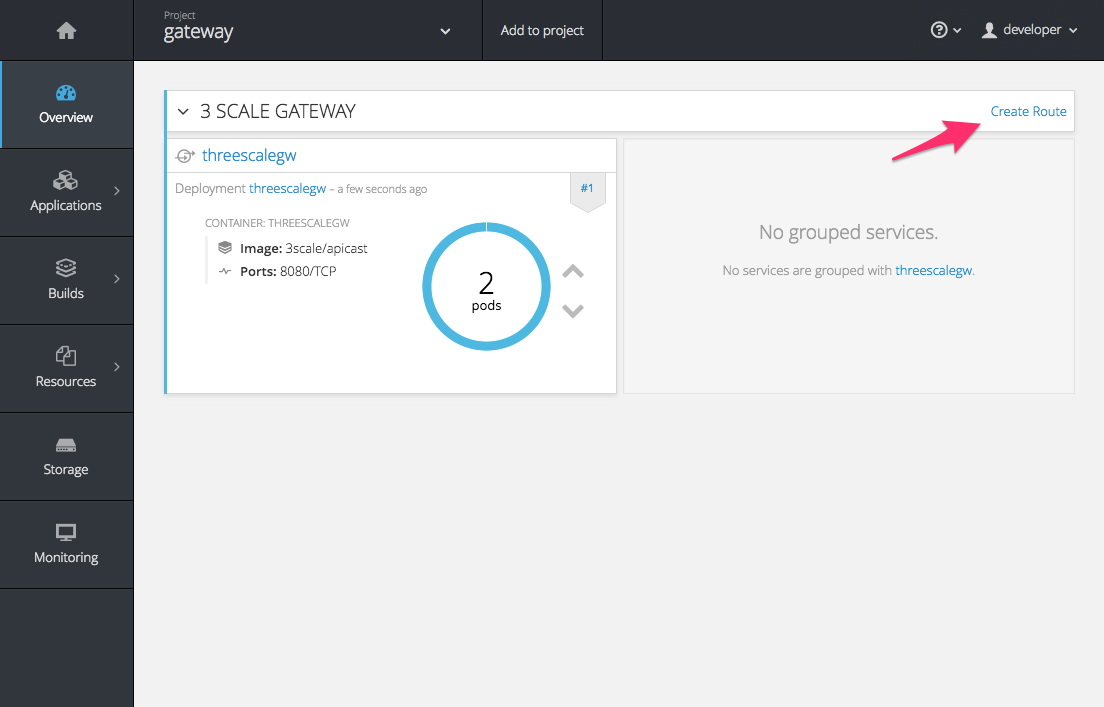
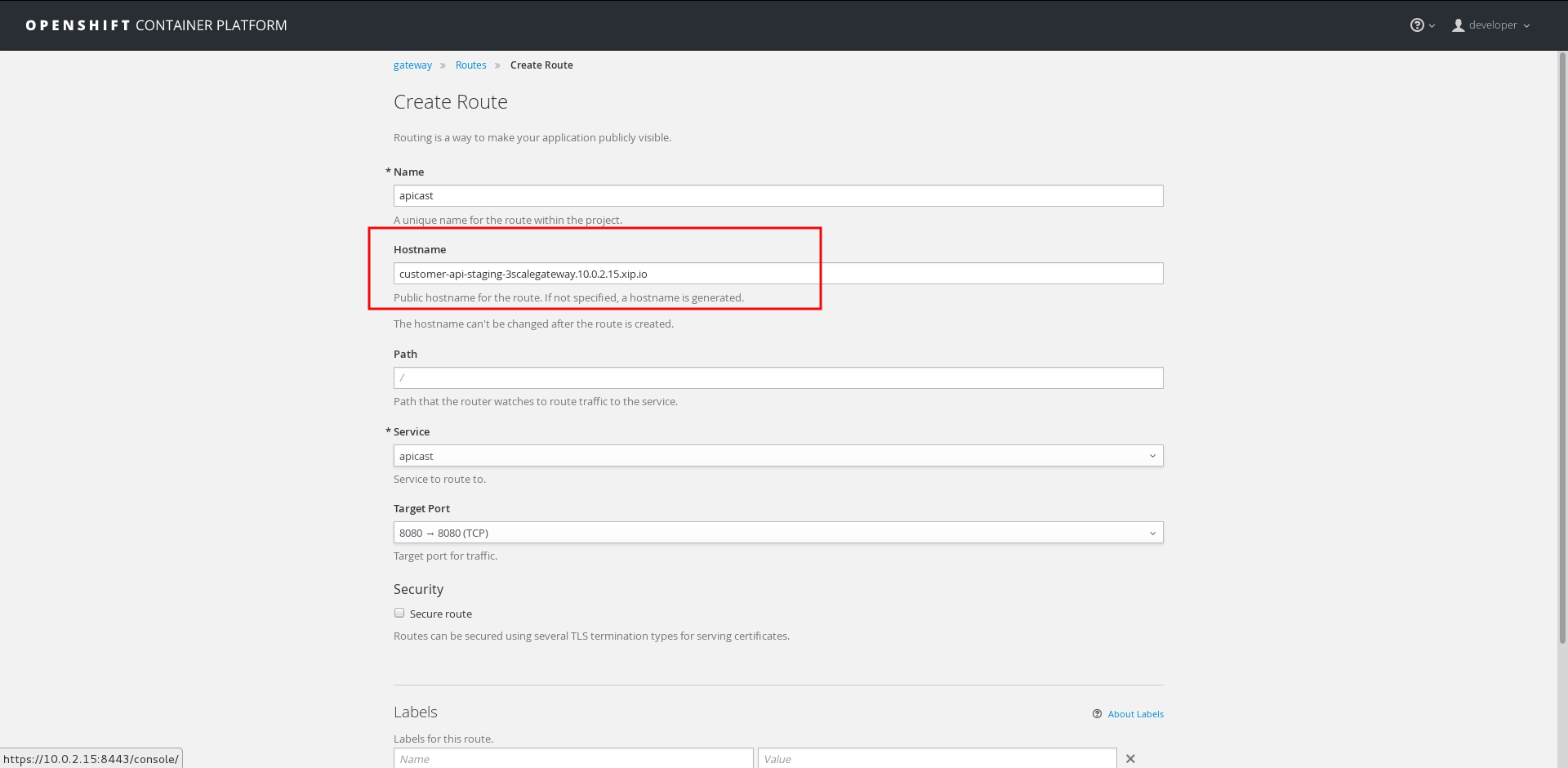
1. If not already opened, open the web console for your OpenShift cluster in your browser: **https://10.0.2.15:8443/console/**
2. Log in using the developer credentials in the section above.

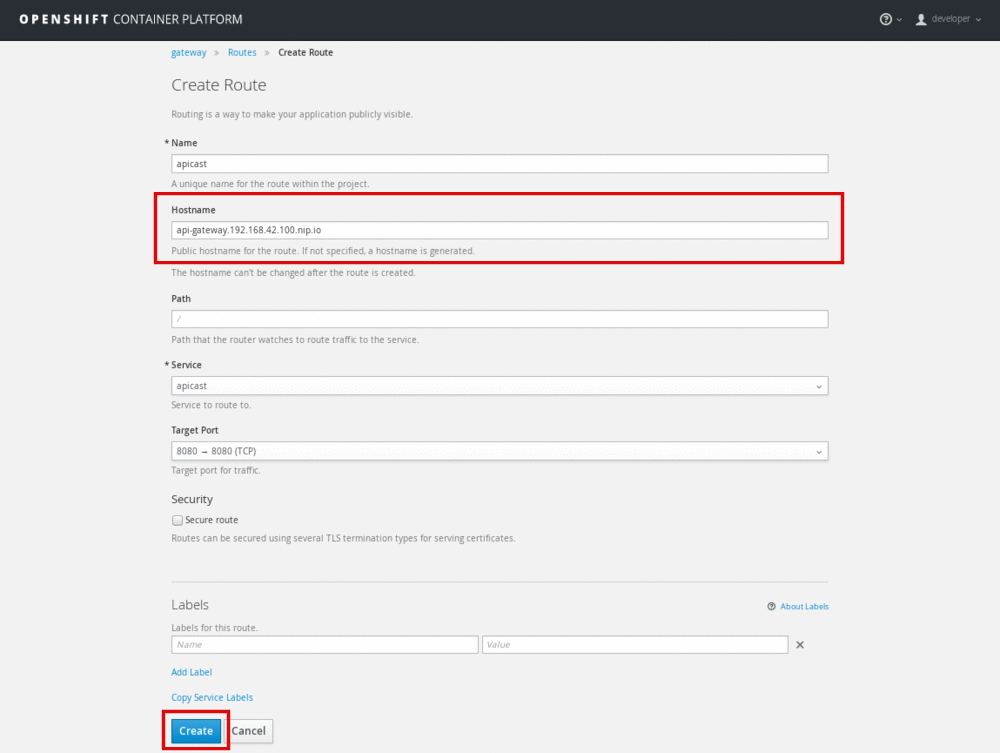
You will see a list of projects, including the **gateway** project you created from the command line above.

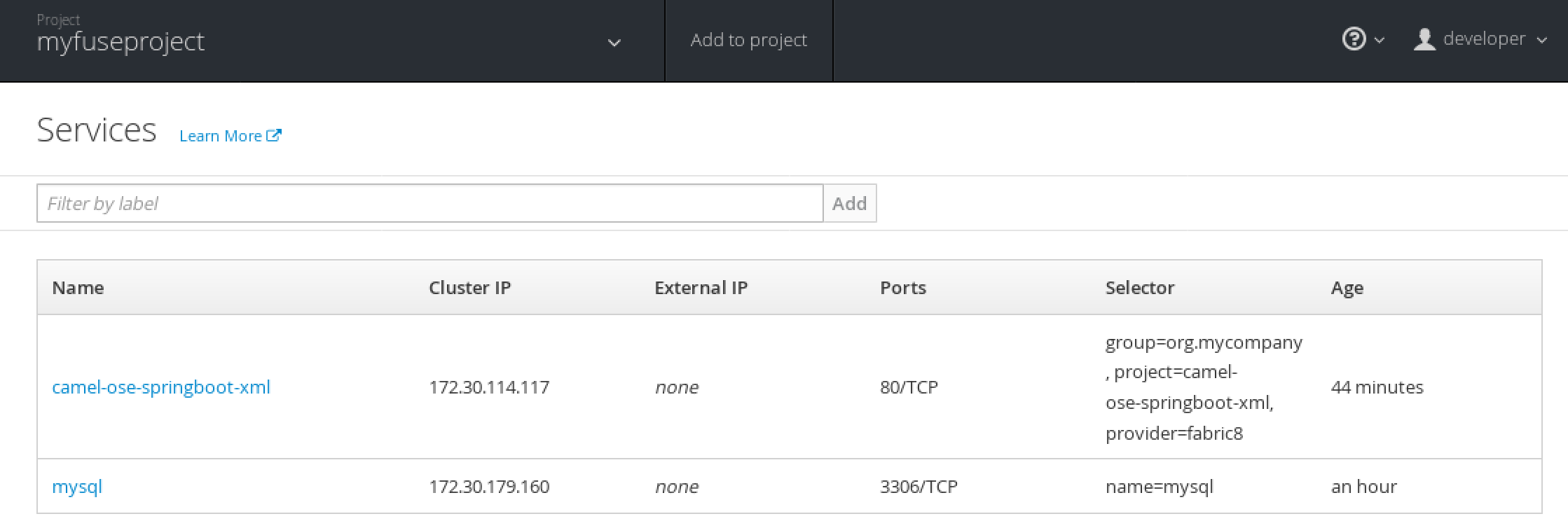
1. Select the **gateway** project by clicking on it, you will see the **Overview** screen.

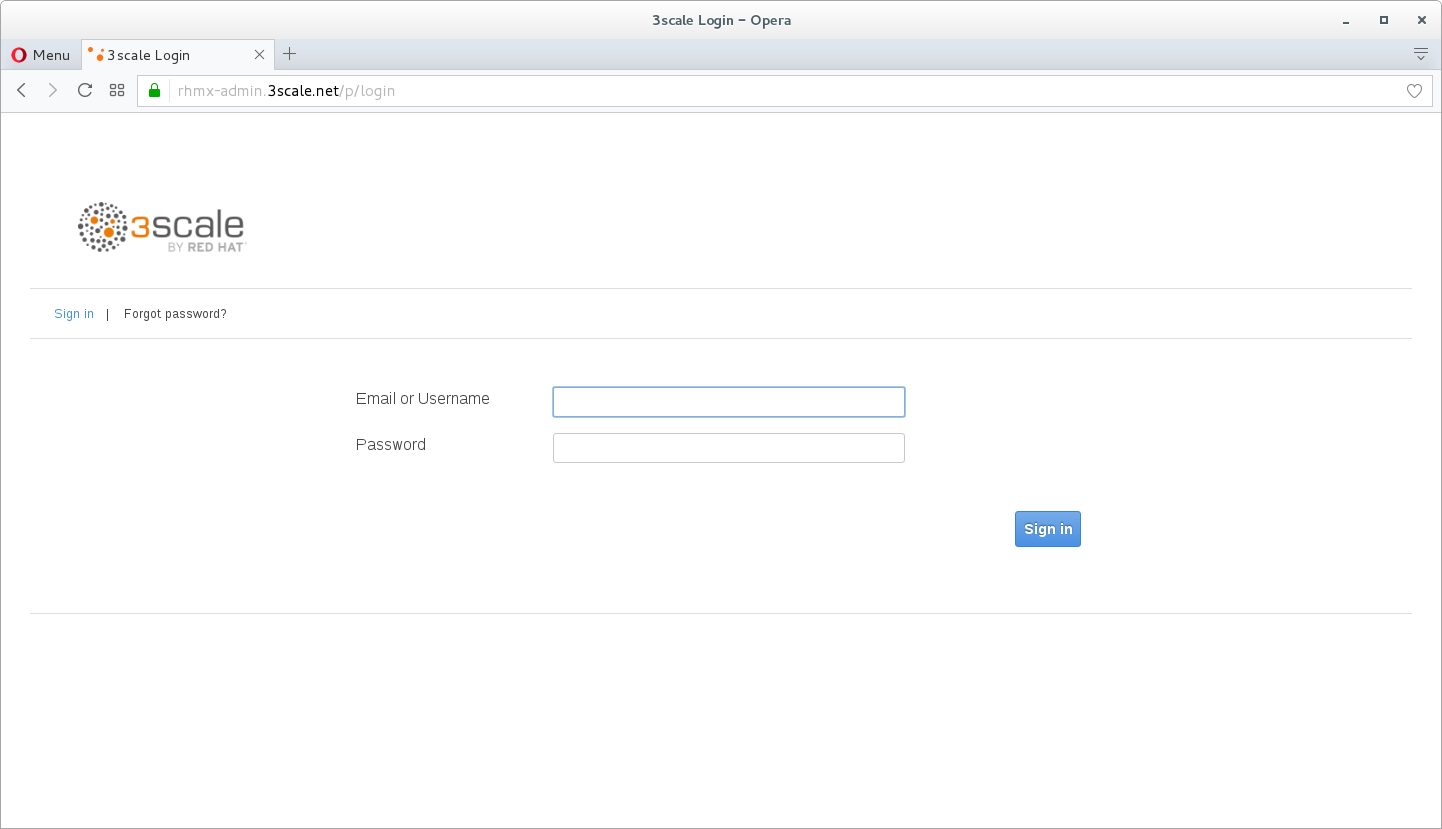
Each APIcast instance, upon starting, downloads the required configuration from 3scale using the settings you provided on the **Integration** page of your 3scale Admin Portal.



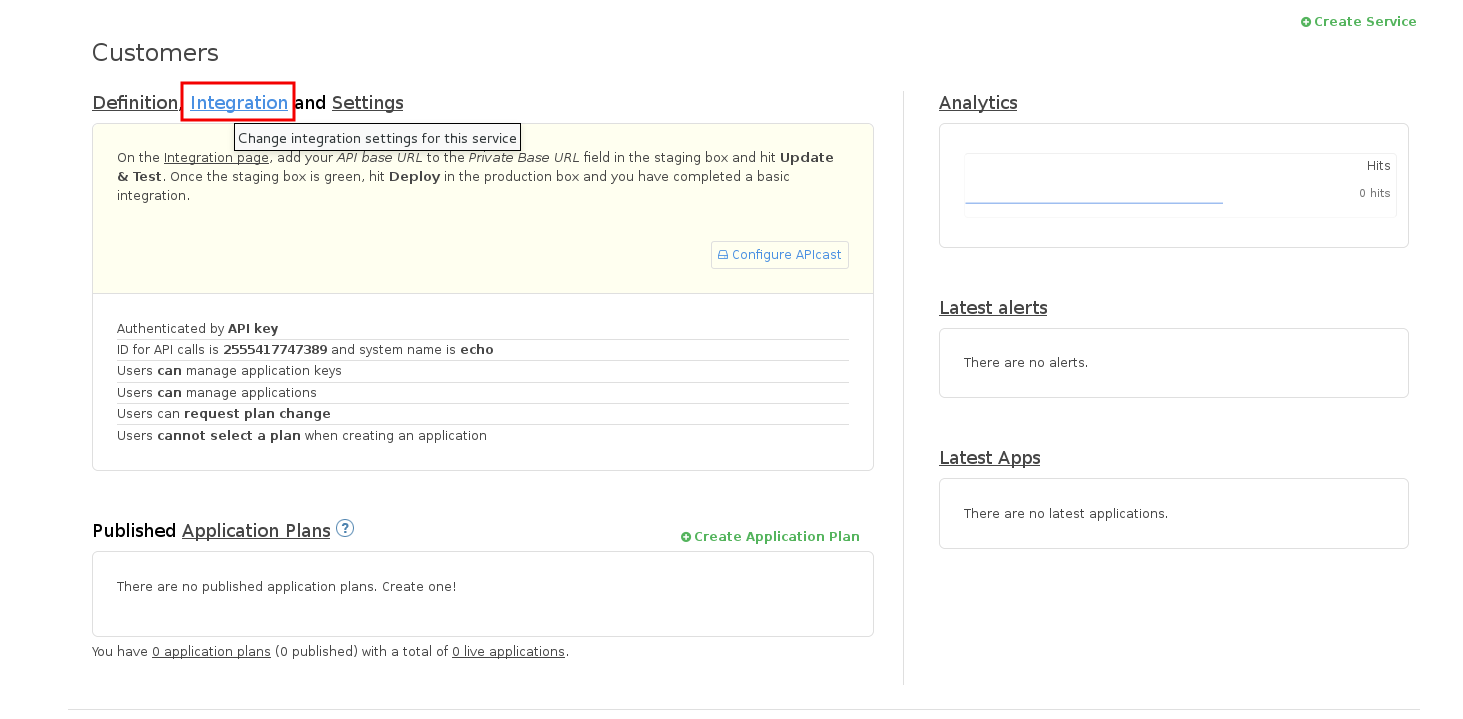
1. In order to allow your APIcast instances to receive traffic, you'll need to create a route.
2. Before creating a route for your APIcast gateway, modify the existing service to map port 80 to port 8080 and redeploy the gateway.   
     
   In OpenShift click on **“Applications -> Services”** and select the **apicast** service. In the right hand corner click on “**Actions -> Edit YAML”**.   
     
   In the YAML file modify the value of the port to “80” as shown in the screenshot below and click on “Save”.  
     
   spec:  
    ports:  
    - name: proxy  
    protocol: TCP  
    port: 80  
    targetPort: 8080  
    - name: management  
    protocol: TCP  
    port: 8090  
    targetPort: 8090  
     
     
   Click on **"Applications -> Deployments"** and select the **apicast** deployment configuration. In the right hand corner click on “Deploy”.   
     
   The APIcast gateway is deployed with the new configuration.   
   
3. Return to the Overview tab and click on **Create Route**.
4. Enter the Hostname in the section **Hostname** (without the http:// and without the port):   
     
   customer-api-staging-3scalegateway.10.0.2.15.xip.io  
     
   then click the **Create** button.  
     
   

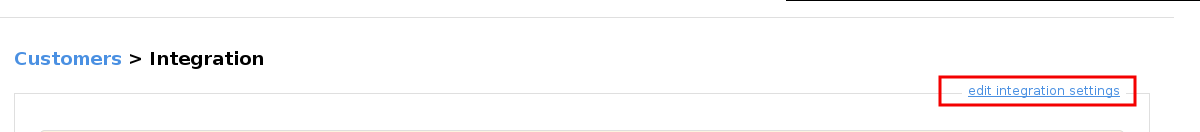


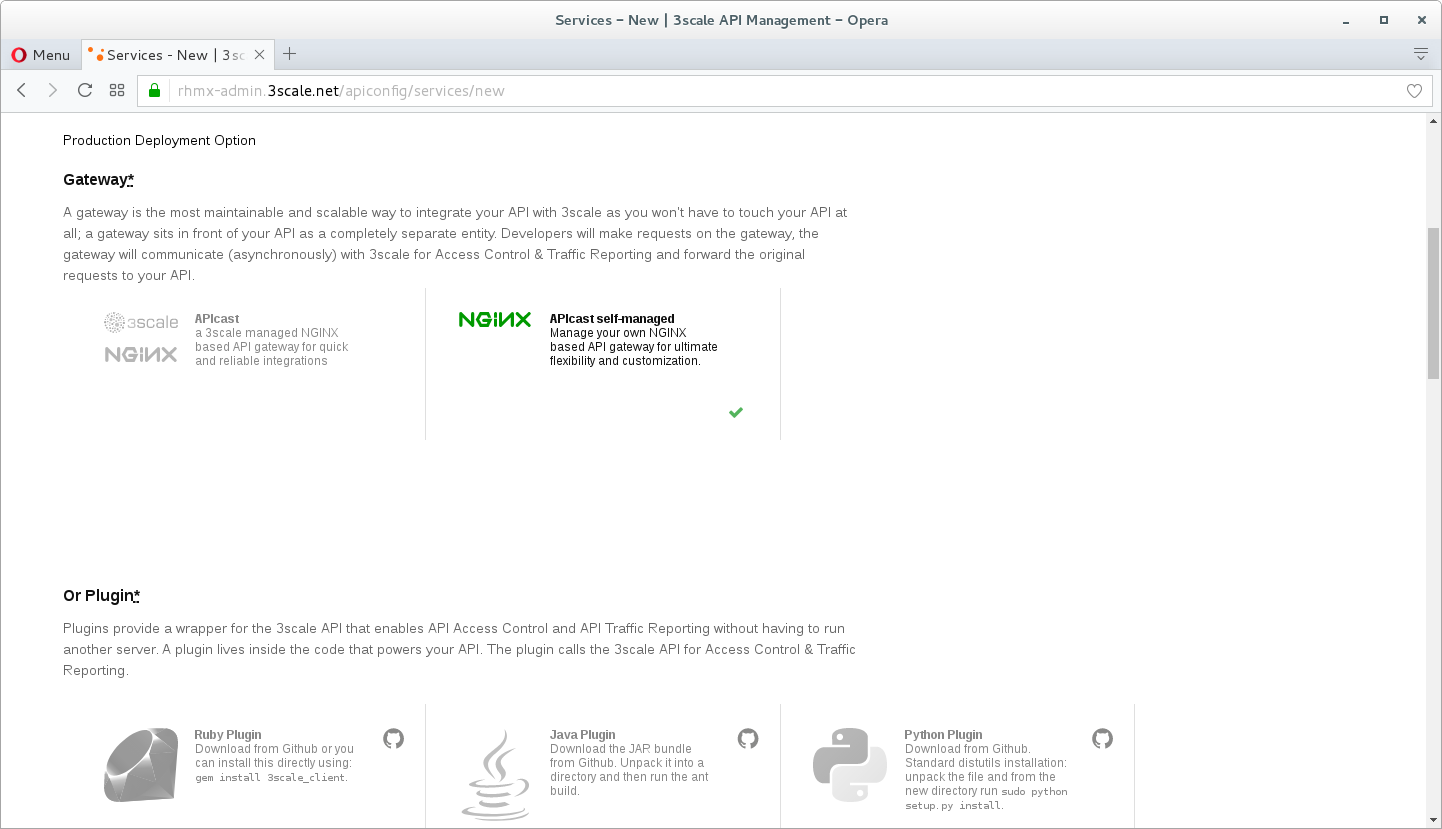
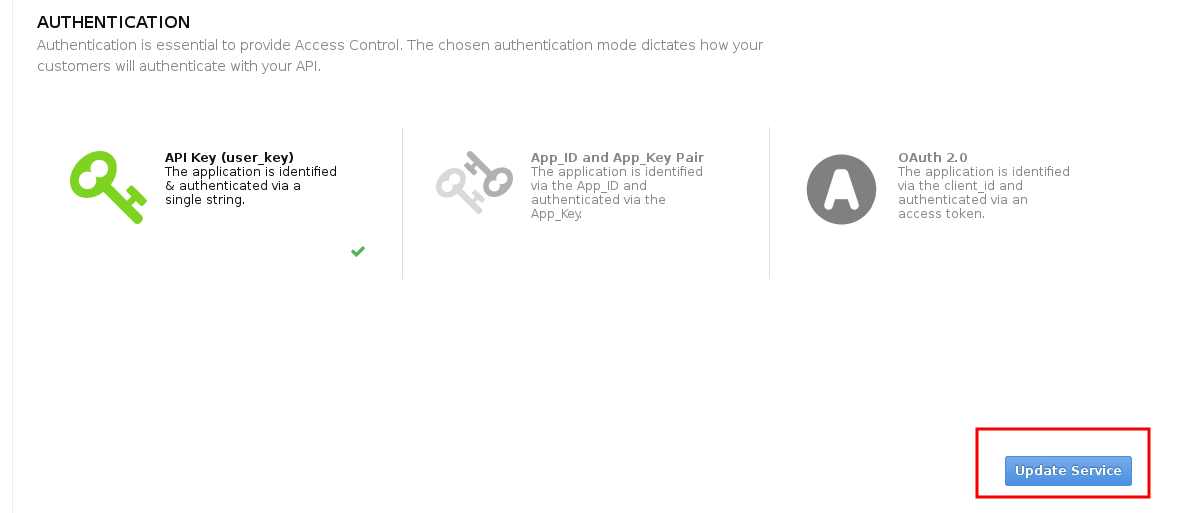
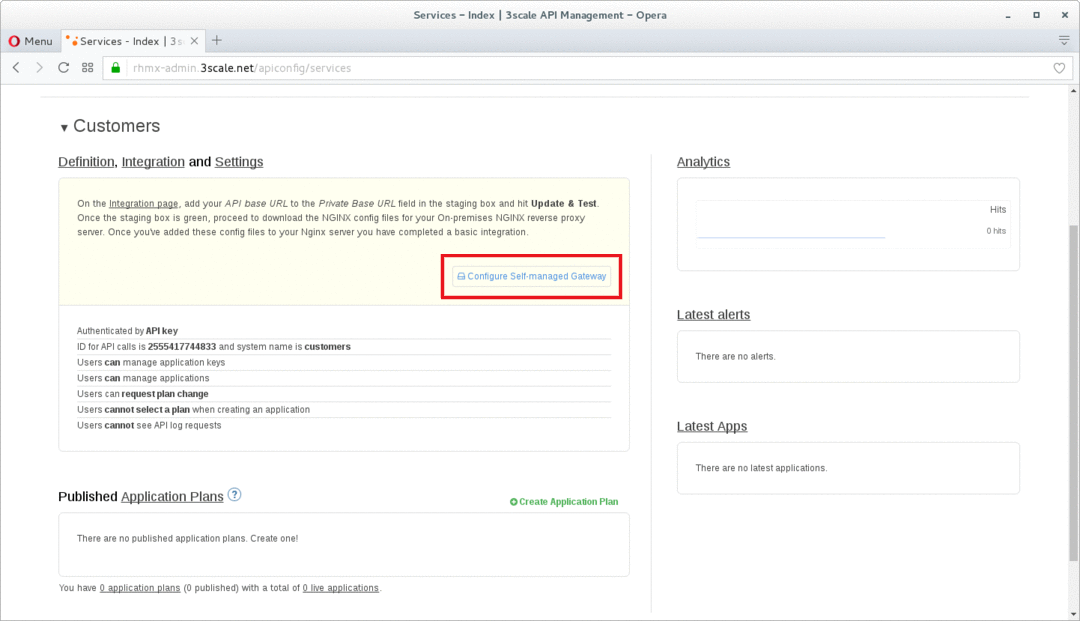
1. Create a second route for the production APIcast gateway with the following details:  
     
   Name: apicast-production  
   Hostname: customer-api-production-3scalegateway.10.0.2.15.xip.io  
   Service: apicast  
     
   You can do this clicking on **"Applications -> Route"** and then **Create Route**.  
     
   Your API Gateways are now ready to receive traffic. OpenShift takes care of load-balancing incoming requests to the route across the two running APIcast instances.  
     
   If you wish to see the APIcast logs, you can do so by clicking **Applications > Pods**, selecting one of the pods and finally selecting **Logs**.  
   Note: You need to complete the following steps to configure 3scale before you can use this route.
2. Before we move on to configure 3scale note the IP address of the *camel-ose-springboot-xml* service in your **myfuseproject** by going to **Applications -> Services**.  
     
   The IP address is listed under Cluster IP, for example "172.30.114.117". Copy this address.  
     
   
3. Login to your 3scale Admin Portal (http://<YOURDOMAIN>-admin.3scale.net) The admin portal provides access to a number of configuration features.



1. Ensure you are on the API page by selecting the **API** tab. Before creating the new service for our Customer API, you must edit the existing Echo API in your 3scale environment. The reason for this is that you can only run one service in the 3scale trial environment. In the full environment you can manage more than one API.   
     
   You edit the existing API by clicking on the **API** (which may be called Echo API or just API) and selecting **Definition**.   
     
   On the **Definition** page click on the **edit** link. Change the name of the service to **Customers**.
2. Return to the API definition page, select the ‘Integration’ link and then select ‘edit integration settings’.

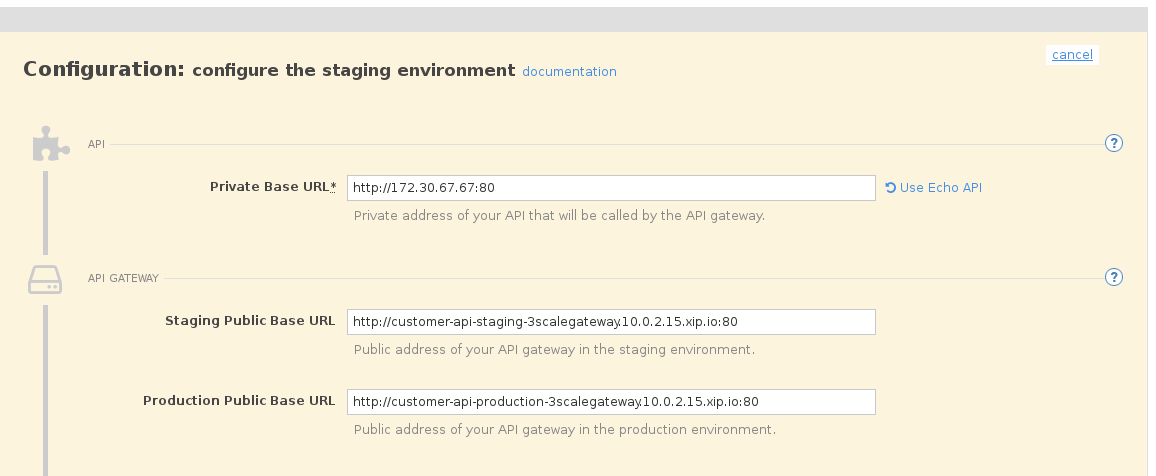




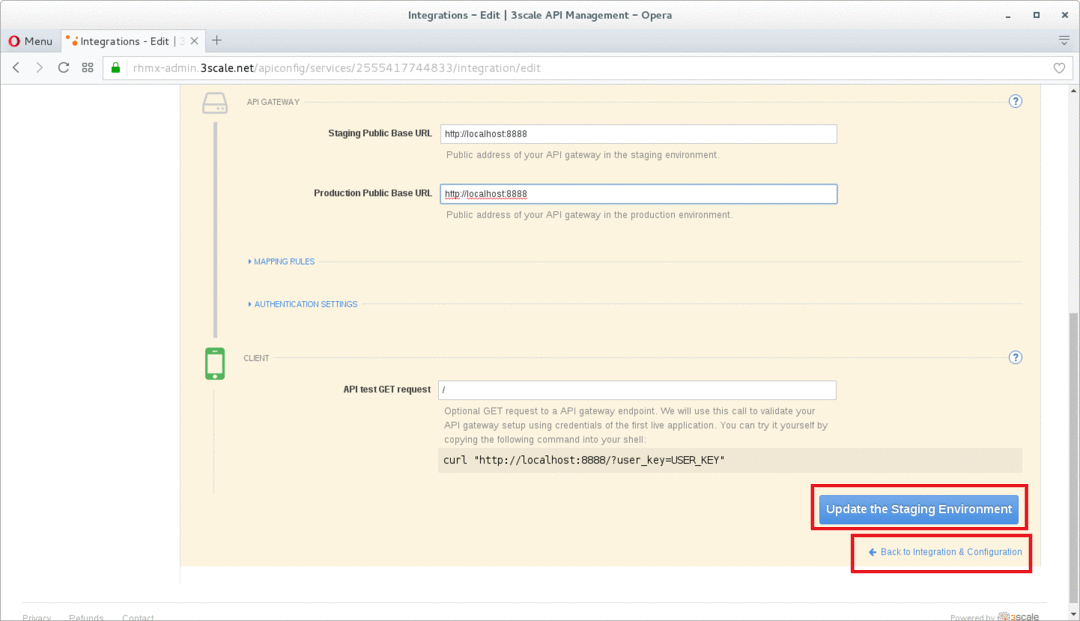
1. Select the **APIcast self-managed** Gateway deployment option.
2. Keep the **API Key (user\_key)** Authentication and Click on **Update Service**  
   
3. Select your new API and click on **Configure Self-managed Gateway**

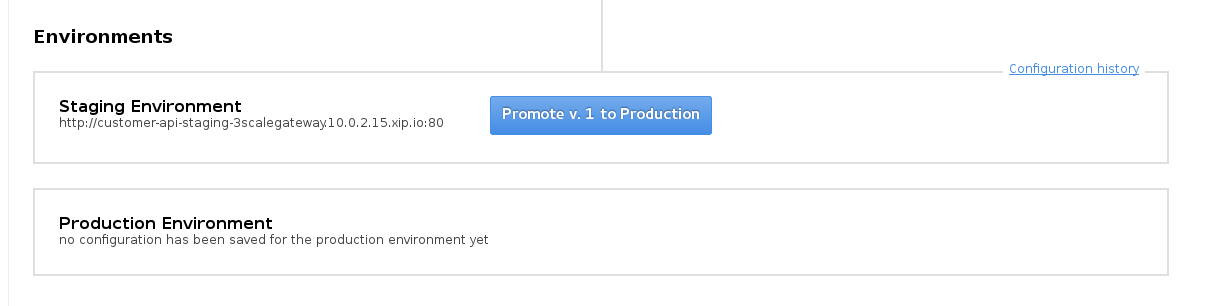
(Optional) If you get a message "**Introducing a brand new APIcast**", upgrade to the latest proxy by clicking on "**Start using the latest APIcast**".

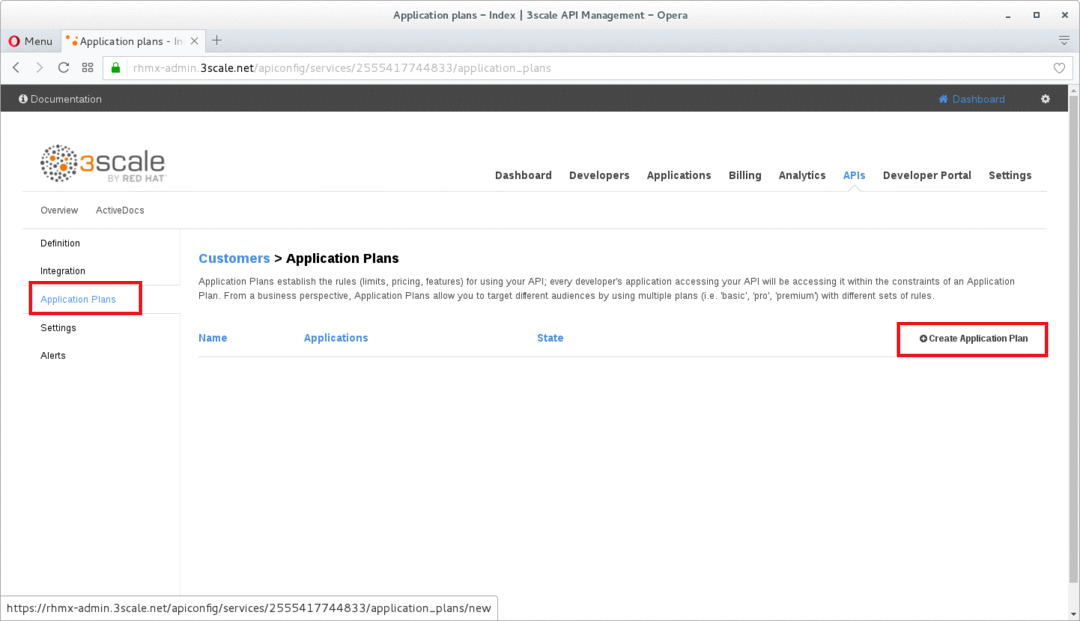
1. Click on the **add the Base URL of your API and save the configuration** button
2. Fill in the information for accessing your API. The private Base URL is the IP address of your Camel service (that you noted down earlier in step 15) and port 80, **for example**  
     
    http://172.30.114.117:80
3. Next fill in the public base URLs that point to the APIcast Gateway deployed on OpenShift::  
      
   http://customer-api-staging-3scalegateway.10.0.2.15.xip.io:80  
    for staging and   
     
   http://customer-api-production-3scalegateway.10.0.2.15.xip.io:80  
    for production



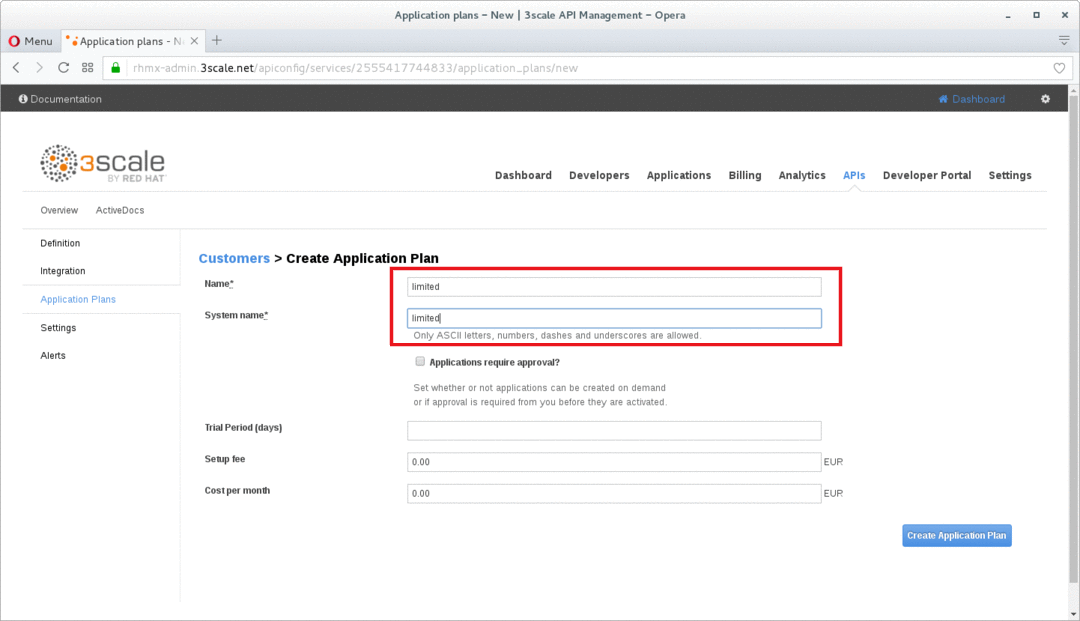
1. To make testing easier in the **API test GET request** field put the query parameters  
     
    /myfuselab/customer/all
2. Click on the **Update the Staging Environment** to save the changes and then click on the **Back to Integration & Configuration** link.

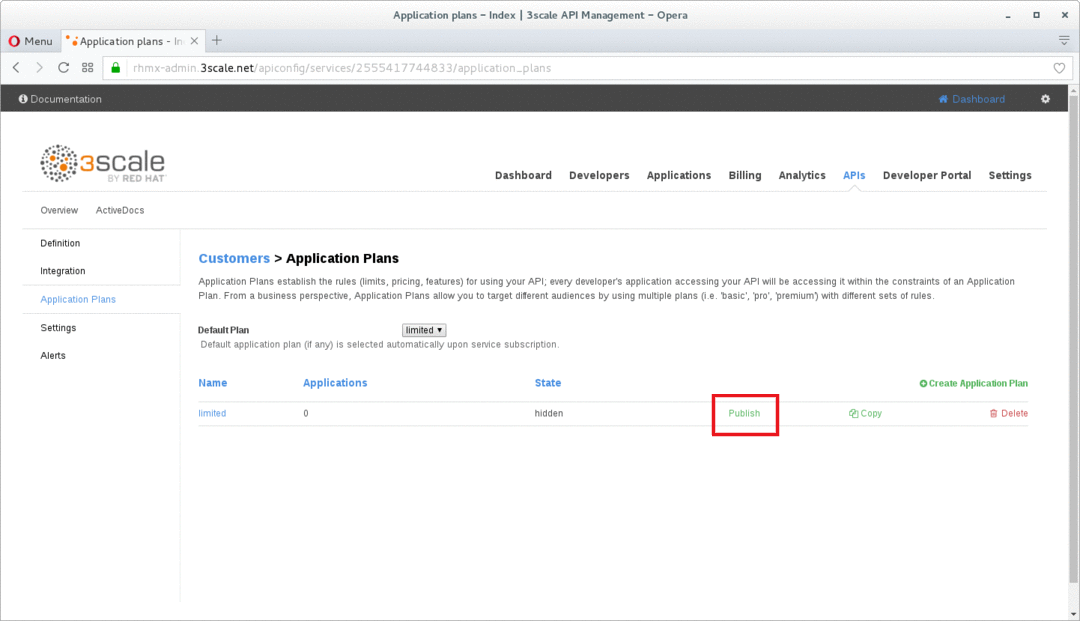
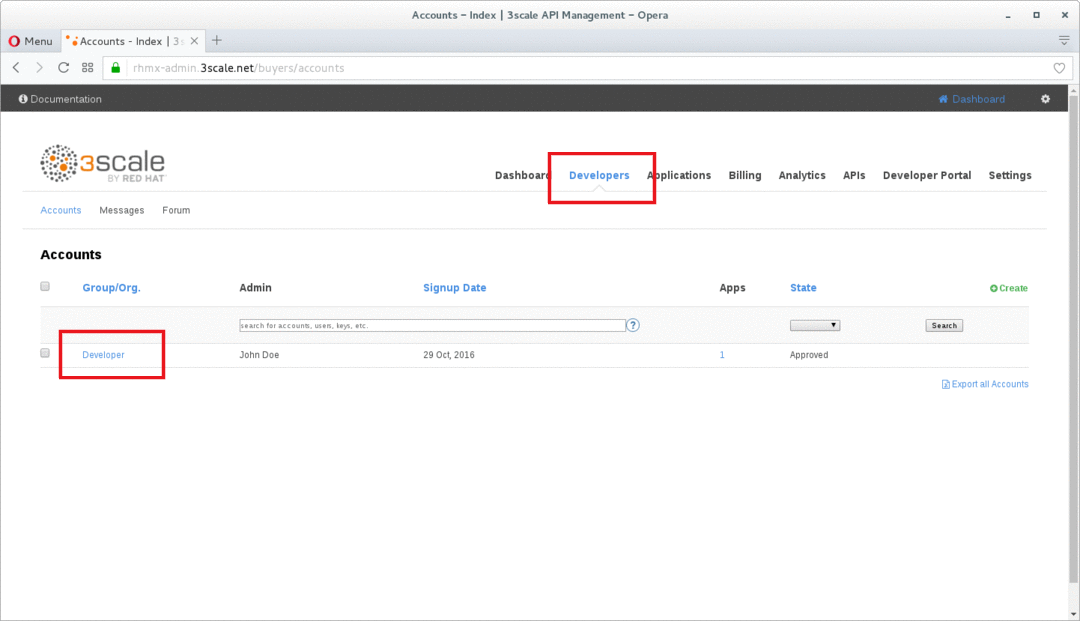
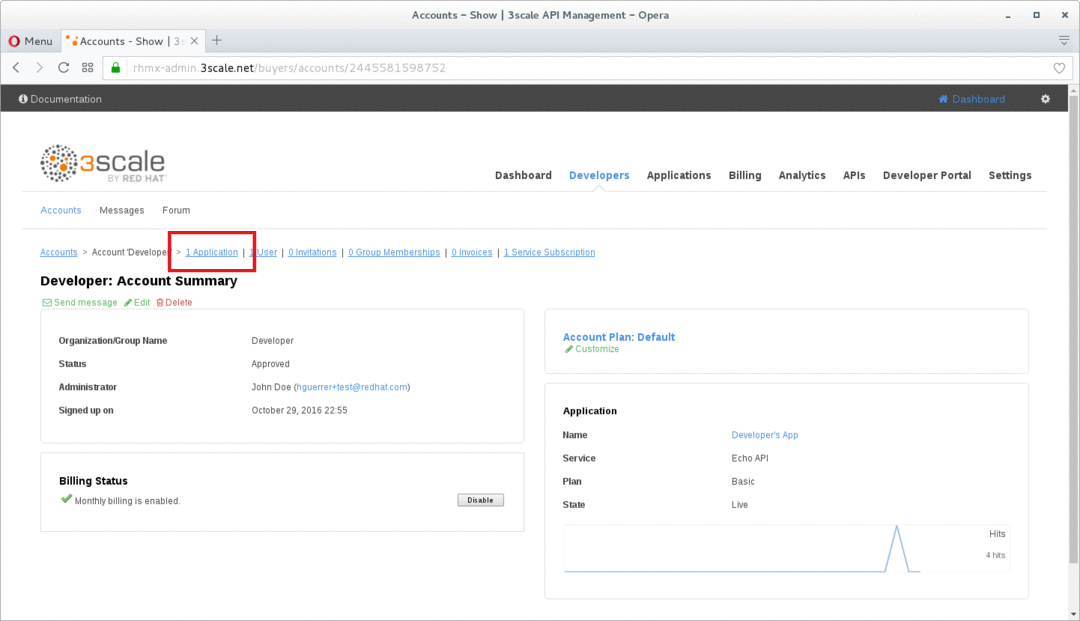
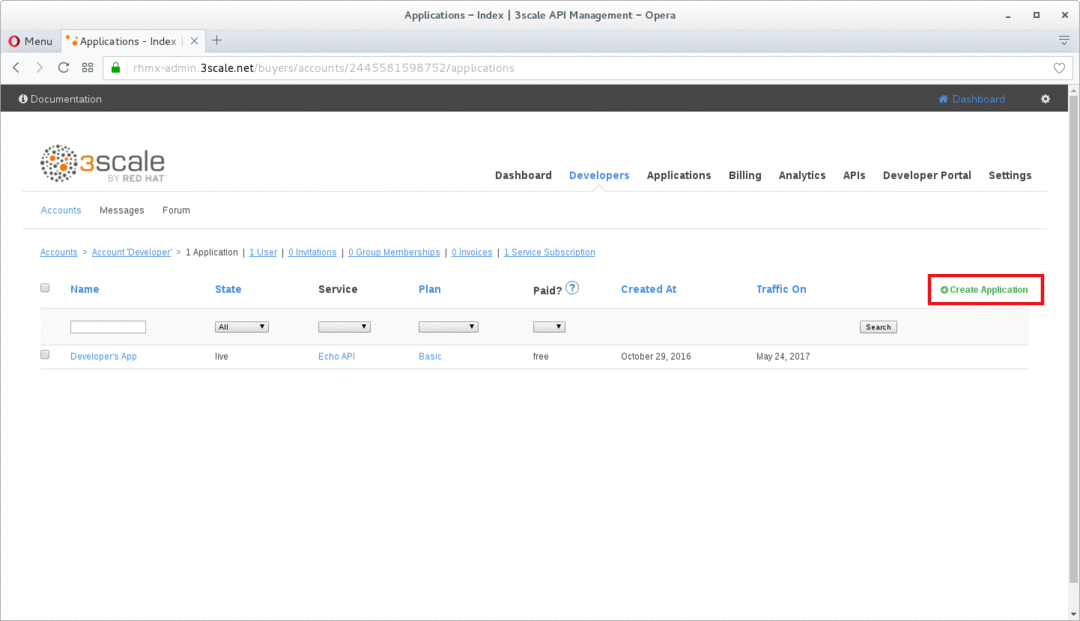
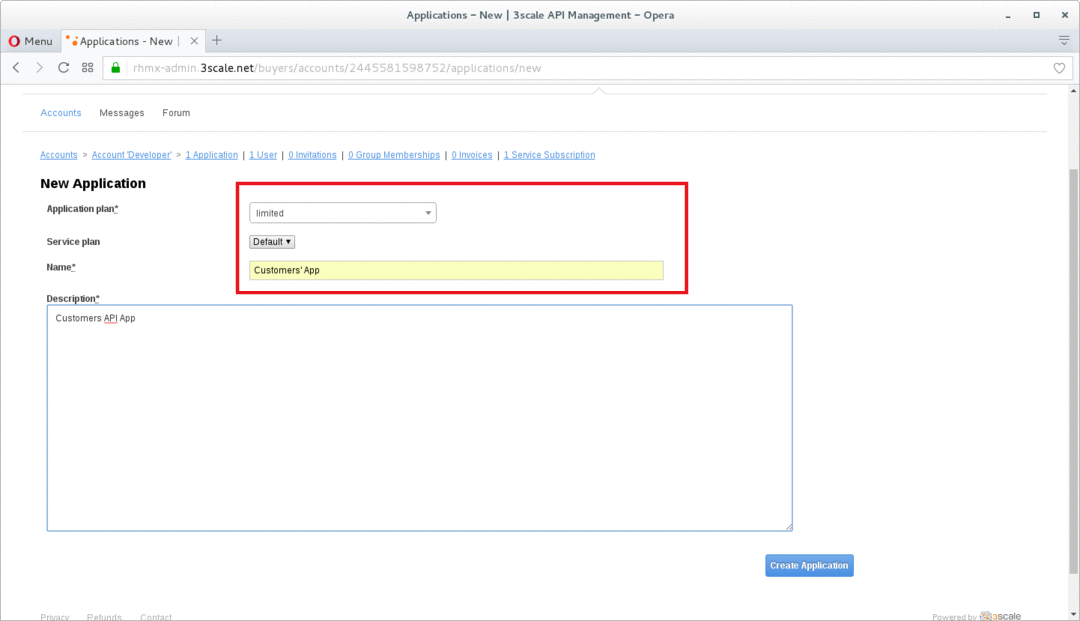
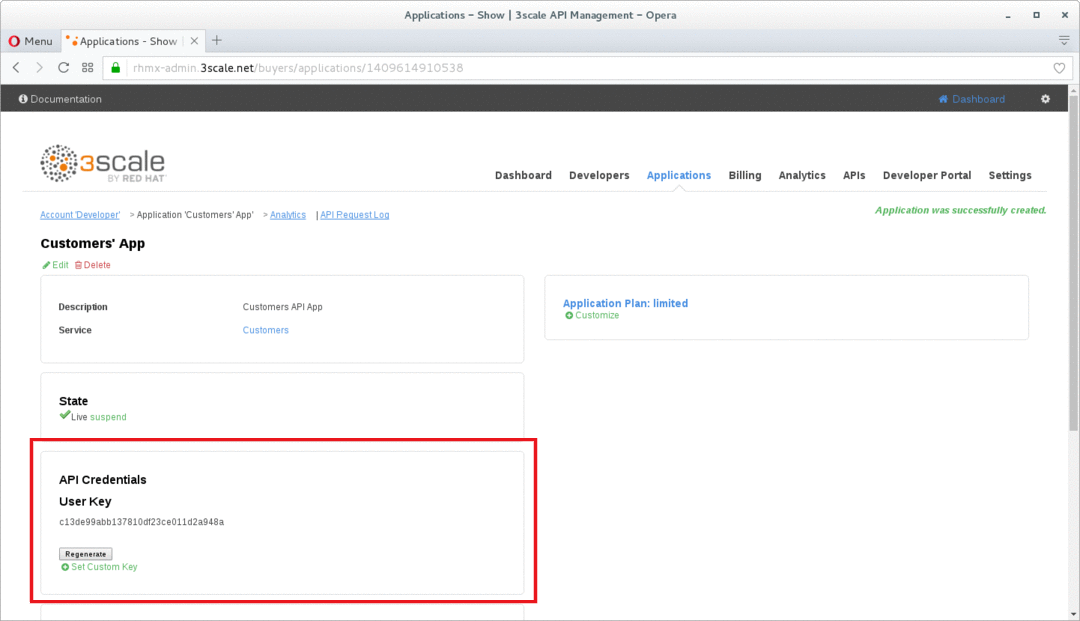


1. Under Environments click on the blue **Promote v. 1 to Production link** next to **Staging Environment**.  
     
   
2. After any configuration changes in the 3Scale API Manager you must redeploy the APIcast Gateway in OpenShift.   
     
   Wait, at least 1 minute and then go back to the OpenShift console   
     
   Go to **Applications** -> **Deployments** in the ***gateway*** project, select ***apicast*** and click **Deploy**. Wait until the gateway was successfully redeployed.
3. Success! Your 3scale access control layer will now only allow authenticated calls through to your backend API.
4. In the previous step, you ensured that only authenticated calls are allowed through to your API. Now you will apply policies to differentiate rate limits.  
     
   In 3scale terms, *applications* define the credentials to access your API. An application is always associated with one *application plan*, which determines the access policies. Applications are stored within *developer accounts* – in the basic 3scale plans only a single application is allowed, but in the higher plans multiple applications per account are allowed.
5. Let's create a new application plan for this example. In order to do this, navigate to the **Application Plans** tab in your Customers API and click on **Create Application Plan**.



1. Fill in the information for the name of your plan. In the form that opens, specify "limited" as the "Name" and the "System name". Then click on **Create Application Plan** button.



1. Your plan is created, now you need you to publish it. Click on the **Publish** link to made it public.
2. To associate the application plan with an application, navigate to the **Developers** tab and click on the Developer link.
3. Click on the Application link to access this developer's applications.
4. Now, create a new Application by clicking the **Create Application** button.
5. Select the previously created application plan ("Customers -> limited") from the combo box and provide "Customers" in the Name and Description field.  
     
   Click on the **Create Application** to save your changes.
6. In the next screen, you will be presented with the auto generated user key that will be used to access your API.
7. Test that APIcast authorizes a valid call to your API, by executing a curl command with your valid *user\_key* to the *hostname* that you configured in the previous step:

curl -i "http://customer-api-production-3scalegateway.10.0.2.15.xip.io/myfuselab/customer/all?user\_key=YOUR\_USER\_KEY"

1. You should see the following messages:

HTTP/1.1 200 OK

Server: openresty/1.11.2.2

Date: Tue, 30 May 2017 20:13:33 GMT

Content-Type: application/json

Transfer-Encoding: chunked

X-Application-Context: application:dev

accept: \*/\*

breadcrumbId: ID-traveler-laptop-rh-mx-redhat-com-45222-1496169770755-0-16

forwarded: for=192.168.42.1;host=customer-api-staging.192.168.42.100.nip.io;proto=http

user-agent: curl/7.29.0

user\_key: c13de99abb137810df23ce011d2a948a

x-3scale-proxy-secret-token: Shared\_secret\_sent\_from\_proxy\_to\_API\_backend\_71cfe31d89d8cf53

x-forwarded-for: 192.168.42.1

x-forwarded-host: customer-api-staging.192.168.42.100.nip.io

x-forwarded-port: 80

x-forwarded-proto: http

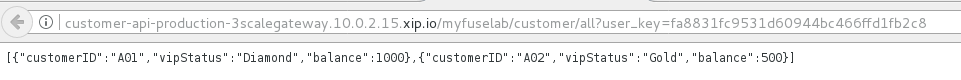
x-real-ip: 172.17.0.1

Set-Cookie: e286b151c44656235d8bdca6ee183477=e58d9930d57779957bf1695b6c805dcd; path=/; HttpOnly

Cache-control: private

[{"CUSTOMERID":"A01","VIPSTATUS":"Diamond","BALANCE":1000},{"CUSTOMERID":"A02","VIPSTATUS":"Gold","BALANCE":500}]

The last line is the same output as when calling the API directly.

1. Try from your web browser, again replacing the user key with your specific API key:   
     
   http://customer-api-production-3scalegateway.10.0.2.15.xip.io/myfuselab/customer/all?user\_key=YOUR\_USER\_KEY  
     
   
2. You have successfully configured 3Scale API Management and Gateway to access your API.

You have now completed all the labs. You created a microservice in JBoss Fuse using Spring-boot using the JBoss Developer Studio. You ran this microservice standalone in JBoss developer studio. You then exposed an API to access you microservice and deployed your microservice as a container running on OpenShift.

Next you configured 3Scale API Gateway as a container on OpenShift and the 3Scale API Management using the SaaS offering. Then you configure access to your microservice API to be controlled by 3Scale and secure by use of a secret and user\_key.

# Appendix

### camel-context.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:camel="http://camel.apache.org/schema/spring"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=" http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-spring.xsd">

<!-- Define a traditional camel context here -->

<bean

class="org.apache.camel.component.servlet.CamelHttpTransportServlet" id="camelHttpTransportServlet"/>

<bean

class="org.springframework.boot.web.servlet.ServletRegistrationBean" id="servlet">

<property name="name" value="CamelServlet"/>

<property name="servlet" ref="camelHttpTransportServlet"/>

<property name="urlMappings" value="/myfuselab/\*"/>

</bean>

<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">

<restConfiguration apiContextPath="api-docs" bindingMode="json"

component="servlet" contextPath="/myfuselab">

<apiProperty key="cors" value="true"/>

<apiProperty key="api.title" value="My First Camel API Lab"/>

<apiProperty key="api.version" value="1.0.0"/>

</restConfiguration>

<rest path="/customer">

<get uri="all">

<description>Retrieve all customer data</description>

<to uri="direct:getallcustomer"/>

</get>

</rest>

<route id="\_route1">

<from id="direct1" uri="direct:getallcustomer"/>

<to id="\_to1" uri="sql:select \* from customerdemo?dataSource=dataSource"/>

<log id="\_log1" message="${body}"/>

</route>

</camelContext>

</beans>

<http://red.ht/camel-context>

### pom.xml

<?xml version="1.0" encoding="UTF-8"?>  
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd" xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  
 <modelVersion>4.0.0</modelVersion>  
 <groupId>org.mycompany</groupId>  
 <artifactId>camel-ose-springboot-xml</artifactId>  
 <version>1.0.0-SNAPSHOT</version>  
 <name>Fabric8 :: Quickstarts :: Spring-Boot :: Camel XML</name>  
 <description>Spring Boot example running a Camel route defined in XML</description>  
 <properties>  
 <spring-boot.version>1.4.1.RELEASE</spring-boot.version>  
 <fabric8.version>2.2.170.redhat-000013</fabric8.version>  
 <maven-compiler-plugin.version>3.6.0</maven-compiler-plugin.version>  
 <fabric8.maven.plugin.version>3.1.80.redhat-000013</fabric8.maven.plugin.version>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <maven-surefire-plugin.version>2.19.1</maven-surefire-plugin.version>  
 <run.profiles>dev</run.profiles>  
 </properties>  
 <dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>io.fabric8</groupId>  
 <artifactId>fabric8-project-bom-camel-spring-boot</artifactId>  
 <version>${fabric8.version}</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
 </dependencies>  
 </dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>javax.inject</groupId>  
 <artifactId>javax.inject</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.camel</groupId>  
 <artifactId>camel-spring-boot-starter</artifactId>  
 <version>2.18.1.redhat-000015</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-actuator</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.jboss.arquillian.junit</groupId>  
 <artifactId>arquillian-junit-container</artifactId>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>io.fabric8</groupId>  
 <artifactId>fabric8-arquillian</artifactId>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.camel</groupId>  
 <artifactId>camel-sql-starter</artifactId>  
 <version>2.18.1.redhat-000015</version>  
 </dependency>  
 <dependency>  
 <groupId>com.h2database</groupId>  
 <artifactId>h2</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.camel</groupId>  
 <artifactId>camel-servlet-starter</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.camel</groupId>  
 <artifactId>camel-jackson-starter</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.camel</groupId>  
 <artifactId>camel-swagger-java-starter</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-jdbc</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
 </dependencies>  
 <repositories>  
 <repository>  
 <releases>  
 <enabled>true</enabled>  
 <updatePolicy>never</updatePolicy>  
 </releases>  
 <snapshots>  
 <enabled>false</enabled>  
 </snapshots>  
 <id>fuse-public-repository</id>  
 <name>FuseSource Community Release Repository</name>  
 <url>https://repo.fusesource.com/nexus/content/groups/public</url>  
 </repository>  
 <repository>  
 <releases>  
 <enabled>true</enabled>  
 <updatePolicy>never</updatePolicy>  
 </releases>  
 <snapshots>  
 <enabled>false</enabled>  
 </snapshots>  
 <id>red-hat-ga-repository</id>  
 <name>Red Hat GA Repository</name>  
 <url>https://maven.repository.redhat.com/ga</url>  
 </repository>  
 <repository>  
 <releases>  
 <enabled>true</enabled>  
 <updatePolicy>never</updatePolicy>  
 </releases>  
 <snapshots>  
 <enabled>false</enabled>  
 </snapshots>  
 <id>red-hat-ea-repository</id>  
 <name>Red Hat EA Repository</name>  
 <url>https://maven.repository.redhat.com/earlyaccess/all/</url>  
 </repository>  
 </repositories>  
 <pluginRepositories>  
 <pluginRepository>  
 <releases>  
 <enabled>true</enabled>  
 <updatePolicy>never</updatePolicy>  
 </releases>  
 <snapshots>  
 <enabled>false</enabled>  
 </snapshots>  
 <id>fuse-public-repository</id>  
 <name>FuseSource Community Release Repository</name>  
 <url>https://repo.fusesource.com/nexus/content/groups/public</url>  
 </pluginRepository>  
 <pluginRepository>  
 <releases>  
 <enabled>true</enabled>  
 <updatePolicy>never</updatePolicy>  
 </releases>  
 <snapshots>  
 <enabled>false</enabled>  
 </snapshots>  
 <id>red-hat-ga-repository</id>  
 <name>Red Hat GA Repository</name>  
 <url>https://maven.repository.redhat.com/ga</url>  
 </pluginRepository>  
 <pluginRepository>  
 <releases>  
 <enabled>true</enabled>  
 <updatePolicy>never</updatePolicy>  
 </releases>  
 <snapshots>  
 <enabled>false</enabled>  
 </snapshots>  
 <id>red-hat-ea-repository</id>  
 <name>Red Hat EA Repository</name>  
 <url>https://maven.repository.redhat.com/earlyaccess/all/</url>  
 </pluginRepository>  
 </pluginRepositories>  
 <build>  
 <defaultGoal>spring-boot:run</defaultGoal>  
 <plugins>  
 <plugin>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>${maven-compiler-plugin.version}</version>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 </configuration>  
 </plugin>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 <version>${spring-boot.version}</version>  
 <executions>  
 <execution>  
 <goals>  
 <goal>repackage</goal>  
 </goals>  
 </execution>  
 </executions>  
 </plugin>  
 <plugin>  
 <groupId>io.fabric8</groupId>  
 <artifactId>fabric8-maven-plugin</artifactId>  
 <version>${fabric8.maven.plugin.version}</version>  
 <executions>  
 <execution>  
 <goals>  
 <goal>resource</goal>  
 <goal>build</goal>  
 </goals>  
 </execution>  
 </executions>  
 </plugin>  
 </plugins>  
 </build>  
</project>

<http://red.ht/pom-xml>