

Predix

Predix Developer Boot Camp

Student Lab Guide

June 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

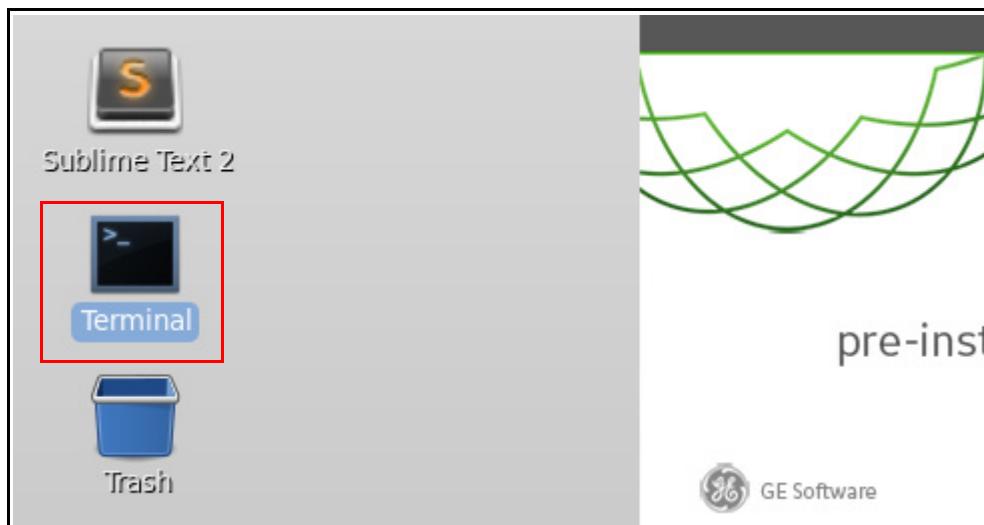
Overview

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 version 1.0)
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       |
|--------------------------------------------------------|-------------------|-------------------|
| logstash-5                                             | free              | Logstash 1.4 serv |
| opment and testing                                     |                   |                   |
| p-rabbitmq                                             | standard          | RabbitMQ is a rob |
| formance multi-protocol messaging broker.              |                   |                   |
| postgres                                               | shared, shared-nr | Reliable PostgrSQ |
| predix-acs                                             | Beta, Enterprise* | Use this service  |
| l framework than basic User Account and Authorization. |                   |                   |
| predix-analytics-catalog                               | Beta, Enterprise* | Add analytics to  |
| with the Analytics Runtime Service.                    |                   |                   |
| predix-analytics-runtime                               | Beta, Enterprise* | Use this service  |
| ion of the analytic orchestration.                     |                   |                   |
| predix-asset                                           | 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** (or **cf s**) 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 12
- *Managing your Environment*, page 15
- *Monitoring your Application*, page 16

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

FAILED

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

The command fails because you have not provided the parameters it needs.

- 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 **ARchive (.war)**, or wrapper file with all of the application files required for deployment.

```
[predix@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
running  2016-03-10 09:01:07 AM  0.0%  641.9M of 1G  153.3M of 1G
```

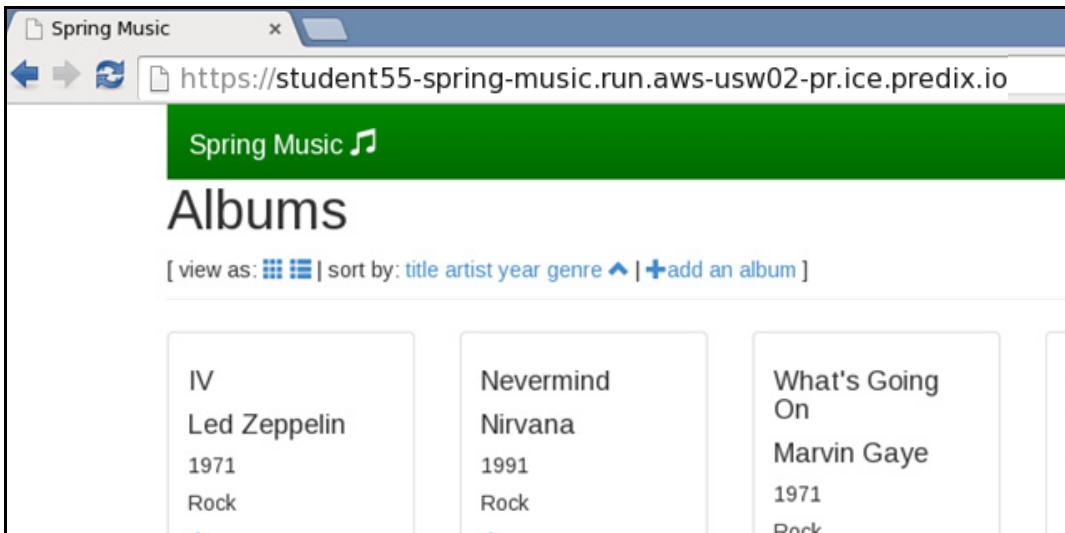
1. Test your application in a web browser

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

name	requested state	instances	memory
urls			
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			

- Copy your application URL (located in the URL column of the output)
- Open the Firefox Web browser, and go to **https://<paste your application URL>**

The Spring Music application displays in your browser.



Exercise 2: Using a Manifest File to Deploy an Application

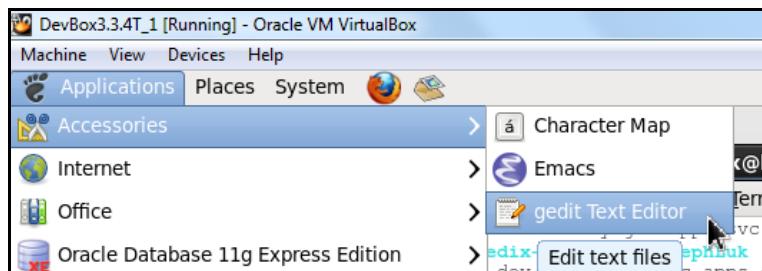
Overview

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

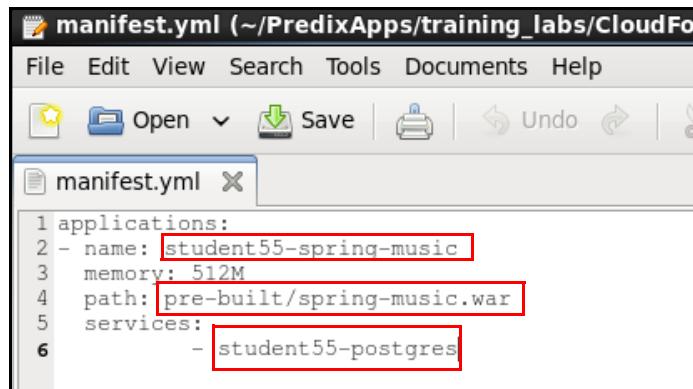
name	requested state	instances	memory
urls			
awsblob	started	1/1	1G
awsblob.run.aws-usw02-pr.ice.predix.io			
dataingestion_service_student1	started	1/1	1G
dataingestion-service-student1.run.aws-usw02-pr.ice.predix.io			
rmd_datasource_student1	started	1/1	1G
rmd-datasource-student1.run.aws-usw02-pr.ice.predix.io			
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)



A screenshot of a code editor window titled "manifest.yml (~/PredixApps/training_labs/CloudFo)". The window shows the following manifest file content:

```
1 applications:
2 - name: student55-spring-music
3   memory: 512M
4   path: pre-built/spring-music.war
5   services:
6     - student55-postgres
```

The lines from 2 to 6 are highlighted with red boxes.

Note: In the manifest file, you are binding the spring-music application to your postgres service

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 environment variables, scale, stop, and delete your application instance.

Steps

1. View the environment variables for your application.

- In the Terminal, run `cf env <your-application-name>`
- Note the environment variables for your application, including the postgres service instance

2. Stop the application in Cloud Foundry.

- In the Terminal, run `cf stop <your-application-name>`
- Example:** `cf stop student55-spring-music`

3. Scale your application up to 3 instances.

- Run `cf scale <your application name> -i 3`
- Run `cf scale <appName>` to display the change
- 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`

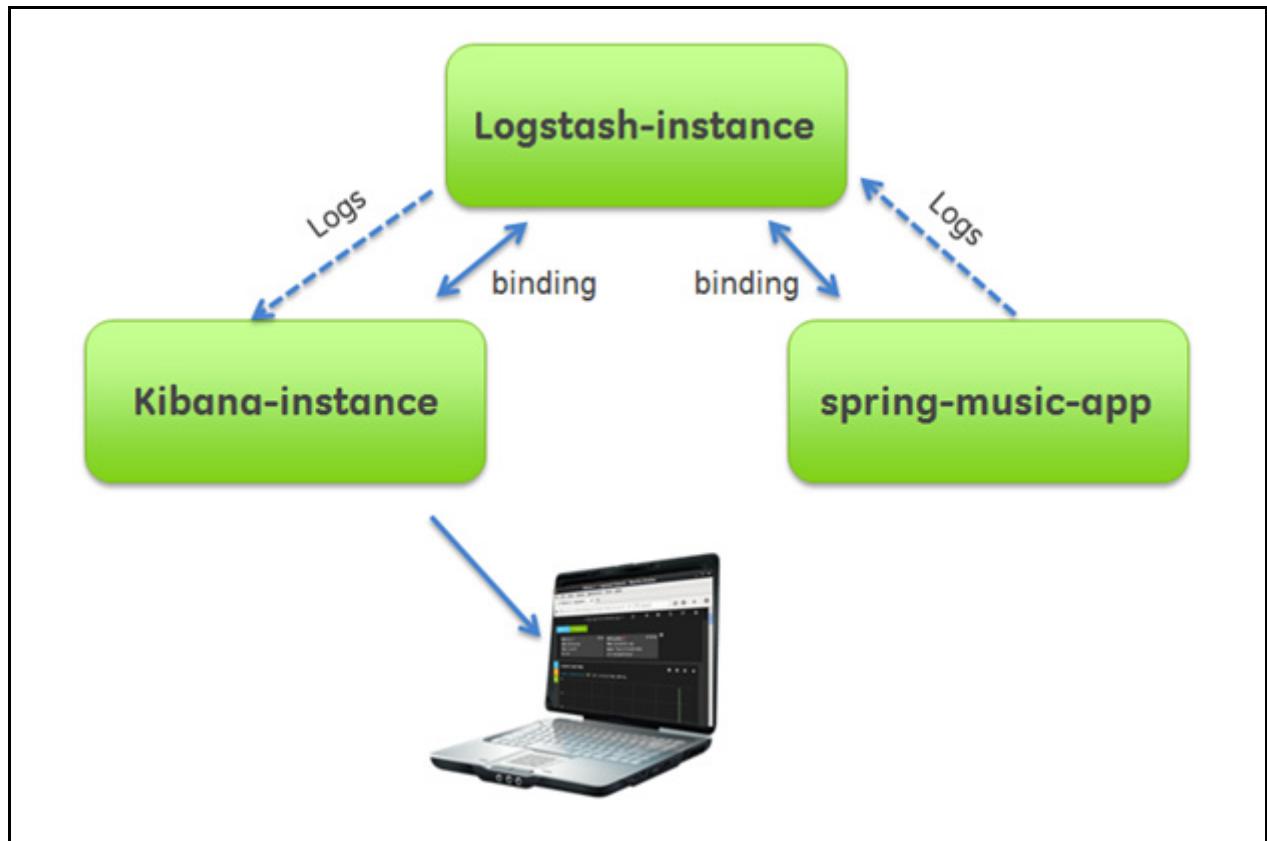
Note: This deletes the application and any orphaned routes from Cloud Foundry.



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 - be patient it may take a minute to complete

- 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 restart 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-pric
e.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.

- In a browser go to **https://<url of your Kibana application>**
 - Your application shows logging information from your spring-music application

Kibana 3 - Logstash ... x +

[kibana-xander-wobegone-diopsimeter.run.aws-usw02-pr.ice.predix.io/#/dashboard/file/apps-logs.json](#)

Logstash Search

QUERY FILTERING ▶

time must ●

field :@timestamp

from : now-24h

to : now

field mustNot ●

field : syslog5424_app

query :"9acc1c73-bc2f-4065-a27a-eb3c8d57b7ad"

EVENTS OVER TIME

View ▶ | Zoom Out | ● * (49) count per 10m | (49 hits)

50

ALL EVENTS

Fields ↻

All (33) / Current (21) 0 to 49

Type to filter...

Note These fields have been extracted from your mapping. Not all fields may be available in your source document.

- @message
- @message.raw
- @source_host
- @source_host.raw

_source (select columns from the list to the left)

```
{"message":287 <14>1 2015-12-11T19:48:35.131971+00:00 loggregator d31112b7-7086-422
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 d31112b7-7086-422
of type [class org.springframework.web.servlet.mvc.ParameterizableViewController],"@versio
{"message":333 <14>1 2015-12-11T19:48:34.915666+00:00 loggregator d31112b7-7086-422
/kill],methods=[],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto public void or
{"message":373 <14>1 2015-12-11T19:48:34.91492+00:00 loggregator d31112b7-7086-422
"\\"[info],methods=[],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto public org
```

Lab 3: Building Microservices

Learning Objectives

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

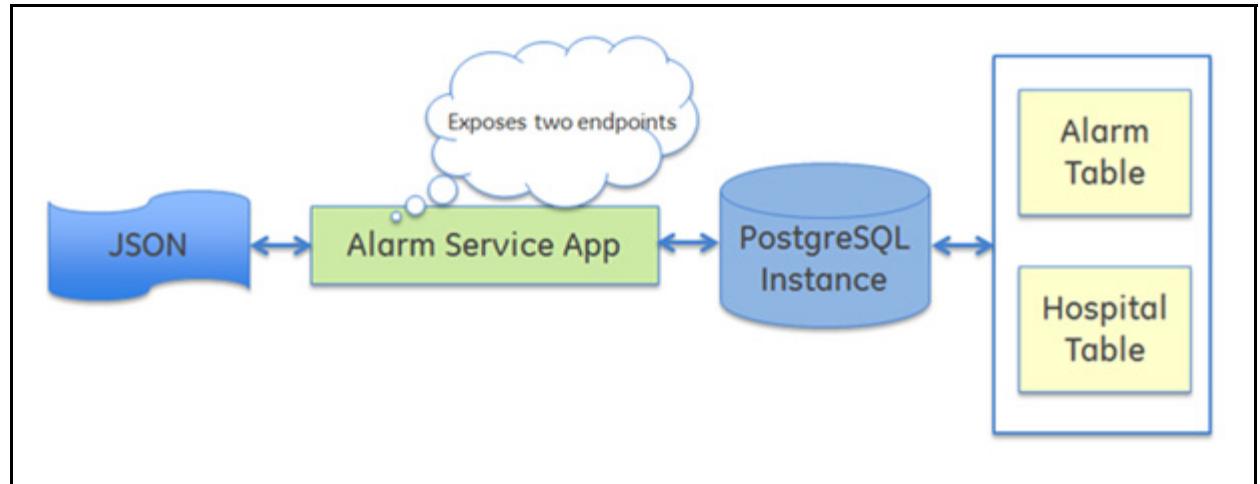
- Use the manifest file to deploy an application
- Build and deploy a Java microservice to the Predix Cloud
- Provide a UI for the microservice

Lab Exercises

- [Adding a Maven Archetype, page 22](#)
- [Adding an Additional API Endpoint to a Microservice, page 33](#)

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.



```
File Edit View Search Terminal Help
[predix@localhost ~]$ sudo /etc/init.d/postgresql-9.3 start
Starting postgresql-9.3 service: [ OK ]
```

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

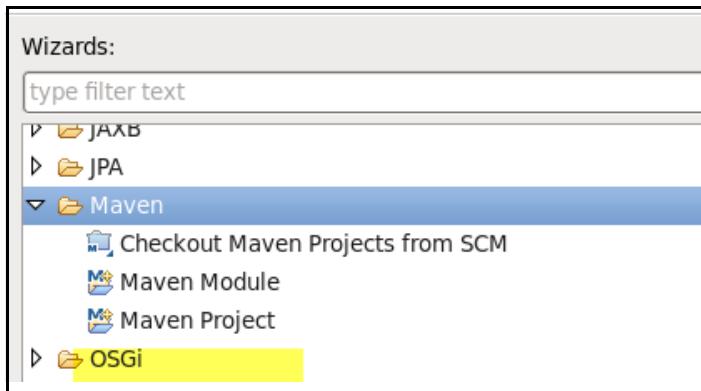
The local postgres service is started because it is required to build the alarmservice project in the Eclipse-STS tool (your next task). This is not the same as the service instance of postgres that you created in Cloud Foundry.

2. Create a new maven project

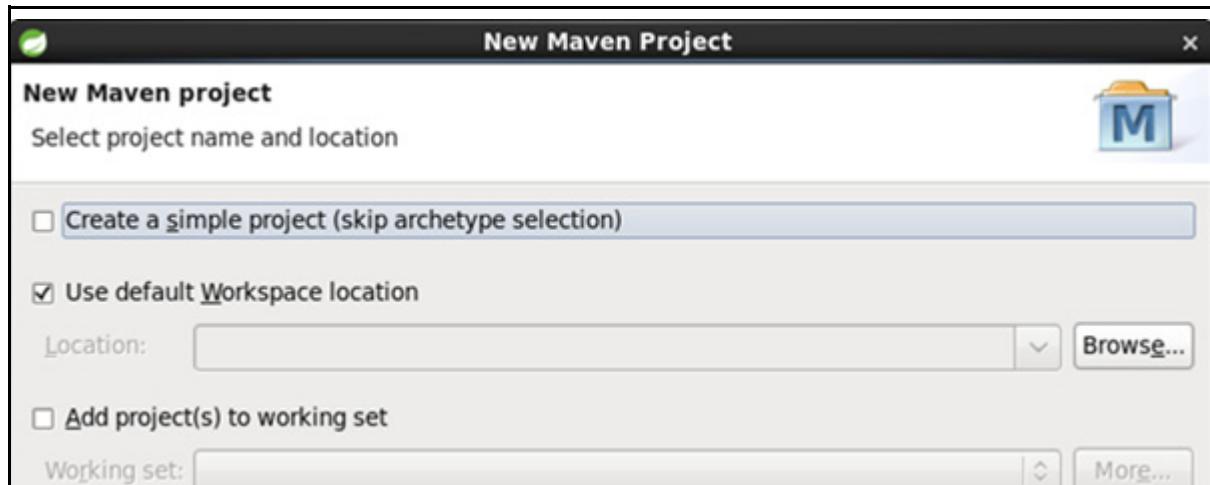
- Open **Eclipse-STS** by double clicking the icon on the desktop



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

New Maven project

Select an Archetype

Catalog: All Catalogs

Filter: `alarm`

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

- Enter the archetype parameter as shown below:

New Maven project

Specify Archetype parameters

Group Id: `com.ge.predix.solsvc.training`

Artifact Id: `alarmservice`

Version: `0.0.1-SNAPSHOT`

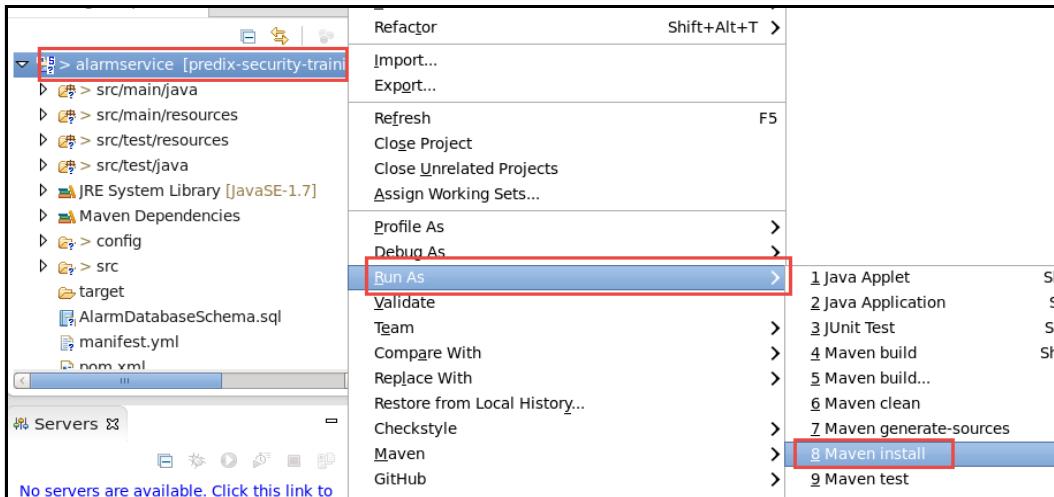
Package: `com.ge.predix.solsvc.training.alarmservice`

Properties available from archetype:

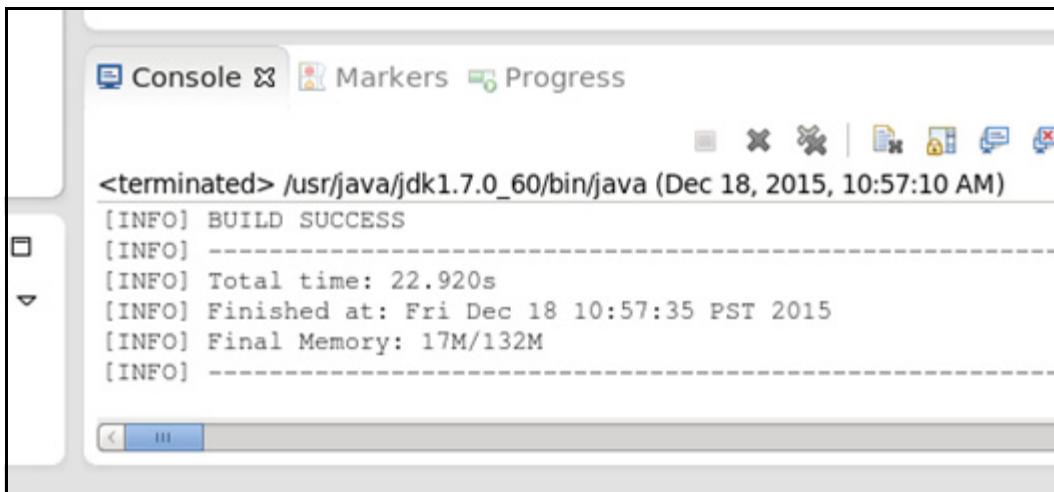
- Click **Finish**

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



- Double-click on the Console tab to maximize the console window

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.

```
--- maven-jar-plugin:2.5:jar (default-jar) @ alarmservice ---
Building jar: /predix/Documents/MyProject/alarmservice/target/alarmservice-0.0.1-SNAPSHOT.jar
--- spring-boot-maven-plugin:1.2.5.RELEASE:repackage (default) @ alarmservice ---
--- maven-install-plugin:2.5.2:install (default-install) @ alarmservice ---
Installing /predix/Documents/MyProject/alarmservice/target/alarmservice-0.0.1-SNAPSHOT.jar to
Installing /predix/Documents/MyProject/alarmservice/pom.xml to /predix/.m2/repository/com/ge/
-----
BUILD SUCCESS
-----
Total time: 13.512s
```

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



The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer view displays the project structure for 'alarmservice'. It includes standard Java source and resource folders, along with 'JRE System Library [JavaSE-1.7]', 'Maven Dependencies', and configuration files like 'config' and 'src'. Two specific files are highlighted: 'AlarmDatabaseSchema.sql' and 'manifest.yml'. The right-hand side of the interface shows the contents of the 'manifest.yml' file in a code editor. The file is a YAML document defining an application named 'predix-alarmservice-student55' using a 'java_buildpack'. It specifies a path to the target JAR file ('target/alarmservice-0.0.1-SNAPSHOT.jar') and lists a service named 'student55-postgres'. An environment variable 'db_service_name' is also defined as 'student55-postgres'. The 'services' and 'db_service_name' lines are currently selected.

```
---
```

```
applications:
```

```
  - name: predix-alarmservice-student55
```

```
    buildpack: java_buildpack
```

```
    path: target/alarmservice-0.0.1-SNAPSHOT.j
```

```
    services:
```

```
      - student55-postgres
```

```
env:
```

```
  db_service_name: student55-postgres
```

- 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/alarmservice-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 database services element to the name of your postgresql service instance. This element is found in the Java application
 - ◆ 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, use the `cf services` command to list all service instances, and find yours.

4. View the db_service_name in your Java application

- In Eclipse, open the `DatabaseConfigCloud.java` file under `src/main/java` under the `com.ge.predix.solsvc.training.alarmservice.jpa` package
- Note the `@Profile` annotation is "cloud" when you are deploying to the cloud

```
@Component  
@Configuration  
@Profile("cloud")  
public class DatabaseConfigCloud extends DataSource implements EnvironmentAware{
```

- Note that the "db_service_name" (user-defined environment variable) comes from the manifest file you updated

```
@Override  
public void setEnvironment(Environment env) {  
    String dbServiceName = env.getProperty("db_service_name");
```

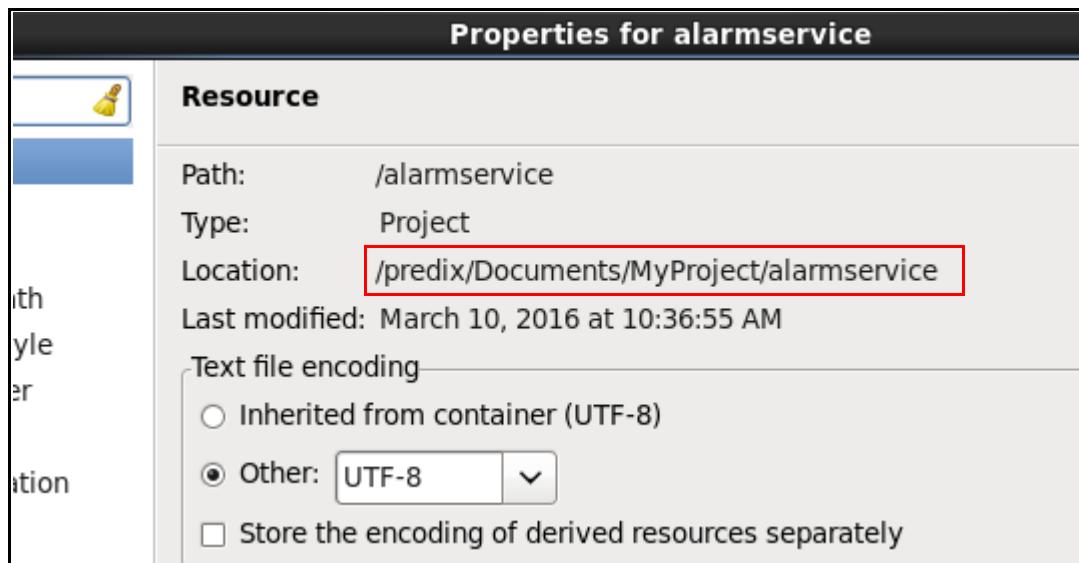
- This file also sets the URI, username and password from the VCAP services environment variables

```
String envURL = env.getProperty("vcap.services."+dbServiceName+".credentials.uri");  
System.out.println("DB Url : "+envURL);  
setUrl(envURL);  
String envUserName = env.getProperty("vcap.services."+dbServiceName+".credentials.username");  
System.out.println("DB userName : "+envUserName);  
if (envUserName != null) {  
    setUsername(envUserName);  
  
String envUserPassword = env.getProperty("vcap.services."+dbServiceName+".credentials.password");  
System.out.println("DB userPassword : "+envUserPassword);  
if (envUserPassword != null) {  
    setPassword(envUserPassword);
```



5. 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-alarmservice-student55 in org Predix
aining / space Training1 as student55...
OK

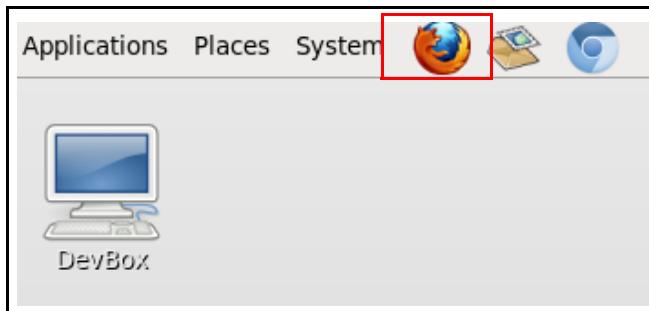
requested state: started
instances: 1/1
usage: 1G x 1 instances
urls: predix-alarmservice-student55.run.aws-usw02-pr.ice.predix.io
last uploaded: Thu Mar 10 18:51:01 UTC 2016
stack: cflinuxfs2
buildpack: java_buildpack

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



6. 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
 - ◆ Prepend **https://** to the beginning of the URL
 - ◆ Append “**/alarmservice**” to the end of your URL and press **Enter**

The content of the alarm table appears

```
arm": "ARTIFACT", "alarmClassification": "TECHNICAL", "patient":  
ientId": 1, "title": null, "firstName": "Mike", "midInit": "", "lastName": "Waldman", "email": "  
irthDate": null}, "eventId": 1, "priority": 1, "numberOfAlarms": 0},  
rm": "PVC", "alarmClassification": "ARRHYTHMIA", "patient":  
ientId": 1, "title": null, "firstName": "Mike", "midInit": "", "lastName": "Waldman", "email": "  
irthDate": null}, "eventId": 2, "priority": 4, "numberOfAlarms": 0},  
rm": "COUPLET", "alarmClassification": "ARRHYTHMIA", "patient":  
ientId": 1, "title": null, "firstName": "Mike", "midInit": "", "lastName": "Waldman", "email": "  
irthDate": null}, "eventId": 3, "priority": 4, "numberOfAlarms": 0},  
rm": "PVC", "alarmClassification": "ARRHYTHMIA", "patient":  
ientId": 1, "title": null, "firstName": "Mike", "midInit": "", "lastName": "Waldman", "email": "  
irthDate": null}, "eventId": 4, "priority": 4, "numberOfAlarms": 0}, {"alarm": "SPO2
```

7. 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-alarmservice-student55 --recent
Connected, dumping recent logs for app predix-alarmservice-student55 in org Pred
ix-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-alarmservice-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
.1 | 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

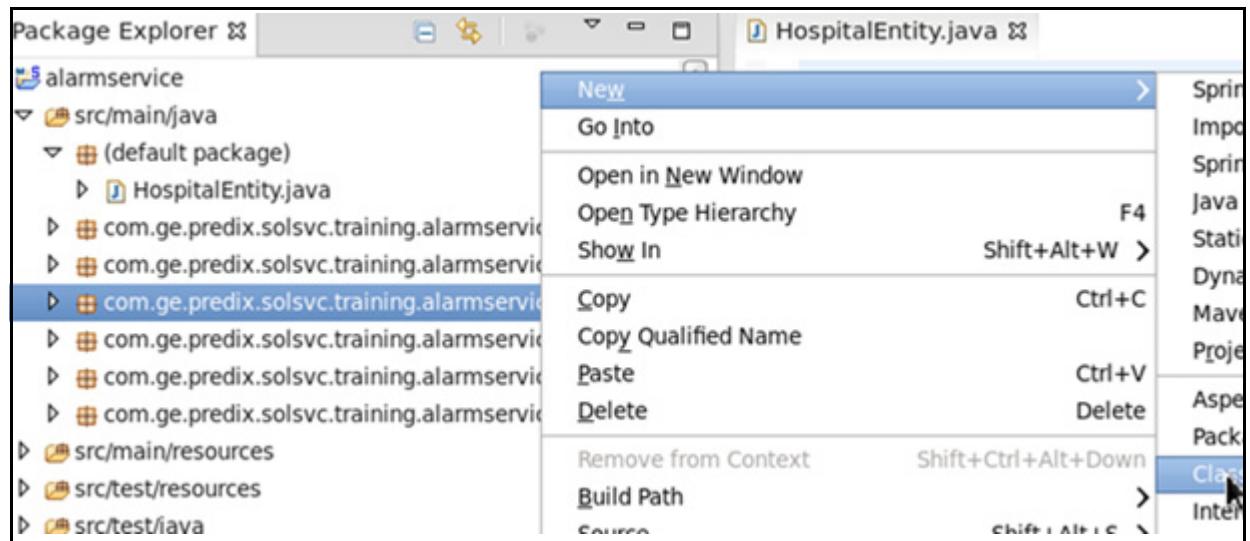
Overview

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

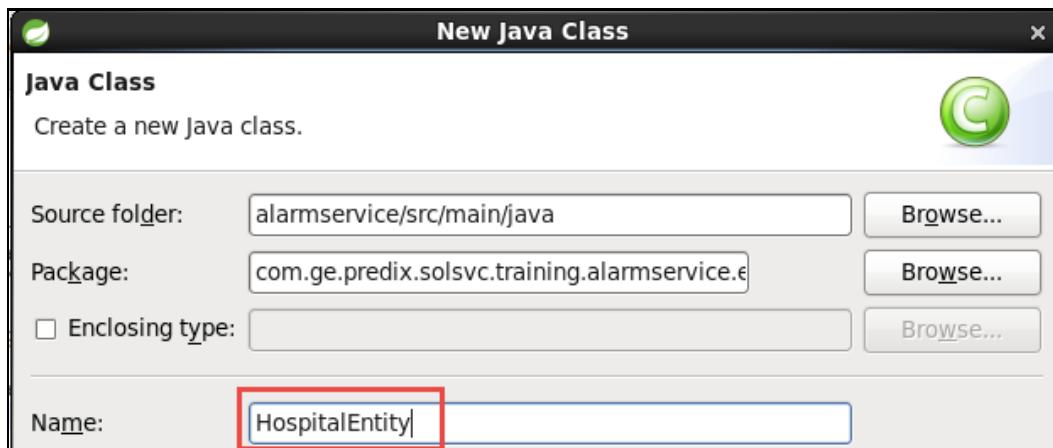
Steps

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**



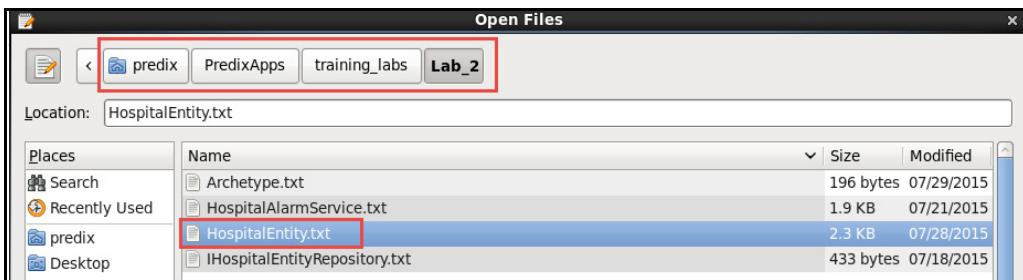
- Click **Finish**

The entity is created

```
package com.ge.predix.solsvc.training.alarmservice.entity;

public class HospitalEntity {
```

- In gedit, open the file
/predix/predixApps/training_labs/fundamentals/Lab2/HospitalEntity.txt

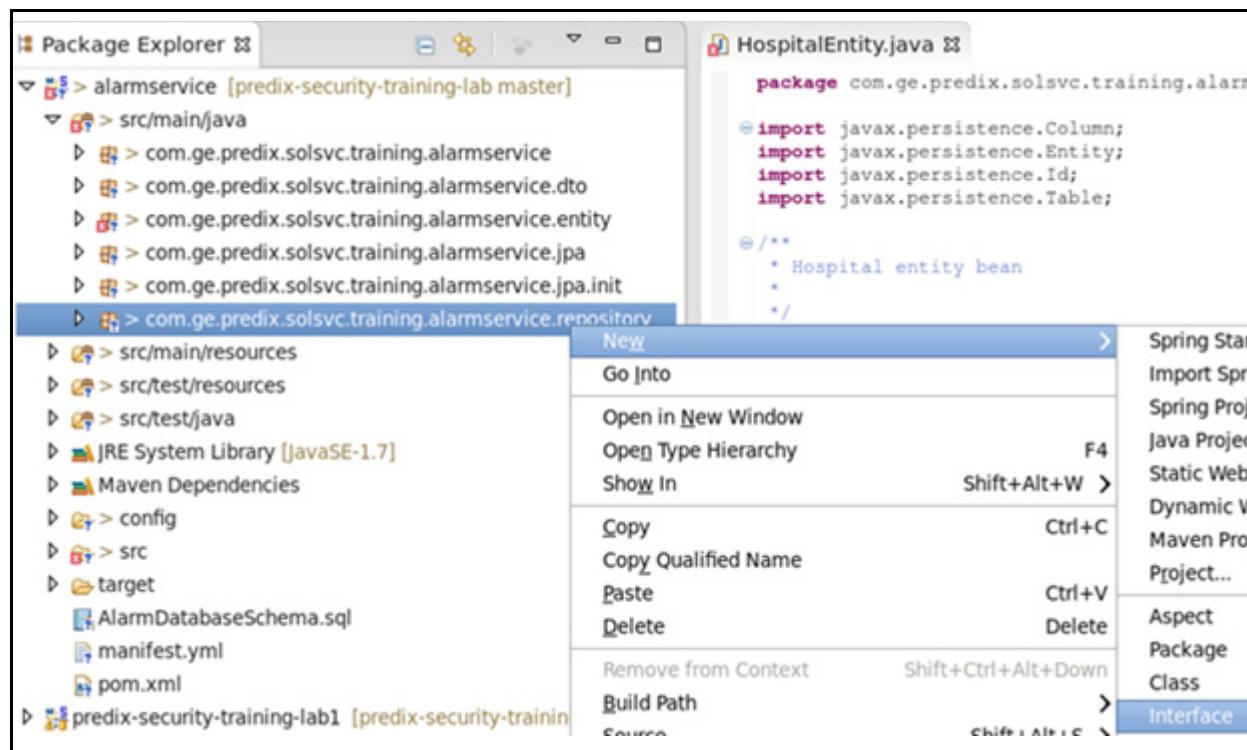


- Copy all content of the file

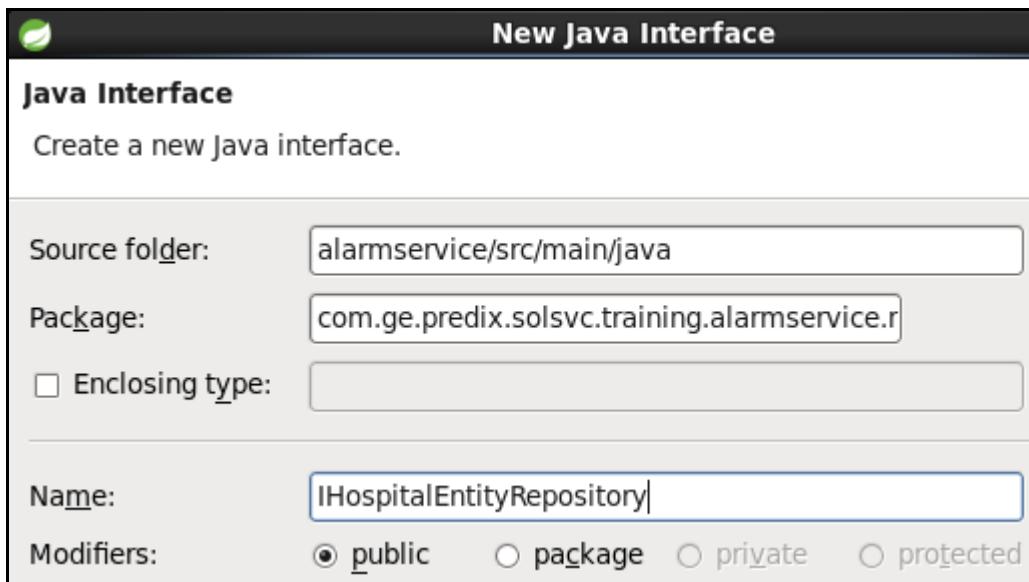
- In Eclipse, replace everything in your **HospitalEntity.java** class
 - ◆ Press <ctrl> + <shift> + <o> to Organize Imports
 - ◆ Press <ctrl> + <s> to save the file

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*



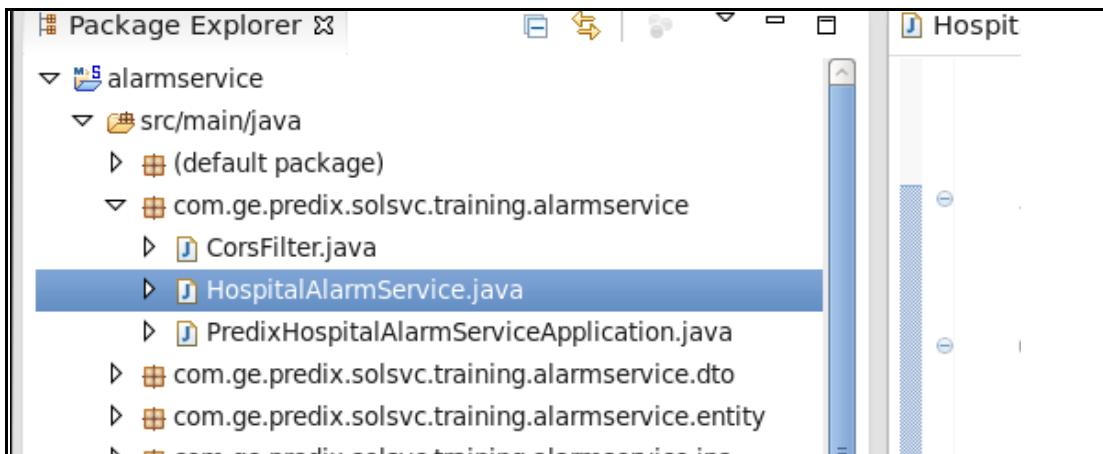
- In the **Name** field, enter **IHospitalEntityRepository**, then click **Finish**



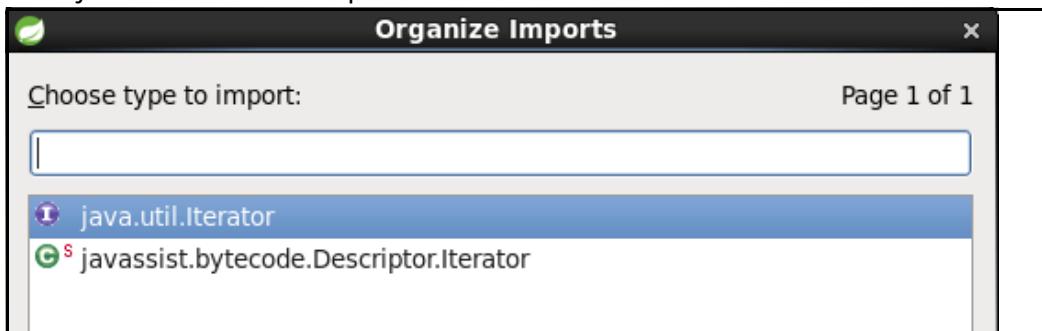
- In gedit, open the file
`/predix/predixApps/training_labs/Lab_2/IHospitalEntityRepository.txt`
- Copy all content of the file
- In Eclipse, replace everything in your **IHospitalEntityRepository.java** class
- Press `<ctrl> + <shift> + <o>` to Organize Imports
- Press `<ctrl> + <s>` to save the file

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/fundamentalsLab_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**



- 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.jpa.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

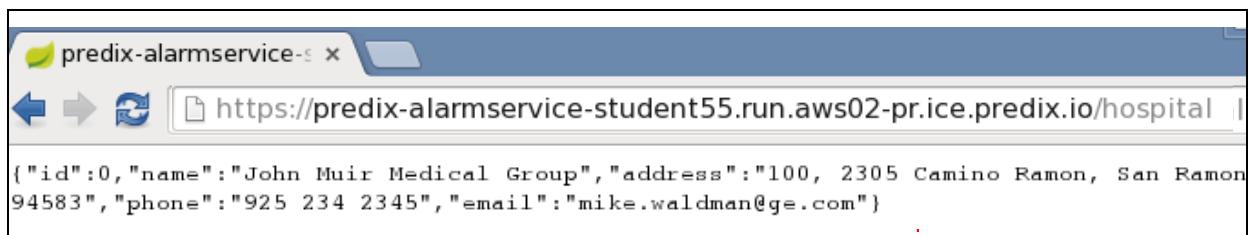
Tip - If you do not remember the URL of your alarmservice follow the steps below:

- 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/power_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/predix-seed-1.1.3/public/bower_component`
`s/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



```
temperature-card.html  temperature-template.txt  temperature-script.txt
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             <td>  
33             <tr><th class="text--right">Phone :</th><td><span>  
34             <tr><th class="text--right">Email :</th><td><span>{  
35           </table>  
36         </div>  
37       </div>  
38     </px-card>  
39   </template>  
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/power_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>
    </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.wunderground.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

```
define(['angular', 'sample-module'], function (angular, sampleModule) {  
  'use strict';  
  return sampleModule.controller('DataControlCtrl', ['$scope', function ($scope)  
  
    $scope.context = {  
      name: 'This is context',  
      // using api from weather underground: http://www.wunderground.com/  
      alarmsurl: 'https://<your alarm service url>/alarmservice',  
      hospitalurl: 'https://<your alarm service url>/hospital'  
    };  
  }]);  
});
```

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

Tip: Be sure to leave “https://” 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 Data Control interface. On the left, a vertical navigation bar has four items: Cards (selected), Interactions, Data Control (selected), and Components. The main content area is divided into two sections: HOSPITAL and ALARMS.

HOSPITAL

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

ALARMS

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 first and last 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
- 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.

The screenshot shows the Data Control interface with the following details:

- Left Navigation Pane:** Cards, Interactions, **Data Control** (selected), Components.
- HOSPITAL Section:** 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.
- ALARMS Section:** A table with columns: Alarm, Classification, Patient First Name, Patient Last Name, Patient Email.

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

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 51*](#)
- [*Fetch a UAA Token, page 54*](#)
- [*Adding a Client and Users to UAA, page 58*](#)
- [*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 l
riakcs	developer	An S3-compatible c

* 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 <plan> <my_uaa_instance> -c
'{"adminClientSecret": "<my_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 `<my_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 take effect
[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	requested state	instances
memory	disk	urls
cf-spring		started 1/1
512M	1G	cf-spring.run.aws-usw02-pr.ice.predix.io

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_SERVICES 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 --authorized_grant_types "authorization_code client_credentials refresh_token" --autoapprove openid --authorities "clients.read clients.write 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_secr
et

Successfully fetched token via client credentials grant.
Target: https://c758a891-d2d0-4d19-a2a3-450496f8557e.predix-uaa-trai
ning.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@g  
mail.com -p us3rl  
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.aaa

- 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
```

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-acs-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-acs	Beta, Enterprise*	Use this service
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	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 <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> with the **issuerID (not the uri)** of your UAA instance - this can be retrieved from the cf env output
- ◆ 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-a2a3-450496f  
8557e.predix-uaa-training.run.aws-usw02-pr.ice.predix.io/oauth/token  
"}'  
Creating service instance cf-acs in org Predix-Training / space Trai  
ning3 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 / space 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-acs-training": [
  {
    "credentials": {
      "uri": "https://predix-acs-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-acs-training.zones.6a5a8113-2b9b-4f2d-93-bae2d023dcff.user"
      }
    },
    "label": "predix-acs-training",
    "name": "cf-acs",
    "plan": "Free",
    "tags": []
  }
]
```

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

- 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:

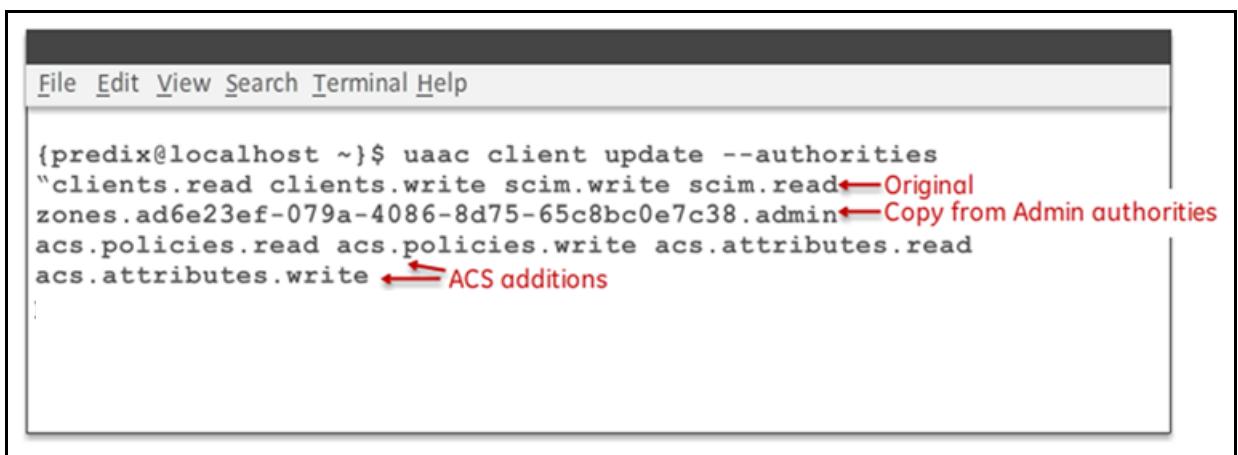
```
uaac clients
```

```
[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 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:



```
{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
```

- ◆ When prompted, enter the client name you created

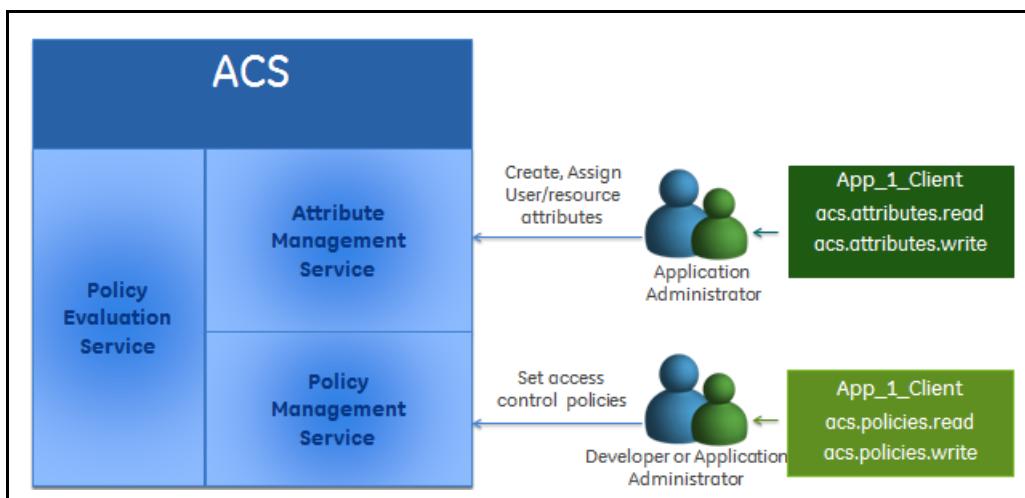
Sample results for updating the OAuth2 client:

```
[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-acss-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-acss-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 your 'application' 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>
```

– Copy/paste the oauth-scope variable from your text file

- Run the command:

```
uaac group add predix-acs-training.zones.<acs_instance_guid>.user
```

```
[predix@localhost ~]$ uaac group add predix-acs-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-acs-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.

- User 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 : *Using the Asset Service*

Learning Objectives

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

- Create a UAA service instance and bind it to an application
- Create an Asset service instance and bind it to an application
- Update the OAuth2 client with permissions to write to the Asset database

Lab Exercises

- *Create an Asset Service Instance*, page 77
- *Bind to Your Asset Service Instance*, page 80
- *Configure UAA Instance for Asset Service*, page 82
- *Fetch a Token from the UAA Service*, page 87
- *Add Locomotives to the Asset Model*, page 91
- *Add Additional Assets to the Model*, page 95
- *Link Domain Objects*, page 98
- *Delete an Asset from the Asset Model*, page 101
- *Query Assets using GEL*, page 102
- *Constructing Transitive Closure Queries*, page 107

Directions

Complete the exercises that follow.



Exercise 1: Create an Asset Service Instance

Overview

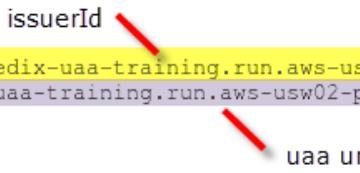
To use a Predix Service, you must first create an instance of the service in Cloud Foundry. You will create your Asset service instance using the UAA service instance you created in the previous exercise; you must have the base URLs for UAA instances that the Asset service instance will trust.

Steps

1. Retrieve environment variables for your UAA service instance.

- In a Terminal run the command
`cf env <your_Spring-Music-app>`
- Under the **VCAP_Services** section, locate your predix-uaa-training entry
 - ◆ Copy the **issuerId** variable value and paste into an empty text file
 - ◆ Copy and paste the **uri** value into the text file

```
predix-uaa-training": [  
{  
  "credentials": {  
    "issuerId": "https://644a721d-b85a-4094-bc17-3a87c759f449.predix-uaa-training.run.aws-usw02-pr.ice.  
    "uri": "https://644a721d-b85a-4094-bc17-3a87c759f449.predix-uaa-training.run.aws-usw02-pr.ice.  
    "zone": {  
      "http-header-name": "X-Identity-Zone-Id",  
      "http-header-value": "644a721d-b85a-4094-bc17-3a87c759f449"  
    }  
  },  
  "label": "predix-uaa-training",  
  "name": "student99-uaa",  your UAA instance  
  "uaa": {  
    "issuerId": "https://644a721d-b85a-4094-bc17-3a87c759f449.predix-uaa-training.run.aws-usw02-pr.ice.  
    "uri": "https://644a721d-b85a-4094-bc17-3a87c759f449.predix-uaa-training.run.aws-usw02-pr.ice.  
    "zone": {  
      "http-header-name": "X-Identity-Zone-Id",  
      "http-header-value": "644a721d-b85a-4094-bc17-3a87c759f449"  
    }  
  }  
}
```



2. Locate the Asset service in the Predix marketplace.

- In the Terminal run the command:

```
cf marketplace
```

service	plans	description
business-operations	Beta	Monetize your service
logstash-2	free	Logstash 1.4 service
logstash-5	free	Logstash 1.4 service
p-rabbitmq-35	standard	RabbitMQ is a robust
pitney-bowes-geoenhancement-service	Beta	Enhance location data
postgres	shared-nr	Reliable PostgreSQL Se
predix-acs	Tiered	Use this service to p
predix-acs-training	Basic, Free	Design precise access
predix-analytics-catalog	Bronze, Silver*, Gold*	Add analytics to the
predix-analytics-runtime	Bronze, Silver*, Gold*	Use this service to s
predix-analytics-ui	Free	Use this browser-base
predix-asset	Tiered	Create and store mach
predix-blobstore	Tiered	Use this binary large
file type.		

- Note the Asset service name and plan name



3. 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 <service> <plan> <service_instance> -c
'{"trustedIssuerIds": [<UAA_TrustedIssuerId>]}'
```

where:

- <service> the **service name (predix-asset)**
- <plan> - the **plan name (Tiered)**
- <service_instance> - **firstname-asset**
- <uaa_url> - the UAA **issuerId** copied from your text file

- A sample command will look something like this:

```
[predix@localhost ~]$ cf create-service predix-asset Tiered
student5-asset -c '{"trustedIssuerIds": ["https://229fbc8e-30
50-4392-8047-86d3fcc5a145.predix-uaa.run.usw-usw03-pr.ice.pr
edix.io/oauth/token"]}'
```

4. 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             |
|-------------------|---------------------|--------|------------------------|
| 123Asset          | predix-asset        | Tiered |                        |
| Asset_Bill        | predix-asset        | Tiered |                        |
| EngineAsset       | predix-asset        | Tiered |                        |
| Joni_acs_instance | predix-acs-training | Free   | trainingsample_DoNotDe |
| Joni_uaa_instance | predix-uaa-training | Free   | trainingsample_DoNotDe |
| MyAsset           | predix-asset        | Tiered |                        |


```

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

Exercise 2: Bind to Your Asset Service Instance

Overview

In this exercise you will bind a sample application to your Asset service instance.

Steps

- Bind the sample application to your Asset service instance.

- In the Terminal run the command:

```
cf bind-service <your-spring-music-app> <yourAssetService>
```

◆ Replace <yourAssetService> with the Asset service instance you just created

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

For example:

```
[predix@localhost env]$ cf bind-service student24-spring-music Engine-asset
Binding service Engine-asset to app student24-spring-music in org Predix-Train
OK
```



2. Verify the sample 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
10-analytics-ui	predix-analytics-ui	Free	
123postgres	postgres	shared-nr	
Engine-asset	predix-asset	Tiered	student24-spring-music
Joe postgres	postgres	shared-nr	
ams-machine-postgres	postgres	shared-nr	ams-httpdata
ams-machine-timeseries	predix-timeseries	Bronze	ams-httpdata
ams-machine-uaa	predix-uaa-training	Free	ams-httpdata
blob-student17	predix-blobstore	Tiered	object-store-student
data_postgres	postgres	shared-nr	
gbr-acs	predix-acs-training	Free	gbr-spring, gr-data

3. Retrieve environment variables for your asset instance.

- In the Terminal run the command: **cf env <your spring-music app name>**
- Under **VCAP services**, copy your Asset service instance variables to a file

```

"predix-asset": [
  {
    "credentials": {
      "instanceId": "594e9981-d52f-4690-a37b-8e8ab06acfd4",
      "scriptEngine_uri": "http://594e9981-d52f-4690-a37b-8e8ab06acfd4.predix-service.usw02-pr.ice.predix.io",
      "uri": "https://predix-asset.run.aws-usw02-pr.ice.predix.io",
      "zone": {
        "http-header-name": "Predix-Zone-Id",
        "http-header-value": "594e9981-d52f-4690-a37b-8e8ab06acfd4",
        "oauth-scope": "predix-asset.zones.594e9981-d52f-4690-a37b-8e8ab06acfd4.usw02-pr.ice.predix.io"
      }
    }
  }
]
  
```

Exercise 3: Configure UAA Instance for Asset Service

Overview

When you created your UAA service instance, an "admin" client was automatically generated for you. In this exercise, you provide your UAA client with the authority to write to the Asset database using the UAA command line interface (uaac).

Steps

1. Target the UAA instance.

- In the Terminal run the command

```
uaac target <uaa_url>
◆ <uaa_url> - paste the uaa uri value from your text file
```

2. Log into UAAC as the admin.

- In the Terminal run the command

```
uaac token client get admin -s <admin_secret>
◆ This is the password you created in the Security lab
```



3. Get current client registrations for your UAA client.

- Run the command: **uaac clients**
- From the command output, copy the authorities for your client (highlighted text)

Sample output:

```
predix@localhost~]$ uaac clients
admin
scope: uaa.none
resource_ids: none
authorized_grant_types: client_credentials
autoapprove:
action: none
authorities: clients.read clients.secret idps.write uaa.resource clients.write
    clients.admin zones.644a721d-b85a-4094-bc17-3a87c759f449.admin idps.read
    scim.write scim.read
last modified: 1459267797539
```

- Paste the authorities into an empty text file

For example:

```
clients.read, zones.644a721d-b85a-4094-bc17-3a87c759f449.
admin clients.secret idps.write uaa.resource
clients.write clients.admin idps.read scim.write scim.read
```

Analysis: When updating client authorities through uaac, you must include the client's existing authorities as well as any new authorities to be added.

4. Add ACS authorities to the existing client authorities.

- Append these ACS authorities to the end of your client's current authorities in the text file:
**acs.policies.read, acs.policies.write acs.attributes.read
clients.write acs.attributes.write**

For example:

```
clients.read
zones.644a721d-b85a-4094-bc17-3a87c759f449.admin,clients.secret
idps.write uaa.resource clients.write clients.admin idps.read
scim.write scim.read acs.policies.read acs.policies.write
acs.attributes.read acs.attributes.write
```

5. Get the predix-asset.zone for your Asset service instance.

- Retrieve environment variables for the trainingsample_DoNotDelete application
 - ◆ Under VCAP_SERVICES locate your Asset service instance
 - ◆ Copy the **oauth-scope** value (this is the ZoneId)

```
{
  "credentials": {
    "instanceId": "83af3b82-4d69-4659-8c50-4281ce5df3d8",
    "uri": "https://predix-asset.run.aws-usw02-pr.ice.predix.io",
    "zone": {
      "http-header-name": "Predix-Zone-Id",
      "http-header-value": "83af3b82-4d69-4659-8c50-4281ce5df3d8",
      "oauth-scope": "predix-asset.zones.83af3b82-4d69-4659-8c50-4281ce5df3d8.user"
    }
  },
  "label": "predix-asset",
  "name": "student30-asset",
  "plan": "Tiered",
}
```

- Append oauth-scope value to the end of your client credentials in the text file

For example:

```
clients.read,clients.write scim.write scim.read
zones.d25492ed-d8d9-4c3d-93f9-faf3bd0efbae.admin acs.policies.read
acs.policies.write acs.attributes.read,acs.attributes.write
predix-asset.zones.eab78369-7cd3-4199-a504-087b2357f044.user
```



6. Update your OAuth client with the new authorities.

- In a Terminal run the uaac command to update your new client:

```
uaac client update <clientName> --authorities "<existing  
client authorities>"
```

- ◆ Paste <existing client authorities> from the text file
- ◆ Surround the list of authorities in quotation marks ("")

Example:

```
uaac client update client2 --authorities  
"clients.read,clients.write,scim.write,scim.read,zones.d25492  
ed-d8d9-4c3d-93f9-faf3bd0efbae.admin,acs.policies.read,acs.po  
licies.write,acs.attributes.read,acs.attributes.write,predix-  
asset.zones.eab78369-7cd3-4199-a504-087b2357f044.user"
```

7. Verify new OAuth authorities for your client.

- In the Terminal run the command: **uaac clients**

- ◆ Look for the "predix-assets.zones" authority - this verifies that your client has the necessary asset authorities

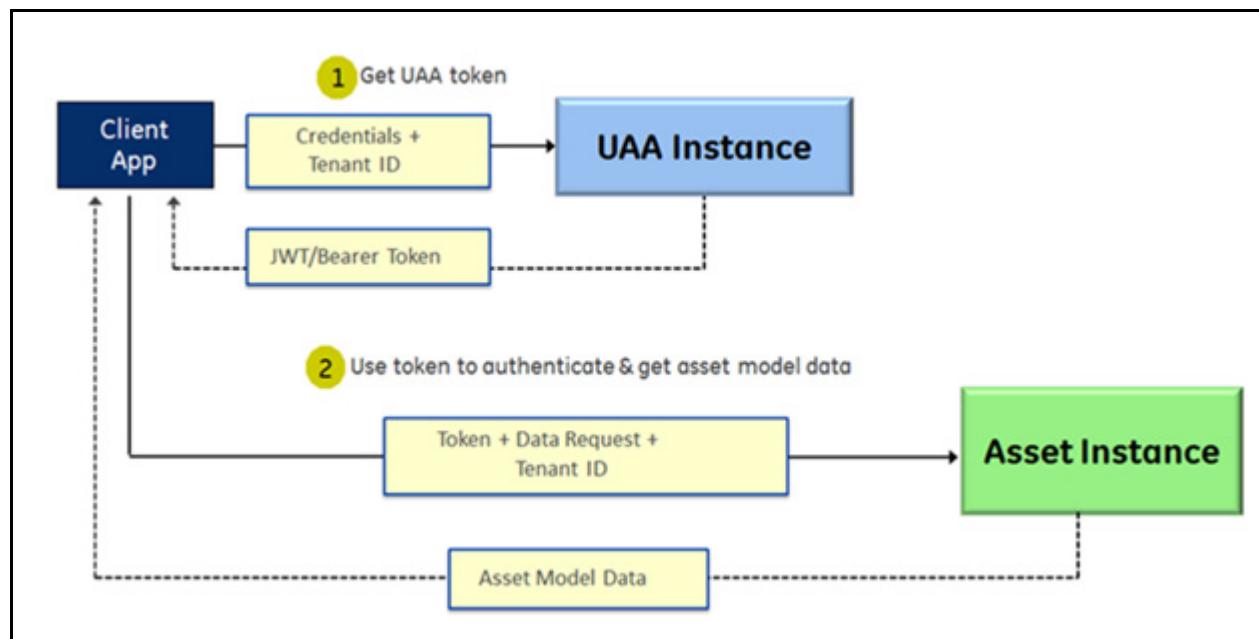
```
predix@localhost ~]$ uaac clients
admin
  scope: uaa.none
  resource_ids: none
  authorized_grant_types: client_credentials
  autoapprove:
  action: none
  authorities: clients.read clients.secret idps.write uaa.resource clients.write clients.adm
    zones.d25492ed-d8d9-4c3d-93f9-faf3bd0efbae.admin idps.read scim.write scim.read
  lastmodified: 1459267797639
student8-client
  scope: uaa.none
  resource_ids: none
  authorized_grant_types: client_credentials
  autoapprove:
  action: none
  authorities: clients.read acs.policies.read acs.policies.write acs.attributes.read
    predix-asset.zones.eab78369-7cd3-4199-a504-087b2357f044.user |clients.write acs.attribu
    zones.d25492ed-d8d9-4c3d-93f9-faf3bd0efbae.admin scim.write scim.read
  signup_redirect_url:
```



Exercise 4: 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.



Steps

1. Launch the Postman REST client and begin configuring the POST request.

- Under the **Applications** menu, select **Chromium Apps > Postman**
- Set the request Method and URL
 - ◆ Method: **POST** (select from drop-down)
 - ◆ URL: paste the UAA **issuerId** value copied from the `cf env` output

The screenshot shows the Postman interface with a POST request configuration. The method is set to POST, and the URL is https://644a721d-b85a-4094-bc17-3a87c759f449.predix-uaa-training.run.aws-usw02-pr.ice.predix.io/oauth/token. The Headers tab shows '(3)' entries, and the Body tab is selected.

2. Add username and password credentials as a header.

- Select the Authorization tab and select **Basic Authentication** (from drop-down)

The screenshot shows the Authorization tab in Postman. The Type dropdown is set to Basic Auth. The Username field contains 'client2' and the Password field contains '*****'. A 'Show Password' button is visible at the bottom right.

- ◆ Enter the user name: **your UAA client name**
- ◆ Password: **your UAA client secret**
 - Click **Ok**; a new authorization header has been added

3. Add a Content-Type header.

- In the Headers tab, under the Authorization header, enter the following
 - ◆ Enter the key: **Content-Type**
 - ◆ Enter the value: **application/x-www-form-urlencoded**
 - Click **Update**; a second header is added

4. Add a Predix Zone ID header.

- In the Headers tab, under the Content-Type header, add the following
 - ◆ Enter the key: **Predix-Zone-Id**
 - ◆ Enter the value: paste the UAA **zone Id** from your cf env output

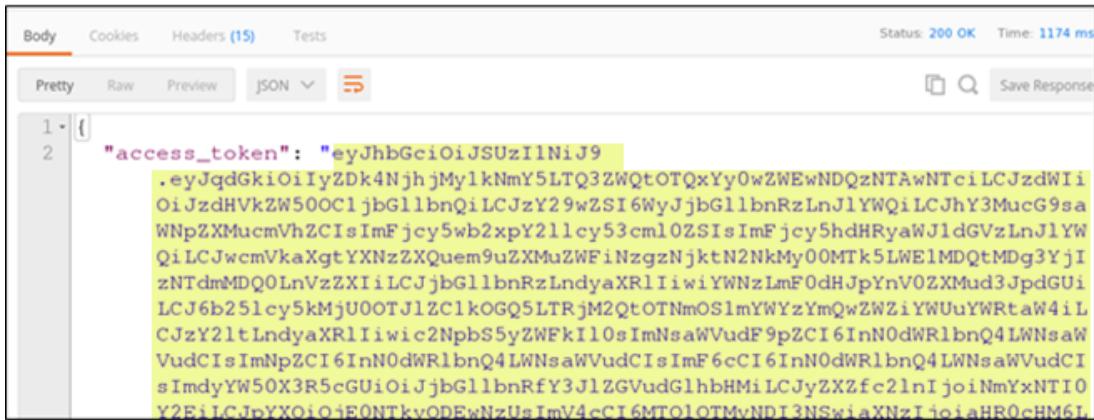
```
"predix-uaa-training": [  
  {  
    "credentials": {  
      "issuerId": "https://644a721d-b85a-4094-bc17-3a87c759f449.predix-uaa.com",  
      "uri": "https://644a721d-b85a-4094-bc17-3a87c759f449.predix-uaa.com/auth/realms/predix-uaa",  
      "zone": {  
        "http-header-name": "X-Identity-Zone-Id",  
        "http-header-value": "644a721d-b85a-4094-bc17-3a87c759f449"  
      }  
    }  
  }  
]
```

5. Construct the body of the response message.

- Select the Body tab and verify "raw" is selected
- Type the following into the body:
grant_type=client_credentials
- Click the **Send** button
- A return code **200 OK** indicates the request was successful

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

- In the Body, copy the UAA token as shown: (only copy text within the quotes, not the quotes themselves)



The screenshot shows a Postman request with the following details:

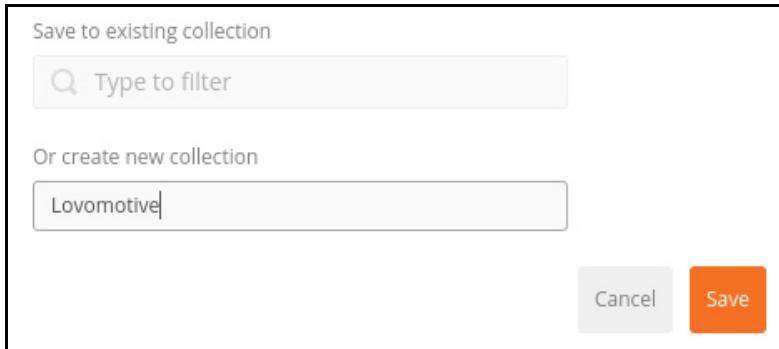
- Body**: Contains the JSON response with the key "access_token" highlighted in yellow.
- Headers (15)**: Shows various headers including Content-Type, Accept, and Authorization.
- Status: 200 OK**
- Time: 1174 ms**
- Buttons**: Pretty, Raw, Preview, JSON, Save Response.

```

1  "access_token": "eyJhbGciOiJSUzI1NiJ9
2   .eyJqdGkiOiIyZDk4NjhjMy1kNmY5LTQ3ZWQtOTQxYy0wZWEwNDQzNTAwNTciLCJzdWIi
  OiJzdHVkZW50OC1jbG11bnQiLCJzY29wZSI6WyJjbG11bnRzLnJlYWQiLCJhY3MucG9sa
  WNpZXMuemVhZC1sImFjcy5wb2xpY21lcyc53cm10ZS1sImFjcy5hdHRyaWJldGVzLnJlYW
  QiLCJwcmVkaXgtYXNzZXQuem9uZXMuZWFiNzgzNjktN2NkMy00MTk5LWE1MDQtMDg3YjI
  zNTdmMDQ0LnVzZXIIiLCJjbG11bnRzLnDyaXR1IiwiYWNzLmF0dHJpYnV0ZXMu3JpdGUI
  LCJ6b251cy5kMjU0OTJ1ZC1kOGQ5LTRjM2QtOTNmOSImYWyzYmQwZWZiYWUuYWRtaW4iL
  CJzY21tLnDyaXR1IiwiC2NpbSS5yZWFkI10sImNsawVvudF9pZCI6InN0dWR1bnQ4LWNsaW
  VudC1sImNpZCI6InN0dWR1bnQ4LWNsaWVvudC1sImF6cCI6InN0dWR1bnQ4LWNsaWVvudC1
  sImdyYW50X3R5cGUioiJjbG11bnRfY3J1ZGVudGlhbHMiLCJyZXZfc2lnIjoimNmYxNTI0
  Y2EiLCJpYXQiOjE0NTkvODEwNzUgImV4cCI6MT0LOMvNDI3NSwiaXNzIjoiaHR0cHM6I"

```

- You will use this token in the next lab to retrieve assets from the asset model
- Save** your request as **GetToken**; when prompted for a collection name, scroll down and in the **Or create a new collection** field, type **Locomotive**



Exercise 5: Add Locomotives to the Asset Model

Overview

In this exercise, you use the Asset Service APIs to add locomotive asset model data. Our asset model that includes five REST collections: locomotive, fleet, engine, manufacturer, customer and aircraft.

Note: All JSON files for the lab can be found in the [PredixApps/training_labs/lab_files/Asset JSON Lab Files.zip](#) file in the DevBox.

Steps

1. Configure a POST request to add locomotiveengine assets.

- Open a new tab in Postman and configure the POST request
 - ◆ Method: POST
 - ◆ Request URL: `http://<asset_uri>/locomotiveengine`
 - `<asset_uri>` is retrieved from your `cf env` output
 - `/locomotiveengine` is an endpoint

```
predix-asset": [  
  {  
    "credentials": {  
      "instanceId": "1b4e7ae2-b002-4a45-b9df-78e9c670b6c4", /  
      "uri": "https://predix-asset.run.aws-usw02-pr.ice.predix.io",  
      "zone": {  
        "http-header-name": "Predix-Zone-Id",           asset zone id  
        "http-header-value": "1b4e7ae2-b002-4a45-b9df-78e9c670b6c4" /  
      }  
    }  
  }  
]
```

2. Add an Authorization header to the request.

- In the Headers tab, begin typing the header name in the first available field
 - ◆ Name: **Authorization**
 - ◆ Value: **Bearer** <paste UAA token from GetToken response>
- **NOTE:** "Bearer" must begin with a capital letter. Add a space then paste in the token.

3. Add additional required headers to your request.

- Add a Content-Type header
 - ◆ Name: **Content-Type**
 - ◆ Value: **application/json**
- Add a Predix-Zone-Id header
 - ◆ Name: Predix-Zone-Id
 - ◆ Value: <paste asset Zone Id>

Your request with headers should look as follows:

The screenshot shows the Postman interface with a POST request to `https://predix-asset.run.aws-usw02-pr.ice.predix.io/locomotiveengine`. The 'Headers (3)' tab is selected. Three headers are listed: Authorization, Content-Type, and Predix-Zone-Id. The 'Predix-Zone-Id' value, which is `78338129-1bba-46a6-a759-f32869abdf7d`, is highlighted with a red arrow pointing to it, and the text 'Asset Zone Id' is written next to the arrow.

Header	Value
Authorization	Bearer eyJhbGciOiJSUzI1NiJ9.eyJqdGkiOiI1YTY3YzRkYSC
Content-Type	application/json
Predix-Zone-Id	78338129-1bba-46a6-a759-f32869abdf7d

4. Configure the request body.

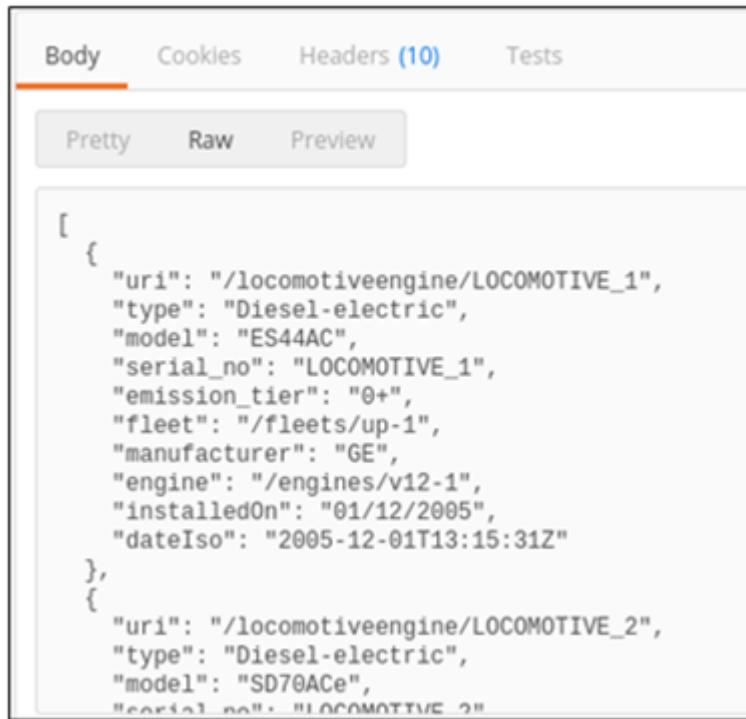
- Select the Body tab and select the **raw** radio button
- Copy/paste content of the **Locomotiveengine.json** file into the body (partial file shown here)

```
[  
 {  
   "uri": "/locomotiveengine/LOCOMOTIVE_1",  
   "type": "Diesel-electric",  
   "model": "ES44AC",  
   "serial_no": "LOCOMOTIVE_1",  
   "emission_tier": "0+",  
   "fleet": "/fleets/up-1",  
   "manufacturer": "GE",  
   "engine": "/engines/v12-1",  
   "installedOn": "01/12/2005",  
   "dateIso": "2005-12-01T13:15:31Z"  
 },  
 {  
   "uri": "/locomotiveengine/LOCOMOTIVE_2",  
   "type": "Diesel-electric",  
   "model": "SD70ACe",  
   "serial_no": "LOCOMOTIVE_2",  
   "emission_tier": "0+",  
   "fleet": "/fleets/up-1",  
   "manufacturer": "/manufacturers/electro-motive-diesel",  
   "engine": "/engines/v16-2-1"  
 }]
```

- Verify the JSON has a both an open "[" and a closing "]" brace in the file
- Click **Send**
 - ◆ A "204 No Content" status is returned indicating the post was successful
- **Save** your request as **PostAssets** in the Locomotive collection

5. Create a **GET** request to return locomotiveengine objects.

- In the current request, change the method to **GET**
- **Send** the request; verify a status code of **200 OK** is returned in the *Header Request*
- Select one of the *Body* tabs to view locomotive assets returned



```
[  
  {  
    "uri": "/locomotiveengine/LOCOMOTIVE_1",  
    "type": "Diesel-electric",  
    "model": "ES44AC",  
    "serial_no": "LOCOMOTIVE_1",  
    "emission_tier": "0+",  
    "fleet": "/fleets/up-1",  
    "manufacturer": "GE",  
    "engine": "/engines/v12-1",  
    "installedOn": "01/12/2005",  
    "dateIso": "2005-12-01T13:15:31Z"  
  },  
  {  
    "uri": "/locomotiveengine/LOCOMOTIVE_2",  
    "type": "Diesel-electric",  
    "model": "SD70ACE",  
    "serial_no": "LOCOMOTIVE_2"  
  }]
```

Exercise 6: Add Additional Assets to the Model

Overview

in this exercise you submit additional POST requests to add engine, manufacturer, customer, fleet and aircraft collections to your asset model. These requests will have the same headers as the POST request used to add locomotive assets, but you will use different JSON for the body.

Steps

1. POST Request 1: Add Fleet assets.

- Select a new tab in Postman and begin constructing your request
 - ◆ Method: POST
 - ◆ URL: `http://<asset_uri>/fleet`
- Add the following headers to the request
 - ◆ Authorization, Content-Type, Predix-Zone-Id
- Add the Body of the request
 - ◆ Copy/paste JSON from the **fleet.json** file

The screenshot shows the Postman interface with a POST request setup. The URL is `https://predix-asset.run.aws-usw02-pr.ice.predix.io/fleet`. The Headers tab is active, containing three entries:

Header	Value
Authorization	Bearer eyJhbGciOiJSUzI1NiJ9.eyJqdGkiOiI1...
Content-Type	application/json
Predix-Zone-Id	1b4e7ae2-b002-4a45-b9df-78e9c670b6c4

- Send the request and verify a status code of "204 No Content" was returned

2. POST Request 2: Add Engine assets.

- Use the current POST request and make the following changes
 - ◆ Change the URL endpoint to **engine**
URL: http:// <asset_uri>/engine
 - ◆ Request Body - replace the JSON with contents from the **engine.json** file
- Send the request and verify a status code of "204 No Content" was returned

3. POST Request 3: Add Manufacturer assets.

- Use the current POST request and make the following changes
 - ◆ Change the URL endpoint to **manufacturer**
URL: http:// <asset_uri>/manufacturer
 - ◆ Request Body - replace the JSON with contents from the **manufacturer.json** file
- Send the request and verify a status code of "204 No Content" was returned

4. POST Request 4: Add Customer assets.

- Use the current POST request and make the following changes
 - ◆ Change the URL endpoint to **customer**
URL: http:// <asset_uri>/customer
 - ◆ Request Body - replace the JSON with contents from the **customer.json** file
- Send the request and verify a status code of "204 No Content" was returned

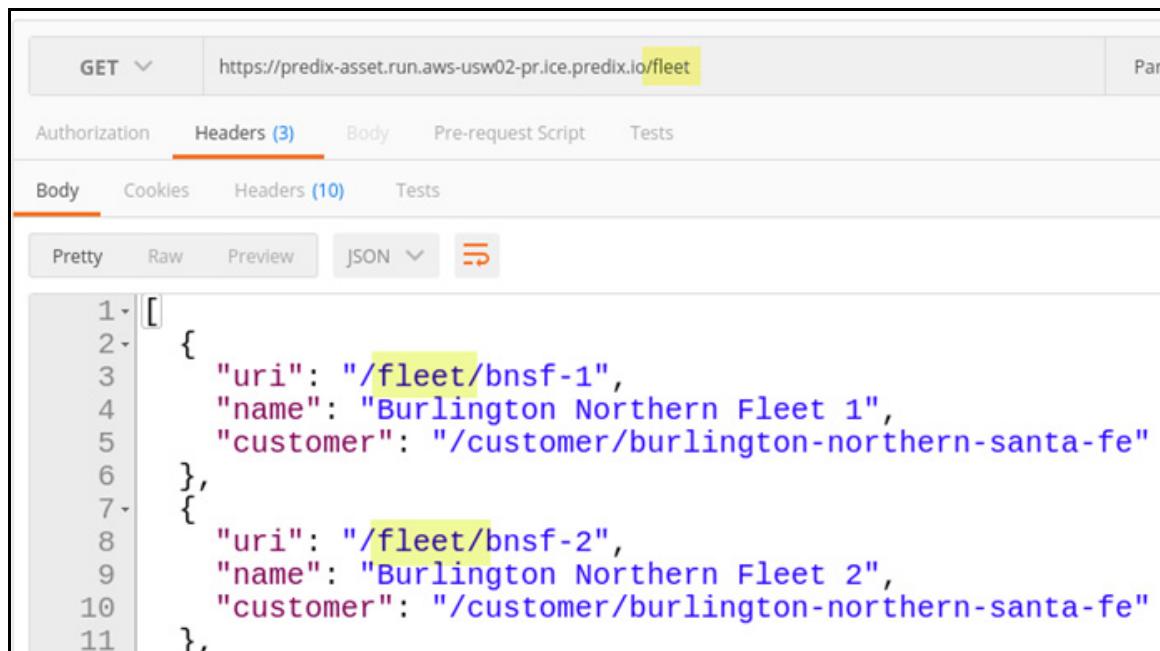


5. POST Request 5: Add Aircraft assets.

- Use the current POST request and make the following changes
 - ◆ Change the URL endpoint to **asset**
URL: http:// <asset_uri>/asset
 - ◆ Request Body - replace the JSON with contents from the **aircraft.json** file
- Send the request and verify a status code of "204 No Content" was returned

6. Verify new asset collections were added to your model.

- Change your POST request to use the **GET** method
- For each request, change the **URI endpoint** to reference a different collection name (e.g. /engine, /fleet, /manufacturer, etc.)
- Verify the correct asset types are returned for each collection



The screenshot shows a Postman request configuration. The method is set to **GET**, and the URL is **https://predix-asset.run.aws-usw02-pr.ice.predix.io/fleet**. The **Headers (3)** tab is selected, and the **Body** tab is also selected, showing a JSON payload:

```
1. [
2.   {
3.     "uri": "/fleet/bnsf-1",
4.     "name": "Burlington Northern Fleet 1",
5.     "customer": "/customer/burlington-northern-santa-fe"
6.   },
7.   {
8.     "uri": "/fleet/bnsf-2",
9.     "name": "Burlington Northern Fleet 2",
10.    "customer": "/customer/burlington-northern-santa-fe"
11.  }
```

Exercise 7: Link Domain Objects

Objective

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

Steps

1. Link a locomotive asset to a different manufacturer.

- Submit a **GET** request to retrieve all locomotive assets in your model
 - ◆ Copy the JSON for just one locomotive asset

Example:

```
[  
  {  
    "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  
    }  
  }]
```

- Update the request with the following changes
 - ◆ Change the request method to **PUT**
 - ◆ Append your locomotive uri to the Asset URL (ensures only that asset is updated)
For example: `https://<asset_uri>/locomotive/10`
 - ◆ In the Body of the request, paste the JSON for your locomotive asset
 - ◆ Add a closing brace "]" to the end of the JSON



- Link the locomotive asset to the Cummins manufacturer
 - ◆ Change the manufacturer property to point to **cummins**
- **Note:** The PUT operation requires you to specify all attributes for an asset

The screenshot shows a Postman interface with a PUT request to the URL `https://predix-asset.run.aws-usw02-pr.ice.predix.io/locomotive/10`. The 'Body' tab is selected, displaying a JSON object with the following content:

```
1 [  
2 {  
3   "uri": "/locomotive/10",  
4   "type": "Diesel-electric",  
5   "model": "SD70ACe",  
6   "serial_no": "0010",  
7   "emission_tier": "0+",  
8   "fleet": "/fleet/up-4",  
9   "manufacturer": "/manufacturer/cummins",  
10  "engine": "/engine/v16-2-5",  
11  "hqLatLng": {  
12    "lat": 47.941049,  
13    "lng": -100.126484  
14  }  
15}  
16]
```

- **SEND** the request
- Verify a status code **204 No Content** is returned

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

- Change the request method to **GET**
- **SEND** the request
- In the *Response*, verify the manufacturer property now references cummins:

```
[  
 {  
   "uri": "/locomotive/10",  
   "type": "Diesel-electric",  
   "model": "SD70ACe",  
   "serial_no": "0010",  
   "emission_tier": "0+",  
   "fleet": "/fleet/up-4",  
   "manufacturer": "/manufacturer/cummins",  
   "engine": "/engine/v16-2-5",  
   "hqLatLng": {  
     "lat": 47.941049,  
     "lng": -100.126484  
   }  
 }  
 ]
```



Exercise 8: 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 updated 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 9: Query Assets using GEL

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 /locomotiveengine domain object
- Append the following fields clause to the end of the URL
?fields=uri,model,manufacturer

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

- **SEND** the request



- Select a Body tab to view the results

```
[  
  {  
    "uri": "/locomotive/10",  
    "model": "SD70ACe",  
    "manufacturer": "/manufacturer/cummins"  
  },  
  {  
    "uri": "/locomotive/11",  
    "model": "ES44AC",  
    "manufacturer": "/manufacturer/GE1"  
  },  
,
```

2. 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

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

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

3. 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

`ps://predix-asset.run.aws-usw02-pr.ice.predix.io/locomotiveengine?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

4. 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)

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

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



5. Construct a forward-relate query to retrieve.

- Change the URL to reference the **engine** domain object
https://<asset_uri>/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

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

- Change the URL to reference the **fleet** domain object
`https://<asset_uri>/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 10: 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** endpoint
`https://<asset_uri>/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"  
},  
}
```

- In the JSON, notice that the **parent** attribute is used to link "child" asset objects to their "parent" asset objects. For example, the **aircraft** object has no parent attribute, which distinguishes it as the top-most object in the hierarchy. The **aircraftEngine** object has a **parent** attribute that points to the /asset/**aircraft** object. This indicates that the **aircraftEngine** is a child to **aircraft**, and so on.



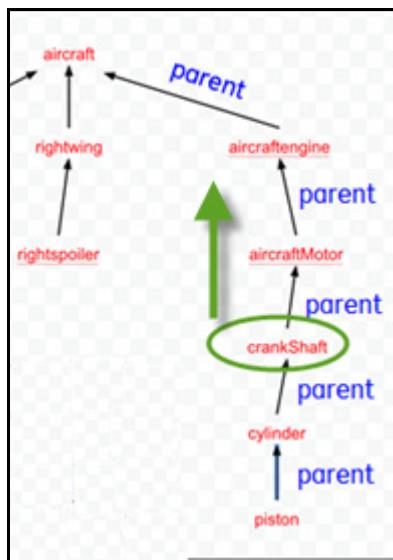
2. Execute a forward-relate transitive closure to retrieve parent assets of crankShaft.

- Append a filter clause (bolded here) to the GET request URL

`https://<asset_uri>/asset?filter=name=crankShaft>parent[t3]`

Note: The `parent` attribute is required in the transitive query; you "traverse" through the levels in the hierarchy along this attribute

- 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://<asset_uri>/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 6: Use Case - Locomotive Client Application

Our customer is a transportation company that owns fleets of locomotives. Sensors on the locomotives collect real-time data such as speed, location, weather conditions, engine temperature, etc. The customer wants to capture data for their locomotives to track fuel consumption so they can improve fuel efficiency and lower costs.

This solution will be a two-phased approach.

Phase I

In Phase I, you build microservices that implement Predix series in a client application to capture and display the following data for two locomotives:

- RPM
- Torque
- Location (latitude and longitude)

Your solution will use the following Predix Services

- **Asset Data Service** to define locomotive assets
- **Time Series Service** to ingest (simulated) and store sensor data
- **User Account and Authentication Service** to authenticate your clients
- **Views** service to display time series data in a dashboard

In our classroom environment, we will use a microservice to generate data rather than connecting to actual edge devices. The data will then be collected and processed using Predix normally.

Phase II (outside scope of this class)

In Phase II of the project, you will create analytics based on this data to make recommendations to the customer on how best to lower the fuel consumption of their fleets.

Locomotive Client Application Overview

The Locomotive client application consists of the following microservices:

locomotive-simulator-service

This microservice will randomly generate 3 datasets and call a locomotive-dataingestion-service which in turn, will push the simulator data into a time series database every 2 seconds.

The dataset consists of:

- RPM
- Torque
- Location (Latitude, Longitude)

locomotive-dataingestion-service

This microservice will ingest data from the locomotive-simulator-service and push the data to the time series database.

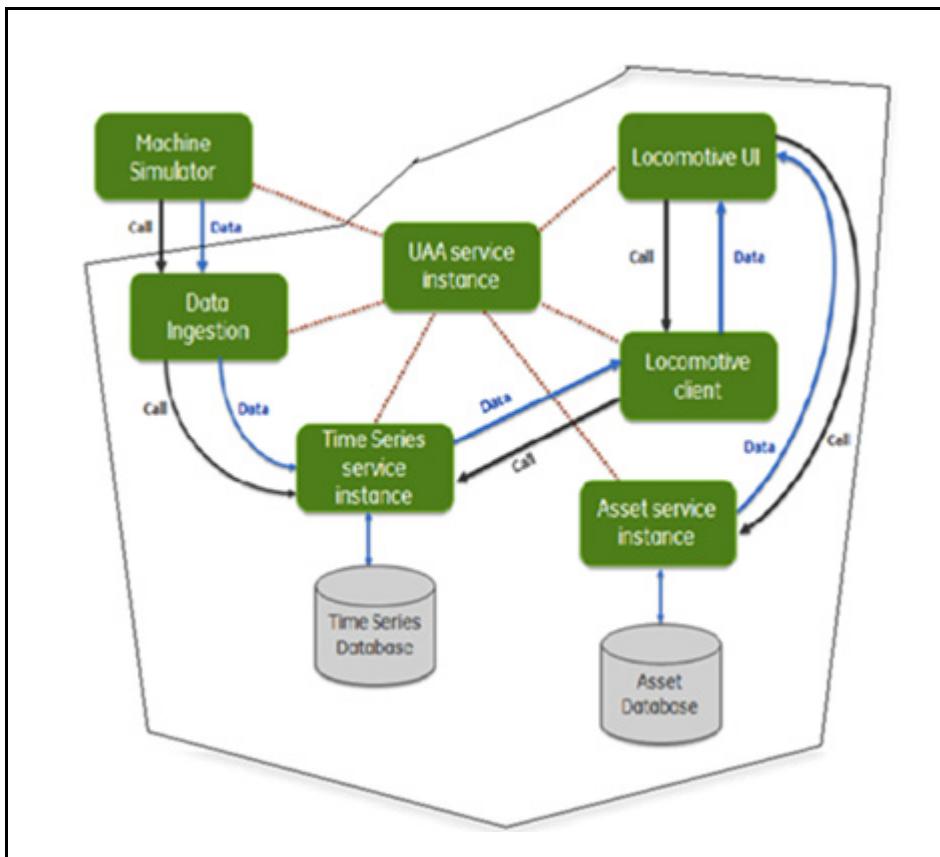
The service has exposed an endpoint /SaveTimeSeriesData which will accept an authorization token in the header and ZoneId, clientId and content json as the request parameter and will inject the content json - (rpm, torque and the location data) into the TimeSeries database.

locomotive-client-service

This microservice exposes the following endpoints:

- **/locomotive/tags** -This endpoint will fetch the tags under which the data is stored in the Time Series database in the UI.
- **/locomotive/datapoints** -This end point will fetch the latest 100 data points for the LOCOMOTIVE_1 rpm, torque and location.

Data and Control Workflow



Exercise 1: Build the Data Ingestion Application

Overview

In this exercise you build a data ingestion application that calls a time series service instance to push data to a time series database.

Steps

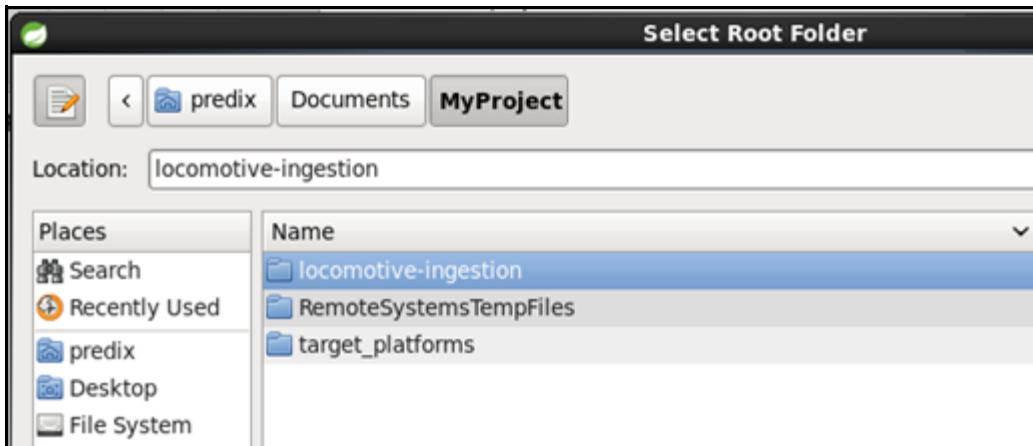
1. Download a Maven project from github.

- In a Terminal, change directories to predix/Documents/MyProject
- Run the command:

```
git clone https://github.com/Training-Predix/  
locomotive-ingestion.git
```

2. Import the locomotive-ingestion project into Eclipse.

- Open Eclipse and right-click in Package Explorer
- Select **Import > Maven > Existing Maven Projects**
- Browse to /predix/Documents/MyProject and select **locomotive-ingestion**



- ◆ Accept all other defaults and click **Next**
- ◆ Click **Finish**; the new project is displayed in Package Explorer

3. Update the Manifest file.

- In Eclipse, open **manifest.yml** file and make the following change:
 - ◆ application name: replace the placeholder **<yourname>** with your name

```

1 ---
2 applications:
3   #- name: student8-locomotive-dataingestion-service
4   - name: <yourname>-locomotive-dataingestion-service
5     buildpack: java_buildpack
6     path: target/data-ingestion-0.0.1-SNAPSHOT.jar

```

- Save and close the file

4. Build the project.

- Right-click on the project name and select **Run as > Maven Install**
- You should see a Build Success message in the Console view

5. Deploy the project to Cloud Foundry.

- In a Terminal change directory to /predix/Documents/MyProject/locomotive-ingestion
- Run the command: **cf push**
- Verify the push is successful and the service starts

```
Showing health and status for app student8-locomotive-dataingestion-service in org Predix-Training /  
space Training5 as student10...  
OK  
  
requested state: started  
instances: 1/1  
usage: 1G x 1 instances  
urls: student8-locomotive-dataingestion-service.run.aws-usw02-pr.ice.predix.io  
last uploaded: Tue Mar 29 22:39:50 UTC 2016  
stack: cflinuxfs2  
buildpack: java_buildpack  
  
#0  state      since            cpu    memory          disk       details  
#0  running    2016-03-29 03:40:32 PM  0.0%  571.7M of 1G  170.6M of 1G
```

Now you have an application that you can bind your Asset and UAA services to.

6. Bind the locomotive-dataingestion-service to your UAA and Asset instances.

- For each of your Asset and UAA service instances run the command:

```
cf bind-service <your locomotive dataingestion app> <your service instance>
```

7. Create a time series service instance.

- In the Terminal, run **cf m** command to see a list of services offered in the marketplace
 - Note the service and plan name for the Time Series service:

service	plans	description
<code>business-operations</code>	beta	Monetize your service using subscriber segmentation.
<code>logstash-5</code>	free	Logstash 1.4 service for applications
<code>p-rabbitmq-35</code>	standard	RabbitMQ is a robust and scalable
<code>predix-acs-training</code>	Basic, Free	Design precise access controls and
<code>predix-analytics-catalog</code>	Bronze, Silver*, Gold*	Add analytics to the Predix cloud
<code>predix-analytics-runtime</code>	Bronze, Silver*, Gold*	Use this service to support elastic
<code>predix-analytics-ui</code>	Free	Use this browser-based user interface
<code>predix-asset</code>	Tiered	Create and store machine asset mod
<code>predix-blochstore</code>	Tiered	Use this binary large object storage
<code>predix-timeseries</code>	Bronze, Silver*, Gold*	Quickly and efficiently manage, inser
<code>predix-tms</code>	Tiered	Use this service to provision serv
.		
<code>predix-uaa</code>	Tiered	Use this service for a full-featured

- Run the command to create a TimeSeries service instance

```
cf create-service predix-timeseries Bronze <your instance name> -c '{"trustedIssuerIds": ["<uaa_trusted_issuer_id> "]}'
```

where `<uaa_trusted_issuer_id>` is retrieved from the `cf env <appName>` output

Example:

```
cf create-service predix-timeseries Bronze student8-timeseries -c '{"trustedIssuerIds": ["https://d25492ed-d8d9-4c3d-93f9-faf3bd0efbae.predix-uaa-training.run.aws-usw02-pr.ice.predix.io/oauth/token"]}'
```

- Run the `cf s` command to verify your timeseries service instance was created successfully

<code>student8-asset</code>	<code>predix-asset</code>	Tiered
<code>student8-timeseries</code>	<code>predix-timeseries</code>	Bronze
<code>student8-uaa</code>	<code>predix-uaa-training</code>	Free

8. Bind the time series service instance to the locomotive-ingestion-service application.

- Run the command:

```
cf bind-service <your locomotive-dataingestion app> <your  
timeseries instance>
```

Example:

```
cf bind-service student8-locomotive-dataingestion-service  
student8-timeseries
```

9. Copy your environment variables to an output file.

- Run the command:

```
cf env <locomotive_ingestion application name> >>filename
```

TIP: Run the `cf a` command to display your locomotive_ingestion application name (you can find it in the manifest.yml file too).

This command copies the env variables to a file for future reference (asset instanceID, instance_name and timeseries zone header value and name, and UAA URI)

- Open the output file with a text editor for use in the next steps

10. Update the OAuth client with permissions to access time series zones.

- Create a text file to store the command and parameters used to update your uaa client
- Add the command to update our uaa client to the text file


```
uaac client update <your client>
```
- Append the following lines to this command:


```
--authorities "<paste existing authorities>
timeseries.zones.<your timeseries zoneId>.user
timeseries.zones.<your timeseries zoneId>.ingest
timeseries.zones.<your timeseries zoneId>.query"
```
- Update the <placeholders> with values from the text file created in the previous step
 - ◆ Replace <your client> with your uaa clientId
 - ◆ Replace <original authorities> with your client authorities; you can get this info by running the uaac clients command
 - ◆ Replace <your timeseries zoneId> with the zoneId of your timeseries service
- Copy the contents of the file and paste them into a Terminal to run the command

- Here is the command executed in the Terminal:

```
[predix@localhost ~]$ uaac client update student8-client --authorities "uaa.resource, predix-asset.zones.199-a504-087b2357f044.user, timeseries.zones.c27d621c-e744-4fa3-b735-2d726bbf57b3.query, timeseries.zone.4fa3-b735-2d726bbf57b3.user, timeseries.zones.c27d621c-e744-4fa3-b735-2d726bbf57b3.ingest, clients.write, scim.read, scim.write"
scope: uaa.none
client_id: student8-client
resource_ids: none
authorized_grant_types: client_credentials
autoapprove:
action: none
authorities: clients.read timeseries.zones.c27d621c-e744-4fa3-b735-2d726bbf57b3.query
  timeseries.zones.c27d621c-e744-4fa3-b735-2d726bbf57b3.user uaa.resource
  timeseries.zones.c27d621c-e744-4fa3-b735-2d726bbf57b3.ingest
  predix-asset.zones.eab78369-7cd3-4199-a504-087b2357f044.user clients.write scim.read
signup_redirect_url:
lastmodified: 1459296525243
```



11. Create groups for timeseries.

- Run the following commands separately:

```
uaac group add timeseries.zones.<your timeseries zone ID>.query  
uaac group add timeseries.zones.<your timeseries zone ID>.ingest  
uaac group add timeseries.zones.<your timeseries zone ID>.user
```

12. Add user to all the groups for time series.

- Run the following commands: (replace <username> with the UAA user you created in the Security lab)

```
uaac member add timeseries.zones.<your timeseries zone ID>.query  
<username>  
uaac member add timeseries.zones.<your timeseries zone ID>.ingest  
<username>  
uaac member add timeseries.zones.<your timeseries zone ID>.user  
<username>
```

13. Update dataingestion manifest file with UAA, Asset and Timeseries service instances.

- In Eclipse, open the **manifest.yml** file and make the following changes:
 - Uncomment all the lines under the **services:** section
 - Replace the <instance Name> placeholders with the names of the UAA, Asset and Timeseries service instances you created in earlier labs.

```
---  
applications:  
  - name: student8-locomotive-dataingestion-service  
    buildpack: java_buildpack  
    path: target/data-ingestion-0.0.1-SNAPSHOT.jar  
    services:  
      - student8-asset  
      - student8-timeseries  
      - student8-uaa
```

- In the **env:** section make the following changes
 - ◆ For each uncommented line, replace every <placeholder> with the information requested (e.g. service instance name, zoneId, clientId, etc.)
- CHECK YOUR WORK! Verify all placeholders have been filled in

```
env:
SPRING_PROFILES_ACTIVE : cloud,clouddev
#predix_asset_name: asha-locomotive-asset
predix_asset_name: student8-asset
#predix_timeseries_name: asha-locomotive-timeseries
predix_timeseries_name: student8-timeseries
#predix_oauthRestHost: 2e93f1db-9f2c-43e9-9fa9-7ba816c8f01c.predix-uaa-training.run.aws-usw02-pr.ice
predix_oauthRestHost: d25492ed-d8d9-4c3d-93f9-faf3bd0efbae.predix-uaa-training.run.aws-usw02-pr.ice.
#predix_oauthClientId: asha-client:clientsecret
predix_oauthClientId: student8-client:clientsecret
#trustedIssuerIdsRegexPattern: ^https://(.*)?2e93f1db-9f2c-43e9-9fa9-7ba816c8f01c.predix-uaa-trai
trustedIssuerIdsRegexPattern: ^https://(.*)?d25492ed-d8d9-4c3d-93f9-faf3bd0efbae.predix-uaa-train
accessTokenEndpointUrl : https://2e93f1db-9f2c-43e9-9fa9-7ba816c8f01c.predix-uaa-training.run.aws-us
accessTokenEndpointUrl : https://d25492ed-d8d9-4c3d-93f9-faf3bd0efbae.predix-uaa-training.run.aws-us
#clientId : asha-client
clientId : student8-client
#clientSecret: clientsecret
clientSecret: clientsecret
#predixWebSocketURI: wss://websocket-server-new-release.run.aws-usw02-pr.ice.predix.io/livestream/me
```

- Save and close your file

14. Update configuration files to include your timeseries and asset environment variables.

- In Eclipse, expand the `src/main/resources` folder and open the **application-cloud.properties** file
- Replace values for these parameters with values in bold:

```
predix.asset.zoneid = asset zone id value
predix.timeseries.zoneid = timeseries zone id value
predix.timeseries.override.oauthClientId = UAA client name
predix.timeseries.override.oauthRestHost = UAA zoneId
predix.timeseries.ingestionPassword = UAA client secret
predix.timeseries.ingestionUsername = UAA client name
```



```
#properties when in cloud foundry profile
logging.level.root=INFO
logging.level.org.springframework=ERROR

predix.oauth.certLocation=
#predix.asset.zoneid=160fbeaf-1e6e-4499-922c-f ea0525e548e
predix.asset.zoneid=<asset zone id value>
#predix.timeseries.zoneid=1c84e446-aef6-44e9-9c38-f3f3d0b9ff7b
predix.timeseries.zoneid=<time series zone id value>

#predix.timeseries.queryUri=https://time-series-store-predix.run.aws-usw02-pr.ice.predix.io/api/v1/datapoints
predix.timeseries.queryUri=https://time-series-store-predix.run.aws-usw02-pr.ice.predix.io/v1/datapoints/quer
predix.timeseries.override.oauthOverride=false

#predix.timeseries.override.oauthClientId=asha-client
predix.timeseries.override.oauthClientId=<UAA client name>

#predix.timeseries.override.oauthRestHost=2e93f1db-9f2c-43e9-9fa9-7ba816c8f01c.predix-uaa-training.run.aws-us
predix.timeseries.override.oauthRestHost=<UAA URI>.predix-uaa-training.run.aws-usw02-pr.ice.predix.io

predix.timeseries.ingestUri=wss://gateway-predix-data-services.run.aws-usw02-pr.ice.predix.io/v1/stream/messa

#predix.timeseries.ingestionPassword=asha-client
#predix.timeseries.ingestionUsername=clientsecret
predix.timeseries.ingestionUsername=<UAA client name>
predix.timeseries.ingestionPassword=<UAA client secret>
```

TIP: copy/paste these values from your output file

- Update the parameters in the following files:
 - ◆ /src/main/resources/**application-cloudev.properties**
 - ◆ config/**application.properties**

```

1 #properties when in cloud foundry profile
2 logging.level.root=INFO
3 logging.level.org.springframework=ERROR
4
5 predix.oauth.certLocation=
6 #predix.asset.zoneid=160fbeaf-1e6e-4499-922c-f ea0525e548e
7 predix.asset.zoneid=<yourAssetZoneID>
8 #predix.timeseries.zoneid=1c84e446-aef6-44e9-9c38-f3f3d0b9ff7b
9 predix.timeseries.zoneid=<yourTimeSeriesZoneID>
0
1 #predix.timeseries.queryUri=https://time-series-store-predix.run.aws-usw02-pr.ice.predix.io/api/v1/datapoi
2 predix.timeseries.queryUri=<https://time-series-store-predix.run.aws-usw02-pr.ice.predix.io/v1/datapoints/>
3 predix.timeseries.override.oauthOverride=false
4
5 #predix.timeseries.override.oauthClientId=asha-client
6 predix.timeseries.override.oauthClientId=<yourclientname>
7
8 #predix.timeseries.override.oauthRestHost=2e93f1db-9f2c-43e9-9fa9-7ba816c8f01c.predix-uaa-training.run.aws
9 predix.timeseries.override.oauthRestHost=<yourUAAZoneID>.predix-uaa-training.run.aws-usw02-pr.ice.predix.i
0
1 predix.timeseries.ingestUri=wss://gateway-predix-data-services.run.aws-usw02-pr.ice.predix.io/v1/stream/me
2
3
4 #predix.timeseries.ingestionPassword=asha-client
5 #predix.timeseries.ingestionUsername=clientsecret
6 predix.timeseries.ingestionUsername=<yourclientusername>
7 predix.timeseries.ingestionPassword=<yourclientsecret>

```

NOTE: Examples of configured parameters are provided in the file.

- Save and close the files

15. Push the locomotive-ingestion application to the cloud.

- Right-click on the **locomotive-ingestion** project and select **Run As > Maven Install**
 - ◆ In the Console, verify the BUILD SUCCESS message is displayed
- In a Terminal, change to the predix/Documents/MyProject/locomotive-ingestion directory
- Run the command: **cf push**
 - ◆ Verify the push completed successfully and the application was started



Exercise 2: Build Your Own Machine Simulator

Overview

Build your own machine simulator which generates data and calls the data ingestion service. The data ingestion service pushes that data to the time series database.

Steps

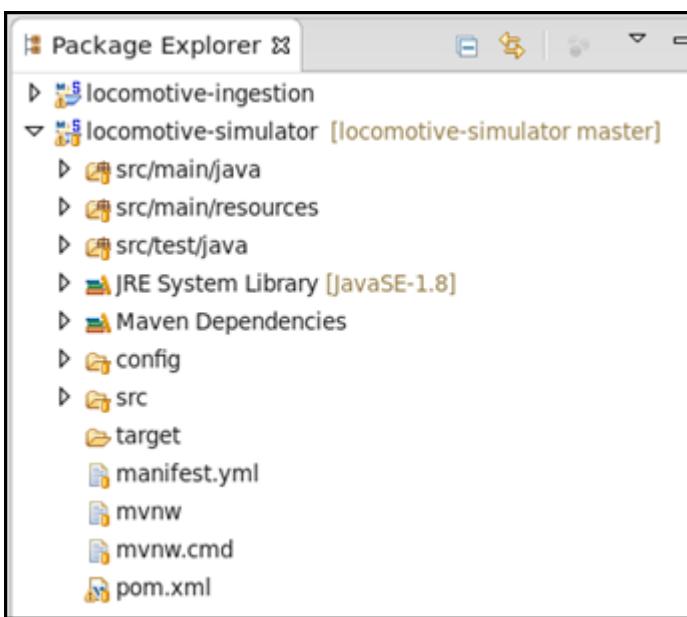
1. Download the simulator service from github.

- In the Terminal cd /predix/Documents/MyProject
- Run the command:

```
git clone https://github.com/Training-Predix/  
locomotive-simulator.git
```

2. Import the Simulator Service into Eclipse.

- **File > Import > Existing Maven Projects**
 - ◆ Root Directory: locomotive-simulator



- Edit the manifest file and give the app a new unique name

```
---
applications:
- name: <yourname>-locomotive-simulator-service
  path: target/locomotive-data-simulator-1.2.0.jar
env:
  SPRING_PROFILES_ACTIVE : cloud,clouddev
```

- Expand the src/main/resources folder and open the **application-cloud.properties** file
 - ◆ Replace the <placeholders> with your UAA and timeseries instances
 - Make the same changes to the **application-cloupdev.properties** file
- NOTE:** Examples of configured parameters are provided in the file.

```
#UAA configuration
#accessTokenEndpointUrl=https://2e93f1db-9f2c-43e9-9fa9-7ba816c8f01c.predix-uaa
#clientId=client
#clientSecret=client
#predix_oauthRestHost=2e93f1db-9f2c-43e9-9fa9-7ba816c8f01c.predix-uaa-training.
#predix_oauthClientId = asha-client:clientsecret
#predix_oauthGrantType = client_credentials
clientId=<yourClientName>

#Timeseries configuration
#timeseriesZone=1c84e446-aef6-44e9-9c38-f3f3d0b9ff7b
timeseriesZone=<your Time Series Zone ID>
#Your Data Ingestion Service URL
#timeseriesUrl=http://locomotive-dataingestion-service-asha.run.aws-usw02-pr.ica
timeseriesUrl=<Your Data Ingestion Service URL>/SaveTimeSeriesData
```

TIP: To find your dataingestion service URL, run the **cf a** command; the service url is displayed under the URL heading.

3. Build the project.

- In Eclipse **RunAs > Maven install**
- Verify the message: BUILD SUCCESS is displayed in the Console

4. Push the simulator service to the cloud.

- In the Terminal change directory to
`/predix/Documents/MyProject/locomotive-simulator directory`
- Run the command: **cf push**
- Verify the service was pushed successfully and started

```
Showing health and status for app student8-locomotive-simulator-service in org Predix-Tra
OK

requested state: started
instances: 1/1
usage: 1G x 1 instances
urls: student8-locomotive-simulator-service.run.aws-usw02-pr.ice.predix.io
last uploaded: Wed Mar 30 16:51:24 UTC 2016
stack: clinuxfs2
buildpack: java-buildpack=v3.6-https://github.com/cloudfoundry/java-buildpack.git#5194155
calculator=2.0.1_RELEASE spring-auto-reconfiguration=1.10.0_RELEASE



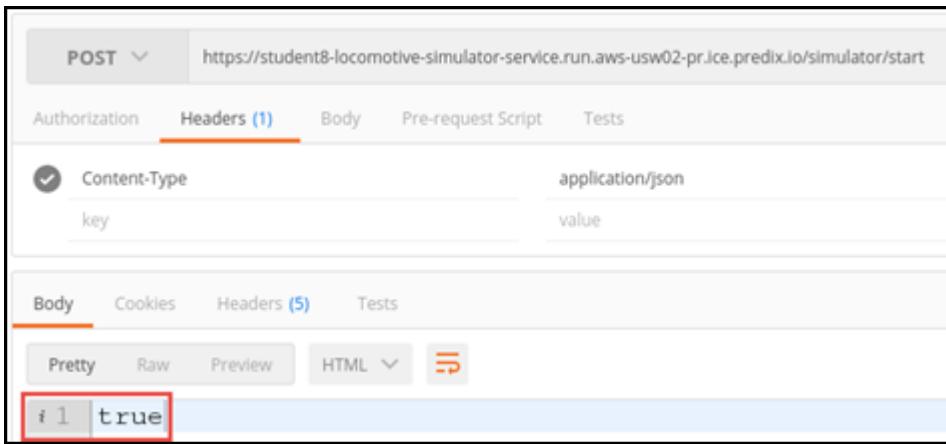
|    | <b>state</b> | <b>since</b>           | <b>cpu</b> | <b>memory</b> | <b>disk</b>  | <b>details</b>                            |
|----|--------------|------------------------|------------|---------------|--------------|-------------------------------------------|
| #0 | running      | 2016-03-30 09:52:11 AM | 0.0%       | 479.6M of 1G  | 137.4M of 1G | [predix@localhost locomotive-simulator]\$ |


```

5. Run the simulator service using the REST client.

- In Postman, select a new tab and configure a POST request
 - ◆ Method: POST
 - ◆ URL: `https://<>Your Data Simulator App Name><run.aws-usw02-pr.ice.predix.io/simulator/start`
- Configure request header
 - ◆ Content-Type: `application/json`
- Click **Send**
- The response should be `true`





Tip: The URL to stop the simulator service is: <https://<Your Data SimulatorName>.run.aws-usw02-pr.ice.predix.io/simulator/stop>

6. Check the logs for the locomotive-simulator service.

- In the Terminal run the command

```
cf logs <your-locomotive-simulation-service> --recent
```

- A rolling log will show data being generated by the simulator service

```
2016-03-30T10:26:51.63-0700 [App/0]      OUT 2016-03-30 17:26:51.632  INFO 29 --- [pool-1-thread-1] .p.s.t.s.s.Locomot
simulatorController : LocomotivesimulatorController :: generateSimulatorData - json result: {"rpmData":{"name":"LOCOM
VE_1_rpm","rpm":1006.0}, "currentTime":1459358811423, "torquedata":{"torque":22971.0,"name":"LOCOMOTIVE_1_torque"}, "locd
": {"longitude": "-73.50074577907473", "latitude": "41.41686319937058"}, "name": "LOCOMOTIVE_1_location"}}
2016-03-30T10:26:52.67-0700 [App/0]      OUT 2016-03-30 17:26:52.671  INFO 29 --- [pool-1-thread-1] .p.s.t.s.s.Locomot
simulatorController : LocomotivesimulatorController :: generateSimulatorData - result: You successfully posted data
2016-03-30T10:26:52.67-0700 [App/0]      OUT 2016-03-30 17:26:52.679  INFO 29 --- [pool-1-thread-1] .p.s.t.s.s.Locomot
simulatorController : LocomotivesimulatorController :: generateSimulatorData - json result: {"rpmData":{"name":"LOCOM
VE_2_rpm","rpm":1007.0}, "currentTime":1459358812671, "torquedata":{"torque":21906.0,"name":"LOCOMOTIVE_2_torque"}, "locd
": {"longitude": "-122.19479374340241", "latitude": "38.34352501320051"}, "name": "LOCOMOTIVE_2_location"}}
2016-03-30T10:26:52.93-0700 [App/0]      OUT 2016-03-30 17:26:52.932  INFO 29 --- [pool-1-thread-1] .p.s.t.s.s.Locomot
simulatorController : LocomotivesimulatorController :: generateSimulatorData - result: You successfully posted data
2016-03-30T10:26:53.62-0700 [App/0]      OUT 2016-03-30 17:26:53.623  INFO 29 --- [pool-1-thread-1] .p.s.t.s.s.Locomot
simulatorController : LocomotivesimulatorController :: generateSimulatorData
2016-03-30T10:26:53.63-0700 [App/0]      OUT 2016-03-30 17:26:53.631  INFO 29 --- [pool-1-thread-1] .p.s.t.s.s.Locomot
simulatorController : LocomotivesimulatorController :: generateSimulatorData - json result: {"rpmData":{"name":"LOCOM
VE_1_rpm","rpm":1007.0}, "currentTime":1459358813623, "torquedata":{"torque":22949.0,"name":"LOCOMOTIVE_1_torque"}, "locd
": {"longitude": "-73.89950368510293", "latitude": "41.37659985251643"}, "name": "LOCOMOTIVE_1_location"}}
```

7. Check the logs for the locomotive-dataingestion-service.

- In the Terminal run the command:

```
cf logs <your-locomotive-data-ingestion-service> --recent
```

- A rolling log will show the data being ingested into the Time Series database every 2 seconds



Exercise 3: Query Time Series Data

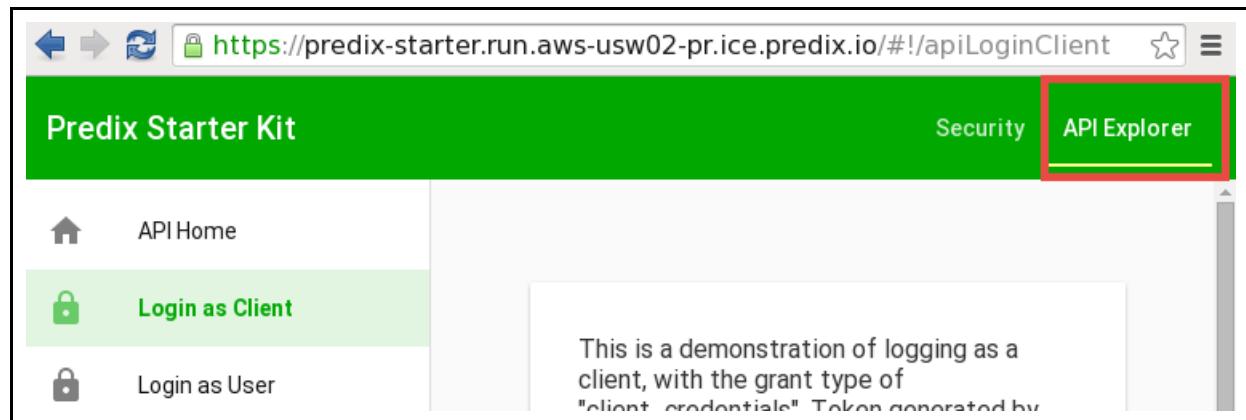
Overview

Use the Predix Starter Kit UI to query time series data generated by your locomotive-ingestion-service.

Steps

1. Login as the uaa client.

- In your browser, navigate to the Predix Starter Kit
<https://predix-starter.run.aws-usw02-pr.ice.predix.io/>
- Ensure API Explorer is selected and choose the **Login as Client** link



- Enter the following information in the Services dialog box
 - ◆ UAA URL: paste the URL from your text file
 - ◆ Client ID: <**your client id**>
 - ◆ Client Secret: <**your client secret**>
- Click **Submit**

A JWT token is returned indicating the request was successful

Services.

UAA URL:

Client ID:

Client Secret:

[See cURL command](#) [Submit](#)

```
{
  "access_token": "eyJhbGciOiJSUzI1NiJ9.eyJqdGkiOiIwYmEzOGFmNy04NjMwLTQzMjA0Yy0zNTE0ZjMxMTIjNzMiLCJzdWIiOiJhcHATY2xpZW50LWlkIiwi"
```

- Click See cURL command to view how the oauth/token extension is added to the URL

```
curl 'https://time-series-store-predix.run.aws-usw02-pr.ice.predix.io/v1/datapoints/latest' -X post -H 'Authorization: Bearer eyJhbGciOiJSUzI1NiJ9.eyJqdGkiOiI2YzMzQwZC0yNWViLTRmYTktODhmNS1kNGJkNGY0YWNjNDkilt43SGfp1CDukr6lLhJQGfdF1GMFeeTn13N5xs08U1zK9JJ-A-E6PofDcNL1Jx70gZ6YtPZQH8t81nZDNRTir62v6Y3psqt2tD2dx407Ms8Rcsxu9tol2sT284tr9zXpMoLFZYy69vC53se2appozY2_xQ0yTmwWwosHQ' --data-binary '{"tags":[{"name":"Compressor-2015:Compress
```

- Get all the tag names in your Time Series instance.

- In the Predix Starter Kit, click **Time series query** on the left
 - Select **Get Tags** from the **Choose Request:** drop down menu
 - The **GET** URL is automatically populated when you select the **Choose Request**
- Request Headers
 - The **predix-zone-id** is populated with the value from VCAP_SERVICES

UAA URL: <https://8ffde7f9-551a-454c-af33-9be3ec0fe4f5.predix-uaa-training.run.aws-usw02-pr.ice.predix.io>

Choose Request: Get Tags

GET <https://time-series-store-predix.run.aws-usw02-pr.ice.predix.io/v1/tags>

Request Headers

predix-zone-id

authorization [View Token](#)

[See cURL command](#) **Submit**

- Click **Submit**
- The results represent a JSON list of all the Tag names that exist in that Time Series instance

```
{  
  "results": [  
    "LOCOMOTIVE_1_location",  
    "LOCOMOTIVE_1_rpm",  
    "LOCOMOTIVE_1_torque",  
    "LOCOMOTIVE_2_location",  
    "LOCOMOTIVE_2_rpm",  
    "LOCOMOTIVE_2_torque"  
  ]  
}
```

3. Build an Absolute Time Bounded query request.

- Select **Time Bounded Request** in the **Choose Request:** drop down menu
- The API URL, Request Body, and the Authorization header are populated
 - ◆ If not displayed, enter the Predix-Zone-Id from your VCAP_SERVICES
- The Request Body
 - ◆ Add the Locomotive tag names as shown here
 - ◆ Remove the End time

Request Body

```
{"cache_time":0,"tags":[{"name":"LOCOMOTIVE_1_location"}, {"name":"LOCOMOTIVE_1_rpm"}, {"name":"LOCOMOTIVE_1_torque"}], "start":1452112200000}
```

1. Identification of sensor by the **tag** name
 2. Absolute **start** times in milliseconds
- Click **Submit** to run the query



- The query returns **all values** within the time period specified for a sensor (partial results shown here):

```
{  
  "tags": [  
    {  
      "name": "LOCOMOTIVE_1_location",  
      "results": [  
        {  
          "groups": [  
            {  
              "name": "type",  
              "type": "number"  
            }  
          ],  
          "attributes": {  
            "LATITUDE": [  
              "Latitude"  
            ],  
            "LONGITUDE": [  
              "Longitude"  
            ]  
          },  
          "values": [  
            [  
              1465508325810,  
              -73.50559935634234,  
              3  
            ],  
            [  
              1465508325810,  
              41.43987489504454,  
              3  
            ],  
            [  
              1465508325810,  
              41.43987489504454,  
              3  
            ]  
          ]  
        ]  
      ]  
    ]  
  ]  
}
```

Each value contains:

1. A timestamp
2. The measurement
3. Quality number
4. Total count of values in the query

4. Build a Query to return only the latest data point.

- Select **Latest Data Points Request** from the Choose Request drop-down menu
 - ◆ Note that the query has no start and end values

Request Body

```
{"tags": [{"name": "LOCOMOTIVE_1_location"}, {"name": "LOCOMOTIVE_1_rpm"}, {"name": "LOCOMOTIVE_1_torque"}]}
```

- Whether using a **GET** or **POST**, a latest data point query returns the latest data point:
 - ◆ Within a set time frame when you name the start and end times
 - ◆ Or, if you do not specify a start or end time, then by default, the query goes back 15 days and returns the latest data point through the current time
- Click **Submit**
- A single data point (the most recent) is returned

```
        "type": "number"
    }
],
"attributes": {
    "LATITUDE": [
        "Latitude"
    ],
    "LONGITUDE": [
        "Longitude"
    ]
},
"values": [
    [
        1465511274159,
        41.458956812490364,
        3
    ]
]
},
"stats": {
    "rawCount": 1
}
```



Exercise 4: Create a Locomotive Client Application

Overview

In the Asset lab, you loaded your data model into your asset service instance. As part of that activity, you added the **asset** OAuth scopes to your UAA client. These additional scopes were included in the UAA token, which allowed the client to access the asset API and post data to the asset database.

The Locomotive client application queries the time series data by calling the time series APIs.

Steps

1. Download the client service from github.

- In the Terminal change directory to `/predix/Documents/MyProject`
- Run the command:

```
git clone https://github.com/Training-Predix/locomotive-client.git
```

2. Import the locomotive project into Eclipse.

- **Import > Existing Maven Project**
- Root Directory: `/predix/Documents/MyProject/locomotive-client`
- The new locomotive-client project is displayed in Package Explorer

3. Configure the manifest file with your UAA, ACS and Asset information.

- Open the **manifest.yml** file and make the following changes:
 - ◆ applications:
 - name: replace <yourname> with your name
 - ◆ env:
 - Replace all <placeholders> with the information requested

```
---
applications:
  - name: <yourname>-locomotive-client-service
    buildpack: java_buildpack
    path: target/client-0.0.1-SNAPSHOT.jar #make sure this matches the artifactID

env:
  SPRING_PROFILES_ACTIVE : cloud,clouddev
  #acs integration
  #Use commented code as reference
  #accessTokenEndpointUrl : https://2e93f1db-9f2c-43e9-9fa9-7ba816c8f01c.predix-
  accessTokenEndpointUrl : https://<yourUAAzone>.predix-uaa-training.run.aws-usw02-pr.ice.predix.io
  #clientId : asha-client
  #clientSecret: clientsecret
  clientId : <yourclientName>
  clientSecret: <yourclientsecret>
  #acsServiceInstanceName : asha-locomotive-acs
  acsServiceInstanceName : <yourACSinstanceName>
  acsUri : https://predix-acs-training.run.aws-usw02-pr.ice.predix.io
  #acsZone : 58bb2518-5402-45d6-9e6f-675dff17f80e
  acsZone : <yourACSzone>
  #acsPolicyEvaluationTokenScope : predix-acs-training.zones.58bb2518-5402-45d6-9e6f-675dff17f80e
  acsPolicyEvaluationTokenScope : predix-acs-training.zones.<yourACSzone>.user
```

TIP: You created an ACS service instance in the Security lab. To retrieve your ACS zoneId, run the command: **cf service <acs_service_instance> --guid**

- Save and close the file



4. Update property files with your UAA and ACS instance.

- In the locomotive-client project, open the **/src/main/resources/application-cloud.properties** file
- Replace the values of the following parameters:
 - ◆ predix_oauthRestHost
 - ◆ predix_oauthClientId
 - ◆ accessTokenEndpointUrl
 - ◆ clientId
 - ◆ clientSecret
- Make these same changes to the **application-clouddev.properties** file

```
#predix_oauthRestHost=2e93f1db-9f2c-43e9-9fa9-7ba816c8f01c.predix-uaa-training.run.aws-usw02-pr.ice.predix.io
predix_oauthRestHost=<yourUAAzone>.predix-uaa-training.run.aws-usw02-pr.ice.predix.io
#predix_oauthClientId = asha-client:clientsecret
predix_oauthGrantType = client_credentials
predix_oauthClientId = <yourclientname>:<yourclientsecret>

#Timeseries configuration
#timeseriesZone=lc84e446-aef6-44e9-9c38-f3f3d0b9ff7b
timeseriesZone=<yourTimeSeriesZoneID>

#Acs configuration
#accessTokenEndpointUrl=https://2e93f1db-9f2c-43e9-9fa9-7ba816c8f01c.predix-uaa-training.run.aws-usw02-pr.ice.predix.io
accessTokenEndpointUrl=https://<yourUAAzone>.predix-uaa-training.run.aws-usw02-pr.ice.predix.io/oauth/token
#clientId=asha-client
#clientSecret=clientsecret
clientId=<yourClientName>
clientSecret=<yourclientsecret>

#acsServiceInstanceName=asha-locomotive-acs
acsServiceInstanceName=<yourACSinstanceName>
acsUri=https://predix-acs-training.run.aws-usw02-pr.ice.predix.io
#acsZone=58bb2518-5402-45d6-9e6f-675dff17f80e
acsZone=<yourACSZone>
#acsPolicyEvaluationTokenScope=predix-acs-training.zones.58bb2518-5402-45d6-9e6f-675dff17f80e.user
acsPolicyEvaluationTokenScope=predix-acs-training.zones,<yourACSZone>.user
```

5. Build the project.

- In Eclipse: **Run As > Maven install**
 - You should see the message **BUILD SUCCESS** in the Console
- TIP:** If you have any errors in your project, right-click on the project and select **Maven > Update project**

6. Deploy the project into Cloud Foundry.

- In the Terminal, change directory to
`/predix/Documents/MyProject/locomotive-client`
- Run the command: **cf push**
- Verify the service started by running the command: **cf a**



Exercise 5: View Data in a Dashboard

Overview

In this lab you will build a dashboard app using the Predix Dashboard Seed. The dashboard will display the locomotive data stored in the time series database. The Dashboard app calls the Locomotive client app to retrieve time series data. It also calls the asset service APIs to retrieve asset model data.

Part I: Create the Locomotive UI App

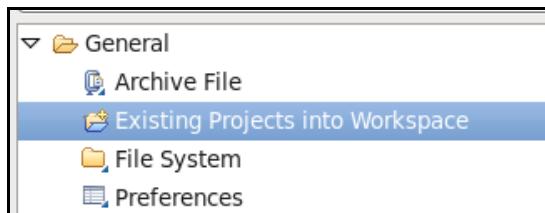
Steps

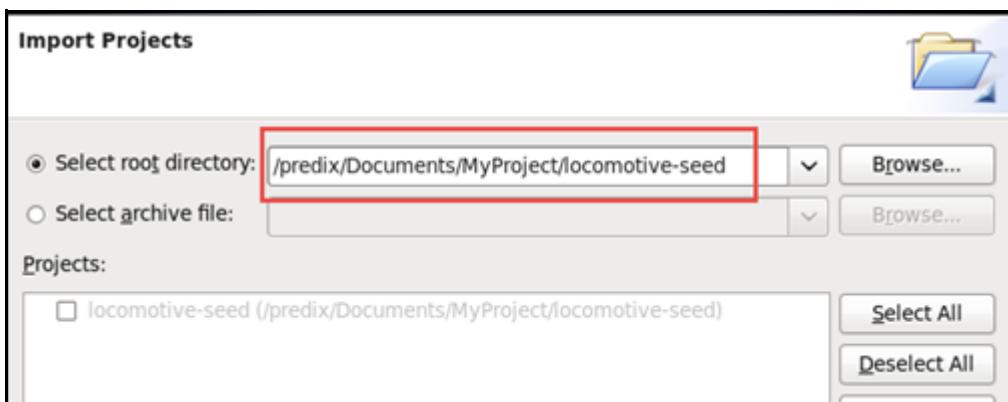
1. Download the Locomotive UI app from github.

- In a Terminal navigate to the /predix/Documents/MyProject folder
- Run the command:
`git clone https://github.com/Training-Predix/locomotive-seed`

2. Import the locomotive-seed project into Eclipse.

- Import > General > Existing Projects into Workspace





3. Update the manifest file.

- In Eclipse open the **manifest.yml** file for the locomotive-seed project
- Make the following changes to the file:
 - ◆ In the **applications:** section
 - application name: add your name to the beginning of the app
(e.g. JohnDoe-predix-seed)
 - ◆ In the **env:** section
 - Replace all <placeholders> with the information requested

TIP: Run the command: `cf service <service_instance> --guid` to quickly retrieve the zonelid for any of your service instances

```

---
applications:
  - name: student8-locomotive-predix-seed
    name: <yourname>-locomotive-predix-seed
    buildpack: predix_openresty_buildpack
    path: dist
    memory: 64M
    stack: cflinuxfs2
  services:
    #-<your Redis svc instance name> #change this to your redis service instance
    #-<your Views svc instance name> #change this to your Views service instance
    #-<your Asset svc instance name> #change this to your Asset service instance

env:
  #UAA_SERVER_URL: https://2e93f1db-9f2c-43e9-9fa9-7ba816c8f01c.predix-uaa-training.run.aws-usw02-pr.ice.
  UAA_SERVER_URL: <your UAA zone ID>.predix-uaa-training.run.aws-usw02-pr.ice.predix.io
  assetUrl: https://predix-asset.run.aws-usw02-pr.ice.predix.io
  #assetZoneId: 160fbeaf-1e6e-4499-922c-fea0525e548e
  assetZoneId: <your Asset Zone ID >
  #predix_oauthClientId: YXNoYS1jbG1lbnQ6Y2xpZW50c2VjcmV0
  predix_oauthClientId: <Base 64 Encoded value for clientId:clientSecret>
  #locomotiveClientUrl: https://locomotive-client-service-Asha.run.aws-usw02-pr.ice.predix.io
  locomotiveClientUrl: <your Locomotive client URL>
  REDIS: redis-1 #redis service name from the marketplace

```

4. Get the Base64 encoded value for your UAA client.

- In a Terminal, run the following command using the base64 CLI:

```
echo clientId:clientSecret|base64
```

- The encoded value is returned (highlighted)

```
[predix@localhost ~]$ echo clientId:clientSecret|base64
Y2xpZW50SWQ6Y2xpZW50U2VjcmV0Cg==
```

- Copy the encoded value and paste into the **<Base64 Encoded value for clientId:clientSecret>** placeholder in the manifest file
- Save the file

5. Deploy the application.

- In a Terminal change directory to
`/predix/Documents/MyProject/locomotive-seed`
- Run the command: **cf push**
 - ◆ The push is successful, but the app fails to start - why?
There are additional settings that need to be configured in the nginx.conf file. You will do this in a later step. For now, having the app deployed will enable you to bind it to a service instance.

Part II: Create redis and Views Predix Services

Overview

The redis service is used to store user sessions while the Views service is used to control the layout within a client-side web application. You render the state of card and deck objects in a web browser by creating a View service instance, then binding an application to the View service instance.

Steps

1. Create a redis service instance.

- In the Terminal run the command: **cf m**
 - ◆ A list of available services in the marketplace is displayed
 - ◆ Note the service and plan names for the services you will create
- Run the command:
`cf cs redis-1 shared-vm <your name>-redis`



2. Create a Views service instance.

- Run the command:

```
cf cs predix-views Standard <yourname>_locomotive_views -c  
'{"trustedIssuerIds": ["https://<uaa_zoneId>.predix-uaa-training.ru  
n.aws-usw02-pr.ice.predix.io/oauth/token"]}'
```

IMPORTANT: Make sure to use an underscore "_" instead of a hyphen "-" in your instance name. Hyphens are not supported for Views.

3. Bind the redis and views instances to the locomotive seed application.

- Run the commands:

```
cf bs <your locomotive-seed app> <your_locomotive_views_service>  
cf bs <your locomotive-seed app> <your-redis-service>
```

4. Output the redis and view VCAP services env variables to a file.

- Run the command:

```
cf env <your locomotive seed app> >>output.txt
```

Part III: Update OAuth Client with Scopes for the Views Service

1. Update the scope and authorities for the client.

- Create a text file to store the command and parameters used to update your uaa client
- Add the command to update our uaa client to the text file

```
uaac client update <your client>
```

- Append the following options to the end of the file:

```
--authorities "<existing authorities> openid scim.me  
uaa.resource views.zones.<view instanceId>.user"  
--scope "openid scim.me timeseries.zones.<time series zone  
id>.ingest timeseries.zones.<time series zone id>.query  
timeseries.zones.<time series zone id>.user  
predix-asset.zones.<asset zone id>.user"  
--autoapprove "openid scim.me"
```

- Update the <placeholders> in the text from the output.txt file you created
 - ◆ Replace <your client> with your uaa clientId
 - ◆ Replace <original authorities> with your client authorities; you can get this info by running the uaac clients command
 - ◆ Replace <view instance Id> with the instanceId of your Views service
- Copy the contents of the file and paste into a Terminal to execute the command

2. Add the view group.

- Run the command:

```
uaac group add views.zones.<view instance id>.user
```

- ◆ Replace <view instance id> with your Views service instance Id

3. Add users to the view group.

- Run the following commands:

```
uaac member add views.zones.<yourViewsInstanceId>.user <user>
```



Part IV: Update Locomotive Seed Application with UAA, Views and redis Service Instances

1. Update the manifest file.

- Make the following changes to the **services:** section
 - ◆ Uncomment all the services and replace with your Redis, Asset and UAA instance names

```
services:
  #<your Redis svc instance name> # change this to your redis service instance name
  #<your Views svc instance name> # change this to your views service instance name
  - student8-asset # asset instance name
```

- Save the file

2. Update settings in the nginx.conf file.

- Open the /dist/**nginx.conf** file
- Make the following changes to the **server** section:
 - ◆ \$client_id - replace <yourClientName> with your clientId
 - ◆ \$uaa_authorization_header - replace <Base64EncodedClientName:ClientSecret> with the Base64 code value you retrieved earlier

```
server
{
  set      $session_name          "predix-seed-session";
  set      $session_secret         U2GoXJPsG4jYx2G3Bn0k99Fle0+yNsqt7D92po40RvU=;
  #set    $client_id              asha-client;
  # Update the client_id variable with the your client Name
  set      $client_id              <yourClientName>;
  #set    $uaa_authorization_header "Basic YXNoYS1jbGllbnQ6Y2xpZW50c2VjcmV0";
  set      $uaa_authorization_header "Basic <Base64EncodedClientName:ClientSecret>";
  set      $user_token             '';
}
```

- Make the following changes to the **location /api/view-service** section:
 - ◆ proxy_set_header - replace <yourViewSvcInstanceId>
 - ◆ proxy_pass - replace <yourViewSvcInstanceId>

```
location /api/view-service
{
  proxy_set_header Authorization $user_token;
  #proxy_set_header predix-zone-id "<%= ENV['vcap_service_asha_locomotive_views_instanceId'] %>";
  proxy_set_header predix-zone-id "<%= ENV['vcap_service_<yourViewSvcInstanceId>_instanceId'] %>";
  rewrite           /api/view-service/(.*) /api/$1 break;
  #proxy_pass   "<%= ENV['vcap_service_asha_locomotive_views_uri'] %>";
  proxy_pass   "<%= ENV['vcap_service_<yourViewSvcInstanceId>_uri'] %>";
}
```

- Save the nginx.conf file

3. Prepare your application for deployment.

- In the **/PredixApps/training_labs/lab_files** directory, locate the following 2 files:
 - ◆ **deployLocomotiveUI.sh**
 - ◆ **npm_shrinkwrap.json**
- Copy and paste these files into the
`/predix/Documents/MyProject/locomotive-seed/predix-seed` directory
- In the Terminal navigate to the
`/predix/Documents/MyProject/locomotive-seed/predix-seed` directory
- Run the command: **./deployLocomotiveUI.sh**
 - ◆ The command completes but may have warnings which is fine



4. Deploy your locomotive-seed application.

- Run the command: **cf push**
- Verify the application deploys successfully and is started
- Copy the URL for the application

```
requested state: started
instances: 1/1
usage: 64M x 1 instances
urls: student8-locomotive-predix-seed.run.aws-usw02-pr.ice.predix.io
last uploaded: Thu Mar 31 21:11:58 UTC 2016
stack: cflinuxfs2
buildpack: predix_openresty_buildpack

     state      since        cpu    memory       disk      details
#0  running   2016-03-31 02:12:16 PM  0.0%  22.7M of 64M  60.7M of 1G
```

Part V: Test Your Locomotive-Seed Application

1. Log on to your locomotive seed application.

- If necessary, start your predix-locomotive-simulator service
- In a browser go to: <https://<>Your Data Simulator App Name>>.run.aws-usw02-pr.ice.predix.io/simulator/start>
- Click **Send** and verify the response is "true"

2. View your application logs.

- View the logs for your **simulator** service to verify data is being generated and written to your dataingestion service
`cf logs <your-locomotive-simulation-service> --recent`
- View the logs for your **dataingestion** service to verify data is being consumed and written to your timeseries service instance
`cf logs <your-locomotive-dataingestion-service> --recent`

3. Launch your locomotive-seed application.

- In a new browser tab, go to: <https://<>your locomotive-predix-seed-app URL>>
- At the login page, enter your **user** credentials (created in Security lab)



4. View locomotive asset data in the dashboard.

- On the landing page asset data is displayed for Locomotive1
 - ◆ Data generated by the locomotive-simulation service includes Location (Latitude, Longitude), RPM and Torque

The screenshot shows the Locomotive Seed Application interface. On the left, there is a navigation sidebar with options: Dashboards (selected), Asset, DataGraph, and Blank Page. The main content area has a title "LOCOMOTIVE LIVE DATA CHART". Below it, a section titled "DATA FOR LOCOMOTIVE 1" displays three data points in a table:

Locomotive 1 Location	LOCOMOTIVE_1.location	[1459525692724,-73.8700388465792,3]
Locomotive 1 RPM Data	LOCOMOTIVE_1.rpm	[1459525692724,1002,3]
Locomotive 1 Torque Data	LOCOMOTIVE_1.torque	[1459525692724,23063,3]

- In the navigation panel on the left, click the Assets tab
 - ◆ These are the assets added using the Asset service APIs to the locomotiveengine collection (**TIP:** Refresh your browser if data is not displayed).

The screenshot shows the Locomotive Seed Application interface. On the left, there is a navigation sidebar with options: Dashboards, Asset (selected), DataGraph, and Blank Page. The main content area has a title "Locomotive Asset Details". It displays two sets of locomotive details side-by-side:

Locomotive Serial No: LOCOMOTIVE_1	Locomotive Serial No: LOCOMOTIVE_2
Type: Diesel-electric	Type: Diesel-electric
Model: ES44AC	Model: SD70ACE
Emission Tier: 0+	Emission Tier: 0+
Fleet: /fleets/up-1	Fleet: /fleets/up-1
Manufacture: GE	Manufacture: /manufacturers/electro-motive-diesel
Engine: /engines/v12-1	Engine: /engines/v16-2-1
Installed On: 01/12/2005	Installed On:
Date ISO: 2005-12-01T13:15:31Z	Date ISO:

At the bottom of each column, there are "View Chart" buttons.

- Click on the **View Chart Details** button for one of your assets
 - ◆ Three graphs display data for location, RPM and torque



Appendix A *OPTIONAL: Working with ACS in the Locomotive Client*

Overview

Security Recap - in the Security lab, you created a user for your UAA client. You also created groups required for ACS in UAA and added the user to those groups. To demonstrate the level of control provided by the ACS service, you will create a second user that has fewer rights -they cannot access the latest time series data endpoint of the locomotive client application.

Part I - Configure UAA User Authority

Steps

1. Create a new user with limited access to time series data.

- In the Terminal run the command

```
uaac user add lesser-user --emails lesser-user@ge.com -p  
lesser-user
```

- Add this new user only to certain groups; they will be able to view timeseries data and asset data

- ◆ In the Terminal run the following commands (separately):

```
uaac member add timeseries.zones.<timeseries_zoneId>.user lesser-user  
uaac member add timeseries.zones.<timeseries_zoneId>.query lesser-user  
uaac member add timeseries.zones.<timeseries_zoneId>.ingest lesser-user  
uaac member add predix-asset.zones.<asset_zoneId>.user lesser-user
```

Part II - Add a New Policy to the ACS Instance

Here, you will add a new policy that defines which users can and cannot access data based on their role. Users with a role of 'admin' can view data while users with a role of 'operator' cannot access data.

1. Retrieve a UAA token.

- In Postman, submit a POST request to get a UAA token
(You will use the token to add a new policy to the ACS instance)
- You can use your existing token, but If your token has expired, get a new one

2. Locate the locomotive-acs policies files.

- In File Finder, navigate to the ReadyTech inbox
- The following file contains the acs policy information:
 - ◆ **md_app_policy.json**
- The following files define attributes for the policy:
 - ◆ **rmd_operator_role_attribute.json**
 - ◆ **rmd_admin_role_attribute.json**

3. Add a new policy to your ACS instance using Postman.

- Configure a new PUT request
 - ◆ Method: PUT ** Remember to do a PUT and not POST
 - ◆ URL:
<https://predix-acs-training.run.aws-usw02-pr.ice.predix.io/v1/policy-set/locomotive-acs-policy>



- Configure request headers
 - ◆ Authorization: Bearer <UAA token >
 - ◆ Content-Type: application/json
 - ◆ Predix-Zone-Id: <your acs-zonelid>
- In the Body add the locomotive-acs-policy JSON policy
 - ◆ Copy and paste the contents of the **rmd_app_policy.json** file into the body

The screenshot shows the Postman interface with a PUT request to the URL `https://predix-acs-training.run.aws-usw02-pr.ice.predix.io/v1/policy-set/locomotive-acs-policy`. The Headers tab shows `Content-Type: application/json`. The Body tab is selected and contains the following raw JSON policy code:

```
1 {
2     "name" : "locomotive-acs-policy",
3     "policies" : [
4         {
5             "name" : "allow-all-HTTP-requests-for-admin",
6             "target" : {
7                 "resource" : {
8                     "uriTemplate" : "/validateuser"
9                 },
10                "subject" : {
11                    "name" : "has-role",
12                    "attributes" : [
13                        {
14                            "issuer" : "https://acs.attributes.int",
15                            "name" : "role"
16                        }
17                    ]
18                },
19                "conditions" : [
20                    {
21                        "name" : "",
22                        "condition" : "match.single(subject.attributes('https://acs.attributes.int'))"
23                    }
24                ]
25            }
26        }
27    ]
28}
```

- Click **SEND**; you should see a 201 Created code returned indicating the policy was successfully added

4. Add attributes to the new ACS policy.

- Create a second PUT request
 - ◆ Method: PUT
 - ◆ URL:
`https://predix-acs-training.run.aws-usw02-pr.ice.predix.io/v1/resource/rmd_admin_role_attribute`
- Configure request headers
 - ◆ Authorization: PUT your UAA token (fetched earlier)
 - ◆ Content-Type: application/json
 - ◆ Predix-Zone-Id: <your acs-zonelid>
- Configure the Body of the request
 - ◆ Copy and paste the contents of the **rmd_admin_role_attribute.json** file
- Click **SEND**; you should see a 201 Created code returned indicating the attribute was successfully added to the policy

- Create a third PUT request
 - ◆ Method: PUT
 - ◆ URL:
`https://predix-acs-training.run.aws-usw02-pr.ice.predix.io/v1/resource/rmd_operator_role_attribute`
- Configure request headers
 - ◆ Authorization: PUT your UAA token (fetched earlier)
 - ◆ Content-Type: application/json
 - ◆ Predix-Zone-Id: <your acs-zonelid>
- Configure the Body of the request
 - ◆ Copy and paste the contents of the `rmd_operator_role_attribute.json` file
- Click **SEND**; you should see a 201 Created code returned indicating the attribute was successfully added to the policy

This client app is now setup as well as the client and the two users. The client has access to timeseries, asset and acs.



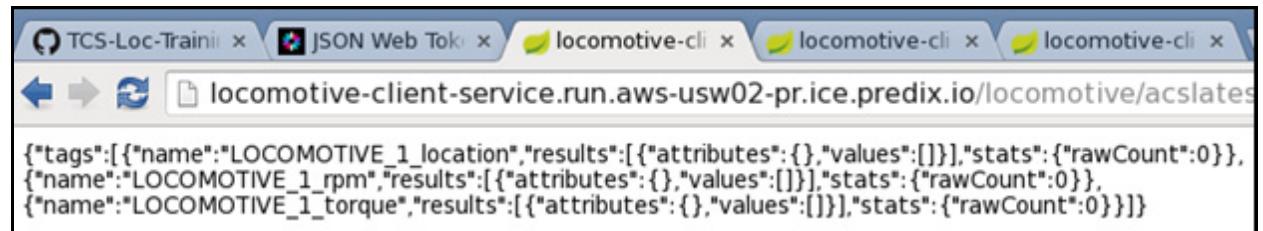
Part III - Test the ACS Policy Using a REST Client

1. Test for users in the 'admin' role.

- In a browser enter the url:

```
http://locomotive-client-service.run.aws-usw02-pr.ice.predix.io/lo  
comotive/acslatest?id=LOCOMOTIVE_1&username=geuser&password=geuser
```

This end point fetches the latest record from the timeseries database provided the username (geuser) has access to the resource after the ACS policy evaluation



A screenshot of a web browser window. The address bar shows the URL: "locomotive-client-service.run.aws-usw02-pr.ice.predix.io/locomotive/acslatest". The main content area displays a JSON object:

```
{"tags": [{"name": "LOCOMOTIVE_1_location", "results": [{"attributes": {}, "values": []}], "stats": {"rawCount": 0}}, {"name": "LOCOMOTIVE_1_rpm", "results": [{"attributes": {}, "values": []}], "stats": {"rawCount": 0}}, {"name": "LOCOMOTIVE_1_torque", "results": [{"attributes": {}, "values": []}], "stats": {"rawCount": 0}}]
```

2. Test for users in the 'operator' role.

- In a browser enter the url:

```
http://locomotive-client-service.run.aws-usw02-pr.ice.predix.io/lo  
comotive/acslatest?id=LOCOMOTIVE_1&username=geopeartor&password=ge  
operator
```

The user 'geopeartor' does not have access to get the latest request of the resource as per ACS policy

The screenshot shows a web browser window with four tabs open. The active tab is titled "locomotive-client-service.run.aws-usw02-pr.ice.predix.io/lo". The page content is a "Whitelabel Error Page". It displays the following text:

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Thu Mar 31 00:17:05 UTC 2016
There was an unexpected error (type=Internal Server Error, status=500).
error="access_denied", error_description="Error requesting access token."

Lab 7: 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 159](#)
- [*Creating a Controller*, page 164](#)
- [*Changing the View and Model*, page 166](#)
- [*Styling Your Application*, page 169](#)
- [*Creating a View to Display a Web Component*, page 174](#)
- [*Creating a Web Component*, page 178](#)
- [*Connecting a Microservice*, page 184](#)

Directions

Complete the exercises that follow.

Note: The code for these exercises is found in the **UI.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

The screenshot shows a web application interface for 'Predix'. On the left, there's a dark sidebar with a menu bar at the top containing three horizontal bars. Below it are four items: 'About' (with a house icon), 'Dashboard' (with a globe icon), 'Cards' (with a grid icon), and 'Components' (with a bar chart icon). The main content area has a white background. At the top, it says 'Dashboard Seed' in bold. Below that is a paragraph of text: 'Use the Dashboard Seed to develop your own industrial monitoring application. The Dashboard Seed is both a tutorial and a template that gets you up and running quickly. It provides you with capabilities such as a contextual browser, data source control, UI elements integration, internationalization, and Cloud Foundry deployment support. Just click the link below to clone this project and begin coding... it's that simple!'. At the bottom of the main area, there's another paragraph: 'Before you begin building an application please read the [Getting Started Github repo.](#)'.

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.

- 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 fileYour 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 `addPatient` 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 code
- This 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 file
- This 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

The screenshot shows a web application interface. On the left is a dark sidebar menu with the following items:

- About**
- Dashboard**
- Cards**
- Interactions
- Data Control
- Patients** (This item is highlighted with a blue background)

The main content area has a title **Patients**. Below it is a table with the following data:

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

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>
    <br/>
    <div class="flex">
```

- ◆ Save the file
- ◆ Refresh your browser



Your web page should look similar to the one below:

The screenshot shows a web application interface. At the top left is the 'Predix' logo. To its right is a vertical navigation bar with icons for 'About', 'Dashboard', 'Cards', 'Components', and 'Patients'. The 'Patients' icon is highlighted with a blue background. The main content area has a title 'Patients'. Below it are two input fields: 'First Name: i.e. Joe' and 'Last Name: i.e. Smith', followed by a 'Add Patient' button. A table titled 'Patient ID First Name Last Name' contains two rows of data: Row 1 (Patient ID 1) has columns Bob and Dylan; Row 2 (Patient ID 2) has columns Joe and Sammy.

- ◆ 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 `
` 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.

This screenshot shows the 'Patients' page of a web application. On the left is a dark sidebar with navigation links: 'About', 'Dashboard', 'Cards', 'Components', and 'Patients'. The 'Patients' link is highlighted with a blue background. The main content area has a title 'Patients' and two input fields: 'First Name: i.e. Joe' and 'Last Name: i.e. Smith', followed by a 'Add Patient' button. Below these is a 'Search:' input field. A table lists 13 patients with columns 'Patient ID', 'First Name', and 'Last Name'. The data is as follows:

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

This screenshot shows the same 'Patients' page after a search. The 'Search:' input field now contains 'Riv'. The table below shows only the patients whose last name starts with 'Rivera':

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
Your Patients page appears as below:

The screenshot shows a web application interface. On the left is a sidebar with the following menu items:

- About
- Dashboard
- Cards
- Interactions
- Data Control
- Patients** (This item is highlighted with a blue background)
- Components

The main content area has a title "Patients". It contains two input fields: "First Name: Last Name: ". Below these is a "Search:" input field. A "Add Patient" button is located next to the last name input field. At the bottom is a table with the following data:

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="l
    </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 :true"`

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
```

- Save your file
- Reload the page in your browser



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

The screenshot shows a mobile application interface. On the left is a dark sidebar menu with the following items:

- About
- Dashboard
- Cards
- Interactions
- Data Control
- Patients** (highlighted in a teal bar)
- Components

The main content area is titled "Patients". It contains the following form fields:

- First Name:
- Last Name:
- Add Patient** (a blue button)
- Search:

Below the search field is a table with patient data:

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

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 "patients" 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'
});

```

- Save your file

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'}
]

```

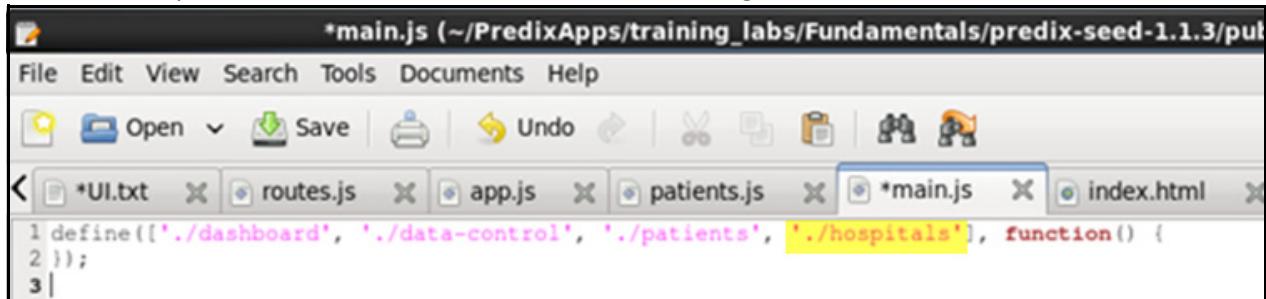
- Press `<ctrl> + <s>` to save the file

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

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



```
*main.js (~/PredixApps/training_labs/Fundamentals/predix-seed-1.1.3/public/scripts/controllers)
```

```
File Edit View Search Tools Documents Help
```

```
Open Save Undo
```

```
*UI.txt routes.js app.js patients.js *main.js index.html
```

```
1 define(['./dashboard', './data-control', './patients', './hospitals'], function() {
```

```
2 });
3 |
```

- ◆ A new **hospitals** reference is added to the file
- Save the file

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

- In 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.

- In 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.

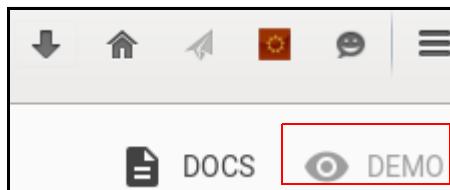
Normally, you would download and install the component and its dependencies from GitHub and then generate the component using a Predix component Yeoman generator. This component renders a simple HTML table that displays hospital data. The Yeoman generator allows developers to specify how to build their 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

Although related, examples are /not/ the same as the test fixtures. Tests are tests, examples are examples.

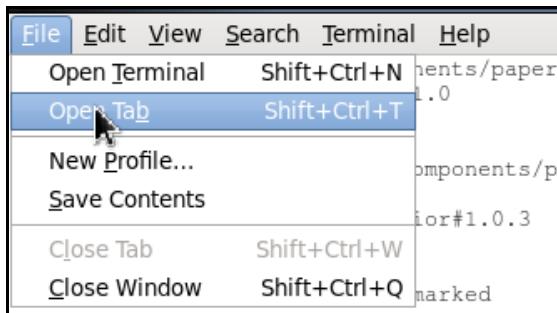
Also see the [documentation](#) and the [UI test fixture](#).

Click on the counter to make it increment:

4

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>
32           {{hospitalDetails.name}}</span></td></tr>
33         <tr><th class="text--right">Hospital Address :</th><td><span>
34           {{hospitalDetails.address}}</span></td></tr>
35         <tr><th class="text--right">Email :</th><td><span>
36           {{hospitalDetails.email}}</span></td></tr>
37         <tr><th class="text--right">Phone :</th><td><span>
38           {{hospitalDetails.phone}}</span></td></tr>
39         </table>
40     </div>
41   </template>
42 </dom-module>Rel
43
44 <script>
45   Polymer({
46     is: 'hospital-info',
47     properties: {
48       hospitalDetails: {
49         type: Object
50       }
51     }
52   });
53 </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.

- Open the `predix-seed-1.1.3/public/views/hospital/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>
```

- Stop all `grunt watch` running processes.
 - ◆ In all terminals running a `grunt watch` process, press `<ctrl>+<c>`
- Run `grunt serve` from the predix seed directory to see the new component:

The screenshot shows the 'Demo Healthcare System' application interface. On the left is a sidebar with navigation links: 'About', 'Dashboard', 'Cards', 'Interactions', 'Data Control', 'Patients', 'Hospitals' (which is highlighted in blue), and 'Components'. The main content area has a title 'HOSPITALS'. Below it, a message says 'There are 2 hospitals within this network.' To the right is a detailed 'Hospital Details' card with the following information:

Hospital Name:	Stanford Hospital and Clinics
Hospital Address:	300 Pasteur Drive
Email:	mike@stanford.edu
Phone:	6507234000

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>
```

Part I: Connecting from a Polymer Web Component

Steps

1. Update bower.json.

- In the /predix-seed-1.1.3/public/bower_components/px-theme directory edit the **bower.json** file
- Change the **px-forms-design** property to:
`"https://github.com/PredixDev/px-forms-design.git#-0.39"`



```
"name": "px-theme",
"version": "0.3.12",
"main": [
  "px-theme.html"
],
"ignore": [
  ".*",
  "npm-debug.log",
  "Gruntfile.js",
  "sass",
  "bower_components",
  "node_modules",
  "test"
],
"dependencies": {
  "polymer": "~v1.0.3",
  "px-iconography-design": "https://github.build.ge.com/PXd/px-iconography-design.git#~0.2.0",
  "px-forms-design": "https://github.com/PredixDev/px-forms-design.git#~0.3.9" 
```

2. 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.

3. 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.

4. 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 8: *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

- *Set Up the Helloworld App, page 191*
- *Create and Bind an Analytics Catalog Service Instance, page 193*
- *Create and Bind an Analytics Runtime Service Instance, page 197*
- *Updating the OAuth 2 Client, page 200*



Exercise 1: Set Up the Helloworld App

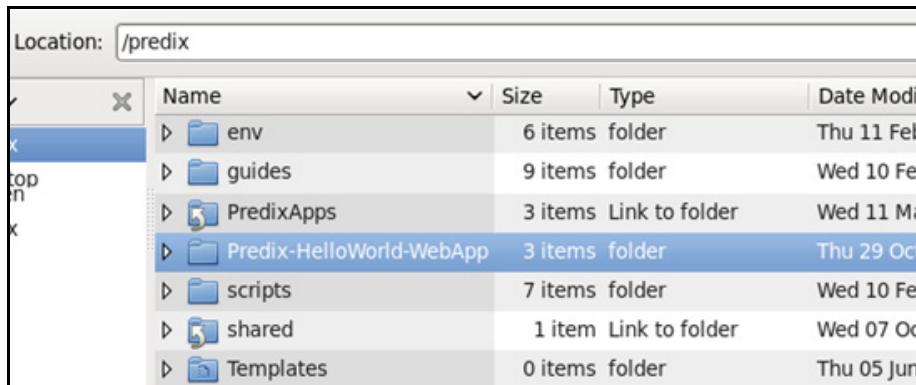
Overview

The helloworld app is a preconfigured app that you will bind to your Catalog, Runtime and UI service instances.

Steps

1. Extract the helloworld application file.

- In a Terminal change directory to
`/predix/Predix-Apps/trianing_labs/AnalyticsLabFiles`
- Unzip the **Predix-HelloWorld-WebApp-master.zip** file
- Rename the `Predix-HelloWorld-WebApp-master` directory to
Predix-HelloWorld-WebApp
- Using File Finder, move the **Predix-HelloWorld-WebApp** folder to the `/predix` folder



- In the Predix-HelloWorld-WebApp folder, edit the **manifest.yml** file
 - ◆ Change the application name to <yourname>-helloworld

```
applications:  
  #- name: hello-world-app # Step 1: Change your application name  
  - name: student5-helloworld  
    buildpack: predix_openresty_buildpack  
    #path: dist  
    memory: 64M
```

- Save and close the file

2. Push the HelloWorld application to the cloud.

- In the Terminal change directory to /predix/Predix-HelloWorld-WebApp
 - ◆ Run the command: **cf push**
- Run the **cf a** command to verify your HelloWorld application is available

3. Bind your hello-word application to your UAA service instance.

- In the Terminal run the command

```
cf bind-service <your HelloWorld app> <your UAA instance>
```



Exercise 2: 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)
```

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

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

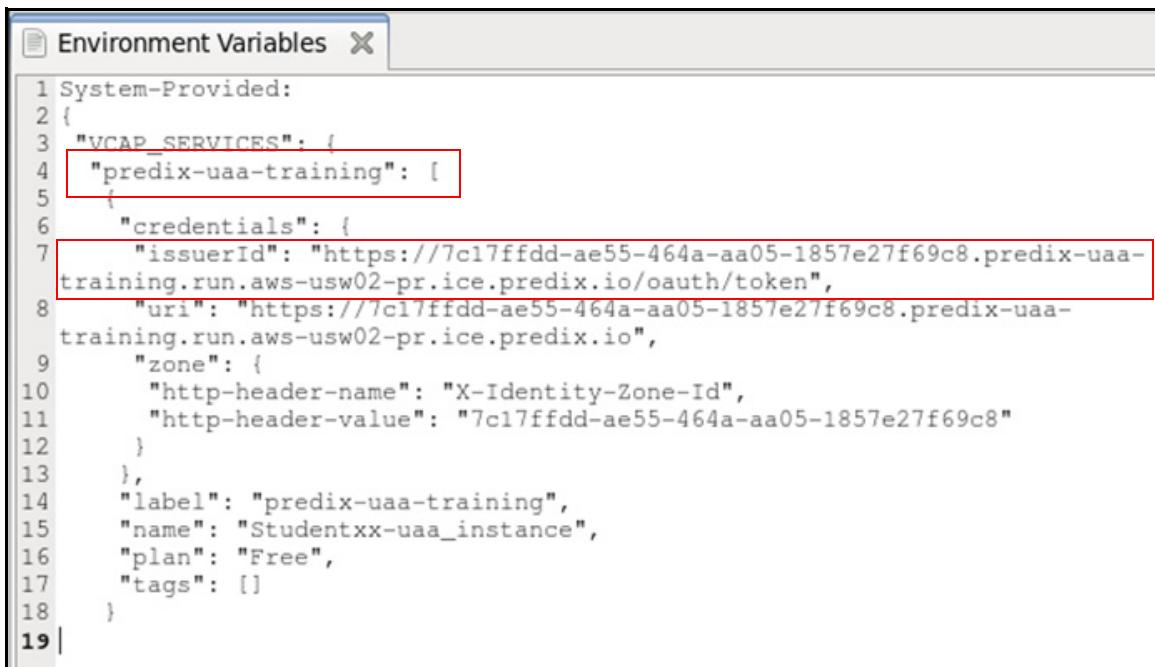
2. View and record your UAA environment variables.

- 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            "label": "predix-uaa-training",
16            "name": "Studentxx-uaa_instance",
17            "plan": "Free",
18            "tags": []
19          }
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 **Bronze**
- <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-Fir
stname75 -c '{"trustedIssuerIds": ["https://a97db685-e3f5-4f10-ad03-f82b
ee5a3808.predix-uaa-training.run.aws-usw02-pr.ice.predix.io/oauth/token"]}''
Creating service instance analytics-catalog-Fir
stname75 in org Predix-Traini
ng / 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 3: 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.

- 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>"]}'
```

where:

- <plan> is **Bronze**
- <my_runtime_instance> is the name of your analytics runtime service instance
- <uaa_instance1_issuerId> is the issuerId of your UAA service instance

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
```

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 4: 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-acss-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-acs-training.zones.e53aa7f1-62c6-47a7-  
91f5-9f5ae9554dd0.user  
analytics.zones.0bfcc3b0-4626-4980-bc5b-cc3fb6543b6.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 id>`
- 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 205
- *Working with the Analytics Catalog*, page 220



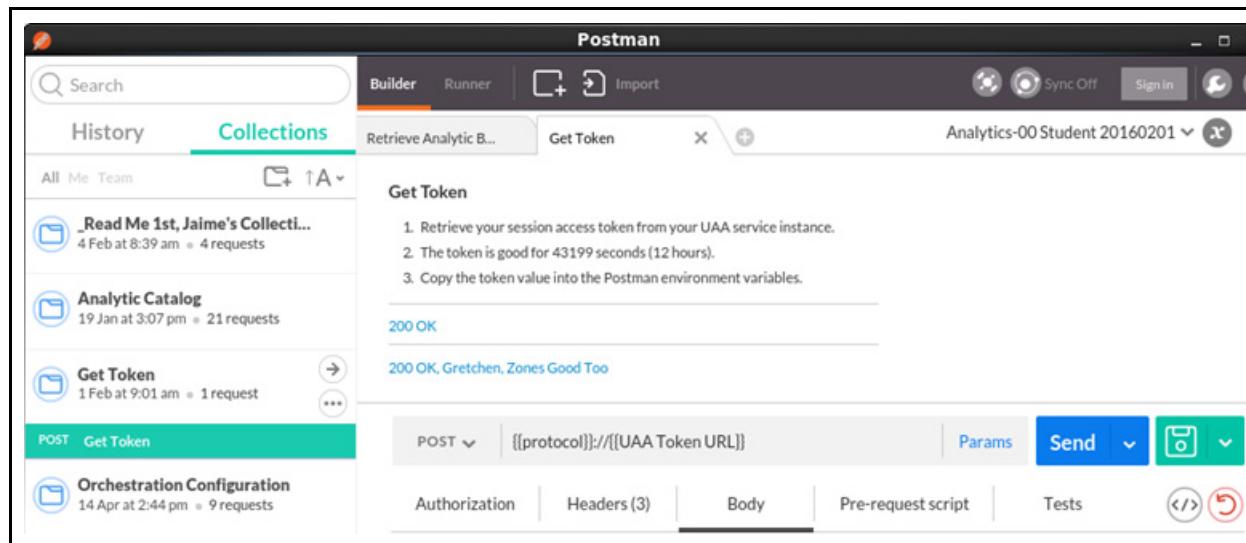
Exercise 1: Using Postman and Getting Your UAA Token Value

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 interface. A modal window titled "Manage environments" is open, displaying a list of environment variables for the environment "Analytics-00 Student 20160201". The variables listed are:

Variable	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.aw
config_uri	predix-analytics-config-release.run.aw
configurationId	Value
Content-Type	application/json
deploymentrequestId	Value
execution_uri	predix-analytics-execution-release.rui
jobId	Value
protocol	https
runtime_tenant	497a5a2f-bf8f-430c-9f45-bf4d3d4c6
scheduler_uri	predix-scheduler-service-release.run.:
token	eyJhbGciOiJSUzI1NiJ9.eyJqdGkiOiJk
UAA Token URL	5d05f50d-387f-4646-a170-fa85af26'
validationrequestId	Value
X-Identity-Zone-Id	5d05f50d-387f-4646-a170-fa85af26'
Key	Value



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 screenshot shows the Postman interface with a collection named "Retrieve All Analytics". The URL field contains the placeholder {{protocol}}://{{catalog_uri}}/api/v1/catalog/analytics. Three environment variables are highlighted with orange arrows: "Content-Type" (application/json), "Authorization" (Bearer {{token}}), and "Predix-Zone-Id" ({{catalog_tenant}}). The "Headers (3)" tab is selected in the navigation bar.

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.

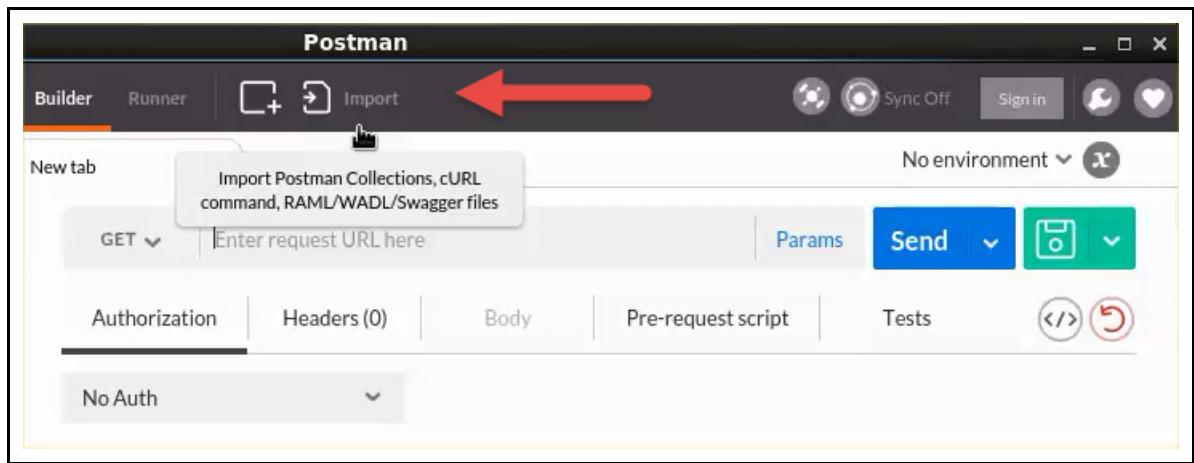
Steps

1. Open the Postman tool.

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

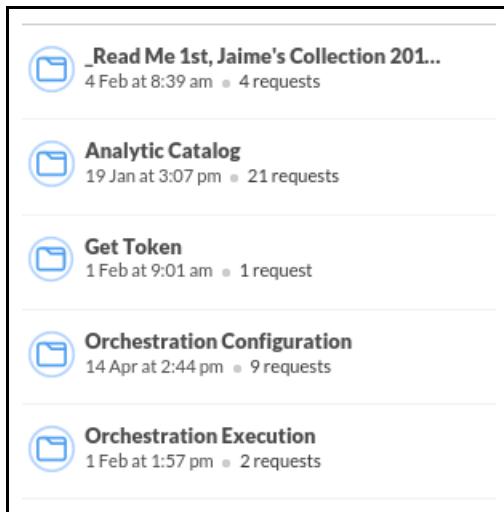
2. Import Postman collections to facilitate API requests.

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



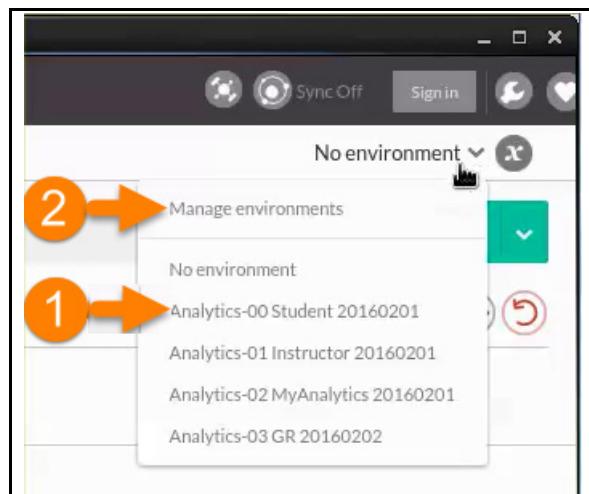
- Browse to `predix/PredixApps/training_labs/AnalyticsLabFiles` for the Postman "dump" file called **Analytics_Backup_2016-02-04.postman_dump**
- Click **Open**, then **Import**.
- A Collection already exists window appears. Close the window without selecting either button. If you get two collections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



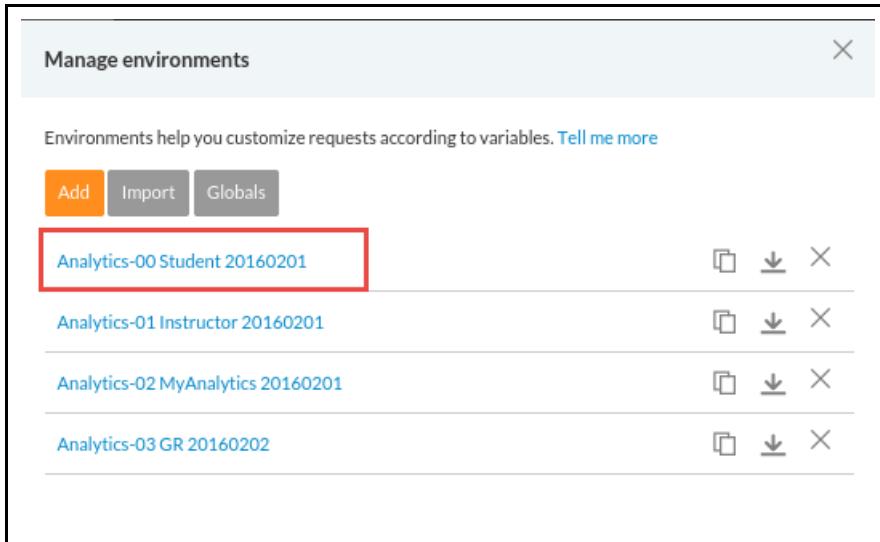
3. Set your Postman environment for Analytics.

- 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**

The screenshot shows the Postman application interface. The top navigation bar includes 'Builder', 'Runner', 'Import', and 'Analytics'. The left sidebar shows 'History' and 'Collections'. Under 'Collections', there are three items: '_Read Me 1st, Jaime's Collection...' (4 Feb at 8:39 am), 'Analytic Catalog' (19 Jan at 3:07 pm), and 'Get Token' (1 Feb at 9:01 am). An orange arrow points from the bottom of the 'Get Token' collection to its name. The main workspace is titled 'Get Token' and shows a POST request. The 'Params' tab contains the URL: `[[protocol]]://[[UAA Token URL]]`. The 'Body' tab is selected and contains the following JSON payload:

```
1 client_id=admin&grant_type=client_credentials&client_secret=myadminsecret
```

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

The screenshot shows the Postman application interface. On the left, the 'Collections' sidebar is open, displaying a list of collections and requests. The 'Analytics' collection is selected, and its requests are listed. The 'Retrieve All Analytics' request is highlighted with a green bar at the top. In the main workspace, the 'Builder' tab is active, showing the request details. The 'Authorization' tab is selected, and a dropdown menu is open, showing options: 'No Auth', 'Basic Auth' (which is highlighted with a cursor icon), 'Digest Auth', 'OAuth 1.0', 'OAuth 2.0', 'Hawk Authentication', and 'AWS Signature'. The request URL is {{protocol}}://{{catalog_uri}}/api/v1/catalog/analytics, and there are three Headers defined.

- ◆ 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**



The screenshot shows the Postman interface with a POST request to {{protocol}}://{{UAA Token URL}}. The Authorization tab is selected, showing 'Basic Auth' with 'Username: client' and 'Password:'. A note says 'The authorization header will be generated and added as a custom header.' Below the fields are 'Show Password' and 'Save helper data to request' checkboxes, and 'Clear' and 'Update request' buttons.

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**

The screenshot shows the Postman interface with a POST request to {{protocol}}://{{UAA Token URL}}. Step 1 is highlighted by an orange arrow pointing to the 'Analytics-00 Student 201602' environment dropdown. Step 2 is highlighted by an orange arrow pointing to the 'Headers (3)' tab, which has 'x-tenant', 'Content-Type', and 'Authorization' checked. Step 3 is highlighted by an orange arrow pointing to the 'Send' button.

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

The screenshot shows a Postman API client interface. At the top, there's a header bar with 'POST' dropdown, 'Params' button, 'Send' button, and a copy icon. Below the header, tabs for 'Authorization', 'Headers (3)', 'Body', 'Pre-request script', and 'Tests' are visible. Under 'Headers (3)', three items are listed: 'x-tenant' with value '[[X-Identity-Zone-Id]]', 'Content-Type' with value 'application/x-www-form-urlencoded', and 'Authorization' with value 'Basic Y2xpZW50OmNsawVuudHNIY3JldA=='. The 'Body' tab is selected, showing a status of '200 OK' and a time of '154 ms'. The response body is displayed as a JSON object:

```

1 {
2   "access_token": "eyJhbGciOiJSUzI1NiJ9.eyJqdGkiOiIwYmFmM2NiMi01ZTJhLTQ0NzIt0Dc1MS0yN2NkMDE2ZmU0MjYiLCJzdWIi0jzbGL
3     bnbQ1LCJzY29nWSI6WjybGLbnRzLnJlYWQiLCJhbmcFseXRpY3Muem9uZXMuZGUx0DMyYWYtNWEyNi00M1LThMTctYjBiTViN2M4MDlilnVzzX
4     iLCJ6b251cy4yNzA5Yjk3MS03MDQyLTQ30DItYjU4Yy1kMWYxZDBmNmEyOGYuYWRtaW4iLCJjbGllbnRzLndyaXRliwiYW5hbHL0awNzLnpvbmVzL
5     jQ1NmIyZjUyLWY3NGItNDQzMClN2RjLWYyZjg5YmVjYjU00S5ic2Vyiwi2NpbSS3cmL0ZSi1InNjaiw0ucmVhZCJdCJjbGLbnRfaWQ10ijjbGLl
6     bnQ1LCJjaWQ10iJjbGLbnQ1LCJncmf90eXBLIjoiY2xpZw50X2NyZWRlnRpYwxiIiwiemV2X3NzOyI6IjE5YTBhNmJ
7     lIiwiawF0IjoxNDU4Mz10TAyLCJleHai0jE0NTgznNzxD1sImLzcyI6Imh0dHBzOi8Vmjcw0W15NzEtNzA0Mi00NzgyLWI10GMtZDFmMWQzjZhMj
8     hmLnByZWRpeC11YWEtdHjaW5pbmcucnVuLmF3cyi11c3cwMi1wcis5pY2UucHJlZG14LmlvL29hdXRoL3Rva2VuIiwiemlkjoiMjcw0W15NzEtNzA0M
9     100NzgyLW110GMtZDFmMWQzjZhMjhmlwiYXVkipbImNsawVuudHm1LCJhbmcFseXRpY3Muem9uZXMuZGUx0DMyYWYtNWEyNi00M1
10    LTlhmTctYjBiYTViN2M4MDliliwiem9uZXMuMjcw0W15NzEtNzA0M100NzgyLWI10GMtZDFmMWQzjZhMjhmlwiYW5hbHL0awNzLnpvbmVzLjQ1NmI
11    yZjUyLWY3NGItNDQzMClN2RjLWYyZjg5YmVjYjU00S5ic2VyiwiX0O_gz9m7u17wSjFKUeFDWwAW4heAjMYqX4NWLmn5EIt-PE6RUsRRM9B9QlNZ
12    oKHQutiz8k0MkzNj3YJMpz9xKDTF5GC1EQzo6D7PYaTHqBiFF1a9x2CL9gpu497VgXdFRrsMcPzAFbIBeBkAVyDpc1Xwz55-L4sM0w8vjZIRRvFB_o
13    YtPr7TsC1tC0eJ9M91gU0nUS4rE01da7W19VJX3JGQRJnMYg_S8L8_sStBpGVSB6yjxHiT4f_jDqUAPH-YiaLPc-sgXNn1j500Km6FJJ-QBnkdo0nXT
14    c891-w7jD1vUSt1kbdj_I2KHZzsVWPoYcxPdm70dhT6AhYaMz5MsW",
15   "token_type": "bearer",
16   "expires_in": 43199,
17   "scope": "clients.read analytics.zones.de1832af-5a26-41c5-9a17-b0ba5b7c809b.user zones.2709b971-7042-4782-b58c
18     -d1f1d0f6a28f.admin.clients.write analytics.zones.456b2f52-f74b-4430-b7dc-f2f89becb549.user scim.write scim.read",
19   "jti": "0ba5cb2-5e2a-4472-8751-27cd016fe426"
20 }

```

Note: The token has a limited life span - you may need to get a new token every few hours.

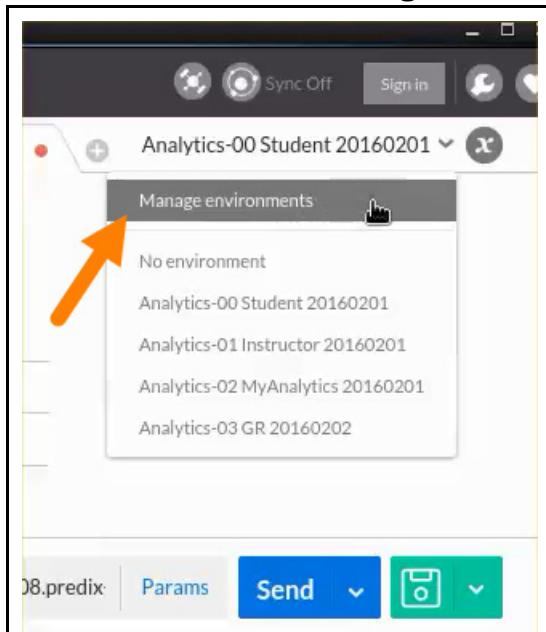
The **access_token** represents the UAA token value. Copy the text between the **access_token** quotes.

The screenshot shows a Postman API response with the "Body" tab selected. The response content is a JSON object with the following structure:

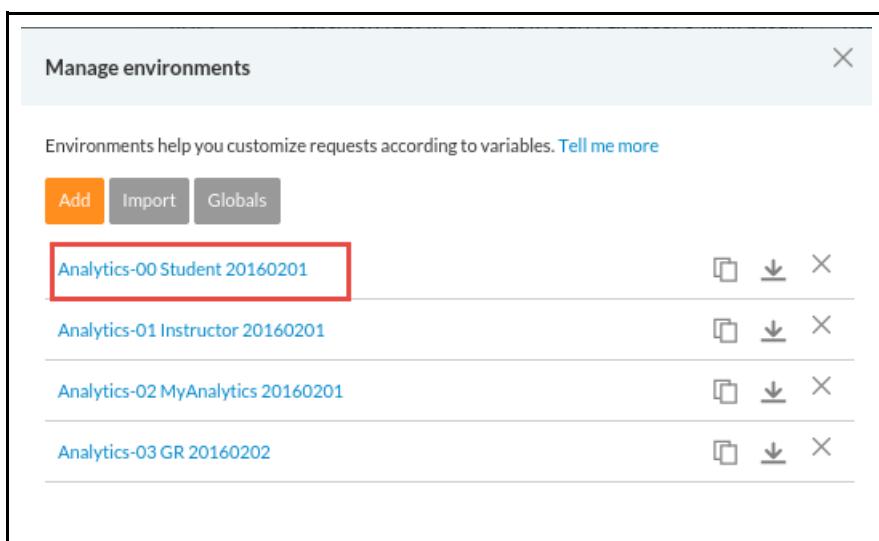
```
1 {  
2   "access_token": "eyJhbGciOiJSUzI1Ni  
hNjkyNTI5MDNhZGY1LCJzdWI0iJhZG1pbpiIs  
VjcmV0IiwiaWRwcy53cmI0ZSIsInVhYSSyZXN  
DAzLWY4MmJlZTVhMzgwOC5hZG1pbpiIsImNsaw  
ZWFlIiwic2NpbS53cmI0ZSIsInNjaW0ucmVhZ  
uIiwiYXpwIjoiYWRtaW4iLCJncmFudF90eXBL  
U1ZTdjYTZjIiwiaWF0IjoxNDU2MTcxMjYwLCI  
GI2ODUtZTNmNS00ZjEwLWFkMDMtZjgyYmVLNw  
c3cwMi1wci5pY2UucHJlZGL4LmlvL29hdXRol  
kMDMtZjgyYmVLNWEzODA4IiwiYXVkJpbImfk  
5hOTdkYjY4NS1lM2Y1LTNmIAtyWQwMy1mODJ  
W5MiR0BRZFLFKrHWWMA1dLDETT2YkOiqXGUft  
tVhw0-JquYo6uh8Qc8nrsd41auFqrViemRats  
-89iGiIdDIvjq3ydYutRYFA1TDNyGOr72gopcI  
WaTQqx9XMQy1WkHag2yGV3t2EMcemH6vQXYgx  
3   "token_type": "bearer",  
4   "expires_in": 43199,  
5   "scope": "clients.read clients.sec  
-4f10-ad03-f82bee5a3808.admin clients  
.read",  
6   "jti": "a63a8fea-2325-46b4-82f5-a69  
7 }
```

A context menu is open over the "access_token" value, with "Copy" highlighted. Other options in the menu include Undo, Redo, Cut, Paste, Paste as plain text, Delete, Spell-checker options, Writing Direction, Select all, Set: Analytics-00 Student 20160201, Set: Globals, EncodeURIComponent, and DecodeURIComponent.

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



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



- ◆ Paste the copied token value into **token**
- ◆ Click **Submit** to save the variable

Manage environments

Back Submit

Analytics-00 Student 20160201

	Value
<input checked="" type="checkbox"/> analyticId	Value
<input checked="" type="checkbox"/> analyticName	+9...a2...df...ac...ab...3d...
<input checked="" type="checkbox"/> scheduler_uri	predix-scheduler-service-release.run...
<input checked="" type="checkbox"/> token	eyJhbGciOiJSUzI1NiJ9eyJqdGkiOiJhNjNhOGZLYS0yMzI1LTQ2YjQtODJmNS1hNjkyNTI5MDNhZGYiLCJzdWIiOiJhZG1pbilIsInNjb3BLIjpBIImNsawVudHMucmVhZCIsImNsawVudHMuc2VjcmV0IiwiaWRwcy53cmI0ZSIisInVhYS5yZXNvdXJjZSIisInpvbmVzLmE5N2RiNjg1LWUzZjUtNGYxMC1hZDAzLWY4MmJLZTVhMzgwOC5hZG1pbilIsImNsawVudHMud3JpdGUiLCJjbGllbnRzLmFkbWluIiwiawRwcy5yZWFKIiwic2NpbS53cmI0ZSIisInNjaw@ucmvhZCJdLCJjbGllbnRfaWQioiJhZG1pbilIsImNpZCI6ImFkbWLuiiwiYXpwIjoiYWRtaW4iLCJncmFudF90eXBILjoiY2xpZW50X2NyZWRLbnRpYwxiIiwiicmV2X3NpZyI6IjU1ZTdjYTZjIiwiawWF0IjoxNDU2MTcxMjYwLCJleHAiOjE0NTYyMTQ0NjAsImLzcyI6Imh0dBzOi8vYTk3Z
<input checked="" type="checkbox"/> UAA Token URL	a97db685-e3f5-4f10-ad03-f82bee5a...

- ◆ Close the **Manage environments** window
- You can click **Save response** to keep an examples for future reference. Choose a name that fits the example, e.g. 200 OK - **Client Token Request**.

x-tenant: {{X-Identity-Zone-Id}}

Content-Type: application/x-www-form-urlencoded

Authorization: Basic YWRtaW46bXlhZG1pbnNlY3JldA==

Header Value Presets

Status: 200 OK Time: 278 ms

Body Cookies Headers (15) Tests (0/0)

Pretty Raw Preview JSON

Save response

```

1 <!
2   "access_token": "eyJhbGciOiJSUzI1NiJ9eyJqdGkiOiJhNjNhOGZLYS0yMzI1LTQ2YjQtODJmNS1hNjkyNTI5MDNhZGYiLCJzdWIiOiJhZG1pbilIsInNjb3BLIjpBIImNsawVudHMucmVhZCIsImNsawVudHMuc2VjcmV0IiwiaWRwcy53cmI0ZSIisInVhYS5yZXNvdXJjZSIisInpvbmVzLmE5N2RiNjg1LWUzZjUtNGYxMC1hZDAzLWY4MmJLZTVhMzgwOC5hZG1pbilIsImNsawVudHMud3JpdGUiLCJjbGllbnRzLmFkbWluIiwiawRwcy5yZWFKIiwic2NpbS53cmI0ZSIisInNjaw@ucmvhZCJdLCJjbGllbnRfaWQioiJhZG1pbilIsImNpZCI6ImFkbWLuiiwiYXpwIjoiYWRtaW4iLCJncmFudF90eXBILjoiY2xpZW50X2NyZWRLbnRpYwxiIiwiicmV2X3NpZyI6IjU1ZTdjYTZjIiwiawWF0IjoxNDU2MTcxMjYwLCJleHAiOjE0NTYyMTQ0NjAsImLzcyI6Imh0dBzOi8vYTk3Z"

```

- Click the Saved Response link to view

The screenshot shows the Postman application interface. At the top, there are three tabs: "Retrieve All Analytics" (red dot), "Get Token" (red dot), and "Analytics-00 Student 20160201" (grey dot). Below the tabs, the title "Get Token" is displayed. A list of instructions follows:

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 are three response sections:

- 200 OK**: Contains the text "200 OK, Gretchen, Zones Good Too".
- 200 OK - Admin Token Request**: This section is highlighted with a red border.



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)

The screenshot shows the 'Manage Environments' interface with the 'Edit Environment' dialog open for the environment 'Analytics-00 Student 20160201'. The dialog lists various environment variables with their values and edit icons. Two variables, 'catalog_tenant' and 'runtime_tenant', are highlighted with red boxes around their respective rows.

Variable	Value	Edit
catalog_tenant	f59bb9e0-da68-45a7-a5c9-bd4a6c74099a	
catalog_uri	predix-analytics-catalog-release.run.aws-usw02	
config_uri	predix-analytics-config-release.run.aws-usw02	
configurationId	value	
Content-Type	application/json	
deploymentRequestId	value	
execution_uri	predix-analytics-execution-release.run.aws-usw02	
jobId	value	
protocol	https	
runtime_tenant	497a5a2f-bf8f-430c-9f45-bf4d3d4c6365	
scheduler_uri	predix-scheduler-service-release.run.aws-usw02	

Cancel **Update**

- 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.

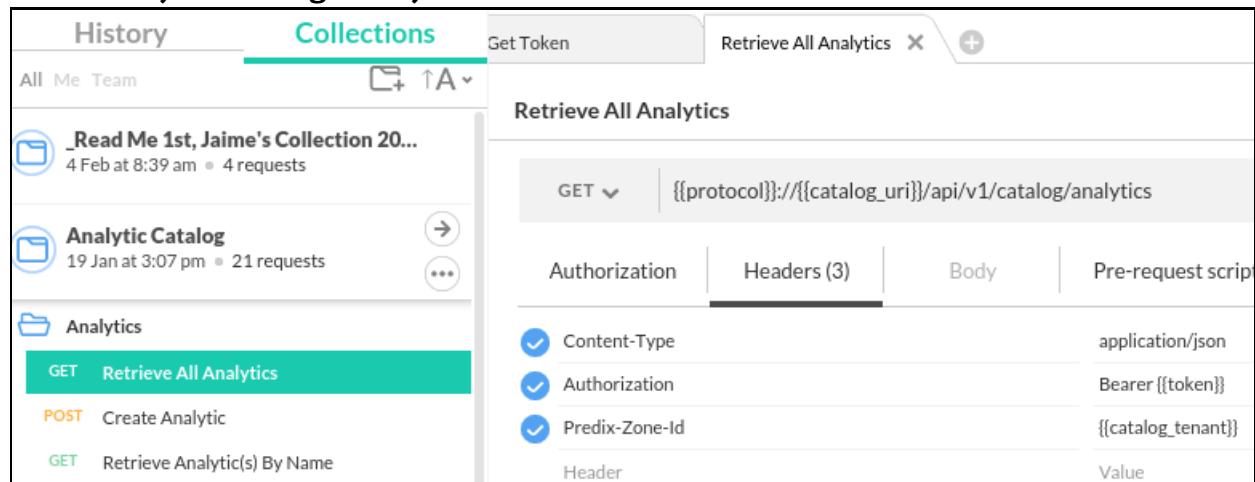
The following steps describe the process for adding a new analytic to the catalog.

- 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 selected. On the left, there's a sidebar with 'Read Me 1st, Jaime's Collection 20...' and 'Analytic Catalog'. Under 'Analytics', three requests are listed: 'GET Retrieve All Analytics' (highlighted in green), 'POST Create Analytic', and 'GET Retrieve Analytic(s) By Name'. The main panel shows the 'Retrieve All Analytics' request details:

- Method:** GET
- URL:** {{protocol}}://{{catalog_uri}}/api/v1/catalog/analytics
- Headers (3):**
 - Content-Type: application/json
 - Authorization: Bearer {{token}}
 - Predix-Zone-Id: {{catalog_tenant}}
- Body:** (None)
- Pre-request script:** (None)

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

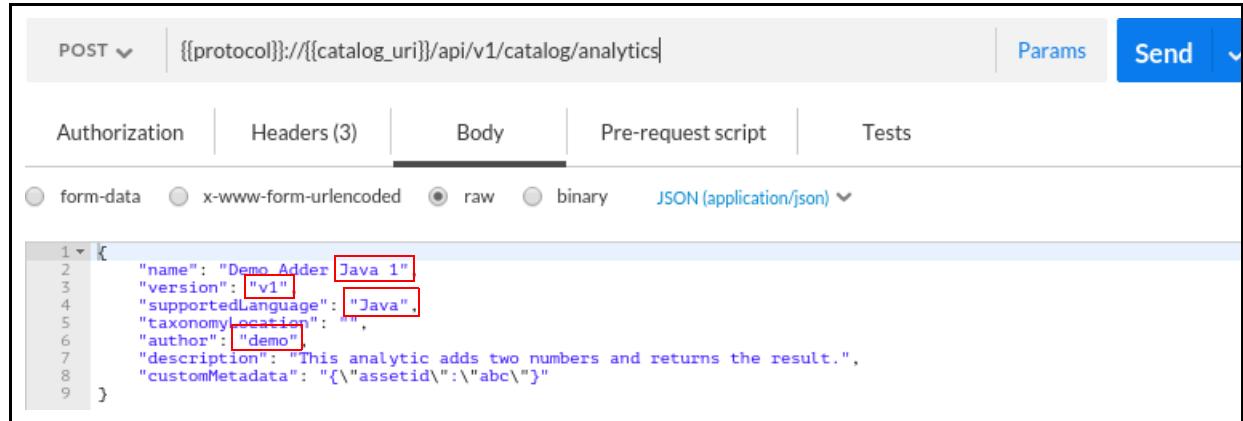
The screenshot shows the Postman application interface. On the left, there's a sidebar with tabs for History, Collections, and a pinned item 'Read Me 1st, Jaime's Collection 20...'. Below these are sections for 'Analytic Catalog' and 'Analytics'. Under 'Analytics', a 'GET Retrieve All Analytics' request is selected, highlighted with a teal bar. To the right, the main panel shows a 'Retrieve All Analytics' collection with a single 'GET {{[protocol]}://{{catalog_uri}}/api/v1/catalog/analytics}' request. The 'Authorization' tab is active, showing 'No Auth'. The 'Body' tab is also active, displaying a JSON response with an orange arrow pointing to the 'Status 200 OK' text. The response body is a JSON object:

```
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:



POST {{protocol}}://{{catalog_uri}}/api/v1/catalog/analytics

Params Send

Authorization Headers (3) Body Pre-request script Tests

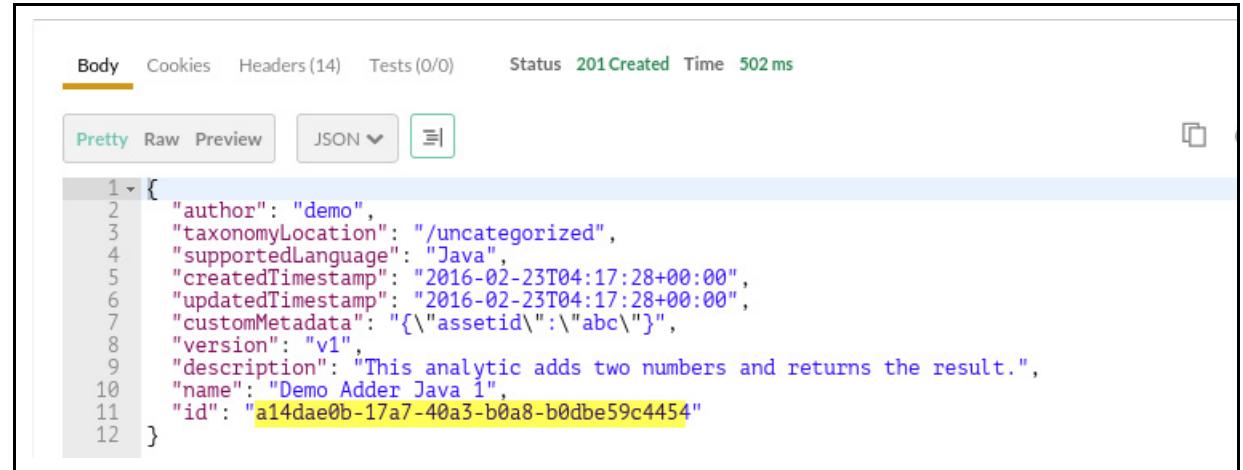
form-data x-www-form-urlencoded raw binary JSON (application/json)

```

1 <pre>{
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 }</pre>

```

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



Body Cookies Headers (14) Tests (0/0) Status 201 Created Time 502 ms

Pretty Raw Preview JSON

```

1 <pre>{
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 }</pre>

```

- 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.

The screenshot shows the Postman interface with the 'Analytics-00 Student' collection selected. A 'Create Artifact' tab is open, displaying a POST request to the URL `[[protocol]]://[[catalog_uri]]/api/v1/catalog/artifacts`. The 'Headers' tab is selected, showing four headers: Content-Type (multipart/form-data), Authorization (Bearer {{token}}), Accept (application/json), and Predix-Zone-Id ({{catalog_tenant}}). The 'Send' button is highlighted in blue at the top right of the request panel.

- 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 shows the Postman interface for creating an artifact. The top navigation bar includes 'Get Token', 'Retrieve All Analytics', 'Create Analytic' (disabled), 'Create Artifact' (disabled), and 'Analytics-00 Student'. The main title is 'Create Artifact'. The request method is set to 'POST' and the URL is `[[protocol]]://[[catalog_uri]]/api/v1/catalog/artifacts`. The 'Body' tab is selected, showing the following parameters:

Key	Type	Value
file	File	Choose Files demo-adder-java-1.0.0.jar
catalogEntryId	Text	<code>[[analyticId]]</code>
type	Text	Executable
description	Text	This analytic adds 2 numbers and provides the sum.
Key	Value	

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

- You should receive the following response:

The screenshot shows the Postman interface with the following details:

- Body** tab selected.
- Status: 201 Created
- Time: 2417 ms
- JSON response (Pretty):


```

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.

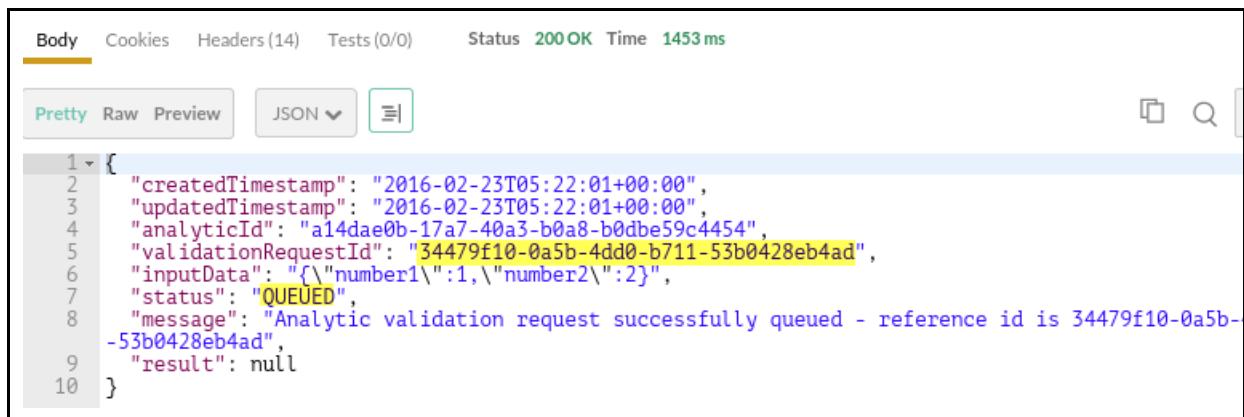
- Add a new Postman tab (window) and select **POST Validate Analytic** from the **Validation/Deployment/Execution** folder of your Postman collections.
- In the Body section, for <value1>, <value2>: Insert any two numbers you would like to add.

The screenshot shows the Postman interface for a POST request:

- Method: POST
- URL: `[[protocol]]://[[catalog_uri]]/api/v1/catalog/analytics/[[analyticId]]/validation`
- Headers (3) tab selected.
- Body tab selected.
- Body type: raw (JSON)
- Body content:


```
1 {"number1":1,"number2":2}
```

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



The screenshot shows the Postman interface with a successful API call. The status is 200 OK and the time taken is 1453 ms. The response body is displayed in Pretty JSON format, showing a single object with the following fields:

```
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 }
```

The field `validationRequestId` is highlighted in yellow.

- 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:

The screenshot shows a Postman interface with the following details:

- Method:** GET
- URL:** {{protocol}}://{{catalog_uri}}/api/v1/catalog/analytics/{{analyticId}}/validation/{{validationId}}
- Authorization:** No Auth
- Headers (5):** None listed.
- Body:** Contains JSON response data.
- Status:** 200 OK
- Time:** 916 ms

The JSON response body is:

```
1 | {
2 |   "createdTimestamp": "2016-02-23T05:22:01+00:00",
3 |   "updatedTimestamp": "2016-02-23T05:23:15+00:00",
4 |   "analyticId": "a14dae0b-17a7-40a3-b0a8-b0dbe59c4454",
5 |   "validationRequestId": "34479f10-0a5b-4dd0-b711-53b0428eb4ad",
6 |   "inputData": "{\"number1\":1,\"number2\":2}",
7 |   "status": "COMPLETED",
8 |   "message": "Analytic validation completed successfully.",
9 |   "result": "{\"result\":3}"
10 | }
```

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 229
- *Schedule Orchestration and Analytic Execution (Scheduling a Job)*, page 237



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)**

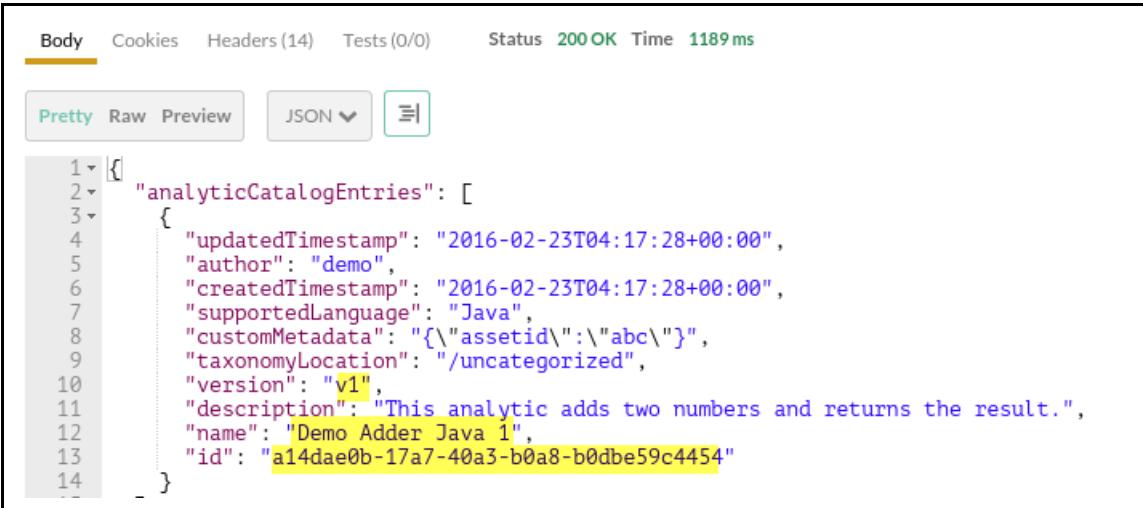
Is available for the student to simply run a “working” orchestration that relies on the environment variables: AnalyticId, AnalyticName, and AnalyticVersion.

- 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

- 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**.



The screenshot shows a Postman interface with the 'Body' tab selected. The status bar indicates a 200 OK response with a time of 1189ms. The JSON response is displayed in Pretty mode:

```

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

```

- 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

The screenshot shows a Postman collection interface with the following details:

- Method:** POST
- URL:** {{[protocol]}://{{execution_uri}}}/api/v1/execution
- Authorization:** (not explicitly shown in the screenshot, but part of the collection)
- Headers:** (4) (not explicitly shown in the screenshot, but part of the collection)
- Body:** (selected tab)
- Content Type:** JSON (application/json) (selected dropdown)
- Body Content (raw 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/20100422\">\n<process id=\"OrchestrationWithOneAnalytic\" isExecutable=\"true\">\n<startEvent id=\"sid-start-event\" name=\"\">\n<outgoing>sid-flow1</outgoing>\n</startEvent>\n<serviceTask completionQuantity=\"1\" id=\"sid-10001\" 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<incoming>sid-flow1</incoming>\n<outgoing>sid-flow2</outgoing>\n</serviceTask>\n</process>\n</definitions>\n"
5
6
7
8
9
10
11

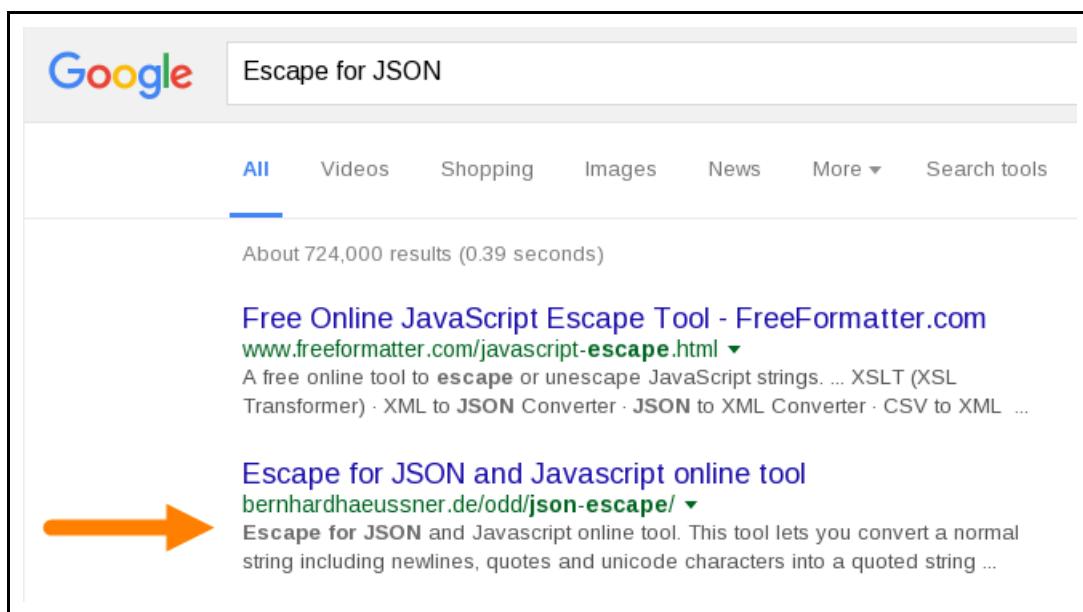
```
- Pre-request script:** (not explicitly shown in the screenshot, but part of the collection)
- Tests:** (not explicitly shown in the screenshot, but part of the collection)

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 (shown here)
-

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\"\n  id=\"sid-81430087-7a44-4be3-8517-914faf923256\"\n  targetNamespace=\"DSP-PM\""
```

- 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        "analytic": {
7          "StepId": "sid-10001",
8          "data": "[{"number1": 5, "number2": 24}]"
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:

Run (Using sample bpmnXML)

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

Body Cookies Headers (14) Tests (0/0) Status 200 OK Time 9258 ms

Pretty Raw Preview 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-11e5-8ab2-32",
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.ice.  
      "execution_uri": "https://predix-analytics-execution-release.run.aws-usw02-pr.ice.  
      "monitoring_uri": "https://predix-monitoring-service-release.run.aws-usw02-pr.ice.  
      "scheduler_uri": "https://predix-scheduler-service-release.run.aws-usw02-pr.ice.  
      "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": []  
  }  
]
```

Key	Value
protocol	https
runtime_tenant	f76daffd-5cb0-475b-83ab-78dabec4f526
scheduler_uri	predix-scheduler-service-release.run.aws-usw02-pr.ice. eyJhbGciOiJSUzI1NiJ9.eyJqdGkiOiJjIi t
token	eyJhbGciOiJSUzI1NiJ9.eyJqdGkiOiJjIi t
UAA Token URL	a97db685-e3f5-4f10-ad03-f82bee5a: 34479f10-0a5b-4dd0-b711-53b0428
validationrequestId	a97db685-e3f5-4f10-ad03-f82bee5a: 34479f10-0a5b-4dd0-b711-53b0428
X-Identity-Zone-Id	a97db685-e3f5-4f10-ad03-f82bee5a: 34479f10-0a5b-4dd0-b711-53b0428

- 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:

The screenshot shows the Postman interface with the following details:

- Method:** GET
- URL:** {{protocol}}://{{scheduler_uri}}/api/v1/scheduler/jobs
- Authorization:** No Auth
- Headers:** (2)
- Body:** (Pretty, Raw, Preview, JSON, etc.)
- Status:** 200 OK, Time: 397 ms
- Response Body:** (Pretty JSON view)

```
1  [
2   "jobs": [],
3   "currentPageSize": 0,
4   "maximumPageSize": 0,
5   "currentpageNumber": 0,
6   "totalElements": 0,
7   "totalPages": 0
8 }
```

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 shows the Postman interface with the 'Collections' tab selected. On the left, the 'Scheduler Service' collection is listed under 'Get Token'. On the right, the 'Create Job' request is being configured:

- Method:** POST
- URL:** {{protocol}}://{{scheduler_uri}}/api/v1/sche
- Authorization:** (radio button selected)
- Headers (3):** (radio button selected)
- Body:** (radio button selected)
- Raw JSON Body Content:**

```
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"  
22      }  
23    ]  
24  }  
25}
```



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

The screenshot shows a Postman API response. The status bar at the top indicates **Status 201 Created** and **Time 1677 ms**. Below the status bar, there are tabs for Body, Cookies, Headers (13), and Tests (0/0). The Body tab is selected and displays a JSON response. The response contains a cron schedule and an execution request. The cron schedule is defined with the following parameters: timeZoneId: "UTC", hours: "*", minutes: "0/2", seconds: "0", dayOfMonth: "*", dayOfWeek: "?", months: "*", years: "*". The execution request includes an inputData field with a complex XML schema. The XML schema defines a process with an activiti:activiti element, a targetNamespace="DSP-PM", typeLanguage="http://www.omg.org/spec/BPMN/20100524/MODEL", and a schemaLocation="http://www.omg.org/spec/BPMN/20100524/MODEL". The process has an id="OrchestrationWithOneAnalytic", an incoming event named="sid-start-event", and an outgoing flow named="sid-flow1".

```

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\" targetNamespace=\"DSP-PM\" typeLanguage=\"http://www.omg.org/spec/BPMN/20100524/MODEL\" schemaLocation=\"http://www.omg.org/spec/BPMN/20100524/MODEL\">\n      <process id=\"OrchestrationWithOneAnalytic\"\n        name=\"\">\n        <outgoing>sid-flow1</outgo

```

- 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://www.w3.org/2001/XMLSchema-instance\"\n  targetNamespace=\"DSP-PM\" typeLanguage=\"\n  xsi:schemaLocation=\"http://www.omg.org/spec/BPMN/20100524/MODEL http://www.omg.org/spec/activiti=http://activiti.org/bpmn\"\">\n    <process id=\"OrchestrationWithOneAnalytic\"\n      name=\"\">\n      <outgoing>sid-flow1</outgoing>\n      completionQuantity=\"1\"\n      id=\"sid-10001\"\n    is
```

- Scrolling down, you'll see the Job ID and status:

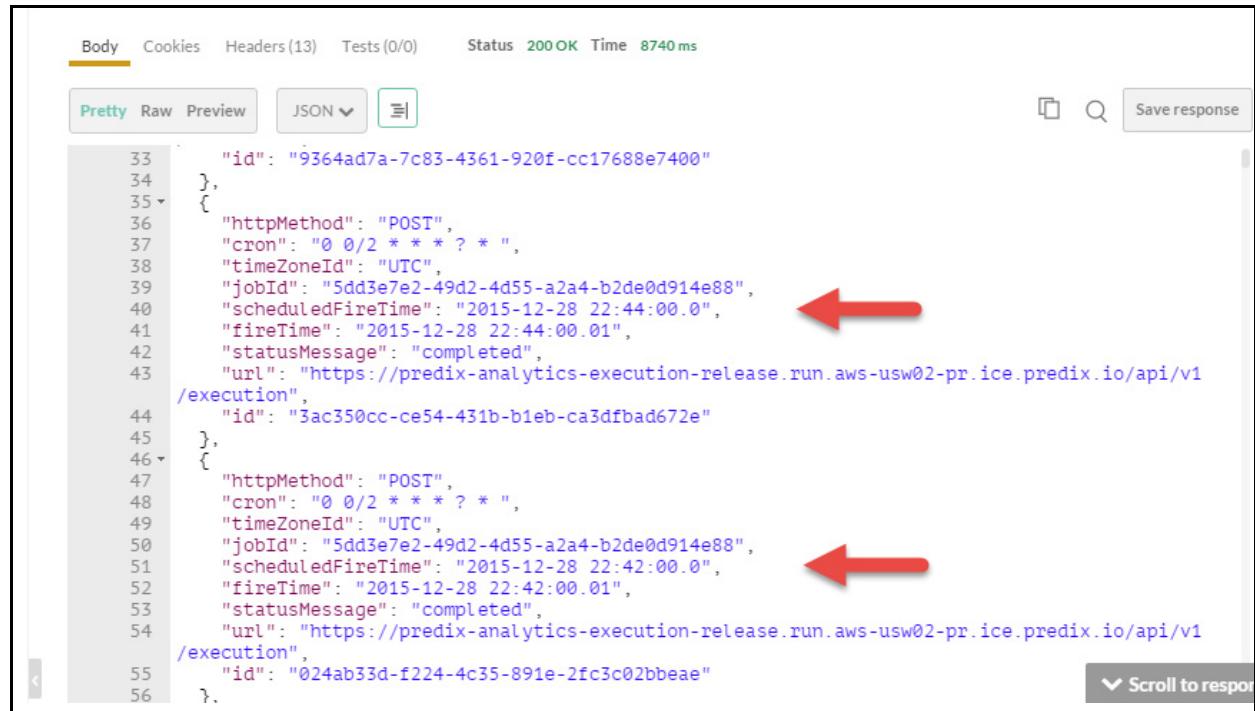
```
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
988
989
989
990
991
992
993
994
995
996
997
997
998
999
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1187
1187
1188
1189
1189
1190
1191
1192
1193
1194
1195
1195
1196
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1287
1287
1288
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1297
1297
1298
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1387
1388
1388
1389
1389
1390
1391
1392
1393
1394
1395
1396
1397
1397
1398
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1489
1490
1491
1492
1493
1494
1495
1496
1497
1497
1498
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1597
1597
1598
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1689
1690
1691
1692
1693
1694
1695
1696
1697
1697
1698
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1797
1798
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1897
1898
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1997
1998
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2097
2098
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2197
2198
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2278
2279
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2288
2289
2289
2290
2291
2292
2293
2294
2295
2296
2297
2297
2298
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2338
2339
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2348
2349
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2378
2379
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2388
2389
2389
2390
2391
2392
2393
2394
2395
2396
2397
2397
2398
2398
```

- 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:



The screenshot shows a Postman response for the 'GET Job History By Job ID' endpoint. The response body is a JSON array containing two job history entries. Each entry includes fields like 'id', 'httpMethod', 'cron', 'timeZoneId', 'jobId', 'scheduledFireTime', 'fireTime', 'statusMessage', and 'url'. Two red arrows point to the 'fireTime' field in both entries, which are '2015-12-28 22:44:00.01' and '2015-12-28 22:42:00.01'. The 'scheduledFireTime' field also shows these same values.

```

33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
      "id": "9364ad7a-7c83-4361-920f-cc17688e7400"
    },
    {
      "httpMethod": "POST",
      "cron": "0 0/2 * * * ? *",
      "timeZoneId": "UTC",
      "jobId": "5dd3e7e2-49d2-4d55-a2a4-b2de0d914e88",
      "scheduledFireTime": "2015-12-28 22:44:00.0",
      "fireTime": "2015-12-28 22:44:00.01",
      "statusMessage": "completed",
      "url": "https://predix-analytics-execution-release.run.aws-usw02-pr.ice.predix.io/api/v1
/execution",
      "id": "3ac350cc-ce54-431b-b1eb-ca3dfbad672e"
    },
    {
      "httpMethod": "POST",
      "cron": "0 0/2 * * * ? *",
      "timeZoneId": "UTC",
      "jobId": "5dd3e7e2-49d2-4d55-a2a4-b2de0d914e88",
      "scheduledFireTime": "2015-12-28 22:42:00.0",
      "fireTime": "2015-12-28 22:42:00.01",
      "statusMessage": "completed",
      "url": "https://predix-analytics-execution-release.run.aws-usw02-pr.ice.predix.io/api/v1
/execution",
      "id": "024ab33d-f224-4c35-891e-2fc3c02bbeae"
    }
  ]
  
```

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

Part IV: Using the Analytics User Interface

Overview

Before using the Analytics UI, you will need to update the OAuth2 Client and then create a user that can log in and use the UI.

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

You will be using environment variables from your catalog and runtime instances as well as the UAA command line interface.

Exercises

- [*Update OAuth2 Client and Create UI User, page 247*](#)
- [*Create Your Analytics UI Service Instance, page 249*](#)
- [*Using the Analytics UI, page 252*](#)



Exercise 1: Update OAuth2 Client and Create UI User

Steps

1. Using terminal, target your UAA service instance.

```
uaac target <uaa_uri>
```

2. Log in to UAA as the admin.

```
uaac token client get admin
```

- Enter the UAA admin password

3. Update authorized_grant_types.

```
uaac client update <client id> --authorized_grant_types  
"client_credentials authorization_code refresh_token"
```

Analysis: An OAuth2 client with authorization_code and refresh_token types is required when creating the Analytics UI service instance. This is the recommended practice. If your environment requires additional types, be sure to include them as well. When defining authorization_grant_types ensure you provide the authorization_code and refresh_token types.

4. Add catalog and runtime groups to UAA.

```
uaac group add  
analytics.zones.<my_analytics_catalog_instance_predix_zone_id>.user
```

```
uaac group add  
analytics.zones.<my_analytics_runtime_instance_predix_zone_id>.user
```

5. Create a user in UAA.

```
uaac user add <user_name> -p [password] --emails  
<user_name>@example.com
```

- Note this information as you will use this user account to login to the Analytics UI.

6. Add the user to the Catalog and Runtime groups.

```
uaac member add analytics.zones.  
<my_analytics_catalog_instance_predix_zone_id>.user <user_name>
```

```
uaac member add analytics.zones.  
<my_analytics_runtime_instance_predix_zone_id>.user <user_name>
```



Exercise 2: Create Your Analytics UI Service Instance

Overview

There are two ways you can create an Analytics UI service instance: using cloud foundry (CF) CLI or using the Predix.io web interface.

Steps (using Cloud Foundry) - Preferred Method

1. Using Cloud Foundry, use the following command at the terminal.

```
cf create-service predix-analytics-ui <plan> <my_service_instance>
-c
'{"uaaHostUri": "<my_uaa_host_uri>","clientId": "<my_client_id>","clientSecret": "<my_client_secret>","domainPrefix": "<my_domain_prefix>","catalogPredixZoneId": "<my_analytics_catalog_instance_predix_zone_id>","runtimePredixZoneId": "<my_analytics_runtime_instance_predix_zone_id>"}'
```

Provide the following information for your UI service instance:

- **<plan>** is the plan (Free) associated with the UI marketplace service.
- **<my_service_instance>** is the name you of the UI service instance you create.
- **<my_uaa_host_uri>** is the host URI obtained from the VCAP_SERVICES environment variable of your UAA service. instance. For example, <https://13fa0384-9e2a-48e2-9d06-2c95a1f4f5ea.predix-uaa.grc-apps.svc.ice.ge.com>.
- **<my_client_id>** is <your client id> to be used with your UAA service instance.
- **<my_client_secret>** is the client secret <your client secret> to be used with your UAA service instance.
- **<my_domain_prefix>** is the domain prefix to be used to access your Analytics UI in a browser. It is case insensitive. For example, assume the base Predix Analytics User

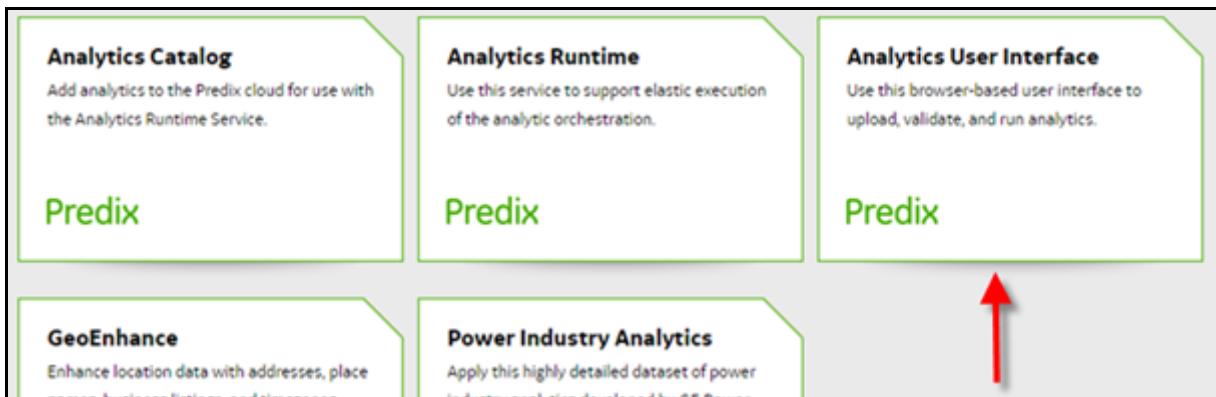
Interface URL is predix-analytics-ui.run.aws-usw02-pr.ice.predix.io. If you provide the domain name "abc" at creation, then your user interface instance URL is **abc**.predix-analytics-ui.run.aws-usw02-pr.ice.predix.io.

- <**my_analytics_catalog_instance_predix_zone_id**> is the service instance GUID of your Analytics Catalog instance (cf service <my-analytics-catalog-service> --guid)
- <**my_analytics_runtime_instance_predix_zone_id**> is the service instance GUID of your Analytics Runtime instance (cf service <my-analytics-runtime-service> --guid)

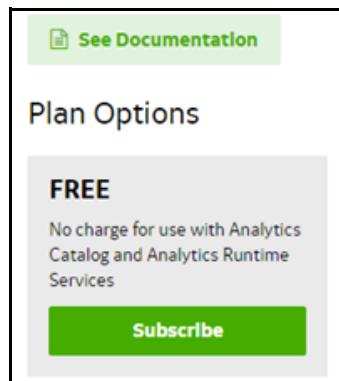
ALTERNATE OPTION: see Appendix Steps (using Predix.io)

After logging into your Predix.io account, and selecting your space, you may use the following steps to create your Analytics UI service instance.

- Click on catalog and select the Analytics UI "tile"



- Click on the **Subscribe** button



- Select your org, space, and UAA instance for your UI service

The image shows a subscription configuration dialog box. It has two dropdown menus: 'Org*' set to 'Predix-Training' and 'Space*' set to 'Training5'. Below these is a note: 'The service you are trying to subscribe to requires User Account & Authentication. [Learn more](#)'. There is a section for 'User Account & Authentication (UAA)*' with options: 'Select existing UAA' (with a dropdown arrow), 'or', and a green 'Subscribe to New UAA' button with a plus sign icon. Below this is a link 'Use other UAA'.

- Click the **create service** button
 - ◆ Enter the same parameters used for the `cf create-service` command
- Note:** There is no need to bind the UI service to an application.

Exercise 3: Using the Analytics UI

Overview

Once the UAA user and the UI service instance is created you will be able to use the UI to work with the catalog, uploading, deploying and testing your analytic applications.

Steps

1. Get the URL for your Analytics UI.

```
cf service <service_instance_name>
```

The output similar to the following is returned:

- Service: predix-analytics-ui
- Plan: <plan>
- Description: Apply a user-interface to configure and manage analytic assets using Predix Analytics.
- Documentation url:
- Dashboard: <**analytics-ui-url**>The Dashboard: field contains the URL

2. Launch the Analytics UI console.

- Open a **Chrome** browser session and paste the <**analytics-ui-url**> into the address
Note: Do not use Firefox - there are security issues with connectivity.

3. Log in with the user credentials created from exercise 1.

- Provide the username and password for the user account



4. Add an analytic.

- Click on New Analytic
- Select a Java, Matlab, or Python executable file
- Provide meta-data

Note: The maximum file size for upload is 250MB. The maximum expanded analytic file size (including directories and files) is 500MB.

5. Deploy and Test the Analytic App

- Select the Deploy and Test tab
- Provide JSON input for test, for example:

```
{"number1":1,"number2":2}
```

- Click the Submit button
- Click on the status tab to view status and result.

Note: On deployment, it may take a few minutes to set up and initialize environment for the app, setting up VM's, dependent files, setting up routes.

6. Testing the Analytic app with other input

- Select the Test tab
- Provide JSON input for test, for example:

```
{"number1":1,"number2":2}
```

- Click the Submit button

Attachments:

File Name	Type	Description
demo-adder-py.zip	Executable	python file adds 2 numbers

Description:
python adder with 2 numbers

Status Deploy and Test **Test**

Analytic Input

```
{"number1":1,"number2":2}
```

Analytic Output

```
{
  "result":3
}
```

Reset **Submit for Test**

7. Downloading an Analytic app's artifact

You can download an artifact previously uploaded with an analytic from the Analytic Detail page.

- To download, click the name of the artifact in the Attachments table. For example, in the following figure the artifact is named demoAdder.zip.

Taxonomy Location: /predix-demo-analytics
Author: predix-analytics
Language: Python
Attachments:

File Name	Type	Description
demoAdder.zip	Executable	

Description:
given 2 long numbers ('number1' and 'number2'), returns added result

Status Deploy and Test Test

There is no deployment in progress. To deploy this analytic, click the Deploy and Test tab to enter inputs and submit. If this analytic has already been deployed, click the Test tab to execute it by providing the necessary inputs.

8. Deleting an Analytic.

- Open the Analytic Catalog page by clicking the Analytics tab.
- Click the Delete icon in the row corresponding to the analytic to be deleted.
- When prompted, accept the change to delete the analytic from the catalog.

Version	Taxonomy Location	Author	Description	
v1	/predix-demo-analytics	DataScience+PredixAnalytics	given data points, return anom...	
v1	/predix-demo-analytics	predix-analytics	given 2 long numbers ('number...	
v1	/predix-demo-analytics	predix-analytics	given 2 long numbers ('number...	

Lab Summary

In this exercise you learned:

- How to use the UI's current features and functionality
- How to set up service instance configurations required
- How to use the UI to upload an analytic, deploy the analytic, and test the analytic
-

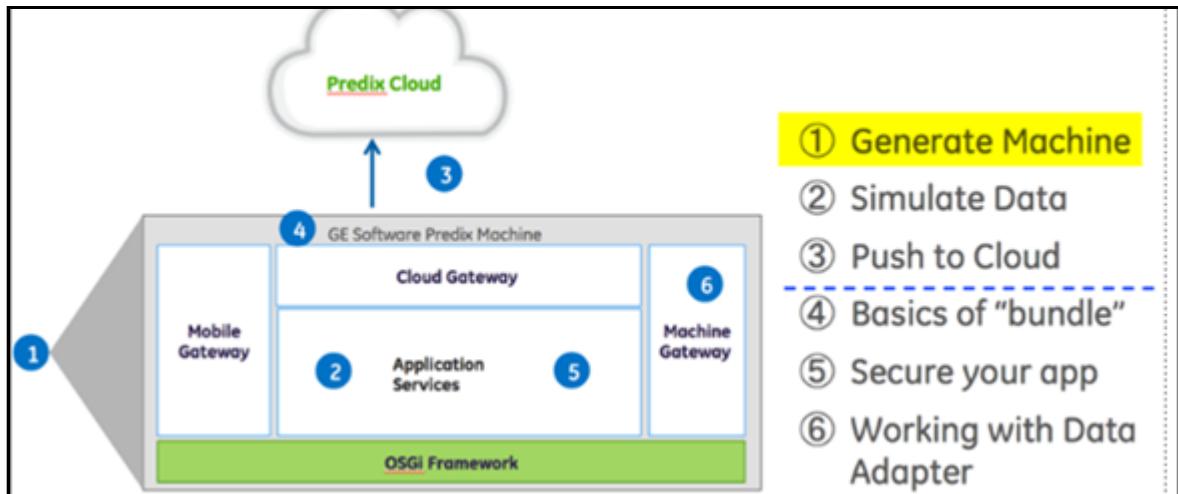


Lab 9: Connecting Machines to Predix Cloud

Learning Objectives

The overall 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



Lab Exercises

- Examine Predix SDK on page 260
- Test Administrator Command Line Options, page 267
- Using the Web Console, page 269
- Import HTTP Data Service Sample from Predix SDK, page 283

Directions

Complete the exercises that follow.

Exercise 1: Examine Predix SDK

Overview

The Predix SDK contains all the scripts and documentation to generate an OSGi container as well as bundles that run inside the container.

Steps

1. Locate the Predix SDK .

- On the DevBox Desktop click the **Places** menu and select **PredixApps** from the list
- Open the **PredixApps/training_labs/ machine/predixmachinesdk-16.2.0** folder and examine the files

/predix/PredixApps/training_labs/machine/predixmachinesdk-16.2.0			
	Name	Size	Type
×	docs	2 items	folder
	apidocs.zip	984.2 KB	Zip archive
	Container_docs.zip	52.9 MB	Zip archive
	eclipse-plugins	4 items	folder
	license	6 items	folder
	samples	2 items	folder
	sample-apps.zip	1.2 MB	Zip archive
	sample-cloud-apps.zip	23.2 MB	Zip archive
	utilities	2 items	folder
	containers	4 items	folder
	GenerateContainers	3.1 KB	shell script
	GenerateContainers.bat	3.6 KB	plain text document
	pom.xml	6.6 KB	XML document
	README.txt	1.9 KB	plain text document
	README.txt	171 bytes	plain text document
	InstallationGuide.pdf	815.0 KB	PDF document

/docs

- Javadoc APIs for Predix machine services
- Prosynt docs for OSGi containers

/eclipse-plugins

- Import into your Eclipse-based tool (STS)

/samples

- Predix machine samples
- Predix cloud sample for Websocket data service

/utilities/containers

- Machine containers and scripts

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/predixmachinesdk-16.2.0
/docs`
- Unzip the documentation file
`unzip apidocs.zip -d apidocs`
- Go to the `machine/predixmachinesdk-16.2.0/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`

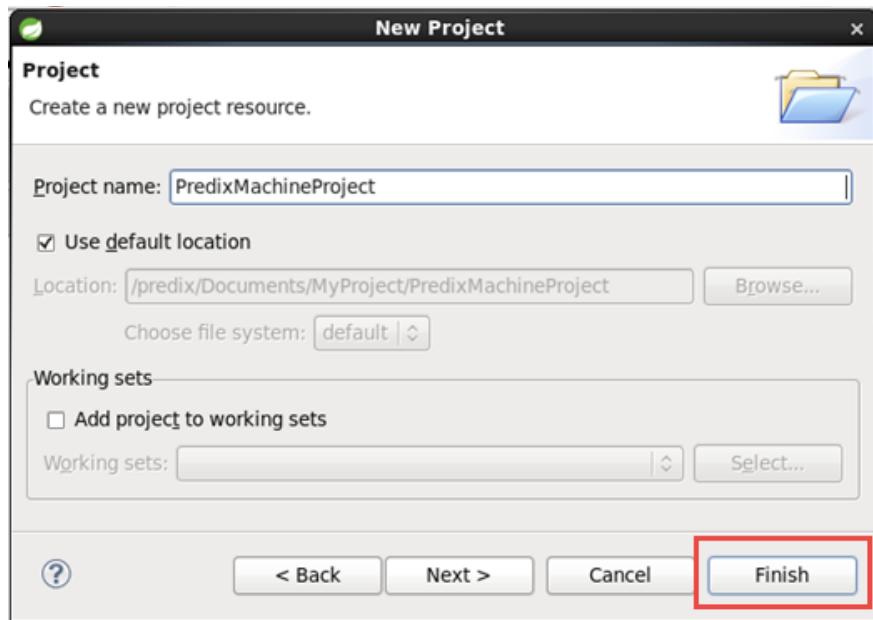
3. Create a debug Machine container.

- 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.

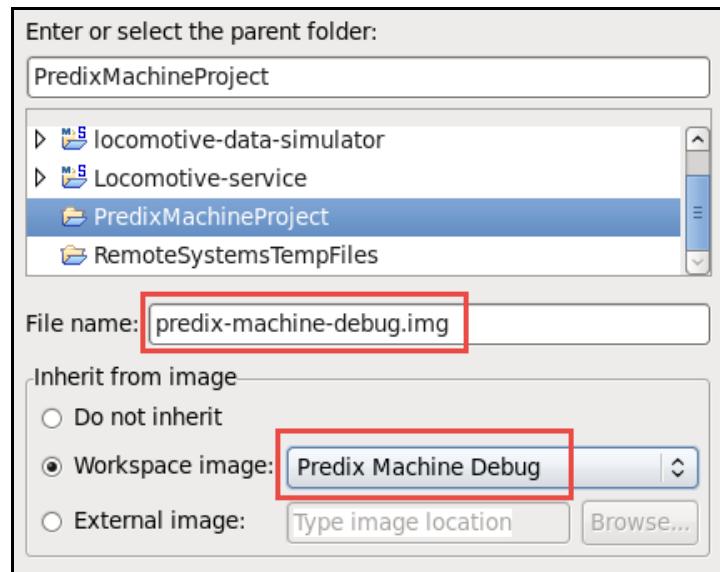
- Click **File > New > Other... > General > Project**
 - ◆ Click **Next**
- Project name: **PredixMachineProject**



- 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**
 - ◆ 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



- 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

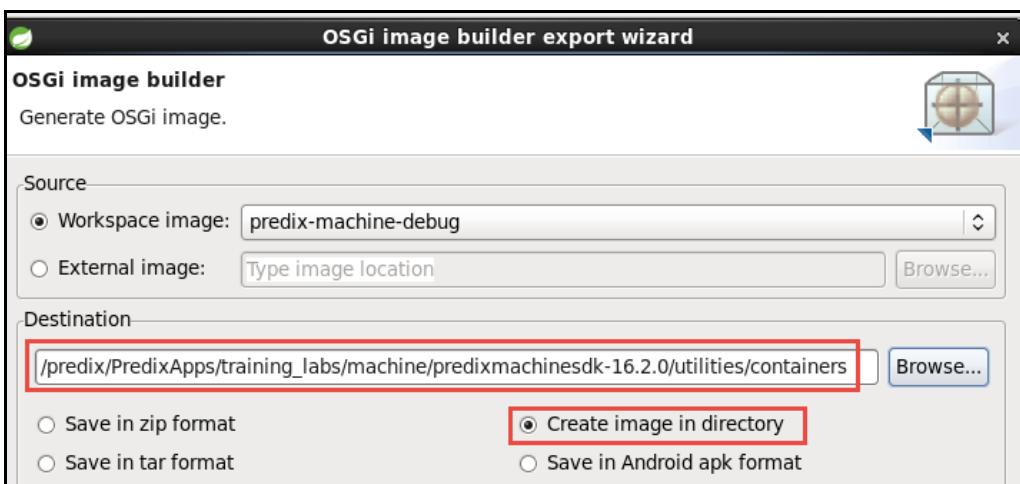


7. Export the container image.

- 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/predixmachinesdk-16.2.0/utilities/containers
 - Click **OK**

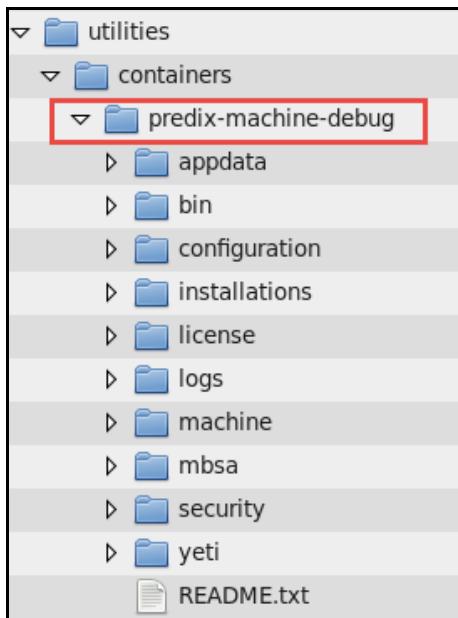


- Click **Finish**

A new folder *predix-machine-debug* is created with all the container files in the *predixmachinesdk-16.2.0/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: location to which you just exported)
- It has the following structure:



9. Start the debug container.

- In the Terminal, go to the /bin directory of predix-machine-debug

```
cd
```

```
predixmachinesdk-16.2.0/utilities/containers/predix-machine-debug/
machine/bin/predix
```

- Run the start script using the command

```
./start_container.sh clean
```

The first time you start machine it will take a few minutes. Look for the *Server is started* message in the terminal window (ignore MQTT broker messages for now)

```
2016-05-24 13:05:07,822[FW Buff Logger]|INFO|com.prosyst.mbs.system.bundle
com.prosyst.mbs.system.bundle-1.0.0|Bundle with id #116 com.ge.dspmicro.webc
e-techconsole, 16.2.0 was started.
#0    INFO    > Bundle with id #116 com.ge.dspmicro.webconsole-techconsole,
.0 was started.
#0    INFO    > [PlatformState] State Change from STARTING to ACTIVE
2016-05-24 13:05:07,966[FW Buff Logger]|INFO|com.prosyst.mbs.system.bundle|
n.prosyst.mbs.system.bundle-1.0.0|[PlatformState] State Change from STARTIN
ACTIVE
Server is started.
```



Exercise 2: Test Administrator Command Line Options

Overview

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 (partial list shown here)

<code>fw>\$ls</code>		
ID	State	Jar Name
<hr/>		
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-8.0.9
4	ACTIVE	com.prosyst.mbs.core.threads-8.0.9
5	ACTIVE	com.prosyst.mbs.core.db-8.0.9
6	ACTIVE	com.prosyst.mbs.osgi.cm.bundle-8.0.9
7	ACTIVE	com.prosyst.mbs.osgi.api-8.0.9
8	ACTIVE	com.prosyst.mbs.osgi.eventadmin-8.0.10
9	ACTIVE	com.prosyst.mbs.osgi.metatype.bundle-8.0.10
10	ACTIVE	com.prosyst.mbs.util.api-8.0.9
11	RESOLVED	com.prosyst.mbs.osgi.compendium.extension-8
12	ACTIVE	jersey-all-2.22.2
13	RESOLVED	com.ge.dspmicro.validatoroute-api-16.2.0
14	RESOLVED	commons-codec-1.10
15	RESOLVED	commons-lang-2.6
16	RESOLVED	com.ge.dspmicro.river-api-16.2.0

- 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 <bundle Id></code>	Displays bundle details
<code>manifest <bundle Id></code>	Displays the manifest of a bundle
<code>services <bundle Id></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 <bundle Id></code>	Start the bundle
<code>stop <bundle Id></code>	Stop the bundle
<code>restart <bundle Id></code>	Restart the bundle
<code>e or exit</code>	Shutdown machine

- Leave the container running to do the next exercise



Exercise 3: Using the Web Console

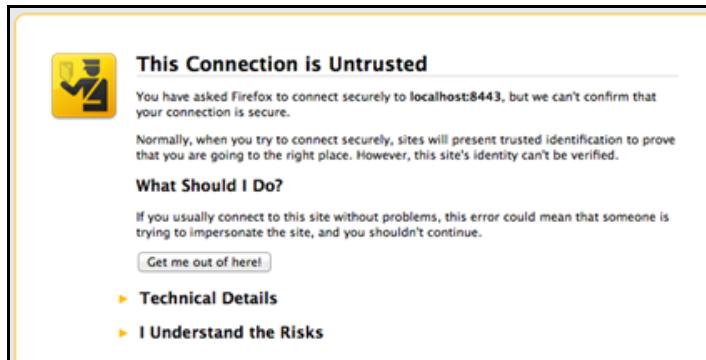
Overview

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 Firebox browser navigate to: <https://localhost:8443/system/console>
 - ◆ The following dialog box is displayed



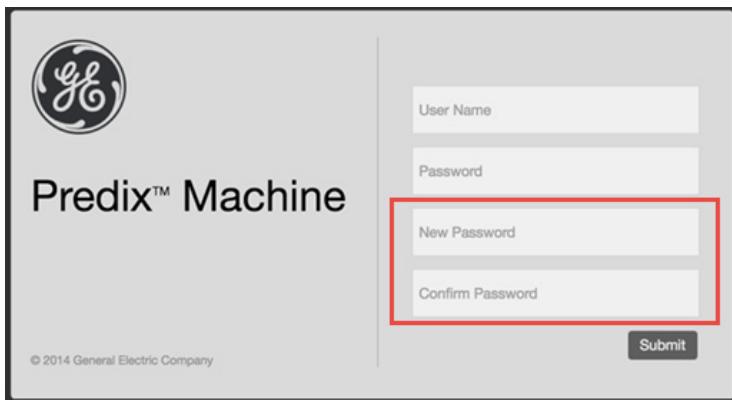
Note: Machine services use a self-signed certificate by default. You can safely add an exception to allow your browser to connect to Machine.

- ◆ Click **I Understand the Risks** and then select **Add Exception**
- ◆ Click **Confirm Security Exception** to complete the process

- Login using the following credentials and click **Sign In**
 - ◆ Username: **predix**
 - ◆ Password: **predix2machine**
- Click **Sign In**



- A second login screen is displayed; here, you are asked to change your password



- ◆ Username: **predix**
- ◆ Password: **predix2machine**
- ◆ New Password: **Predix@2016a**
- ◆ Confirm Password: **Predix@2016a**
- Clicking **Submit** logs you out and displays the login screen again
 - ◆ Enter your username (**predix**) and new password (**Predix@2016a**)

- The Web Console is displayed

Id	Name	Version	Category	Status	Actions
0	System Bundle (com.prosyt.mbs.system_bundle)	8.0.0	system	Active	
82	Apache HttpClient OSGi bundle (org.apache.httpcomponents.httpclient)	4.4.1	system	Active	

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

2. Try executing the following commands from the various menus in the web console.

Id	Name	Version	Category	Status	Actions
0	System Bundle (com.prosyt.mbs.system_bundle)	8.0.0	system	Active	
82	Apache HttpClient OSGi bundle (org.apache.httpcomponents.httpclient)	4.4.1	system	Active	

Menu > Command	Action
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

3. 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.

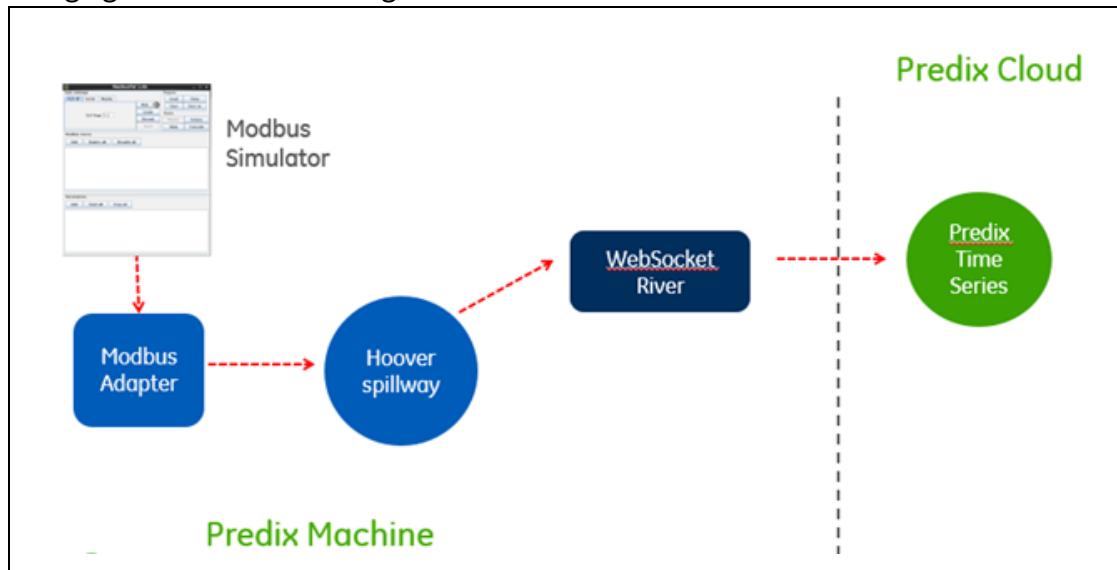


Part II: Sending Data to the Cloud using HTTP River

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



Lab Exercises

- *Part II: Sending Data to the Cloud using HTTP River, page 273*
- *Configure Modbus Adapter in Machine Services, page 278*
- *Configure a Spillway for HTTP, page 280*
- *Import HTTP Data Service Sample from Predix SDK, page 283*
- *Import HTTP Data Service Sample from Predix SDK, page 283*
- *Configure HTTP River, page 289*

Exercise 1: Download and Start Modbus PLC Simulator

Overview

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.

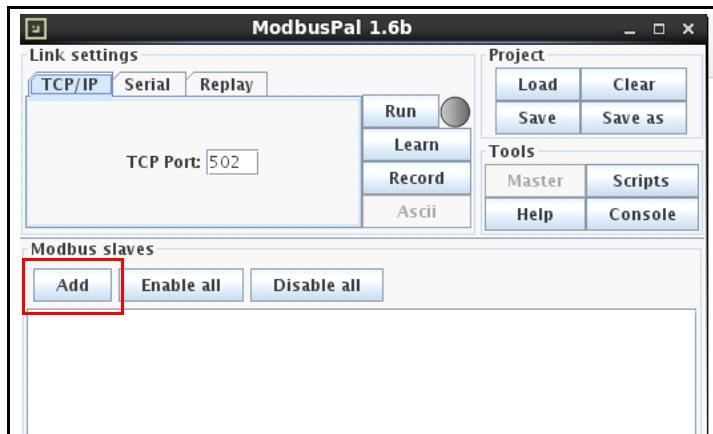
- From a Terminal navigate to *~/PredixApps/training_labs/machine*
- Run the command to start the simulator

```
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



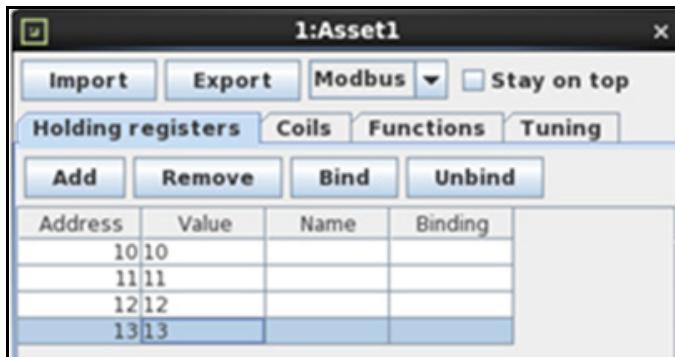
- Configure registers (sensors) for this asset
 - For Asset1, click on the edit button (the eye icon)
 - In the *Holding Registers* tab, click on the **Add** button



- Reset the number of registers
 - ◆ From: 10 (type this in)
 - ◆ To: 13 (type this in)
 - Click **Add**



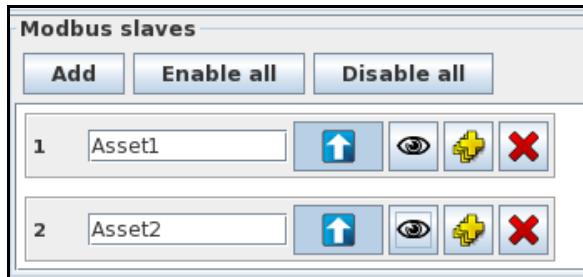
- Set the values for the registers to random numbers. The configuration should look similar to the picture below:



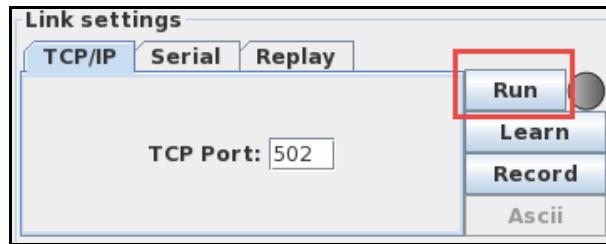
- Close the editor window by clicking on the X in the right corner

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



Exercise 2: Configure Modbus Adapter in Machine Services

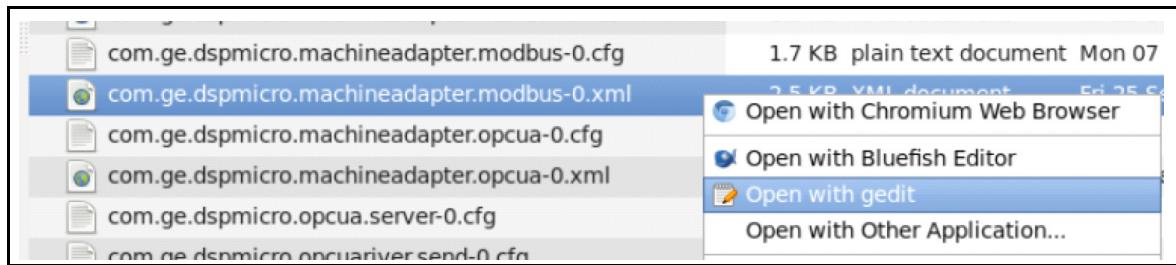
Overview

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/predixmachinesdk-16.2.0/`
`utilities/containers/predix-machine-debug/configuration/machine`
 - ◆ The configuration properties for the Modbus bundle can be found in
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>
    </dataSubscriptionConfig>
```

- 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.

Exercise 3: Configure a Spillway for HTTP

Overview

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
`predix-machine-debug/configuration/machine/com.ge.dspmicro.hoover.spillway-0.config`
 - ◆ Ensure the `com.ge.dspmicro.hoover.spillway.destination` property is **Http Sender Service**

```
67 # [Required] Destination Data River name to where the data will be sent.  
68 # Change to the Data River by replacing the value with: Sender Service  
69 com.ge.dspmicro.hoover.spillway.destination="Http Sender Service"
```

- Save the file

2. Collect “data forward” data.

- In the Terminal, open a new tab (*File > Open Tab*)
- Navigate to the `/machine/bin/predix` directory of the predix-machine-debug folder
 - `cd predixmachinesdk-16.2.0/utilities/containers/predix-machine-debug/machine/bin/predix`
- Start the machine container with the start script
 - `./start_container.sh clean`
 - ◆ Look for the message “Server is started”

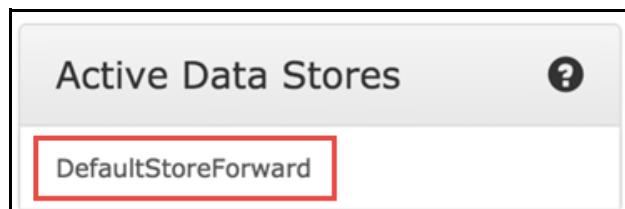


- The following messages are expected at this step (you have not completed configuring your machine)

```
2016-05-25 10:43:37,383[Thread-27] |ERROR|
com.ge.dsmpmicro.storeforward.impl.StoreForwardImpl|101-com.ge.dsmpmicro.storeforward-
impl-16.2.0|Unexpected exception from the forward callback in datastore
DefaultStoreForward. Data with id 1 will be re-sent.
com.ge.dsmpmicro.river.api.RiverException: java.lang.IllegalStateException: No Predix
cloud configuration is available. Verify the configuration of the client identity service.
    at com.ge.dsmpmicro.httpriver.send.impl.HttpRiverSendImpl.send
(HTTPRiverSendImpl.java:307)
    at com.ge.dsmpmicro.httpriver.send.impl.HttpRiverSendImpl.transfer
(HTTPRiverSendImpl.java:734)
    at com.ge.dsmpmicro.hoover.impl.spillway.SpillwayImpl.forward
(SpillwayImpl.java:703)
    at com.ge.dsmpmicro.storeforward.impl.StoreForwardImpl$ForwardThread.run
(StoreForwardImpl.java:766) [101:com.ge.dsmpmicro.storeforward-impl:16.2.0]
Caused by: java.lang.IllegalStateException: No Predix cloud configuration is available.
Verify the configuration of the client identity service.
    ... 4 more
2016-05-25 10:43:37,461[Component Resolve Thread (Bundle 38)] |WARN|
com.ge.dsmpmicro.machineadapter.opcua.impl.OPCUAMachineAdapterImpl|111-
com.ge.dsmpmicro.machineadapter-opcua-16.2.0|OPC-UA Machine Adapter is not configured. It
will not be used
```

3. View stored data in the Web Console.

- In a browser, open Admin Web Console (<https://localhost:8443/system/console>)
 - ◆ Username: **predix**
 - ◆ Password: **Predix@2016a**
- From the **Predix** menu, select **Store and Forward**
 - ◆ Select **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.

Note: You will see errors in the terminal window until the data river is configured

A screenshot of the Admin Web Console. On the left, under "Active Data Stores", the "DefaultStoreForward" store is listed. On the right, under "Store Contents", a table shows the 10 oldest items in DefaultStoreForward, with 7 items total. The table has columns for "ID" and "Timestamp".

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

Exercise 4: Import HTTP Data Service Sample from Predix SDK

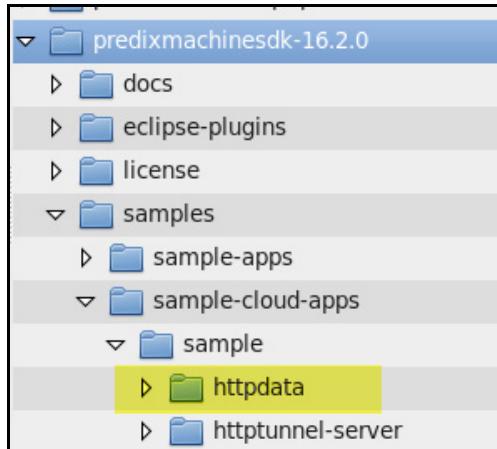
Overview

In this exercise 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 > Maven > Existing Maven Projects**, click **Next**
- Browse to the *predixmachinesdk-16.2.0/samples/sample-cloud-apps/sample/httpdata* folder



- ◆ Click **OK**
- ◆ Click **Finish**
- The HTTP Data Service sample imports as **predixmachine-http-data** project

2. Update the manifest file.

- In the **predixmachine-http-data** project open the **manifest.yml** file and update the following properties:

```
name: <your-name>-httpdata
path: ./target/predixmachine-http-data-<version>-SNAPSHOT.jar
  - Update the <version> with the version of the .jar file in the /target directory
services:
  - <your-name>-httpdata-postgres
  - <your-name>-uaa
```

- ◆ Replace <your-name> with the name used to create your UAA and postgres instances
 - For example:

```
name: student5-httpdata
path: ./target/predixmachine-http-data-16.1.0-SNAPSHOT.jar
services:
  - student5-httpdata-postgres
  - student5-uaa
```

- Save and close the manifest file
- In Eclipse, right-click on the project name and select **Run As > Maven install**
 - ◆ Verify the BUILD SUCCESS message is displayed

```
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ predixmachine-http-data ---
[INFO] Installing /predix/Documents/PredixApps/training_labs/machine/predixmachinesdk-16.2.0/samples/
[INFO] Installing /predix/Documents/PredixApps/training_labs/machine/predixmachinesdk-16.2.0/samples/
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 4.139s
[INFO] Finished at: Fri May 27 11:00:01 PDT 2016
[INFO] Final Memory: 23M/156M
[INFO] -----
```



3. Push the project to the cloud.

- In the Terminal, navigate to the `samples/sample-cloud-apps/httpdata` folder (where the `manifest.yml` file is)

```
/predix/PredixApps/training_labs/machine/predixmachinesdk-16.2.0
/samples/sample-cloud-apps/sample/httpdata
```

- Push the app to the cloud and make sure it starts

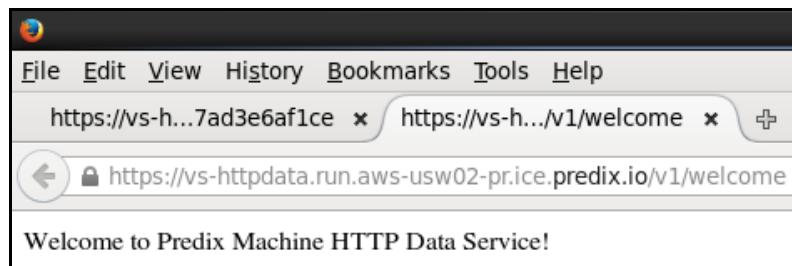
```
cf push
cf apps
```

```
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

      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]$
```

- Open a browser window and enter the application url followed by `/v1/welcome`
`https://<app_url>.run.aws-usw02-pr.ice.predix.io/v1/welcome`
 - You should see "Welcome to Predix Machine HTTP Data Service" indicating your service has been deployed successfully



4. Update your UAA app client with authority for Machines.

- Log into your UAA instance as the admin client
- In a Terminal run the commands

```
uaac target <paste UAA uri here>
uaac token client get admin -s <your admin secret>
```

```
[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-usw0
2-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-usw0
2-pr.ice.predix.io
Context: admin, from client admin
```

- Retrieve the existing authorities for your app client
 - ◆ Run the command: **uaac clients**
- Copy the existing authorities for your app client to a file

```
authorities: clients.read acs.policies.write clients.secret idps.write uaa.resource
timeseries.zones.8339eb48-4a17-44d7-8db6-1a23066ebbee.ingest acs.attributes.writ
timeseries.zones.8339eb48-4a17-44d7-8db6-1a23066ebbee.user clients.admin scim.re
predix-asset.zones.2aa65441-0eb7-4f7f-9bde-5b821941f3a6.user zones.bdf0839c-9aa6
timeseries.zones.8339eb48-4a17-44d7-8db6-1a23066ebbee.query acs.attributes.read
views.zones.26a77004-c3b1-4afe-b49e-74b6f1dfffd3b.user idps.read scim.write
```

- ◆ Run the command: **uaac client update -i** and press **Enter**
- ◆ You are prompted to fill in the following parameters (enter the **bolded** values)

```
client name: <your client id>
client secret: <your client secret>
verify client secret: <your client secret>
scope [list]: openid
authorized grant types [list]: authorization_code password
client_credentials refresh_token
```



```
authorities [list]: <your existing authorities from the file>
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
```

- Run the command: **uaac clients**

- ◆ Verify your app client has the new scope, grant_types and uaa.resource authorities:

```
admin
  scope: uaa.none
  resource_ids: none
  authorized_grant_types: client_credentials
  autoapprove:
  action: none
  authorities: clients.read acs.policies.write clients.secret idps.write uaa.resource
    predix-asset.zones.2aa65441-0eb7-4f7f-9bde-5b821941f3a6.user zones.bdf0839c-9aa
    scim.write
  lastmodified: 1465330848776
client2
  scope: openid predix-asset.zones.2aa65441-0eb7-4f7f-9bde-5b821941f3a6.user scim.me
    timeseries.zones.8339eb48-4a17-44d7-8db6-1a23066ebbee.query timeseries.zones.83
  resource_ids: none
  authorized_grant_types: authorization_code client_credentials password refresh_toke
  autoapprove: scim.me openid
  action: none
  authorities: clients.read acs.policies.write clients.secret idps.write uaa.resource
    acs.attributes.write timeseries.zones.8339eb48-4a17-44d7-8db6-1a23066ebbee.user
    predix-asset.zones.2aa65441-0eb7-4f7f-9bde-5b821941f3a6.user zones.bdf0839c-9aa
    timeseries.zones.8339eb48-4a17-44d7-8db6-1a23066ebbee.query acs.attributes.read
    idps.read scim.write
  lastmodified: 1465592277435
```

5. 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/token",
    "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 /configuration/machine/**com.ge.dsppmicro.predixcloud.identity.config** file

- ◆ Make the following changes to the file

```
com.ge.dsppmicro.predixcloud.identity.uaa.token.url="<UAA issuerid>"
com.ge.dsppmicro.predixcloud.identity.uaa.clientid="<your clientid>"
com.ge.dsppmicro.predixcloud.identity.uaa.clientsecret="<your clientid>"
```

```
12
13 #
14 # Predix Cloud UAA client credentials
15 #
16 com.ge.dsppmicro.predixcloud.identity.uaa.clientid="client2"
17 com.ge.dsppmicro.predixcloud.identity.uaa.clientsecret="myadminsecret"
18 com.ge.dsppmicro.predixcloud.identity.uaa.clientsecret.encrypted=""
19
```

- Save and close the file



Exercise 5: Configure HTTP River

Overview

Configure the HTTP Machine Services and start the flow of data.

Steps

1. Configure the HTTP River.

- Get the **application route** by running the command `cf apps` in the Terminal

```
79B:~ 502439379$ cf apps | grep http  
data          started    1/1   1G   1G      annams-httpdata.run.aws-usw02-pr.ice.predix.io
```

- Edit the `/configuration/machine/com.ge.dspmicro.httpriver.send-0.config` file

- ◆ Add the application route

```
com.ge.dspmicro.httpriver.send.destination.host=<application route>
```

Example:

```
49  
50 # [Required] A friendly and unique name of the HTTP River.  
51 com.ge.dspmicro.httpriver.send.river.name="Http Sender Service"  
52  
53 # [Required] Route to the river receive application. (e.g. myapp.mycloud.com)  
54 com.ge.dspmicro.httpriver.send.destination.host="ams-  
httpdata.run.aws-usw02-pr.ice.predix.io"  
55
```

- Save the file

2. Restart the Predix Machine.

- In the Terminal navigate to the **/machine/bin/predix** directory of predix-machine-debug
 - ◆ Run the command **./start_container.sh clean**
- You will see a success message repeated for each transaction every few seconds

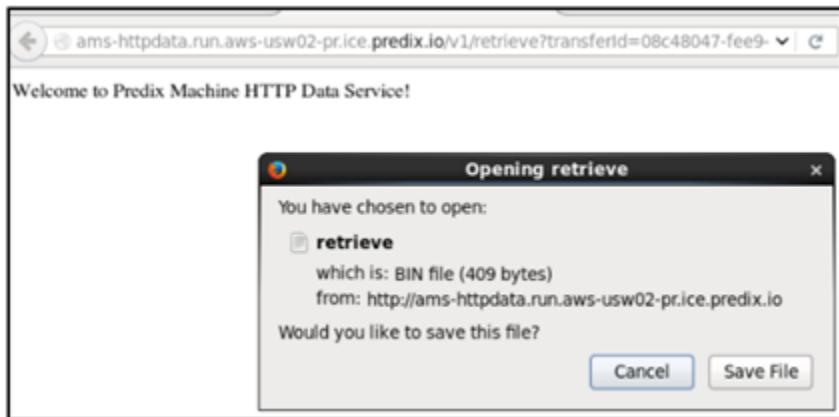
```
3-23 10:55:13,090[Thread-24]|INFO|com.ge.dspmicro.httpriver.send.impl.HttpRiverSendImpl|76-com.g  
river-send-15.3.0| [200] transferId: 08c48047-fee9-4417-a8e1-ec5da3f6a0c1  
: 27c7f71b-9eb5-4918-b186-01d0992a6994  
me: Http Sender Service  
Type: application/octet-stream  
Disposition: null  
Description: 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
```

3. Verify Data is stored in the cloud.

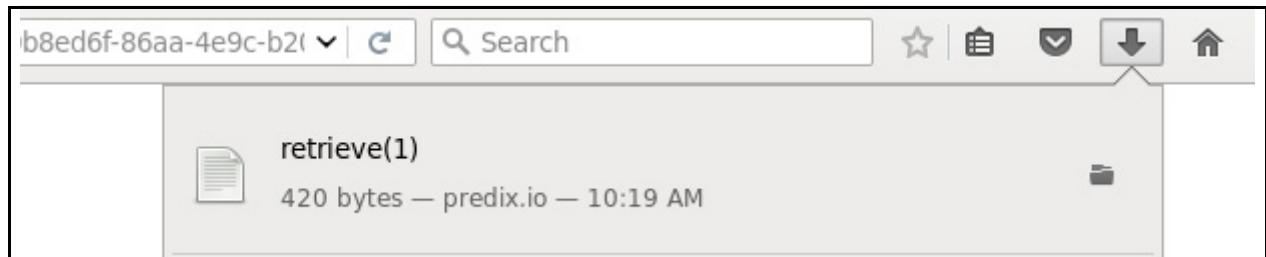
- Copy one of the transfer IDs from the Terminal (ctrl + shift + O)
- Retrieve a specific piece of data based on the transfer ID
 - ◆ Enter the following in your browser address bar:

https://<your_httpdata_url>/v1/retrieve?transferId=<your_transferId>

You will be prompted to open a file



- ◆ Click **Save File**
- ◆ Click the downloads button and then click the retrieve file



The file opens in your text editor

The screenshot shows a text editor window with a toolbar at the top containing icons for Open, Save, Undo, Redo, Cut, Copy, Paste, Find, and Replace. The title bar of the editor window says "retrieve". The main content area displays the following JSON-like code:

```
{"address": "com.ge.dspmicro.machineadapter.modbus://127.0.0.1:502/2/20", "datatype": "0000000000000000"}, {"address": "com.ge.dspmicro.machineadapter.modbus://127.0.0.1:502/1/10", "datatype": "0000000000000000"}]
```

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 from which ModbusPal was started .
- Stop the Machine Container by typing **exit** in the Terminal

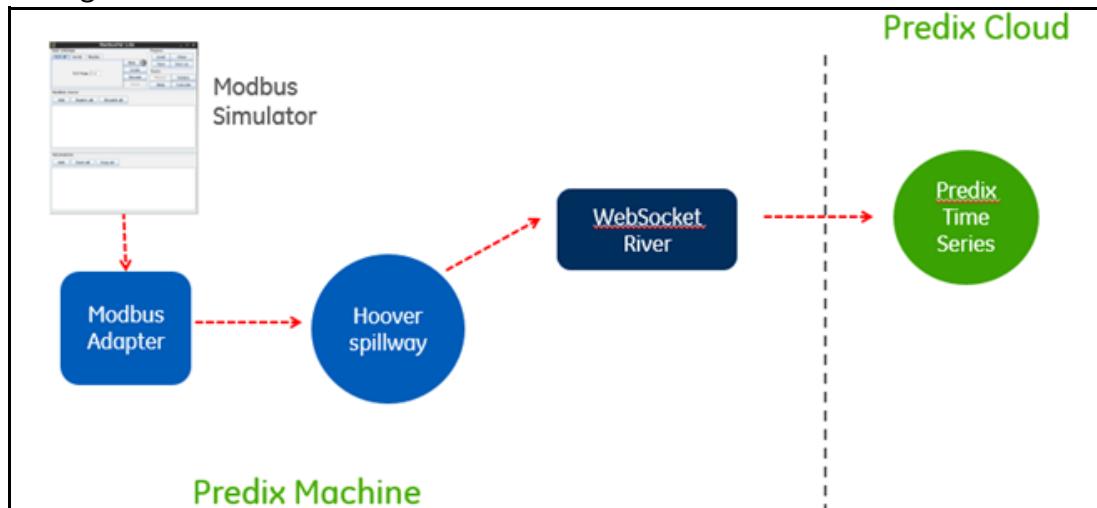


Part III: Connect to the Cloud with Web Socket River

Learning Objectives

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

- Configure a connection between machine and a Time Series database
- Configure the Web Socket Data River in Predix Machine



Lab Exercises

- *Part II: Sending Data to the Cloud using HTTP River, page 273*
- *Validate Websocket River, page 299*
- *Validating Data in Timeseries, page 305*

Directions

Complete the exercises that follow.

Exercise 1: Data Service Setup

Overview

In this exercise you update your UAA client with the time series authorities.

Steps

1. Get time series environment variables.

Note: You created a Time Series service instance in a previous lab and bound it to the locomotive-ingestion-service application. Now you will bind the service instance to your httpdata app.

- In the Terminal, bind the timeseries instance to your httpdata application in order to view environment variables
 - ◆ Use the `cf bind-service` (`cf bs`) command to bind your timeseries service instance to your httpdata application:
`cf bind-service <your_httpdata> <your_timeseries_instance>`
- Get the VCAP environment variables from your application
 - ◆ In the Terminal run the command: `cf env <your-name>-httpdata`
 - ◆ Copy the timeseries variables (ingest and query) to a text file

2. Configure the Predix machine for UAA.

- Your `/configuration/machine/com.ge.dsmpmicro.predixcloud.identity.config` file has already been updated with the following (you did this for the HTTP data river):
`com.ge.dsmpmicro.predixcloud.identity.uaa.token.url=<UAA issuerid>"`
`com.ge.dsmpmicro.predixcloud.identity.uaa.clientid=<your clientid>"`
`com.ge.dsmpmicro.predixcloud.identity.uaa.clientsecret=<your clientid>"`



3. Configure the Hoover Spillway for the Web Socket River.

- Update the /configuration/machine/**com.ge.dsppmicro.hooverspillway-0.config** file by changing the destination river from "Http Sender Service" to "WS Sender Service"

```
# [Required] Destination Data River name to where the data will be sent.  
# Change to the Data River by replacing the value with: Sender Service  
com.ge.dsppmicro.hoover.spillway.destination="WS Sender Service"
```

4. Update the Web Socket River configuration with the time series environment variables.

- Open the /configuration/machine/**com.ge.dsppmicro.websocketriver.send-0.config** file and update the three time series environment variables:
 - com.ge.dsppmicro.websocketriver.send.destination.url="wss://<time series ingest url>/v1/stream/messages"
 - com.ge.dsppmicro.websocketriver.send.header.zone.name=""
 - com.ge.dsppmicro.websocketriver.send.header.zone.value=""

```
# [Required] The URL of the WebSocket service to send to. Must begin  
with ws:// or wss://  
com.ge.dsppmicro.websocketriver.send.destination.url="wss://gateway-  
predix-data-services.run.aws-usw02-pr.ice.predix.io/v2/stream/  
messages"
```

```
# [Required] The name of the header where the zone ID will be  
inserted  
com.ge.dsppmicro.websocketriver.send.header.zone.name="Predix-Zone-  
Id"
```

```
# [Required] The zone ID for the TimeSeries service instance  
com.ge.dsppmicro.websocketriver.send.header.zone.value="967aa495-5f8e-  
b545-b5000f1f975a0"
```

Exercise 2: Validate Websocket River

Overview

In this exercise you validate that the WebSocket River can connect to Predix Cloud as an Administrator before restarting your Modbus simulator.

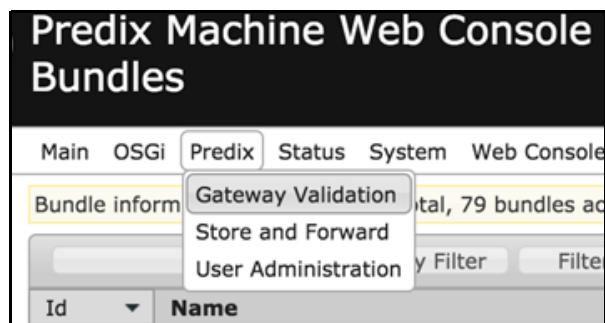
Steps

1. Start Predix machine.

- In a Terminal, navigate to
`/predix/Documents/PredixApps/training_labs/machine/predixmachinesdk-16.2.0/utilities/containers/predix-machine-debug/machine/bin/predix`
- Run the command `./start_container.sh clean`

2. Open the Web Console and validate your WebSocket river route.

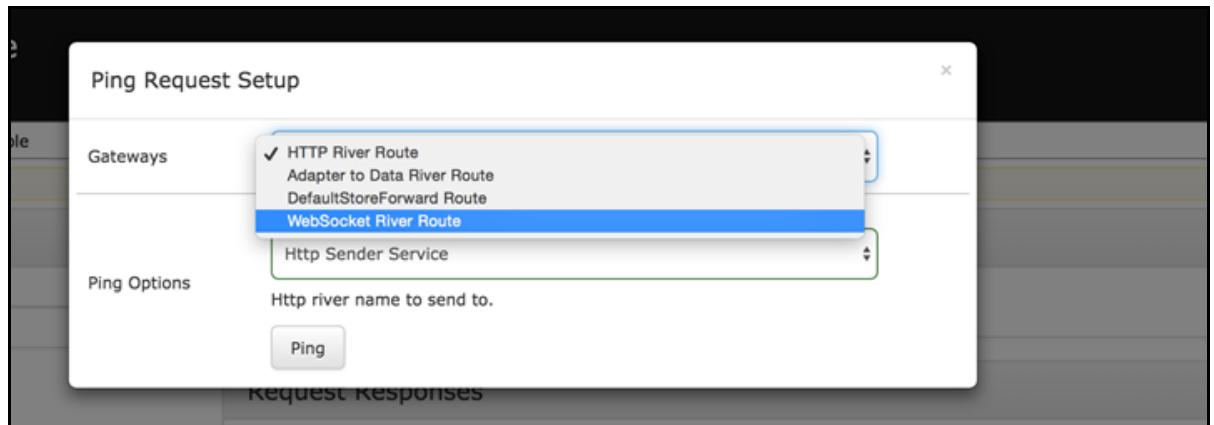
- Open the Web Console (<https://localhost:8443/system/console>)
- Select **Predix > Gateway Validation**



- Select the "+" sign beside the Gateway Ping Requests



- The Ping Request Setup dialog opens
 - ◆ Select **WebSocket River Route** from the dropdown list



- The name of the river will auto-fill
- Select the **Ping** button
- If your websocket river is configured correctly you will get a successful response

The screenshot shows the 'Gateway Ping Requests' section with one entry: 'Initiate Time' (2016-05-12 15:55:39) and 'Gateway Name' (WebSocket River Route). To the right, the 'Request Details' section shows 'WebSocket River Name' (WS Sender Service). Below these, the 'Request Responses' section displays a single row: 'Hop Name' (WebSocket River Send Route), 'Status' (SUCCESSFUL), and 'Message' (Pong message received from endpoint successfully). The 'Message' cell is highlighted with a red border.

3. Update the modbus machine adapter for the Web Socket River.

- Open the /configuration/machine/**com.ge.dspmicro.machineadapter.modbus-0.xml** file:
 - ◆ Remove TCP/IP comments for <**dataNodeConfigs**> data model and update the node configuration as follows:
 - comment out the 2nd unit id using <!-- and -->
 - register names: Node-1-1 and Node-1-2
 - DataType = "SHORT"

```
<dataNodeConfigs>

    <channel protocol="TCP_IP" tcpIpAddress="127.0.0.1"
tcpIpPort="502">
        <unit id="1">
            <register name="Node-1-1" dataType="SHORT"
address="0" registerType="HOLDING" description="temperature"/>
            <register name="Node-1-2" dataType="SHORT"
address="1" registerType="HOLDING" description="pressure"/>
        </unit>
        <!--
        <unit id="2">
            <register name="Node-2-1" dataType="INTEGER"
address="20" registerType="HOLDING" description="temperature"/>
            <register name="Node-2-2" dataType="INTEGER"
address="21" registerType="HOLDING" description="pressure"/>
        </unit>
        -->
    </channel>
```

- Update the **<dataSubscriptionConfigs>** as follows:
 - ◆ Change the update interval to "5"
 - ◆ Change the node names to **Node-1-1** and **Node-1-2**

```
<!-- Both updateInterval and startPointOffset are in  
seconds. If startPointOffset == -1, it will start immediately -->  
<dataSubscriptionConfigs>  
  
    <dataSubscriptionConfig name="Temperature_Subscription"  
updateInterval="5" startPointUnit="MINUTES" startPointOffset="10">  
        <nodeName>Node-1-1</nodeName>  
        <nodeName>Node-1-2</nodeName>  
    </dataSubscriptionConfig>  
  
!-- REMOVE THIS LINE FOR SERIAL  
    <dataSubscriptionConfig name="Pressure_Subscription"  
updateInterval="3600" startPointUnit="HOURS" startPointOffset="600"  
        <nodeName>Node-3-2</nodeName>  
        <nodeName>Node-4-2</nodeName>  
    </dataSubscriptionConfig> REMOVE THIS LINE-->  
  
</dataSubscriptionConfigs>  
  
</modbusMachineAdapterConfig>
```

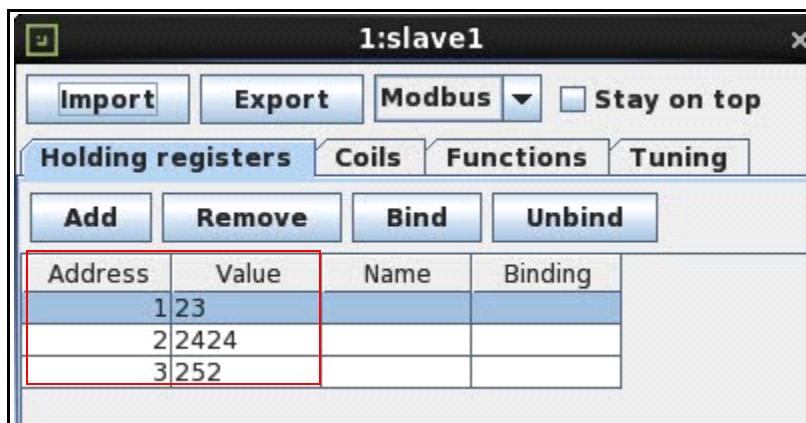


4. Start and configure the Modbus simulator for the Time Series data.

- Start Modbus Pal from the `~/PredixApps/training_labs/machine` folder with this command:

```
sudo java -jar ModbusPal.jar
```

- Configure the simulator it with one asset (slave) and 3 registers with the addresses 1, 2, and 3 and with random values as follows:



- Close the register window and start the simulator

Exercise 3: Validating Data in Timeseries

Overview

In this exercise you will query the data coming into timeseries through websocket river.

Steps

1. Use the Starter Kit UI to log into your UAA client.

- In a browser go to <https://predix-starter.run.aws-usw02-pr.ice.predix.io/>
- Make sure the API Explorer tab is selected
- Click **Login as Client** in the left navigation panel

The screenshot shows the 'Predix Starter Kit' interface. At the top, there's a green header bar with the title 'Predix Starter Kit' on the left, 'Security' in the middle, and 'API Explorer' on the right, which is highlighted with a red box. Below the header is a sidebar on the left containing several icons and links: 'API Home' (home icon), 'Login as Client' (lock icon, highlighted with a red box), 'Login as User' (lock icon), 'Time series ingest' (cloud icon), 'Time series query' (cloud icon), 'Analytics catalog' (list icon), 'Analytics runtime' (graph icon), and 'Websocket client' (grid icon). To the right of the sidebar is a main content area. It contains a text box with the following text: 'This is a demonstration of logging as a client, with the grant type of "client_credentials". Token generated by this call is used to call Predix Industrial Services.' Below this text is a graduation cap icon. Further down, there are input fields for 'UAA URL:' (containing 'https://your-uaa-instance.predix-uaa.run.aws-usw'), 'Client ID:' (empty input field), and 'Client Secret:' (empty input field). At the bottom of this section are two buttons: 'See cURL command' and a large green 'Submit' button.

- ◆ Enter your UAA URL
- ◆ Enter **sample-clientid** for your Predix Machine OAuth2 client
- ◆ Enter **sample-clientid** for the password for your Client ID
- Click **Submit**

You should see an encrypted client token as a response.

UAA URL:

Client ID:

Client Secret:

```
{
  "access_token": "eyJhbGciOiJSUzIiNiJ9.eyJqdGkiOiIwZjg4M2YxNyI1jNWF1LTrkZjAtYTBlNC04MGY4MjMxYj1INTIiLCJzdWIiOiJodHRwLWNsaWVudCIsInNjb3B1IjpBInVhYSSyZXNvdXJjZS1iInRpBmVwZXJpZXMuem9uZXMuOTY3YWE0OTUtNNY4ZS00Mj12LW11NDUtYjUwMGYxZjk3NNEWlmluZ2VzdCisInRpBmVwZXJpZXMuem9uZXMuOTY3YWE0OTUtNNY4ZS00Mj12LW11NDUtYjUwMGYxZjk3NNEWlVzZXiilCJ0aW1c2VyaKvzLnpvbmVzLjk2N2FhNDk1LTVmOGUtNDIyN1i1NTQ1LW11MDBmMWY5NzVhMC5xdWyeSjdLCjgbG11bnRfaWQiO1JodHRwLWNsaWVudCisImNpZC16Imh0dHAtY2xpZW50IiwiYXpwIjoiHR0cC1jbG11bnQiLCJncmFudF90eXB1IjoiY2xpZW50X2NyZWR1bnRpYWxzIiwiitmV2XJNpZyI6IjEOZDZjNmE1LCJpYXQlojEONj0MDASNTgsImV4cC16MTQ2MjQ0NDE1OCwiaXNzIjoiiaHR0cHM6Ly83MTg5YzKMS050WRkLTQzZGQtY3MjZi02YmFkMTMxDzkoGEucHJ1ZG14LXvhYS10cmFpbmluZy5ydW4uYXdzLXVzdzAyLXBylmljZS5wcmVkaXgwaWvb2F1dGgvG9rZW4iLCJ6aWQ1O1i3MtqYzZKMS050WRkLTQzZGQtYjMyZi02YmFkMTMxDzkoGEc1LCJhdWoiOlisiaHR0cC1jbG11bnQiLCJ1YNEiilCJ0aW1c2VyaKvzLnpvbmVzLjk2N2FhNDk1LTVmOGUtNDIyN1i1NTQ1LW11MDBmMWY5NzVhMCJdfQ.dgwa0b3fgOrsOVCktv8r9_uqkb10sSFU9NBYQKYeBlFewinOiiii_jJaw_Uz_VQMiLw2ND8t6oMxa3UEEtTTPp49JYFX_NSpO_n2Bnd1xDb45za3W1-VSS3czE3mXVK17DEFgxGNKZI3OVL1qY7VVA3erp_mSDvNS7b1t5gTvnAMdIH_GCqaE3mQNdmagkfOzUSufBu5b9_sYU266nbq184BugxV_PYTb3ZfMeS6zV551PBEW4B1CUvWEB4FG5Zy5gYMO49UjZUB-8SfYShgD3fnwulyUao3R3F6idY-FPY_e11NBadinZJFpWgJX-1UHCi8wjjREmhIQWJEPLzAQO",
  "token_type": "bearer",
  "expires_in": 43199,
  "scope": "uaa.resource timeseries.zones.967aa495-5f8e-4226-b545-b500f1f975a0.ingest timeseries.zones.967aa495-5f8e-4226-b545-b500f1f975a0.user timeseries.zones.967aa495-5f8e-4226-b545-b500f1f975a0.query",
  "jti": "0f883f17-c5ae-4df0-a0f4-80f8231b9b52"
}
```

- Click the **Time Series Query** link in the navigation panel on the left
Your UAA URL and client token are already stored in the page.
- ◆ Choose the **Get Tags** option from the **Choose Request:** drop-down
- ◆ Enter the predix-zone-id from your predix time series **query** instance

The screenshot shows the GE Digital Time Series Store interface. On the left sidebar, under the 'Time series query' section, several options are listed: 'Login as Client', 'Login as User', 'Time series ingest', 'Time series query' (which is highlighted with a red box), 'Analytics catalog', and 'Analytics runtime'. Below these is a 'Logout' option. To the right, a main panel displays the following information:

- Make a few basic queries against your time series instance.**
- UAA URL:** https://7189c6d1-99dd-41dd-b32f-6bad13146d8a.pr.ice.predix.io
- Choose Request:** A dropdown menu set to 'Get Tags' (highlighted with a red box).
- Request Headers:**
 - predix-zone-id:** 495-5f8e-4226-b545-b500f1f975a0 (highlighted with a red box)
 - authorization:** Bearer [View Token](#)
- Buttons:** 'See cURL command' (grey) and 'Submit' (green).

- ◆ Click **Submit**
- Your time series data tags appear in the results.

The screenshot shows the 'See cURL command' output area, which contains a green 'Submit' button. Below it is a JSON response:

```
{
  "results": [
    "Node-1-1",
    "Node-1-2",
    "null"
  ]
}
```

- Change the **Choose Request:** drop-down to **Latest Datapoints Request**

- ◆ In the Request Body, replace the tag name with the name of the first node as shown:

Request Body

```
{"tags": [{"name": "Node-1-1"}]}
```

- ◆ Click **Submit**

Your request brings back the data from your Time Series service instance

```
{  
  "tags": [  
    {  
      "name": "Node-1-1",  
      "results": [  
        {  
          "groups": [  
            {  
              "name": "type",  
              "type": "number"  
            }  
          ],  
          "attributes": {  
            "address": [  
              "com.ge.dspmicro.machineadapter.modbus-//127.0.0.1-502/1/0"  
            ],  
            "category": [  
              "REAL"  
            ],  
            "datatype": [  
              "SHORT"  
            ]  
          },  
          "values": [  
            [  
              1461631731081,  
              23,  
              3  
            ]  
          ]  
        }  
      ],  
      "stats": {  
        "rawCount": 1  
      }  
    }  
  ]  
}
```

