Predix

Predix Fundamentals

Student Lab Guide

October 2015

GE Digital

Predix

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 microservices on the Predix platform.

Course Prerequisites:

- Install the most recent DevBox version
  - Download Virtual Box 4.3.1.5
  - Download and import DevBox into your Virtual Box

Start the VM in Oracle VirtualBox

- Start DevBox

Tip: Make sure you are on the GE network

- Login with:
  Username: **predix**
  Password: **predix**

Note: All lab exercises will be completed in your Devbox.

Set Your Environment

- Download the scripts from the following link using firefox on the DevBox:

http://gis06299.devcloud.ge.com:8000/public/training/guides/fundamentals.sh

- Open a Terminal (icon on the Desktop), and run the following commands:

```
cd ~/Downloads
chmod 777 fundamentals.sh
./fundamentals.sh
```

Lab 1: Getting Started with Cloud Foundry

Learning Objectives

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

- Log in to Cloud Foundry
- Explore the Service Catalog/Marketplace
- Create a Service Instance

Lab Exercises

Cloud Foundry Commands

Command	Description
cf login	Login to cloud foundry
cf marketplace	Display all services in the catalog
cf create service	Creating a new service
cf apps	Display all microservices in your space
cf push	Deploy a service
cf services	Display all service instances in your space

Exercise 1: Logging into Cloud Foundry

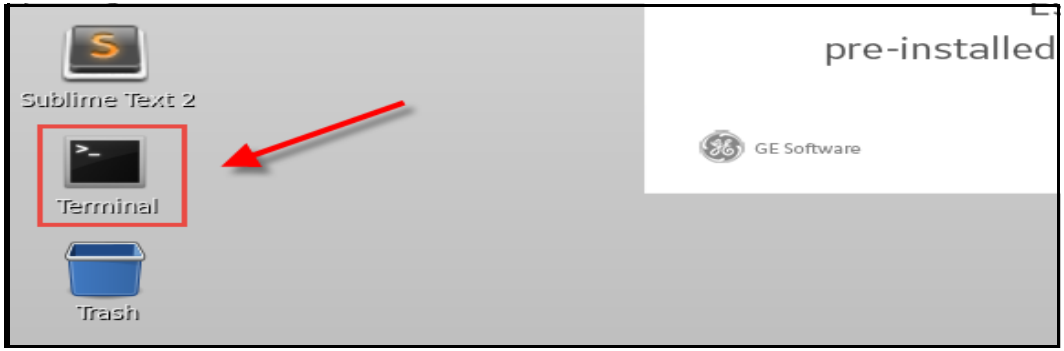
Overview

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

Steps

1. Logging in to CF.

- Open a Terminal
  - Double click on the Terminal icon on your desktop



- Log in to CF
  - Run this command:

```
cf login
```

Note: This command directs you to GE's API endpoint for CF. If you are having trouble logging in; in your Terminal run this command:

```
cf login -a https://api.grc-apps.svc.ice.ge.com/
```

2. Entering you CF credentials.

- You are prompted to enter your email
  - Type in your email and press **Enter** (example: owen.matter@ge.com)
- You are prompted to enter your password
  - Type **`P@ssword1`** and press **Enter**

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

3. Selecting an org.

Note: Not all users will be prompted to select an org

- You are prompted to select an org
  - Type the number with the selection for "predix-adoption" and press **Enter**

4. Selecting a space.

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

```
[predix@localhost spring-music]$ cf login
API endpoint: https://api.grc-apps.svc.ice.ge.com

Email> georgia.smith@ge.com

Password>
Authenticating...
OK

Targeted org predix-adoption

Select a space (or press enter to skip):
1. training1
2. training2
3. training3

Space>
```

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

Tip ~To change your space:

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

You are now logged in.

```
API endpoint:   https://api.grc-apps.svc.ice.ge.com (API version: 2.28.0)
User:           Owen.Matter@ge.com
Org:            predix-adoption
Space:          training2
[predix@localhost predix-seed-1.1.3]$
```

Exercise 2: Creating a Service Instance

Overview

In this exercise, you will use CF commands to display all services in the marketplace and create a instance of the postgreSQL service in your CF space.

Steps

1. Displaying services in the marketplace.

- In your Terminal, run this command:

  `cf marketplace`

  All available services appear

```
SF01502434206A:scripts 502434206$ cf marketplace
Getting services from marketplace in org predix-adoption / space training1 as asha.yadav@ge.
OK

service                    plans              description
bizops                     beta-plan          Predix BizOps Service by Nurego Inc.
logstash                   free               Logstash 1.4 service for application developmer
logstash14-stc-logging-5   beta-plan          Data pipeline to process logsl and event data.
p-rabbitmq                 standard           RabbitMQ is a robust and scalable high-performa
p-riakcs2                  developer          An S3-compatible object store based on Riak CS
stc-analytics-catalog      beta-plan          STC Analytics Catalog
stc-analytics-runtime      beta-plan          STC Analytics Runtime
stc-asset                  beta-plan          Service to model and manage industrial assets
stc-monitoring             beta-plan          Manage and monitor apps. By New Relic
stc-postgresql-2           free               PostgreSQL 9.3 service for application developr
stc-redis-6                shared-vm          Redis service to provide a key-value store
stc-time-series            beta-plan          Predix Time-Series Service
time-series-dev            Development Plan   Time-Series Service
```

2. Creating a managed service instance.

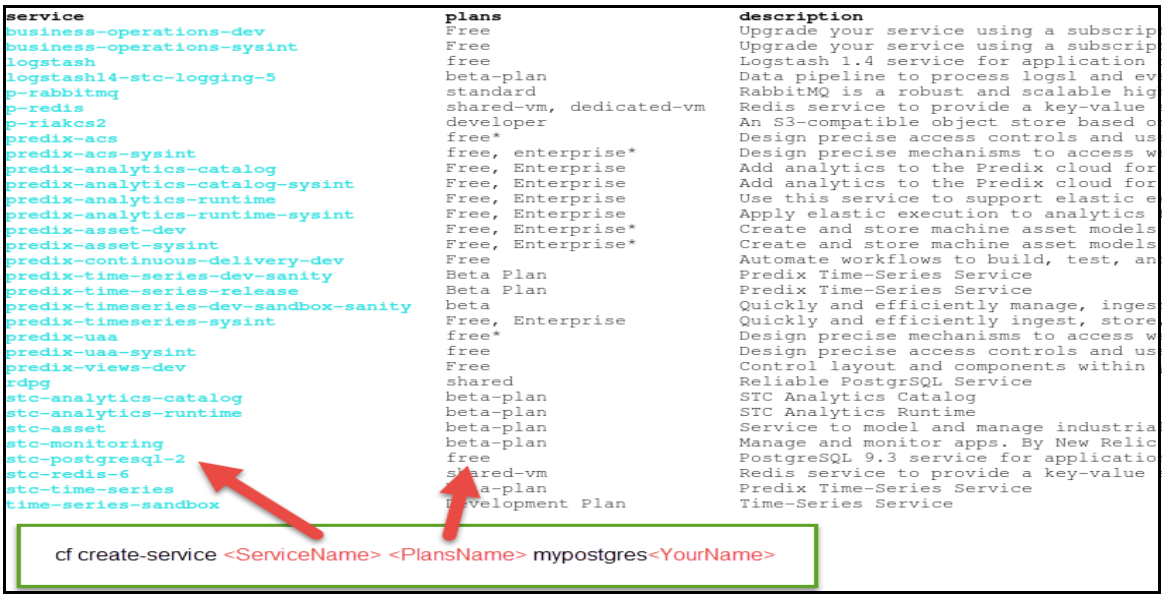
- In the Terminal run the command to create a service instance with the following syntax:

```
cf create-service <servicename> <plansname> mypostgres<yourname>
```

  - Replace <ServiceName> with a service name from the list
  - Replace <PlansName> with a the plans name associated with the service
  - Replace <YourName> with your name

  Sample command line for creating an service instance:

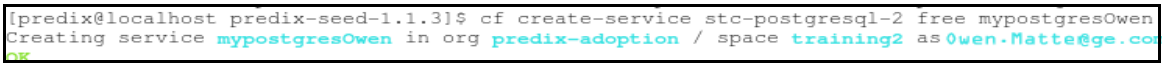
  `cf create-service stc-postgresql-2 free mypostgresJoeSmith`



```
service                                plans                      description
business-operations-dev                Free                       Upgrade your service using a subscrip
business-operations-sysint             Free                       Upgrade your service using a subscrip
logstash                               free                       Logstash 1.4 service for application
logstash14-stc-logging-5               beta-plan                  Data pipeline to process logsl and ev
p-rabbitmq                             standard                   RabbitMQ is a robust and scalable hig
p-redis                                shared-vm, dedicated-vm    Redis service to provide a key-value
p-riakcs2                              developer                  An S3-compatible object store based o
predix-acs                             free*                      Design precise access controls and us
predix-acs-sysint                      free, enterprise*          Design precise mechanisms to access w
predix-analytics-catalog               Free, Enterprise           Add analytics to the Predix cloud for
predix-analytics-catalog-sysint        Free, Enterprise           Add analytics to the Predix cloud for
predix-analytics-runtime               Free, Enterprise           Use this service to support elastic e
predix-analytics-runtime-sysint        Free, Enterprise           Apply elastic execution to analytics
predix-asset-dev                       Free, Enterprise*          Create and store machine asset models
predix-asset-sysint                    Free, Enterprise*          Create and store machine asset models
predix-continuous-delivery-dev         Free                       Automate workflows to build, test, an
predix-time-series-dev-sanity          Beta Plan                  Predix Time-Series Service
predix-time-series-release             Beta Plan                  Predix Time-Series Service
predix-timeseries-dev-sandbox-sanity   beta                       Quickly and efficiently manage, inges
predix-timeseries-sysint               Free, Enterprise           Quickly and efficiently ingest, store
predix-uaa                             free*                      Design precise mechanisms to access w
predix-uaa-sysint                      free                       Design precise access controls and us
predix-views-dev                       Free                       Control layout and components within
rdpg                                   shared                     Reliable PostgrSQL Service
stc-analytics-catalog                  beta-plan                  STC Analytics Catalog
stc-analytics-runtime                  beta-plan                  STC Analytics Runtime
stc-asset                              beta-plan                  Service to model and manage industria
stc-monitoring                         beta-plan                  Manage and monitor apps. By New Relic
stc-postgresql-2                       free                       PostgreSQL 9.3 service for applicatio
stc-redis-6                            shared-vm                  Redis service to provide a key-value
stc-time-series                        beta-plan                  Predix Time-Series Service
time-series-sandbox                    Development Plan           Time-Series Service
```

cf create-service <ServiceName> <PlansName> mypostgres<YourName>

  - The image below indicates your service is created



```
[predix@localhost predix-seed-1.1.3]$ cf create-service stc-postgresql-2 free mypostgresOwen
Creating service mypostgresOwen in org predix-adoption / space training2 as Owen.Matte@ge.com
OK
```

Tip: More Info on managing service instance using CF CLI:
http://docs.pivotal.io/pivotalcf/devguide/services/managing-services.html

3. Displaying all services instances in your space.

- In your Terminal, run this command:

  `cf services`

  - All services appear

```
dix@localhost ~]$ cf s
ing services in org predix-adoption / space training1 as asha.yadav@ge.com...

                      service                 plan        bound apps                                                                  last
alytics               stc-analytics-runtime   beta-plan                                                                               crea
set                   stc-asset               beta-plan                                                                               crea
stgres-hiro           rdpg                    shared                                                                                  crea
stgres-prabhat        rdpg                    shared                                                                                  crea
stgresgarryliu        rdpg                    shared                                                                                  crea
stgresgarycristofoli  rdpg                    shared      predix-alarmservice-garycristofoli                                          crea
stgreshelinyuan       rdpg                    shared                                                                                  crea
stgresJunghoEric      rdpg                    shared                                                                                  crea
stgreskei             rdpg                    shared                                                                                  crea
stgresLachlanHope     rdpg                    shared                                                                                  crea
stgresRaisaHashem     rdpg                    shared                                                                                  crea
stgrestangpei         rdpg                    shared                                                                                  crea
pg                    rdpg                    shared                                                                                  crea
elic                  stc-monitoring          beta-plan                                                                               crea
gresql93              stc-postgresql-2        free        training1-predix-dataingestion, training1-predix-datariver                  crea
ixAsset               stc-asset               beta-plan   training1-predix-dataingestion, training1-predix-dataseed-212059861          crea
ixTimeseries          stc-time-series         beta-plan   training1-predix-dataingestion                                              crea
PersistenceService_   user-provided                        predix-seed-dev-212441999zzz
```

Lab 2: Build and Deploy a Microservice

Learning Objectives

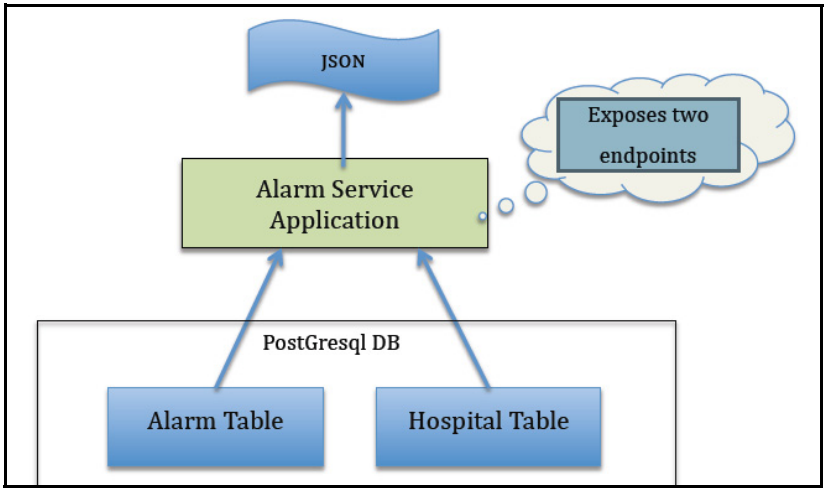
By the end of the lab, you will be able to:

- Use the manifest file to deploy an application
- Build and deploy a java miscroservice to the Predix Cloud

Lab Exercises

Directions

As part of this lab, you will be building an Alarm Service microservice using a maven archetype and open JPA framework. The service exposes 2 endpoints, one to fetch data from the alarm table and another to fetch data from the hospital table.



Exercise 1: Adding a Maven Archetype

Overview

In this exercise you will build and deploy a java microservice to the Predix Cloud.

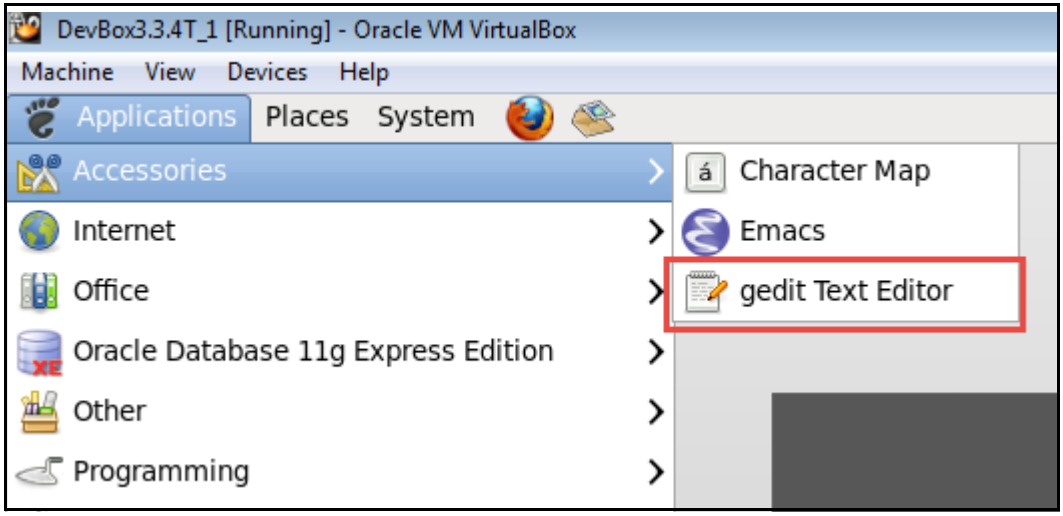
Steps

1. Updating the archetype file.

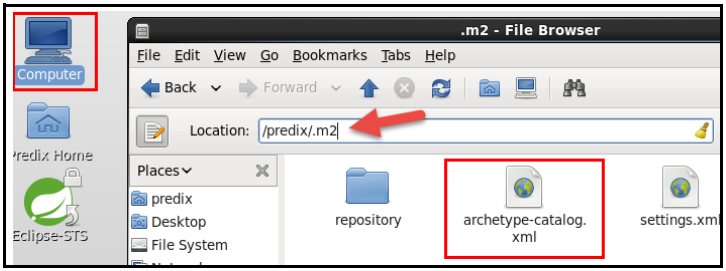
Maven archetypes are used to generate bootstrap applications as a starting point in your application.

Tip: All files used in this lab are located in */PredixApps/training_labs/Lab_2*

- Open **gedit** from the Applications tab



- In gedit, open the file */predix/PredixApps/training_labs/Lab_2/**Archetype.txt***
  - Copy all contents of the file
- In your file browser, under the location bar, type `/predix/.m2` and press **Enter**
- Open **archetype-cataglog.xml**



  - Paste contents above the `</archetypes> tag`

```
    <version>15.1.0</version>

    <description>dspmicro-consumer-archetype-training</description>

  </archetype>

  <archetype>
    <groupId>com.ge.predix.solsvc.training</groupId>
    <artifactId>predix-hospital-alarm-service-archetype</artifactId>
    <version>1.0.0</version>
  </archetype>

</archetypes>

</archetype-catalog>
```

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

2. Running the postgreSQL service.

- In your Terminal, run this command:

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

  - A notice that postgresSQL has started appears

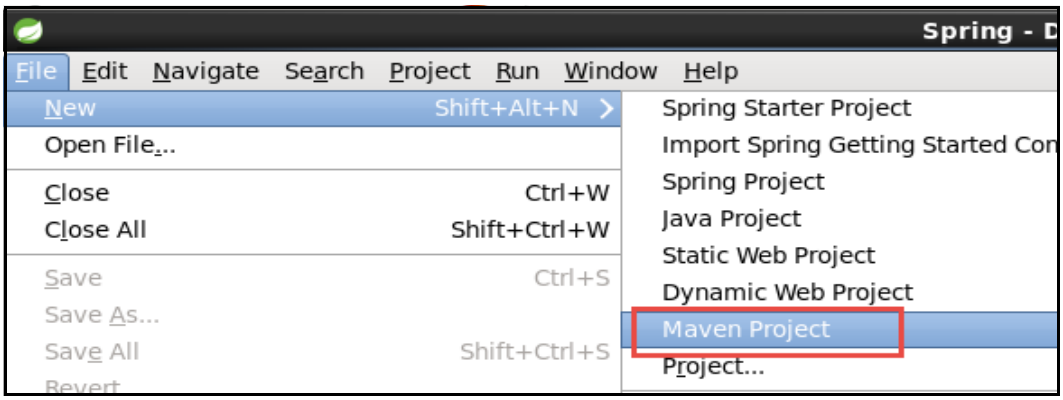
```
[predix@localhost fundamentals]$ sudo /etc/init.d/postgresql-9.3 start
Starting postgresql-9.3 service:                           [  OK  ]
```

Tip: PostgreSQL is an open source RDBMS used to manage data in a relational format. Having this service running is critical to building the java project. The local postgresql database is used to run the tests during the build. After the application is deployed to the Predix Cloud, the local service is no longer needed.

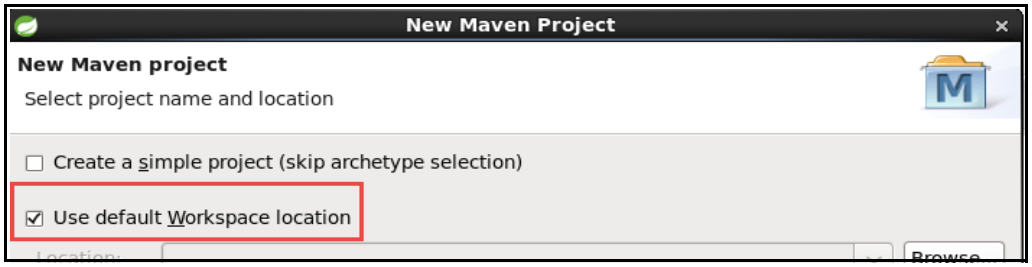
3. Creating a new maven project.

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

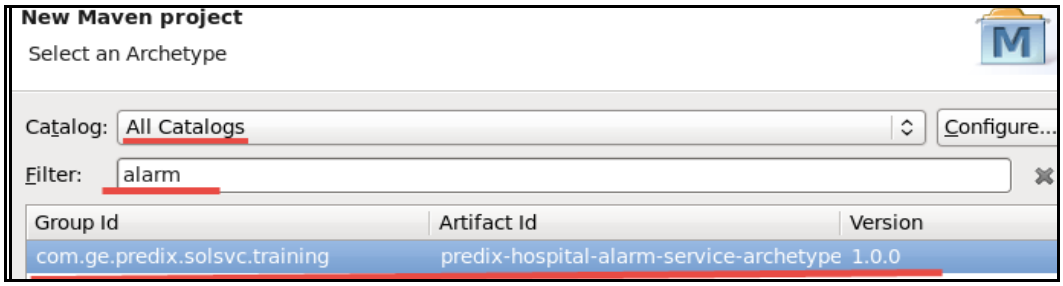
- From the File menu, select *New* > *Maven Project*



- Accept all defaults on the screen and click Next

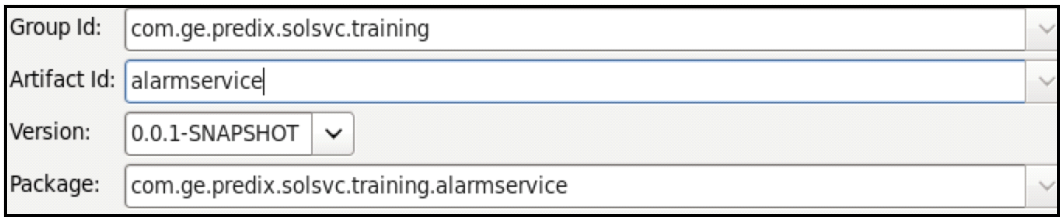


- Enter project details
  - Catalog: **All Catalogs**
  - Filter: type **`alarm`**
  - Select *predix-hospital-alarm-service-archetype*
- Click **Next**

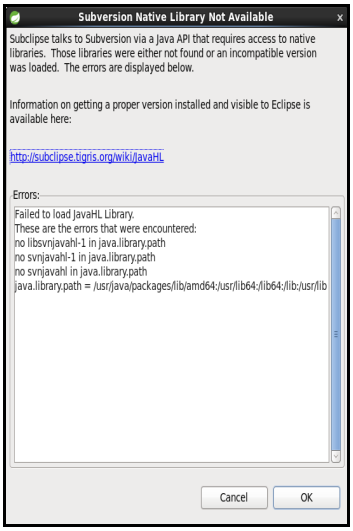


- Enter the archetype parameter as show below:

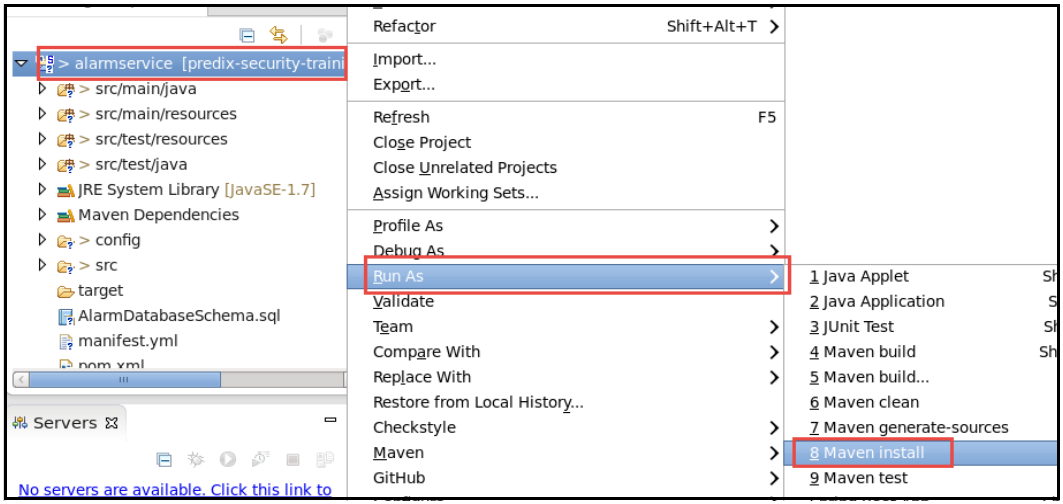
Alert: Make sure the Group Id is entered as shown below. The field might be pre-populated with incorrect information.



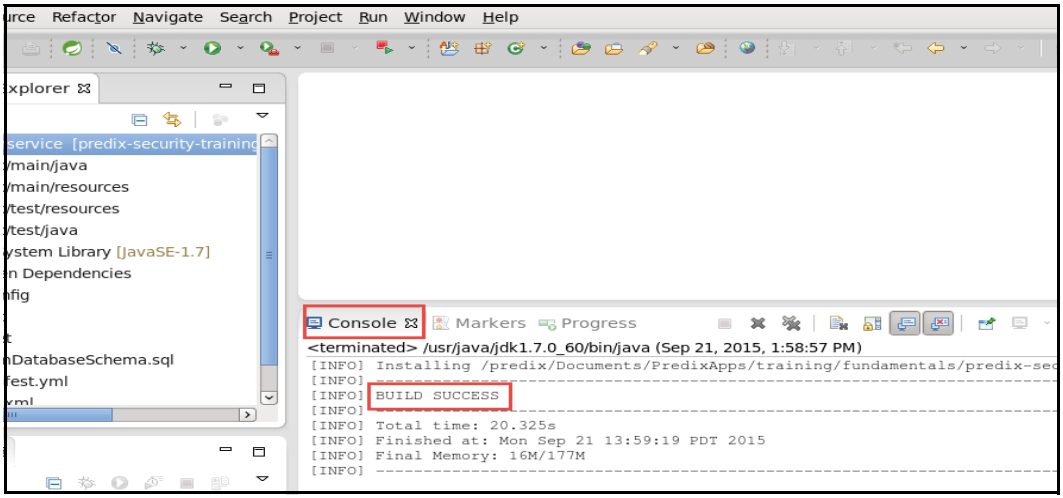
- Click **Finish**
- Click **OK** when this pop up window appears



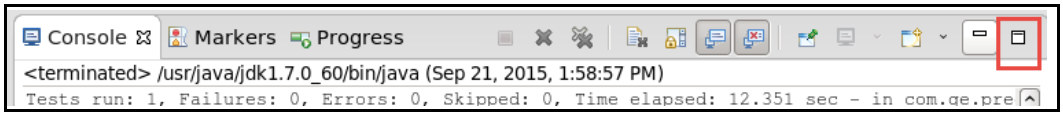
- The new alarmservice project appears in the Package Explorer
- In Package Explorer, right-click on the project root **(alarmservice)** and select *Run As > Maven Install*



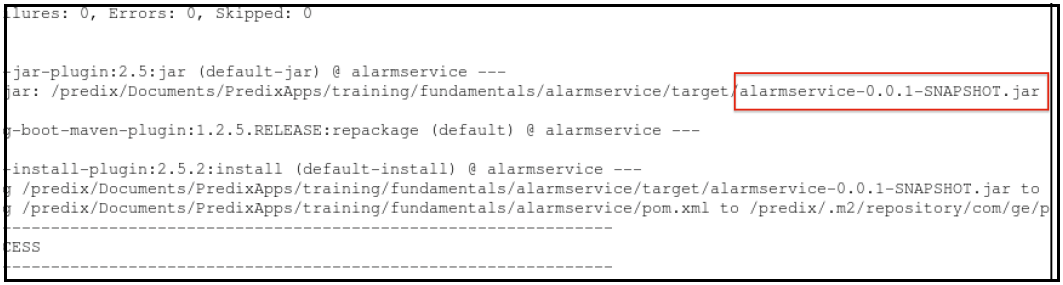
- In the console window below, a message of "BUILD SUCESS' appears



- Click the max console tab to maximize the console window



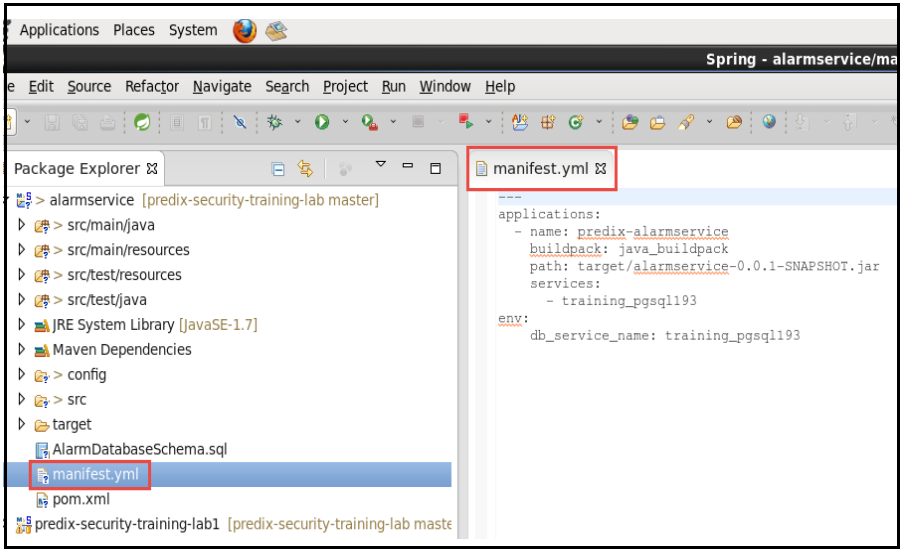
Notice the name of the output JAR file is **alarmservice-0.0.1-SNAPSHOT.jar**. This is the output of the build that is deployed to Cloud Foundry.



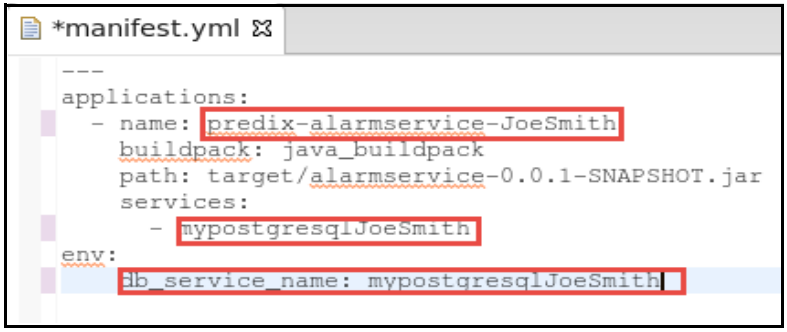
4. Updating the manifest file.

Note: Application manifests define how to deploy applications to Cloud Foundry (how many instances to create, how much memory to allocate, what services applications should use, etc.).

- In Eclipse, open file **manifest.yml** under the alarmservice project



- 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 service instance you created in Lab 1
    - *Logging into Cloud Foundry*, page 2
  - **db_service_name**: Change the services element to use the postgresql microservice
    - Service instance you created in Lab 1
  - Press <ctrl> + <s> to save the file

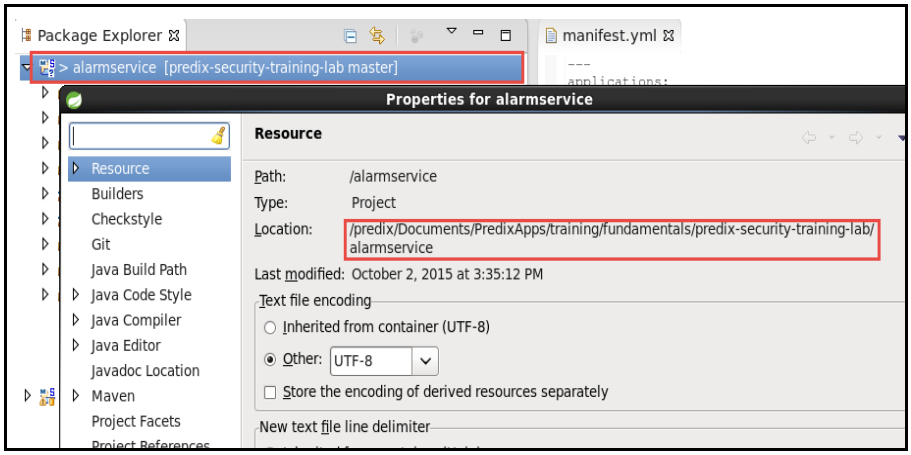


To do this: If you do not remember the service instance you created in Lab 1, follow the steps below to check your service instance.

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

5. Deploying the microservice to the Predix Cloud.

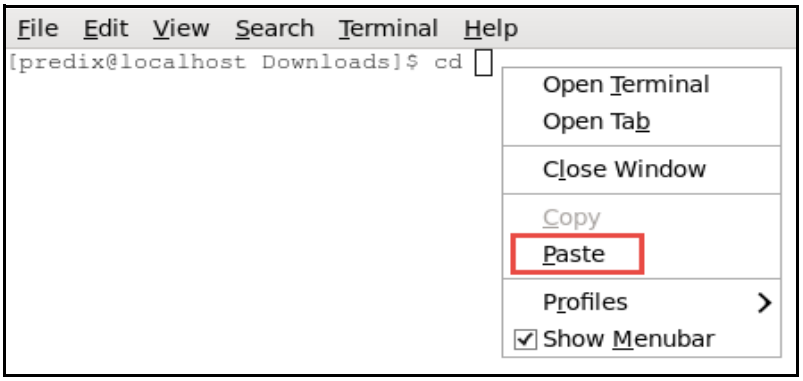
- In Eclipse, click on the project root **(alarmservice)**
- To open the properties window, press <alt> + <enter>
- Copy the alarmservice location



- In the Terminal, navigate to the alarmservice directory by running this command:

  cd `<location of alarm service project>`

  - Replace <location of alarm service project> with the copied location from Eclipse by right clicking, then selecting *paste*



- Deploy the microservice by running this command:

  cf push

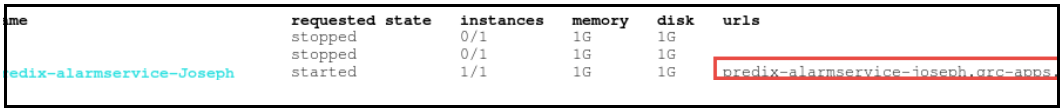
> ***Tip:*** The message App Started verifies the app is deployed and an instance app has started.

6. Testing your service in a browser.

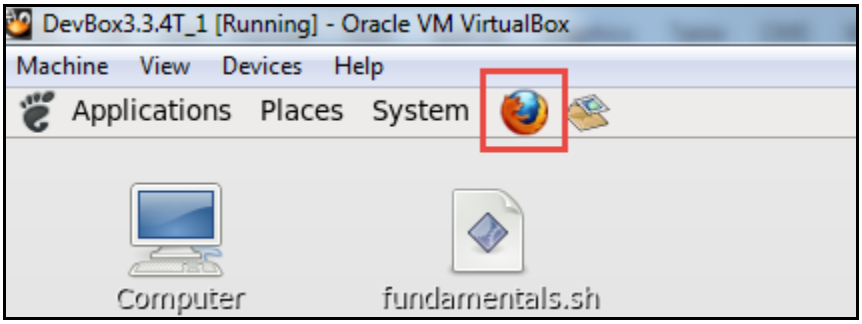
- In the Terminal, run this command:

  cf a

  - All microservices deployed in the space are displayed
- Copy the alarmservice URL
  - Located your alarmservice in the list and highlight the URL
  - Press `<ctrl>` + `<shift>` + `<c>` to copy the url



- Open Firefox by clicking the firefox icon on top



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

- The content of the alarm table appears

7. Viewing recent logs on your microservice.

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

Tip: To find your Microservice Name, follow the instructions below:
- In a Terminal, type **`cf a`**
- Locate your Microservice Name under the name column

```
onnected, dumping recent logs for app predix-alarmservice-YourName in org predix-adopti

015-07-28T14:07:58.40-0700 [API/0]      OUT Created app with guid 3b732fec-bb36-476d-93
015-07-28T14:08:40.47-0700 [DEA/27]     OUT Got staging request for app with id 3b732fe
015-07-28T14:08:42.33-0700 [API/1]      OUT Updated app with guid 3b732fec-bb36-476d-93
015-07-28T14:08:42.70-0700 [STG/27]     OUT -----> Downloaded app package (24M)
015-07-28T14:08:43.74-0700 [STG/0]      OUT -----> Java Buildpack Version: v2.7.1 | htt
015-07-28T14:09:32.47-0700 [STG/0]      OUT -----> Downloading Open Jdk JRE 1.8.0_51 fr
.0s)
015-07-28T14:09:33.75-0700 [STG/0]      OUT        Expanding Open Jdk JRE to .java-buil
015-07-28T14:09:38.84-0700 [STG/0]      OUT -----> Downloading Spring Auto Reconfigurat
configuration-1.7.0_RELEASE.jar (5.0s)
```

Every log line contains these four fields:
- Timestamp
- Log type (origin code)
- Channel: either STDOUT or STDERR
- Message

Exercise 2: Adding another API Endpoint to a Microservice

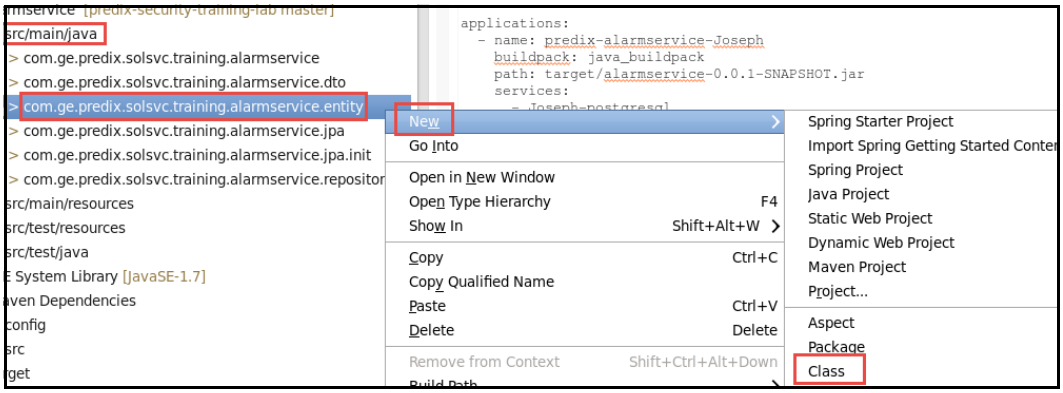
Overview

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

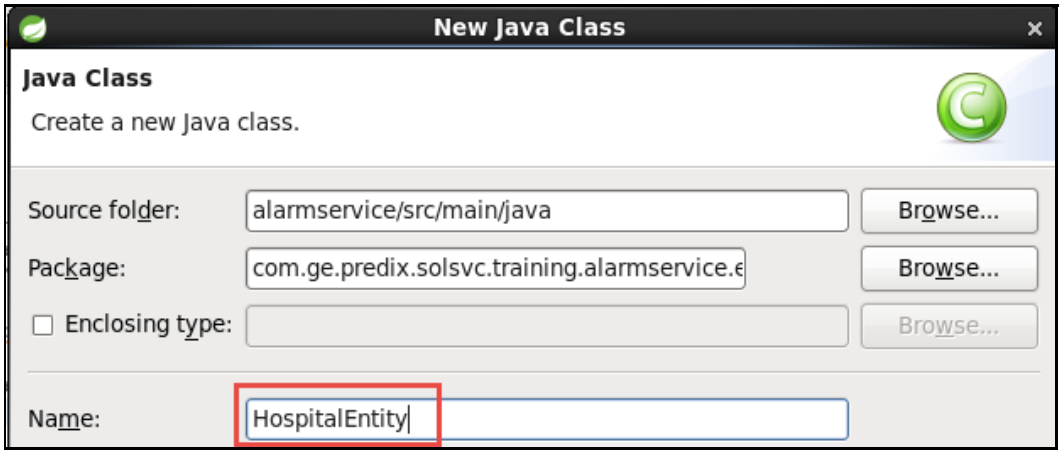
Steps

1. Creating an Entity.

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

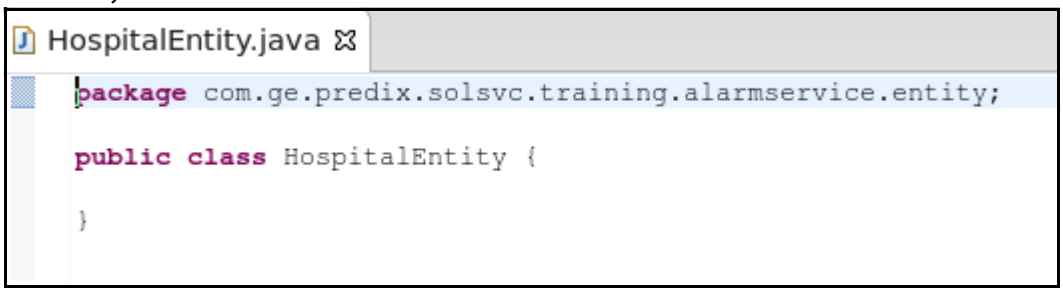


- In the Name field type **HospitalEntity**

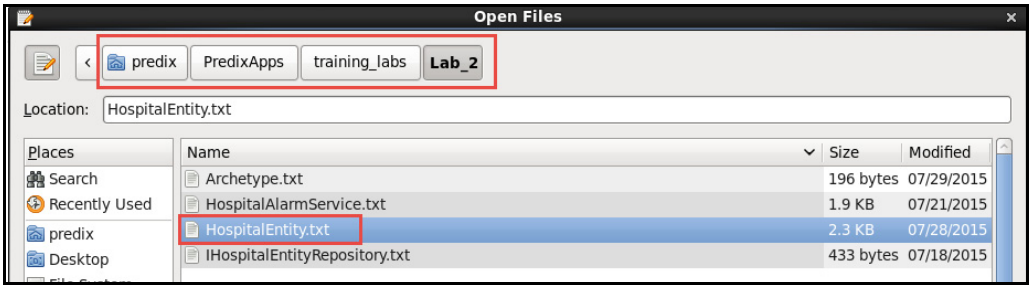


- Press **Finish**

  The entity is created



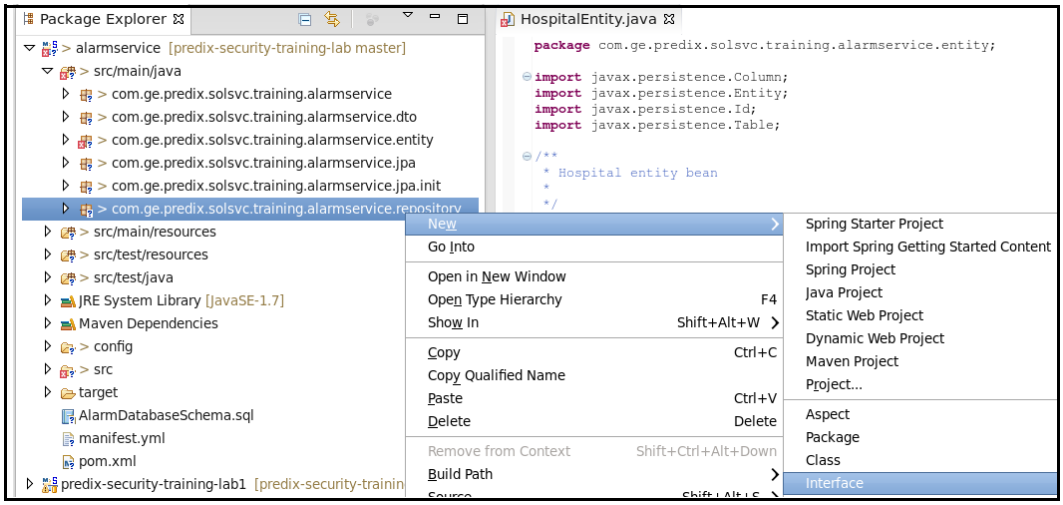
- In gedit, open the file */predix/predixApps/training_labs/Lab2/**HospitalEntity.txt***



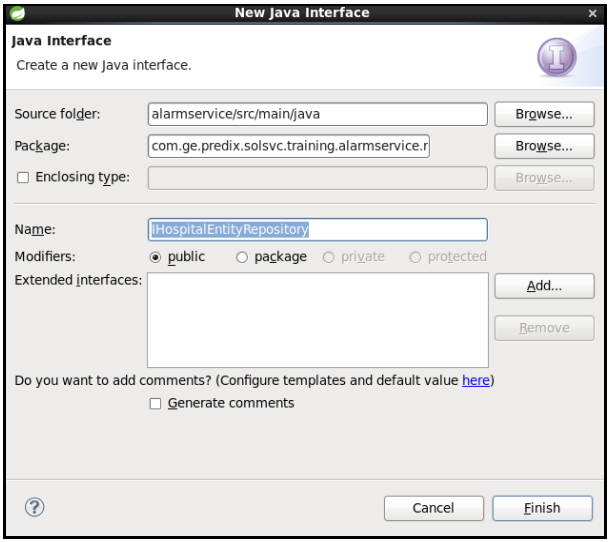
- Copy all content of the file
- In Eclipse, replace everything in your **HospitalEntity.java** class
  - Press `<ctrl> + <shift> + <o>` to Organize Imports
  - Press `<ctrl> + <s>` to save the file

2. Creating an interface.

- In Eclipse, under directory: *"/src/main/java"* right click on the package **com.ge.predix.solsvc.training.alarmservice.repository**
- Select *New -> Interface*



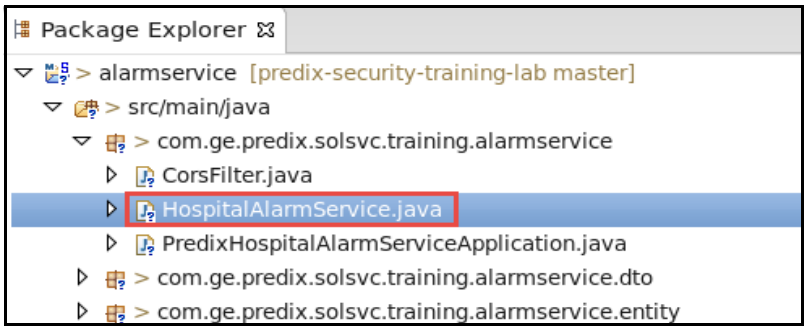
- Under name, enter **IHospitalEntityRepository,** then click **Finish**



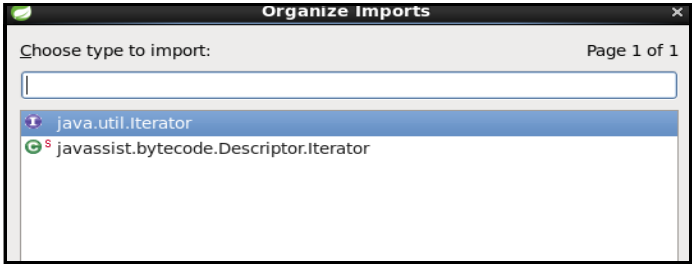
- In gedit, open the file */predix/predixApps/training_labs/Lab_2/**IHospitalEntityRepository.txt***
- Copy all content of the file
- In Eclipse, replace everything in your **IHospitalEntityRepository.java** class
- Press `<ctrl> + <shift> + <o>` to Organize Imports
- Press `<ctrl> + <s>` to save the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

3. Adding mapping to create the service.

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



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



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

4. Populating the AlarmService and Hospital data.

- In Eclipse, under package **com.ge.predix.solsvc.alarmsrvice.jpa.init**, double click file **InitAlarmServiceData.java** to edit the file
- Uncomment the usage of hospitalRepo by removing (`/*`) and (`*/`)
- Uncomment the usage 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. Compiling and Deploying the microservice.

- In your Terminal, 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
```

- In your Terminal, run this command:

  `cf push`

The message "App Started" verifiers the app was successfully deployed.

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

App started

OK
```

6. Testing your service.

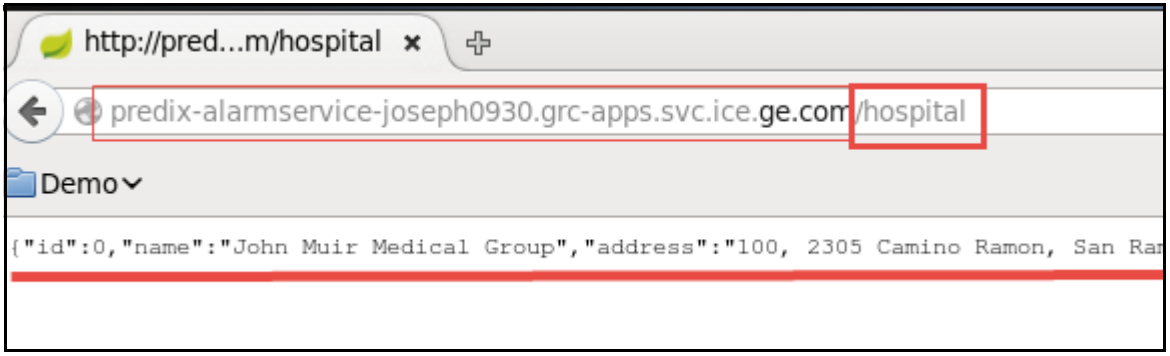
- In Firefox, place your service url in the web address, append "/hopsital" to the end of your url and press **Enter**

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

- In your Terminal run this command:

  `cf a`
- Locate your microservice name under the "name" column and find the url to the right under the "urls" column

- Your data appears

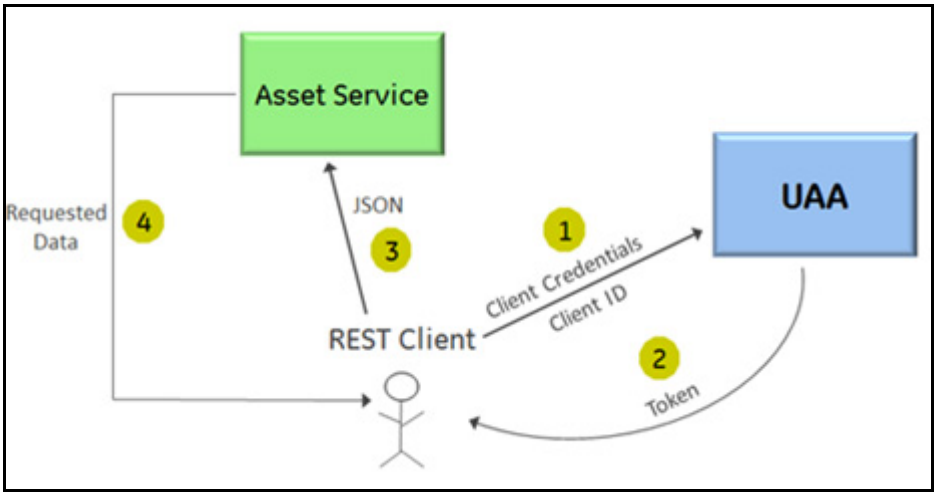


Lab 3: Authenticating Against UAA

Learning Objectives

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

- Retrieve a UAA (User Account and Authentication) token from the UAA service
- Retrieve data from the Asset service using the UAA service “bearer” token



Lab Exercises

- *Fetching a Token from the UAA Service*, page 28
- *Retrieving Data from the Asset Service*, page 33

Exercise 1: Fetching a Token from the UAA Service

Overview

To access and update data, users need to authenticate using a UAA token. In this exercise, you use the REST client to request an authorization token from the UAA service. The token becomes part of every data request to the Asset service so that the service knows the request is coming from an authenticated user.

Steps

1. Launch the REST client.

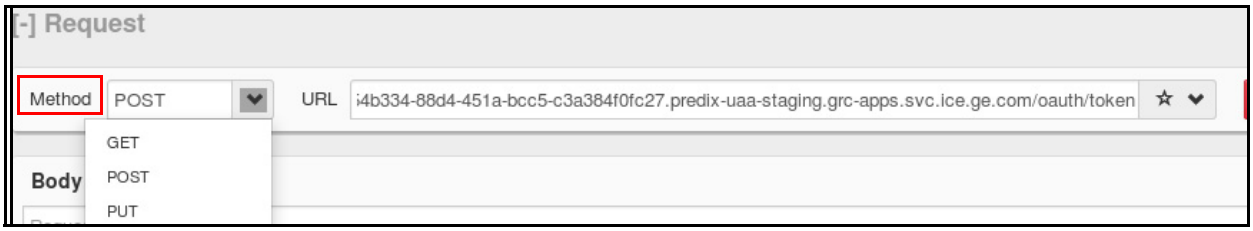
The REST client is a browser-based developer tool used to build and test HTTP requests.

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

2. Begin configuring the POST request.

A POST request is an HTTP request to a web server. The web server accepts and stores the data enclosed in the body of the request message. It is typically used when uploading a file or submitting a completed web form.

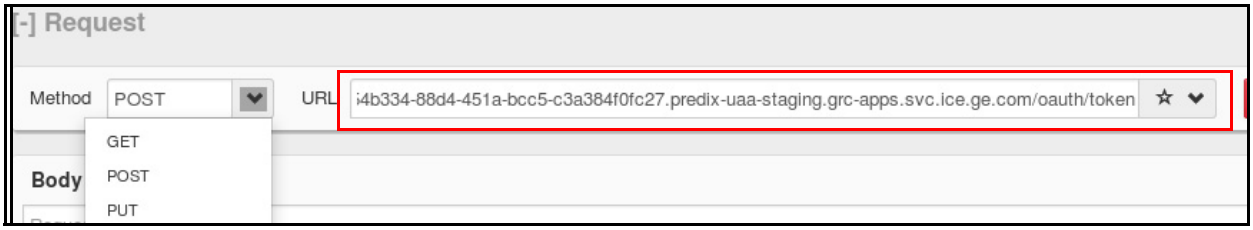
- Set the Request Method by selecting **POST** from the **Method** menu



The **URL** field indicates the web address of the instance of the UAA service provider against which you will authenticate.

- Set the URL by entering the following in the address (**URL**) field

  `https://4354b334-88d4-451a-bcc5-c3a384f0fc27.predix-uaa-staging.grc-apps.svc.ice.ge.com/oauth/token`



3. Provide credentials to the service.

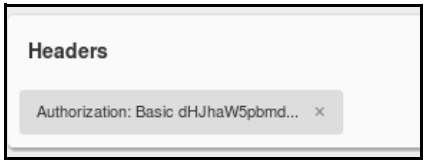
It is also necessary to provide a username and password to the authentication service to authenticate. First, you indicate what type of authentication method you will use.

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

Next, you enter the credentials for the user accessing the service.

- Enter the Username: **training_client**
- Enter the Password: **training_secret**
- Click **Okay**

A new authorization header has been added. Headers are located at the start of an HTTP request message and define how the request operates.



4. Add a Content-Type header.

A content type tells the server what type of message is coming its way. In this case, you are indicating a web application

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

5. Add an x-tenant header.

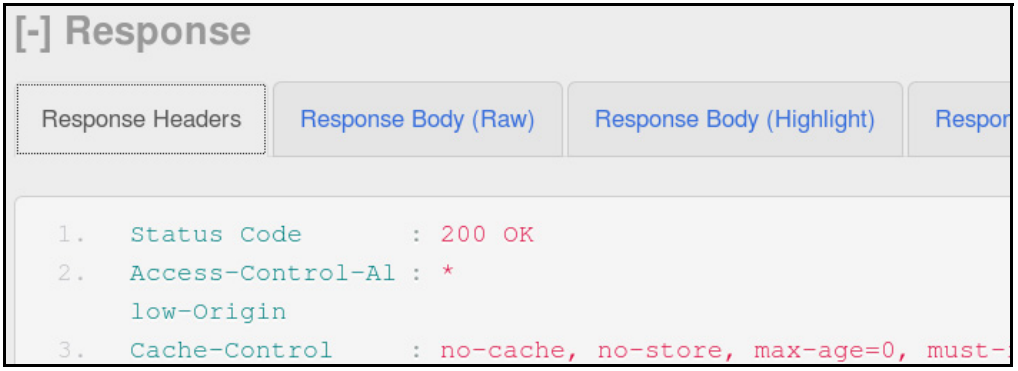
The x-tenant header tells the server which tenant (customer) is authenticating so that only users from that space may access the application.

- In the Headers drop-down, select **Custom Header**
- Enter the name: **`x-tenant`**
- Enter the value: **`55b09b29-6036-4501-8ff4-83f6b2807412`**
- Click **Okay**; a third header is added

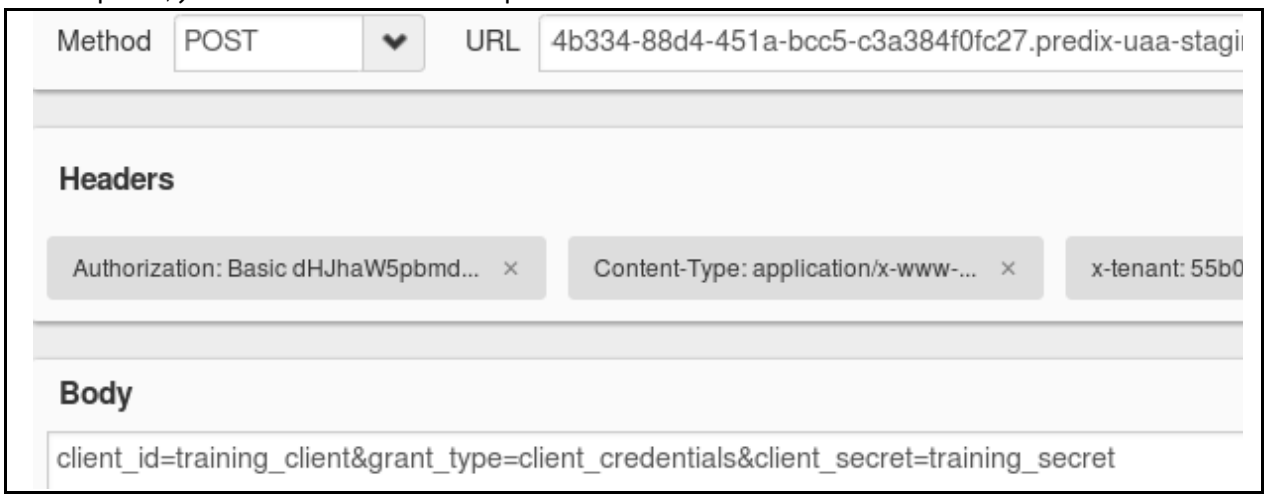
6. Construct the body of the message.

In this section of the POST, you indicate additional message parameters, such as the client's ID.

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

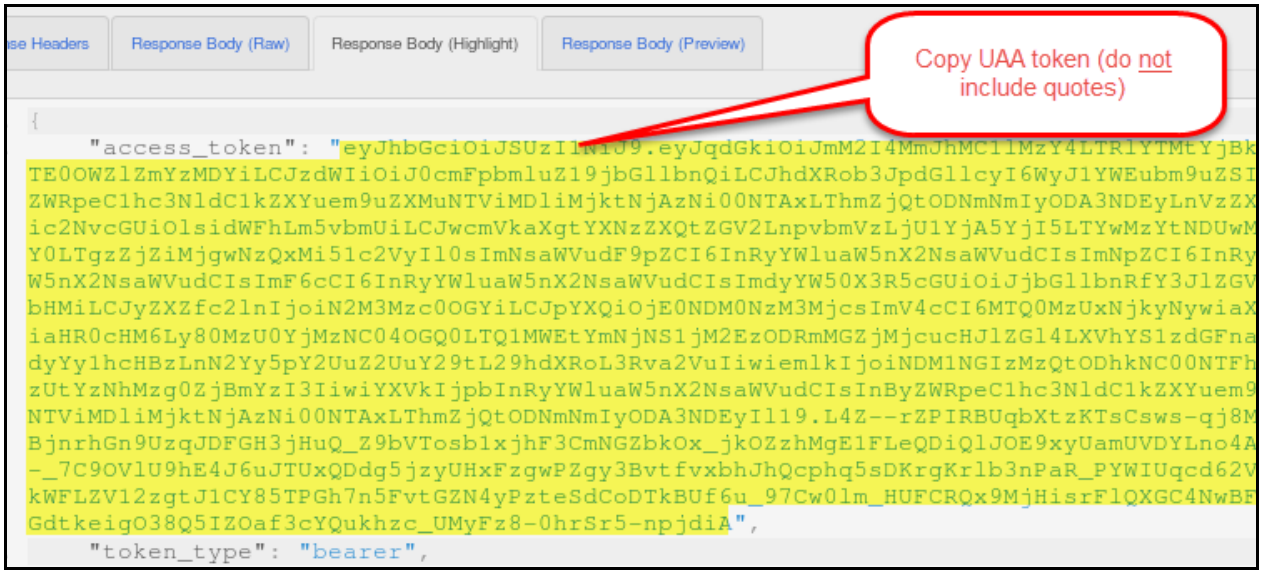


At this point, you have built a POST request that looks like this:



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

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



Exercise 2: Retrieving Data from the Asset Service

Overview

In this exercise you use the (bearer) access token you retrieved in the prior lab. With that token, you create a GET request and retrieve data from a database, again using the Asset service instance. The database contains asset data for locomotives, engines, fleets and manufacturers.

Steps

1. Create a **GET** request for fleet (locomotive) objects.

- Open a new tab in Firebox and open the REST client
- In the REST client set the Method and URL settings
  - Method: **GET**
  - URL: **`http://predix-asset-rc.grc-apps.svc.ice.ge.com/fleet`**
- Add a Custom Header
  - Name: **`Content-Type`**
  - Value: **`application/json`**
- Add a second Custom Header
  - Name: **`Authorization`**
  - Value: **`Bearer`** *<paste the Token returned by UAA>*

- Add a third Custom Header
  - Name: **`x-tenant`**
  - Value: **`55b09b29-6036-4501-8ff4-83f6b2807412`**

    Your three headers should appear as below

- **Send** the request
- Verify a status code of **200 OK** is returned in the *Response Headers* tab
- Select any *Response Body* tab to view the data returned on fleet assets


```
[
    {
        "uri": "/fleet/bnsf-1",
        "name": "Burlington Northern Fleet 1",
        "customer": "/customer/burlington-northern-santa-fe"
    },
    {
        "uri": "/fleet/bnsf-2",
        "name": "Burlington Northern Fleet 2",
        "customer": "/customer/burlington-northern-santa-fe"
    },
    {
        "uri": "/fleet/bnsf-3",
        "name": "Burlington Northern Fleet 3",
        "customer": "/customer/burlington-northern-santa-fe"
    },
    {
        "uri": "/fleet/bnsf-4",
        "name": "Burlington Northern Fleet 4",
        "customer": "/customer/burlington-northern-santa-fe"
    },
```

Lab 4: Predix Experience

Learning Objectives

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

- Build a UI microservice using the dashboard seed
- Package the microservice for deployment
- Push the microservice to the Predix Cloud

Lab Exercises

Exercise 1: Updating UI microservice

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.

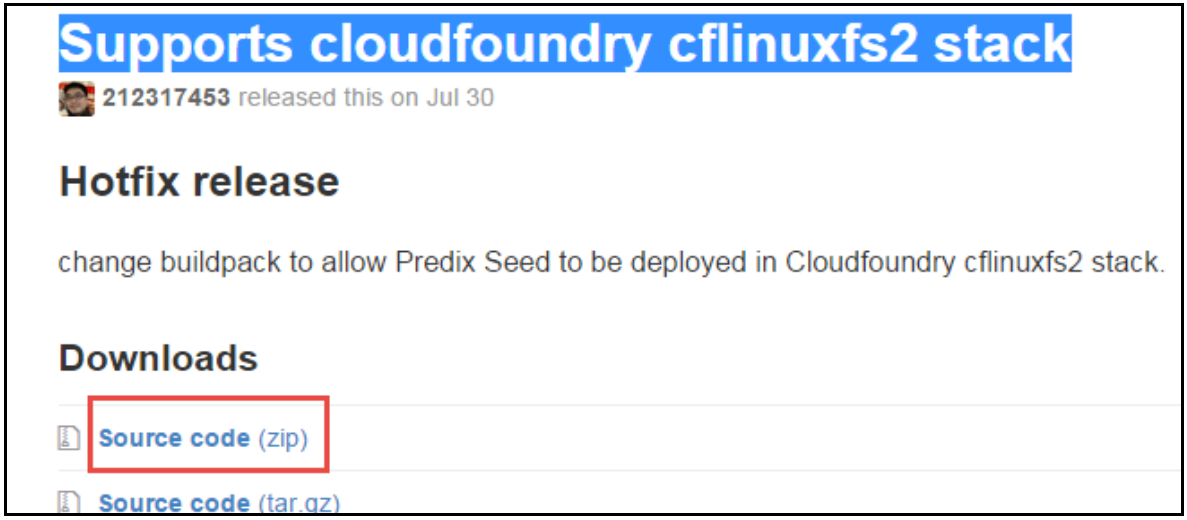
Steps

1. Downloading the Dashboard Seed app.

- Open firefox and go to this web address:

https://github.build.ge.com/Predix-Experience/predix-seed/releases

  - Under "Supports cloudfoundry cflinuxfs2 stack", click **Source Code** to download the file



- After the file is downloaded, in your Terminal, run this command:
  `cd ~/Downloads`
- Unzip the predix-seed-1.1.3.zip into your working directory by running this command:

`unzip predix-seed-1.1.3.zip -d ~/PredixApps/training/fundamentals/`

- Change directories to your working directory by running this command:
  `cd ~/PredixApps/training/fundamentals/predix-seed-1.1.3`

2. Loading npm and bower dependencies.

- In your Terminal, run these commands:
  `npm install`
  `bower install`

Note: These dependencies are required for your application.

3. Downloading a table design from Github.

- To get the design for a px table from github, in a Terminal run this command:

`bower install https://github.build.ge.com/PXd/px-tables-design.git --save`

Note: You will use this table design to display Alarm data.

4. Importing a table design into the project.

- In gedit, open the file /PredixApps/training/fundamentals/predix-seed-1.1.3/public/bower_components/px-defaults-design/***_settings.defaults.scss***
- Add the following code to the file:

```
$inuit-enable-table--fixed:true;
@import "px-tables-design/_base.tables.scss";
```

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

5. Editing an existing card to display alarm data.

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

- Open file **temperature-script.txt** under /predix/PredixApps/training_labes/Lab_4
  - 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

```
<script>
  Polymer({
        is: 'temperature-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/data-control.js
          **/
          var self = this;
          this.getData(this.context.hospitalurl).then(function(data) {
                // following data structure from http://api.wunderground.com/
                // alert(data.name);
                self.hospitalName = data.name;
                self.hospitalAddress = data.address;
                self.hospitalPhone = data.phone;
                self.hospitalEmail = data.email;
          }, function(reason) {
                // on rejection
                console.error('ERROR', reason);
          });
        },
        behaviors: [px.card]
  });
</script>
```

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

Note: A card is a composable, interactive user interface module that can be shared across different environments. Cards are editable containers that reside within the content area of the screen.

6. Editing an existing card to display hospital data.

- Open the file /predix/PredixApps/training_labes/Lab_4/***fetch-data-card-template.txt***
  - Copy all contents of the file
- In gedit, open the file /PredixApps/training/fundamentals/predix-seed-1.1.3/public/bower_components/px-sample-cards/***fetch-data-card.html***
- In your **fetch-data-card.html** file, replace the <**template**><**/template**> tags and everything within the tags with the content you just copied

```
data-control.js    fetch-data-card.html
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}}</t
                                <td>{{ item.patient.firstName}}</td>
                                <td>{{ item.patient.lastName}}</td>
                                <td>{{ item.patient.email}}</td>
                                <td>{{ item.priority}}</td>
                                <td>{{ item.numberOfAlarms}}</td>
                              </tr>
                      </template>
                </table>
              </div>
        </div>
      </px-card>
</template>
```

- Open file **fetch-data-card-script.txt** under /predix/PredixApps/training_labes/Lab_4
  - Copy all contents of the file

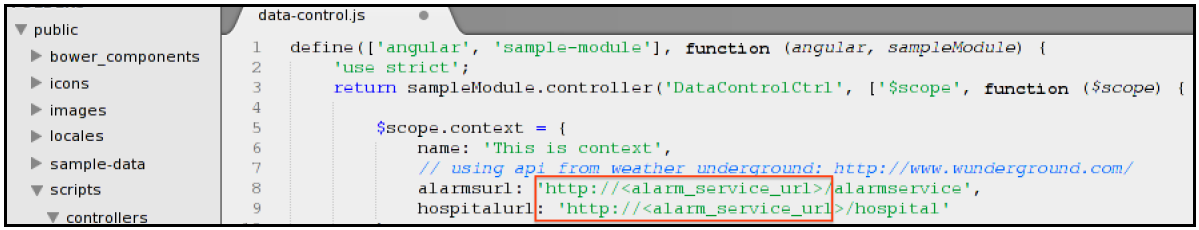
- In your **fetch-data-card.html** file, replace the **<script>…</script>** tag and everything within the tag with the content you just copied

```
<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/data-control.js
          **/
          var self = this;
          this.getData(this.context.alarmsurl).then(function(data) {
          // following data structure from http://api.wunderground.com/
          //self.currentTemperature = data['current_observation']['temp_f'];
          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

7. Connecting UI to the microservices.

- Open the file /predix/PredixApps/training_labes/Lab4/***scope.txt***
  - Copy all contents of the file
- In gedit, open the file /PredixApps/training/fundamentals/predix-seed-1.1.3/public/scripts/controllers/***data-control.js***
  - Replace the entire content you just copied



- Replace <alarm_service_url> with the url of you alarmservice published in lab 2

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

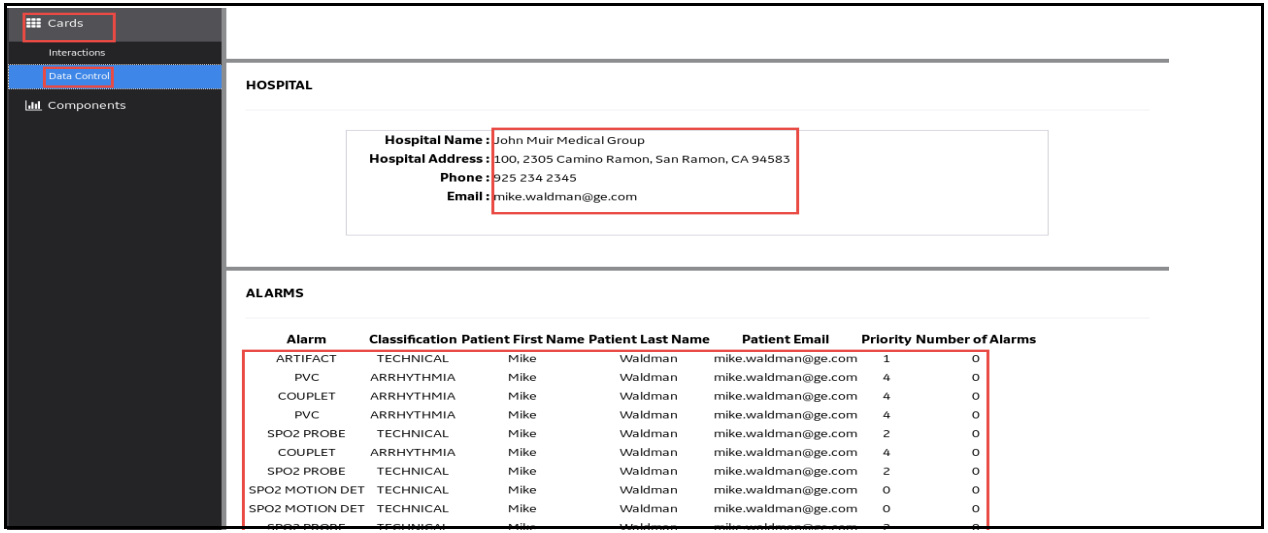
- In your Terminal run this command:

  `cf a`
- Locate your microservice name under the "name" column and find the url to the right under the "urls" column

8. Testing your microservice locally.

- To start your local server, in your Terminal, run this command:

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



- In your Terminal, press <ctrl> + <c> to stop running your local server

Note: The grunt commands allows you to test you microservice locally before deploying it to CF.

9. Deploying the microservice to Cloud Foundry.

- Create a package for deployment
  - In your Terminal, run this command:

```
grunt dist
```

- Update the manifest file
  - In gedit, open the file *PredixApps/training/fundamentals/predix-seed-1.1.3/**manifest.yml***
  - Append your first and last name to the microservice name

```
applications:
  - name: predix-seed-dev-YourName
    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 CF
  - In your terminal run this command:

```
cf push
```

```
App started

OK

App predix-seed-dev-YourName was started using this command `bash boot.sh`

Showing health and status for app predix-seed-dev-YourName in org predix-adoption
OK

requested state: started
instances: 1/1
usage: 64M x 1 instances
urls: predix-seed-dev-yourname.grc-apps.svc.ice.ge.com
last uploaded: Tue Jul 21 00:36:28 UTC 2015

     state     since                    cpu    memory         disk       details
#0   running   2015-07-20 05:36:27 PM   0.0%   4.3M of 64M    8M of 1G
```

Verify the microservice is in the running state

10. Testing your microservice.

- In Firefox, input your microservice url in the web address

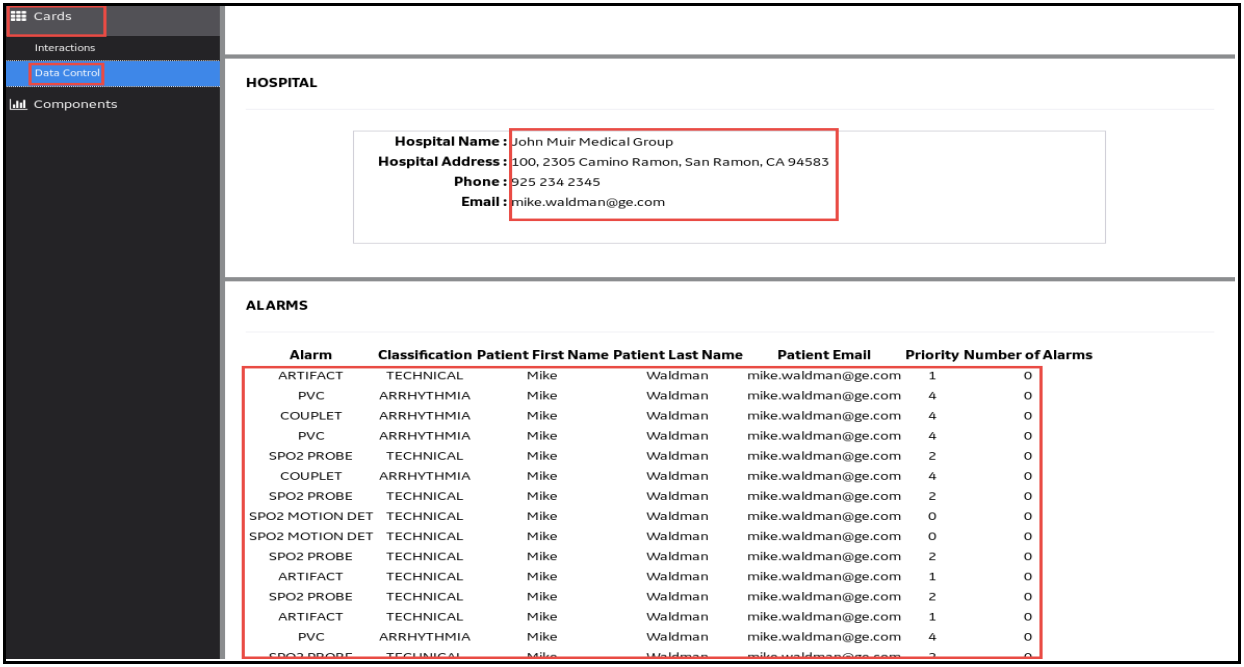
Tip - If you do not remember your microservice url, follow the steps below:

- In your Terminal run this command:

  ```
  cf a
  ```

- Locate your microservice name under the "name" column and find the url to the right under the "urls" column

- On the left navigation pane select Cards, then select Data Control
  The Hospital and Alarm data appears



You have successfully modified the two card dashboard seeds to display the hospital and alarm data.