

# Predix

# Predix Developer Boot Camp

## Student Lab Guide

March 2016



GE Digital

# Predix

---

© 2016 General Electric Company.

GE, the GE Monogram, and Predix are either registered trademarks or trademarks of General Electric Company. All other trademarks are the property of their respective owners.

This document may contain Confidential/Proprietary information of GE, GE Global Research, GE Digital, and/or its suppliers or vendors. Distribution or reproduction is prohibited without permission.

THIS DOCUMENT AND ITS CONTENTS ARE PROVIDED "AS IS," WITH NO REPRESENTATION OR WARRANTIES OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO WARRANTIES OF DESIGN, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. ALL OTHER LIABILITY ARISING FROM RELIANCE UPON ANY INFORMATION CONTAINED HEREIN IS EXPRESSLY DISCLAIMED.

Access to and use of the software described in this document is conditioned on acceptance of the End User License Agreement and compliance with its terms.

# Getting Started

This guide provides step-by-step instructions for lab exercises. Each lab corresponds to a topic covered in class and provides students with hands-on experience developing UI components on the Predix platform.

## Course Prerequisites:

- Courses are intended to introduce developers to using the Predix Platform. It is assumed that students have familiarity with:
  - ◆ Development platforms and frameworks
  - ◆ Java, HTML, JavaScript, CSS, Angular, Polymer and Sass

## Log into the Hosted Environment

- You received an email prior to the start of class with your access code. Use that access code to log into

<https://predix.instructorled.training>

**Tip:** You do not need to provide a password.

- Click the **Lab** Link and click **Connect to the lab**
- Click the **Log in** button (you do not need to change the Login or Password here)
- Click the **Yes** button to resize the desktop to your screen  
You will see a message that the desktop is being resized and you are connected to the remote session.

**Note:** All lab exercises will be completed in this hosted environment on the DevBox.

## How to Copy and Paste in your Environment

- In a Terminal window
  - ◆ Copy using **Ctrl + Shift + C**
  - ◆ Paste using **Ctrl + Shift + V**
- All other applications
  - ◆ Copy using **Ctrl + C**
  - ◆ Paste using **Ctrl + V**







# Lab 1: Getting Started with Cloud Foundry

## Learning Objectives

By the end of the lab, you will be able to:

- Log into Cloud Foundry
- Explore the Predix Service Catalog/Marketplace
- Create a service instance

## Lab Exercises

- [Logging into Cloud Foundry, page 2](#)
- [Creating a Service Instance, page 5](#)

## Cloud Foundry Commands

Command	Description
cf login	Login to cloud foundry
cf marketplace	Display all services in the catalog
cf create-service	Create a new service instance
cf apps	Display all applications in your space
cf push	Deploy an application
cf services	Display all service instances in your space



---

## Exercise 1: Logging into Cloud Foundry

---

### Overview

In this exercise you will practice logging in to Cloud Foundry.

### Steps

1. Log into Cloud Foundry.

- Double-click the Terminal window icon on your desktop to open a Terminal window



- Enter this command to log into Cloud Foundry:  
`cf login -a https://api.system.aws-usw02-pr.ice.predix.io`

**Note:** This command logs you into GE's API endpoint for Cloud Foundry.

## 2. Enter your Cloud Foundry credentials.

- You are prompted to enter your email
  - ◆ Type in your student account (your instructor will provide this)
- You are prompted to enter your password (your instructor will provide this)
  - ◆ Press **Enter**

**Tip:** Your password will not appear on the screen as you are typing.

## 3. Select a space.

- You are prompted to select a space
  - ◆ Enter the number of the targeted space that your instructor provides

```
[predix@localhost ~]$ cf login -a https://api.system.aws-usw02-pr.ice.pre
API endpoint: https://api.system.aws-usw02-pr.ice.predix.io

Email> student55

Password>
Authenticating...
OK

Targeted org Predix-Training

Select a space (or press enter to skip):
1. Training1
2. Training2
3. Training3

Space> 1
Targeted space Training1
```

The terminal displays the API endpoint, user, and organization and space which you are logged into.

**Tip** ~To change your space:

- In the terminal, run this command:  
`cf target -s <Name of the Space>`

You are now logged in.

```
API endpoint:  https://api.system.aws-usw02-pr.ice.predix.io (API versio
.0)
User:         student55
Org:         Predix-Training
Space:       Training1
[predix@localhost ~]$ █
```

## Exercise 2: Creating a Service Instance

### Overview

In this exercise, you will use Cloud Foundry commands to display all services in the marketplace and create an instance of the postgres service in your Cloud Foundry space.

### Steps

1. Display marketplace services.

- In the Terminal, run this command:

```
cf marketplace
```

All available services appear. You will be creating an instance of the postgres service.

```
[predix@localhost ~]$ cf m
Getting services from marketplace in org Predix-Training / space
.
OK
```

service	plans	description
<b>logstash-5</b> opment and testing	free	Logstash 1.4 serv
<b>p-rabbitmq</b> formance multi-protocol messaging broker.	standard	RabbitMQ is a rob
<b>postgres</b>	shared, shared-nr	Reliable PostgrSQ
<b>predix-acs</b> l framework than basic User Account and Authorization.	Beta, Enterprise*	Use this service
<b>predix-analytics-catalog</b> with the Analytics Runtime Service.	Beta, Enterprise*	Add analytics to
<b>predix-analytics-runtime</b> ion of the analytic orchestration.	Beta, Enterprise*	Use this service
<b>predix-asset</b>	Beta, Enterprise*	Create and store

## 2. Create a new postgres service instance in your space.

- Enter the command to create your own service instance with the following syntax:

```
cf create-service <service> <plan> <your_name-service_name>
```

### Example:

```
[predix@localhost ~]$ cf create-service postgres shared-nr sample-postgres
Creating service instance sample-postgres in org Predix-Training / space Training2 as
OK
[predix@localhost ~]$ █
```

## 3. Display all services instances in your space.

- In the Terminal, run the **cf services** command

The service instances in your space are listed.

name	service	plan	bound apps
annas-httpdata-postgres	postgres	shared-nr	
dprodbCon	postgres	shared-nr	dProAdminModule, dProDBPack
sample-postgres	postgres	shared-nr	

```
[predix@localhost ~]$ █
```

## Lab 2: *Deploying and Monitoring Applications*

### Learning Objectives

By the end of the lab, you will be able to use the Cloud Foundry CLI (Command Line Interface) tool to:

- Add a service
- Monitor a service

### Lab Exercises

- [Deploying an Application, page 8](#)
- [Using a Manifest File to Deploy an Application, page 11](#)
- [Managing your Environment, page 14](#)
- [Monitoring your Application, page 15](#)

### Directions

Complete the exercises that follow.



---

## Exercise 1: Deploying an Application

---

### Overview

In this exercise, you will deploy an application to Cloud Foundry from the command line interface (CLI).

### Steps

1. Change to the `spring-music` directory in the Terminal window.

- To determine your current directory, enter `pwd` and press **Enter**

```
[predix@localhost ~]$ pwd
/predix
[predix@localhost ~]$ █
```

- If you are not in the home directory (`/predix`), run the command: `cd` and then `pwd` again to confirm you are in the root directory
- Change to the `spring-music` directory by entering the command below:

```
[predix@localhost ~]$ cd PredixApps/training_labs/CloudFoundryLabs/spring-music/
[predix@localhost spring-music]$ pwd
/predix/PredixApps/training_labs/CloudFoundryLabs/spring-music
[predix@localhost spring-music]$ █
```

- Run the command `cf push` to publish the application instance of the web application. The command fails because you have not provided the parameters it needs.

**FAILED**

```
Error: Manifest file is not found in the current directory, please
provide either an app name or manifest
[preidix@localhost spring-music]$
```

- Run the command `cf push -h` and read through the information presented

**Tip:** `-h` is for help. Any command used with `-h` tells the CLI to return help information for the command. You will see the command definition, its syntax, and its options (parameters).

- Run the `cf push` command with the following syntax to correctly publish your application: `cf push <app-name> -p <path/war_filename>`

**Note:** You should be in the `spring-music` directory, which contains the pre-built sub-directory with the **Web application AR**chive (.war), or wrapper file with all of the application files required for deployment.

```
[preidix@localhost spring-music]$ cf push student55-spring-music -p pre-built/spring-music.war
Creating app student55-spring-music in org Predix-Training / space Training1 as student55...
OK
```

Your application should successfully publish to Cloud Foundry.

```
Showing health and status for app student55-spring-music in org Predix-Training
student55...
OK

requested state: started
instances: 1/1
usage: 1G x 1 instances
urls: student55-spring-music.run.aws-usw02-pr.ice.predix.io
last uploaded: Thu Mar 10 17:00:23 UTC 2016
stack: cflinuxfs2
buildpack: java-buildpack=v3.5.1-http://github.com/pivotal-cf/pcf-java-buildpack
k-like-jre=1.8.0_73 open-jdk-like-memory-calculator=2.0.1_RELEASE spring-auto-re
_RELEASE tomcat-access-logging-support=2.5.0_RELEASE tomcat-ins...

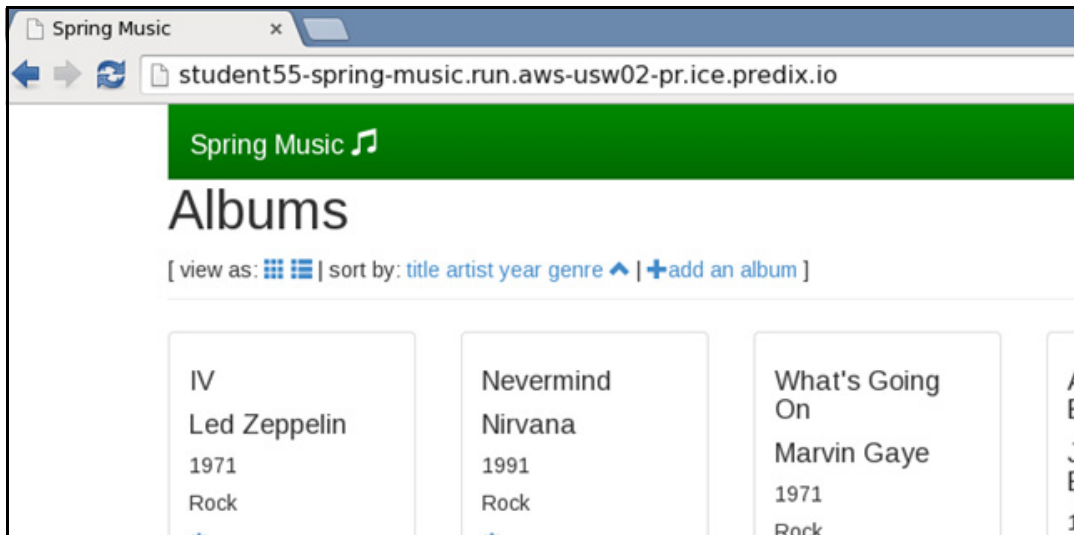
#0   state      since                cpu    memory       disk          det
#0   running    2016-03-10 09:01:07 AM  0.0%   641.9M of 1G  153.3M of 1G
```

1. Test your application in a web browser

- Enter the command `cf a` to find the URL of your application

name	requested state	instances	memory
<b>urls</b>			
<code>awsblob</code>	started	1/1	1G
<code>awsblob.run.aws-usw02-pr.ice.predix.io</code>			
<code>dataingestion_service_student1</code>	started	1/1	1G
<code>dataingestion-service-student1.run.aws-usw02-pr.ice.predix.io</code>			
<code>rmd_datasource_student1</code>	started	1/1	1G
<code>rmd-datasource-student1.run.aws-usw02-pr.ice.predix.io</code>			
<code>rmd_ref_app_ui_student1</code>	started	1/1	64M
<code>rmd-ref-app-ui-student1.run.aws-usw02-pr.ice.predix.io</code>			
<code>student55-spring-music</code>	started	1/1	1G
<code>student55-spring-music.run.aws-usw02-pr.ice.predix.io</code>			
<code>supplierlogin</code>	started	1/1	1G
<code>supplierlogin.run.aws-usw02-pr.ice.predix.io</code>			

- Copy your application URL (located in the URL column of the output)
- Open the Firefox Web browser, and paste your application URL  
The Spring Music application displays in your browser.



---

## Exercise 2: Using a Manifest File to Deploy an Application

---

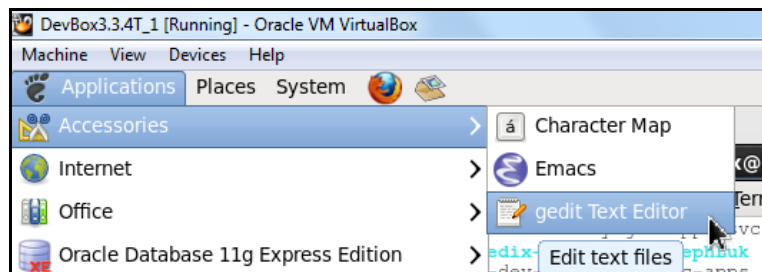
### Overview

In this exercise you will edit a manifest file and use it to deploy your application instance. The manifest file includes a list of parameters that indicate how the solution should be deployed. Some of the parameters are required, and some are optional. You can also provide these parameters in the command line, but a manifest file is usually used to reduce the complexity of the command, and to save the information for later reuse.

### Steps

1. Add deployment parameters to the manifest file.

- Open the gedit text editor from the Applications menu at the top left of the DevBox
  - ◆ From the Applications menu, select *Accessories>gedit Text Editor*



- Enter **Ctrl + O** and browse to the **manifest.yml** file in the following folder  
`predix/PredixApps/training_labs/CloudFoundryLabs/cf-spring-mvc-demo`

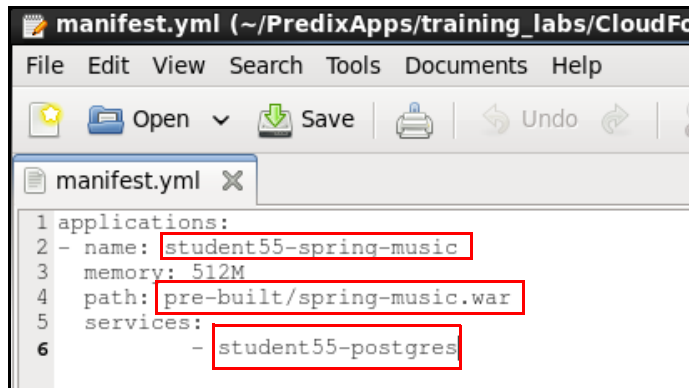
- Edit the manifest file as follows
  - ◆ Change the application name to the name of the application (microservice) you created in the previous exercise
    - To verify the correct name, enter **cf a** command and locate your application (microservice) name

name	requested state	instances	memory
awsblob	started	1/1	1G
awsblob.run.aws-usw02-pr.ice.predix.io			
dataingestion_service_student1	started	1/1	1G
dataingestion-service-student1.run.aws-usw02-pr.ice.predix.io			
rmd_datasource_student1	started	1/1	1G
rmd-datasource-student1.run.aws-usw02-pr.ice.predix.io			
rmd_ref_app_ui_student1	started	1/1	64M
rmd-ref-app-ui-student1.run.aws-usw02-pr.ice.predix.io			
student55-spring-music	started	1/1	1G
student55-spring-music.run.aws-usw02-pr.ice.predix.io			
supplierlogin	started	1/1	1G
supplierlogin.run.aws-usw02-pr.ice.predix.io			

- ◆ Change the path to **pre-built/spring-music.war**
- ◆ Change the service listed to (keep the hyphen and space in the file)
  - <your postgres service created in lab 1>
 This binds the postgresSQL service to your application (microservice)
- ◆ To find your service name, enter **cf s** (services) in the CLI and look for your postgresSQL service
- ◆ Save the file to the following folder  
**predix/PredixApps/training\_labs/CloudFoundryLabs/spring-music**

**Tip:** You are saving the manifest file into the same folder from which you deployed the spring-music application earlier so that you can use it to re-deploy the application without entering the parameters into the command line.

Your file should read as follows (but with your individual service and application names)



```
manifest.yml (~/PredixApps/training_labs/CloudFo
File Edit View Search Tools Documents Help
Open Save Undo
manifest.yml X
1 applications:
2 - name: student55-spring-music
3 memory: 512M
4 path: pre-built/spring-music.war
5 services:
6 - student55-postgres
```

## 2. Deploy the application to Cloud Foundry and confirm it is running.

- From the `spring-music` directory, deploy your spring-music service using the `cf push` command
  - ◆ Navigate to the spring-music directory in the CLI (you can check your directory by using the `pwd` command)
  - ◆ Run the `cf push` command (no parameters listed)
  - ◆ Copy the application URL into the web browser
  - ◆ Run the `cf a` command to see the application URL

---

## Exercise 3: Managing your Environment

---

### Overview

In this exercise you will scale, stop, and delete your application instance.

### Steps

1. Stop the application in Cloud Foundry.

- In the Terminal, run `cf stop <your-application-name>`

**Example:** `cf stop student55-spring-music`

The application stops

2. Scale your application up to 3 instances.

- Run `cf scale <your application name> -i 3`
- Enter the command to scale your application back down to 1 instance

**Tip:** The command to delete an application is shown below. **Do not delete** this application as you will use it in later labs.

`cf delete <your application name> -r`

**Note:** This deletes the application and any orphaned routes from Cloud Foundry.

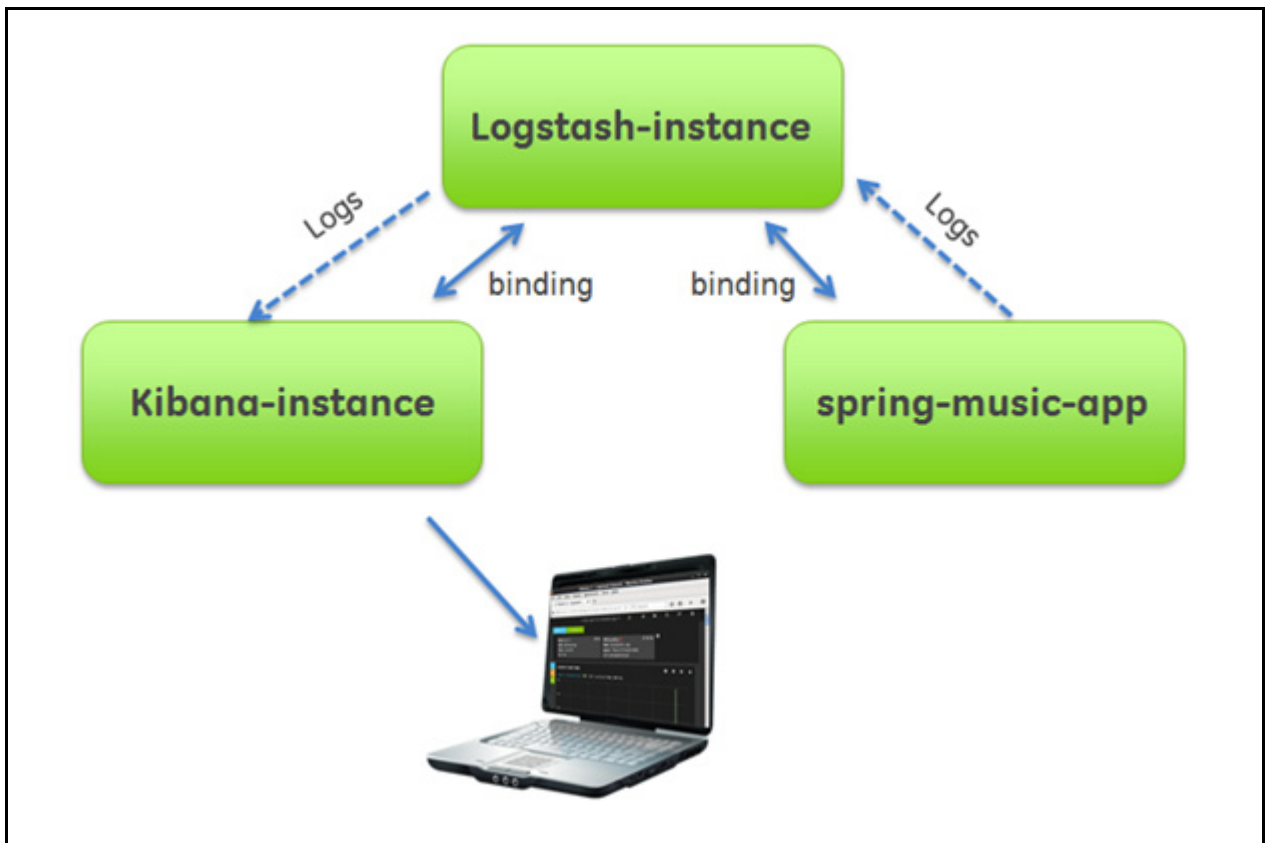
---

## Exercise 4: Monitoring your Application

---

### Overview

In this exercise you will create and bind a logging service instance to the spring-music application. You will also create an instance of the Kibana logging service (web application) and bind that to the logging service instance. Together, these three will provide a web-interface in which you can view spring-music application logs. The relationship is shown below.





## Steps

1. Create a Logstash service instance and bind it to your spring-music application.

- Run the following commands in the Terminal

- ◆ **cf create-service logstash-5 free your\_name-logstash**

Replace your\_name with your first name. This creates a Logstash service instance.

```
[predix@localhost spring-music]$ cf create-service logstash-5 free Xander-logstash
Creating service instance Xander-logstash in org Predix-Training / space Training1 as student66...
OK
[predix@localhost spring-music]$ █
```

- ◆ **cf bind-service your\_name-spring-music your\_name-logstash**

Use your spring-music application name and the name of the Logstash service instance you just created

```
[predix@localhost spring-music]$ cf bind-service Xanderspring-music Xander-logstash
Binding service Xander-logstash to app Xanderspring-music in org Predix-Training / space Training1 as student66...
OK
```

## 2. Restage your application and clone the Kibana logging application.

- **cf restage your\_name-spring-music**

This command re-deploys your application from the Cloud Foundry database.

- Clone the Kibana application from GitHub

```
git clone https://github.com/cloudfoundry-community/kibana-me-logs.git
```

This copies the Kibana logging application to your space

```
[predix@localhost spring-music]$ git clone https://github.com/cloudfoundry-community/kibana-me-logs.git
Cloning into 'kibana-me-logs'...
remote: Counting objects: 388, done.
remote: Total 388 (delta 0), reused 0 (delta 0), pack-reused 388
Receiving objects: 100% (388/388), 1000.67 KiB | 0 bytes/s, done.
Resolving deltas: 100% (84/84), done.
Checking connectivity... done.
[predix@localhost spring-music]$ █
```

## 3. Deploy the Kibana UI to Cloud Foundry.

- Use this command to change to the `kibana-me-logs` directory

- ◆ **cd kibana-me-logs**

- Run this command to deploy the Kibana UI application (change `your_name` to your first name)

- ◆ **cf push kibana-your\_name --no-start --random-route -b https://github.com/heroku/heroku-buildpack-go.git**

4. Bind the Kibana application to your Logstash service instance and start the application.

- Run this command to bind the Kibana application to your Logstash service instance  
**cf bind-service <your-kibana-app-name> <your-service-instance-name>**

```
[predix@localhost kibana-me-logs]$ cf bind-service kibana-Xander Xander-logstash
Binding service Xander-logstash to app kibana-Xander in org Predix-Training / space Training1 as student66...
OK
TIP: Use 'cf restage kibana-Xander' to ensure your env variable changes take effect
[predix@localhost kibana-me-logs]$
```

- Run this command to start the Kibana application  
**cf start <your-kibana-app-name>**

```
requested state: started
instances: 1/1
usage: 128M x 1 instances
urls: kibana-xander-wobegone-diopsimeter.run.aws-usw02-pr.ice.predix.io
last uploaded: Fri Dec 11 19:54:28 UTC 2015
stack: cflinuxfs2
buildpack: https://github.com/heroku/heroku-buildpack-go.git
```

5. Test the application.

- Use the URL of the Kibana application you just deployed and started in your Web browser to test the application

Your application shows logging information from your spring-music application.

The screenshot shows the Kibana 3 Logstash Search interface. The browser tab is 'Kibana 3 - Logstash ...'. The address bar shows 'kibana-xander-wobegone-diopsimeter.run.aws-usw02-pr.ice.predix.io/#/dashboard/file/apps-logs.json'. The main heading is 'Logstash Search'. Below the heading are two tabs: 'QUERY' and 'FILTERING'. Under 'FILTERING', there are two filter boxes:

- time must** (green dot): field: @timestamp, from: now-24h, to: now.
- field mustNot** (red dot): field: syslog5424\_app, query: "9acc1c73-bc2f-4065-a27a-eb3c8d57b7ad".

Below the filters is the 'EVENTS OVER TIME' section. It shows 'View', 'Zoom Out', and a green dot with '(49) count per 10m | (49 hits)'. A vertical axis on the left is labeled '50'.

The screenshot shows the 'ALL EVENTS' section of the Kibana Logstash Search interface. The heading is 'ALL EVENTS'. Below it is 'Fields' with a search icon. On the right, it says '0 to 49'. Below 'Fields' is a filter input box with 'Type to filter...'. A note states: 'Note These fields have been extracted from your mapping. Not all fields may be available in your source document.' Below the note is a list of fields with checkboxes:

- @message
- @message.raw
- @source\_host
- @source\_host.raw

Below the fields list is a section titled '\_source (select columns from the list to the left)'. It contains three log messages:

```

{"message": "287 <14>1 2015-12-11T19:48:35.131971+00:00 loggregator d31112b7-7086-422b-8000-000000000000 web application directory /home/vcap/app/.java-buildpack/tomcat/webapps/ROOT has finished"}
{"message": "255 <14>1 2015-12-11T19:48:34.924135+00:00 loggregator d31112b7-7086-422b-8000-000000000000 of type [class org.springframework.web.servlet.mvc.ParameterizableViewController]"}
{"message": "333 <14>1 2015-12-11T19:48:34.915666+00:00 loggregator d31112b7-7086-422b-8000-000000000000 /kill],methods=[],params=[],headers=[],consumes=[],produces=[],custom=[]}"
{"message": "373 <14>1 2015-12-11T19:48:34.91492+00:00 loggregator d31112b7-7086-422b-8000-000000000000 \"/}

```



## Lab 3: Building a Microservice

### Learning Objectives

By the end of the lab, you will be able to:

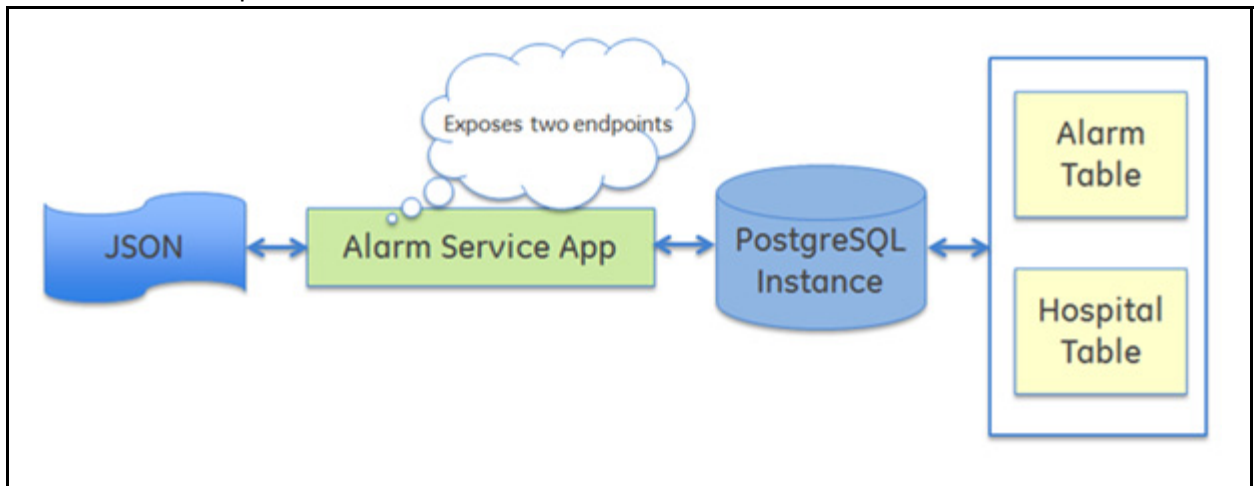
- Use the manifest file to deploy an application
- Build and deploy a Java microservice to the Predix Cloud
- Provide a UI for the microservice

### Lab Exercises

- [Adding a Maven Archetype, page 22](#)
- [Adding an Additional API Endpoint to a Microservice, page 32](#)

### Directions

As part of this lab, you will be building an Alarm Service microservice using a maven archetype. The service exposes two endpoints; one fetches data from the alarm table and the other fetch data from the hospital table.



---

## Exercise 1: Adding a Maven Archetype

---

### Overview

In this exercise you will build a Java microservice (application) and deploy it to the Predix Cloud.

In this lab we're testing an application locally before deploying it. To test locally, we're starting a local postgresQL instance. Once the application is deployed to the Predix Cloud, the local database service is no longer needed.

### Steps

1. Start the PostgreSQL service.

- In the terminal, run this command:

```
sudo /etc/init.d/postgresql-9.3 start
```

A notice that postgresql has started appears.

```
[predix@localhost ~]$ sudo /etc/init.d/postgresql-9.3 start
Starting postgresql-9.3 service:                [ OK ]
[predix@localhost ~]$ █
```

**Note:** Every time you restart the DevBox, the local postgres service is shut down.

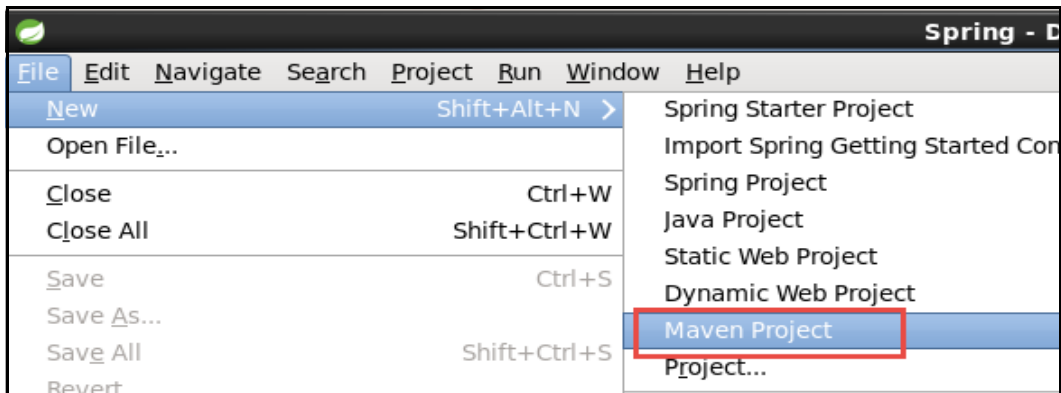
The local postgres service is started because it is required to build the alarmservice project in the Eclipse-STS tool (your next task). This is not the same as the service instance of postgres that you created in Cloud Foundry..

## 2. Create a new maven project

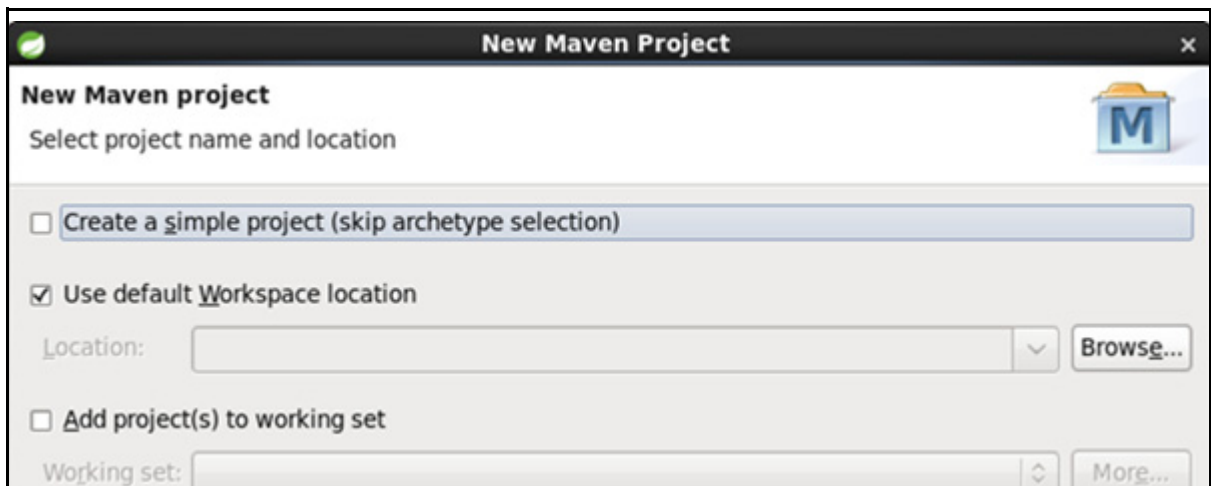
- Open **Eclipse-STS** by double clicking the icon on the desktop



- From the File menu, select *New > Maven Project*



- Accept all defaults on the screen and click **Next**





- In the Select an Archetype window, enter `alarm` in the **Filter** field
- Select `predix-hospital-alarm-service-archetype` (the text highlights in blue)
- Click **Next**

The screenshot shows the 'New Maven project' dialog box in the 'Select an Archetype' step. The 'Catalog' is set to 'All Catalogs'. The 'Filter' field contains the text 'alarm'. Below the filter, a table lists available archetypes. The first entry is highlighted in blue:

Group Id	Artifact Id	Version
com.ge.predix.solsvc.training	predix-hospital-alarm-service-archetype	1.0.0

- Enter the archetype parameter as shown below:

The screenshot shows the 'New Maven project' dialog box in the 'Specify Archetype parameters' step. The parameters are filled as follows:

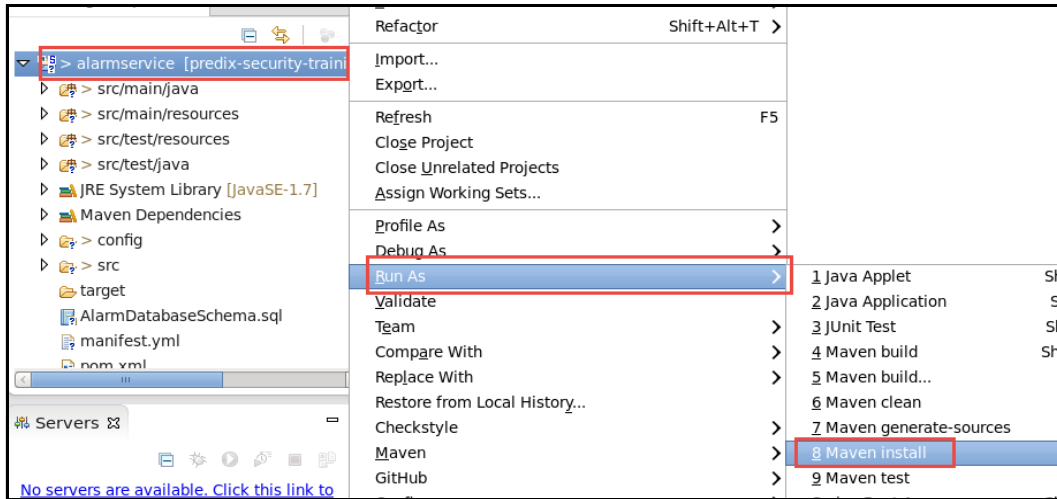
- Group Id: `com.ge.predix.solsvc.training`
- Artifact Id: `alarmservice`
- Version: `0.0.1-SNAPSHOT` (with a dropdown arrow)
- Package: `com.ge.predix.solsvc.training.alarmservice`

Below the parameters, it says 'Properties available from archetype:'.

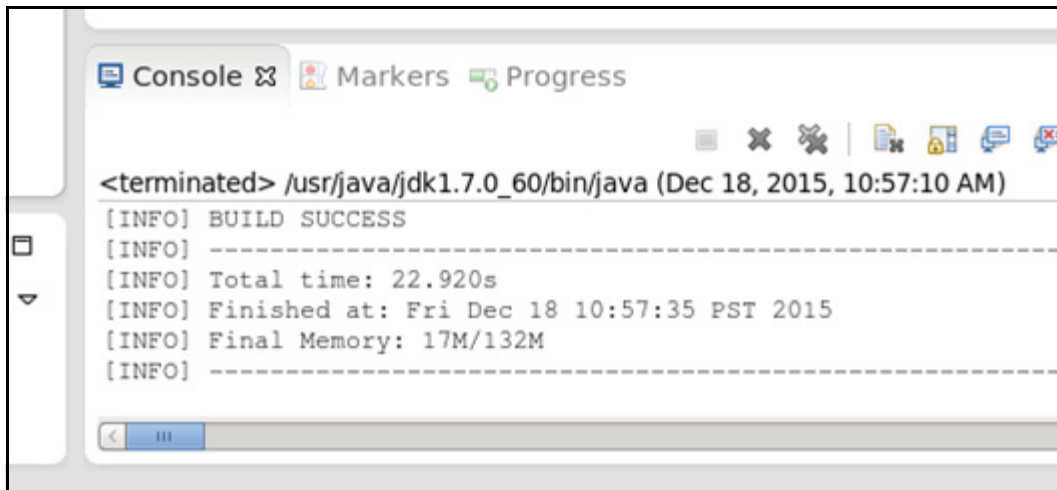
- Click **Finish**

The new alarm service project appears in the Package Explorer.

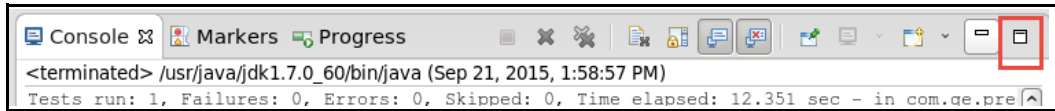
- In Package Explorer, right click on the project root (**alarmservice**) and select *Run As > Maven Install*



At the bottom of the console in the center, the message, “BUILD SUCCESS” appears.



- ◆ Click the maximize console button to maximize the console window



```
<terminated> /usr/java/jdk1.7.0_60/bin/java (Sep 21, 2015, 1:58:57 PM)
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 12.351 sec - in com.ge.pre
```

Notice that the name of the output JAR file is **alarmservice-0.0.1-SNAPSHOT.jar**. This is the output of the build that is deployed to Cloud Foundry.

```
alarmservice ---
ning/machine/alarmservice/target/alarmservice-0.0.1-SNAPSHOT.jar

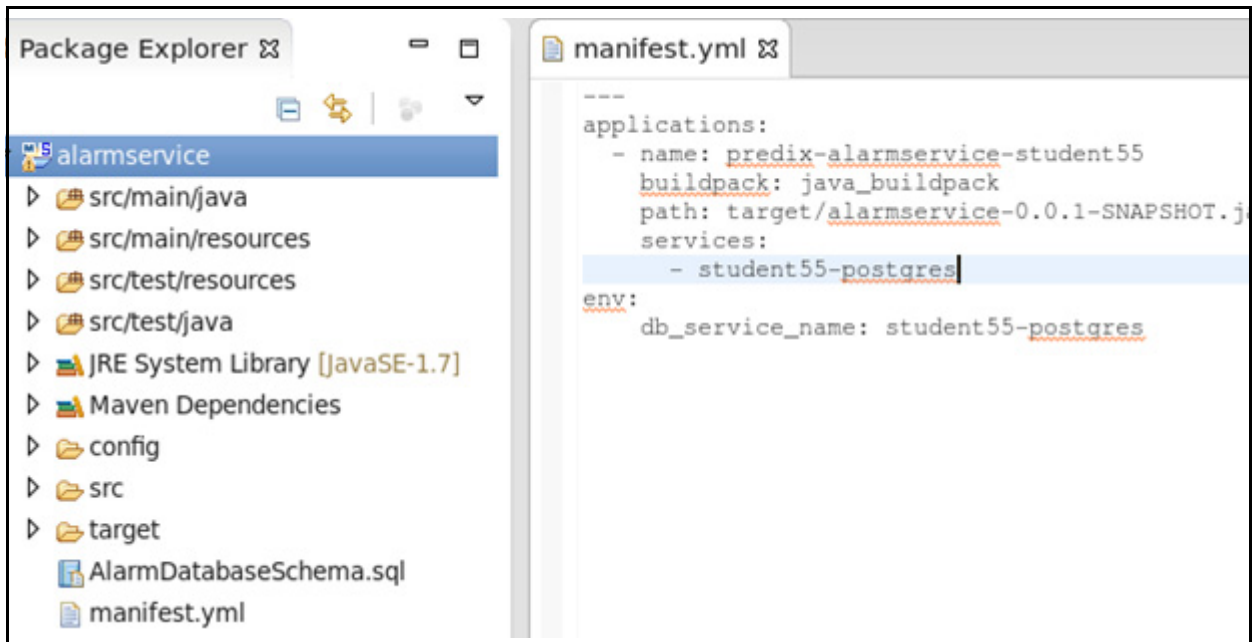
ckage (default) @ alarmservice ---

-install) @ alarmservice ---
g/machine/alarmservice/target/alarmservice-0.0.1-SNAPSHOT.jar to /predix/.m2
g/machine/alarmservice/pom.xml to /predix/.m2/repository/com/ge/predix/solsv
-----
```

### 3. Update the manifest file.

**Note:** Application manifests provide application deployment parameters to Cloud Foundry (how many instances to create, how much memory to allocate, what services applications should use, and so forth).

- In Eclipse, double-click the **manifest.yml** file under the alarmservice project to open it



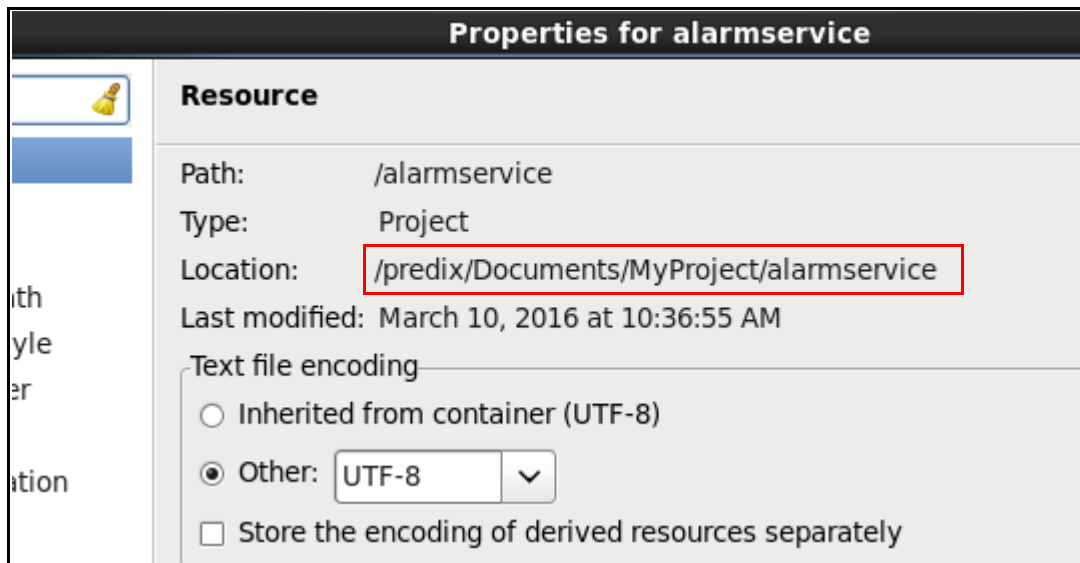
- Update the manifest file
  - ◆ **name:** Append your first and last name to the service name (example: alarm-service-FirstNameLastName)
  - ◆ **path:** Make sure the path is: “target/alarm-service-0.0.1-SNAPSHOT.jar”
  - ◆ **services:** Change the services element to use the name of the postgres service instance you created in Lab 1
  - ◆ **db\_service\_name:** Change the services element to the name of the postgresql service instance you created in Lab 1
  - ◆ Press <ctrl> + <s> to save the file

**To do this:** If you do not remember the name of the service instance you created in Lab 1, follow the steps below to determine the name of your service instance.

- In the Terminal, run the command `cf services`
- Under the name column, find the instance service you created

4. Deploy the microservice to the Predix Cloud.

- In Eclipse, right click on the project root (**alarmservice**)
- Select **Properties** at the bottom of the menu to open the Properties window
- Copy the alarmservice location (select the text, right-click, copy)



- In the Terminal, navigate to the alarmservice directory by running this command:  
`cd <location of alarm service project>`
  - ◆ Type `cd`, then paste the copied path from Eclipse
- Deploy the microservice by running this command:  
`cf push`  
The Terminal shows the deployment steps of the application.
- Once it completes, select the URL, right-click and select **Copy** to copy the URL of your application

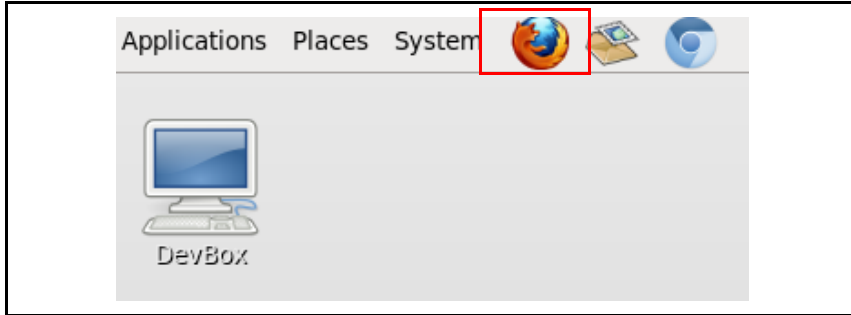
```
Showing health and status for app predix-alarm-service-student55 in org Predix Training / space Training1 as student55...
OK

requested state: started
instances: 1/1
usage: 1G x 1 instances
urls: predix-alarm-service-student55.run.aws-usw02-pr.ice.predix.io
last uploaded: Thu Mar 10 18:51:01 UTC 2016
stack: cflinuxfs2
buildpack: java_buildpack
```

	state	since	cpu	memory	disk
<b>ails</b>					
<b>#0</b>	running	2016-03-10 10:51:48 AM	2.2%	595.6M of 1G	148.4M of 1G

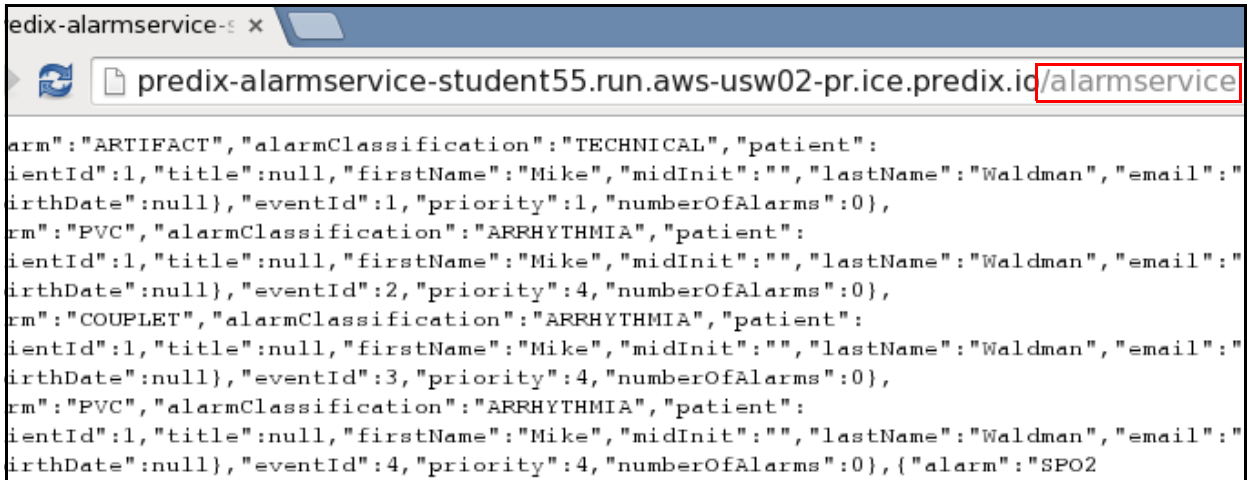
5. Test your application in a browser.

- Open the Firefox browser by clicking the Firefox icon at the top of your DevBox



- Paste the URL you copied from the Terminal into your browser
- Append **"/alarmservice"** to the end of your URL and press **Enter**

The content of the alarm table appears



## 6. View the recent microservice logs.

- In the Terminal, run this command:
  - cf logs <yourMicroserviceName> --recent**
  - ◆ Replace <yourMicroserviceName> with your microservice name

**Tip:** To find your application (microservice) name, follow the instructions below:

- In the Terminal, type **cf a**
- Locate your application name under the name column

```
[predix@localhost alarmservice]$ cf logs predix-alarm-service-student55 --recent
Connected, dumping recent logs for app predix-alarm-service-student55 in org Predix-Training / space Training1 as student55...

2016-03-10T10:47:49.12-0800 [API/2]      OUT Created app with guid 3b02fb43-26e4-4025-9c89-3ed4dacdc2f8
2016-03-10T10:47:52.80-0800 [API/3]      OUT Updated app with guid 3b02fb43-26e4-4025-9c89-3ed4dacdc2f8 ({"route"=>"f61b924c-fe78-43e0-a805-fe89fc77a239"})
2016-03-10T10:50:41.68-0800 [API/2]      OUT Updated app with guid 3b02fb43-26e4-4025-9c89-3ed4dacdc2f8 ({"name"=>"predix-alarm-service-student55", "buildpack"=>"java_buildpack", "environment_json"=>"PRIVATE DATA HIDDEN"})
2016-03-10T10:51:12.22-0800 [DEA/49]     OUT Got staging request for app with id 3b02fb43-26e4-4025-9c89-3ed4dacdc2f8
2016-03-10T10:51:14.06-0800 [STG/49]     OUT -----> Downloaded app package (24M)
2016-03-10T10:51:15.02-0800 [STG/0]     OUT -----> Java Buildpack Version: v3.5
.l | http://github.com/pivotal-cf/pcf-java-buildpack.git#d6c19f8
```

Each log line contains these four fields:

- Time stamp
- Log type (origin code)
- Channel: either STDOUT or STDERR
- Message



## Exercise 2: Adding an Additional API Endpoint to a Microservice

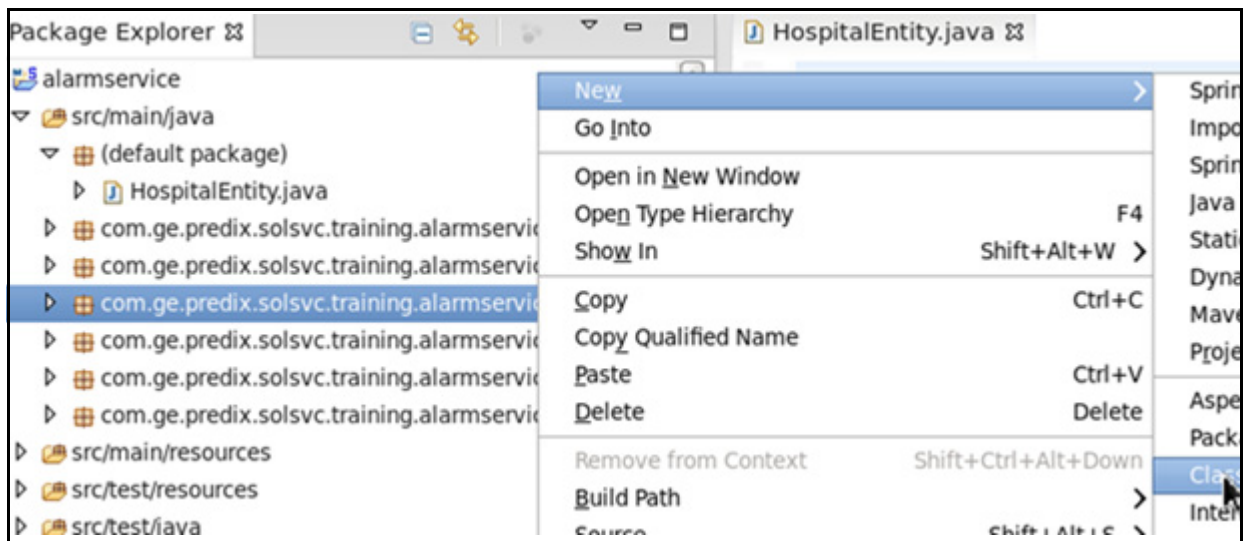
### Overview

In this exercise, you will create another endpoint for the alarmservice microservice. This endpoint will be used to query the hospital table.

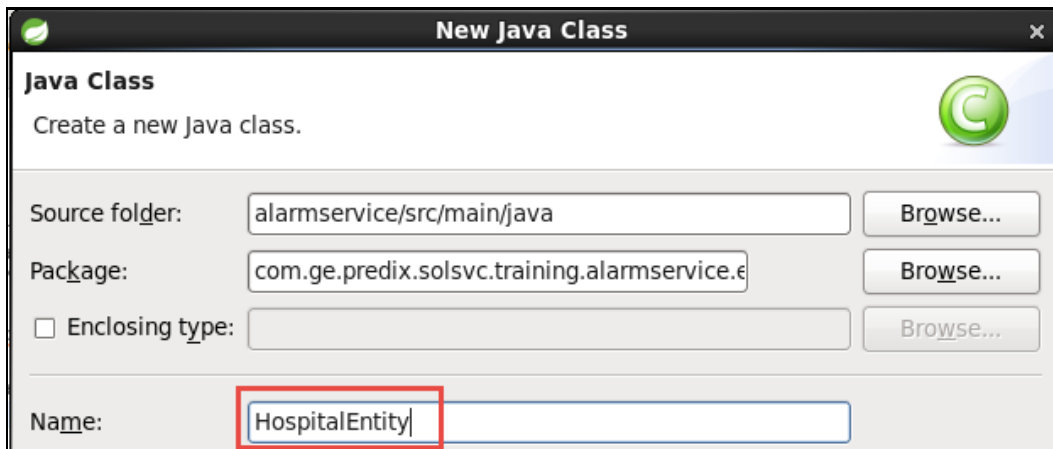
### Steps

1. Create an Entity.

- In Eclipse, under the “/src/main/java” directory, right click on the package **com.ge.predix.solsvc.training.alarmservice.entity**
- Select *New-> Class*

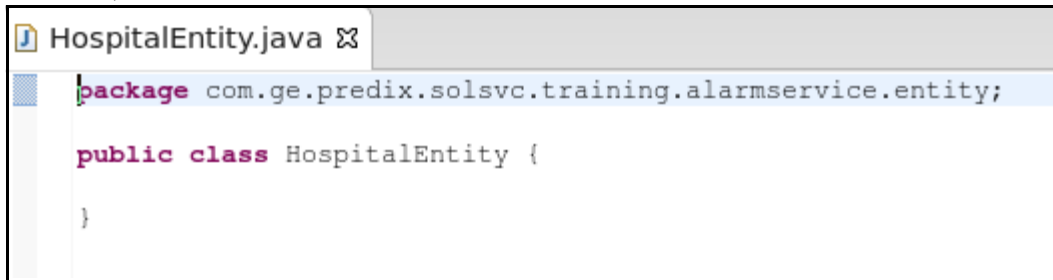


- In the Name field type **HospitalEntity**



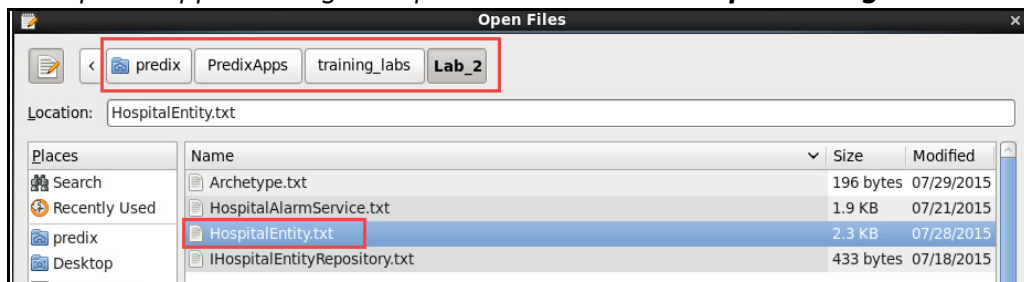
- Click **Finish**

The entity is created



- In gedit, open the file

*/predix/predixApps/training\_labs/fundamentals/Lab2/HospitalEntity.txt*

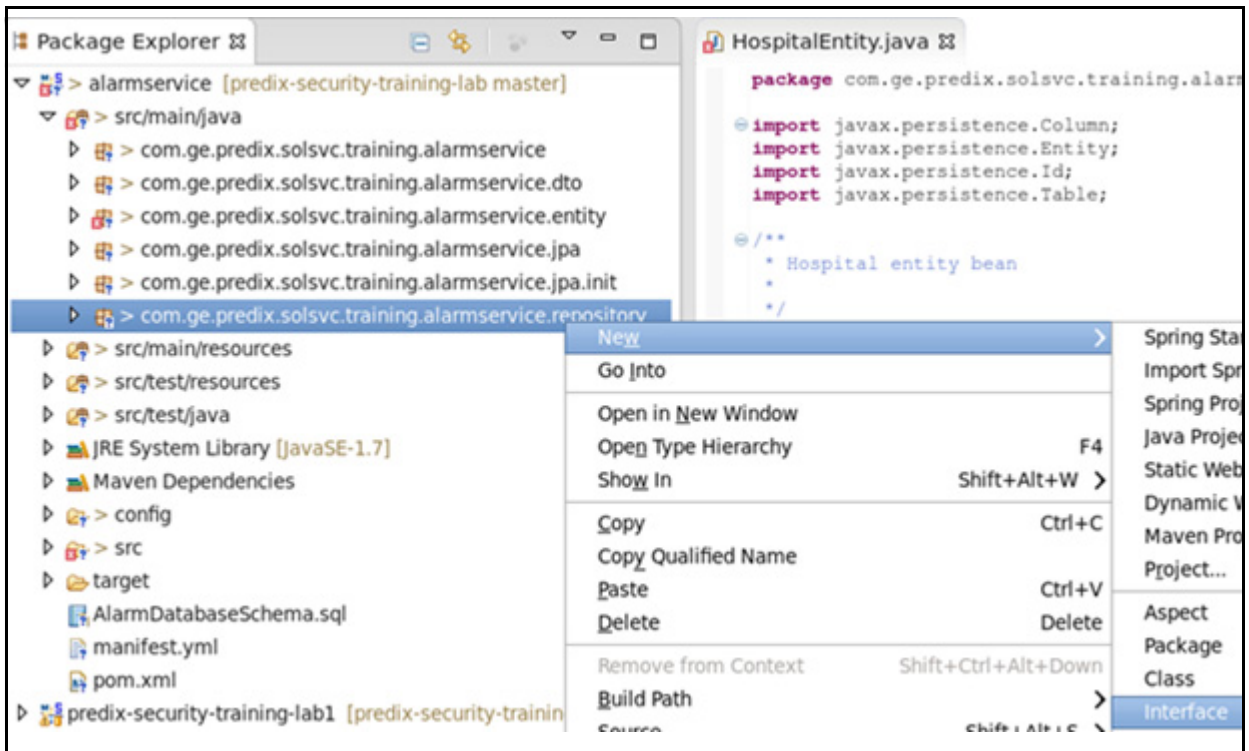


- Copy all content of the file
- In Eclipse, replace everything in your **HospitalEntity.java** class

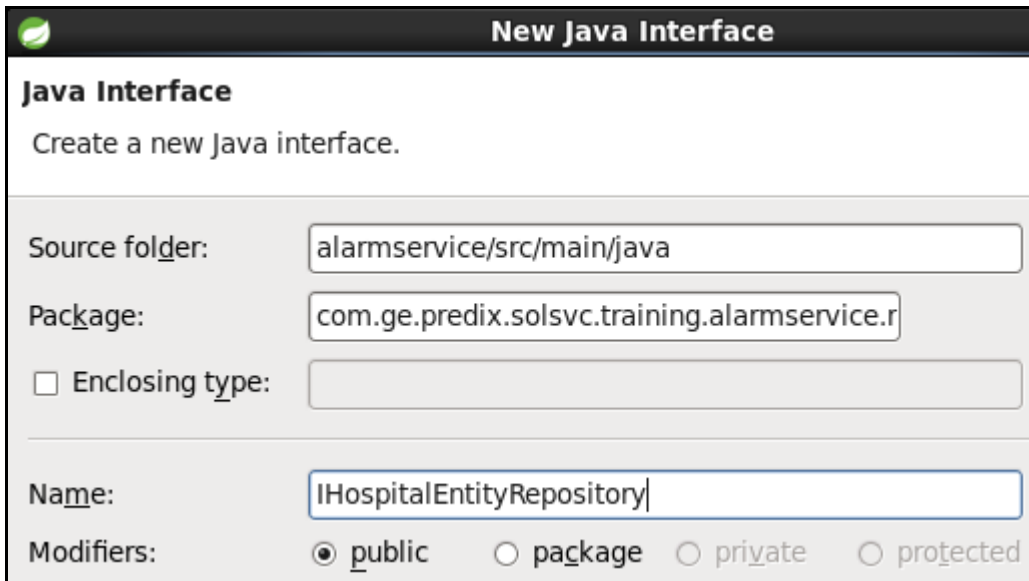
- ◆ Press <ctrl> + <shift> + <o> to Organize Imports
- ◆ Press <ctrl> + <s> to save the file

2. Create an interface.

- In Eclipse, under the “/src/main/java” directory on the package **com.ge.predix.solsvc.training.alarmservice.repository**
- Right-click and select *New -> Interface*



- In the **Name** field, enter **IHospitalEntityRepository**, then click **Finish**



**New Java Interface**

**Java Interface**  
Create a new Java interface.

Source folder:

Package:

Enclosing type:

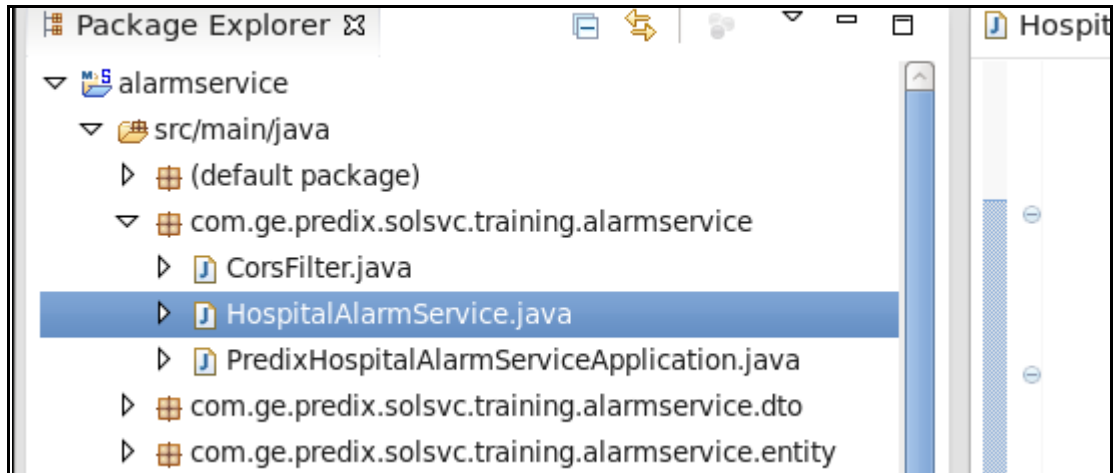
Name:

Modifiers:  public  package  private  protected

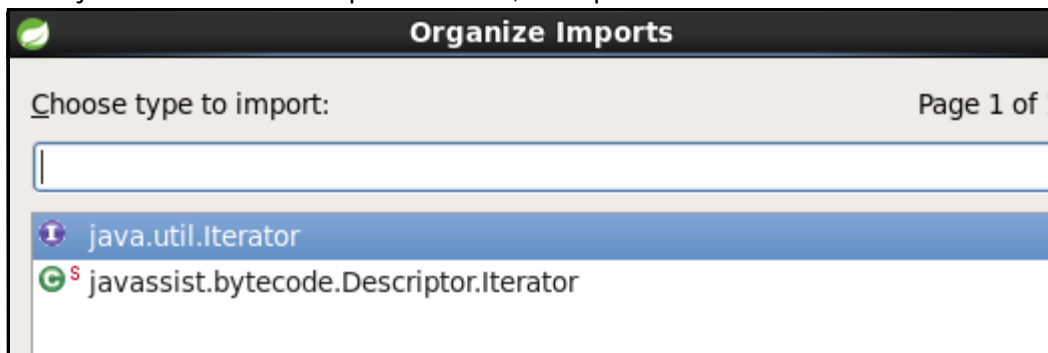
- In gedit, open the file  
`/predix/predixApps/training_labs/fundamentals/Lab_2/IHospitalEntityRepository.txt`
- Copy all content of the file
- In Eclipse, replace everything in your **IHospitalEntityRepository.java** class
- Press `<ctrl> + <shift> + <o>` to Organize Imports
- Press `<ctrl> + <s>` to save the file

### 3. Add mapping to create the service.

- In Eclipse, under the package **com.ge.predix.solsvc.training.alarmservice**, double click **HospitalAlarmService.java** to edit the file



- In gedit, open the file `/predix/predixApps/training_labs/fundamentals/Lab_2/HospitalAlarmService.txt`
- Copy all content of the file
- In Eclipse, replace everything in your **HospitalAlarmService.java** class
- Press `<ctrl> + <s>` to save the file
- Press `<ctrl> + <shift> + <o>` to Organize Imports
- Select `java.util.iterator` and press **Finish**, then press `<ctrl> + <s>` to save the file



#### 4. Populate the Alarm Service and Hospital data.

- In Eclipse, under the package **com.ge.predix.solsvc.training.alarmservice.jp.a.init**, double click **InitAlarmServiceData.java** to edit the file
- Uncomment the use of hospitalRepo by removing `(/*)` and `(*/*)`
- Uncomment the use of HospitalEntity by removing `(/*)` and `(*/*)`

```
@Autowired
private IHospitalEntityRepository hospitalRepo;

@PostConstruct
public void initAlarmServiceData(){
    HospitalEntity he = new HospitalEntity();
    he.setAddress("100, 2305 Camino Ramon, San Ramon, CA 94583");
    he.setPhone("925 234 2345");
    he.setName("John Muir Medical Group");
    he.setEmail("mike.waldman@ge.com");
    hospitalRepo.save(he);
}
```

- Press `<ctrl> + <shift> + <o>` to Organize Imports
- Press `<ctrl> + <s>` to save the file

#### 5. Compile and deploy the microservice.

- In the Terminal, from the alarmservice directory, run this command:  
**mvn clean install**
- The message “BUILD SUCCESS” indicates the microservice was built successfully

```
[INFO] Installing /predix/Documents/PredixApps/tr
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 18.643 s
[INFO] Finished at: 2015-09-24T10:57:55-08:00
[INFO] Final Memory: 29M/392M
```

- From the same directory in the Terminal, run this command:

**cf push**

The message “App started” verifies that the application was successfully deployed.

```
0 of 1 instances running, 1 starting
1 of 1 instances running
```

```
App started
```

```
OK
```

## 6. Test your Alarm Service application (microservice).

- In Firefox, replace “alarmservice” at the end of the URL with “hospital” and refresh the browser

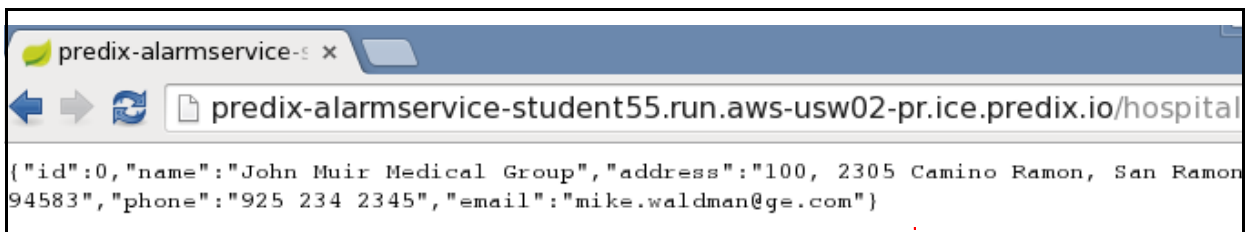
**Tip** - If you do not remember the URL of your alarmservice follow the steps below:

- In the terminal run this command:

**cf a**

- Locate your microservice name under the “name” column and find the URL to the right under the “urls” column

- Your data appears



---

## Exercise 3: Updating the Microservice UI

---

### Overview

In this exercise you will create a UI microservice to display the data fetched by the alarm service. You will use the dashboard seed service pack and modify it to create the UI microservice.

### Steps

1. Import a table design into the project.

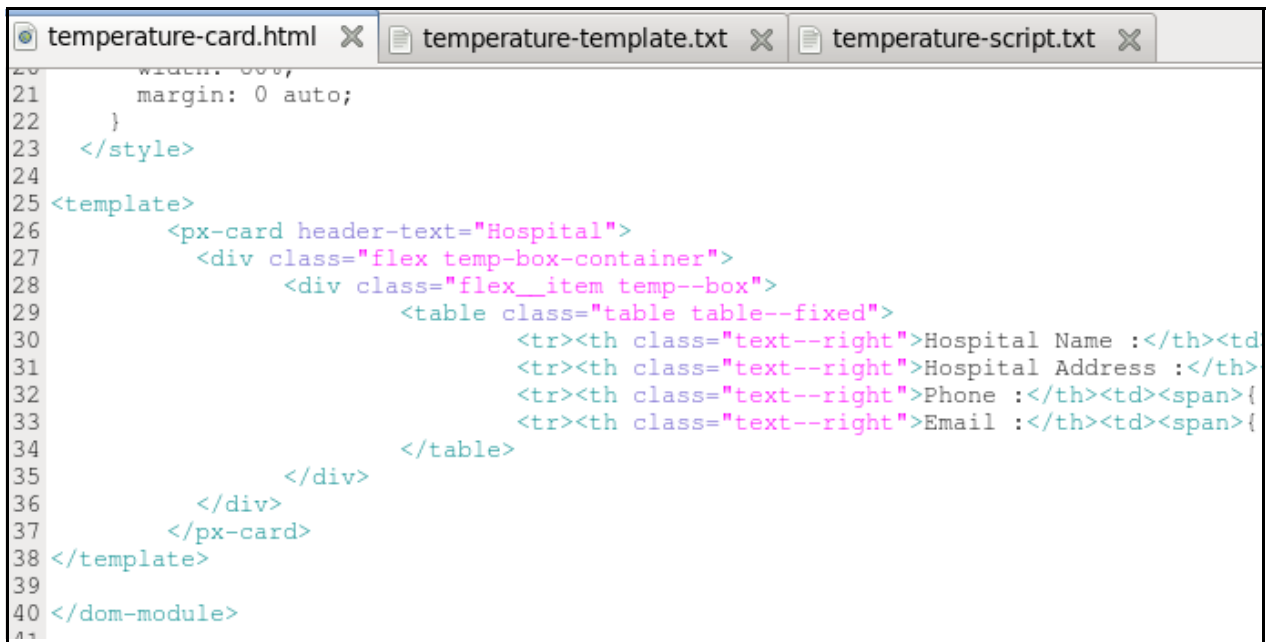
- In gedit, open the `_settings.defaults.scss` file from the `/PredixApps/training_labs/fundamentals/predix-seed-1.1.3/public/bower_components/px-defaults-design/` folder
- Add the following code to the bottom of the file:

```
$inuit-enable-table--fixed:true;  
@import "px-tables-design/_base.tables.scss";
```
- ◆ Press `<ctrl> + <s>` to save the file



## 2. Edit an existing card to display alarm data.

- Open the `temperature-template.txt` file from the `/predix/PredixApps/training_labs/fundamentals/Lab_4/` folder
  - ◆ Copy all contents of the file
- In gedit, open the `temperature-card.html` file in the `/PredixApps/training_labs/fundamentals/predix-seed-1.1.3/public/bower_components/px-sample-cards/` folder
- In your `temperature-card.html` file, replace the `<template>` `</template>` tags and everything within the tags with the content you just copied



```
20     width: 100%;
21     margin: 0 auto;
22 }
23 </style>
24
25 <template>
26     <px-card header-text="Hospital">
27         <div class="flex temp-box-container">
28             <div class="flex__item temp--box">
29                 <table class="table table--fixed">
30                     <tr><th class="text--right">Hospital Name :</th><td>
31                     <tr><th class="text--right">Hospital Address :</th>
32                     <tr><th class="text--right">Phone :</th><td><span>{
33                     <tr><th class="text--right">Email :</th><td><span>{
34                 </table>
35             </div>
36         </div>
37     </px-card>
38 </template>
39
40 </dom-module>
```

- Open the `temperature-script.txt` file under in the `/predix/PredixApps/training_labs/fundamentals/Lab_4` folder
  - ◆ Copy all contents of the file

- In your `temperature-card.html` file, replace the `<script> </script>` tag and everything within the tag with the content you just copied

```
38 </template>
39
40 </dom-module>
41
42 <script>
43   Polymer({
44     is: 'temperature-card',
45     init: function() {
46       this.getTemperature();
47     },
48     getTemperature: function() {
49       /**
50        * use card's getData api to get the temperature data
51        * using the URL from context
52        * see the predix-seed repo /public/scripts/controllers/data-control.js
53        */
54       var self = this;
55       this.getData(this.context.hospitalurl).then(function(data) {
56         // following data structure from http://api.wunderground.com/
57         // alert(data.name);
58         self.hospitalName = data.name;
59         self.hospitalAddress = data.address;
60         self.hospitalPhone = data.phone;
61         self.hospitalEmail = data.email;
62       }, function(reason) {
63         // on rejection
64         console.error('ERROR', reason);
65       });
66     },
67     behaviors: [px.card]
68   });
69 </script>
```

- Press `<ctrl> + <s>` to save the file

**Note:** The highlighted “hospitalurl” will be used later to get data into the card and display it.

### 3. Edit an existing card to display hospital data.

- Open the **fetch-data-card-template.txt** file from `/predix/PredixApps/training_labs/fundamentals/Lab_4` and copy all the contents
- In gedit, open the **fetch-data-card.html** file from the `/PredixApps/training_labs/fundamentals/predix-seed-1.1.3/public/browser_components/px-sample-cards` folder
- In your **fetch-data-card.html** file, replace the `<template>` `</template>` tags and everything within the tags with the content you just copied

```
<template>
  <px-card header-text="Alarms">
    <div class="layout center temperature-box">
      <div class="layout__item">
        <table class="table table--fixed">
          <tr>
            <th>Alarm</th>
            <th>Classification</th>
            <th>Patient First Name</th>
            <th>Patient Last Name</th>
            <th>Patient Email</th>
            <th>Priority</th>
            <th>Number of Alarms</th>
          </tr>
          <template is="dom-repeat" items="{{employees}}">
            <tr>
              <td>{{ item.alarm}}</td>
              <td>{{ item.alarmClassification}}</td>
              <td>{{ item.patient.firstName}}</td>
              <td>{{ item.patient.lastName}}</td>
              <td>{{ item.patient.email}}</td>
              <td>{{ item.priority}}</td>
              <td>{{ item.numberofAlarms}}</td>
            </tr>
          </template>
        </table>
      </div>
    </div>
  </px-card>
</template>
</dom-module>
```

- Open the **fetch-data-card-script.txt** file in the `/predix/PredixApps/training_labs/fundamentals/Lab_4` folder
  - ◆ Copy all contents of the file
  - ◆ In your **fetch-data-card.html** file, replace the `<script>` `</script>` tag and everything within the tag with the content you just copied

```
</dom-module>

<script>

    Polymer({
      is: 'fetch-data-card',
      init: function() {
        this.getTemperature();
      },
      getTemperature: function() {
        /**
         * use card's getData api to get the temperature data
         * using the URL from context
         * see the predix-seed repo /public/scripts/controllers/
         */
        var self = this;
        this.getData(this.context.alarmsurl).then(function(data) {
          // following data structure from http://api.wundergroup.com
          //self.currentTemperature = data['current_observation']
          console.log(data.length+" Records ");
          console.log(JSON.stringify(data));
          self.employees = data;
        }, function(reason) {
          // on rejection
          console.error('ERROR', reason);
          employees = 'error';
        });
      },
      behaviors: [px.card]
    });
</script>
```

- Press `<ctrl> + <s>` to save the file

#### 4. Connect the UI to the microservice.

- Open the `scope.txt` file from the `/predix/PredixApps/training_labs/fundamentals/Lab4` folder
  - ◆ Copy all contents of the file
- In gedit, open the `data-control.js` file from the `/PredixApps/training_labs/fundamentals/predix-seed-1.1.3/public/scripts/controllers` folder
  - ◆ Replace the entire content of the file with the text you just copied

```
$scope.context = {  
  name: 'This is context',  
  // using api from weather underground: http://www.wunderground.com/  
  alarmsurl: 'http://predix-alarmservice-student55.run.aws-usw02-pr.ice.predix.io/alarmservice',  
  hospitalurl: 'http://predix-alarmservice-student55.run.aws-usw02-pr.ice.predix.io/hospital'  
};
```

- Replace the alarm service URL with the URL of your alarm service and save the file

**Tip:** Be sure to leave “http://” at the beginning and the /alarmservice and /hospital at the end when you paste in your replacements.

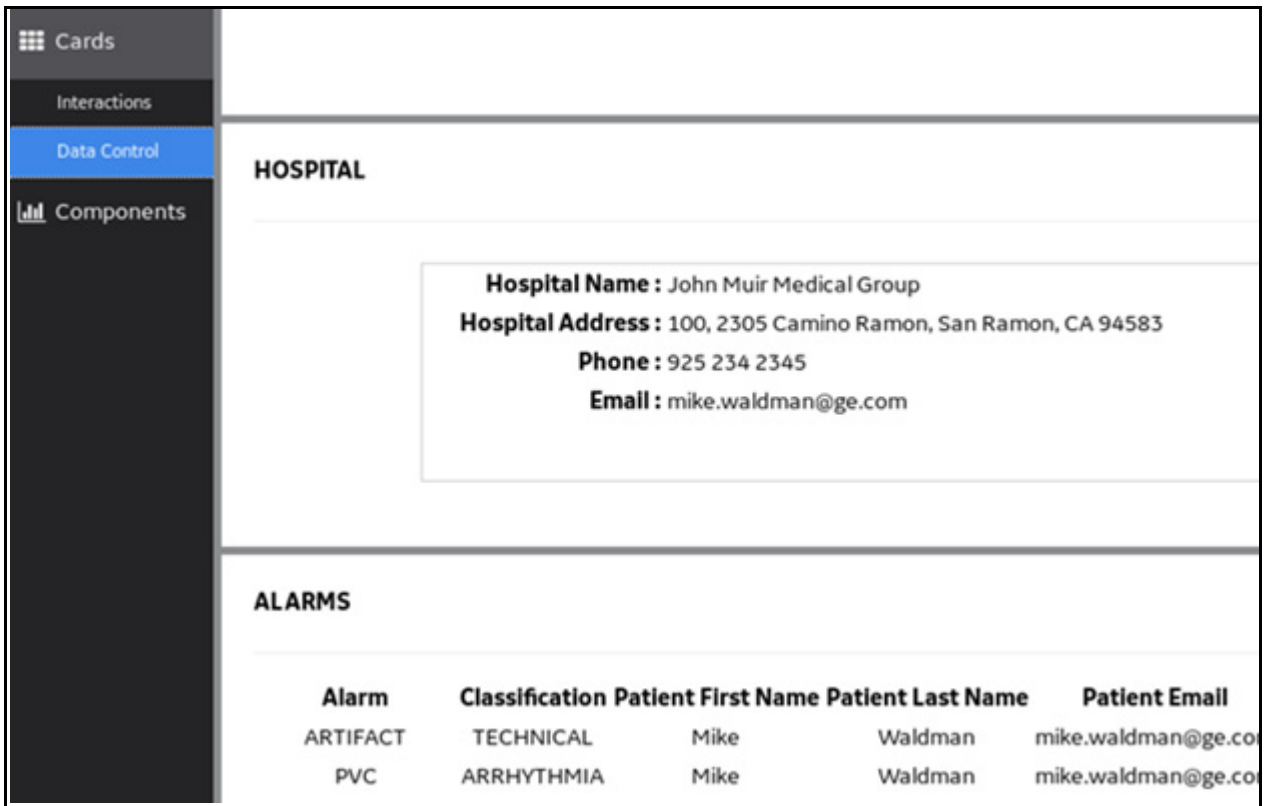
**Tip:** To determine the URL of your alarm service, run the `cf a` command in the terminal. Locate your microservice name and find the URL to the right.

## 5. Test your microservice locally.

- To start your local server, run the **grunt serve** command from the `predix-seed-1.1.3` folder:

Your browser opens the Predix seed application.

- On the left navigation pane select *Cards*, then select *Data Control*
- Verify that the Hospital and Alarm data appear



The screenshot shows the Predix seed application interface. On the left, there is a dark navigation pane with the following items: 'Cards' (selected), 'Interactions', 'Data Control' (highlighted in blue), and 'Components'. The main content area is divided into two sections. The top section is titled 'HOSPITAL' and contains a card with the following information:

- Hospital Name :** John Muir Medical Group
- Hospital Address :** 100, 2305 Camino Ramon, San Ramon, CA 94583
- Phone :** 925 234 2345
- Email :** mike.waldman@ge.com

The bottom section is titled 'ALARMS' and contains a table with the following data:

Alarm	Classification	Patient First Name	Patient Last Name	Patient Email
ARTIFACT	TECHNICAL	Mike	Waldman	mike.waldman@ge.co
PVC	ARRHYTHMIA	Mike	Waldman	mike.waldman@ge.co

- In the terminal, press `<ctrl> + <c>` to stop running your local server

**Note:** The grunt commands allows you to test your microservice locally before deploying it to Cloud Foundry.

## 6. Deploy the microservice to Cloud Foundry.

- Create a package for deployment
  - ◆ In your Terminal, run this command:  
**grunt dist**  
This packages the file.
- Update the manifest file
  - ◆ In gedit, open the file  
*PredixApps/training\_labs/fundamentals/predix-seed-1.1.3/manifest.yml*
  - ◆ Append your name to the microservice name

```
applications:  
- name: predix-seed-dev-student55  
  buildpack: https://github.com/muymoo/staticfile-buildpack.git  
  path: dist  
  memory: 64M  
  stack: cflinuxfs2
```

- Press <ctrl> + <s> to save the file
- Deploy the microservice to Cloud Foundry
  - ◆ In your Terminal run the **cf push** command and verify that the service has started

```
requested state: started  
instances: 1/1  
usage: 64M x 1 instances  
urls: predix-seed-dev-student55.run.aws-usw02-pr.ice.predix.io  
last uploaded: Thu Mar 10 21:37:37 UTC 2016  
stack: cflinuxfs2  
buildpack: https://github.com/muymoo/staticfile-buildpack.git
```

	state	since	cpu	memory	disk
ails					
#0	running	2016-03-10 01:38:06 PM	0.0%	32.6M of 64M	134.9M

```
[predix@localhost predix-seed-1.1.3]$ █
```

7. Test your application (microservice).

- In Firefox, input your Predix-seed microservice URL into the address bar

**Tip:** If you do not remember your microservice URL, follow the steps below:

- In your terminal run this command:  
`cf a`
- Locate your microservice name under the “name” column and find the URL to the right under the “urls” column

- On the left navigation pane select Cards, then select Data Control  
The Hospital and Alarm data appears. You have successfully modified the two cards to display the hospital and alarm data.





## Lab 4: *Implementing Security in Predix*

### Learning Objectives

By the end of the lab, you will be able to:

- Create a UAA Service Instance and bind it to an application
- Add a new OAuth2 client
- Create a user
- Create an ACS instance and bind it to an application
- Update an OAuth2 client to work with ACS
- Manage ACS User Access

### Lab Exercises

- [Create a UAA Service Instance, page 49](#)
- [Fetch a UAA Token, page 52](#)
- [Adding a Client and Users to UAA, page 56](#)
- [Create an ACS Instance, page 64](#)
- [Bind your Application to the ACS instance, page 66](#)
- [Update an OAuth2 Client to Work with ACS, page 68](#)
- [Manage ACS User Access, page 72](#)



## Exercise 1: Create a UAA Service Instance

### Overview

In this exercise you will create an UAA service instance and bind your Spring-Music application to that instance. UAA is the Cloud Foundry service that manages users and OAuth2 clients. It is primarily an OAuth2 provider and issues Java web tokens for client applications to use when the applications act on behalf of users. The UAA instance serves as the trusted issuer of tokens and all access to services is provided by authenticating through this trusted issuer.

### Steps

1. Verify the UAA service is available in the marketplace.

- In the Terminal run the command:

```
cf marketplace
```

- ◆ The **predix-uaa-training\*** service is listed along with its Plan name and description

```
predix-acs-training      Basic, Free      Design precise ac
predix-analytics-catalog Beta, Enterprise* Add analytics to
predix-analytics-runtime Beta, Enterprise* Use this service
predix-asset            Beta, Enterprise* Create and store
predix-mobile           Beta, Enterprise* Design, develop,
predix-timeseries       Beta, Enterprise* Quickly and effic
predix-uaa              Beta, Enterprise* Use this service
predix-uaa-training     Free, Basic      Design precise me
predix-views            Beta, Enterprise* Control layout and
redis-1                 shared-vm        Redis service to
riakcs                  developer        An S3-compatible
* These service plans have an associated cost. Creating a service
```

\*For training only. In production, you will use the predix-uaa service

## 2. Create a UAA service instance.

When you create a UAA service instance, you create the administrative user (client) with its password. The name of this client is “admin”.

- In the Terminal, create a UAA service instance with the following syntax:

```
cf create-service predix-uaa-training <plan> <uaa_instance> -c
'{"adminClientSecret":"<admin_secret>"}
```

- ◆ Replace <plan> with the plan name

**Note:** Use the “Free” plan type to ensure all labs function properly.

- ◆ Replace <uaa\_instance> with an instance name of your choice
- ◆ Replace <admin\_secret> with a password of your choice
- Verify a status of **OK** is returned

Example:

```
[predix@localhost spring-music]$ cf create-service predix-uaa-
training Free cf-uaa -c '{"adminClientSecret":"admin_secret"}'
Creating service instance cf-uaa in org Predix-Training / space
Training3 as student55...
OK
```

- **Write down** the admin password as you will use it in later labs:\_\_\_\_\_

**WARNING:** There is no way to determine this password later on; do not rely on your memory!

## 3. Bind your previously-created Spring-Music application to your UAA service instance.

- In the Terminal run the command:

```
cf bind-service <Spring-Music-app> <uaa_instance>
```

- ◆ Replace <Spring-Music-app> with your Spring-Music application name
- ◆ Replace <uaa\_instance> with name of your instance



- A return status of **OK** indicates the application was successfully bound

```
[predix@localhost spring-music]$ cf bind-service cf-spring cf-uaa
Binding service cf-uaa to app cf-spring in org Predix-Training / space
as student55...
OK
TIP: Use 'cf restage cf-spring' to ensure your env variable changes t
[predix@localhost spring-music]$ █
```

## Exercise 2: Fetch a UAA Token

### Overview

In this exercise you will use the UAA Command Line Interface (UAAC) to fetch a token from the UAA service instance created in the previous lab. The UAAC has been installed on your DevBox.

### Steps

1. Find your sample application name in the space.

- In the Terminal run the command:
 

```
cf a
```
- Locate your application in the list and verify it has started

name	memory	disk	urls	requested state	instances
cf-spring	512M	1G	cf-spring.run.aws-usw02-pr.ice.predix.io	started	1/1

2. Retrieve your UAA instance details from the VCAP\_SERVICES environment variable.

When you bind your app to the UAA instance, the connection details for your UAA instance are populated in VCAP\_SVCS environment variables.

- In the Terminal, run the command:
 

```
cf env <app_name>
```
- In the **VCAP\_SERVICES** variable, locate the entry for your UAA service instance
- Copy the UAA environment variables to a new file in your text editor for later use

**Note:** Look for the name of your service instance (example: `cf-uaa`)

- Under the *credentials* section, copy the **uri** value for your service instance (copy the string between the quotes)

```
,  
],  
"predix-uaa-training": [  
  {  
    "credentials": {  
      "issuerId": "https://c758a891-d2d0-4d19-a2a3-450496f8557e.predix-uaa-training.run.aws-usw02-pr.ice.predix.io/oauth/token",  
      "uri": "https://c758a891-d2d0-4d19-a2a3-450496f8557e.predix-uaa-training.run.aws-usw02-pr.ice.predix.io",  
      "zone": {  
        "http-header-name": "X-Identity-Zone-Id",  
        "http-header-value": "c758a891-d2d0-4d19-a2a3-450496f8557e"  
      }  
    },  
    "label": "predix-uaa-training",  
    "name": "cf-uaa",  
    "plan": "Free",  
    "tags": []  
  }  
]
```

### 3. Specify your UAA instance as the intended target.

**Note:** You must first point the `uaac` CLI tool to your UAA service instance in order to view the UAA instance details.

- In the Terminal run the command:
  - `uaac target <uri>`
  - ◆ Replace `<uri>` with the uri of your UAA instance

```
[predix@localhost ~]$ uaac target https://c758a891-d2d0-4d19-a2a3-450496f8557e.predix-uaa-training.run.aws-usw02-pr.ice.predix.io

Target: https://c758a891-d2d0-4d19-a2a3-450496f8557e.predix-uaa-training.run.aws-usw02-pr.ice.predix.io
```

### 4. Fetch a UAA token from your UAA instance.

- In the Terminal run the command to log in using the administrative client:
  - `uaac token client get admin -s <admin_secret>`
  - Note:** The `admin` client is the default client id that has all permissions
- Replace `<admin_secret>` with the password you created when creating the service instance
  - A successful fetch notice indicates you have retrieved the token

```
[predix@localhost ~]$ uaac token client get admin -s admin_secret

Successfully fetched token via client credentials grant.
Target: https://c758a891-d2d0-4d19-a2a3-450496f8557e.predix-uaa-training.run.aws-usw02-pr.ice.predix.io
Context: admin, from client admin
```

- You are now logged in as the administrative client

5. Decrypt the token to view its contents.

- In the Terminal run the command:

**uaac token decode**

```
[predix@localhost ~]$ uaac token decode
Note: no key given to validate token signature

jti: 7ccd0272-e849-4e6d-bcb0-055dcf6503fc
sub: admin
scope: clients.read
       zones.c758a891-d2d0-4d19-a2a3-450496f8557e.admin
       clients.secret idps.write uaa.resource clients.write
       clients.admin idps.read scim.write scim.read
client_id: admin
cid: admin
azp: admin
grant_type: client_credentials
rev_sig: 2158b991
iat: 1458334971
exp: 1458378171
iss: https://c758a891-d2d0-4d19-a2a3-450496f8557e.predix-uaa-training.run.aws-usw02-pr.ice.predix.io/oauth/token
zid: c758a891-d2d0-4d19-a2a3-450496f8557e
aud: admin clients zones.c758a891-d2d0-4d19-a2a3-450496f8557e idps
     uaa scim
```

- The token contains basic information including
  - ◆ scope - a list of authorities for the admin client
  - ◆ client\_id - unique name to the system for the client id
  - ◆ Authorization grant type - mode of authorization



---

## Exercise 3: Adding a Client and Users to UAA

---

### Overview

In this exercise you will create an OAuth2 client you use to create users for in your UAA instance. When you create a UAA service instance, a default administrator account (admin client) is automatically generated that contains *all* permissions. As a best practice, you create an OAuth2 client and define the scopes instead of using the admin client to create users.

For service-to-service security, the generally-recommended grant type is “client\_credentials”. for web applications, the recommended grant type is “authorization\_code”

### Steps

1. Create an OAuth2 client with a subset of admin permissions.

- In the Terminal, run the command:

```
uaac client add -h
```

This displays all parameters available to create a new OAuth client

- To create the OAuth 2 client, the command line syntax is:

```
uaac client add <client_name> -s <client_secret>
--authorized_grant_types "<grant_types>" --autoapprove openid
--authorities "<list of client scopes>"
```

- ◆ Replace **<client\_name>** with your client name and write it here \_\_\_\_\_
- ◆ Replace **<client\_secret>** with your chosen password and write it here\_\_\_\_\_
- ◆ Replace **<grant\_types>** with **authorization\_code client\_credentials refresh\_token**
- ◆ Replace **<list of client scopes>** with **clients.read clients.write scim.read scim.write**



*Example:*

```
[predix@localhost ~]$ uaac client add cf-client -s client_secret --a
uthorized_grant_types "authorization_code client_credentials refresh
_token" --autoapprove openid --authorities "clients.read clients.wri
te scim.read scim.write"

scope: uaa.none
client_id: cf-client
resource_ids: none
authorized_grant_types: authorization_code client_credentials
    refresh_token
autoapprove:
action: none
authorities: clients.read clients.write scim.write scim.read
lastmodified: 1458336998687
id: cf-client
```

## 2. Fetch a token for the *new* OAuth2 client.

- In the Terminal, run the command:  
`uaac token client get <client_name> -s <client_secret>`
  - ◆ Replace `<client_name>` with the name of your new oauth2 client
  - ◆ Replace `<client_secret>` with the password you created when creating the uaa client
- Verify the token was fetched successfully - you are now logged in as the new client

```
[predix@localhost ~]$ uaac token client get cf-client -s client_secret  
et  
  
Successfully fetched token via client credentials grant.  
Target: https://c758a891-d2d0-4d19-a2a3-450496f8557e.predix-uaa-training.run.aws-usw02-pr.ice.predix.io  
Context: cf-client, from client cf-client
```

Alternatively, you can exclude the `-s` portion of the command, and you will be prompted to enter the client secret. This is more secure since the password is not displayed as you login as *the* new client.

### 3. Add a user to your UAA service instance.

**Analysis:** For applications accessing your UAA instance, you can create additional users with required scopes. You must be logged in as a client with the necessary authorities.

- In the Terminal, run the command:

```
uaac user add -h
```

This displays all parameters available for creating a new user

- In the Terminal run the command:

```
uaac user add <user_name> --emails <user_email> -p <user_password>
```

- ◆ Replace `<user_name>` with a value of your choice, preferably the email
- ◆ Replace `<user_email>` with the email of the user (e.g. `user1@test.com`)
- ◆ Replace `<user_password>` with a value of your choice

```
[predix@localhost ~]$ uaac user add user1@gmail.com --emails user1@gmail.com -p us3r1
user account successfully added
```

- ◆ Write down the user name: \_\_\_\_\_
- ◆ Write down the email: \_\_\_\_\_
- ◆ Write down the user password: \_\_\_\_\_

You will use these later on in the lab.

4. Create the groups in your UAA instance.

- Give your user read and write privileges (group names are scim.read and scim.write - the user token (permission) must have the same name as the group).

```
uaac group add scim.read
```

```
uaac group add scim.write
```

*Sample code:*

```
[predix@localhost ~]$ uaac group add scim.read
meta
  version: 0
  created: 2016-03-18T21:43:02.926Z
  lastmodified: 2016-03-18T21:43:02.926Z
  schemas: urn:scim:schemas:core:1.0
  id: bfd7b491-e2c5-47ea-899c-eb826c47499d
  displayname: scim.read
[predix@localhost ~]$ uaac group add scim.write
meta
  version: 0
  created: 2016-03-18T21:43:10.476Z
  lastmodified: 2016-03-18T21:43:10.476Z
  schemas: urn:scim:schemas:core:1.0
  id: 16b2b2cd-e663-4e5a-9ef6-68387d55d892
  displayname: scim.write
```

5. Add the new user to the required groups.

```
uaac member add scim.read <user_name>
```

```
uaac member add scim.write <user_name>
```

```
[predix@localhost ~]$ uaac member add scim.read user1@gmail.com
success
[predix@localhost ~]$ uaac member add scim.write user1@gmail.com
success
```



## 6. View user details.uaa

- To view all users, run the command:

**uaac users**

The user now has access to the UAA zone

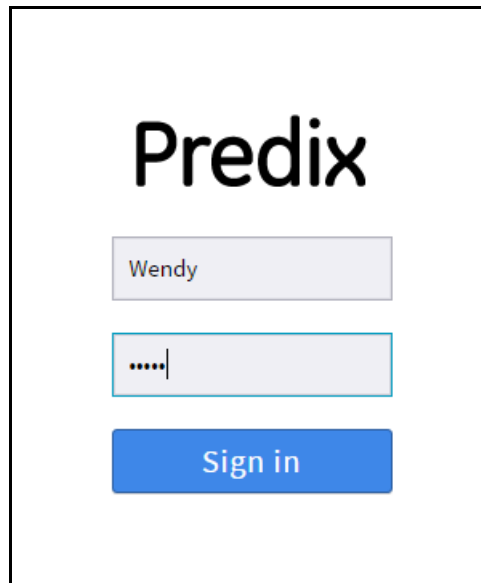
```
[predix@localhost ~]$ uaac users
resources:
-
  id: 80f03846-5fa5-413b-a6b3-a3e7fa212055
  meta
    version: 0
    created: 2016-03-18T21:41:29.477Z
    lastmodified: 2016-03-18T21:41:29.477Z
  name
  emails:
  -
    value: user1@gmail.com
    primary: false
  groups:
  -
    value: 16b2b2cd-e663-4e5a-9ef6-68387d55d892
    display: scim.write
    type: DIRECT
  -
    value: bfd7b491-e2c5-47ea-899c-eb826c47499d
    display: scim.read
    type: DIRECT
  approvals:
  active: true
  verified: false
  origin: uaa
  schemas: urn:scim:schemas:core:1.0
  username: user1@gmail.com
  zoneid: c758a891-d2d0-4d19-a2a3-450496f8557e
  passwordlastmodified: 2016-03-18T21:41:29.000Z
schemas: urn:scim:schemas:core:1.0
```

## 7. View the UAA login page.

To view the page your user should see when they try to log into an application, paste the uri of your UAA instance into a browser and append `/login` at the end

<https://03fffb3a-da91-4e1b-8a77-dcc131317176.predix-uaa-training.run.aws-usw02-pr.ice.predix.io/login>

- At the Predix login screen, enter the **user** name (**not** email) and user password
- Click **Sign in**



- The login is successful, but an error message is displayed

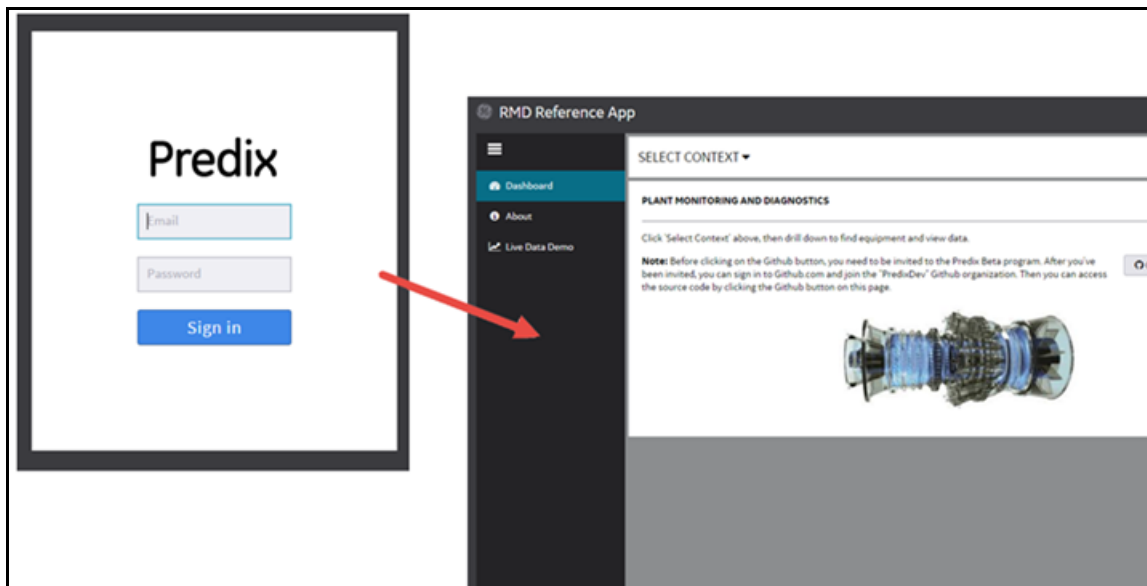
You should not see this page.  
Set up your redirect URI.

```

client add [name]
--scope <list>
--authorized_grant_types <list>
--authorities <list>
--access_token_validity <seconds>
--refresh_token_validity <seconds>
--redirect_uri <list>
--autoapprove <list>
--signup_redirect_url <url>
--clone <other>, get default
settings from other
-s | --secret <secret>, client
secret
-i | --[no-]interactive,
interactively verify all values
    
```

**Note:** When creating the new client, we did not specify a redirect attribute. You can add the redirect by updating the client.

If the redirect\_uri had been set you would have been directed to your application.





---

## Exercise 4: Create an ACS Instance

---

### Overview

In this exercise you will create an ACS instance, and bind your application to the instance.

### Steps

1. Verify that the UAA service is available in the marketplace.

- In the Terminal run the command:

```
cf marketplace
```

The **predix-ac-training** service is listed along with its Plan name and description

service	plans	description
logstash	free	Logstash 1.4 serv
logstash-5	free	Logstash 1.4 serv
p-rabbitmq	standard	RabbitMQ is a rob
p-rabbitmq-35	standard	RabbitMQ is a rob
postgres	shared, shared-nr	Reliable PostgrSQ
predix-ac	Beta, Enterprise*	Use this service
predix-ac-training	Basic, Free	Design precise ad
predix-analytics-catalog	Beta, Enterprise*	Add analytics to
predix-analytics-runtime	Beta, Enterprise*	Use this service
predix-asset	Tiered	Create and store
predix-mobile	Beta, Enterprise*	Design, develop,

■

## 2. Create an ACS service instance.

**Note:** You will use the training version of ACS for this exercise rather than the production version.

- In the Terminal run the command to create a service instance with the following syntax:
 

```
cf create-service predix-acs-training <plan> <my_acs_instance> -c '{"trustedIssuerIds":"<uaa_instance_issuerID>"}'
```

  - ◆ Replace <plan> with the plan name (e.g. Tiered, Basic, Free)
  - ◆ Replace <my\_acs\_instance> with an instance name of your choice
  - ◆ Replace <uaa\_instance\_issuerID> is the **issuerID** (not the uri) of your UAA instance that you saved when you created your UAA instance.
  - ◆ Verify a status of OK is returned

*Example:*

```
[predix@localhost ~]$ cf create-service predix-acs-training Free cf-acs -c '{"trustedIssuerIds":"https://c758a891-d2d0-4d19-a2a3-450496f8557e.predix-uaa-training.run.aws-usw02-pr.ice.predix.io/oauth/token"}'
Creating service instance cf-acs in org Predix-Training / space Training3 as student55...
OK
```

---

## Exercise 5: Bind your Application to the ACS instance

---

### Overview

You must bind your application to your ACS instance to provision its connection details in the VCAP\_SERVICES environment variable.

### Steps

1. Bind your application to the new ACS instance.

- In the Terminal execute the command:

```
cf bind-service <Spring-Music-app> <my_acs_instance>
```

- ◆ Replace **<Spring-Music-app>** with **your application name**
- ◆ Replace **<my\_acs\_instance>** with your ACS instance name

*Example:*

```
[predix@localhost ~]$ cf bind-service cf-spring cf-acs
Binding service cf-acs to app cf-spring in org Predix-Training / spa
ce Training3 as student55...
OK
TIP: Use 'cf restage cf-spring' to ensure your env variable changes
take effect
-
```

- Verify the binding:

```
cf env <your_app_name>
```

```

"predix-ac-training": [
  {
    "credentials": {
      "uri": "https://predix-ac-training.run.aws-usw02-pr.ice.predix
io",
      "zone": {
        "http-header-name": "Predix-Zone-Id",
        "http-header-value": "6a5a8113-2b9b-4f2d-9343-bae2d023dcff",
        "oauth-scope": "predix-ac-training.zones.6a5a8113-2b9b-4f2d-9
43-bae2d023dcff.user"
      }
    },
    "label": "predix-ac-training",
    "name": "cf-ac",
    "plan": "Free",
    "tags": []
  }
]

```

Notice the label is **predix-ac-training** and the name is **cf-ac**

- Copy the ACS information into your text editor file. You need the OAuth scope to configure the OAuth client to work with ACS.

---

## Exercise 6: Update an OAuth2 Client to Work with ACS

---

### Overview

To enable applications to manage policies and attributes using ACS, you need to update your OAuth2 client with the required OAuth2 scopes and authorities. In this exercise, you will establish your OAuth2 client to handle Policy Management Services. This will handle tokens sent by the application or client.

### Steps

1. Update your Client to work with ACS

- Login as admin to make these changes. Run the command:

```
uaac token client get admin
```

Specify the admin password (<admin\_secret>) when prompted

```
[predix@localhost ~]$ uaac token client get admin
Client secret:  *****

Successfully fetched token via client credentials grant.
Target: https://03fffb3a-da91-4e1b-8a77-dcc131317176.predix-uaa-t
Context: admin, from client admin
```

- Run the command:

**uaac clients**

```
[predix@localhost ~]$ uaac clients
admin
  scope: uaa.none
  resource_ids: none
  authorized_grant_types: client_credentials
  autoapprove:
  action: none
  authorities: clients.read
    zones.c758a891-d2d0-4d19-a2a3-450496f8557e.admin
    clients.secret idps.write uaa.resource clients.write
    clients.admin idps.read scim.write scim.read
  lastmodified: 1458333539642
cf-client
  scope: uaa.none
  resource_ids: none
  authorized_grant_types: authorization_code client_credentials
    refresh_token
  autoapprove:
  action: none
  authorities: clients.read clients.write scim.write scim.read
  lastmodified: 1458336998687
```

Note the client authorities in both the admin client and the client you will use for your application (cf-client in this case).

- Build the `uaac client update --authorities` command so that the OAuth client has the appropriate authorities to work with your ACS service instance
  - ◆ Open a new file in your gedit text editor and enter the update command
  - ◆ Enter a space and double quote after the `--authorities` flag
  - ◆ List all the authorities (single space in between), including those that the client already has
    - Original list, from your client environment variables (see prior page graphic):  
`clients.read clients.write scim.write scim.read`
    - Enter a space, then enter the `zones.<UAA_uri>.admin` variable listed under the admin authorities (see prior page graphic)
    - Enter a space, then enter the required ACS scopes:  
`acs.policies.read acs.policies.write acs.attributes.read acs.attributes.write`  
`<oauth-scope>` (from the application environment variable for ACS)
  - ◆ End your command with a double quote

Example:

```

File Edit View Search Terminal Help

{predix@localhost ~}$ uaac client update --authorities
"clients.read clients.write scim.write scim.read ←Original
zones.ad6e23ef-079a-4086-8d75-65c8bc0e7c38.admin ←Copy from Admin authorities
acs.policies.read acs.policies.write acs.attributes.read
acs.attributes.write ←ACS additions
predix-acs-training.zones.ccff1702-ef11-4769-a527-deade4c917b0.user"
                                     ↑
                                     Copy from app env oauth-scope variable
  
```

- ◆ When prompted, enter the client name you created

*Sample results for updating the OAuth2 client:*

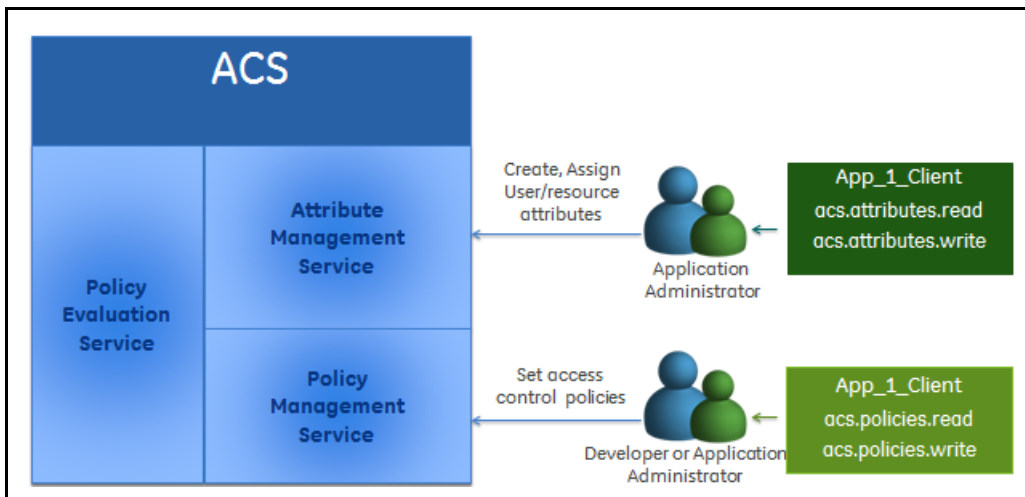
```
[predix@localhost ~]$ uaac client update --authorities "clients.read
clients.write scim.write scim.read zones.c758a891-d2d0-4d19-a2a3-45
0496f8557e.admin acs.policies.read acs.policies.write acs.attributes
.read acs.attributes.write predix-acs-training.zones.6a5a8113-2b9b-4
f2d-9343-bae2d023dcff.user"
Client name: cf-client
  scope: uaa.none
  client_id: cf-client
  resource_ids: none
  authorized_grant_types: authorization_code client_credentials
    refresh_token
  autoapprove:
  action: none
  authorities: clients.read acs.policies.read acs.policies.write
    predix-acs-training.zones.6a5a8113-2b9b-4f2d-9343-bae2d023dcff
    .user zones.c758a891-d2d0-4d19-a2a3-450496f8557e.admin
    acs.attributes.read clients.write acs.attributes.write
    scim.write scim.read
  lastmodified: 1458342001662
```



## Exercise 7: Manage ACS User Access

### Overview

Your client will manage the resources for Attribute Management Services, and the account and permissions will need to be set up. You created a user in a previous lab. When your user logs in and creates a report, the user will need read permissions.



## Steps

1. Create the groups required for ACS in UAA.

You can use Admin to create groups, although as a best practice, you should use your application client to manage policies and attributes in your application. You updated your application client in Exercise 3 to enable that client to manage ACS attributes and policies.

- Ensure your UAA instance is the intended target

```
uaac target <uri>
```

- Logon as the administrative client

```
uaac token client get <app client>
```

Specify the `<client_secret>` when prompted

- Create the groups required for ACS in UAA:

```
uaac group add acs.policies.read
```

```
uaac group add acs.policies.write
```

```
uaac group add acs.attributes.read
```

```
uaac group add acs.attributes.write
```

```
uaac group add <acs_oauth_scope>
```

Where `<acs_oauth_scope>` is the ACS oauth-scope environment variable

```
[predix@localhost ~]$ uaac group add predix-ac-training.zones.6a5a8113-2b9b-4f2d-9343-bae2d023dcff.user
meta
  version: 0
  created: 2016-03-18T23:06:59.470Z
  lastmodified: 2016-03-18T23:06:59.470Z
  schemas: urn:scim:schemas:core:1.0
  id: 6762e45c-cde7-4345-8a26-e9c7034e5315
  displayname: predix-ac-training.zones.6a5a8113-2b9b-4f2d-9343-bae2d023dcff.user
```

2. To use the Policy Management service the user/client must authenticate using a JSON Web Token (JWT) bearer token that includes the `acs.policies.read` scope for reading the policies or the `acs.policies.write` scope for writing the policies. To use the Attribute Management service, the user/client must authenticate using a JWT that includes the `acs.attributes.read` and the `acs.attributes.write` scopes.

- Use the `uaac users` command to find the user created previously
- Assign membership to the required scope:

```
uaac member add acs.policies.read <user_name>
uaac member add acs.policies.write <user_name>
uaac member add acs.attributes.read <user_name>
uaac member add acs.attributes.write <user_name>
uaac member add <acs_oauth_scope> <user_name>
```

You will see a message indicating you have succeeded assigning membership to your user.

The user is now able to authenticate through ACS Policy Management, Attribute Management, and Policy Evaluation services.





## Lab 5: UI Basics

### Learning Objectives

By the end of the lab, students will be able to:

- Add a link and route for the Patients page to the Predix Starter Pack
- Show data in a table
- Customize the px-theme component to style the application
- Customize a reusable Predix web component
- Use the customized component in your application
- Fetch data from a Polymer web component

### Lab Exercises

- [Adding a Route using Angular JS, page 77](#)
- [Creating a Controller, page 82](#)
- [Changing the View and Model, page 84](#)
- [Styling Your Application, page 87](#)
- [Creating a View to Display a Web Component, page 93](#)
- [Creating a Web Component, page 96](#)
- [Connecting a Microservice, page 102](#)

### Directions

Complete the exercises that follow.

**Note:** The code for these exercises is found in the **UILab.txt** file. Your instructor will provide you with the path to this file.

---

## Exercise 1: Adding a Route using Angular JS

---

### Overview

In this exercise you will start up the predix seed web application and add a new page to the application, using the Predix Angular UI-router. (This is not the built-in router Angular JS ships with.) You will add a new Angular controller and use it to add data to the page.

### Steps

**Note:** The Predix seed application is already installed, along with its required npm and bower installations. Normally, you would run npm install and bower installs after saving the seed application files to your directory.

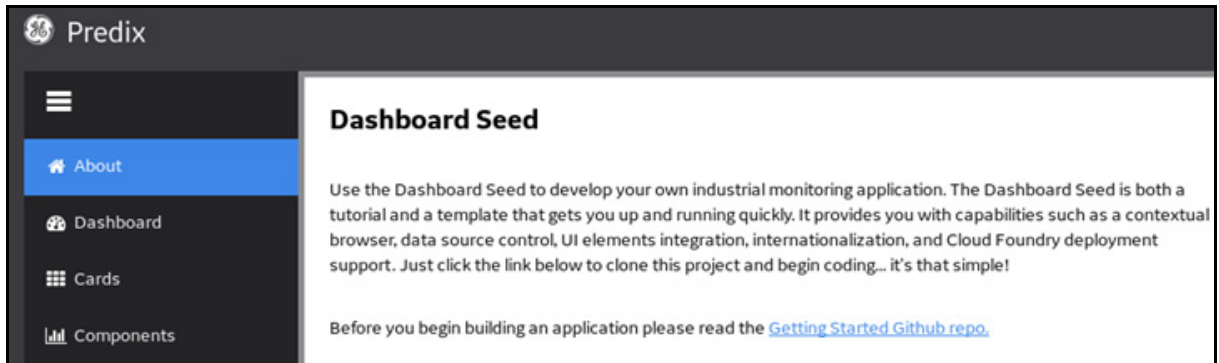
#### 1. Test the application locally.

- In the Terminal, use the following command to navigate to the predix seed app:
  - ◆ `cd ~/PredixApps/training_labs/fundamentals/predix-seed-1.1.3`
- Run the following command to start the local web server
  - ◆ `grunt serve`

The command line interface (CLI) responds with several lines, ending as follows.

```
Running "clean:build" (clean) task
Running "connect:livereload" (connect) task
Started connect web server on http://localhost:9000
Running "watch" task
Waiting...
█
```

The web browser opens and loads the predix starter web application



**Note:** Press <Ctrl> + <C> when you need to do something else in your Terminal. This stops the watch task of the grunt serve command.

Leave the grunt watch running in your Terminal. To run additional commands in the Terminal, open a new window using the **File** menu (File>Open Tab).

2. Add a new navigation link (route).

On the left side of the web page, there is an existing navigation component. You will add a new link called **Patients**

The code provided adds an Angular UI route which is used to associate a view and a controller with a URL. The `routes.js` file contains all of the routes for the application and you will add a new state in this file.

**Note:** The paths noted in the lab instructions assume you are starting in the `predix-seed-1.1.3` directory unless otherwise directed.

As noted previously, to copy and paste code into your application files, open the **UILab.txt** file in the gedit text editor and refer to it for the duration of the lab. This file is in the `predix/PredixApps/training_labs/fundamentals` folder.



- In your text editor, navigate to the `public/scripts` directory (under the `predix-seed-1.1.3` directory)
  - ◆ Open the `routes.js` file
  - ◆ Add a new state to the file
    - Remove the semicolon after the components statement
    - Paste in the code provided at the end of the `$stateProvider` section just after `/components.html`

Ensure that your code reads exactly as below:

```

    })
    .state('components', {
      url: '/components',
      templateUrl: 'views/components.html'
    })
    .state('patients', {
      url: '/patients',
      templateUrl: '/views/patient/index.html',
      controller: 'PatientsCtrl'
    });

```

- ◆ Save the file (**<Ctrl + S>**)

3. Create a new link (tab) in the navigation and add some data for display in the view.

- Create a new link
  - ◆ In your text editor, navigate to the `public/scripts` directory
  - ◆ Open the `app.js` file
  - ◆ Add a comma after `label: 'Data Control'`
  - ◆ Add the code provided to the `tabs` array

This code defines router paths and is where the name of the route is matched to the controller and view template.

Your code should read as follows:

```

//Global application object
window.App = $rootScope.App = {
  version: '1.0',
  name: 'Predix Seed',
  session: {},
  tabs: [
    {icon: 'fa-home', state: 'about', label: 'About'},
    {icon: 'fa-tachometer', state: 'dashboard', label: 'Dashboard'},
    {icon: 'fa-th', state: 'cards', label: 'Cards', subitems: [
      {state: 'interactions', label: 'Interactions'},
      {state: 'dataControl', label: 'Data Control'},
      {state: 'patients', label: 'Patients'}
    ]},
    {icon: 'fa-bar-chart', state: 'components', label: 'Components'}
  ]
};
});

```

#### 4. Store some dummy data for display in the view.

- In the same file, store some dummy data for display in the view

- ◆ Locate this line in the file:

```
predixApp.controller('MainCtrl', ['$scope',
'$rootScope', function, ...
```

- ◆ After this line, paste the code provided into the root scope

Your code should read as follows:

```

predixApp.controller('MainCtrl', ['$scope', '$rootScope', function ($scope,
$rootScope) {
  //we'll store patients in rootScope for now, so we can use them on multiple
views later.
  $rootScope.patients = [
    { id: 1, firstName: 'Bob', lastName: 'Dylan' },
    { id: 2, firstName: 'Joe', lastName: 'Sammy' }
  ];
  //Global application object

```

- ◆ Save the file

---

## Exercise 2: Creating a Controller

---

### Overview

In this exercise, you add the Patients controller to the application.

### Steps

1. Create the new patients controller in the `patients.js` file.

- In your text editor, navigate to the `public/scripts/controllers` directory in the seed application
    - ◆ Create a file called `patients.js` and paste the code provided into the file
- Your code should read as follows:

```
'use strict';
define(['angular', 'sample-module'], function(angular, controllers) {
  // Controller definition
  controllers.controller('PatientsCtrl', ['$rootScope', '$scope', function($rootScope, $scope) {
    $scope.form = {};
    $scope.addPatient = function () {
      var patient = {
        id: $scope.patients.length + 1,
        firstName: $scope.form.firstName,
        lastName: $scope.form.lastName
      };
      $rootScope.patients.push(patient);
      $scope.form = {};
    };
  }]);
});
```

- ◆ Save the file

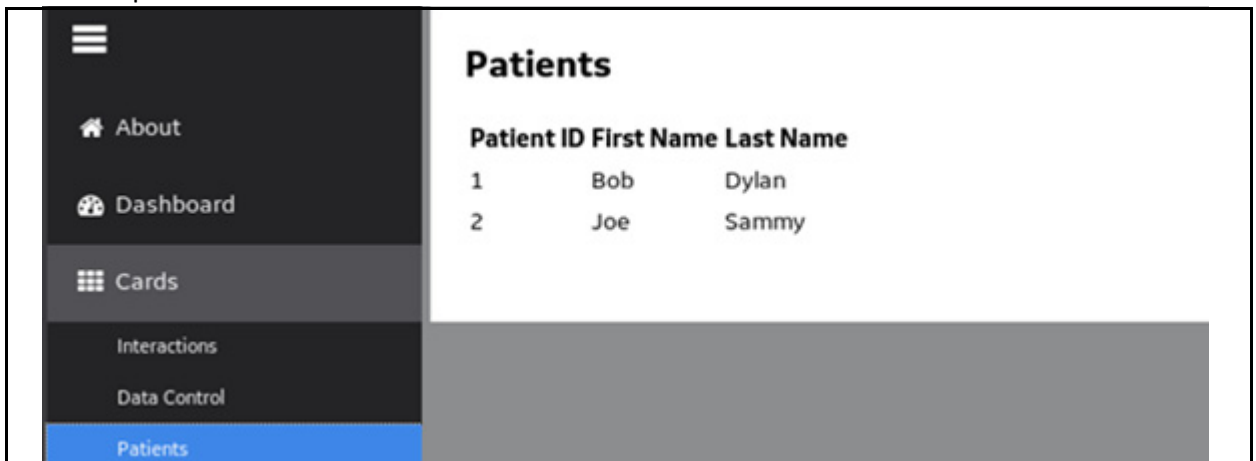
This code defines the Patients controller (PatientsCtrl), the add.Patient function, and the form (model)

## 2. Add a reference to the new controller.

- In your text editor, navigate to the `/public/scripts/controllers` directory
  - ◆ Open the file `main.js` file
  - ◆ Paste the code provided into the file, replacing all codeThis reference ensures that the new controller is loaded into the browser.
  - ◆ Save the file

## 3. Create a new view to display a table of patient data.

- Create a new directory called `patient` in the `public/views` folder
- Create a new file in the `patient` folder called `index.html`
- In your text editor, navigate to the `public/views/patient` directory
  - ◆ Open the `index.html` file
  - ◆ Paste the code provided into the file
  - ◆ Save the fileThis view uses the Angular `ng-repeat` directive to iterate over the list of patients in scope.
  - ◆ Refresh the browser and the application appears with the new navigation link and the patient data



---

## Exercise 3: Changing the View and Model

---

### Overview

In this exercise, you will add a form (with two name fields) that allows the user to dynamically change the view and the model (data) on the web page. You will add **First Name** and **Last Name** input fields and an **Add Patient** button to allow users to add patient data to the table. Finally, you will add a search field to provide filtering.

### Steps

1. Add **Name** entry fields and an **Add Patient** button.

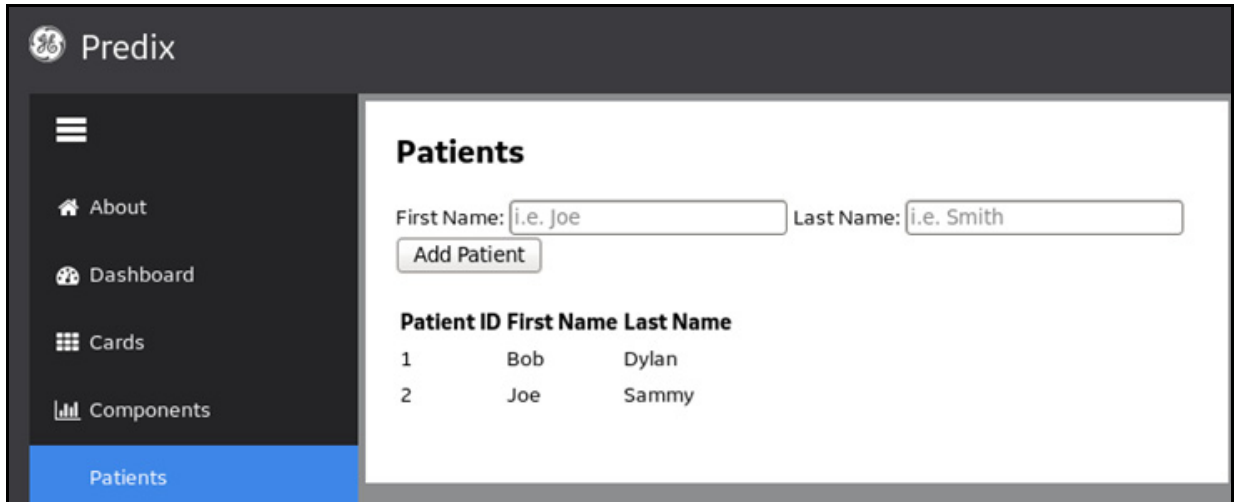
- In your text editor, navigate to the `views/patient` directory
  - ◆ Open the `index.html` file
  - ◆ Enter the code provided after the `<h2>` tag

Your code should read as follows:

```
<px-card>
  <article role="article">
    <h2 class="u-mt0 u-pt+>Patients</h2>
    <div class="flex">
      <form ng-submit="addPatient()">
        <label for="first-name">First Name: </label>
        <input type="text" id="first-name" placeholder="i.e. Joe" ng-model="form.firstName">
        <label for="last-name">Last Name: </label>
        <input type="text" id="last-name" placeholder="i.e. Smith" ng-model="form.lastName">
        <button type="submit">Add Patient</button>
      </form>
    </div>
  </article>
</br/>
  <div class="flex">
```

- ◆ Save the file
- ◆ Refresh your browser

Your web page should look similar to the one below:



- ◆ Add some patient First and Last Names to test your page  
The **Add Patient** button should submit the names to your list.

2. Add a search field to filter the patient data on the page.

- In your text editor, navigate to the `public/views/patient` directory
  - ◆ Open the `index.html` file
  - ◆ Add the code provided after the `<br/>` tag

```

</div>
<br/>
<p>Search: <input ng-model="filterText"></p>
<div class="flex">

```

3. Iterate over the patient table.

- ◆ Find the following line in the same file:
 

```
<tr ng-repeat="patient in patients">
```
- ◆ Replace the line with the code provided

- ◆ Save the file
- ◆ Refresh the browser
- ◆ Navigate to the **Patients** page and test the search/filter functionality

The application allows you to add multiple patient names and filter on them.

**Patients**

First Name:  Last Name:

Search:

Patient ID	First Name	Last Name
1	Bob	Dylan
2	Joe	Sammy
3	Johnny	Rivera
4	Sammy	Rivera
5	Tammy	Rivera
6	Albert	Rivera
7	John	Smythe-Smith
8	Edward	Rivera
9	Sam	Spade
10	Sarah	O'Connor
11	Wilson	Wilson
12	Tim	Taylor
13	Frank	Underwood

**Patients**

First Name:  Last Name:

Search:

Patient ID	First Name	Last Name
3	Johnny	Rivera
4	Sammy	Rivera
5	Tammy	Rivera
6	Albert	Rivera
8	Edward	Rivera

---

## Exercise 4: Styling Your Application

---

### Overview

In the last lab, you created a Patient Input Form without any styling. In this lab, you'll use Predix styles to give the form a nice look and feel.

Normally, you use the GitHub repository to copy Predix projects (web parts, components, elements) to your laptop. We have already staged the `px-theme`, `px-library-design`, `px-forms-design`, and the `generator-px-comp` projects on your DevBox. All of the projects' dependencies have also been installed using `npm install` and `bower install`. To see the actual steps for this, see the **Installing Px Theme Components from GitHub** section in the Appendix at the end of this guide.

### Steps.

1. Generate the CSS files and add the `px-forms-design` component.

- From the `predix/PredixApps/training_labs/fundamentals/px-theme` directory in the Terminal, run the `grunt` command

The Sass pre-processor generates the CSS files, including the `px-forms-design` component.

**Tip:** Grunt is a command line tool that runs tasks for JavaScript. Here it assures that the Sass files generate the appropriate CSS. In the next step, the `grunt watch` command ensures that CSS files are updated as changes are made to `.scss` files. This allows you to immediately see your code changes in the web application.

- ◆ Open a new Terminal window by selecting **Open Tab** from the **File** menu
- ◆ Run the `grunt watch` command from the `/px-theme/sass` directory
- ◆ In your text editor, navigate to the `px-theme/sass` directory
- ◆ Open the `px-page-theme.scss` file



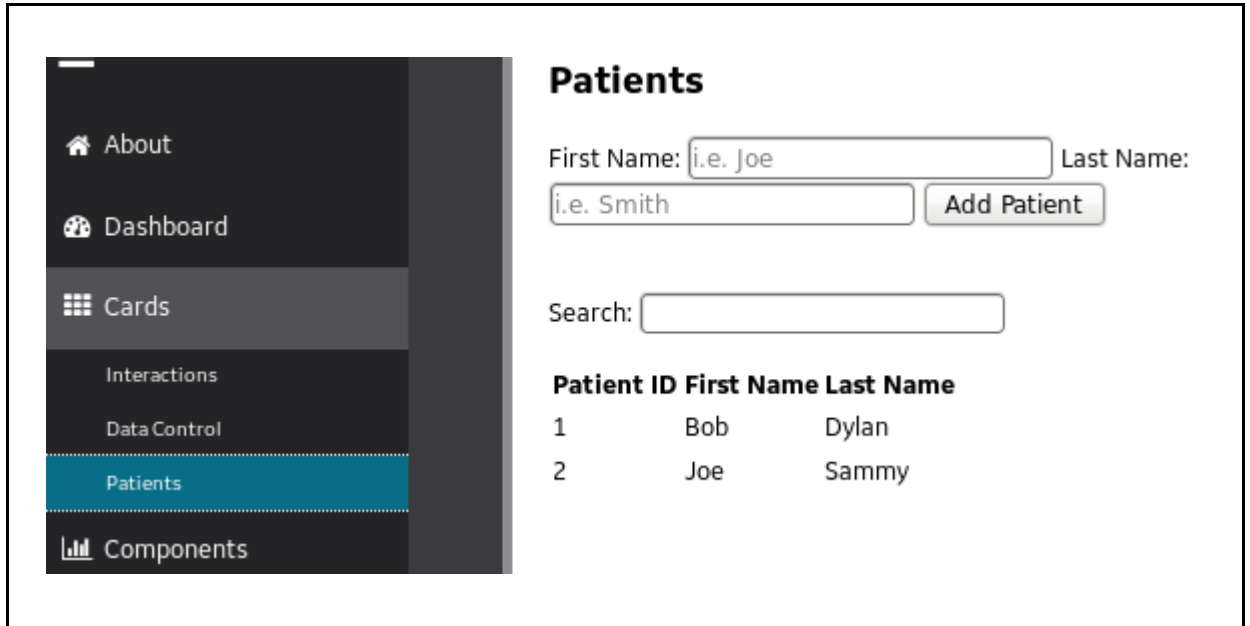
- ◆ Insert the code provided just above the `//App` line so your code appears as follows:

```
@import "px-theme.scss";
@import "px-forms-design/_base.forms.scss";
// App
html {
  position: relative;
```

- Run the following commands in the Terminal:
  - ◆ From the `/predix/PredixApps/training_labs/fundamentals/px-theme` directory, run the **bower link** command
  - ◆ From the `predix-seed-1.1.3` directory, run **bower link px-theme**  
These commands link your px-theme project to the predix-seed-1.1.3 project directly.
  - ◆ Reload the “Patients” page in your browser

**Tip:** You may need to run the `grunt serve` command in the `predix-seed-1.1.3` directory again.

Your Patients page appears as below:



**Patients**

First Name:  Last Name:

Search:

Patient ID	First Name	Last Name
1	Bob	Dylan
2	Joe	Sammy

## 2. Add classes to the HTML.

- In your text editor, navigate to the `predix-seed-1.1.3/public/views/patient` directory
  - ◆ Open the `index.html` file, and replace all of the HTML from and including the `<form>` tags with the code provided

Your code should appear as follows:

```

    <div class="flex">
<form ng-submit="addPatient()" >
  <ol class=list-bare>
    <li class=form-field>
      <label for="first-name">First Name: </label>
      <input type="text" id="first-name" placeholder="i.e. Joe" ng-model="f
    </li>
    <li class=form-field>
      <label for="last-name">Last Name: </label>
      <input type="text" id="last-name" placeholder="i.e. Smith" ng-model="
    </li>
  </ol>
  <input class="btn btn--primary" type="submit" value="Add Patient">
</form>
</div>
<br/>

```

If you reload the browser now, you will not see the changes because the `input--small` and `btn--primary` classes are not included by default. Px only includes classes you want to use.

## 3. Add the class to your project - primary button.

- To include these classes, in the `px-theme/sass` directory, open the `px-page-theme.scss` file.
  - ◆ Add the code provided in the `//Objects` section on the line after `"$inuit-enable-btn--bare"`



#### 4. Add the class to your project - small input.

- ◆ Add the code provided in the same section on the line before

`@import "px-forms-design/_base.forms.scss";` and save

**Note:** Be sure to place the code exactly as instructed in the .scss file because line order is critical.

Your code should appear as follows:

```
// Objects
$inuit-enable-btn--bare : true;
$inuit-enable-btn--primary : true;

@import "px-buttons-design/_objects.buttons.scss";

$inuit-enable-layout--small : true;
$inuit-enable-layout--flush : true;
$inuit-enable-layout--full : true;

@import "px-layout-design/_objects.layout.scss";

@import "px-theme.scss";
$inuit-enable-input--small : true;
@import "px-forms-design/_base.forms.scss";
// App
```

- Reload the page in your browser

Your Patients page should have a new blue button and nicely formatted, small text fields.

**Patients**

First Name:

Last Name:

[Add Patient](#)

Search:

Patient ID	First Name	Last Name
1	Bob	Dylan

---

## Exercise 5: Creating a View to Display a Web Component

---

### Overview

In this exercise, you will create a view in order to display a hospital web component. The web component displays the number of hospitals within the network. In order to create the view, you create another new state for the Angular UI router to use. You will also create a new link to be able to navigate to the view. Finally, you will add a controller with some dummy data to display in the view.

### Steps

1. Add a new state.

- In your text editor, navigate to the `predix-seed-1.1.3/public/scripts` directory
  - ◆ Open the `routes.js` file
  - ◆ Add a new state to the file
    - Remove the semicolon after the `components` statement
    - Paste in the code provided at the end of the `$stateProvider` section and save the file

Ensure that your code reads exactly as below:

```
    })
    .state('patients', {
      url: '/patients',
      templateUrl: '/views/patient/index.html',
      controller: 'PatientsCtrl'
    })
    .state('hospitals', {
      url: '/hospitals',
      templateUrl: 'views/hospital/index.html',
      controller: 'HospitalsCtrl'
    });
```

## 2. Create a new Hospital link for the view.

- In your text editor, navigate to the `public/scripts` folder
  - ◆ Open the `apps.js` file
  - ◆ Add the given state and label information to the `tabs` array and align the text (using spaces) underneath the prior state information
  - ◆ Add a comma after `'Patients' }`

Your code should read as below:

```

name: 'Predix Seed',
session: {},
tabs: [
  {icon: 'fa-home', state: 'about', label: 'About'},
  {icon: 'fa-tachometer', state: 'dashboard', label: 'Dashboard'},
  {icon: 'fa-th', state: 'cards', label: 'Cards', subitems: [
    {state: 'interactions', label: 'Interactions'},
    {state: 'dataControl', label: 'Data Control'},
    {state: 'patients', label: 'Patients'},
    {state: 'hospitals', label: 'Hospitals'}
  ]},
  {icon: 'fa-bar-chart', state: 'components', label: 'Components'}
]

```

- Press `<ctrl> + <s>` to save the file

## 3. Add the controller for the view.

- In your text editor, navigate to the `public/scripts/controllers` directory
  - ◆ Create a new file called `hospitals.js`
  - ◆ Paste in the code provided to create the controller

The file includes some dummy hospital data.

  - ◆ Save the file as `hospitals.js`

4. Add a reference to the new controller.

- In your text editor, navigate to the `public/scripts/controllers` directory
  - ◆ Open the `main.js` file
  - ◆ Replace all of the contents of the file with the given code.
  - ◆ Press `<ctrl> + <s>` to save the file

5. Create the view to display the hospital web component.

- In your File Finder, navigate to the `public/views` directory and create a new folder called `hospital`
- Create a new `index.html` file there
  - ◆ Open the file with your editor and enter the code provided
  - ◆ Save the file

6. Test your application locally.

- Open your browser, refresh your page and navigate to the **Hospitals** link.  
You should see a tab with the text, *“There are 2 hospitals within this network.”*





---

## Exercise 6: Creating a Web Component

---

### Overview

In this exercise, you will customize a Predix component and connect it to the seed application. We have staged the component on the DevBox for you.

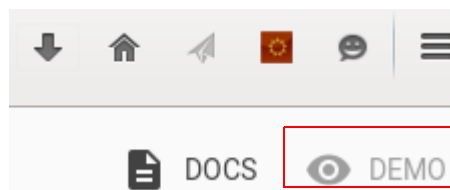
Normally, you would download and install the component and its dependencies from GitHub and then generate the component using a Predix component Yeoman generator. This component renders a simple HTML table that displays hospital data. The Yeoman generator allows developers to specify how to build their web application. It uses the yo scaffolding tool from Yeoman as well as a package manager like bower or npm, and a build tool like Grunt.

To see the actual steps for this, see the **Creating a Web Component** section in the Appendix at the end of this guide.

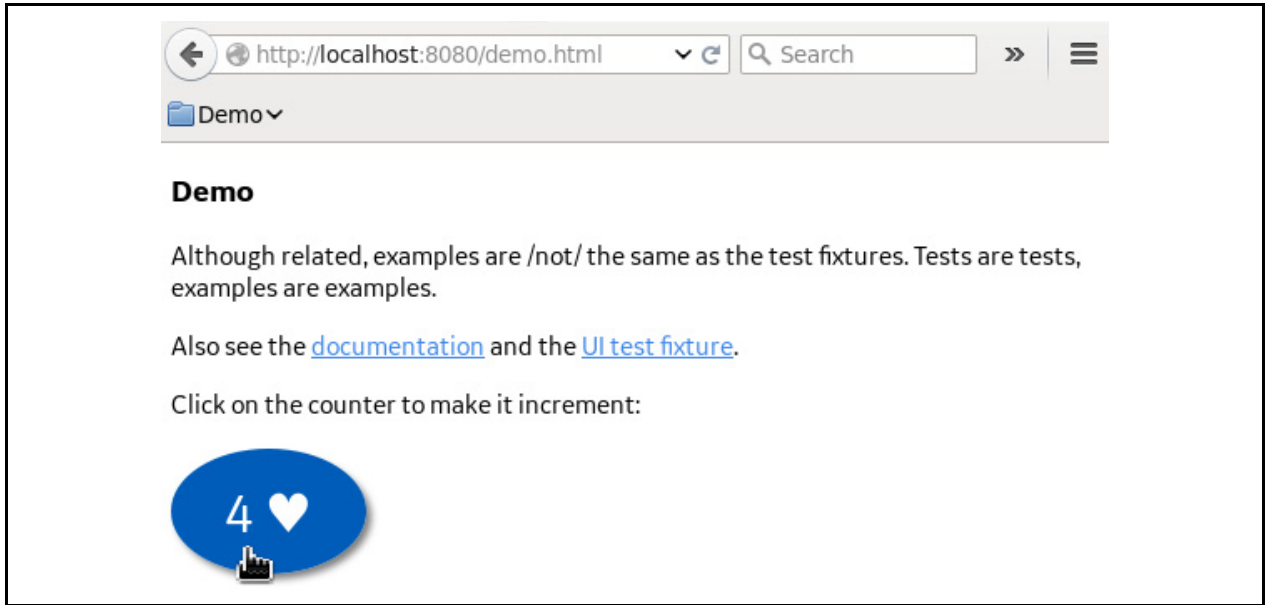
### Steps

1. Test the web component locally.

- In the Terminal, change directories and start the local test by running these commands
  - ◆ `cd ~/predix/PredixApps/training_labs/fundamentals/hospital-info`
  - ◆ `grunt firstrun`  
The `grunt firstrun` command interprets the Sass into CSS and starts a local web server to test the component by itself.
  - ◆ Click the **DEMO** link in the upper right hand corner to see the web component

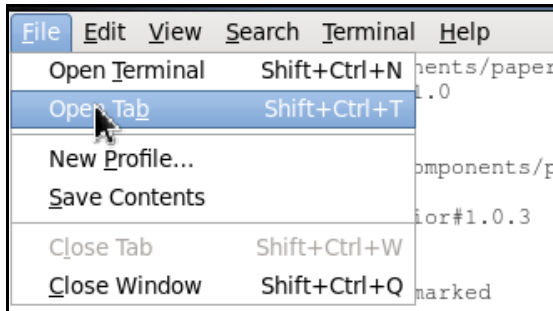


- ◆ Click the number to increment it and test the demo page



2. Tell the grunt service to watch for project changes.

- Open a new Terminal window by selecting **Open Tab** from the **File** menu in the Terminal



- Run the command: `grunt watch`  
This tells the grunt service to look for changes in the project. It automatically processes the Sass code (.scss files) into CSS as you make changes.

3. Replace the code in the component section of the hospital-info-sketch file.

- In your text editor, navigate to the `/training_labs/fundamentals/hospital-info/sass/directory`
- Open the `hospital-info-sketch.scss` file
- Replace the Component section with the code provided
- Save the file

4. Replace the dom-module section in the hospital-info.html file.

- Navigate to the `/training_labs/fundamentals/hospital-info` directory
  - ◆ Open the `hospital-info.html` file
  - ◆ Replace the `<dom-module>` section with the code provided
  - ◆ Save the file

5. Replace the script section in the hospital-info.html file.

- In the same file, replace the `<script>` section with the code provided
- Save the file
- See the next page to make sure your code is correctly placed.

Your file should read as follows:

```

21 @demo demo.html
22 -->
23 <<dom-module id="hospital-info">
24   <link rel="import" type="css" href="css/hospital-info.css"/>
25   <template>
26     <div class="flex">
27       <h4>Hospital Details</h4>
28     </div>
29     <div>
30       <table class="table hospital-table">
31         <tr><th class="text--right">Hospital Name :</th><td><span>
{{hospitalDetails.name}}</span></td></tr>
32         <tr><th class="text--right">Hospital Address :</th><td><span>
{{hospitalDetails.address}}</span></td></tr>
33         <tr><th class="text--right">Email :</th><td><span>
{{hospitalDetails.email}}</span></td></tr>
34         <tr><th class="text--right">Phone :</th><td><span>
{{hospitalDetails.phone}}</span></td></tr>
35       </table>
36     </div>
37   </template>
38 </dom-module>Rel
39
40 <script>
41   Polymer({
42     is: 'hospital-info',
43     properties: {
44       hospitalDetails: {
45         type: Object
46       }
47     }
48   });
49 </script>

```

## 6. Connect the hospital-info component to the seed app.

- In the Terminal, run these commands:
  - ◆ (from the hospital-info directory) **bower link**
  - ◆ **cd ~/PredixApps/training\_labs/fundamentals/predix-seed-1.1.3**
  - ◆ **bower link hospital-info**
- In your text editor, navigate to the `predix-seed-1.1.3/public` directory
  - ◆ Open the `index.html` file
  - ◆ Add the code provided at the end of the `<card.html>` section as follows

```
card.html"/>
  <link rel="import" href="bower_components/px-sample-cards/hide-card.html"/>
  <link rel="import" href="bower_components/px-sample-cards/toggle-card.html"/>
  <link rel="import" href="bower_components/px-sample-cards/card-to-card.html"/>
  <link rel="import" href="bower_components/px-sample-cards/description-card.html"/>
  <link rel="import" href="bower_components/px-sample-cards/fetch-data-card.html"/>
  <link rel="import" href="bower_components/px-sample-cards/temperature-card.html"/>
  <link rel="import" href="bower_components/hospital-info/hospital-info.html"/>
</head>
```

- ◆ Modify the `<h1>` tag, after the `svg>` tag, replace the word “Predix” with a name for your app, such as “My Healthcare System”

```
407.7-264.7 407.9zM-262.4 391.5c0-2.3 2.3-6.6 3.6-6.
391.5zM-249.4 390.5c0-2.8 1.9-5.6 3-5.1C-245.2 386-2
svg>
  My| Healthcare System
  </h1>
</div>
</header>
<div class="viewport">
  <div class="layout layout--full layout--flush">
```

- ◆ Save the file

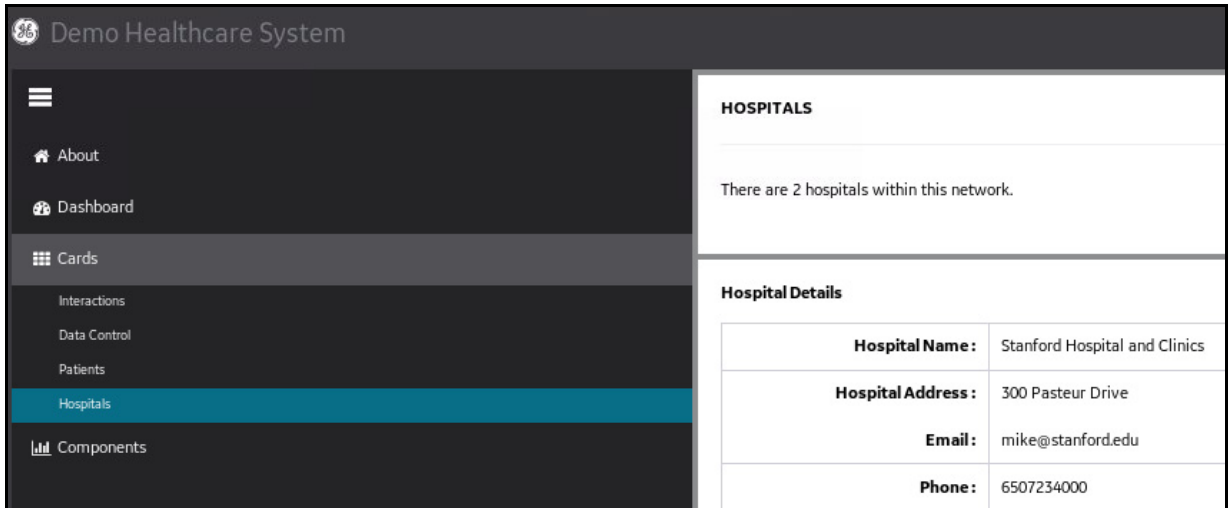
## 7. Add the px-card to show hospital information in the project.

- In your text editor, navigate to the `predix-seed-1.1.3/public/views/hospital` directory
  - ◆ Open the `index.html` file

- ◆ Add the code provided to the bottom of the file and save the file

```
<px-card header-text="Hospitals">
  <article role="article">
    <div class="flex">
      <p>There are {{hospitals.length}} hospitals within this network.</p>
    </div>
  </article>
</px-card>
<px-card>
  <hospital-info hospital-details="{{hospitalDetails}}"></hospital-info>
</px-card>
```

- ◆ Run **grunt serve** from the predix seed directory to see the new component:



This component can be used anywhere in your application. You could use it in a different application by installing it into that application using `bower install`. In future, there will be a Predix web component catalog for sharing your work with other teams.

---

## Exercise 7: Connecting a Microservice

---

### Overview

Before Polymer components, most API calls were made from Angular controllers or services and this is still a supported pattern in Predix. However, in this exercise, you will fetch data from a Polymer iron-ajax element.

You use the following syntax to bring data into a Polymer web component. You use the URL of a microservice to do this, but in our lab we'll bring data in from a json file as the URL.

```
<iron-ajax auto url="{{microservice_Url}}"
last-response="{{data}}"></iron-ajax>
```

### Connecting from a Polymer Web Component

1. Install a Polymer element.

- In a Terminal window, change to the `predix-seed-1.1.3` directory and run the following command:

```
bower install polymerelements/iron-ajax --save
```

- Enter **1** when prompted

This command installs the polymer iron-ajax element. It also provides a reference to the polymer iron-ajax elements in the `bower.json` file.

2. Use the iron-ajax component to fetch data and pass the info to the hospital-info component.

- Navigate to the `index.html` file in the `predix-seed-1.1.3/public/views/hospital` folder
- Replace all of the code in the file with the code provided and save the file.  
Note that the `hospital-details.json` file (object) is in place of a URL that would normally provide an endpoint into a microservice.

The options used below the `<iron-ajax` tag are `auto`, `url`, `handle-as` and `last-response`. The `auto` option tells the system to make the rest call when the `iron-ajax` element loads or when the `URL` or `params` options changes. The `URL` tells the program to go to that location to get data. `Handle-as` tells the program if the data returning will be XML, json, a blob, a document, or other data type. `Last-response` refers to the most recent response from the ajax request.

3. Create a mock-data file to replaces data coming from a URL).

- In the `predix-seed-1.1.3/public` create a file called `hospital-details.json`
- Copy and paste the code provided into the new file and save it.
- In the Terminal, from the `predix-seed-1.1.3` directory, run the **grunt serve** command

The details in the `hospital-details` file display in the Polymer element in the web browser.







## Lab 6 : Using the Asset Service

### Learning Objectives

By the end of this lab, students will be able to:

- Create an Asset service instance in Cloud Foundry
- Bind an application to an Asset service instance
- Login as the administrator client
- Add a new client
- Create a user
- Invoke Service Asset APIs to retrieve, add and delete asset objects
- Use JSON to define asset objects
- Construct filters using Graph Expression Language to query assets

### Lab Exercises

- [Creating an Asset Service Instance, page 109](#)
- [Binding to Your Asset Service Instance, page 111](#)
- [Fetch a Token from the UAA Service, page 113](#)
- [Retrieve Asset Model Data, page 118](#)
- [Add an Asset to the Asset Model, page 122](#)
- [Link Domain Objects, page 125](#)
- [Delete an Asset from the Asset Model, page 127](#)
- [Construct GEL Queries, page 128](#)
- [Constructing Transitive Closure Queries, page 132](#)

### Directions

Complete the exercises that follow.



## Exercise 1: Creating an Asset Service Instance

### Overview

To use a Predix Service, you must first create an instance of the service in Cloud Foundry. This first exercise provides you with the steps to create an instance of the asset service using the trusted issuer ID from the UAA instance you created previously. A text file called Asset\_Lab.txt has been placed in your environment for you to copy some of the commands used in this lab.

### Steps

1. Display all services in the Predix marketplace.

- In the Terminal run the command: `cf marketplace`

```
[predix@localhost ~]$ cf marketplace
Getting services from marketplace in org Predix-Training / space Training2 as student14...
OK
```

service	plans	description
<b>logstash</b>	free	Logstash 1.4 service for application dev
<b>logstash-5</b>	free	Logstash 1.4 service for application dev
<b>p-rabbitmq</b>	standard	RabbitMQ is a robust and scalable high-p
<b>p-rabbitmq-35</b>	standard	RabbitMQ is a robust and scalable high-p
<b>postgres</b>	shared, shared-nr	Reliable PostgreSQL Service
<b>predix-acs</b>	Beta, Enterprise*	Use this service to provide a more power
<b>predix-acs-training</b>	Basic, Free	Design precise access controls and user
<b>predix-analytics-catalog</b>	Beta, Enterprise*	Add analytics to the Predix cloud for us
<b>predix-analytics-runtime</b>	Beta, Enterprise*	Use this service to support elastic exec
<b>predix-asset</b>	Tiered	Create and store machine asset models an
<b>predix-mobile</b>	Beta, Enterprise*	Design, develop, and deploy Industrial I
<b>nectivity.</b>		
<b>predix-timeseries</b>	Bronze, Silver*, Gold*	Quickly and efficiently manage, ingest,
<b>predix-uaa</b>	Beta, Enterprise*	Use this service for a full-featured OAU

**Analysis:** The Asset service is available in the Service catalog. To use the Asset service, you will first need to create an instance of the service in your space. This is done using the `cf create-service` command.

## 2. Create an Asset service instance.

- In the Terminal run the `cf create-service --help` command to display a list of required arguments:

```
cf create-service predix-asset <plan> <asset_instance> -c
'{"trustedIssuerIds":["<uaa_url>"]}'
```

where:

- Replace `<plan>` with the plan name (e.g. Beta, Free)
- Replace `<asset_instance>` with an instance name of your choice
- `<uaa_url>` - IssuerId from your UAA instance created earlier

- Your command will look similar to this:

```
[predix@localhost ~]$ cf create-service predix-asset Tiered cf-asset
-c '{"trustedIssuerIds":["https://c758a891-d2d0-4d19-a2a3-450496f85
57e.predix-uaa-training.run.aws-usw02-pr.ice.predix.io/oauth/token"]}
}'
Creating service instance cf-asset in org Predix-Training / space Tr
aining3 as student55...
OK
```

## 3. Verify the Asset service instance was created in your training space.

- To list all services in your training space, run the command:

```
cf services
```

```
[predix@localhost ~]$ cf services
Getting services in org Predix-Training / space Training2 as student14...
OK
```

name	service	plan	bound apps
<b>123Asset</b>	predix-asset	Tiered	
<b>Asset_Bill</b>	predix-asset	Tiered	
<b>EngineAsset</b>	predix-asset	Tiered	
<b>Joni_acs_instance</b>	predix-acs-training	Free	trainingsamp

Your instance of the Asset service should now be available in the training space.

---

## Exercise 2: Binding to Your Asset Service Instance

---

### Overview

In this exercise you will bind your Spring-Music application to your Asset service instance.

### Steps

1. Bind your sample application to your Asset service instance.

- In the Terminal run the command:

```
cf bind-service <Spring-Music-app> <AssetService_YourName>
```

Where <Spring-Music-app> is the name of your Spring-Music application

Where <AssetService\_YourName > is the name of your Asset service

**Note:** you can abbreviate the command `cf bind-service` as `cf bs`

*For example:*

```
[predix@localhost ~]$ cf bind-service trainingsample_DoNotDelete EngineAsset
Binding service EngineAsset to app trainingsample_DoNotDelete in org Predix-Training / s
OK
```

## 2. Verify that your application is bound to your asset service instance.

- In the Terminal run the command: `cf s`
- ◆ Your service instance should now be listed and bound to the sample application

name	service	plan	bound apps
123Asset	predix-asset	Tiered	trainingsamp
AssetService_KV	predix-asset	Tiered	trainingsamp
Asset Bill	predix-asset	Tiered	trainingsamp
EngineAsset	predix-asset	Tiered	trainingsamp
Joni_acs_instance	predix-acs-training	Free	trainingsamp
Joni_uaa_instance	predix-uaa-training	Free	trainingsamp
MyAsset	predix-asset	Tiered	
SanRamon_postgres	postgres	shared-nr	
Terry_postgres	postgres	shared-nr	
abc_Asset	predix-asset	Tiered	trainingsamp

## 3. Display environment variables for your application.

- In the Terminal run the command:  
`cf env <Spring-Music-app>`

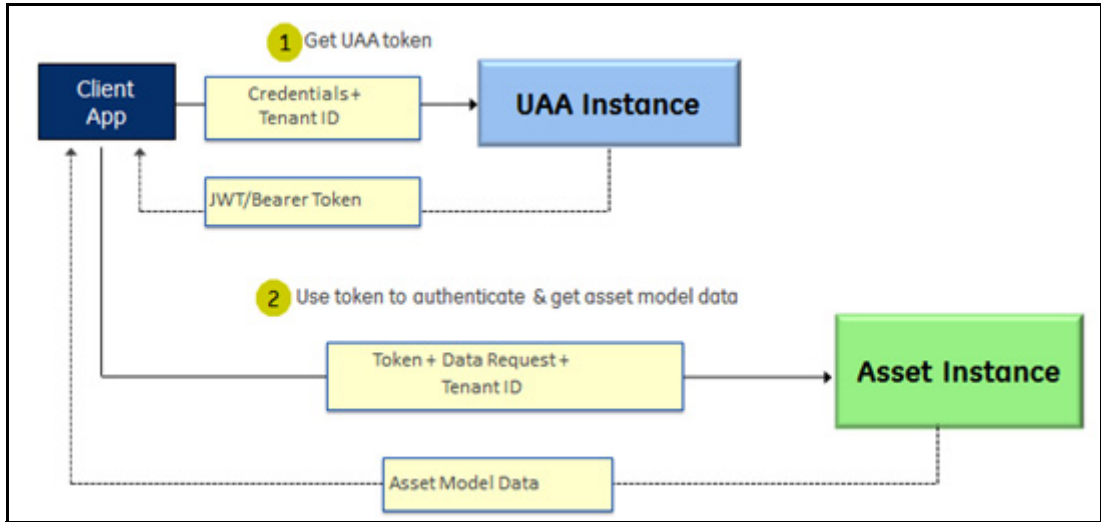
```

"label": "predix-asset",
"name": "EngineAsset",
"plan": "Tiered",
"tags": []
},
{
  "credentials": {
    "instanceId": "b0e931fa-61c9-4cea-9463-ac2863c24a78",
    "uri": "https://predix-asset.run.aws-usw02-pr.ice.predix.io",
    "zone": {
      "http-header-name": "Predix-Zone-Id",
      "http-header-value": "b0e931fa-61c9-4cea-9463-ac2863c24a78",
      "oauth-scope": "predix-asset.zones.b0e931fa-61c9-4cea-9463-ac2863c24a78.user"
    }
  },
  "label": "predix-asset",
  "name": "AssetService_KV",
  "plan": "Tiered",
  "tags": []
}

```

- ◆ The uri field displays the REST endpoint for your Asset service instance

## Exercise 3: Fetch a Token from the UAA Service



### Overview

To access and update asset model data, users require a UAA token. Here, you use the REST client to request the token from the UAA Service. The token needs to be added to every data request to the Asset Service.

**Note:** Even though you have created an Asset service instance, you will use a pre-configured asset service instance so that you can search across data that has already been loaded.



## Steps

1. Launch the REST client.

- In Firefox, launch the REST client (click on the red icon shown here)



- You will build a POST request that looks like this: (steps on next page)

[ - ] Request

Method  URL

**Headers**

Authorization: Basic dHJhaW5pbmd... x Content-Type: application/x-www-... x x-tenant: eacafb4a-6ee6-4df1-aca... x

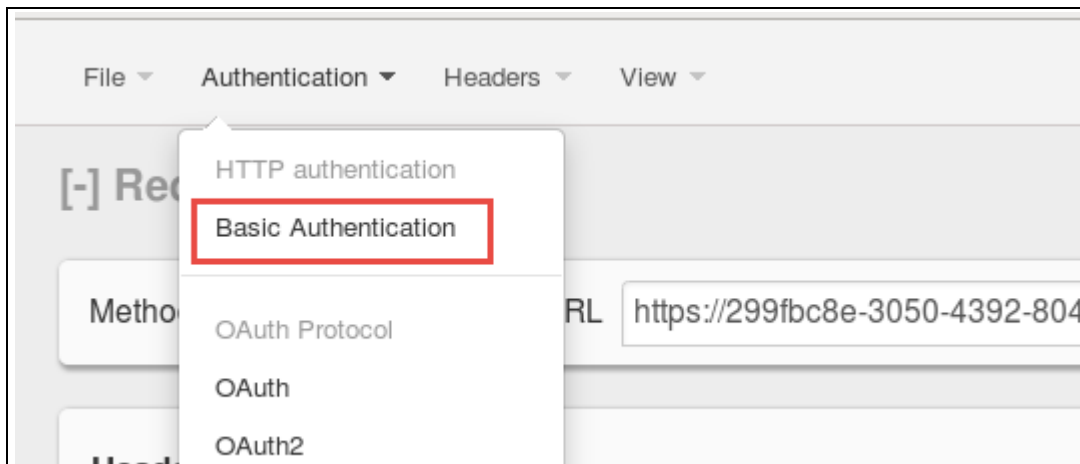
2. Begin configuring the POST request.

- Set the Method and URL (copy and paste this from the Asset\_Lab.txt file)
  - ◆ Method: **POST** (select from drop-down)
  - ◆ URL:  
`https://299fbc8e-3050-4392-8047-86d3fcc5a145.predix-uaa.run.aws-usw02-pr.ice.predix.io/oauth/token`

**WARNING:** When pasting text from a PDF, spaces are inserted into the URL at line breaks. Remove any spaces before continuing on to the next step!

3. Add username and password credentials as a header.

- Click on the **Authentication** drop-down and select **Basic Authentication**



- ◆ Enter the user name: **training\_client**
- ◆ Password: **training\_secret**
- ◆ Click **Okay**; a new authorization header has been added

4. Add a Content-Type header.

- In the Headers drop-down, select **Custom Header**
  - ◆ Enter the name: **Content-Type**
  - ◆ Enter the value: **application/x-www-form-urlencoded**
    - Click **Okay**; a second header is added

5. Add an x-tenant header.

- In the Headers drop-down, select **Custom Header**
- Enter the name: **x-tenant**
- Enter the value: **eaccfb4a-6ee6-4df1-acab-afbe41aaa506**

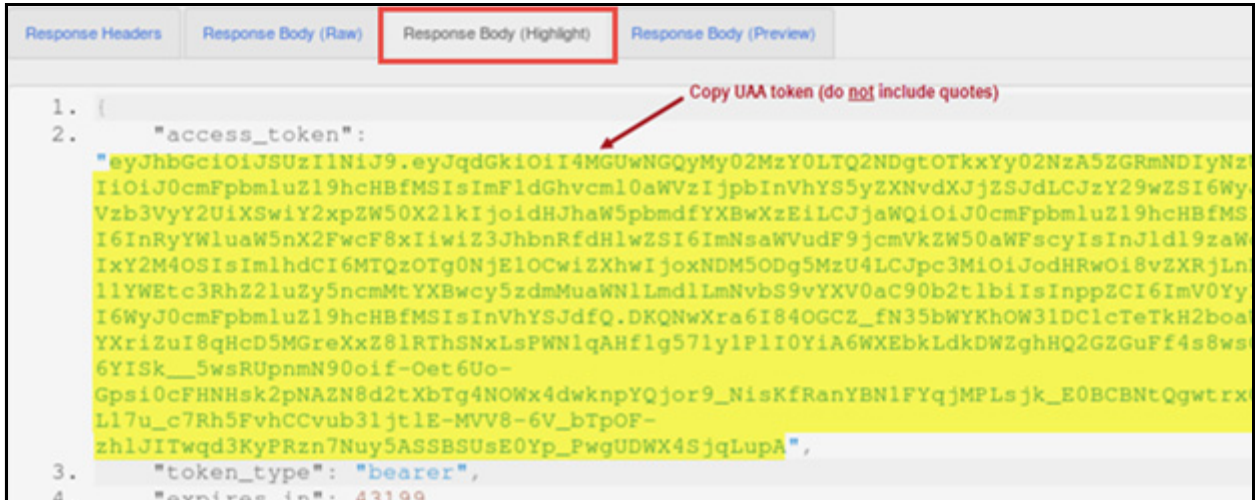
6. Construct the body of the response message.

- In the Body section enter the following:  
`client_id=training_client&grant_type=client_credentials&client_secret=training_secret`
- Click the **Send** button
- A return code **200 OK** indicates the request was successful



7. Copy the token from the response for later use.

- Under the Response section, select the *Response Body (Highlight)* tab
- Copy the UAA token as shown: (only copy text within the quotes, not the quotes themselves)



- You will use this token in the next lab to retrieve assets from the asset mode

---

## Exercise 4: Retrieve Asset Model Data

---

### Overview

In this exercise you retrieve locomotive asset objects using the Asset Service APIs.

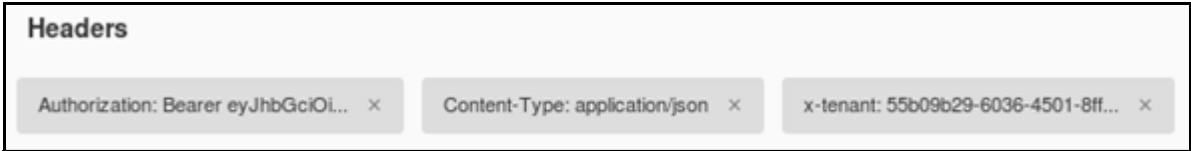
### Steps

1. Create a **GET** request for all locomotive objects.

- Open a new tab in Firebox and open the REST client
- In the REST client set the Method and URL settings
  - ◆ Method: **GET**
  - ◆ URL:  
`https://predix-asset.run.aws-usw02-pr.ice.predix.io/locomotive`
- Add a Custom Header
  - ◆ Name: **Content-Type**
  - ◆ Value: **application/json**
- Add a second Custom Header
  - ◆ Name: **Authorization**
  - ◆ Value: **Bearer** <paste the Token returned by UAA>



- Add a third Custom Header
  - ◆ Name: **x-tenant**
  - ◆ Value: **eaccfb4a-6ee6-4df1-acab-afbe41aaa506**



- **Send** the request; verify a status code of **206 Partial Content** is returned in the *Header Request*

## 2. Review the returned asset data.

- Select the *Response Body (Preview)* tab to view locomotive assets returned

```
[
  {
    "uri": "/locomotive/1",
    "type": "Diesel-electric",
    "model": "ES44AC",
    "serial_no": "001",
    "emission_tier": "0+",
    "fleet": "/fleet/up-1",
    "manufacturer": "/manufacturer/GE",
    "engine": "/engine/v12-1",
    "hqLatLng":
    {
      "lat": 33.914605,
      "lng": -117.253374
    }
  },
  {
    "uri": "/locomotive/10",
    "type": "Diesel-electric",
    "model": "SD70ACe",
    "serial_no": "0010",
    "emission_tier": "0+",
    "fleet": "/fleet/up-4",
    "manufacturer": "/manufacturer/electro-motive-diesel",
    "engine": "/engine/v16-2-5",
    "hqLatLng":
    {
      "lat": 47.941049,
      "lng": -100.126484
    }
  },
],
```

3. Copy JSON for a single asset (for use in next exercise).

- In the *Response Body (Preview)* tab, copy the JSON code for the first locomotive asset as shown here

```
{  
  {  
    "uri": "/locomotive/1",  
    "type": "Diesel-electric",  
    "model": "ES44AC",  
    "serial_no": "001",  
    "emission_tier": "0+",  
    "fleet": "/fleet/up-1",  
    "manufacturer": "/manufacturer/GE",  
    "engine": "/engine/v12-1",  
    "hqLatLng":  
    {  
      "lat": 33.914605,  
      "lng": -117.253374  
    }  
  },  
}
```

- ◆ Include the starting bracket "[" and the closing brace "}," for the asset



---

## Exercise 5: Add an Asset to the Asset Model

---

### Overview

Post a request to add a new locomotive asset to your asset model.

### Steps

1. Use the POST method to add a new asset.

- Open a new tab in Firefox and open the REST client
  - ◆ Change the Method to **POST**
  - ◆ Verify the URL:

`https://predix-asset.run.aws-usw02-pr.ice.predix.io/locomotive`

2. Add authentication and content-type headers.

- Add a Custom Header
  - ◆ Name: **Content-Type**
  - ◆ Value: **application/json**
  - ◆ Click **Okay**
- Add a Custom Header
  - ◆ Name: **x-tenant**
  - ◆ Value: **eaccfb4a-6ee6-4df1-acab-afbe41aaa506**
- Add a Custom Header
  - ◆ Name: **Authorization**
  - ◆ Value: **Bearer** <paste the Token returned by UAA>

**TIP:** Copy and paste the UAA token from the browser tab where you created your first POST request



### 3. Construct the body of the message.

- Paste the contents of the asset you copied into the **Body** of the request
- Make the following changes to the JSON
  - ◆ **uri**: replace `locomotive/1` with `locomotive/your_name`
  - ◆ **serial\_no**: replace the existing value with a value of your choice
  - ◆ **model**: enter a value of your choice
- Remove the comma after the closing curly brace `}"`
- Add a closing bracket `]"` after the last curly brace `}"`

```
Body
[
  {
    "uri": "/locomotive/testuser",
    "type": "Diesel-electric",
    "model": "ES44test",
    "serial_no": "099999",
    "emission_tier": "0+",
    "fleet": "/fleet/up-1",
    "manufacturer": "/manufacturer/GE",
    "engine": "/engine/v12-1",
    "hqLatLng":
    {
      "lat": 33.914605,
      "lng": -117.253374
    }
  }
]
```

- **Send** the request
- In the *Response Headers* tab a **204 No Content** status code is returned, indicating the asset was successfully added

4. Submit a GET request to retrieve the new asset from the asset model.

- Select the browser tab that contains the **GET** request for all locomotive assets
- Append the name of your *new* locomotive asset to the end of the URL (in our example, the name is /testuser)

Method  URL

- **Send** the request
- Select the *Response Body (Highlight)* tab to view the returned locomotive asset



```

1. [
2.   {
3.     "uri": "/locomotive/testuser",
4.     "type": "Diesel-electric",
5.     "model": "ES44test",
6.     "serial_no": "099999",
7.     "emission_tier": "0+",
8.     "fleet": "/fleet/up-1",
9.     "manufacturer": "/manufacturer/GE",
10.    "engine": "/engine/v12-1",
11.    "hqLatLng":
12.      {
13.        "lat": 33.914605,
14.        "lng": -117.253374
15.      }
16.  }
17. ]

```

- This verifies that you successfully added a new asset to the asset model

---

## Exercise 6: Link Domain Objects

---

### Objective

In this exercise you link your new locomotive asset object to a different manufacturer object.

### Steps

1. Link the locomotive asset to a different manufacturer.

- In the REST client change the Method to: **PUT**
  - ◆ URL: verify it points to your locomotive object (our example uses /testuser):  
`https://predix-asset.run.aws-usw02-pr.ice.predix.io/locomotive/<your_asset>`
- Copy the asset information from the response section to the Body section, and change the `manufacturer` property to point to **cummins**

**Note:** The PUT operation requires you to specify all attributes for an asset

```
{
  "uri": "/locomotive/testuser",
  "type": "Diesel-electric",
  "model": "ES44test",
  "serial_no": "099999",
  "emission_tier": "0+",
  "fleet": "/fleet/up-1",
  "manufacturer": "/manufacturer/cummins",
  "engine": "/engine/v12-1",
  "hqLatLng":
  {
    "lat": 33.914605,
    "lng": -117.253374
  }
}
```

- **SEND** the request
- Verify a status code **204 OK** is returned

2. Send a GET request to verify the new link.

- In the REST client change the Method to: **GET**
- **SEND** the request
- In the *Response Body (Raw)* tab, verify the manufacturer property now references cummins:

```
{  
  "uri": "/locomotive/testuser",  
  "type": "Diesel-electric",  
  "model": "ES44test",  
  "serial_no": "099999",  
  "emission_tier": "0+",  
  "fleet": "/fleet/up-1",  
  "manufacturer": "/manufacturer/cummins",  
  "engine": "/engine/v12-1",  
}
```

---

## Exercise 7: Delete an Asset from the Asset Model

---

### Overview

In this exercise you use the DELETE API method to remove the locomotive asset (added in the previous exercise) from the asset model.

### Steps

1. Use the DELETE method to remove an asset from the asset model.

- Update the GET request
  - ◆ Change the Method to **DELETE**
  - ◆ Use the existing URL that points to your new locomotive asset
- **Send** the request
- In the *Response Headers* tab, a return code of **204 No Content** indicates the Delete operation was successful

2. Submit a GET request to verify the asset was deleted.

- In the REST client change the Method to **GET**
- **Send** the request
- In the *Response Headers* tab, the response code should be **404 Not Found**

This indicates that the asset was successfully deleted

---

## Exercise 8: Construct GEL Queries

---

### Overview

In this exercise you construct different GEL queries to control asset data returned by GET requests.

### Steps

1. Add a **fields** clause to display selected fields for locomotive objects.

- In the REST client, verify the Method is: **GET**
- Verify the request URL references the `/locomotive` domain object
- Append the following fields clause to the end of the URL

**?fields=uri,model,manufacturer**

URL `https://predix-asset.run.aws-usw02-pr.ice.predix.io/locomotive?fields=uri,model,manufacturer`

- **SEND** the request
- Select the *Response Body (Preview)* tab to view the results

```
[
  {
    "uri": "/locomotive/1",
    "model": "ES44AC",
    "manufacturer": "/manufacturer/GE"
  },
  {
    "uri": "/locomotive/10",
    "model": "SD70ACe",
    "manufacturer": "/manufacturer/electro-motive-diesel"
  },
]
```

1. Add a **filter** to query all locomotives of *type* Diesel-electric.

- In the REST client, append the following filter clause to the end of the URL  
**?filter=type=Diesel-electric**

URL `https://predix-asset.run.aws-usw02-pr.ice.predix.io/locomotive?filter=type=Diesel-electric`

- **SEND** the request
- Select the *Response Body (Preview)* tab to view assets returned

2. Query locomotives that are Diesel-electric **and** are a model of SD70ACe.

- Append the following filter clause to the end of the URL  
**?filter=type=Diesel-electric:model=SD70ACe**

URL `https://predix-asset.run.aws-usw02-pr.ice.predix.io/locomotive?filter=type=Diesel-electric:model=SD70ACe`

**Note:** The `:` symbol denotes an AND operation

- **SEND** the request
- Verify the correct subset of locomotive objects are returned



### 3. Query locomotives that have an engine type of v12-6.

**Note:** up until this point, you have constructed filters with simple attributes; now you will add a filter with an attribute that references another domain object.

- Append the following filter clause to the end of the URL  
`?filter=engine=/engine/v12-6`  
 (the “/engine/” references a path to the engine domain object)

URL `https://predix-asset.run.aws-usw02-pr.ice.predix.io/locomotive?filter=engine=/engine/v12-6`

- **SEND** the request
- A single locomotive object is returned

### 4. Construct a forward-relate query to retrieve.

- Change the URL to reference the **engine** domain object  
`https://predix-asset.run.aws-usw02-pr.ice.predix.io/engine`
- Append the following filter clause to the end of the URL  
`?filter=type=Diesel-electric>engine`

URL `https://predix-asset.run.aws-usw02-pr.ice.predix.io/engine?filter=type=Diesel-electric>engine`

- **SEND** the request
- All engines that are part of Diesel-electric type locomotives are returned
- Query Logic
  - ◆ The query first returns all objects with a type property=Diesel-electric
  - ◆ From that result set, find all objects that have an engine relationship to other objects
  - ◆ From that new set of objects, return only engine objects

5. Construct a backwards-relate query to retrieve fleet assets for customer CSX.

- Change the URL to reference the **fleet** domain object  
`https://predix-asset.run.aws-usw02-pr.ice.predix.io/fleet`
- Append the following filter clause to the end of the URL  
`?filter=name=CSX<customer`

URL `https://predix-asset.run.aws-usw02-pr.ice.predix.io/fleet?filter=name=CSX<customer`

- SEND** the request
- All fleets owned by customer “CSX” are returned

```
{
  "uri": "/fleet/csx-1",
  "name": "CSX Fleet 1",
  "customer": "/customer/csx"
},
{
  "uri": "/fleet/csx-2",
  "name": "CSX Fleet 2",
  "customer": "/customer/csx"
},
{
  "uri": "/fleet/csx-3",
  "name": "CSX Fleet 3",
  "customer": "/customer/csx"
}
```

- Query logic:
  - The query first returns all objects with a name property of CSX
  - From those objects, traverse *backwards* on the customer relationship (this returns another set of objects)
  - From that new set of objects, return only fleet objects

---

## Exercise 9: Constructing Transitive Closure Queries

---

### Overview

In this exercise you work with the *asset* data model which is more representative of a hierarchical structure containing many levels of objects. You use the transitive closure operator to traverse the asset model and return *all* asset objects for each level specified in the query.

### Steps

1. Explore the *asset* data model.

- Submit a GET request to return all objects in the *asset* collection
  - ◆ Change the URL to reference the **asset** collection  
`https://predix-asset.run.aws-usw02-pr.ice.predix.io/asset`
- The request returns all objects in the data model (partial list shown here)

```
{
  "uri": "/asset/aircraft",
  "name": "GE-Aircraft"
},
{
  "uri": "/asset/aircraftEngine",
  "name": "aircraftEngine",
  "parent": "/asset/aircraft"
},
{
  "uri": "/asset/aircraftMotor",
  "name": "aircraftMotor",
  "parent": "/asset/aircraftEngine"
},
{
  "uri": "/asset/crankShaft",
  "name": "crankShaft",
  "parent": "/asset/aircraftMotor"
},
}
```

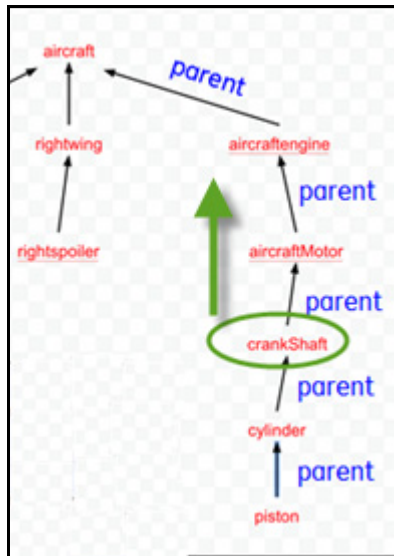
**Things to note:**

In the JSON, notice that the `parent` attribute is used to link asset objects between hierarchy levels.

For example, the `aircraft` object has no `parent` attribute which distinguishes it as the top-most parent object in the hierarchy. The `aircraftEngine` object has a `parent` attribute that points to the `/asset/aircraft` object. This indicates that `aircraftEngine` is a child to `aircraft`, and so on.

2. Execute a forward-relate transitive closure to retrieve parent assets of crankShaft.

- Append a filter clause (bolded here) to the GET request URL  
**`https://predix-asset.run.aws-usw02-pr.ice.predix.io/asset?filter=name=crankShaft>parent[t3]`**
- ◆ The query retrieves 3 levels of parent objects for all crankShaft objects



Query execution logic

- ◆ **?filter=name=crankShaft** returns all assets with a `name` attribute of **crankShaft**
- ◆ **>parent[t3]** - from those crankShaft objects, traverse the `parent` relationship in a forward (upwards) direction
- The query returns the following asset objects:
  - ◆ aircraftMotor
  - ◆ aircraftEngine
  - ◆ aircraft

### 3. Execute a backwards-relate transitive closure.

- Append a filter clause (bolded here) to the GET request URL  
`https://predix-asset.run.aws-usw02-pr.ice.predix.io/asset?filter=name=aircraftMotor<parent[t2]`

Query execution logic

- ◆ **?filter=name=aircraftMotor** will return all assets that have a `name` attribute of **aircraftMotor**
- ◆ **<parent[t2]** - from those aircraftMotor objects, traverse the `parent` relationship in a backward (downward) direction
- The query returns the following asset objects:
  - ◆ crankShaft
  - ◆ cylinder
- Experiment with different token [t] levels to traverse up and down the parent relationship in the hierarchy.





## Lab 7: Working with Analytics

### Part I: Your Dev Environment and UAA

#### Learning Objectives

By the end of this lab, students will be able to:

- Create and bind the Analytics service instances to an application
- Configure the UAA service instance for authorization and access

#### Lab Exercises

- [Create and Bind an Analytics Catalog Service Instance, page 137](#)
- [Create and Bind an Analytics Runtime Service Instance, page 141](#)
- [Updating the OAuth 2 Client, page 144](#)

## Exercise 1: Create and Bind an Analytics Catalog Service Instance

### Overview

Before you begin this exercise, make sure that:

- an instance of the UAA service has been configured as your trusted issuer.
- an application has been bound to your UAA service instance

### Steps

1. List the services in the Cloud Foundry marketplace.

**cf marketplace (or cf m)**

```
[predix@localhost ~]$ cf m
Getting services from marketplace in org Predix-Training / space Training3 as
OK
```

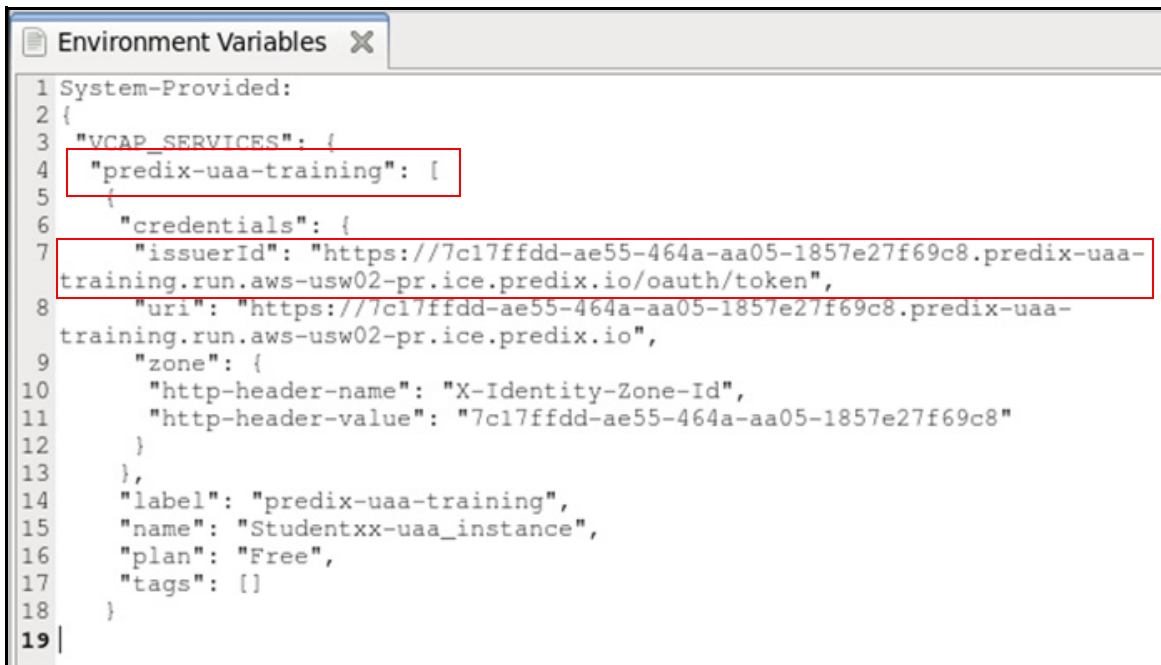
service	plans	description
<b>business-operations</b>	beta	Monetize your
<b>logstash-5</b>	free	Logstash 1.4 s
<b>p-rabbitmq</b>	standard	RabbitMQ is a
<b>p-rabbitmq-35</b>	standard	RabbitMQ is a
<b>pitney-bowes-geoenhancement-service</b>	Beta	Enhance locati
<b>postgres</b>	shared-nr	Reliable Postg
<b>predix-acs</b>	Tiered	Use this servi
<b>predix-acs-training</b>	Basic, Free	Design precise
<b>predix-analytics-catalog</b>	Bronze, Silver*, Gold*	Add analytics
<b>predix-analytics-runtime</b>	Bronze, Silver*, Gold*	Use this servi
<b>predix-analytics-ui</b>	Free	Use this brows
<b>predix-asset</b>	Tiered	Create and sto
<b>predix-blobstore</b>	Tiered	Use this binar

The Analytics Catalog service, `predix-analytics-catalog`, is listed as one of the available services.



## 2. View and record your your UAA environment variables.

- In an earlier lab, you bound an application to your UAA service instance. Use the following syntax to view the UAA environment variables associated with that application:
  - cf env <app\_name>**
- Copy the environment variables from the Terminal into a new file in your text editor
  - Your file will look similar to the graphic below and you will use the Issuer ID when you create your analytics catalog service instance (next step).



```

1 System-Provided:
2 {
3   "VCAP_SERVICES": {
4     "predix-uaa-training": [
5       {
6         "credentials": {
7           "issuerId": "https://7c17ffdd-ae55-464a-aa05-1857e27f69c8.predix-uaa-
8 training.run.aws-usw02-pr.ice.predix.io/oauth/token",
9           "uri": "https://7c17ffdd-ae55-464a-aa05-1857e27f69c8.predix-uaa-
10 training.run.aws-usw02-pr.ice.predix.io",
11           "zone": {
12             "http-header-name": "X-Identity-Zone-Id",
13             "http-header-value": "7c17ffdd-ae55-464a-aa05-1857e27f69c8"
14           }
15         },
16         "label": "predix-uaa-training",
17         "name": "Studentxx-uaa_instance",
18         "plan": "Free",
19         "tags": []
20       }
21     ]
22   }
23 }

```

### 3. Create your catalog service instance.

- Use the following syntax to create your analytics catalog service instance:

```
cf create-service predix-analytics-catalog <plan>
<my_catalog_instance> -c
'{"trustedIssuerIds":["<uaa_instance1_issuerId>"]}'
```

where:

- <plan> is the pricing plan associated with a service.
- <my\_catalog\_instance> is the name of your analytics catalog service instance.
- <uaa\_instance\_issuerId> is the issuerId of your UAA service instance (refer to your gedit text file).

For example:

```
[predix@localhost ~]$ cf cs predix-analytics-catalog Bronze analytics-catalog-Firstname75 -c '{"trustedIssuerIds":["https://a97db685-e3f5-4f10-ad03-f82bee5a3808.predix-uaa-training.run.aws-usw02-pr.ice.predix.io/oauth/token"]}'
Creating service instance analytics-catalog-Firstname75 in org Predix-Training / space Training3 as student75...
OK
```

#### 4. Bind your application to your Analytics Catalog Service Instance.

**Note:** You must bind your Analytics Catalog service instance to your application to provision connection details for your Analytics Catalog service instance in the VCAP\_SERVICES environment variable

- From your terminal:

```
cf bind-service (or cf bs) <app_name> <my_catalog_instance>
```

For example:

```
[predix@localhost ~]$ cf bs hello-world-app-Firstname75 analytics-catalog-Fi  
rstname75  
Binding service analytics-catalog-Firstname75 to app hello-world-app-Firstna  
me75 in org Predix-Training / space Training3 as student75...  
OK  
TIP: Use 'cf restage hello-world-app-Firstname75' to ensure your env variabl  
e changes take effect
```

---

## Exercise 2: Create and Bind an Analytics Runtime Service Instance

---

### Overview

You will follow similar steps to create the analytics runtime service instance.

### Steps

1. Create your Analytics runtime service instance, which will require two issuer IDs.

- Use the following syntax to create your analytics runtime service instance

```
cf create-service predix-analytics-runtime <plan>
<my_runtime_instance> -c
'{"trustedIssuerIds":["<uaa_instance1_issuerId">,"
"<uaa_instance2_issuerId">"]}'
```

where:

- <plan> is the pricing plan associated with a service.
- <my\_runtime\_instance> is the name of your analytics runtime service instance.
- <uaa\_instance1\_issuerId> is the issuerId of your UAA service instance.
- <uaa\_instance2\_issuerId><sup>1</sup>: is the issuerId provided in the `2nd_id.txt` file in your `predix/PredixApps/training_labs/AnalyticsLabFiles` folder

For example:

```
[predix@localhost ~]$ cf create-service predix-analytics-runtime Bronze anal
tps://a97db685-e3f5-4f10-ad03-f82bee5a3808.predix-uaa-training.run.aws-usw02
40e3-965b-5b839e860018.predix-uaa.run.aws-usw02-pr.ice.predix.io/oauth/token
Creating service instance analytics-runtime-Firstname75 in org Predix-Traini
OK
```

- 
1. This second trusted issuer ID is required to run the job scheduler. A "beta" URL is provided.

## 2. Bind your application to your Analytics Runtime Service Instance.

- From your terminal:

```
cf bind-service (or cf bs) <app_name> <my_runtime_instance>
```

For example:

```
[predix@localhost ~]$ cf bs hello-world-app-Firstname75 analytics-runtime-Firstname75
Binding service analytics-runtime-Firstname75 to app hello-world-app-Firstname75 in org Predix-Training / space Training3 as student75...
OK
TIP: Use 'cf restage hello-world-app-Firstname75' to ensure your env variable changes take effect
```

### 3. View and record some key environment variables.

- List the environment variables using the following command:

```
cf env <app_name>
```

In this example, the system returns:

```
[predix@localhost ~]$ cf env hello-world-app-Firstname75
Getting env variables for app hello-world-app-Firstname75 in org Predix-Training / space Training:
OK

System-Provided:
{
  "VCAP_SERVICES": {
    "predix-analytics-catalog": [
      {
        "credentials": {
          "catalog_uri": "https://predix-analytics-catalog-release.run.aws-usw02-pr.ice.predix.io",
          "zone-http-header-name": "Predix-Zone-Id",
          "zone-http-header-value": "0bfcc3b0-4626-4980-bc5b-cc3fbb6543b6",
          "zone-oauth-scope": "analytics.zones.0bfcc3b0-4626-4980-bc5b-cc3fbb6543b6.user"
        },
        "label": "predix-analytics-catalog",
        "name": "analytics-catalog-Firstname75",
        "plan": "Bronze",
        "tags": []
      }
    ],
    "predix-analytics-runtime": [
      {
        "credentials": {
          "config_uri": "https://predix-analytics-config-release.run.aws-usw02-pr.ice.predix.io",
          "execution_uri": "https://predix-analytics-execution-release.run.aws-usw02-pr.ice.predix.io",
          "monitoring_uri": "https://predix-monitoring-service-release.run.aws-usw02-pr.ice.predix.io",
          "scheduler_uri": "https://predix-scheduler-service-release.run.aws-usw02-pr.ice.predix.io",
          "zone-http-header-name": "Predix-Zone-Id",
          "zone-http-header-value": "f76daffd-5cb0-475b-83ab-78dabec4f526",
          "zone-oauth-scope": "analytics.zones.f76daffd-5cb0-475b-83ab-78dabec4f526.user"
        },
        "label": "predix-analytics-runtime",
        "name": "analytics-runtime-Firstname75",
        "plan": "Bronze",
        "tags": []
      }
    ]
  }
}
```

- Copy this text to your gedit text file for future reference.

**Tip:** Be sure to copy all of the environment variables into a file for later use.

---

## Exercise 3: Updating the OAuth 2 Client

---

### Overview

To enable applications to access the Analytics Catalog and Runtime services, your JSON Web Token (JWT) must contain both of the following `zone_auth_scope` values, found in your application environment variables:

```
analytics.zones.<catalog_instance_guid>.user
analytics.zones.<runtime_instance_guid>.user
```

The OAuth2 client uses an authorization grant to request an access token. OAuth2 defines four grant types. Based on the type of authorization grant that you have used, you must update your OAuth2 client to generate the required JWT.

This lab exercise will show you how to edit the scope of the access to include Catalog and Runtime services.

In this exercise you will update the OAuth 2 client with the required scopes to work with the Analytics Catalog and Runtime services.

### Steps

1. Target your UAA service instance.

You will use the UAA command line interface (UAAC) to work with your UAA instance. This has been installed on the DevBox for you and information on this topic is in the Predix.io web site.

- From your Terminal, enter the following command:

```
uaac target <uaa_instance_url>
```

where: `<uaa_instance_url>` is the URL of your UAA service instance (saved in your gedit text file).



## 2. Log into your UAA service instance.

- Log in and verify your default administrator account and password. You generated these when you created your UAA service instance in a prior lab. Refer to the password you wrote down at that time.
  - Use the following command:
 

```
uaac token client get admin
```

 You are prompted for your administrative password (Client secret)
  - Enter the admin password you created at the prompt.
 

The system responds with a message that you have successfully fetched a token via the client credentials grant with a context of admin, from client admin.

## 3. Update the OAuth2 client with the authorities (scopes) required.

- Run the following command:

```
uaac clients
```

You will see the following information based on the client authorities you created in the security lab. You will update (add to) these in the next step:

```
Studentxx-client
  scope: uaa.none
  resource_ids: none
  authorized_grant_types: authorization_code client_credentials
  refresh_token
  autoapprove:
  action: none
  authorities: clients.read acs.policies.read acs.policies.write
  acs.attributes.read clients.write acs.attributes.write
  zones.da38b1c2-8743-418f-94ff-0744f648e671.admin scim.write
  scim.read predix-ac-training.zones.e53aa7f1-62c6-47a7-91f5-
  9f5ae9554dd0.user
  signup_redirect_url:
  lastmodified: 1457723490390
```

- Copy this set of authorities into a new text file in gedit.



- Run the following command:
 

```
uaac client update <client> --authorities "<set_of_authorities>"
```

 where <client> is the name of the uaa client you created in the Security lab.  
 where <set\_of\_authorities> is your existing set of authorities (copied from your Terminal as shown above), plus authorities required for the platform services (analytics catalog and runtime).
- As in the following example:

```
uaac client update client --authorities "clients.read acs.policies.read
acs.policies.write acs.attributes.read clients.write acs.attributes.write
zones.da38b1c2-8743-418f-94ff-0744f648e671.admin scim.write scim.read
predix-ac-training.zones.e53aa7f1-62c6-47a7-91f5-9f5ae9554dd0.user
analytics.zones.0bfcc3b0-4626-4980-bc5b-cc3fbb6543b6.user
analytics.zones.f76daffd-5cb0-475b-83b-78dabec4f526.user"
```

The two highlighted lines above are examples of the OAuth scopes for the catalog and runtime services. You retrieve these from your application's environment variables. Copy these values from the zone-oauth-scope variable(s) under the predix-analytics-catalog and predix-analytics-runtime environment variables (one from each).

**Tip:** Listed authorities have a single space between them. Be sure to include opening and closing quotation marks with the statement. Use your gedit text editor to collect the authorities before entering the final command in the Terminal.

- Run the following command to retrieve the updated set of authorities

```
uaac clients
```

After running the command, you can see that the for the client authorities have been updated to include the additional authorities for your catalog and runtime services.

4. Get the token again with the client credential grant.

- Run the command:
  - `uaac token client get client`
- Validate with UAAC that the scopes were updated in the token by running the command:
  - `uaac token decode`
- You can also look at the scopes using the `uaac clients` command:

```

admin
  scope: uaa.none
  resource_ids: none
  authorized_grant_types: client_credentials
  autoapprove:
  action: none
  authorities: clients.read clients.secret idps.write
              uaa.resource
              zones.2709b971-7042-4782-b58c-d1f1d0f6a28f.admin
              clients.write clients.admin idps.read scim.write
              scim.read
  lastmodified: 1458333847349
client
  scope: uaa.none
  resource_ids: none
  authorized_grant_types: authorization_code
  client_credentials_refresh_token
  autoapprove:
  action: none
  authorities: clients.read acs.policies.read
              acs.policies.write acs.attributes.read
              analytics.zones.de1832af-5a26-41c5-9a17-
              b0ba5b7c809b.user
              zones.2709b971-7042-4782-b58c-d1f1d0f6a28f.admin
              clients.write acs.attributes.write
              analytics.zones.456b2f52-f74b-4430-b7dc-
              f2f89becb549.user scim.write scim.read
  signup_redirect_url:
  lastmodified: 1458345782159
  
```

Note the updated values for scope and also the grant type.

## Part II: Working with the Analytics Catalog through a REST Client

### Overview

By the end of this lab, students will be able to:

- Configure and use Postman (a REST client) to work with the API
- Set up appropriate authorization and access for Analytics Catalog and Runtime
- Create a catalog entry, upload an analytic executable, test and validate the analytic

### Lab Exercises

- [Using Postman and Getting Your UAA Token Value, page 149](#)
- [Working with the Analytics Catalog, page 164](#)



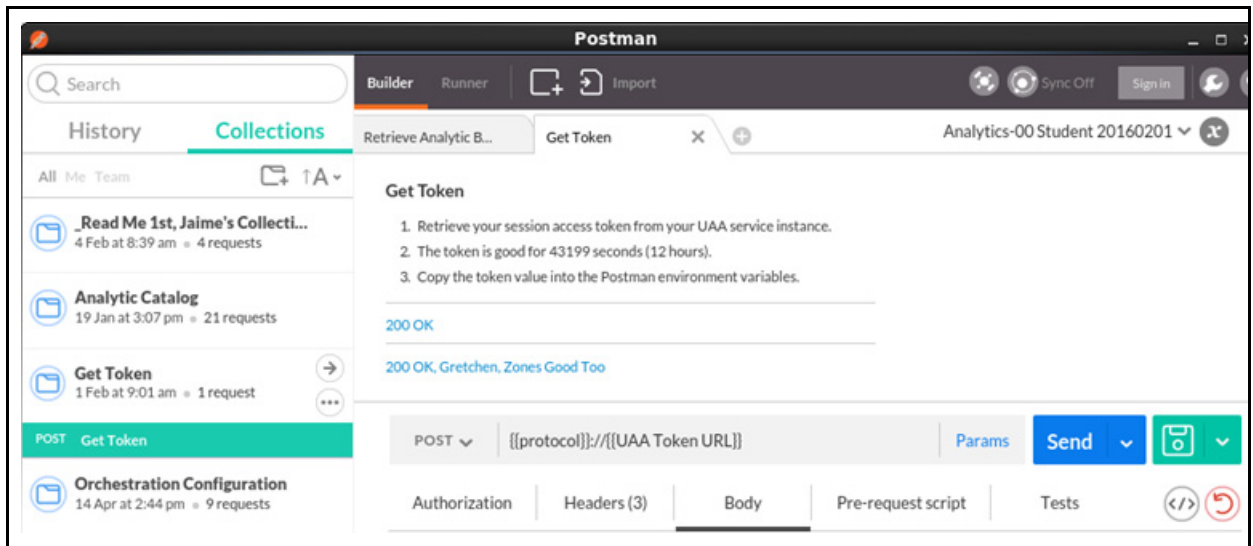
## Exercise 1: Using Postman and Getting Your UAA Token Value

### Overview

In this exercise, you will be using Postman, a Chrome browser extension, which allows you to:

- Create and send any HTTP request using the awesome Postman Builder. Requests are saved to history and can be replayed later.
- Manage and organize your APIs with Postman Collections for a more efficient testing and integration workflow.

A screen shot of Postman follows:



Environment variable values may be viewed when managing the environments.

The screenshot shows the Postman application with a 'Manage environments' dialog box open. The dialog is titled 'Analytics-00 Student 20160201' and contains a list of environment variables. Each variable has a blue checkmark icon to its left and a 'Value' field to its right. The variables and their values are as follows:

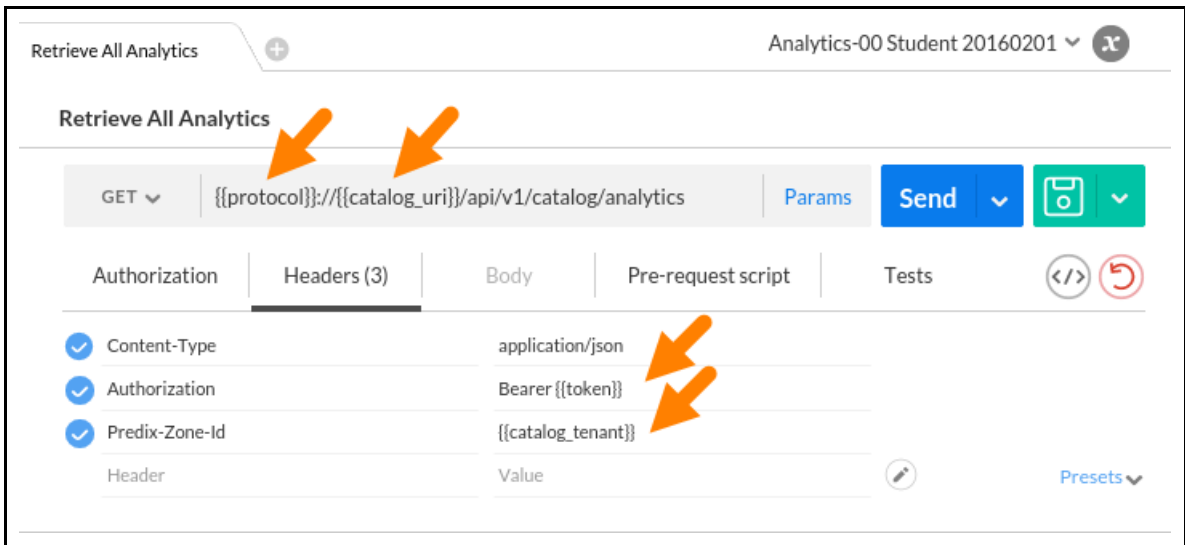
Variable Name	Value
analyticId	Value
analyticName	Value
analyticVersion	Value
artifactId	Value
bpmnXML_content	Value
catalog_tenant	f59bb9e0-da68-45a7-a5c9-bd4a6c74
catalog_uri	predix-analytics-catalog-release.run.a
config_uri	predix-analytics-config-release.run.aw
configurationId	Value
Content-Type	application/json
deploymentrequestId	Value
execution_uri	predix-analytics-execution-release.ru
jobId	Value
protocol	https
runtime_tenant	497a5a2f-bf8f-430c-9f45-bf4d3d4c6
scheduler_uri	predix-scheduler-service-release.run.:
token	eyJhbGciOiJIUzU1NiJ9.eyJqdGkiOiJk
UAA Token URL	5d05f50d-387f-4646-a170-fa85af26'
validationrequestId	Value
X-Identity-Zone-Id	5d05f50d-387f-4646-a170-fa85af26'
Key	Value

At the top right of the dialog, there are 'Back' and 'Submit' buttons. A pencil icon is visible at the bottom right of the variable list, indicating an edit function.

You will use your application environment variables and Postman's Collections manager to configure HTTP requests to your UAA service instance, and to your Analytics Catalog and Runtime service instances.

The example above shows the environment variables needed for the Analytics services. Variable values are referenced using double curly bracket notation (ex: `{{token}}`) in the request (URL, params, header, body). Different variables are used for different types of HTTP requests.

For example:



The variables `protocol`, `catalog_uri`, `token`, and `catalog_tenant` were created with their corresponding values and are maintained in Postman's environment manager.

In this exercise, You retrieve a security token to make REST calls to your Analytics catalog and runtime services. You use Postman to make this call to your UAA service instance.

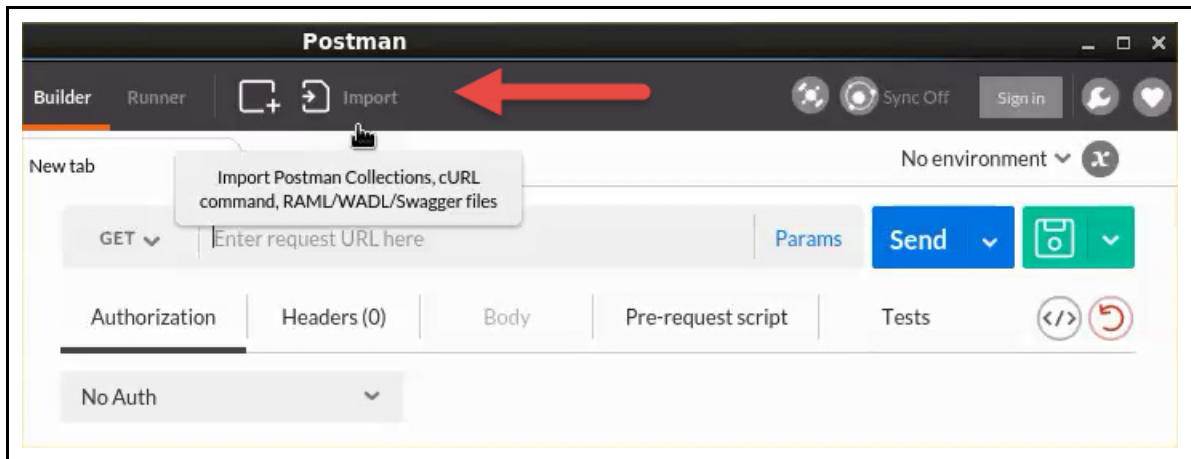
## Steps

1. Open the Postman tool.

- Go to the **Applications** menu in the DevBox
  - ◆ Choose **Chromium Apps**
  - ◆ Click **Postman**

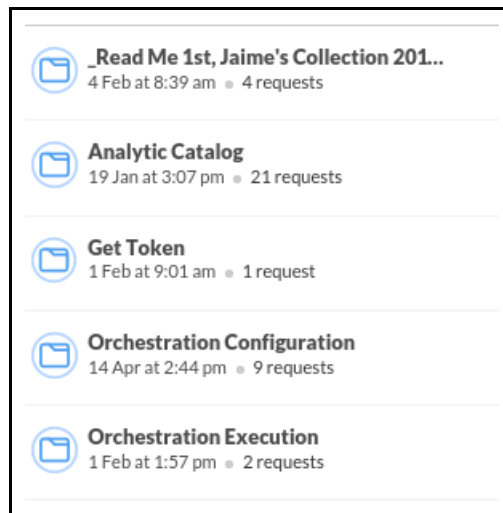
2. Import Postman collections to facilitate API requests.

- Click the **Import** button (top of page) in Postman.
- Click **Choose Files**



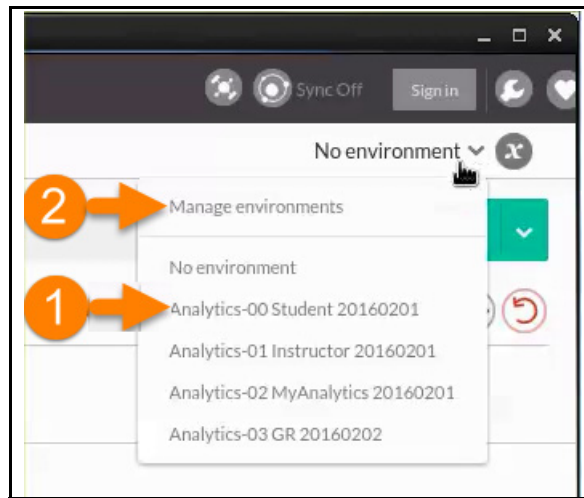
- Browse to `predix/PredixApps/training_labs/AnalyticsLabFiles` for the Postman "dump" file called **`Analytics_Backup_2016-02-04.postman_dump`**
- Click **Open**, then **Import**.
- A Collection already exists window appears. Close the window without selecting either button. If you get two collections with the same name, simply delete one of them.

- Click the *Collections* tab of Postman which should now contain the collections as shown (partial list):



3. Set your Postman environment for Analytics.

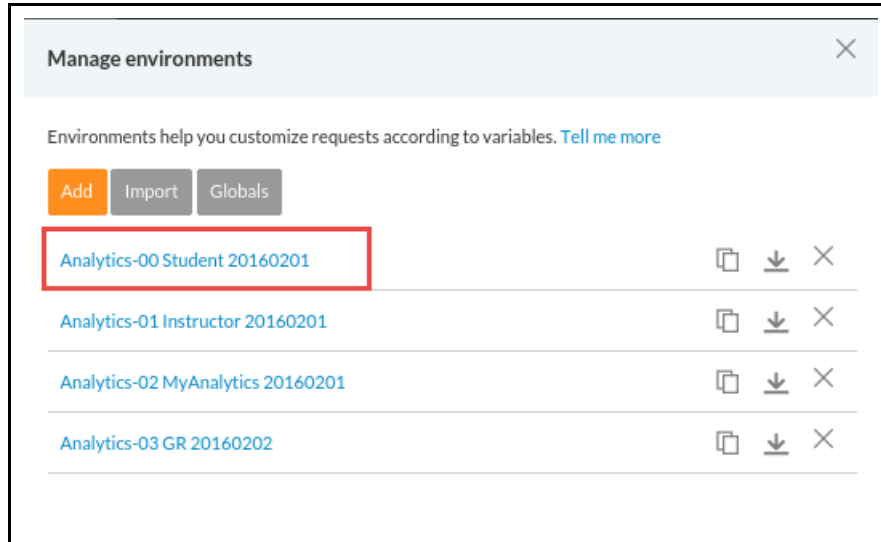
- Select Analytics-00 environment.
- Then, click **Manage Environments**.





#### 4. Set your Postman environment for Analytics.

- Open the Postman Analytics-00 environment variables:



- Set the following variables where:
  - ◆ **protocol:** https (may already be set)
  - ◆ **UAA Token URL:** UAA instance, "issuerId" value/URL (shown below, do not include the "https://")
  - ◆ **X-Identity-Zone-Id:** UAA instance http-header-value

```
"predix-uaa-training": [
  {
    "credentials": {
      "issuerId": "https://a97db685-e3f5-4f10-ad03-f82bee5a3808.predix-uaa-train
      "uri": "https://a97db685-e3f5-4f10-ad03-f82bee5a3808.predix-uaa-training.r
      "zone": {
        "http-header-name": "X-Identity-Zone-Id",
        "http-header-value": "a97db685-e3f5-4f10-ad03-f82bee5a3808"
      }
    }
  },

```

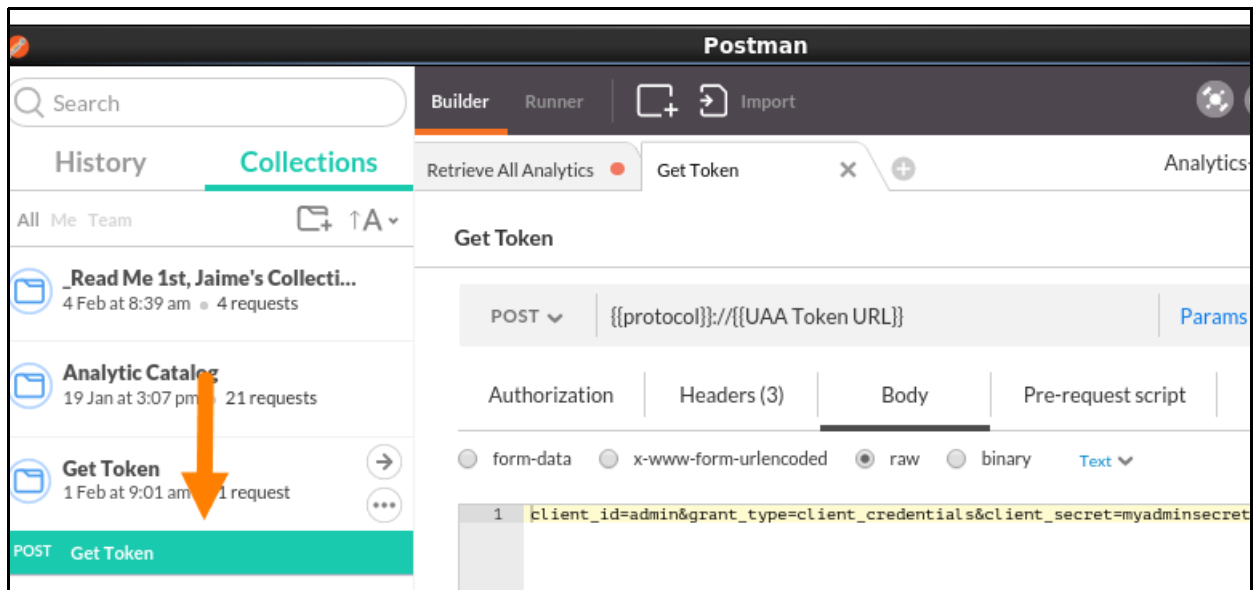
The variables should be set as follows:

✓ protocol	→	https
✓ runtime_tenant		497a5a2f-bf8f-430c-9f45-bf4d3d4c6
✓ scheduler_uri		predix-scheduler-service-release.run.:
✓ token		eyJhbGciOiJSUzI1NiJ9.eyJqdGkiOiJk
✓ UAA Token URL	→	a97db685-e3f5-4f10-ad03-f82bee5a:
✓ validationrequestId		Value
✓ X-Identity-Zone-Id	→	a97db685-e3f5-4f10-ad03-f82bee5a:

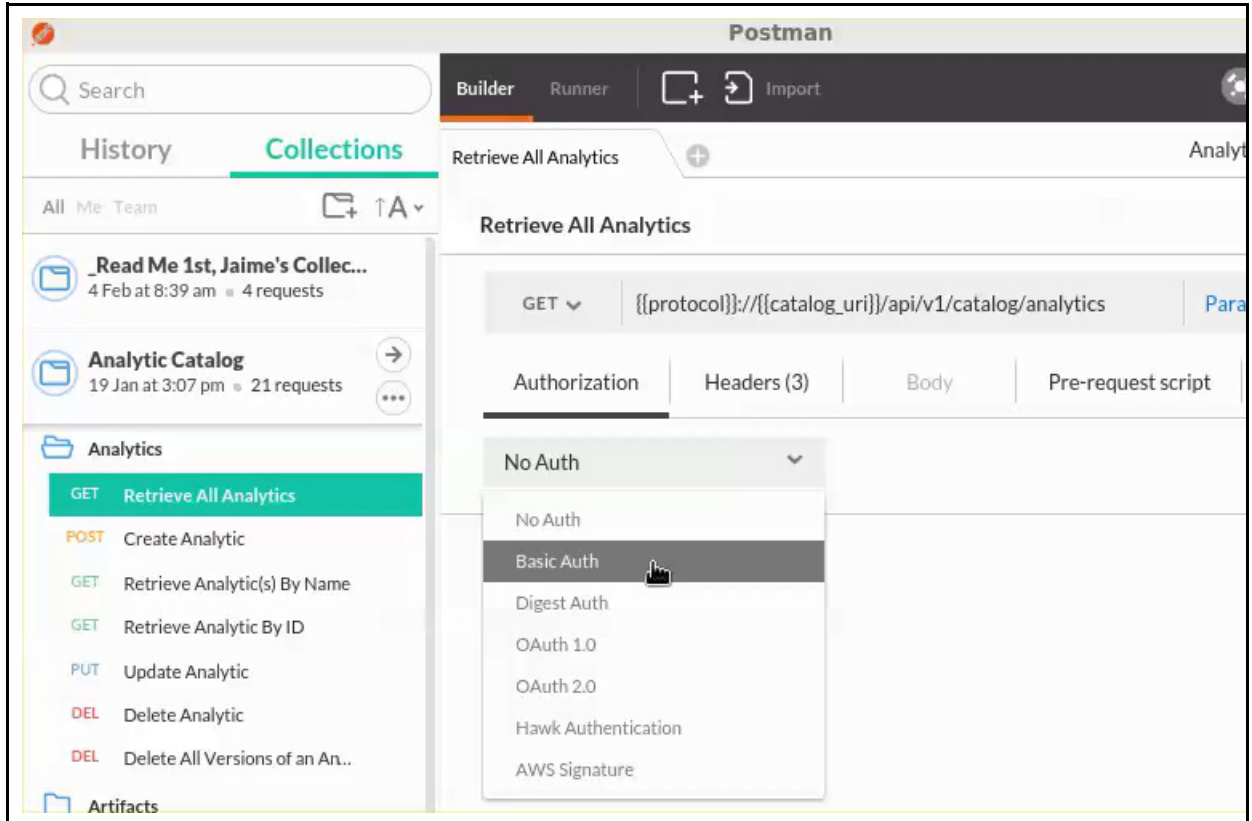
- ◆ Click **Update**
- ◆ Close the **Manage environments** window

5. Create the **Get Token** request for your UAA client in Postman.

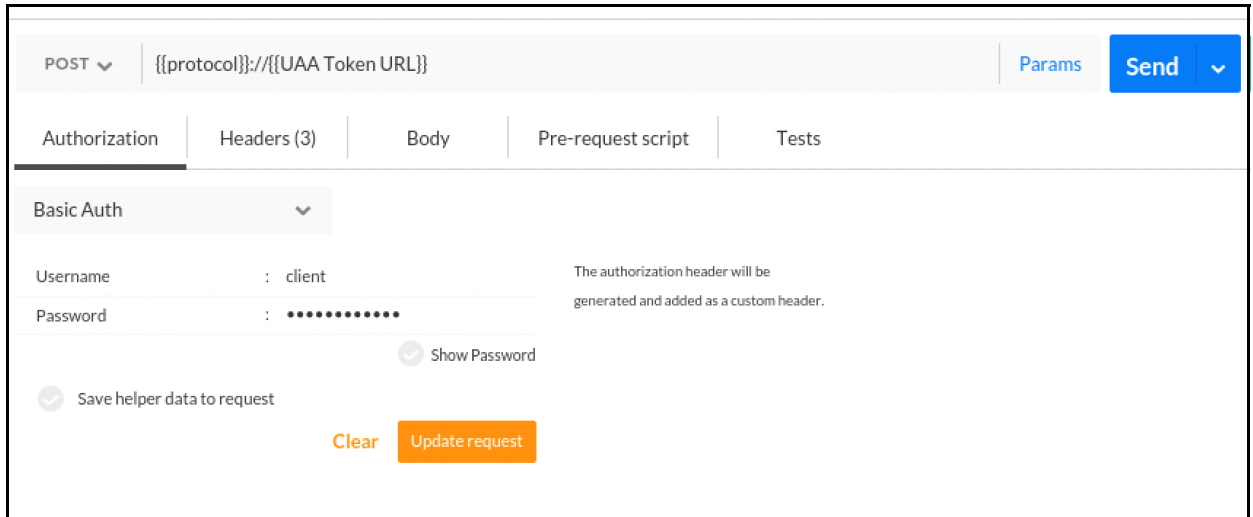
- Expand **Get Token** under Collections
- Select **POST Get Token**



- Enter the UAA client authorization data
  - ◆ Click the **Authorization** tab
  - ◆ Select **Basic Authentication** under the **Type** menu



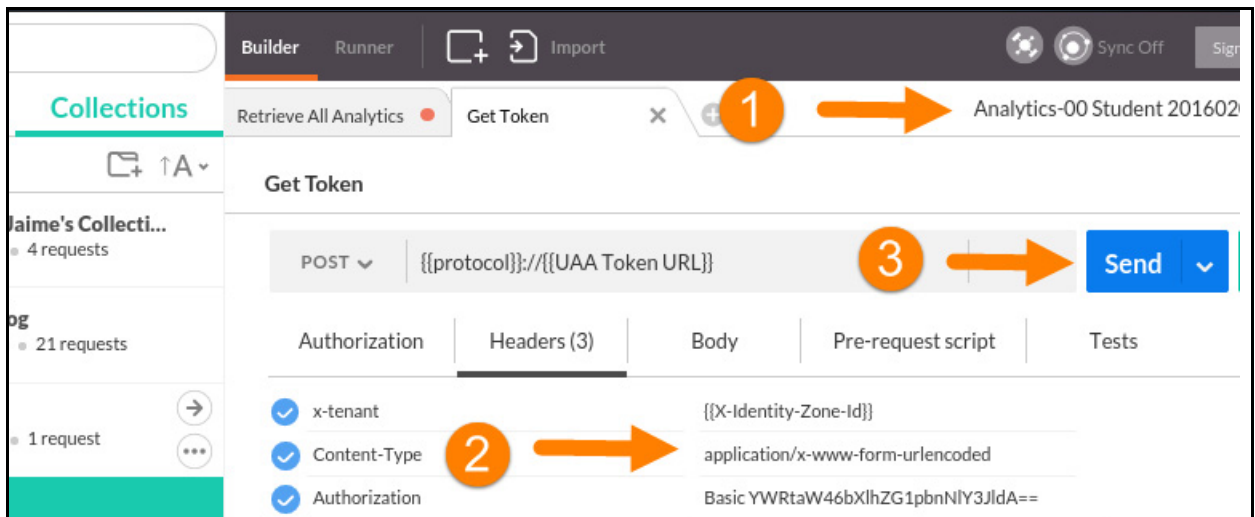
- ◆ Change the body of the request by removing the client ID and password (leave in grant types)
- ◆ Enter:
  - Username: `<UAA_client>`
  - Password: `<client_password>`
- ◆ Click **Update request**



This will add the Authorization header to your token request.

- Verify Header data

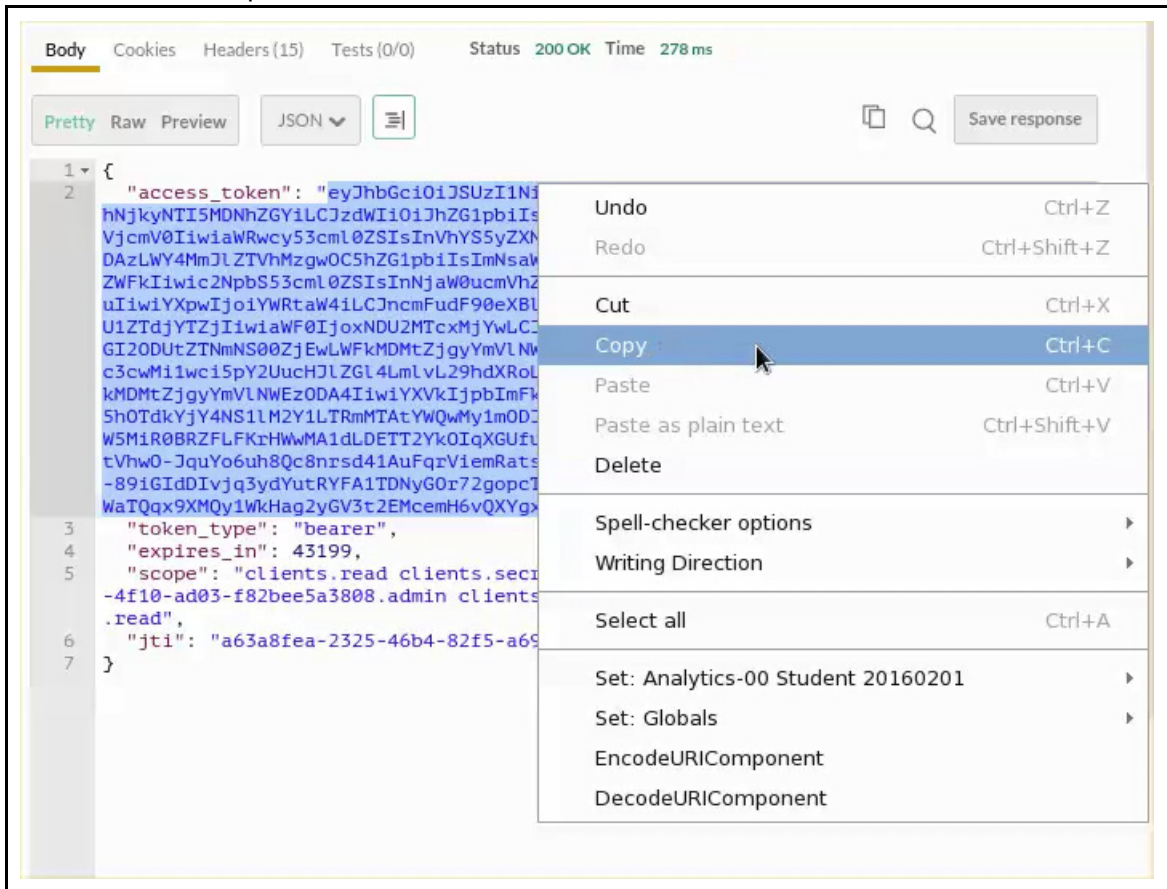
1. Ensure your Postman is in the Analytics-00 environment
2. Drag and drop your headers so they appear in the following order
3. Click **Send**



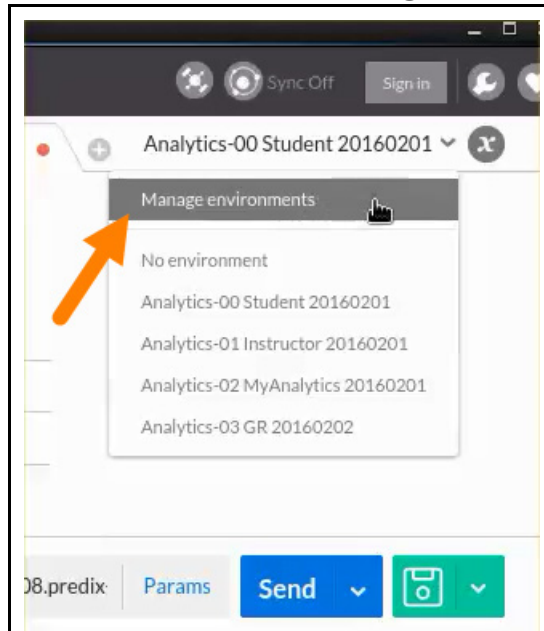
This should result with a **Status** of **200 OK**.



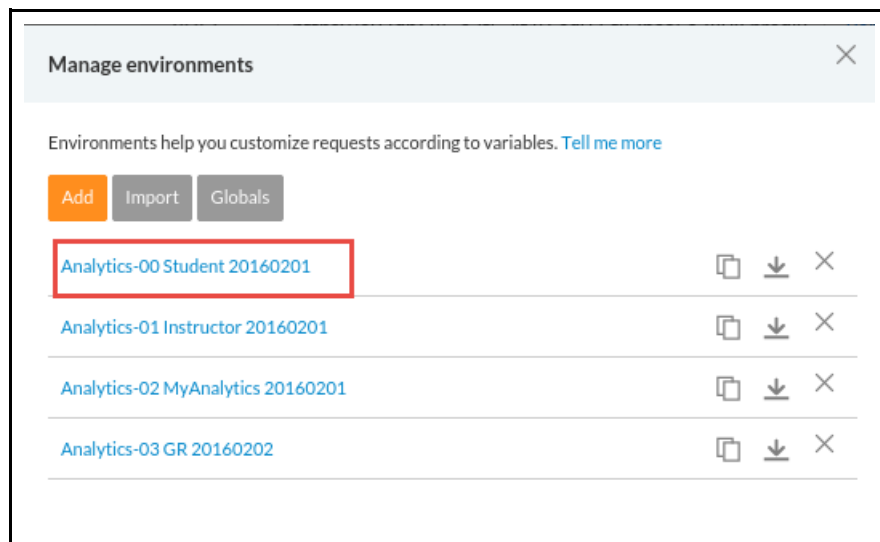
The **access\_token** represents the UAA token value. Copy the text between the **access\_token** quotes.



- Paste the **access\_token** value into the Postman variable “token”.
- ◆ Click **Analytics-00 Student 20160201/Manage environments**



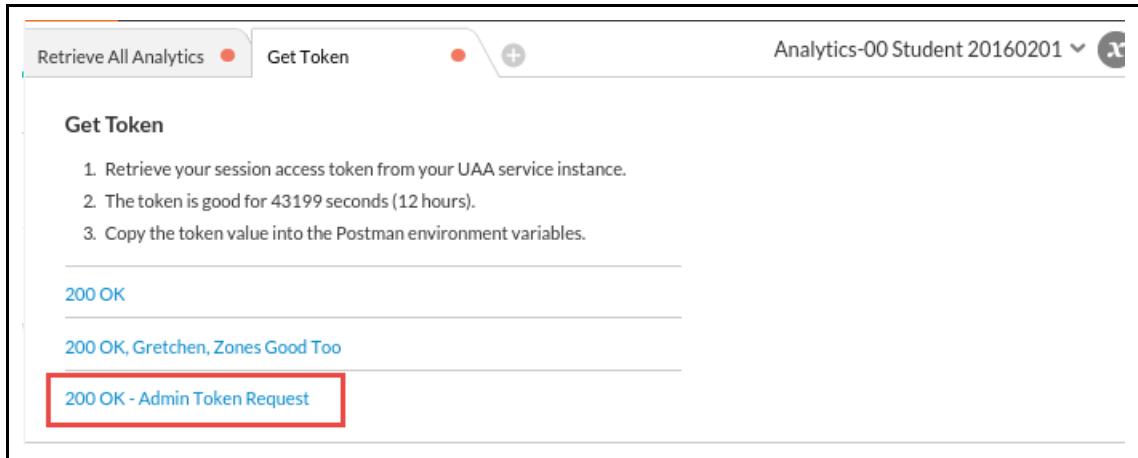
- ◆ Click **Analytics-00 Student 20160201**







- Click the Saved Response link to view



The screenshot shows a Postman interface with two tabs: 'Retrieve All Analytics' and 'Get Token'. The 'Get Token' tab is active. Below the tabs, the title 'Get Token' is followed by a list of instructions:

1. Retrieve your session access token from your UAA service instance.
2. The token is good for 43199 seconds (12 hours).
3. Copy the token value into the Postman environment variables.

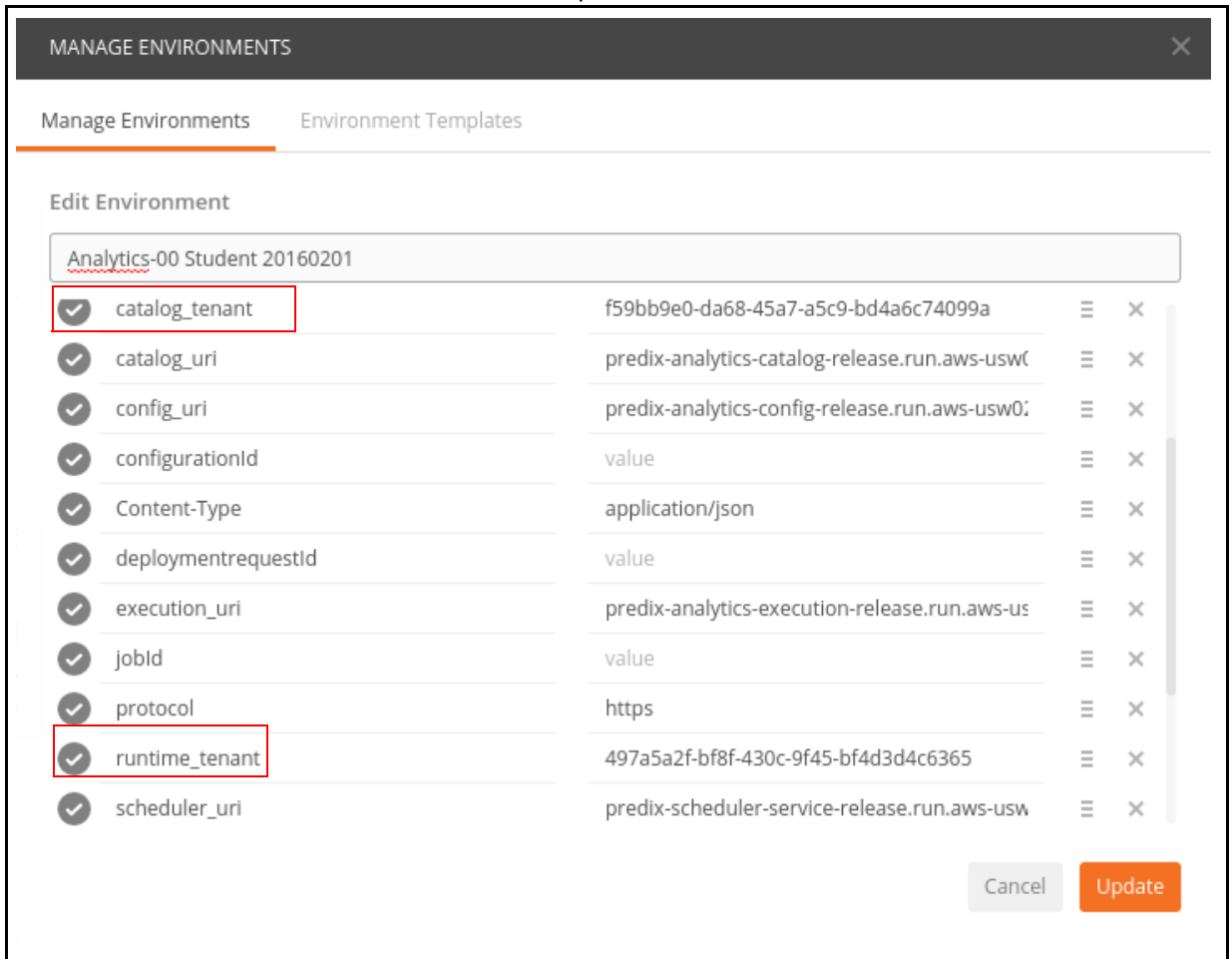
Below the instructions, there is a list of response history items:

- 200 OK
- 200 OK, Gretchen, Zones Good Too
- 200 OK - Admin Token Request

The '200 OK - Admin Token Request' item is highlighted with a red rectangular box.

6. Enter all the required Postman environment variables.

- To facilitate the other requests, set up the following environment variables:
  - ◆ catalog\_tenant (catalog - zone-http-header-value)
  - ◆ runtime\_tenant (runtime - zone-http-header-value)



- Set **Content-type**: application/json.
- Click **Submit**.

Note that other variables have been set for you: catalog\_uri, config\_uri, execution\_uri, and scheduler\_uri.

## Exercise 2: Working with the Analytics Catalog

### Overview

For this lab exercise, you will work with a simple Analytic application demo, upload it to the catalog, then deploy and test it. The application adds two numbers together.

The following steps describe the process for adding a new analytic to the catalog.

- Create the analytic.
- Upload the analytic to the catalog.
- Deploy and test the analytic in the Cloud Foundry environment

### Steps

1. Get all analytic catalog entries.

- If necessary, retrieve a new access token and paste it into the Token variable value for your environment.
- Add a new Postman tab (window) and select **GET Retrieve All Analytics** from the **Analytic Catalog>Analytics** folder structure

The screenshot shows the Postman interface with the 'Collections' tab active. The left sidebar shows a folder structure: 'Analytics' > 'Retrieve All Analytics' (selected). The main area displays the endpoint configuration for 'Retrieve All Analytics'.

**Endpoint:** GET `{{protocol}}://{{catalog_uri}}/api/v1/catalog/analytics`

Authorization	Headers (3)	Body	Pre-request script
	<ul style="list-style-type: none"> <li>✓ Content-Type: application/json</li> <li>✓ Authorization: Bearer {{token}}</li> <li>✓ Predix-Zone-Id: {{catalog_tenant}}</li> </ul>		
Header		Value	

- Click **Send** and you should receive a response status of 200 OK.

The screenshot shows the Postman interface with the 'Collections' tab active. A request named 'Retrieve All Analytics' is selected. The URL is `{{protocol}}://{{catalog_uri}}/api/v1/catalog/analyt`. The authorization is set to 'No Auth'. The response status is '200 OK' and the time taken is '1523 ms'. The response body is shown in 'Pretty' format as JSON:

```

1 {
2   "analyticCatalogEntries": [],
3   "totalPages": 0,
4   "currentPageNumber": 0,
5   "totalElements": 0,
6   "currentPageSize": 0,
7   "maximumPageSize": 25
8 }

```

- Review the body of the response, and note that there are no analytics entries as of yet. In the next steps, you will create a catalog entry and upload your sample analytic application.
- Save (and update) this GET request to your Postman collection.

## 2. Create a catalog entry for your sample Analytic application.

- Add a new Postman tab (window) and select **POST Create Analytic** from your Postman collections.
- In the body of the request, replace existing text with the following:

```

1 {
2   "name": "Demo Adder Java 1",
3   "version": "v1",
4   "supportedLanguage": "Java",
5   "taxonomyLocation": "",
6   "author": "demo",
7   "description": "This analytic adds two numbers and returns the result.",
8   "customMetadata": "{\"assetid\": \"abc\"}"
9 }

```

- Click **Send**, scroll down and you should see the following response:

```

1 {
2   "author": "demo",
3   "taxonomyLocation": "/uncategorized",
4   "supportedLanguage": "Java",
5   "createdTimestamp": "2016-02-23T04:17:28+00:00",
6   "updatedTimestamp": "2016-02-23T04:17:28+00:00",
7   "customMetadata": "{\"assetid\": \"abc\"}",
8   "version": "v1",
9   "description": "This analytic adds two numbers and returns the result.",
10  "name": "Demo Adder Java 1",
11  "id": "a14dae0b-17a7-40a3-b0a8-b0dbe59c4454"
12 }

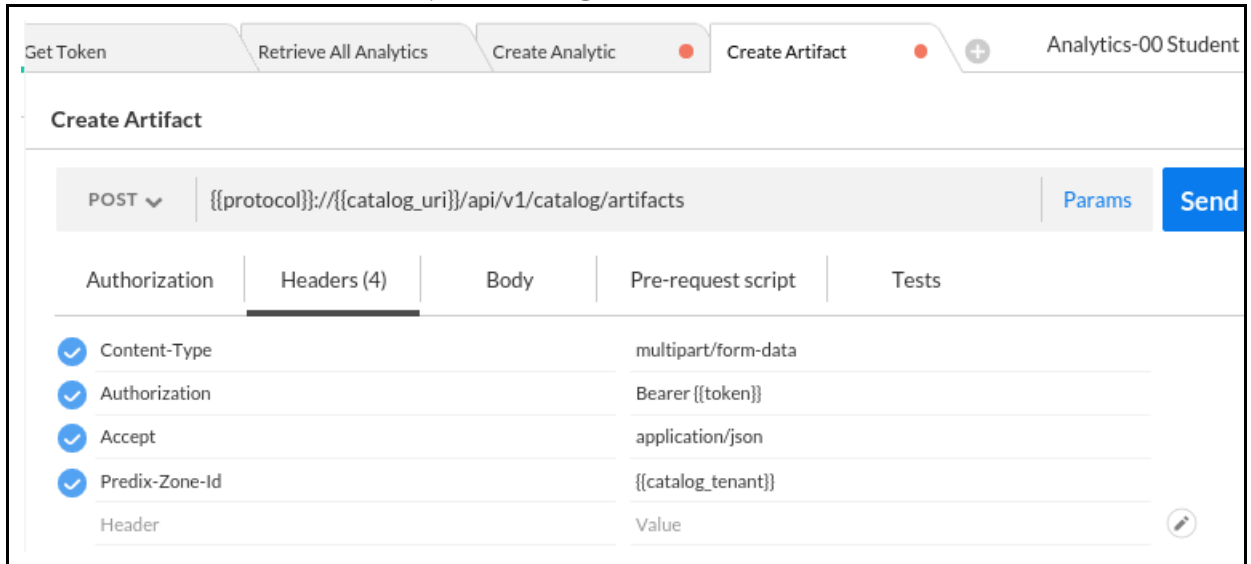
```

- Note the **id** value created for the catalog entry (Catalog Entry ID) in the next step.
- Optionally, Save (and update) this request to your Postman collection.

3. Attach an executable artifact.

In this step, you will upload the sample analytic Java application to the Analytics Catalog.

- Add a new Postman tab (window) and select **POST Create Artifact** from your Postman collections under the **Analytic Catalog>Artifacts** folder.



- In Postman update the variable: **analyticId** (with the Catalog Entry ID) from the prior step response
  - ◆ In the Body of the request, set or verify parameters as follows:
    - type: **Executable**
    - description: **This analytic adds 2 numbers and provides the sum.**

#### 4. Upload an artifact and attach it to an analytic catalog entry.

- Click **Choose Files** to upload an artifact and attach it to an analytic catalog entry.
- Navigate to `predix/PredixApps/training_labs/AnalyticsLabFiles` and choose the `demo-adder-java-1.0.0.jar` file.
- Click **Open** to upload it for this catalog entry.

The screenshot displays the 'Create Artifact' interface. At the top, there are tabs for 'Get Token', 'Retrieve All Analytics', 'Create Analytic', and 'Create Artifact'. The 'Create Artifact' tab is active. Below the tabs, the URL is set to `{{protocol}}://{{catalog_uri}}/api/v1/catalog/artifacts`. The request method is 'POST'. The body is configured as 'form-data' with the following fields:

Field	Value	Type
file	demo-adder-java-1.0.0.jar	File
catalogEntryId	{{analyticid}}	Text
type	Executable	Text
description	This analytic adds 2 numbers and provides the sum.	Text
Key	Value	Text

5. Click **Send** to begin the upload.

- You should receive the following response:

The screenshot shows a REST client interface with the following details:

- Body: Cookies Headers (14) Tests (0/0) Status 201 Created Time 2417 ms
- View options: Pretty (selected), Raw, Preview, JSON (dropdown), and a menu icon.
- Response body (lines 1-8):
 

```

1 {
2   "createdTimestamp": "2016-02-23T04:49:12+00:00",
3   "updatedTimestamp": "2016-02-23T04:49:12+00:00",
4   "filename": "demo-adder-java-1.0.0.jar",
5   "description": "This analytic adds 2 numbers and provides the sum.",
6   "id": "cef31ec1-7bee-440f-8534-8bf62eb48424",
7   "type": "Executable"
8 }
      
```

- Copy the “id” value of the artifact and paste it into your Postman environment **artifactId** variable.
- Optionally, save this request to your Postman collection, by clicking on the disk icon. This updates your collection request with the current parameters.

6. Deploy and validate the analytic.

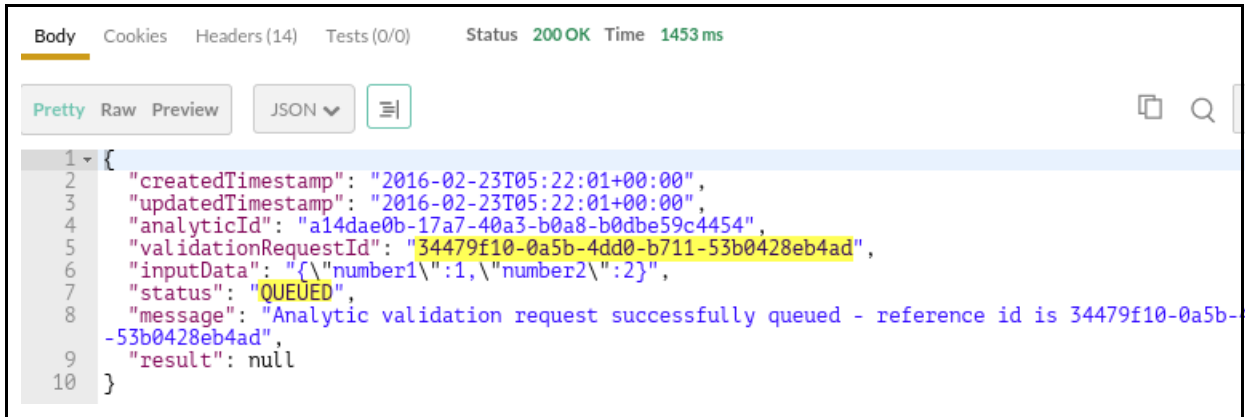
- Add a new Postman tab (window) and select **POST Validate Analytic** from the **Validation/Deployment/Execution** folder of your Postman collections.
- In the Body section, for <value1>, <value2>: Insert any two numbers you would like to add.

The screenshot shows a REST client interface with the following details:

- Method: POST
- URL: `{{protocol}}://{{catalog_uri}}/api/v1/catalog/analytics/{{analyticId}}/validation`
- Buttons: Params, Send (dropdown)
- Section: Authorization | Headers (3) | Body (selected) | Pre-request script | Tests
- Body type: form-data | x-www-form-urlencoded | raw (selected) | binary | JSON (application/json) (dropdown)
- Request body (line 1): `{ "number1": 1, "number2": 2 }`

- Click **Send**, and you should receive the following response:





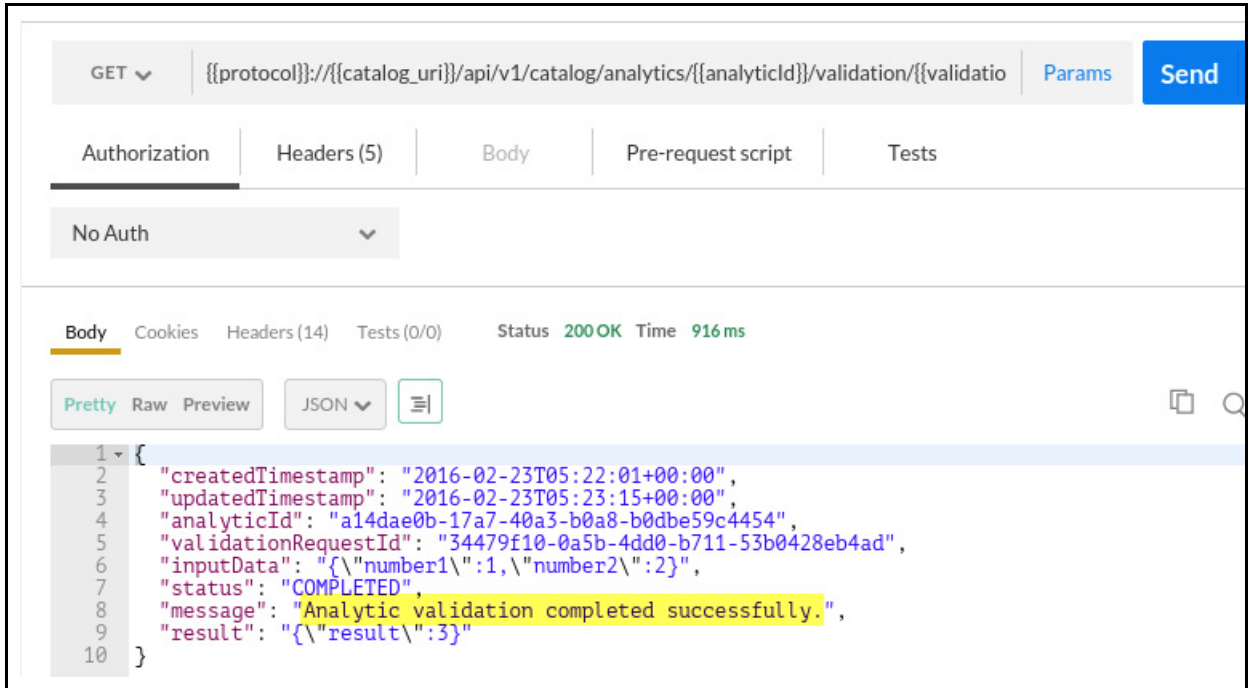
The screenshot shows a web browser's developer console with the 'Body' tab selected. The status is '200 OK' and the time taken is '1453 ms'. The response is displayed in 'Pretty' format as a JSON object. The 'validationRequestId' field is highlighted in yellow.

```
1 {
2   "createdTimestamp": "2016-02-23T05:22:01+00:00",
3   "updatedTimestamp": "2016-02-23T05:22:01+00:00",
4   "analyticId": "a14dae0b-17a7-40a3-b0a8-b0dbe59c4454",
5   "validationRequestId": "34479f10-0a5b-4dd0-b711-53b0428eb4ad",
6   "inputData": "{\"number1\":1,\"number2\":2}",
7   "status": "QUEUED",
8   "message": "Analytic validation request successfully queued - reference id is 34479f10-0a5b-53b0428eb4ad",
9   "result": null
10 }
```

- Copy and paste the **validationRequestId** to your Postman environment variables

7. Poll for the validation status.

- Add a new Postman tab (window) and select **GET Retrieve Validation Status** from the **Validation/Deployment/Execution** folder of your Postman collections.
- Click Send, and you should receive the following:



**Note:** You may need to “Send” or poll it a few times to get a “completed” status.

## Exercise Summary

In this exercise you learned how to:

- Create an Analytic Catalog entry.
- Upload the analytic artifact to the catalog for the assigned entry ID.
- Deploy and test the analytic in the Cloud Foundry environment.

## Part III: Runtime Orchestrations and Job Schedules

### Learning Objectives

By the end of this lab, students will be able to:

- Understand and create an orchestration of an analytic
- Create, test and validate a job schedule for an analytic

### Lab Exercises

- [Running an Orchestration with One Analytic, page 173](#)
- [Schedule Orchestration and Analytic Execution \(Scheduling a Job\), page 181](#)



---

## Exercise 1: Running an Orchestration with One Analytic

---

### Overview

You will use the following information, from the previous lab, to run an orchestration:

- Analytic catalog ID
- Analytic name
- Analytic version

The following exercise walks you through the process for running an orchestration with one analytic.



To facilitate learning, your Postman Collections under Orchestration Execution, there are two requests:

- **POST Run (Using sample bpmnXML)**  
Is available for the student to closely examine the sample bpmnXML workflow and sample orchestration request text by allowing the student to view and edit both items.
- **POST Run (Using environment variables)**  
Is available for the student to simply run a “working” orchestration that relies on the environment variables: AnalyticId, AnalyticName, and AnalyticVersion.
- It is useful to have a working version for testing, validation, and comparison purposes of both requests.
- The following steps are for use with the **POST Run (Using sample bpmnXML)** request.

## Steps

1. Get the ID, name, and version of the analytic to run an orchestration.

- Add a new Postman tab (window) and select **GET Retrieve All Analytics** from the **Analytic Catalog/Analytics** folder of your Postman collections.
- Click **Send**.

```

1  {
2  "analyticCatalogEntries": [
3  {
4    "updatedTimestamp": "2016-02-23T04:17:28+00:00",
5    "author": "demo",
6    "createdTimestamp": "2016-02-23T04:17:28+00:00",
7    "supportedLanguage": "Java",
8    "customMetadata": "{\"assetid\":\"abc\"}",
9    "taxonomyLocation": "/uncategorized",
10   "version": "v1",
11   "description": "This analytic adds two numbers and returns the result.",
12   "name": "Demo Adder Java 1",
13   "id": "a14dae0b-17a7-40a3-b0a8-b0dbe59c4454"
14  }
  ]
}

```

For this example, we are interested in the following Analytic info:

```

"version": <v1>,
"name": <demo adder java 1>,
"id": <a14dae0b-17a7-40a3-b0a8-b0dbe59c4454>

```

- If not already done, add these values to your Postman environment variables.
  - ◆ **Analytics-00 Student/Manage environments**
    - version: analyticVersion
    - name: analyticName
    - id: analyticId

2. Copy the sample BPMN workflow file with gedit.

- Go to Predix.io and copy the sample BPMN workflow file at: <https://www.predix.io/docs#ODGDJxqF> (See step 1 of Procedure of this web page)
- Copy or download **Running an Orchestration with One Analytic**.

3. Using gedit, paste and edit the sample BPMN workflow file.

- Replace <Analytic Catalog Entry Id>, <Analytic Name> and <Analytic Version> with the analytic catalog entry ID, analytic name, and analytic version from the previous step.

### Before editing:

```

9      <process id="OrchestrationWithOneAnalytic" isExecutable="true">
10
11         <startEvent id="sid-start-event"
12             name="">
13             <outgoing>sid-flow1</outgoing>
14         </startEvent>
15
16         <serviceTask completionQuantity="1"
17             id="sid-10001"
18             isForCompensation="false"
19             name="<Analytic Catalog Entry Id>::<Analytic Name>::<Analytic>"
20             startQuantity="1"
21             activiti:delegateExpression="\${ javaDelegate}"
22             xmlns:activiti="http://activiti.org/bpmn">
23             <incoming>sid-flow1</incoming>
24             <outgoing>sid-flow2</outgoing>
25         </serviceTask>
26

```

**After editing:**

```

9   <process id="OrchestrationWithOneAnalytic" isExecutable="true">
10
11   <startEvent id="sid-start-event"
12             name="">
13     <outgoing>sid-flow1</outgoing>
14   </startEvent>
15
16   <serviceTask completionQuantity="1"
17               id="sid-10001"
18               isForCompensation="false"
19               name="a14dae0b-17a7-40a3-b0a8-b0dbe59c4454::Demo Adder Java 1::v1"
20               startQuantity="1"
21               activiti:delegateExpression="\${ javaDelegate}"
22               xmlns:activiti="http://activiti.org/bpmn">
23     <incoming>sid-flow1</incoming>
24     <outgoing>sid-flow2</outgoing>
25   </serviceTask>

```

- Save the file as **Sample BPMN Workflow**

## 4. Send an orchestration execution request.

- Add a new Postman tab (window) and select **POST Run (Using Sample bpmnXML)** from the **Orchestration Execution** Postman collection



POST

Authorization | Headers (4) | **Body** | Pre-request script | Tests

form-data  x-www-form-urlencoded  raw  binary

```

1  {
2  "id": "Execution of Orchestration with One Analytic",
3  "name": "Orchestration with One Analytic",
4  "bpmnXml": "<?xml version='1.0' encoding='UTF-8'>\n<definitions xmlns='http://www.omg.org/spec/BPMN/20100521/xml'>\n  <process id='sid-start-event'>\n    <startEvent id='sid-start-event' name=''>\n      <outgoing flowId='sid-flow1'>\n        <serviceTask id='sid-10001' completionQuantity='1' isForCompensation='false' name='a14dae0b-17a7-40a3-b0a8-b0dbe59c4454::Demo Adder Java 1::v1' startQuantity='1' activiti:delegateExpression='\${ javaDelegate}' xmlns:activiti='http://activiti.org/bpmn'>\n        </serviceTask>\n      </outgoing>\n    </startEvent>\n  </process>\n</definitions>\n  <analytics xmlns='http://www.omg.org/spec/BPMN/20100521/xml'>\n    <analyticStep id='sid-10001'>\n      <inputData>\n        <data>{\n          "number1" : 5,\n          "number2" : 24\n        }\n      </data>\n    </analyticStep>\n  </analytics>\n</definitions>\n</xml>"
5  "analyticInputData": [
6  {
7    "analyticStepId": "sid-10001",
8    "data": "{ \"number1\" : 5, \"number2\" : 24 }"
9  }
10 ]
11 }

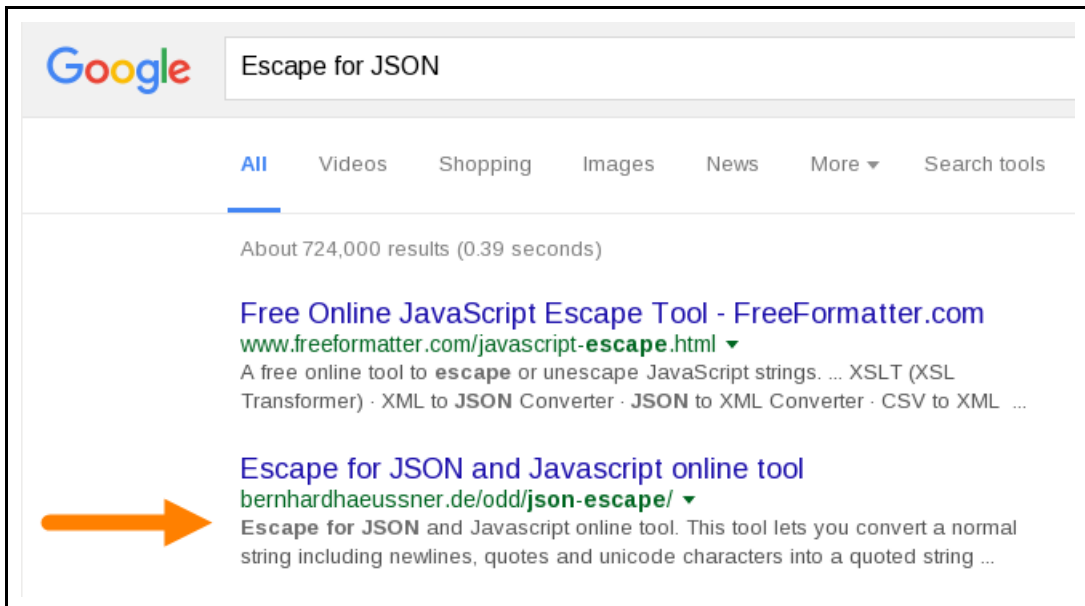
```

The request has two important fields: bpmnXml and analyticInputData.

- **bpmnXml**: you will be replacing the current value with the BPMN workflow XML from the previous step after you “Escape for JSON”. (Step 5)
- **analyticInputData**: is an array of input data for the orchestration. Each element of analyticInputData array represents the input data for the corresponding service task in the BPMN workflow. In the BPMN workflow XML, the service task identified by /definition/process/serviceTask/@id. This service task id is correlated by analyticInputData.analyticStepId in JSON orchestration execution request. The input data for the service task can be specified with analyticInputData.data. (Step 6)

5. Format the BPMN Workflow text using an “Escape for JSON” tool.

- Using Google, search for "Escape for JSON" and select the Escape tool as indicated below.



- Paste the contents of the edited BPMN Workflow text in to the Escape tool, and click **Escape** to convert accordingly.



- Copy the “Escaped” text and paste/replace the contents of the bpmnXml field of the Orchestration Execution Request (body of the Postman **POST Run** from the **Orchestration Execution** window).

## Escape for JSON and Javascript online tool

This tool lets you convert a normal string including newlines, quotes and unicode characters into a quoted string literal ready to use in your JSON or Javascript source. [hide](#)

© Public Domain. NO WARRANTY EXPRESSED OR IMPLIED. USE AT YOUR OWN RISK. Created 2011 by [Bernhard Häußner](#). Escaping code borrowed from Douglas Crockford's [JSON.js](#).

Normal:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
```

↓ escape ↓

Escaped & quoted:

```
"<?xml version=\\"1.0\\" encoding=\\"UTF-8\\"?>\n<definitions
xmlns=\\"http://www.omg.org/spec/BPMN/20100524/MODEL\\" \n
xmlns:xsi=\\"http://www.w3.org/2001/XMLSchema-instance\\" \n
expressionLanguage=\\"http://www.w3.org/1999/XPath\\"
id=\\"sid-81430087-7a44-4be3-8517-914faf923256\\" \n
targetNamespace=\\"DSP-PH\\""
```

- Copy the “Escaped” text.
- Return to Run (Using sample bpmnXML) in Postman, and paste the copied text into the **bpmnXml** field of the orchestration request text (replacing current text)

**Run (Using sample bpmnXML)**

POST `{{protocol}}//{{execution_uri}}/api/v1/execution`

Authorization | **Headers (4)** | Body | Pre-request script | Tests

form-data  x-www-form-urlencoded  raw  binary [JSON \(application/json\)](#) ▼

```

1 {
2   "id": "Execution of Orchestration with One Analytic",
3   "name": "Orchestration with One Analytic",
4   "bpmnXml": "<?xml version='1.0' encoding='UTF-8'?>\n<definitions xmlns='http://www.omg.org/spec/BPMN/
5     {
6       "analyticStepId": "sid-10001",
7       "data": "{ \"number1\" : 5, \"number2\" : 24 }"
8     }
9   }
10 }
  
```

- Be sure to include the quotation marks for the content when replacing and ensure that **only one set of opening and closing quotation marks enclose the field value** as shown in the following image.
- Note that there is a comma at the end of the code.
- Check your Request parameters, and click **Send**

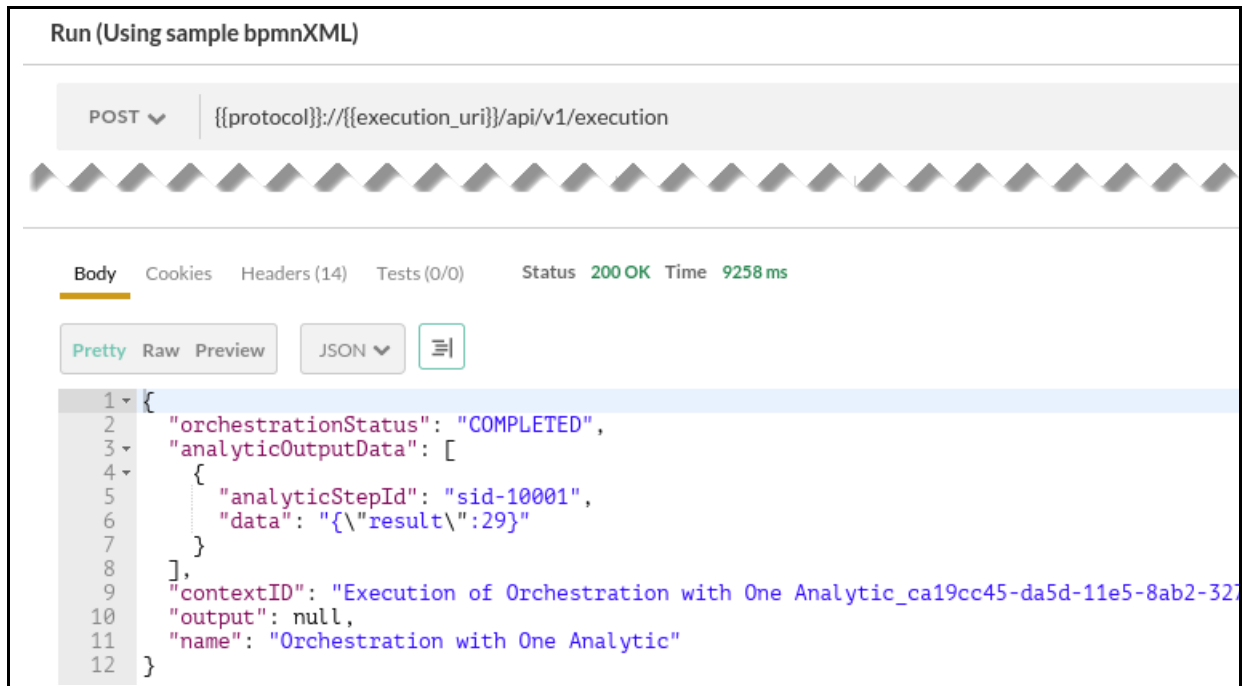
**Run (Using sample bpmnXML)**

POST `{{protocol}}//{{execution_uri}}/api/v1/execution`

Authorization | **Headers (4)** | Body | Pre-request script | Tests

<input checked="" type="checkbox"/> Accept	application/json
<input checked="" type="checkbox"/> Content-Type	application/json
<input checked="" type="checkbox"/> Authorization	Bearer {{token}}
<input checked="" type="checkbox"/> Predix-Zone-Id	{{runtime_tenant}}
Header	Value

- You should receive a response as follows:



The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** `{{protocol}}://{{execution_uri}}/api/v1/execution`
- Status:** 200 OK
- Time:** 9258 ms
- Body:** Pretty (selected), Raw, Preview
- JSON:** JSON (selected)
- Response Body (JSON):**

```
1 {
2   "orchestrationStatus": "COMPLETED",
3   "analyticOutputData": [
4     {
5       "analyticStepId": "sid-10001",
6       "data": "{\"result\":29}"
7     }
8   ],
9   "contextID": "Execution of Orchestration with One Analytic_ca19cc45-da5d-11e5-8ab2-327
10  "output": null,
11  "name": "Orchestration with One Analytic"
12 }
```

- Optionally, save your API Request to your Postman collection (use the "disk" icon).

---

## Exercise 2: Schedule Orchestration and Analytic Execution (Scheduling a Job)

---

### Overview

Scheduling the execution of an orchestration or an individual analytic can be done on time-based intervals.

REST APIs are provided for managing a scheduled analytic or orchestration execution (also called a job). Using these APIs you can perform the following actions.


- Create a job definition
- Retrieve job definitions
- Retrieve job history
- Update job definitions
- Delete existing jobs

When creating or updating a job, you can enable a job execution by setting the job state to Active. To disable a job, set the state to Inactive.

## Steps

1. If needed, add the "scheduler\_uri" of your Runtime instance to your list of API variables.

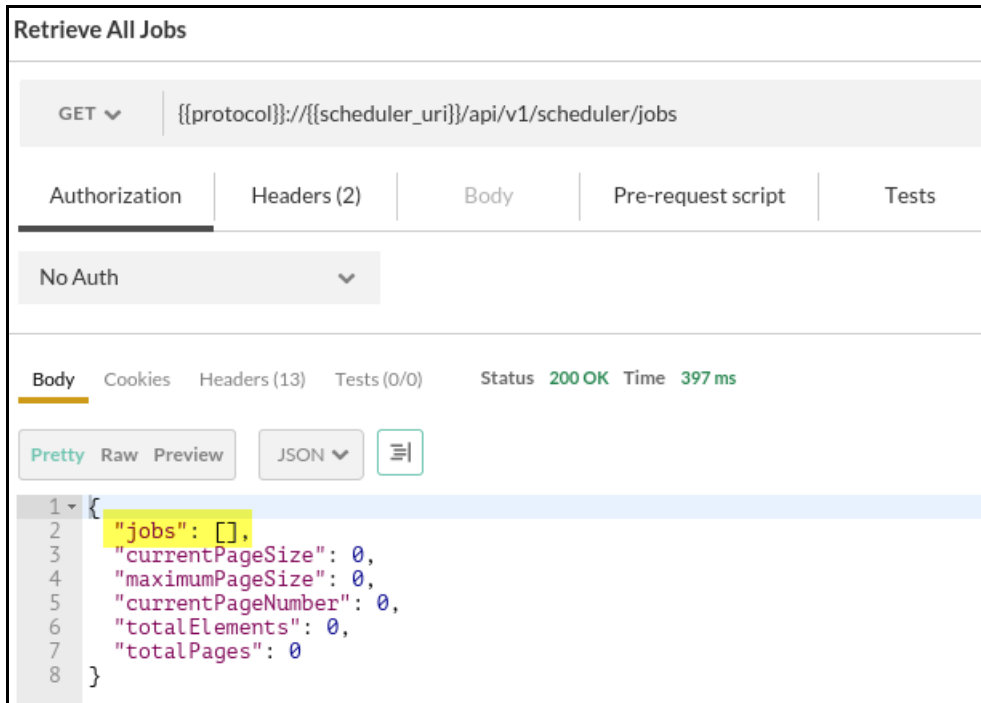
```
"predix-analytics-runtime": [
  {
    "credentials": {
      "config_uri": "https://predix-analytics-config-release.run.aws-usw02-pr.ic
      "execution_uri": "https://predix-analytics-execution-release.run.aws-usw02
      "monitoring_uri": "https://predix-monitoring-service-release.run.aws-usw02
      "scheduler_uri": "https://predix-scheduler-service-release.run.aws-usw02-p
      "zone-http-header-name": "Predix-Zone-Id",
      "zone-http-header-value": "f76daffd-5cb0-475b-83ab-78dabec4f526",
      "zone-oauth-scope": "analytics.zones.f76daffd-5cb0-475b-83ab-78dabec4f526.
    },
    "label": "predix-analytics-runtime",
    "name": "analytics-runtime-Firstname75",
    "plan": "Bronze",
    "tags": []
  }
}
```

<input checked="" type="checkbox"/>	protocol	https
<input checked="" type="checkbox"/>	runtime_tenant	f76daffd-5cb0-475b-83ab-78dabec4f
<input checked="" type="checkbox"/>	scheduler_uri	predix-scheduler-service-release.run.a 
<input checked="" type="checkbox"/>	token	eyJhbGciOiJSUzI1NiJ9.eyJqdGkiOiJjIj
<input checked="" type="checkbox"/>	UAA Token URL	a97db685-e3f5-4f10-ad03-f82bee5a:
<input checked="" type="checkbox"/>	validationrequestId	34479f10-0a5b-4dd0-b711-53b0428
<input checked="" type="checkbox"/>	X-Identity-Zone-Id	a97db685-e3f5-4f10-ad03-f82bee5a:
	Key	Value 

- Click **Submit** to save the variable.

2. Check your scheduler to see if you have jobs ready to run.

- Open a new tabbed window in Postman, and select **Retrieve All Jobs** from the Scheduler Service.
- Click **Send**, and you should receive a response as follows:



Note that you currently do not have jobs scheduled.

### 3. Schedule a job.

Let's take a look at the Postman configuration for creating a job. Add a new Postman tabbed window, and select **POST Create Job** from the Scheduler Service collection.

The screenshot displays the Postman interface with the 'Collections' tab selected. The left sidebar shows a list of collections, with 'Scheduler Service' expanded to show the 'Create Job' endpoint. The main panel shows the configuration for this endpoint:

- Method:** POST
- URL:** `{{protocol}}//{{scheduler_uri}}/api/v1/sche`
- Authorization:** (None)
- Headers (3):** (None)
- Body:** raw

The raw body content is as follows:

```

1 {
2   "name": "test1",
3   "description": "test description",
4   "state": "Active",
5   "cron": {
6     "seconds": "0/5",
7     "minutes": "**",
8     "hours": "**",
9     "dayOfMonth": "**",
10    "months": "**",
11    "dayOfWeek": "?",
12    "years": "**",
13    "timeZoneId": "UTC"
14  },
15  "executionRequest": {
16    "url": "http://some.url",
17    "httpMethod": "GET",
18    "httpHeaders": [
19      {
20        "name": "Content-Type",
21        "value": "application/json"

```

- Edit the text of the body to set up a job schedule to trigger every 2 minutes on exactly every 2 minute mark. The following image shows all the fields that will need editing.

### Create Job

POST {{protocol}}://{{scheduler\_uri}}/api/v1/scheduler/jobs

Authorization | Headers (3) | **Body** | Pre-request script | Tests

form-data
  x-www-form-urlencoded
  raw
  binary
 JSON (application/json) ▾

```

1  [
2  {
3    "name": "test1",
4    "description": "test description",
5    "state": "Active",
6    "cron": {
7      "seconds": "0",
8      "minutes": "0/2",
9      "hours": "**",
10     "dayOfMonth": "**",
11     "months": "**",
12     "dayOfWeek": "?",
13     "years": "**",
14     "timeZoneId": "UTC"
15   },
16   "executionRequest": {
17     "url": "https://predix-analytics-execution-release.run.aws-usw02-pr.ice.predix.io",
18     "httpMethod": "POST",
19     "httpHeaders": [
20       {
21         "name": "Content-Type",
22         "value": "application/json"
23       },
24       {
25         "name": "Accept",
26         "value": "application/json"
27       },
28       {
29         "name": "Predix-Zone-Id",
30         "value": "{{runtime_tenant}}"
31       }
32     ],
33     "inputData": "{\n  \"id\": \"Execution of Orchestration with One Analytic\",\n  \"name\": \"
34   }
        
```

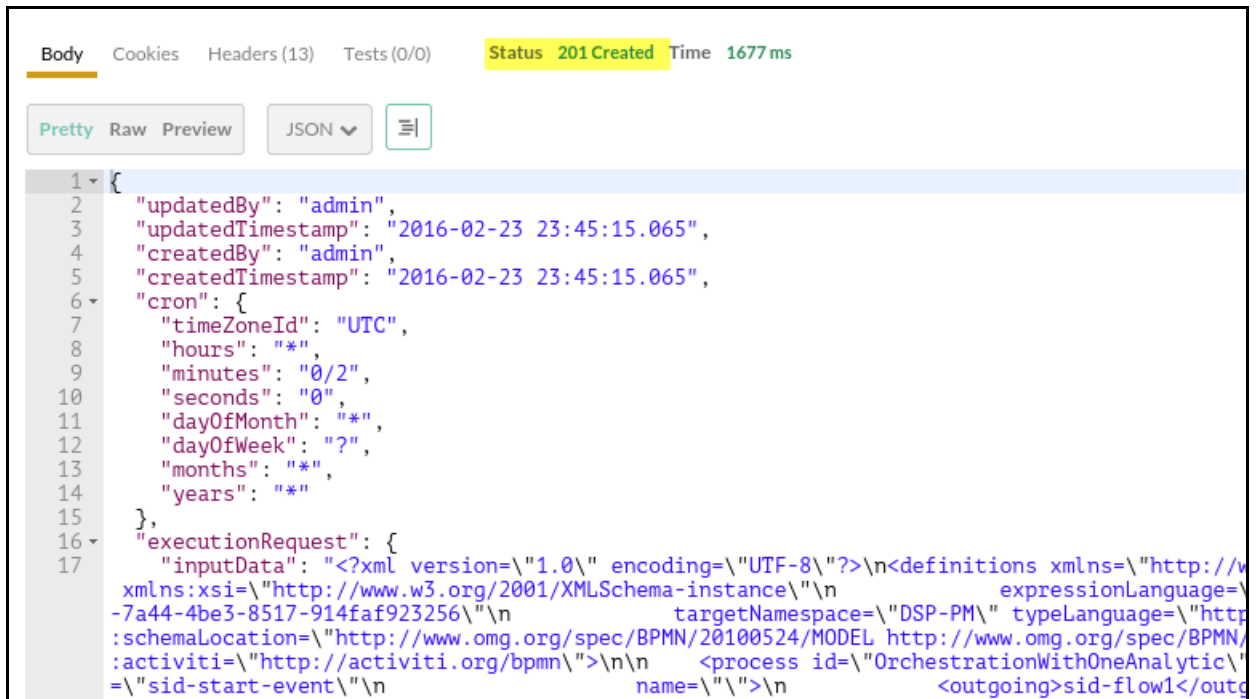


- Edit/Add the following fields and values as follows:

```
"seconds": "0",
"minutes": "0/2",
"url": "<Runtime Instance execution_uri>",
"httpMethod": "POST",
"name": "Accept",
"value": "application/json"
"inputData": copy from Escape for JSON
```

**Note:** Be sure to include the open/closing quotation marks for the **inputData** field value.

4. Click **Send**, and you should receive the following response:



```
1 {
2   "updatedBy": "admin",
3   "updatedTimestamp": "2016-02-23 23:45:15.065",
4   "createdBy": "admin",
5   "createdTimestamp": "2016-02-23 23:45:15.065",
6   "cron": {
7     "timeZoneId": "UTC",
8     "hours": "*",
9     "minutes": "0/2",
10    "seconds": "0",
11    "dayOfMonth": "*",
12    "dayOfWeek": "?",
13    "months": "**",
14    "years": "*"
15  },
16  "executionRequest": {
17    "inputData": "<?xml version='1.0' encoding='UTF-8'?>\n<definitions xmlns='http://www.omg.org/spec/BPMN/20100524/MODEL http://www.omg.org/spec/BPMN/20100524/MODEL http://www.w3.org/2001/XMLSchema-instance'\n      xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'\n      expressionLanguage='http://www.w3.org/2001/XMLSchema-instance'\n      targetNamespace='DSP-PM'\n      typeLanguage='http://www.w3.org/2001/XMLSchema-instance'\n      schemaLocation='http://www.omg.org/spec/BPMN/20100524/MODEL http://www.omg.org/spec/BPMN/20100524/MODEL'\n      <process id='OrchestrationWithOneAnalytic'\n        <start event id='sid-start-event'\n          <outgoing>sid-flow1</outgoing>
```

- Note the status returned of "201 Created" and the resulting job schedule.
- Update your Postman environment variables with the job Id.

5. Use the GET Request to Retrieve All Jobs, view your running job.

- Add a new Postman tab (window) and select **GET Retrieve All Jobs** from the Scheduler Service collection.

You should see a similar response as the following:

```

Body Cookies Headers (13) Tests (0/0) Status 200 OK Time 320 ms
Pretty Raw Preview JSON
1 {
2   "jobs": [
3     {
4       "updatedBy": "admin",
5       "updatedTimestamp": "2016-02-23 23:45:15.065",
6       "createdBy": "admin",
7       "createdTimestamp": "2016-02-23 23:45:15.065",
8       "cron": {
9         "timeZoneId": "UTC",
10        "hours": "*",
11        "minutes": "0/2",
12        "seconds": "0",
13        "dayOfMonth": "*",
14        "dayOfWeek": "?",
15        "months": "*",
16        "years": "*"
17      },
18      "executionRequest": {
19        "inputData": "<?xml version='1.0' encoding='UTF-8'>\n<definitions xmlns='http:
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
expressionLangu
-81430087-7a44-4be3-8517-914faf923256'\n
targetNamespace='DSP-PM' typeLangu
xsi:schemaLocation='http://www.omg.org/spec/BPMN/20100524/MODEL http://www.omg.org/spe
:activiti='http://activiti.org/bpmn'>\n\n  <process id='OrchestrationWithOneAnalytic\
='sid-start-event'\n
name=''>\n
<outgoing>sid-flow1</out
k completionQuantity='1'\n
id='sid-1001'\n
is
  
```

- Scrolling down, you'll see the Job ID and status:

```

20   =\"\" sourceRef=\"sid-10001\" targetRef=\"sid-ēnd-event\"/>\n\n   </process>\n\n</definitio
21   \"httpMethod\": \"POST\",
22   \"httpHeaders\": [
23     {
24       \"name\": \"HTTP_HEADER_Content-Type\",
25       \"value\": \"application/json\"
26     },
27     {
28       \"name\": \"HTTP_HEADER_Accept\",
29       \"value\": \"application/json\"
30     },
31     {
32       \"name\": \"HTTP_HEADER_Predix-Zone-Id\",
33       \"value\": \"f76daffd-5cb0-475b-83ab-78dabec4f526\"
34     }
35   ],
36   \"url\": \"https://predix-analytics-execution-release.run.aws-usw02-pr.ice.predix.io\"
37 },
38 \"description\": \"test description\",
39 \"name\": \"test1\",
40 \"id\": \"18673c37-38d5-4b98-a2a9-5ba3f5bd66b8\",
41 \"state\": \"ACTIVE\"
42 }
43 ],
44 \"currentPageSize\": 0,
45 \"maximumPageSize\": 0,
46 \"currentPageNumber\": 0,
47 \"totalElements\": 0,
48 \"totalPages\": 0

```

- Enter the Job ID in the Postman environment variable and save it.

6. View the Job schedule results by Get Job History by Job ID

- Add a new Postman tab (window) and select **GET Job History By Job ID** from the Scheduler Service collection.
- Click **Send** and again after a few minutes, you should receive a response as follows:

```

33   "id": "9364ad7a-7c83-4361-920f-cc17688e7400"
34   },
35   {
36     "httpMethod": "POST",
37     "cron": "0 0/2 * * * ? * ",
38     "timeZoneId": "UTC",
39     "jobId": "5dd3e7e2-49d2-4d55-a2a4-b2de0d914e88",
40     "scheduledFireTime": "2015-12-28 22:44:00.0",
41     "fireTime": "2015-12-28 22:44:00.01",
42     "statusMessage": "completed",
43     "url": "https://predix-analytics-execution-release.run.aws-usw02-pr.ice.predix.io/api/v1
/execution",
44     "id": "3ac350cc-ce54-431b-b1eb-ca3dfbad672e"
45   },
46   {
47     "httpMethod": "POST",
48     "cron": "0 0/2 * * * ? * ",
49     "timeZoneId": "UTC",
50     "jobId": "5dd3e7e2-49d2-4d55-a2a4-b2de0d914e88",
51     "scheduledFireTime": "2015-12-28 22:42:00.0",
52     "fireTime": "2015-12-28 22:42:00.01",
53     "statusMessage": "completed",
54     "url": "https://predix-analytics-execution-release.run.aws-usw02-pr.ice.predix.io/api/v1
/execution",
55     "id": "024ab33d-f224-4c35-891e-2fc3c02bbeae"
56   }

```

You can see the job being run every 2 minutes. Note the time stamp and the 2 minute intervals shown in the history.

### Exercise Summary

In this exercise you learned how to:

- Create and validate an orchestration of an analytic
- Create, test and validate a job schedule for an analytic

## Lab 8: Connecting Machines to Predix Cloud

### Learning Objectives

- Become familiar with the Predix SDK and Predix OSGI Containers
- Configure a standard Predix Machine container to collect data from the Modbus adapter
- Configure a connection between Machine and Predix Cloud
- Create a basic bundle and deploy into the Machine services container
- Add security to a consumer bundle
- Generate gateway data using a machine adapter (Machine Gateway API)

### Lab Exercises

- [Using Predix SDK to Generate a Machine on page 191](#)
- [Generate and Capture Simulated Data on page 204](#)
- [Connect to the Cloud with the HTTP Data River and HTTP Service on page 216](#)
- [Working with Consumer Bundles on page 230](#)
- [Adding Security to Bundles on page 237](#)
- [Generate Gateway Data with the Machine Adapter on page 245](#)

### Directions

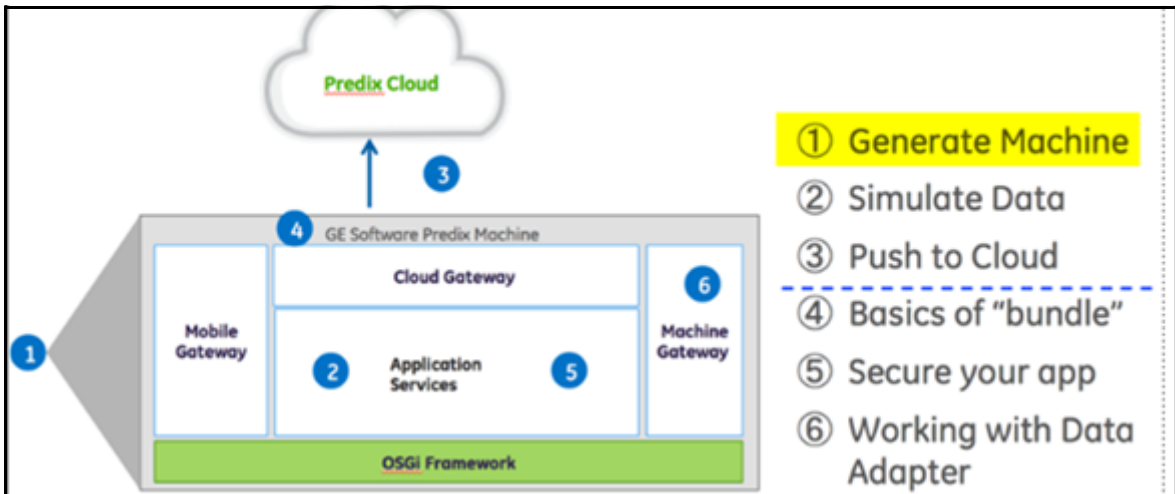
Complete the exercises that follow.

## Exercise 1: Using Predix SDK to Generate a Machine

### Learning Objectives

The goal of this lab is to familiarize you with the Predix SDK and Predix OSGI Containers. At the end of the lab, you will have done the following

- Explore the Predix SDK
- Create and navigate a machine OSGI container
- Configure the web console to administer bundles in the container



### Directions

Complete the exercises that follow.

## Part I - Examine the Predix SDK

### Steps

1. Locate the Predix SDK .

- On the DevBox Desktop click the **Places** menu and select **PredixApps** from the list
- Open the *training\_labs/ machine/predixsdk-15.3.0* folder and examine the files

<ul style="list-style-type: none"> <li>▼ predixsdk-15.3.0           <ul style="list-style-type: none"> <li>▼ docs               <ul style="list-style-type: none"> <li>apidocs.zip</li> <li>Container_docs.zip</li> </ul> </li> <li>▶ eclipse-plugins               <ul style="list-style-type: none"> <li>InstallationGuide.pdf</li> </ul> </li> <li>▶ license</li> <li>▼ samples               <ul style="list-style-type: none"> <li>sample-apps.zip</li> <li>sample-cloud-apps.zip</li> </ul> </li> <li>▼ utilities               <ul style="list-style-type: none"> <li>▼ containers                   <ul style="list-style-type: none"> <li>GenerateContainers</li> <li>GenerateContainers.bat</li> <li>pom.xml</li> <li>README.txt</li> <li>README.txt</li> </ul> </li> </ul> </li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>/docs           <ul style="list-style-type: none"> <li>• Javadoc APIs for Predix machine services</li> <li>• <a href="#">Prosyst docs</a> for <a href="#">OSGi</a> containers</li> </ul> </li> <li>/eclipse-plugins           <ul style="list-style-type: none"> <li>• Import into your Eclipse-based tool (STS)</li> </ul> </li> <li>/samples           <ul style="list-style-type: none"> <li>• Predix machine samples</li> <li>• Predix cloud sample for HTTP data service</li> </ul> </li> <li>/utilities/containers           <ul style="list-style-type: none"> <li>• Machine containers and scripts</li> </ul> </li> </ul>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

2. Install the Machine documentation and samples.

- Open a Terminal (**Terminal** icon on Desktop) and go to the *docs* directory
 

```
cd PredixApps/training_labs/machine/predixsdk-15.3.0/docs
```
- Unzip the documentation file
 

```
unzip apidocs.zip -d apidocs
```
- Go to the *samples* directory and unzip the two sample folders
 

```
cd ../samples
unzip sample-apps.zip -d sample-apps
unzip sample-cloud-apps.zip -d sample-cloud-apps
```

3. Create a debug Machine container.

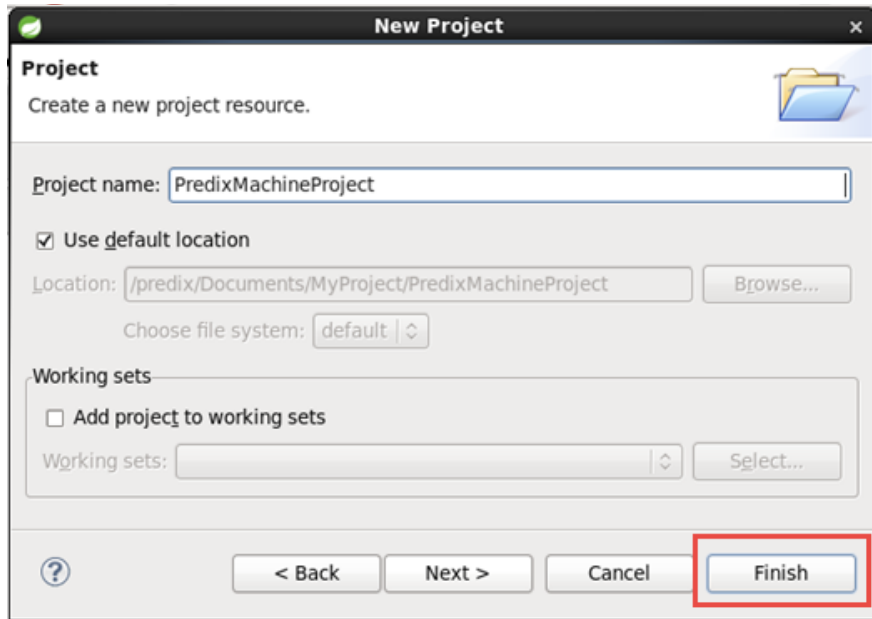
- Open Eclipse (**Eclipse-STS** icon on Desktop)
- Ensure you are in the Predix SDK perspective (select the *GE Predix SDK tab* in the upper right corner of the tool)



4. Create a new project that will hold the machine container image.

- Click **File > New > Other... > General > Project**
  - ◆ Click **Next**
- Project name: `PredixMachineProject`
- 
- 
- 
- 
- 
-





- Accept all other default settings and click **Finish**

5. Create a container image to use for the remainder of this training.

- In the Navigator, select the PredixMachineProject and click on **File > New > Image Description**

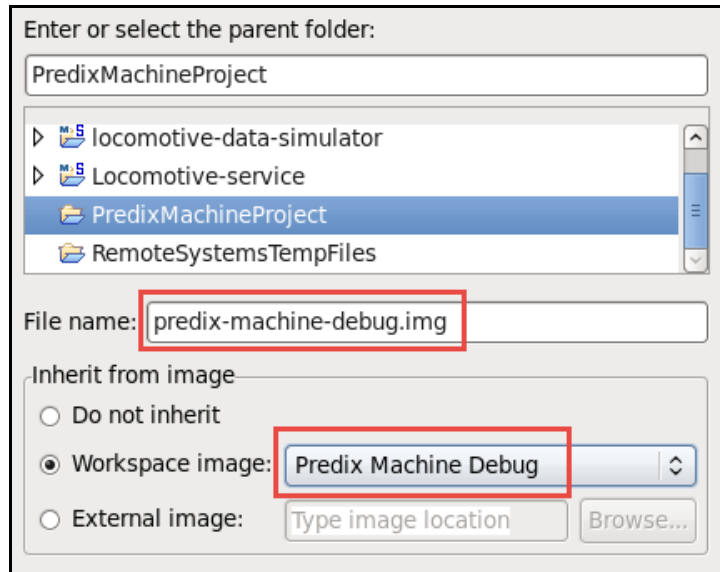
- File name: **predix-machine-debug**

- Configure the Inherit Image section

- ◆ Workspace image: Predix Machine Debug (select from list)

**Note:** : If you do not select Predix Machine Debug you will not be able to open the Predix Administrator Console

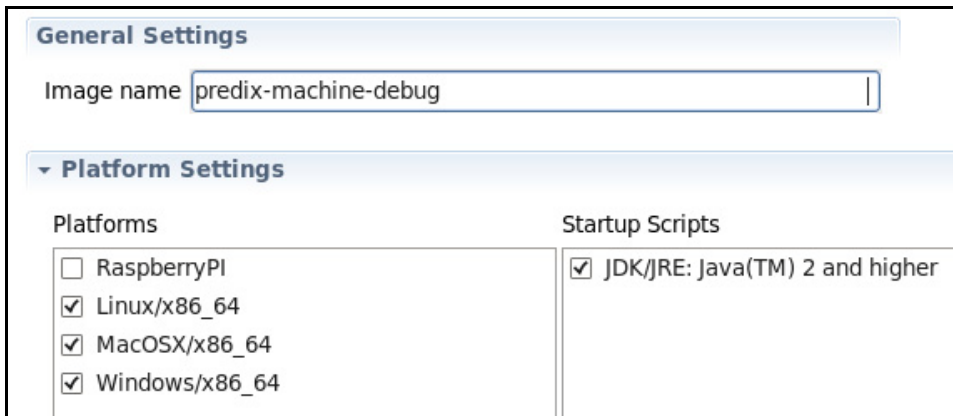
(See next page)



- Click **Finish**; a new tab *predix-machine-debug* will open in the editor window

6. Export the Debug Machine Container.

- **Platform Settings:** verify 3 target platforms and JDK/JRE startup script are selected



## 7. Export the container image.

- Click the **Export** button in the top right corner (scroll the screen to the right to see the toolbar)



- ◆ The OSGi image builder export wizard is opened
- Configure the export
  - ◆ Check **Create image in directory** target for your container
  - ◆ Destination: browse to  
PredixApps/training\_labs/machine/predixsdk-15.3.0/utilities/containers
    - Click **OK**

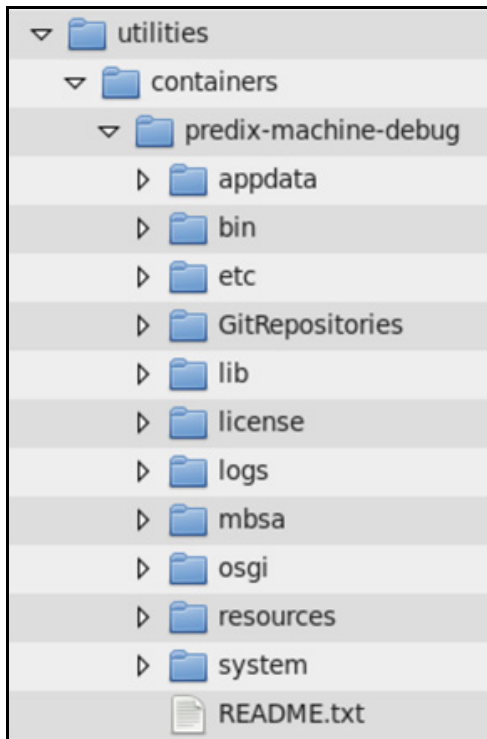
 A screenshot of the "OSGi image builder" dialog box. The title bar says "OSGi image builder" and the subtitle is "Generate OSGi image." There is a small icon of a cube with a globe in the top right corner. The dialog is divided into two main sections: "Source" and "Destination".
   
 In the "Source" section, there are two radio buttons: "Workspace image:" (selected) and "External image:". The "Workspace image:" dropdown menu shows "predix-machine-debug". The "External image:" field contains "Type image location" and has a "Browse..." button.
   
 In the "Destination" section, there is a text field containing the path "/predix/PredixApps/training\_labs/machine/predixsdk-15.3.0/utilities/containers" and a "Browse..." button. Below this, there are four radio buttons: "Save in zip format", "Save in tar format", "Create image in directory" (selected), and "Save in Android apk format".
   
 Red boxes highlight the "Workspace image:" dropdown, the "Destination" text field, and the "Create image in directory" radio button.

- ◆ Click **Finish** and click **Yes** at the prompt

A new folder *predix-machine-debug* is created with all the container files in the *predixsdk-15.3.0/utilities/containers* directory.

8. Examine the exported container image folder contents.

- On the Desktop click the **File Finder** icon and locate the predix-machine-debug container (hint: where you just exported it to)
- It has the following structure:



9. Start the debug container.

- In the Terminal, go to the /bin directory of predix-machine-debug  
`cd predixsdk-15.3.0/utilities/containers/predix-machine-debug/bin`
- Run the start script using the command  
`./predixmachine clean`

The first time you start machine it will take a few minutes. Look for the *Server is started* message in the terminal window.

```
2016-03-11 12:22:08,255[FW Buff Logger]|INFO|com.prosyst.mbs.system.bundle|2
com.prosyst.mbs.system.bundle-1.0.0|Bundle with id #116
com.ge.dspmicro.webconsole-predixcloudenroll, 15.3.0 was started.
#0 INFO > [PlatformState] State Change from STARTING to ACTIVE
2016-03-11 12:22:08,264[FW Buff Logger]|INFO|com.prosyst.mbs.system.bundle|2
com.prosyst.mbs.system.bundle-1.0.0|[PlatformState] State Change from STARTI
to ACTIVE
Server is started.
fw>
```

## Part II - Test Administrator Command Line Options

In this exercise, you discover console commands to interact with the Machine services.

### Steps

1. Discover available commands in the command line interface (CLI).

- In the Terminal running the machine container
  - ◆ If you don't see a `fw>$` prompt, press return until you see `fw>`
  - ◆ Type `ls` (or `list`) at the command line to see the machine bundles in your container, along with their state



- ◆ Notice the bundle id beside each one

```

predix@localhost:~/PredixApps/training_labs/machine/predixsdk-15.3.0/utilities/cont
File Edit View Search Terminal Help
fw>$
fw>$ls
ID      State      Jar Name
-----
0       ACTIVE    System Bundle
1       ACTIVE    javax.servlet-api-3.1.0
2       ACTIVE    org.osgi.compendium-5.0.0
3       ACTIVE    com.prosyst.mbs.core.api-8.0.6
4       ACTIVE    com.prosyst.mbs.core.threads-8.0.5
5       ACTIVE    com.prosyst.mbs.core.db-8.0.5
6       ACTIVE    com.prosyst.mbs.osgi.cm.bundle-8.0.5
7       ACTIVE    com.prosyst.mbs.osgi.api-8.0.5
8       ACTIVE    com.prosyst.mbs.osgi.metatype.bundle-8.0.7
9       ACTIVE    com.prosyst.mbs.util.api-8.0.5
10      RESOLVED  com.prosyst.mbs.osgi.compendium.extension-8.0
11      ACTIVE    jersey-all-2.10.1
    
```

- Now try running these commands to display information about machine services

**Command**

**Action**

<code>?</code>	Displays list of all commands available
<code>ls</code>	Lists the bundles
<code>bundle &lt;bundle Id&gt;</code>	Displays bundle details
<code>manifest &lt;bundle Id&gt;</code>	Displays the manifest of a bundle
<code>services &lt;bundle Id&gt;</code>	Displays services provided by the bundle

- Be aware of the following commands available through the terminal window but do not try them yet

Command	Action
<code>start &lt;bundle Id&gt;</code>	Start the bundle
<code>stop &lt;bundle Id&gt;</code>	Stop the bundle
<code>restart &lt;bundle Id&gt;</code>	Restart the bundle
<code>e or exit</code>	Shutdown machine

- Leave the container running to do the next exercise

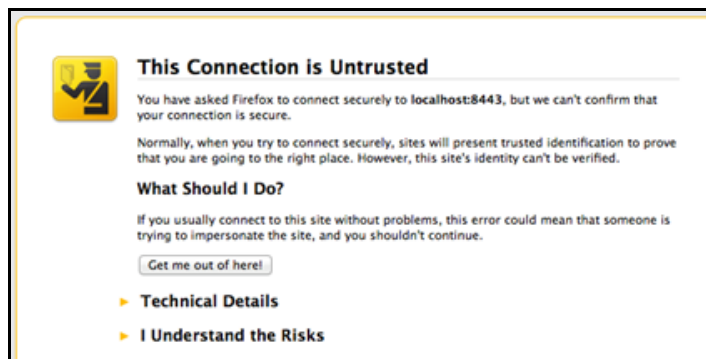
## Part III - Using the Web Console

The Debug container can be managed in one of two ways: in a terminal window through command line options, or through a web interface.

### Steps

1. Access the Web Console.

- In the Terminal, verify your machine container is running
- In a Firefox browser navigate to: <https://localhost:8443/system/console>
- The following dialog box is displayed

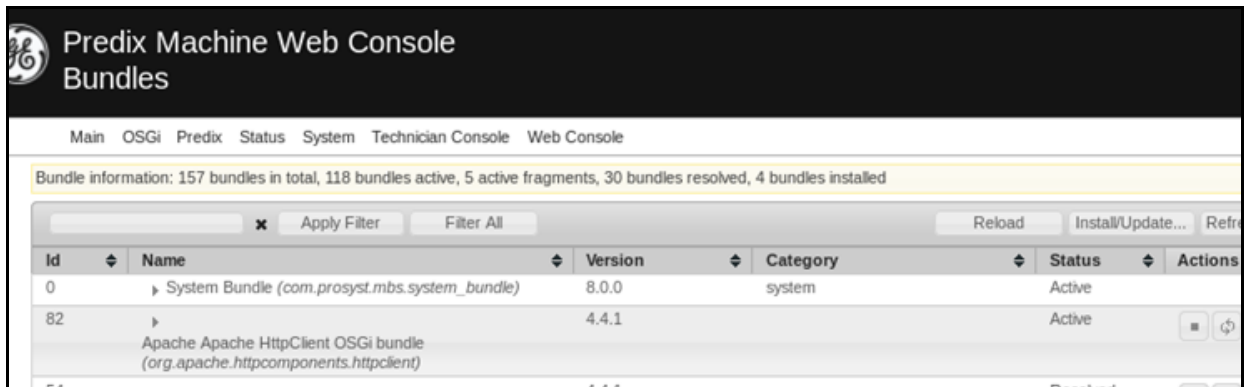


**Note:** Machine services use a self-signed certificate by default. You can safely add an exception to allow your browser to connect to Machine.

- ◆ Click **I Understand the Risks** and then select **Add Exception**
- ◆ Click **Confirm Security Exception** to complete the process

2. Log onto the Web Console.

- Login using the following credentials
  - ◆ Username: `predix`
  - ◆ Password: `predix2machine`
- The console is displayed

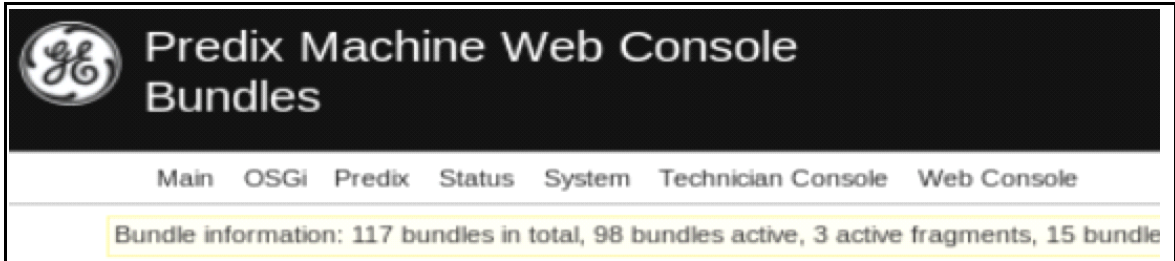


The Web Console administers all of the features offered by the Machine container. Once the container starts on the machine, all the functionalities are controlled through the UI. It is a personal preference whether to administer the console through the Admin Web Console or at the command line.

- Keep the container running for the next exercise



3. Try executing the following commands from the various menus in the web console.



<b>Men &gt; Command</b>	<b>Action</b>
System > Shell	Opens terminal window for command line option
OSGi > Bundles	Displays full list of installed bundles
Technician Console > Configuration	Displays bundle configuration files
Predix > Store and Forward	Displays data files waiting for connection to be restored

4. Shut down web console and machine server.

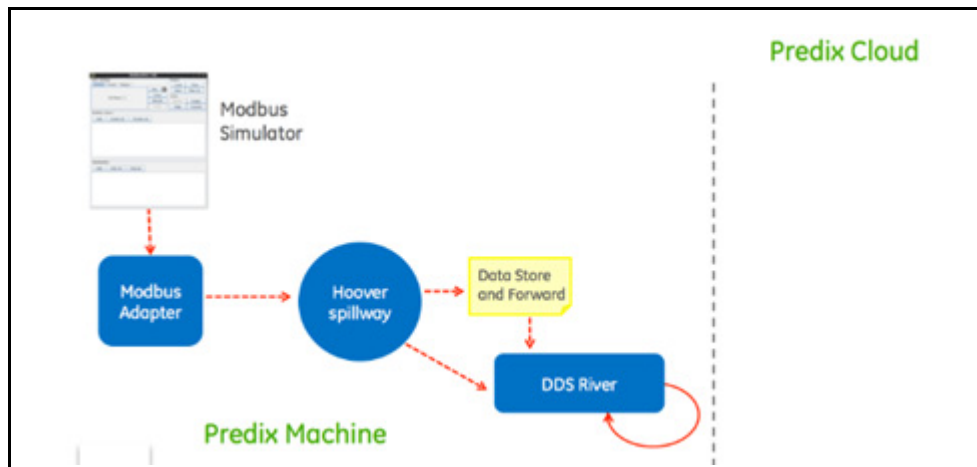
- Close the Administrator Web Console window (close the browser)
- In the Terminal at the `fw?` prompt, type `exit` to stop the machine container  
This will take a minute or two to bring everything down and return you to the terminal window prompt.

## Exercise 2: Generate and Capture Simulated Data

### Learning Objectives

By the end of the lab, students will be able to:

- Configure the Modbus adapter in the Machine container
- Configure a spillway for processing data
- Stage generated data through Data Store and Forward
- Configure a DDS Data River and move data locally



## Part I - Download and Start the Modbus PLC Simulator

In this exercise, you configure a standard Machine container to collect data from Modbus adapter.

**Note:** If your Predix Debug Machine is running, stop the container. You will restart after reconfiguring the bundles.

### Steps

1. Download the Java ModbusPal simulator.

- In a browser go to <http://sourceforge.net/projects/modbuspal/files/modbus>
  - ◆ Download the **ModbusPal.jar** file
- In File Finder, navigate to the *Downloads* folder
  - ◆ Copy the **ModbusPal.jar** file to the *PredixApps/training\_labs/machine* folder

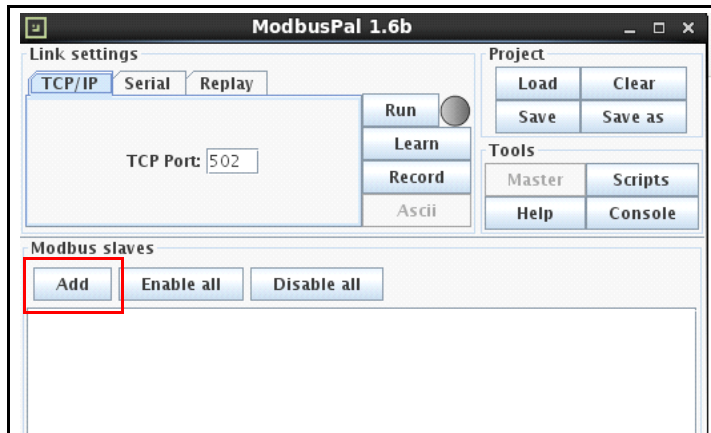
2. Start the Modbus simulator.

- From a Terminal navigate to `~/PredixApps/training_labs/machine`
- Run the command to start the simulator

```
sudo java -jar ModbusPal.jar
```
- A new dialog window opens

### 3. Configure the simulator.

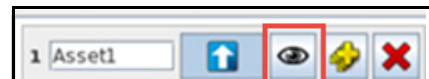
- In the Modbus slaves section, click **Add** to add a slave



- Select **1**, update the Slave name to **Asset1**, and click the **Add** button



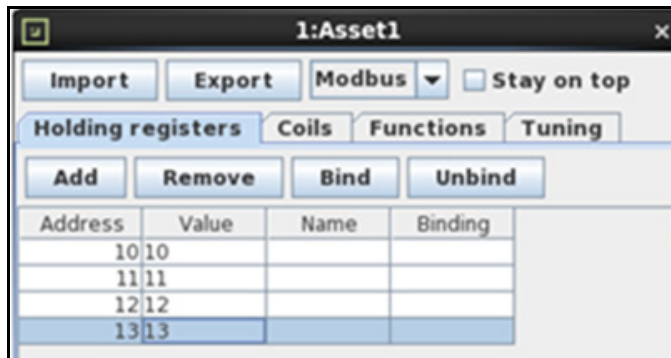
- Configure registers (sensors) for this asset
  - ◆ For Asset1, click on the edit button (the eye icon)
  - ◆ In the *Holding registers tab*, click the **Add** button



- Reset the number of registers
  - ◆ From: 10 (type this in)
  - ◆ To: 13 (type this in)
    - Click **Add**



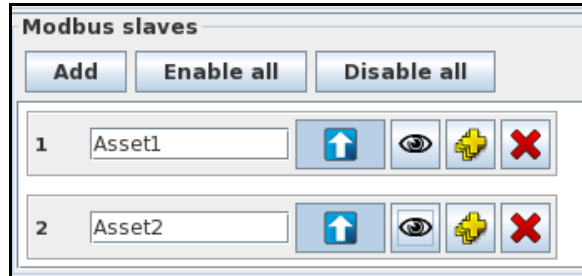
- Set the values for the registers to random numbers. The configuration should look similar to the picture below:



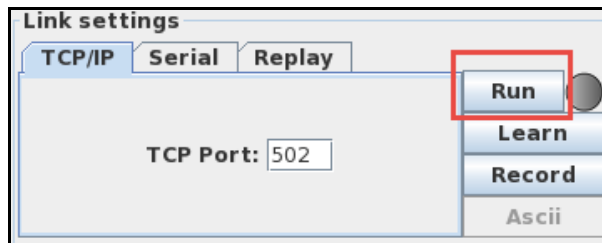
- Close the editor window by clicking on the **X** in the right corner

## 4. Add a second asset.

- Add a second asset and configure the register range from 20 to 23



- Click the **Run** button to start the simulator



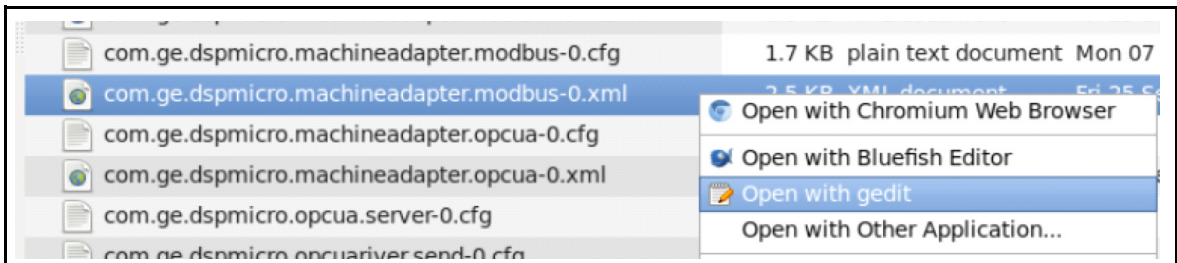
## Part II - Configure a Modbus Adapter in Machine Services

Predix Machine supports Modbus through the Modbus adapter bundle. This bundle must be configured to work with a given Modbus device.

### Steps

1. Edit configuration settings for Modbus adapter.

- Open File Finder and navigate to /predix/PredixApps/training\_labs/machine/predixsdk-15.3.0/utilities/containers/predix-machine-debug/etc
  - ◆ The configuration properties for the Modbus bundle can be found in etc/**com.ge.dspmicro.machineadapter.modbus-0.xml**.
  - ◆ Open this file in a text editor





- Remove the TCP/IP comments for <dataNodeConfigs> (remove highlighted lines)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<modbusMachineAdapterConfig>
  <name>Onsite monitor modbus nodes</name>
  <description>Onsite monitor modbus nodes</description>
  <dataNodeConfigs>
    <!-- REMOVE THIS LINE FOR TCP/IP -->
    <channel protocol="TCP_IP" tcpIpAddress="127.0.0.1" tcpIpPort="502">
      <unit id="1">
        <register name="Node-1-1" dataType="INTEGER" address="10" r
        <register name="Node-1-2" dataType="DECIMAL" address="11" r
      </unit>
      <unit id="2">
        <register name="Node-2-1" dataType="INTEGER" address="20" r
        <register name="Node-2-2" dataType="INTEGER" address="21" r
      </unit>
    </channel>
    REMOVE THIS LINE -->
    <!-- REMOVE THIS LINE FOR SERIAL -->
    <channel protocol="SERIAL" portName="COM1" baudRate="9600" parity="
```

- Remove the TCP/IP comments for <dataSubscriptionConfigs>

```
<dataSubscriptionConfigs>
  <!-- REMOVE THIS LINE FOR TCP/IP -->
  <dataSubscriptionConfig name="Temperature_Subscription"
startPointOffset="10">
    <nodeName>Node-2-1</nodeName>
    <nodeName>Node-1-1</nodeName>
  </dataSubscriptionConfig>
  REMOVE THIS LINE -->
  <!-- REMOVE THIS LINE FOR SERIAL -->
  <dataSubscriptionConfig name="Pressure_Subscription" upd
startPointOffset="600">
    <nodeName>Node-3-2</nodeName>
    <nodeName>Node-4-2</nodeName>
```

- Save the file

**Note:** This configures the Modbus adapter to listen for values on port 502 at IP address 127.0.0.1. The Modbus PLC simulator software runs at this location by default.

## Part III - Configure a Spillway for DDS

The Data River service allows data to be passed through spillways for processing. In this example, we will set up a simple subscription-based spillway to listen to the Modbus adapter.

### Steps

1. Set the spillway destination in the configuration file.

- In a text editor, open the file `etc/com.ge.dspmicro.hoover.spillway-0.cfg`
  - ◆ Change the `com.ge.dspmicro.hoover.spillway.destination` property to **Sender Service**

```
50 # [Required] Destination Data River name to where the data will be sent.
51 # Change to the Data River by replacing the value with: Sender Service
52 com.ge.dspmicro.hoover.spillway.destination=SenderService
```

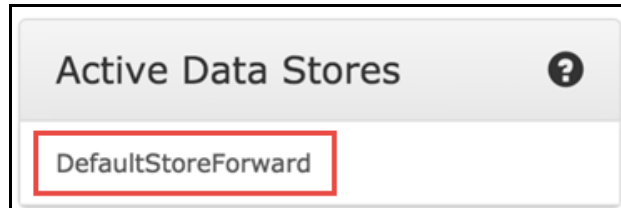
- Save the file

2. Collect "data forward" data.

- In the Terminal, open a new tab (*File > Open Tab*)
- Navigate to the `/bin` directory of the `predix-machine-debug` folder  
`cd predixsdk-15.3.0/utilities/containers/predix-machine-debug/bin`
- Start the machine container with the start script  
`./predixmachine clean`
  - ◆ Look for the message "Server is started"

### 3. View stored data in the Web Console.

- In a browser, open Admin Web Console (<https://localhost:8443/system/console>)
  - ◆ From the **Predix** menu, select **Store and Forward**
  - ◆ Click **DefaultStoreForward**



Within a minute you should see a numbered list of data objects being captured and stored every 60 seconds. Because there is no connection to the cloud yet, this data is saved. Only the last 10 data objects are shown. You will see errors in the terminal window until the data river is configured.

 A screenshot of the web console interface. On the left, there are two panels: 'Active Data Stores' with a help icon and a list containing 'DefaultStoreForward', and 'Inactive Data Stores' with a help icon and the text 'No inactive data stores'. On the right, the 'Store Contents' panel shows a table of data objects. The table has two columns: 'ID' and 'Timestamp'. Below the table, it says '10 oldest items in DefaultStoreForward - 7 items total'.
 

ID	Timestamp
1	3/9/2016, 10:26:10 AM
2	3/9/2016, 10:27:10 AM
3	3/9/2016, 10:28:10 AM
4	3/9/2016, 10:32:10 AM
5	3/9/2016, 10:33:10 AM

- Stop the Machine Container by typing **exit** in the Terminal (do not stop by using Ctrl+C)

#### 4. Configure the Data River Sender service.

**Note:** The Data River Send service is configured by default to transmit data from all spillways to configured Data River Receiver services.

- In a text editor, open the etc/**com.ge.dspnet.datariver.send-0.cfg** file
  - ◆ Change the setting for secure transfer to: **false**

```
60 # [Required] boolean flag for TLS/Secure (true) or TCP/non-secure
    for value change to take effect.
61 com.ge.dspnet.datariver.send.secure.transfer=false
```

- Save and close the file

#### 5. Configure the Data River Receiver service.

- In a text editor, open the etc/**com.ge.dspnet.datariver.receive-0.cfg** file
  - Note:** The Data River Receiver service is configured to listen for a Data River Sender service in the same container by default.
  - ◆ Change the setting for secure transfer to: **false**

```
60 # [Required] boolean flag for TLS/Secure (true) or TCP/non-secure
    for value change to take effect.
61 com.ge.dspnet.datariver.receive.secure.transfer=false
```

- Save and close the file

#### 6. Restart the Machine and forward stored data.

- From the /bin directory restart the machine container
  - ◆ **./predixmachine clean**
- Open the Admin Web console in a browser (<https://localhost:8443/system/console>)
  - ◆ Select **Predix >Store and Forward**

The data under DataStoreForward is gone (it was sent to the data river receiver you just configured)

- In the terminal window you will see a success message repeated every few seconds

```

fw>$
fw>$2016-02-04 16:38:10,337[Thread-40]|INFO|com.ge.dspnet.datariver.send.impl.Date
com.ge.dspnet.datariver-send-15.3.0|Status Code: -1,100. Request valid. Transfer I
b0c9-311d75cc09d6.
2016-02-04 16:38:10,340[pool-9-thread-1]|INFO|com.ge.dspnet.datariver.receive.imp
com.ge.dspnet.datariver-receive-15.3.0|Status Code: -2,107. Opening new transfer s
efeb4219-4d18-44d0-b0c9-311d75cc09d6
2016-02-04 16:38:10,342[pool-11-thread-1]|INFO|com.ge.dspnet.datariver.send.impl.
com.ge.dspnet.datariver-send-15.3.0|Status Code: -1,202. TRANSFER_RESPONSE receive
Transfer ID: efeb4219-4d18-44d0-b0c9-311d75cc09d6.
2016-02-04 16:38:10,353[pool-10-thread-1]|INFO|com.ge.dspnet.datariver.send.impl.
com.ge.dspnet.datariver-send-15.3.0|Total Bytes Sent = 409. Transfer ID: efeb4219-
2016-02-04 16:38:10,354[pool-8-thread-1]|INFO|
com.ge.dspnet.datariver.receive.filecontenthandler.FileContentHandler|82-com.ge.ds
filehandler-15.3.0|
Transfer "efeb4219-4d18-44d0-b0c9-311d75cc09d6" completed.
Total bytes received:409 B
Total time: 0 seconds
Average speed: 399.4 KiB/s
Finished at: 2016/02/04 16:38:10,354
2016-02-04 16:38:10,358[pool-11-thread-1]|INFO|com.ge.dspnet.datariver.send.impl.
com.ge.dspnet.datariver-send-15.3.0|Status Code: -1,200. Successful transfer with

```

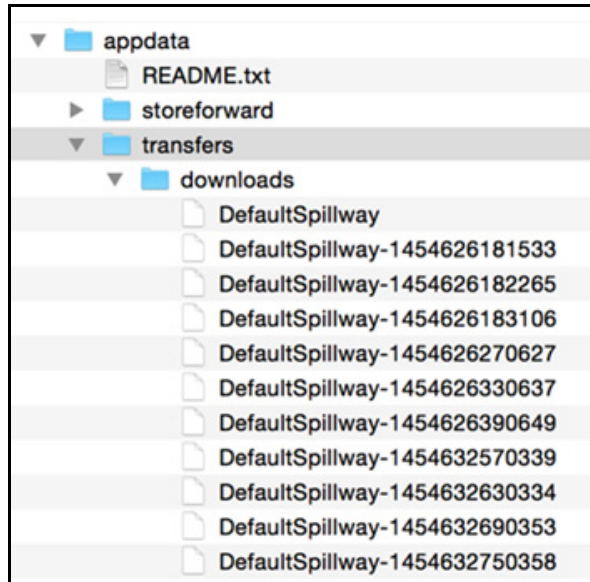
## 7. View stored data in the file system.

- In File Finder, navigate to **predixsdk-15.3.0/utilities/containers/predix-machine-debug/appdata/transfers/downloads**

**Note:** Because the DDS Data River endpoint defaults to localhost, the receiver will store data it receives in files in this directory.



- You should see a new file created once every 60 seconds containing values from the Modbus PLC simulator



- The contents of the file are JSON formatted and should be similar to the following:

```

DefaultSpillway-1454626181533
[{"address":"com.ge.dspmicro.machineadapter.modbus://127.0.0.1:502/2/20","datatype":"INTEGER","name":"Node-2-1","category":"REAL","value":0,"timestamp":1454624710138,"quality":"NOT_SUPPORTED (20000000) "}, {"address":"com.ge.dspmicro.machineadapter.mo127.0.0.1:502/1/10","datatype":"INTEGER","name":"Node-1-1","category":"REAL","value":0,"timestamp":1454624710139,"quality":"NOT_SUPPORTED (20000000) "}]
```

---

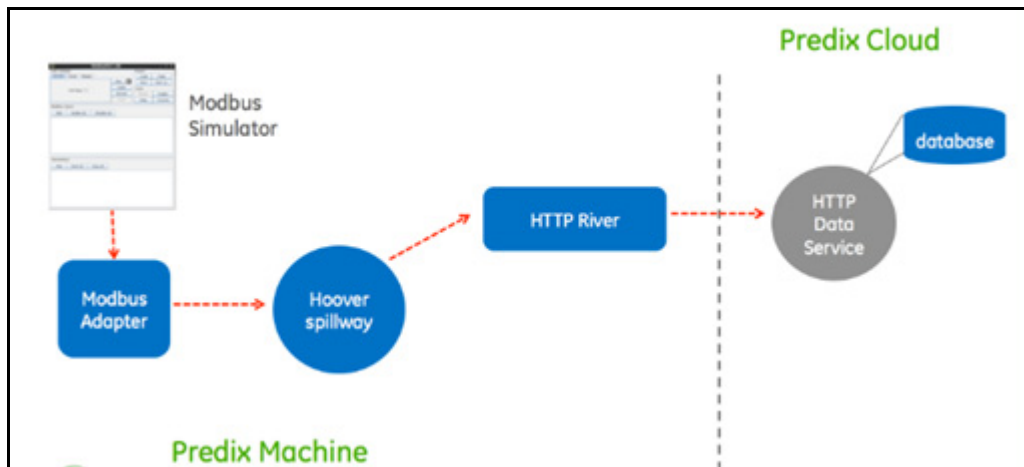
## Exercise 3: Connect to the Cloud with the HTTP Data River and HTTP Service

---

### Learning Objectives

By the end of the lab, students will be able to:

- Configure a connection between machine and Predix Cloud
- Build and deploy an HTTP Data Service in Predix Cloud
- Configure the HTTP Data River in Predix Machine



## Part I - Cloud Foundry Setup

In this exercise you verify UAA and Postgres service instances using the Cloud Foundry CLI. These service instances will be used by the HTTP Data Service you will configure in the next exercise.

**IMPORTANT:** Make sure you are not behind a corporate firewall and there are no proxies set

### Steps

1. Cloud Foundry setup.

- In the Terminal open a new tab and run the command: **env |grep http**
  - ◆ The values for `http(s)_proxy` and `HTTP(S)_PROXY` should not be set

2. Log into Cloud Foundry using your student credentials.

- In the Terminal run the command: **cf login**
- Enter your login information
  - Email> **student<XX>** (where XX is any number from 2-50, e.g. student14)
  - Password> **chang3m3**
  - Space> **2**



3. Determine the names of your postgres and UAA instances.

- In the Terminal run the `cf ps` command  
Note the name of the postgres and UAA service instances you created in earlier labs.

4. Stop the Predix machine container.

- Stop the Machine Container by typing `exit` in the Terminal



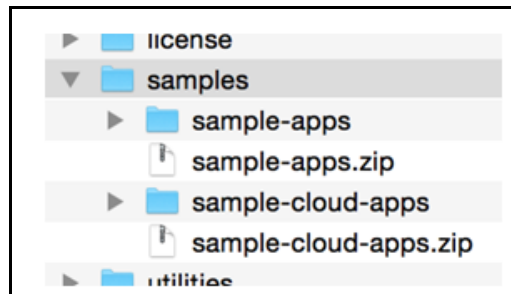
## Part II - Import the HTTP Data Service Sample from the Predix SDK

In this section you import and build the HTTP Data Service sample that runs in the cloud. This service is the endpoint for the HTTP River sending data from Predix Machine.

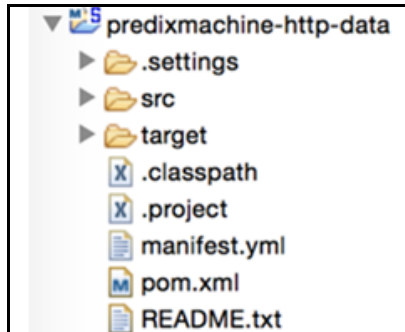
### Steps

1. Import a Maven project into Eclipse.

- In Eclipse, select **File > Import > Existing Maven Project**, click **Next**, and browse to the *samples* folder in the *predixmachinesdk*



- ◆ Click **Finish**



- The HTTP Data Service sample imports as **predixmachine-http-data** project

## 2. Update the manifest file.

- In the project open the **manifest.yml** file and update the following properties:

```
name: <your-name>-httpdata
services:
- <your-postgres-instance>
- <your-uaa-instance>
```

- ◆ Replace **<your-postgres-instance>** with the name of the postgres instance you created in an earlier lab.
- ◆ Replace **<your-uaa-instance>** with the name of the UAA instance you created in an earlier lab.

- For *example*:

```
name : student4-httpdata
services:
- student4-postgres
- student4-uaa
```

- Save and close the manifest file

### 3. Build the sample application.

- In Eclipse, right-click on the project name and select **Run As > Maven Install**

```
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ predixmachine-http-dat
[INFO] Installing /predix/Documents/PredixApps/training_labs/machine/predixsdk-15.3.0/se
[INFO] Installing /predix/Documents/PredixApps/training_labs/machine/predixsdk-15.3.0/se
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 3.950s
[INFO] Finished at: Tue Mar 22 11:51:27 PDT 2016
[INFO] Final Memory: 24M/215M
[INFO]
```

- In the Terminal, navigate to the *samples/sample-cloud-apps/httpdata* folder (where the manifest.yml file is)

```
cd /predix/PredixApps/training_labs/machine/predixsdk-15.3.0/
samples/sample-cloud-apps/sample/httpdata
```

- Push the app to the cloud and make sure it starts

```
cf push
cf apps
```

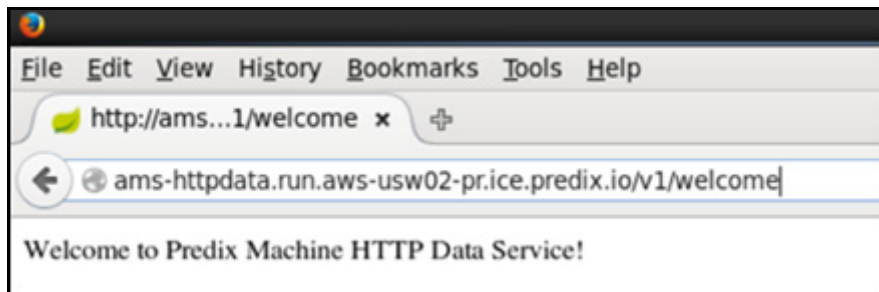
```
Showing health and status for app ams-httpdata in org Predix-Training / space Training1 as student
OK

requested state: started
instances: 1/1
usage: 1G x 1 instances
urls: ams-httpdata.run.aws-usw02-pr.ice.predix.io
last uploaded: Tue Mar 22 18:57:52 UTC 2016
stack: cflinuxfs2
buildpack: java-buildpack=v3.5.1-http://github.com/pivotal-cf/pcf-java-buildpack.git#d6c19f8 java-jdk-like-memory-calculator=2.0.1_RELEASE postgresql-jdbc=9.4.1208 spring-auto-reconfiguration=1.1

#0 state since cpu memory disk details
#0 running 2016-03-22 11:58:32 AM 0.0% 483.9M of 1G 143.7M of 1G
[predix@localhost httpdata]$
[predix@localhost httpdata]$ cf apps | grep ams
ams-httpdata started 1/1 1G 1G ams-httpdata.run.aws-
[predix@localhost httpdata]$
```

#### 4. Test your application.

- Open a browser window and enter the application route followed by **`/v1/welcome`**  
**`http://<app_route>/v1/welcome`**  
Where `<app_route>` is the URL of the application you just pushed to the cloud. The route can be located with the `cf apps` command.
- ◆ You should see "Welcome to Predix Machine HTTP Data Service" indicating your service has been deployed successfully



## Part III - Add a New Client ID to the UAA Instance

### Steps

1. Get the URL of the UAA instance.

- Get the VCAP environment variables from your application
  - ◆ In the Terminal run the command: `cf env <your-name>-httpdata`
  - ◆ Copy the contents of **UAA uri** (don't include the quotes)

```
,
predix-uaa-training": [
{
  "credentials": {
    "issuerId": "https://cedc4d21-793d-4297-8c40-4f65ed192109.predix-uaa-training.run.aws-usw02-pr.ice.p
    "uri": "https://cedc4d21-793d-4297-8c40-4f65ed192109.predix-uaa-training.run.aws-usw02-pr.ice.p
    "zone": {
      "http-header-name": "X-Identity-Zone-Id",
      "http-header-value": "cedc4d21-793d-4297-8c40-4f65ed192109"
    }
  },
  "label": "predix-uaa-training",
  "name": "annasch-uaa-instance",
  "plan": "Free",
  "tags": []
}
]
```

## 2. Get authenticated by the UAA instance and add a new client.

- Use the UAA command line interface (UAAC) to run the following commands:

```
uaac target <paste UAA uri here>
```

```
uaac token client get admin -s <admin_secret>
```

Where **<admin\_secret>** is the admin password you created in the security lab

```
[predix@localhost ~]$ uaac target https://3a699b63-eedf-40c9-9104-f9039c202c94.predix-uaa-training.run.aws-usw02-pr.ice.predix.io
Target: https://3a699b63-eedf-40c9-9104-f9039c202c94.predix-uaa-training.run.aws-usw02-pr.ice.predix.io
[predix@localhost ~]$
[predix@localhost ~]$ uaac token client get admin -s adminpassword
Successfully fetched token via client credentials grant.
Target: https://3a699b63-eedf-40c9-9104-f9039c202c94.predix-uaa-training.run.aws-usw02-pr.ice.predix.io
Context: admin, from client admin
```

- Create a new UAA client with the `uaac client add` command
  - Run the command: `uaac client add -i` and press **Enter**
  - You are prompted to fill in the following parameters (enter the **bolded** values)

```
client name: sample-clientid
```

```
new client secret: sample-clientid
```

```
verify client secret: sample-clientid
```

```
scope [list]: openid
```

```
authorized grant types [list]: authorization_code password
```

```
client_credentials refresh_token
```

```
authorities [list]: uaa.resource
```

```
access token validity (seconds): press Enter
```

```
refresh token validity (seconds): press Enter
```

```
redirect uri (list): press Enter
```

```
autoapprove (list): openid
```

```
signup redirect url (url): press Enter
```



```
[predix@localhost ~]$
[predix@localhost ~]$ uaac client add -i
Client name: sample-clientid
New client secret: *****
Verify new client secret: *****
scope (list): openid
authorized grant types (list): authorization_code password client_credentials refres
h_token
authorities (list): uaa.resources
access token validity (seconds):
refresh token validity (seconds):
redirect uri (list):
autoapprove (list): openid
signup redirect url (url):
  scope: openid
  client_id: sample-clientid
  resource_ids: none
  authorized_grant_types: authorization_code client_credentials password
  refresh_token
autoapprove: openid
action: none
authorities: uaa.resources
signup_redirect_url:
  lastmodified: 1458702545049
  id: sample-clientid
[predix@localhost ~]$
```

### 3. Configure Cloud Foundry credentials.

- Get the VCAP environment variables from your application
  - ◆ In the Terminal, run the command: `cf env <your-name>-httpdata`
  - ◆ Copy the **UAA issuerId** value (do not include quotes)

```
},
"predix-uaa-training": [
  {
    "credentials": {
      "issuerId": "https://cedc4d21-793d-4297-8c40-4f65ed192109.predix-uaa-training.run.aws-usw02-pr.ice.predix.io/oauth/to
      "uri": "https://cedc4d21-793d-4297-8c40-4f65ed192109.predix-uaa-training.run.aws-usw02-pr.ice.predix.io",
      "zone": {
        "http-header-name": "X-Identity-Zone-Id",
        "http-header-value": "cedc4d21-793d-4297-8c40-4f65ed192109"
      }
    },
    "label": "predix-uaa-training",
    "name": "annasch-uaa-instance",
    "plan": "Free",
    "tags": []
  }
]
}
```



- Edit the `/etc/com.ge.dspmicro.predixcloud.identity.cfg` file
  - ◆ Make the following changes to the file
 

```
com.ge.dspmicro.predixcloud.identity.uaa.token.url=<UAA issuerid> (paste from cf env output)
com.ge.dspmicro.predixcloud.identity.uaa.clientid=sample-clientid
com.ge.dspmicro.predixcloud.identity.uaa.clientsecret=sample-clientid
```
- Save and close the file

## Part IV - Configure Spillway for HTTP

Configure the HTTP Machine Services and start the flow of data.

### Steps

1. Configure the spillway for HTTP.

- Edit the `/etc/com.ge.dspmicro.hoover.spillway-o.cfg` file
  - ◆ Make the following changes to the file
 

```
com.ge.dspmicro.hoover.spillway.destination=Http Sender Service
```
- Save and close the file

2. Configure the HTTP River.

- Get the **application route** by running the command `cf apps` in the Terminal

```
798:~ 502439379$ cf apps | grep http
data          started      1/1    1G    1G    annams-httpdata.run.aws-usw02-pr.ice.predix.io
```

- Edit the `/etc/com.ge.dspmicro.httpriver.send-0.cfg` file

- ◆ Add the application route

`com.ge.dspmicro.httpriver.send.destination.host=<application route>`

Example:

```

# [Required] A friendly and unique name of the HTTP River.
com.ge.dspmicro.httpriver.send.river.name=Http Sender Service

# [Required] Route to the river receive application. (e.g. myapp.mycloud.com)
com.ge.dspmicro.httpriver.send.destination.host=annams-httpdata.run.aws-usw02-pr.ice.
    
```

3. Restart the Predix Machine.

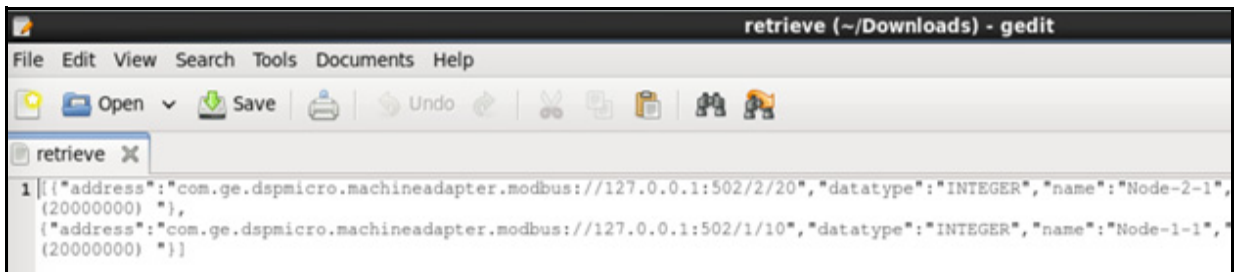
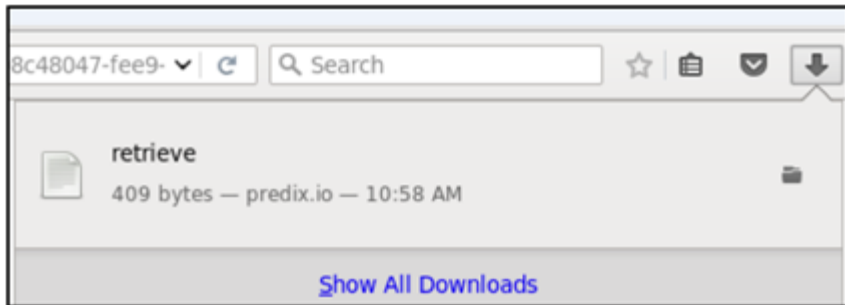
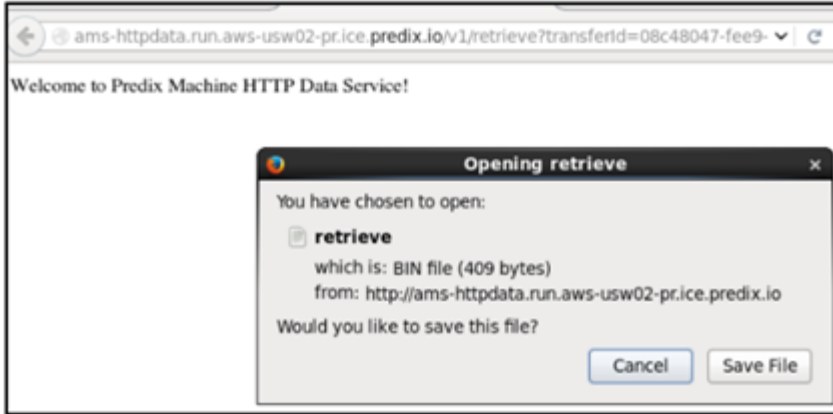
- In the Terminal navigate to the `/bin` directory of `predix-machine-debug`
  - ◆ Run the command `./predixmachine clean`
  - You will see a success message repeated for each transaction every few seconds

```

3-23 10:55:13,090[Thread-24]||INFO|com.ge.dspmicro.httpriver.send.impl.HttpRiverSendImpl|76-com.g
river-send-15.3.0|[200] transferId: 08c48047-fee9-4417-a8e1-ec5da3f6a0c1
: 27c7f71b-9eb5-4918-b186-01d0992a6994
me: Http Sender Service
Type: application/octet-stream
Disposition: null
Description: null
mp: 1458730510939
d: sample-clientid
3-23 10:56:11,059[Thread-24]||INFO|com.ge.dspmicro.httpriver.send.impl.HttpRiverSendImpl|76-com.g
river-send-15.3.0|Transfer completed successfully.
3-23 10:56:11,059[Thread-24]||INFO|com.ge.dspmicro.httpriver.send.impl.HttpRiverSendImpl|76-com.g
river-send-15.3.0|[200] transferId: 72acbb91-289a-4a04-baba-2af08ed02774
: 27c7f71b-9eb5-4918-b186-01d0992a6994
me: Http Sender Service
Type: application/octet-stream
Disposition: null
Description: null
    
```

- To verify that data is stored in the cloud, open a browser and retrieve a specific piece of data based on the transferId

```
https://<app>route>/v1/retrieve?transferId=<copy a transferId from terminal>
```



4. Stop ModbusPal and shut down Machine.

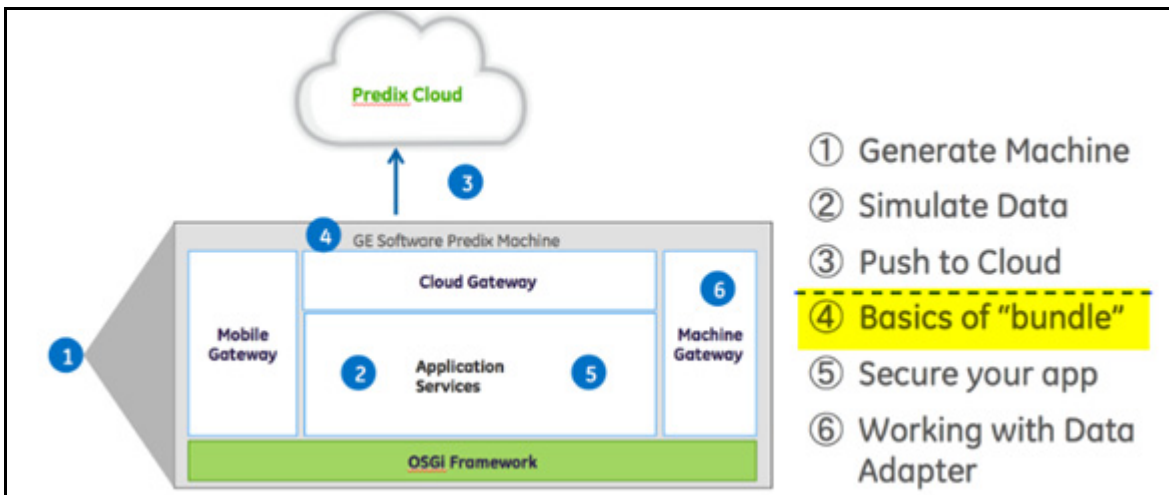
- Stop ModbusPal by selecting the “x’ in the upper right corner of the ModbusPal tool or typing **ctrl-c** in the terminal window where ModbusPal was started from.
- Stop the Machine Container by typing **exit** in the Terminal

## Exercise 4: Working with Consumer Bundles

### Learning Objectives

By the end of the lab, students will be able to:

- Create a basic bundle that can be deployed into the Machine services container
- Use a maven archetype to quickly create a project with predefined components



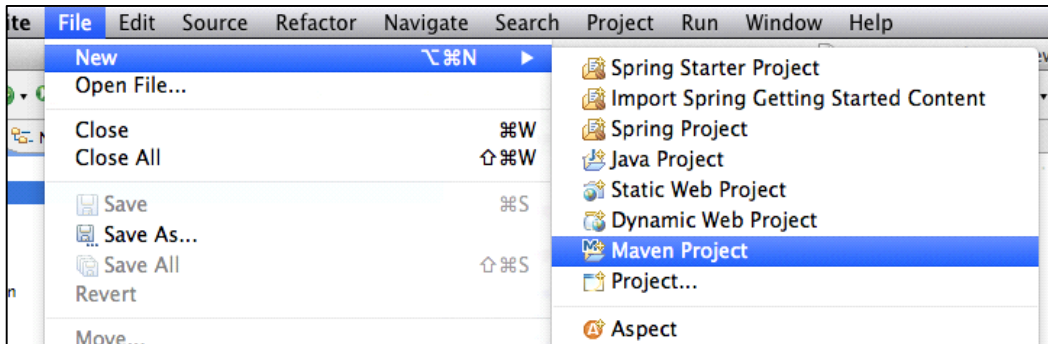
### Part I - Create a Basic Consumer Bundle

In this exercise, you create a basic bundle that can be deployed into the Machine services container. To simplify the process, you will use a maven archetype (a project template plug-in) to quickly create projects based on predefined project models.

## Steps

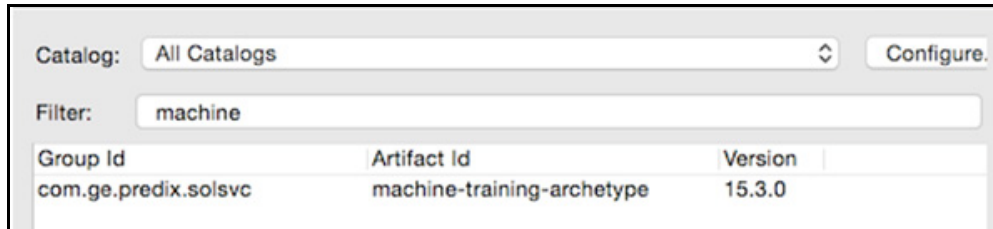
1. Create a new Java project from a maven archetype.

- In Eclipse, verify you are in the Predix SDK perspective
- From the **File** menu, select *New > Maven Project*
  - ◆ Accept the default values and click **Next**

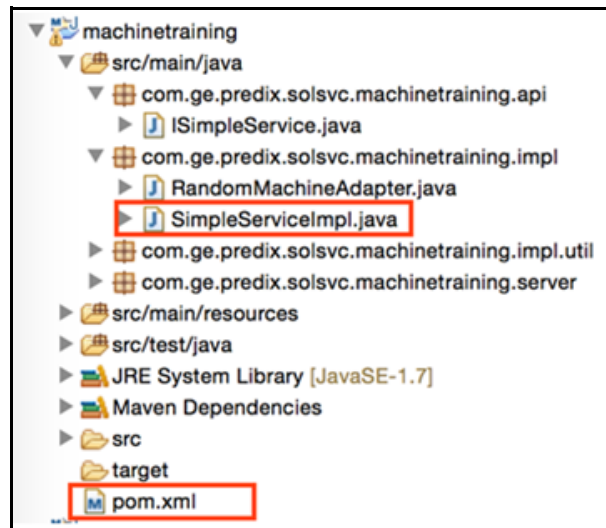


2. Select an archetype and define parameters.

- In the Filter field, type **machine** to locate the machine-training-archetype
  - ◆ Select the **archetype machine-training-archetype**, version 15.3.0 .



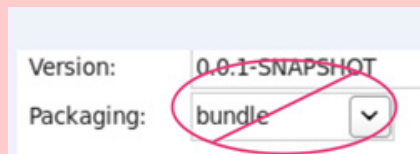
- ◆ Click **Next**
- Enter the following parameters on the next screen
  - ◆ Group Id: **com.ge.predix.solsvc**
  - ◆ Artifact Id: **machinetraining**
  - ◆ Package: **com.ge.predix.solsvc.machinetraining**
    - Click **Finish**
- A new project called *machinetraining* is displayed in the Navigator window



### 3. Examine the project files.

- Expand the *machinetraing* project and open the **SimpleServiceImpl.java** file
  - ◆ Locate the annotations for @Component, @Activate, and @Deactivate
- Open the **pom.xml** file and select the *pom.xml* tab in the editor window.

**Note:** Do not select the drop-down for Packages in the Overview tab as this changes the packaging type.



- ◆ Locate the <packaging> tag
  - This defines the build as an OSGI bundle
- ◆ Locate the <instructions> tag at the bottom of the file
  - These properties become part of the bundle manifest.mf file

```

<configuration>
  <obrRepository>NONE</obrRepository>
  <instructions>
    <Bundle-SymbolicName>${project.artifactId}</Bundle-SymbolicName>
    <Bundle-Name>${project.artifactId} built from arch-basic-
    <Bundle-Version>${project.version}</Bundle-Version>
    <Bundle-Classpath>.</Bundle-Classpath>
    <Export-Package>com.ge.predix.solsvc.machine.training.api
    <Import-Package>aQute.bnd.annotation.metatype;version=${i
      javax.ws.rs.*;version=${import.javax.ws.rs},
      org.slf4j;version=${import.org.slf4j};provider=paxlog
      org.osgi.service.component;version=${import.org.osgi.
      org.osgi.framework;version=${import.org.osgi.framework
      org.osgi.service.cm;version=${import.org.osgi.service
      com.ge.predix.solsvc.machine.training.api;version=${i
      com.ge.dspmicro.machinegateway.api;version=${import.c
      com.ge.dspmicro.machinegateway.api.adapter;version=${
      com.ge.dspmicro.machinegateway.types;version=${import
    <Service-Component>*</Service-Component>
  </instructions>
</configuration>

```





4. Build and deploy the maven project.

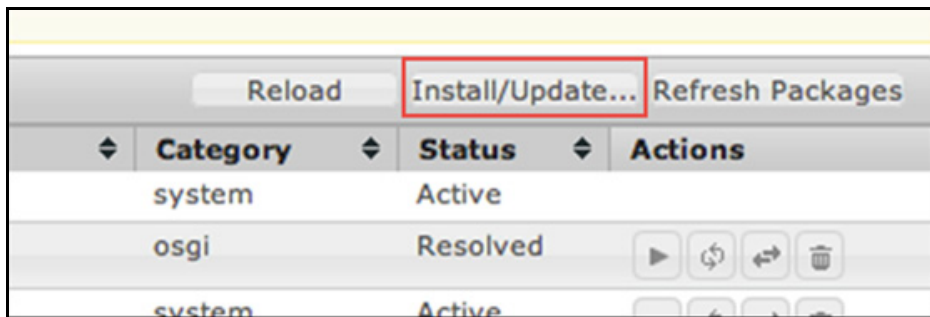
- Right-click on the *machinetraining* project and select **Run as > mvn install**
- In the Console tab, the following messages are displayed

```
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 5.746 s
[INFO] Finished at: 2016-01-15T17:45:08-08:00
[INFO] Final Memory: 22M/216M
[INFO] -----
```

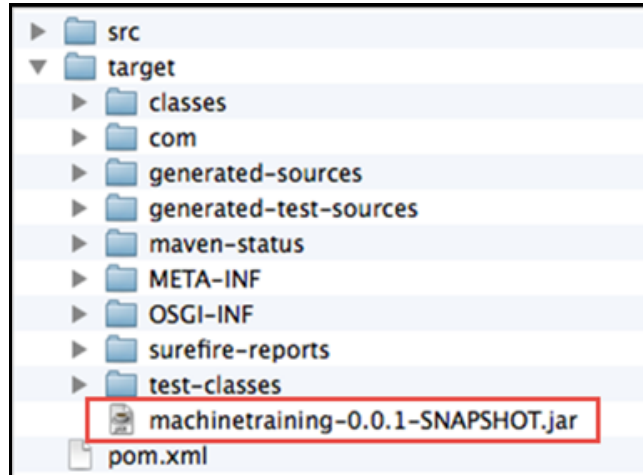
- ◆ Verify the BUILD SUCCESS message is displayed

5. Deploy the bundle to the machine services container using the Web Console.

- Start the machine container from the command line
- Reopen the Web Console
  - ◆ In a Firefox browser navigate to <https://localhost:8443/system/console>
  - ◆ Log in as username: **predix** password: **predix2machine**
- Click **Install/Update...**



- Select the bundle
  - ◆ **Browse** to the project's target directory located in the project workspace */predix/Documents/MyProject/machinetraining*
  - ◆ Select the bundle name **machinetraining-0.0.1-SNAPSHOT.jar**



- ◆ Click the **Install or Update**.

6. Start the bundle from the Web Console.

- Click on the Id column to sort the bundle list by descending order  
(If you do not see your bundle at the top, click **Reload**. The bundle should be in the *Installed* state)

Predix Machine Web Console  
Bundles

Main OSGi Predix Status System Technician Console Web Console

Bundle information: 158 bundles in total, 118 bundles active, 5 active fragments, 30 bundles resolved, 5 bundles installed

Apply Filter Filter All

Id	Name	Version	Category
157	machinetraing built from arch-basic-dspmicro-module (machinetraing)	0.0.1.SNAPSHOT	
156	webconsole-predixcloudenroll (com.ge.dspmicro.webconsole-predixcloudenroll)	15.3.0	

- ◆ Under the Actions bar, click the **Start** button
- ◆ You should see the message “*this is a test string*” in the terminal window where you started predix machine

```
2014-06-19 11:42:19,908[Component Resolve Thread (Bundle
14)]|INFO|com.ge.dspmicro.training.impl.SimpleServiceImpl|71-
machinetraing-
0.0.1.SNAPSHOT this is a test string
```

---

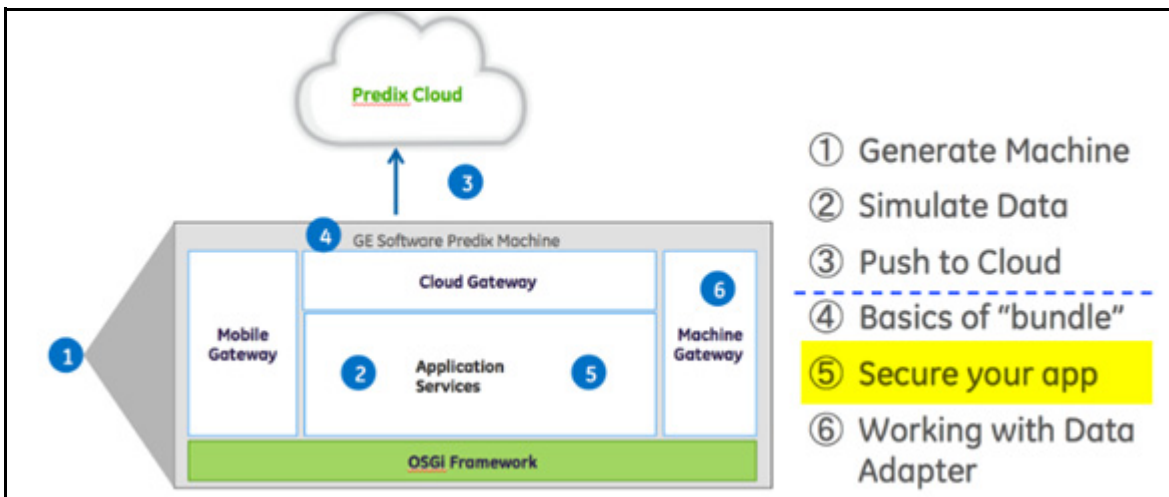
## Exercise 5: Adding Security to Bundles

---

### Learning Objectives

This goal of this lab is to provide hands on experience with:

- Adding a new machine service to your consumer bundle
- Using the security service to encrypt a variable in the consumer bundle configuration file



## Part I - Adding Security to a Consumer Bundle

In this exercise you add a new machine service to your consumer bundle. You then call the security utility APIs to encrypt the consumer bundle's password.

### Steps

1. Create a configuration file for the consumer bundle.

First you need to create the consumer's configuration file with a non-encrypted password.

- Go to `<predix-machine-debug>/etc` folder and create a new configuration file
  - ◆ File name: `com.ge.predix.solsvc.machinetraing.cfg`
- **TIP:** You may copy any of the existing `.cfg` files and remove the all the settings.
- Add the following 2 properties to the file

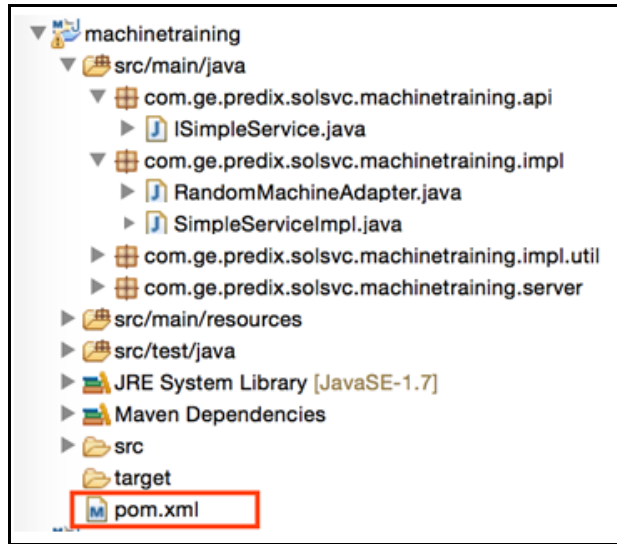
```
com.ge.predix.solsvc.machinetraing.uri=https://localhost:8443/testserver/test
com.ge.predix.solsvc.machinetraing.password=secretpassword
```

```
com.ge.predix.solsvc.machinetraing.uri=https://localhost:8443/testserver/test
com.ge.predix.solsvc.machinetraing.password=secretpassword
```

- Save and close the file

2. Add a security package to pom.xml file,

- In Eclipse, open the **pom.xml** file for the machinetraining consumer bundle



- Locate the comment `<!-- DSP micro API package versions -->`  
You will notice several `<import.com.ge.dspmicro.<service>.api>` statements. These define the range of supported API levels for the package that you are using in your application.
- Ensure that the following import is listed:
 

```
<import.com.ge.dspmicro.security.admin.api>"[1.1,2.0]"</import.com.ge.dspmicro.security.admin.api>
```
- In that same file locate the `<Import-Package>` tag
  - ◆ If not already there, add the following package at the end of the package list:
 

```
com.ge.dspmicro.security.admin.api;version=${import.com.ge.dspmicro.security.admin.api}
```

```

port -Package>aQuote.bnd.annotation.metatype;version=${import.aQuote.bnd.annotation.metatype},
javax.ws.rs.*;version=${import.javax.ws.rs},
org.slf4j;version=${import.org.slf4j};provider=paxlogging,
org.osgi.service.component;version=${import.org.osgi.service.component},
org.osgi.framework;version=${import.org.osgi.framework},
org.osgi.service.cm;version=${import.org.osgi.service.cm},
com.ge.predix.solsvc.machine.training.api;version=${import.com.ge.predix.solsvc.machine.train
com.ge.dspmicro.machinegateway.api;version=${import.com.ge.dspmicro.machinegateway.api},
com.ge.dspmicro.machinegateway.api.adapter;version=${import.com.ge.dspmicro.machinegateway.ap
com.ge.dspmicro.machinegateway.types;version=${import.com.ge.dspmicro.machinegateway.api},
com.ge.dspmicro.security.admin.api;version=${import.com.ge.dspmicro.security.admin.api}
Import -Package>

```

**Note:** Make sure you check the syntax and place a comma separating this package from the one above it.

- Save and close the file.

### 3. Create a variable for the configuration property.

- In the machine training project, locate the **SimpleServiceImpl.java** file
- Add the following static field to your **SimpleServiceImpl** class just below the existing variables for SERVICE\_PID and \_logger:

```

private static final String PROP_PASSWORD = SERVICE_PID +
    ".password";

```

(this variable corresponds to the value you added to the configuration file)



4. Add necessary code for consuming a new service.

- In **SimpleServiceImple.java** add another private field to the class as a handle to the security service:
 

```
private ISecurityUtils      secUtils;
```

  - ◆ If you get an error, press **ctrl+shift+o** to organize your imports
  - ◆ Select the following import: **com.ge.dspmicro.security.admin.api**
- Create a setter for this field annotated with `@Reference`
  - ◆ Add the following code for the setter method just below the `deactivate` method

```
@Reference
public void setSecurityUtils(ISecurityUtils secUtils)
{
    this.secUtils = secUtils;
}
```

- ◆ Press **ctrl+shift+o** to organize your imports
- ◆ Select **aQute.bnd.annotation.component** from the list to import the following package into your app:

```
import aQute.bnd.annotation.component.Reference;
```

## 5. Implement security APIs.

The activate method is guaranteed to be called when a service starts. This is a good place to encrypt a property.

- Add the highlighted code to the end of the activate method
 

```
        this.secUtils.encryptConfigProperty(SERVICE_PID, PROP_PASSWORD);
```
- In order to use the encrypted property, access the value through security utils.
  - ◆ Add the following code just below the line you added in your activate method above
 

```
        _logger.info("Original Value : " +
            String.valueOf(this.secUtils.getDecryptedConfigProperty(SERVICE_
            PID, PROP_PASSWORD)));
```

(Obviously you would never log a password in a real use case.)

- Your completed code should look like this

```

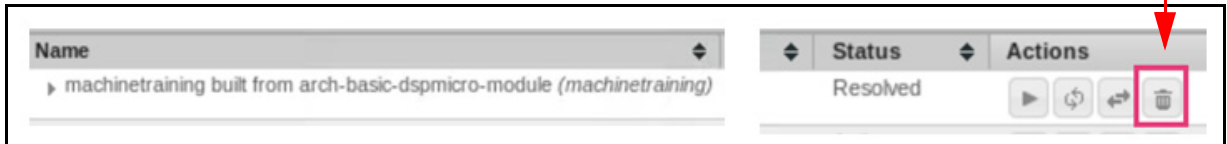
@Activate
public void activate(ComponentContext ctx)
{
    _logger.info(Messages.getString("SimpleServiceImpl.test_message")); //NON-NLS-1$
    //initialize service
    if ( _logger.isDebugEnabled() )
    {
        _logger.debug("Service Started"); //NON-NLS-1$
    }

    this.secUtils.encryptConfigProperty(SERVICE_PID, PROP_PASSWORD);
    _logger.info("Original Value : " + String.valueOf(this.secUtils.getDecryptedConfigProperty(
        SERVICE_PID, PROP_PASSWORD)) );
}

```

## 6. Build the project and deploy the bundle.

- Delete the **machinetraining.jar** from the running Predix Machine container before importing the updated app.



- Deploy the updated bundle
  - ◆ Select the **Install/Update** button
  - ◆ Browse to the updated **machinetraining-0.0.1-SNAPSHOT.jar** file and select open
  - ◆ Select “Install or Update” to import the bundle
  - ◆ Click **Start** to launch the bundle.
- You will see the original password displayed in the terminal window where machine is running

```

0" started.
2016-03-23 12:46:53,419[Component Resolve Thread (Bundle 34)]|INFO|com.ge.predix.solsvc.machinetraini
impleServiceImpl|118-machinetraining-0.0.1.SNAPSHOT|this is a test string
2016-03-23 12:46:53,421[Component Resolve Thread (Bundle 34)]|INFO|com.ge.dspmicro.security.admin.imp
yUtilsImpl|23-com.ge.dspmicro.securityadmin-15.3.0|Property "com.ge.predix.solsvc.machinetraining.pass
been encrypted
2016-03-23 12:46:53,423[Component Resolve Thread (Bundle 34)]|INFO|com.ge.predix.solsvc.machinetraini
impleServiceImpl|118-machinetraining-0.0.1.SNAPSHOT|[Original Value : secretpassword]
2016-03-23 12:46:53,450[Component Resolve Thread (Bundle 34)]|INFO|org.glassfish.jersey.server.Applic

```

7. Verify that the property has been encrypted.

- Open the **machinetraining.cfg** file in the predix-machine-debug/etc folder  
The password field has been wiped out and an encrypted property has been appended to the file.
- The configuration file will look like the following:

```
com.ge.predix.solsvc.machinetraining.password =  
com.ge.predix.solsvc.machinetraining.password.encrypted =  
mTN/VnEVdqNFAOt4Ld1URBSd1JEtZdhTwVbzAdzGCwg=
```

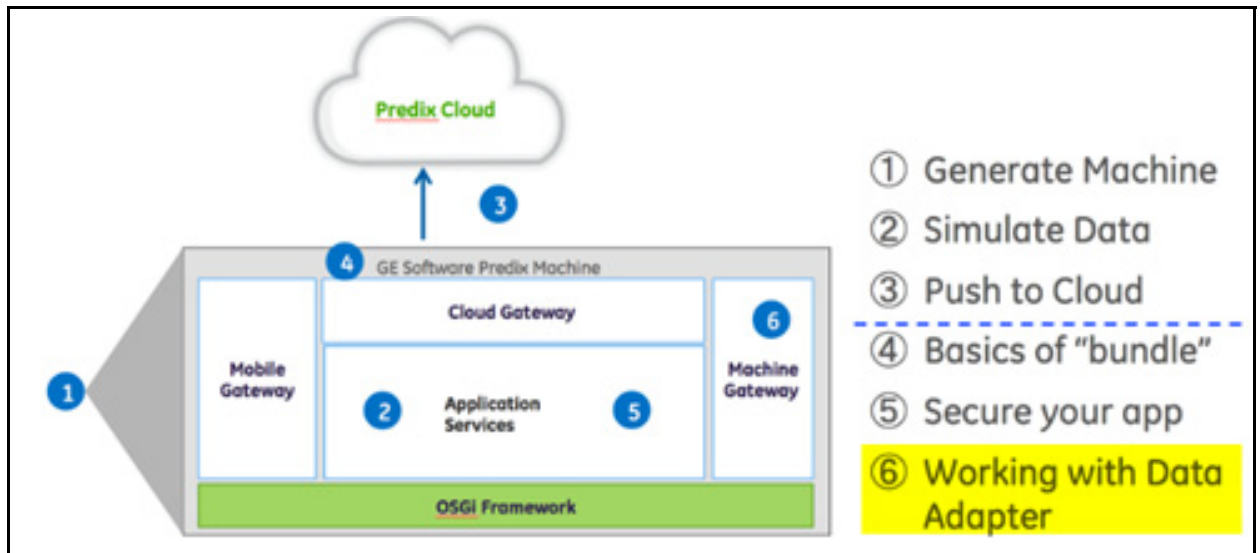
The password is clearly no longer human readable.

## Exercise 6: Generate Gateway Data with the Machine Adapter

### Learning Objectives

At the end of this lab you will be able to:

- Read data from an adapter
- Create a simple adapter that returns random data
- Implement the interface and register the service with Machine Gateway



## Part I - Configure a Consumer Bundle

In this exercise you configure a consumer bundle.

### Steps

1. Verify that the machinegateway packages are imported into the consumer bundle.

- In Eclipse, open the **pom.xml** file
  - ◆ Locate the comment `<!-- DSP micro API package versions -->`  
Notice several `<import.com.ge.dspmicro.<service>.api>` statements
  - ◆ Ensure that the following import is listed:
 

```
<import.com.ge.dspmicro.machinegateway.api>"[1.1,2.0]"</import.com.ge.dspmicro.machinegateway.api>
```
- Locate the `<Import-Package>` tag
  - ◆ If not already there, add the following 3 packages at the end of the package list
 

```
com.ge.dspmicro.machinegateway.api;version=${import.com.ge.dspmicro.machinegateway.api},

com.ge.dspmicro.machinegateway.api.adapter;version=${import.com.ge.dspmicro.machinegateway.api},

com.ge.dspmicro.machinegateway.types;version=${import.com.ge.dspmicro.machinegateway.api}
```

**Note:** Make sure you check the syntax and place a comma separating these packages from the ones above.

## 2. Add necessary code for consuming a new service.

- If not already there, add the private field for Machine Gateway into your **SimpleServiceImpl** class

```
private IMachineGateway gateway;
```

- Add the setter for the machine gateway field annotated with `@Reference`

```
@Reference
```

```
public void IMachineGateway gateway)
```

```
{
```

```
    this.gateway = gateway;
```

```
}
```

- Save your work

## 3. Implement the IMachineAdapter Interface.

An implementation has been provided for you as part of the archetype. Take a look at **RandomMachineAdapter.java**. It provides a mock implementation of a machine adapter. When `readData` is called from this adapter, it will return a random integer wrapped in the **PDataValue** class.

In a real use case, this interface should be implemented using a Java library for whatever device technology with which this adapter will interface.

## 4. Retrieve the adapter through the Machine Gateway API.

**Adapter type** identifies an adapter. A common convention, which is also used by the Random Machine Adapter, is to use the service PID as the adapter type.

- Add this private field to the **SimpleServiceImpl** class

```
private UUID randomNodeId;
```



- Insert this code into your activate method, below the code you added in the previous labs

```

for (IMachineAdapter adapter :
    this.gateway.getMachineAdapters()) {
    if ( adapter.getInfo() == null )
    {
        continue;
    }
    if (
RandomMachineAdapter.SERVICE_PID.equals(adapter.getInfo().getAdapterType()) )
    {
        this.randomNodeId =
adapter.getNodes().get(0).getNodeId();
break;
    }
}

```

Now that you have the nodeId of the single node in the adapter, you can read it directly from the MachineGateway API. There are two ways to read data: either directly from the MachineAdapter or through MachineGateway. In this example you will read from MachineGateway so you do not have to store the reference to the adapter.

- Add the following code just below the `for{}`  loop you added in the previous step

```

PDataValue value =
this.gateway.readData(this.randomNodeId);

_logger.info("Reading Random Value " +
value.getEnvelope());

```

- Import **PDataValue** from `com.ge.dspmicro.machinegateway.types` if needed



- Your completed code should look something like this

```

this.secUtils.encryptConfigProperty(SERVICE_PID, PROP_PASSWORD);
_logger.info("Original Value : " + String.valueOf(this.secUtils.getDecryptedConfigProperty(
    SERVICE_PID, PROP_PASSWORD)) );

for (IMachineAdapter adapter :
    this.gateway.getMachineAdapters()) {
    if (adapter.getInfo() == null) {
        continue;
    }
    if (RandomMachineAdapter.SERVICE_PID.equals(adapter.getInfo().getAdapterType())) {
        this.randomNodeId = adapter.getNodes().get(0).getNodeId();
        break;
    }
}
PDataValue value = this.gateway.readData(this.randomNodeId);
_logger.info("Reading Random Value " + value.getEnvelope());

```

## 5. Build and deploy your bundle.

- Build the machinetraing project (**Run As > maven install**)
- From the Web Console, delete the existing machinetraing bundle
- Select Import/Install and browse to the updated machinetraing.0.0.1-SNAPSHOT.jar
- Import the updated bundle and start

```

predix@localhost:~/PredixApps/training_labs/machine/predixsdk-15.3.0/utilities/containers/prec
File Edit View Search Terminal Help
fw>$
fw>$
fw>$2016-03-23 14:41:59,776[FW Buff Logger]||INFO|com.prosyst.mbs.system.bundle|22-com.prosyst.s
-1.0.0|Bundle with id #119 machinetraing, 0.0.1.SNAPSHOT was started.
#0 INFO > Bundle with id #119 machinetraing, 0.0.1.SNAPSHOT was started.
2016-03-23 14:41:59,785[Component Resolve Thread (Bundle 34)]||INFO|com.ge.dspmicro.machinegate
atewayImpl|108-com.ge.dspmicro.machinegateway-impl-15.3.0|Machine adapter "455f81a7-ffa1-4134-1
" started.
2016-03-23 14:41:59,790[Component Resolve Thread (Bundle 34)]||INFO|com.ge.predix.solsvc.machin
impleServiceImpl|119-machinetraing-0.0.1.SNAPSHOT|this is a test string
2016-03-23 14:41:59,790[Component Resolve Thread (Bundle 34)]||INFO|com.ge.predix.solsvc.machin
impleServiceImpl|119-machinetraing-0.0.1.SNAPSHOT|Original Value : secretpassword
2016-03-23 14:41:59,791[Component Resolve Thread (Bundle 34)]||INFO|com.ge.predix.solsvc.machin
impleServiceImpl|119-machinetraing-0.0.1.SNAPSHOT|Reading Random Value 445106119
2016-03-23 14:41:59,832[Component Resolve Thread (Bundle 34)]||INFO|org.glassfish.jersey.serv

```

---

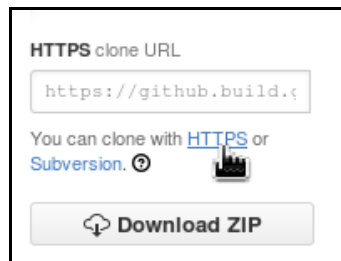
## **Boot Camp Appendix**

---

## Installing Px Theme Components from GitHub

**Note:** The following list is for your information only. It indicates the steps that you would normally take to set up your px-theme project. These were completed for you in order to provide the Px elements required for the lab exercises.

- Connect to the px-theme repository on GitHub  
You need a GitHub account as well as a Predix.io account to do this.
- On the right-hand side of the web page, click the HTTPS link under the **HTTPS** clone URL



- Copy the link in the box. You will use this in the next command in the Terminal window
- In the Terminal, change to directory into which the project should install
  - ◆ `cd ~/predix/PredixApps/training_labs`
  - ◆ Clone the px-theme library:
 

```
git clone https://github.com/PredixDev/px-theme.git
```

 In the Terminal, you should receive confirmation that this has completed correctly.
- In the Terminal, run the following commands to set up your px-theme project:
  - ◆ `cd px-theme`
  - ◆ `npm install`
  - ◆ `bower install`

The `npm` and `bower install` commands install dependencies for the application in the `px-theme` directory.

**Tip:** Look at the repository in GitHub for the PXd library to find styles or elements that can be used in your applications. You'll use some styles from the `px-forms-design` component

- In your browser, open the `px-forms-design` library in GitHub
- Copy the “bower install” line from the Readme information under the “Installation” heading
  - ◆ In the Terminal, run this command:
 

```
bower install --save https://github.com/PredixDev/px-forms-design.git
```

 This installs the `px-forms-design` module and its dependencies.

## Creating a Web Component

- To Download the Yeoman generator for the Predix component, in the Terminal, run the following commands:

```
git clone https://github.com/PredixDev/generator-px-comp.git
cd generator-px-comp
npm link
```

The `git clone` command clones the generator from GitHub. The `npm link` command installs the `generator-px-comp`.

- To create a directory and run the Predix component generator, in the Terminal, run the following commands:

```
mkdir hospital-info
cd hospital-info
yo px-comp
```

The `yo px-comp` command runs the generator, which creates the new component. During this creation, you will be prompted for a component name, mix-ins, and PXd Sass modules. Answer the questions as follows:

- ◆ ? What is the component's name, must have a "-", e.g. 'px-thing'?
 

Enter **my-table** and press **Enter**

- ◆ ?Optional: Local paths to mix-ins the component uses, comma-separated (e.g. './px-my-mixin,..px-my-other-mixin')

Press **Enter**

- ◆ ?Which of these common Pxd Sass modules does your component need? (You can add more later in bower.json)

Use your arrow keys to scroll down to **Tables** and press the Space Bar to select **Tables**

- ◆ Press **Enter**

```
[predix@localhost hospital-info]$ yo px-comp

Hello! Answer the prompts to scaffold a Px component.

The generated component itself is not fancy (it makes a circle on the screen that increments a counter when clicked), but contains the Bower config, Gruntfile, tests, etc. common to all Px components...

? What is the component's name, must have a "-", e.g. 'px-thing'? my-table
? Optional: Local paths to mix-ins the component uses, comma-separated (e.g. './px-my-mixin,..px-my-other-mixin')
? Which of these common PXD Sass modules does your component need? (You can add more later in bower.json)
   Buttons
   Lists
   Forms
   Headings
   Tables
```

- To download and install the Bower dependencies for the component, in the Terminal, run the following commands:

- ◆ **bower install**
- ◆ **bower link**

