

Predix

Working with Analytics

Student Lab Guide

January 2016



GE Digital

Predix

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 Software, 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 applications on the Predix platform.

Course Prerequisites:

- Have a Cloud Foundry account
- Install the most recent DevBox version

Start the DevBox in Oracle VirtualBox

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

Additional Course Downloads and Set Up:

The following instructions and notes may be modified by your instructor.

1. Download Postman at getpostman.com. You may be instructed to use Postman either in your development or local (PC/laptop) environment.
2. Your instructor may provide additional instructions for downloading additional course files (e.g. zip, application, Postman files)



Lab 1: Your Dev Environment and UAA

Learning Objectives

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

- Create the UAA service instance
- Create and work with the Analytics Catalog service instance
- Bind an application to UAA and Analytics Catalog service instances
- Configure the UAA service instance for authorization and access

Lab Exercises

- *Create Your UAA Service Instance*, page 2
- *Bind Your UAA Service Instance to an Application*, page 7
- *Create and Bind an Analytics Catalog Service Instance*, page 14
- *Create and Bind an Analytics Runtime Service Instance*, page 18
- *Use the UAA Service*, page 22



Exercise 1: Create Your UAA Service Instance

Overview

In this lab exercise, you will create a UAA service instance and bind a simple Hello-World application to it.

Steps

1. Log into Cloud Foundry (CF).

In the Terminal, run the command:

```
cf login -a <API Endpoint provided by your instructor>
```



A terminal window titled "predix@localhost:~". The session starts with "File Edit View Search Terminal Help" menu. The user runs "cf login" and is prompted for an API endpoint, which is set to "https://api.system.aws-usw02-pr.ice.predix.io". The user enters their email "jaime.villanueva@ge.com" and password. The terminal shows "Authenticating..." followed by "OK". It then targets the org "jaime.villanueva@ge.com". The user is prompted to select a space, choosing option 1 ("dev"). The space is targeted successfully. Finally, the user's details are displayed: API endpoint, User, Org, and Space, all set to "dev".

```
[predix@localhost ~]$ cf login
API endpoint: https://api.system.aws-usw02-pr.ice.predix.io
Email> jaime.villanueva@ge.com
Password>
Authenticating...
OK
Targeted org jaime.villanueva@ge.com
Select a space (or press enter to skip):
1. dev
2. test
Space> 1
Targeted space dev

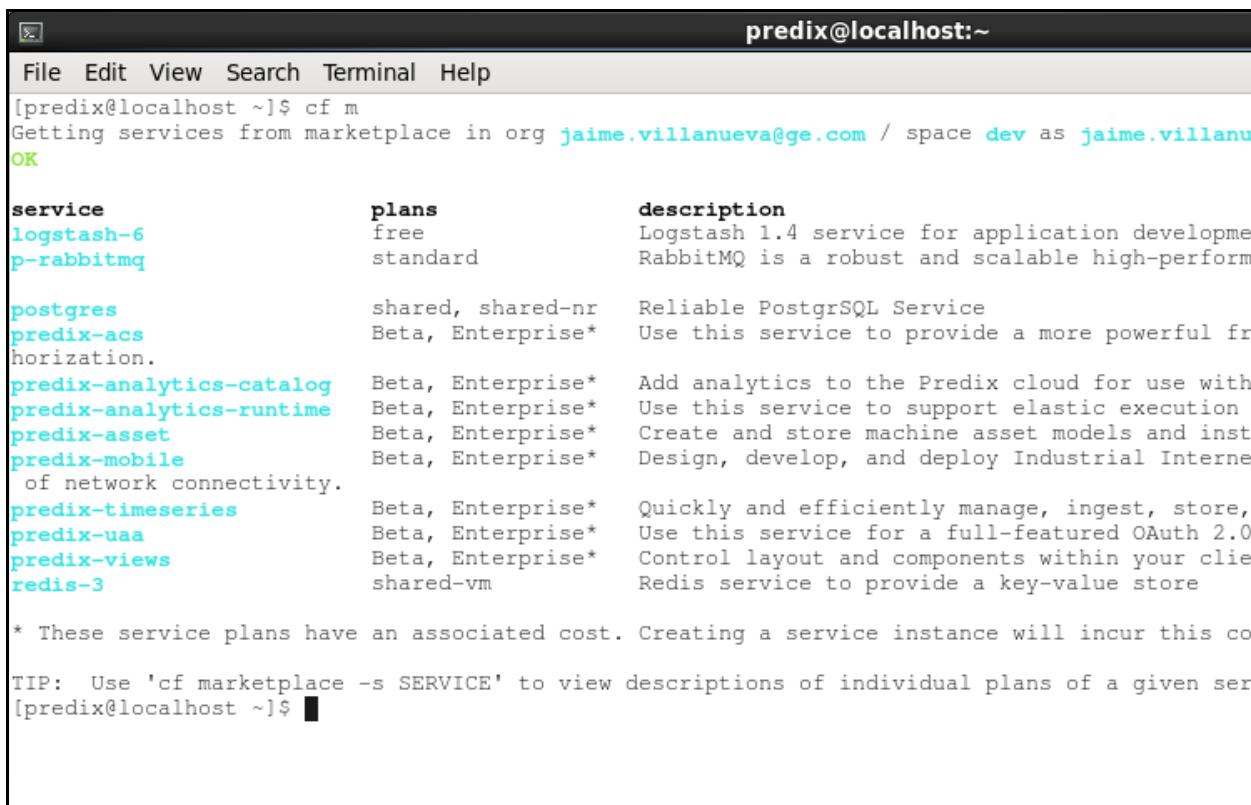
API endpoint: https://api.system.aws-usw02-pr.ice.predix.io (API version: 2.28.0)
User: jaime.villanueva@ge.com
Org: jaime.villanueva@ge.com
Space: dev
[predix@localhost ~]$
```

- login with the User ID and password provided by your instructor
- select a space (e.g. training1)

2. Display all services in the Predix marketplace.

- ◆ In the Terminal run the command:

```
cf marketplace (or cf m)
```



```
[predix@localhost ~]$ cf m
Getting services from marketplace in org jaime.villanueva@ge.com / space dev as jaime.villanueva@ge.com OK



| service                  | plans             | description                                                                           |
|--------------------------|-------------------|---------------------------------------------------------------------------------------|
| logstash-6               | free              | Logstash 1.4 service for application development                                      |
| p-rabbitmq               | standard          | RabbitMQ is a robust and scalable high-performance message broker.                    |
| postgres                 | shared, shared-nr | Reliable PostgreSQL Service                                                           |
| predix-acss              | Beta, Enterprise* | Use this service to provide a more powerful front-end for your application.           |
| predix-analytics-catalog | Beta, Enterprise* | Add analytics to the Predix cloud for use with your applications.                     |
| predix-analytics-runtime | Beta, Enterprise* | Use this service to support elastic execution environments.                           |
| predix-asset             | Beta, Enterprise* | Create and store machine asset models and instances.                                  |
| predix-mobile            | Beta, Enterprise* | Design, develop, and deploy Industrial Internet of Things (IIoT) mobile applications. |
| predix-timeseries        | Beta, Enterprise* | Quickly and efficiently manage, ingest, store, and analyze time-series data.          |
| predix-uaa               | Beta, Enterprise* | Use this service for a full-featured OAuth 2.0 provider.                              |
| predix-views             | Beta, Enterprise* | Control layout and components within your client application.                         |
| redis-3                  | shared-vm         | Redis service to provide a key-value store.                                           |



* These service plans have an associated cost. Creating a service instance will incur this cost.



TIP: Use 'cf marketplace -s SERVICE' to view descriptions of individual plans of a given service.



```
[predix@localhost ~]$ █
```


```

Ensure that you have the following services:

- predix-uaa
- predix-analytics-catalog
- predix-analytics-runtime

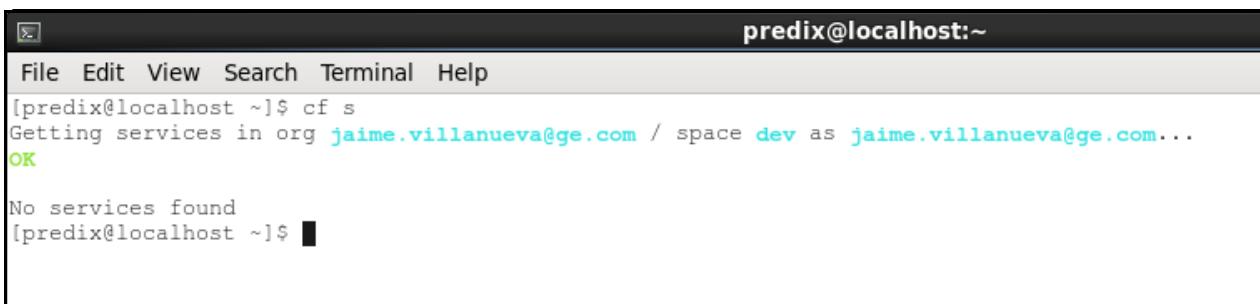
You will be creating your own service instances from these three marketplace services.



3. View a listing of service instances in your environment.

- ◆ In the Terminal run the command:

```
cf services (or cf s)
```



A terminal window titled "predix@localhost:~". The window shows the command "cf services" being run, followed by its output: "Getting services in org jaime.villanueva@ge.com / space dev as jaime.villanueva@ge.com... OK". It then displays "No services found".

```
[predix@localhost ~]$ cf s
Getting services in org jaime.villanueva@ge.com / space dev as jaime.villanueva@ge.com...
OK

No services found
[predix@localhost ~]$
```

There may be some or no service instances listed as we have yet to create three for our Analytics to run.

4. Create a UAA Service Instance.

- In the Terminal run the `following` command:

```
cf create-service <service> <plan> <my_uaa_instance> -c
'{"adminClientSecret":"<my_secret>"}'
```

where:

- <service> the UAA service name (e.g. predix-uaa)
- <plan> - is the plan associated (e.g. Beta) with the service
- <service_instance> is **predix-uaa-<FirstinitialLastinitial>**
- <my_secret> This will be the password for the default "admin" account that will be created. Use **myadminsecret**
- **(DO NOT USE a different password as other activities rely on this value.)**

For example:

```
cf create-service predix-uaa Beta predix-uaa-JaimeVillanueva -c
  '{"adminClientSecret":"myadmininsecret"}'
```

You should receive the following response.



The screenshot shows a terminal window titled "predix@localhost:~". The user has run the command "cf create-service predix-uaa Beta predix-uaa-JaimeVillanueva -c '{"adminClientSecret":"myadmininsecret"}'". The output shows the marketplace services available, the creation of the "predix-uaa-JaimeVillanueva" service instance, and its status.

```
[predix@localhost ~]$ cf m
Getting services from marketplace in org jaime.villanueva@ge.com / space dev as jaime.villanueva@ge.com
OK

service           plans          description
logstash-6       free          Logstash 1.4 service for application development
p-rabbitmq        standard      RabbitMQ is a robust and scalable high-performance
                               messaging system
postgres          shared, shared-nr Reliable PostgreSQL Service
predix-acs         Beta, Enterprise* Use this service to provide a more powerful authorization.
predix-analytics-catalog Beta, Enterprise* Add analytics to the Predix cloud for use with
predix-analytics-runtime Beta, Enterprise* Use this service to support elastic execution
predix-asset        Beta, Enterprise* Create and store machine asset models and instances
predix-mobile       Beta, Enterprise* Design, develop, and deploy Industrial Internet of Things applications
of network connectivity.
predix-timeseries  Beta, Enterprise* Quickly and efficiently manage, ingest, store, and analyze time-series data
predix-uaa          Beta, Enterprise* Use this service for a full-featured OAuth 2.0 authentication and authorization
predix-views         Beta, Enterprise* Control layout and components within your client application
redis-3            shared-vm      Redis service to provide a key-value store

* These service plans have an associated cost. Creating a service instance will incur this cost.

TIP: Use 'cf marketplace -s SERVICE' to view descriptions of individual plans of a given service.

[predix@localhost ~]$ cf create-service predix-uaa Beta predix-uaa-JaimeVillanueva -c '{"adminClientSecret":"myadmininsecret"}'
Creating service instance predix-uaa-JaimeVillanueva in org jaime.villanueva@ge.com / space dev as jaime.villanueva@ge.com...
OK
[predix@localhost ~]$ cf s
Getting services in org jaime.villanueva@ge.com / space dev as jaime.villanueva@ge.com...
OK

name              service    plan   bound apps  last operation
predix-uaa-JaimeVillanueva  predix-uaa  Beta          create succeeded
[predix@localhost ~]$
```

- Your UAA instance is created (with the following specification):
 - ◆ A client, "admin"



- ◆ **Note:** An admin client is created for bootstrap purpose. You can create additional clients to use with your application¹.
- ◆ A client secret (that you specified while creating the service instance e.g."myadminsecret").
- ◆ Running the cf services (or cf s) command will list your service instances.

1. The UAA service allows administration of users, groups and their authorization and access. Refer to Predix.io documentation for additional information or help on this topic.

Exercise 2: Bind Your UAA Service Instance to an Application

Overview

In this exercise you will clone the Hello World Starter Pack to create your first Predix application, push it to the cloud, and watch it run. You will be binding your UAA service instance to this application.

Binding an application to our UAA service instance will allow you to view various environment variables.

Steps

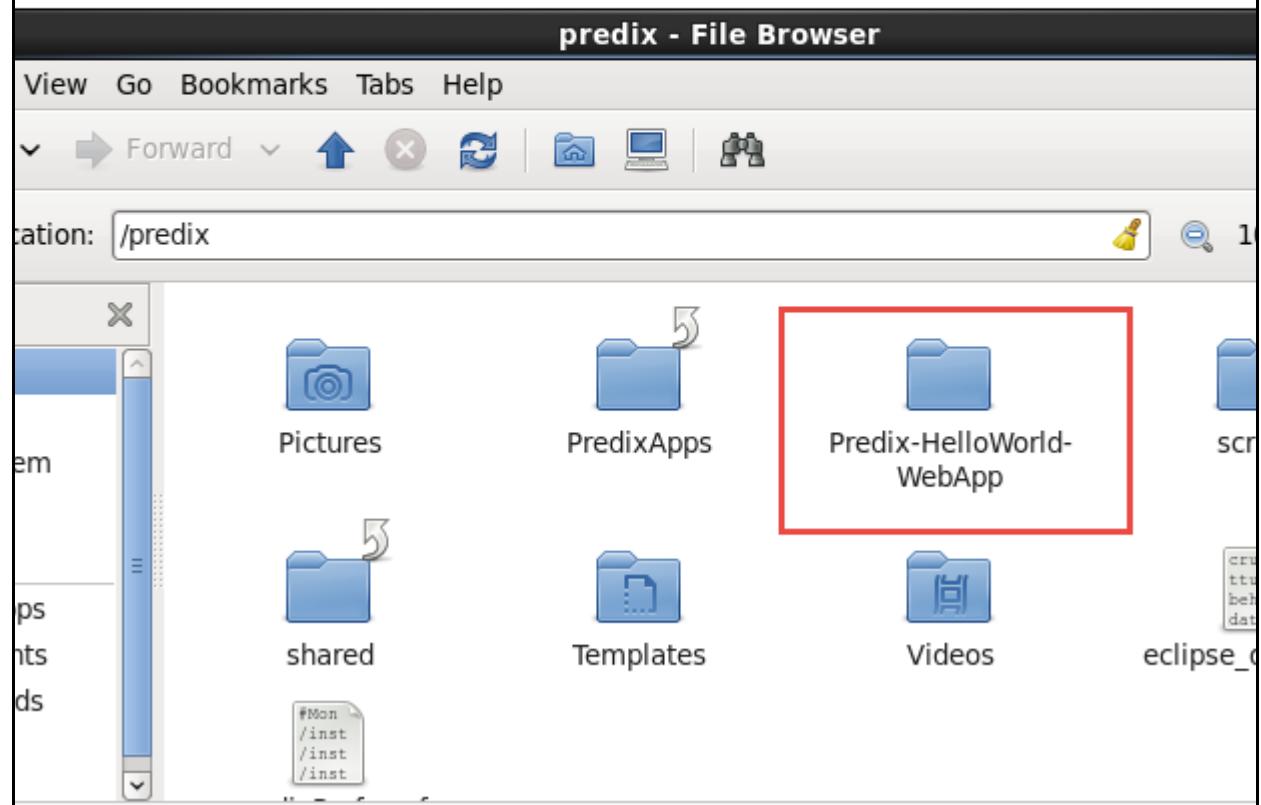
1. Get the Predix Hello World Starter Pack

- From github (This github team site requires authorized access.):
`git clone https://github.com/PredixDev/Predix-HelloWorld-WebApp`
- **Alternate:** Your development environment may already include this folder and contents at **/predix/Predix-HelloWorld-WebApp**.



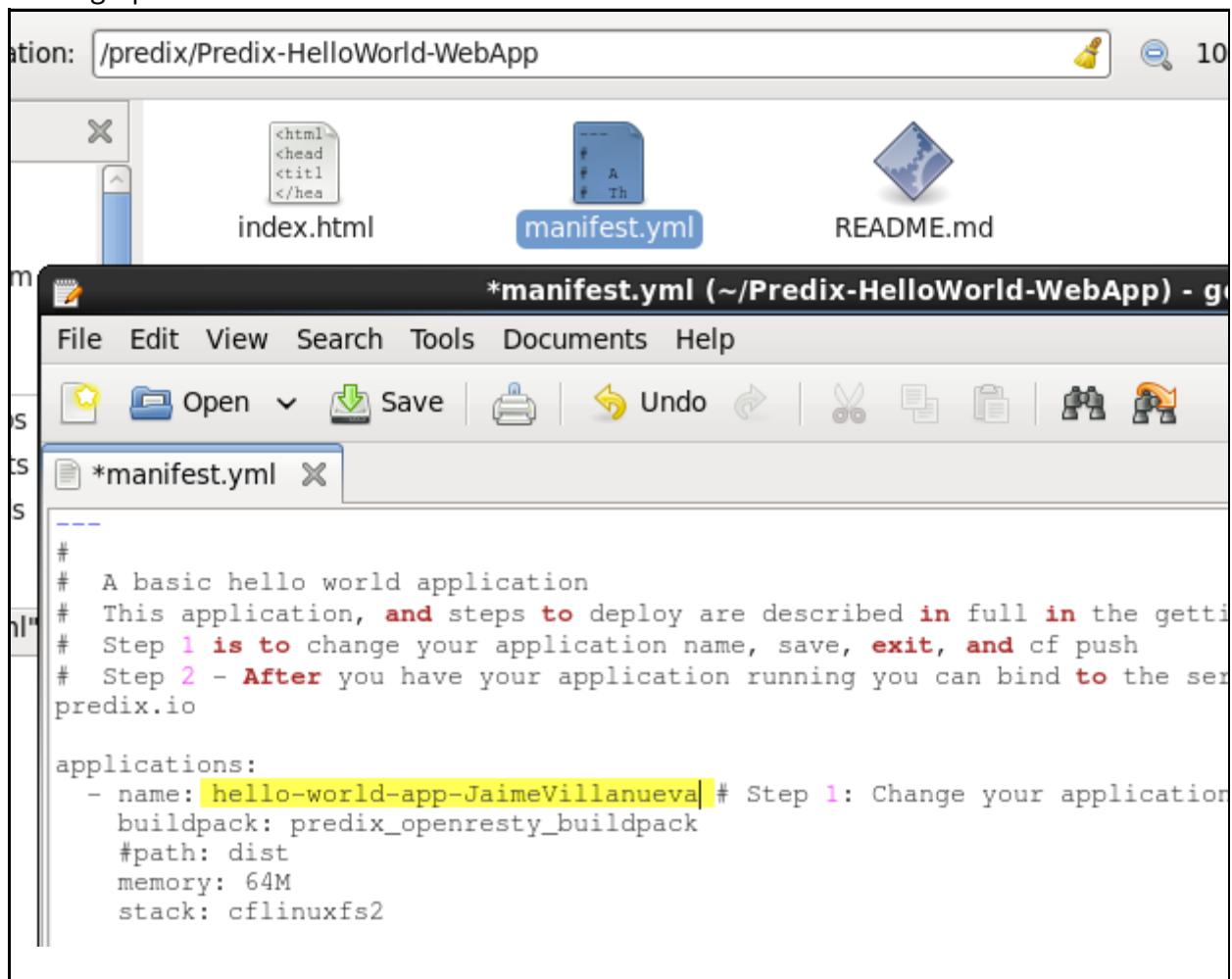
Working with Analytics

```
ix@localhost ~]$ git clone https://github.com/PredixDev/Predix-HelloWorld-WebA  
ng into 'Predix-HelloWorld-WebApp'...  
e: Counting objects: 7, done.  
e: Total 7 (delta 0), reused 0 (delta 0), pack-reused 7  
king objects: 100% (7/7), done.  
ing connectivity... done.  
ix@localhost ~]$
```



2. Using the gedit application, edit the manifest.yml file

- Edit the manifest.yml file in the Predix-HelloWorld-WebApp directory to update the name of the application to HelloWorld-<FirstinitialLastinitial>. For example see following graphic:



3. Push your Predix application to the cloud.

- Make sure that you are in the Predix-HelloWorld-WebApp directory (`cd /predix/Predix-HelloWorld-WebApp`).
- Use the Cloud Foundry CLI to push your Predix application to the cloud. For example:
`cf push hello-world-app-<FirstInitialLastInitial>`

The command produces the following final output:

```
Starting app hello-world-app-JaimeVillanueva in org jaime.villanueva@ge.com / space dev
----> Downloaded app package (4.0K)
-----> Buildpack version 1.2.1
grep: Staticfile: No such file or directory
-----> Using root folder
-----> Copying project files into public/
-----> Setting up nginx
grep: Staticfile: No such file or directory
-----> Uploading droplet (11M)

1 of 1 instances running

App started

OK

App hello-world-app-JaimeVillanueva was started using this command `sh boot.sh`

Showing health and status for app hello-world-app-JaimeVillanueva in org jaime.villanueva@ge.com...
OK

requested state: started
instances: 1/1
usage: 64M x 1 instances
urls: hello-world-app-jaimevillanueva.run.aws-usw02-pr.ice.predix.io
last uploaded: Thu Dec 17 23:47:13 UTC 2015
stack: cflinuxfs2
buildpack: predix_openresty_buildpack

state      since          cpu    memory      disk      details
#0  running   2015-12-17 03:47:28 PM  0.0%  10.8M of 64M  26.7M of 1G
[predix@localhost Predix-HelloWorld-WebApp]$
```

4. Verify your application is available and running in your space.

- Use the Cloud Foundry CLI to see your application in Cloud Foundry:

```
cf apps (or cf a)
```

- To view your app on the web, enter the application URL into your browser. For example:

```
hello-world-app-<FirstinitialLastinitial>.run.aws-usw02-pr.ice.pre  
dix.io
```

```
App started

OK

App hello-world-app-JaimeVillanueva was started using this command `sh boot.s

Showing health and status for app hello-world-app-JaimeVillanueva in org jaim
com...
OK

requested state: started
instances: 1/1
usage: 64M x 1 instances
urls: hello-world-app-jaimenvillanueva.run.aws-usw02-pr.ice.predix.io
last uploaded: Thu Dec 17 23:47:13 UTC 2015
stack: cflinuxfs2
buildpack: predix_openresty_buildpack

      state      since          cpu      memory      disk
#0  running   2015-12-17 03:47:28 PM  0.0%  10.8M of 64M  26.7M of 1G
[predix@localhost Predix-HelloWorld-WebApp]$
```



5. Bind your application to your UAA service instance.

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

For example:

```
cf bind-service hello-world-app-JaimeVillanueva  
predix-analytics-runtime-JaimeVillanueva
```

```
predix@localhost:~/Predix-HelloWorld-WebApp  
inal Help  
HelloWorld-WebApp]$ cf s  
me.villanueva@ge.com / space dev as jaime.villanueva@ge.com...  
  
service      plan    bound apps   last operation  
predix-uaa   Beta          create succeeded  
HelloWorld-WebApp]$ cf a  
illanueva@ge.com / space dev as jaime.villanueva@ge.com...  
  
requested state  instances   memory   disk    urls  
ueva  started        1/1       64M     1G     hello-world-app-jaimenvil  
HelloWorld-WebApp]$ cf bs hello-world-app-JaimeVillanueva predix-uaa-JaimeVilla  
JaimeVillanueva to app hello-world-app-JaimeVillanueva in org jaime.villanuev  
world-app-JaimeVillanueva' to ensure your env variable changes take effect  
HelloWorld-WebApp]$ █
```

6. Show all env variables for the app bound to your UAA service instance.

```
cf env <appname>
```

The screenshot shows a terminal window titled "predix@localhost:~/Predix-HelloWorld". The window has a menu bar with File, Edit, View, Search, Terminal, and Help. The terminal output is as follows:

```
[predix@localhost Predix-HelloWorld-WebApp]$ cf env hello-world-app-JaimeVillanueva
Getting env variables for app hello-world-app-JaimeVillanueva in org jaime.villanueva.com
..
OK

System-Provided:
{
  "VCAP_SERVICES": {
    "predix-uaa": [
      {
        "credentials": {
          "issuerId": "https://58a17288-7004-4aff-bb37-88ed04c06c97.predix-uaa.run.aws-us-east-1.amazonaws.com",
          "uri": "https://58a17288-7004-4aff-bb37-88ed04c06c97.predix-uaa.run.aws-us-east-1.amazonaws.com",
          "zone": {
            "http-header-name": "X-Identity-Zone-Id",
            "http-header-value": "58a17288-7004-4aff-bb37-88ed04c06c97"
          }
        }
      }
    ]
  }
}
```

For your predix-uaa service instance, note the following values for reference (i.e. use gedit to copy and paste the environment variables):

- issuerID
- uri
- http-header-value

Using the gedit application in your devbox, copy these values for reference.



Exercise 3: 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
logstash-6	free	Logstash 1.4 service for appli
p-rabbitmq	standard	RabbitMQ is a robust and scalal
postgres	shared, shared-nr	Reliable PostgreSQL Service
predix-acs	Beta, Enterprise*	Use this service to provide a r
horization.		
predix-analytics-catalog	Beta, Enterprise*	Add analytics to the Predix cl
predix-analytics-runtime	Beta, Enterprise*	Use this service to support el
predix-asset	Beta, Enterprise*	Create and store machine asset
predix-mobile	Beta, Enterprise*	Design, develop, and deploy In
of network connectivity.		
predix-timeseries	Beta, Enterprise*	Quickly and efficiently manage
predix-uaa	Beta, Enterprise*	Use this service for a full-fe
predix-views	Beta, Enterprise*	Control layout and components
redis-3	shared-vm	Redis service to provide a key

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

2. Create your catalog service instance.

```
cf create-service predix-analytics-catalog <plan>
<my_service_instance> -c
'{"trustedIssuerIds": ["<uaa_instance1_issuerId>",
"<uaa_instance2_issuerID>"]}'
```

where:

- <plan> is the plan associated with a service.
- <my_service_instance> is the name of your service instance.
- <uaa_instance_issuerId> is the issuerId of your trusted issuer (refer to your gedit text file).

For example:

```
cf create-service predix-analytics-catalog Beta
predix-analytics-catalog-<FirstinitialLastinitial> -c
'{"trustedIssuerIds": ["https://58a17288-7004-4aff-bb37-88ed04c06c9
7.predix-uaa.run.aws-usw02-pr.ice.predix.io/oauth/token"]}'
```



```
predix@localhost:~/Predix-HelloWorld-WebApp
File Edit View Search Terminal Help
[predix@localhost Predix-HelloWorld-WebApp]$ cf create-service predix-analytics-catalog-JaimeVillanueva -c '{"trustedIssuerIds":["https://58a17288-7004-4aff-1aws-usw02-pr.ice.predix.io/oauth/token"]}'
Creating service instance predix-analytics-catalog-JaimeVillanueva in org jaime.villanueva as jaime.villanueva@ge.com...
OK
[predix@localhost Predix-HelloWorld-WebApp]$ cf s
Getting services in org jaime.villanueva@ge.com / space dev as jaime.villanueva@ge.com...
OK

name                                     service          plan
last operation
predix-analytics-catalog-JaimeVillanueva   predix-analytics-catalog    Beta
  create succeeded
predix-uaa-JaimeVillanueva                 predix-uaa          Beta
  create succeeded
[predix@localhost Predix-HelloWorld-WebApp]$
```

3. Bind an application to an Analytics Catalog Service Instance.

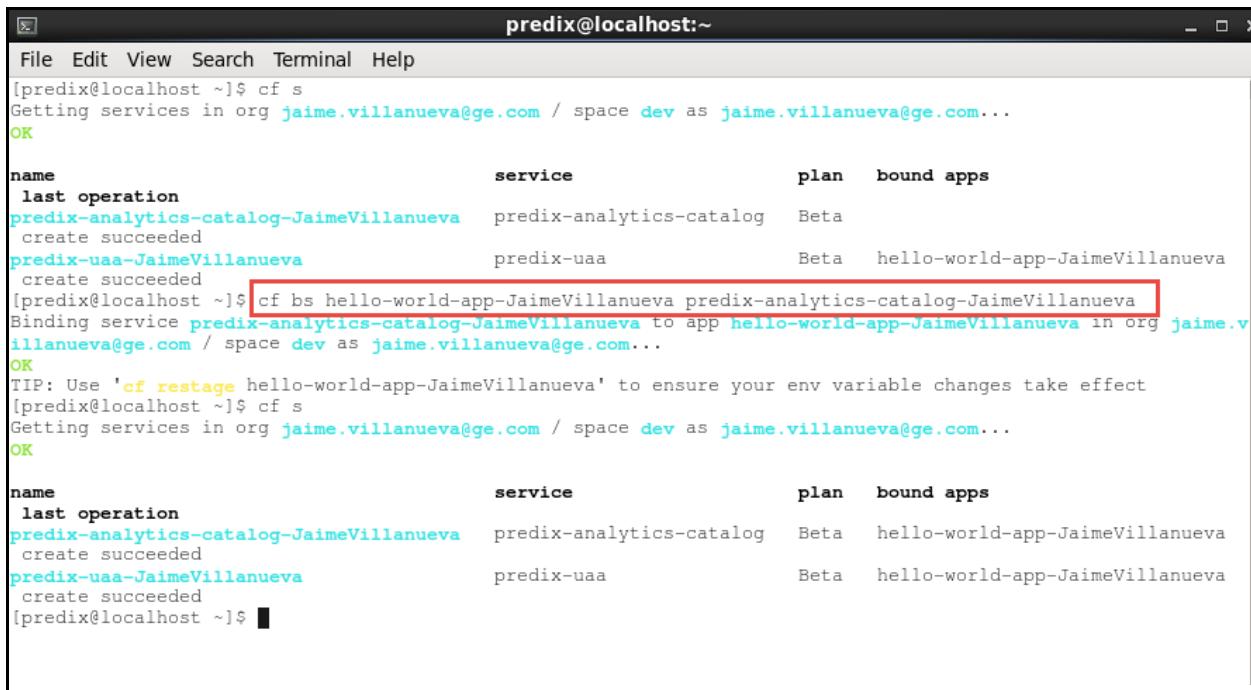
Note: You must bind your Analytics Catalog service instance to your application to provision connection details for your Analytics Catalog service instance in the VCAP_SERVICES environment variable

From your terminal:

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

For example:

```
cf bs hello-world-app-JaimeVillanueva
predix-analytics-catalog-JaimeVillanueva
```



The screenshot shows a terminal window titled "predix@localhost:~". The user has run the command "cf bind-service" to bind the "predix-analytics-catalog" service instance "JaimeVillanueva" to the "hello-world-app" app. The terminal output shows the creation of the service instance and its binding to the app.

```
[predix@localhost ~]$ cf s
Getting services in org jaime.villanueva@ge.com / space dev as jaime.villanueva@ge.com...
OK

name          service      plan  bound apps
last operation
predix-analytics-catalog-JaimeVillanueva predix-analytics-catalog  Beta
create succeeded
predix-uaa-JaimeVillanueva      predix-uaa           Beta  hello-world-app-JaimeVillanueva
create succeeded
[predix@localhost ~]$ cf bs hello-world-app-JaimeVillanueva predix-analytics-catalog-JaimeVillanueva
Binding service predix-analytics-catalog-JaimeVillanueva to app hello-world-app-JaimeVillanueva in org jaime.villanueva@ge.com / space dev as jaime.villanueva@ge.com...
OK
TIP: Use 'cf restage hello-world-app-JaimeVillanueva' to ensure your env variable changes take effect
[predix@localhost ~]$ cf s
Getting services in org jaime.villanueva@ge.com / space dev as jaime.villanueva@ge.com...
OK

name          service      plan  bound apps
last operation
predix-analytics-catalog-JaimeVillanueva predix-analytics-catalog  Beta  hello-world-app-JaimeVillanueva
create succeeded
predix-uaa-JaimeVillanueva      predix-uaa           Beta  hello-world-app-JaimeVillanueva
create succeeded
[predix@localhost ~]$
```



Exercise 4: Create and Bind an Analytics Runtime Service Instance

Overview

Before you begin, 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
logstash-6	free	Logstash 1.4 service for application monitoring.
p-rabbitmq	standard	RabbitMQ is a robust and scalable message broker.
postgres	shared, shared-nr	Reliable PostgreSQL Service
predix-acs	Beta, Enterprise*	Use this service to provide a secure authentication and authorization.
predix-analytics-catalog	Beta, Enterprise*	Add analytics to the Predix cloud.
predix-analytics-runtime	Beta, Enterprise*	Use this service to support end-to-end analytics.
predix-asset	Beta, Enterprise*	Create and store machine asset data.
predix-mobile	Beta, Enterprise*	Design, develop, and deploy Internet of things applications with network connectivity.

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

2. Create your Analytics runtime service instance, which will require two issuerids.

```
cf create-service predix-analytics-runtime <plan>
<my_service_instance> -c
'{"trustedIssuerIds": ["<uaa_instance1_issuerId>",
"<uaa_instance2_issuerID>"]}'
```

where:

- <plan> is the plan associated with a service.
- <my_service_instance> is the name of your service instance.
- <uaa_instance_issuerId> is the issuerId of your trusted issuer (refer to your gedit file).
- <uaa_instance2_issuerID>¹: **Instructor will provide this information at the time of class**

For example:

```
cf create-service predix-analytics-runtime Beta
predix-analytics-runtime-JaimeVillanueva -c
'{"trustedIssuerIds": ["https://58a17288-7004-4aff-bb37-88ed04c06c9
7.predix-uaa.run.aws-usw02-pr.ice.predix.io/oauth/token", "<uaa_in
stance2_issuerID>"]}'
```

1. This second trusted issuer ID is required to run the job scheduler. A "beta" URL will be provided by the instructor.



Working with Analytics

```
[predix@localhost ~]$ cf create-service predix-analytics-runtime Beta predix-analytics-runtime-J
--c '{"trustedIssuerIds":["https://58a17288-7004-4aff-bb37-88ed04c06c97.predix-uaa.run.aws-usw02
.io/oauth/token"]}'
Creating service instance predix-analytics-runtime-JaimeVillanueva in org jaime.villanueva@ge.co
as jaime.villanueva@ge.com...
OK
[predix@localhost ~]$ cf s
Getting services in org jaime.villanueva@ge.com / space dev as jaime.villanueva@ge.com...
OK

name                                     service          plan   bound apps
last operation
predix-analytics-catalog-JaimeVillanueva predix-analytics-catalog  Beta    hello-world-app-Jai
create succeeded
predix-analytics-runtime-JaimeVillanueva   predix-analytics-runtime  Beta
create succeeded
predix-uaa-JaimeVillanueva                predix-uaa            Beta    hello-world-app-Jai
create succeeded
[predix@localhost ~]$
```

3. Bind the Hello World application to your Analytics Runtime Service Instance.

You must bind your Analytics runtime service instance to your application to provision connection details for your Analytics Catalog service instance in the VCAP_SERVICES environment variable.

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

For example:

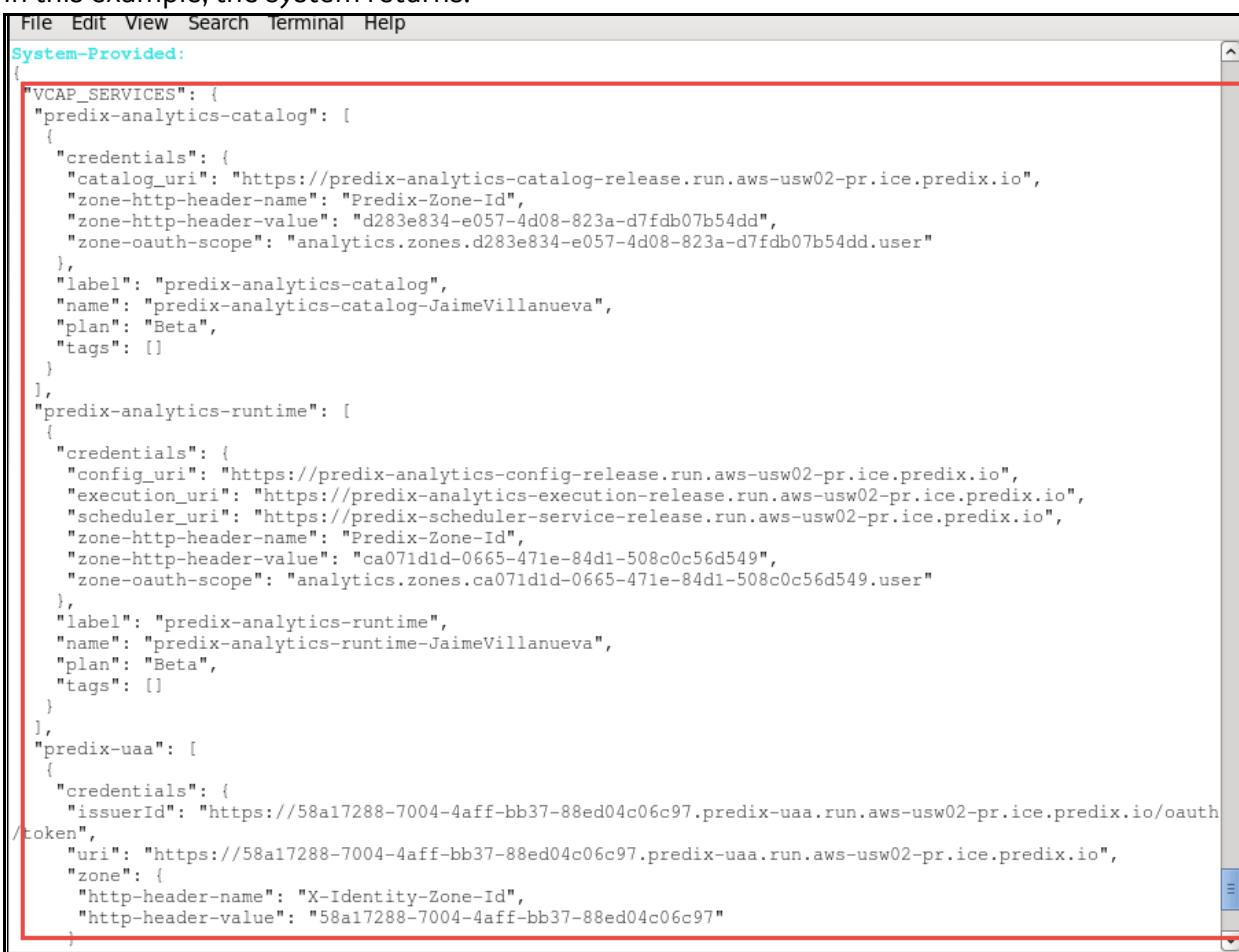
```
cf bs hello-world-app-JaimeVillanueva
predix-analytics-runtime-JaimeVillanueva
```

```
predix-uaa-JaimeVillanueva           predix-uaa          Beta    hello-world-app-JaimeVillanueva
create succeeded
[predix@localhost ~]$ cf bs hello-world-app-JaimeVillanueva predix-analytics-runtime-JaimeVillanueva
Binding service predix-analytics-runtime-JaimeVillanueva to app hello-world-app-JaimeVillanueva in org jaime.
villanueva@ge.com / space dev as jaime.villanueva@ge.com...
OK
TIP: Use 'cf restage hello-world-app-JaimeVillanueva' to ensure your env variable changes take effect
[predix@localhost ~]$ cf s
Getting services in org jaime.villanueva@ge.com / space dev as jaime.villanueva@ge.com...
OK
```

4. View and note some key environment variables.

```
cf env hello-world-app-JaimeVillanueva
```

In this example, the system returns:



```
File Edit View Search Terminal Help
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": "d283e834-e057-4d08-823a-d7fdb07b54dd",
          "zone-oauth-scope": "analytics.zones.d283e834-e057-4d08-823a-d7fdb07b54dd.user"
        },
        "label": "predix-analytics-catalog",
        "name": "predix-analytics-catalog-JaimeVillanueva",
        "plan": "Beta",
        "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",
          "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": "ca071d1d-0665-471e-84d1-508c0c56d549",
          "zone-oauth-scope": "analytics.zones.ca071d1d-0665-471e-84d1-508c0c56d549.user"
        },
        "label": "predix-analytics-runtime",
        "name": "predix-analytics-runtime-JaimeVillanueva",
        "plan": "Beta",
        "tags": []
      }
    ],
    "predix-uaa": [
      {
        "credentials": {
          "issuerId": "https://58a17288-7004-4aff-bb37-88ed04c06c97.predix-uaa.run.aws-usw02-pr.ice.predix.io/oauth/token",
          "uri": "https://58a17288-7004-4aff-bb37-88ed04c06c97.predix-uaa.run.aws-usw02-pr.ice.predix.io",
          "zone": {
            "http-header-name": "X-Identity-Zone-Id",
            "http-header-value": "58a17288-7004-4aff-bb37-88ed04c06c97"
          }
        }
      }
    ]
  }
}
```

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



Exercise 5: Use the UAA Service

Overview

Before you begin, 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

In this exercise, you will use the UAA service to set up your access and authorization scopes for your Analytics Catalog and Analytics Runtime service instances.

Steps

1. Target your UAA service instance.

You can use the UAA command-line interface (UAAC) to work with your UAA instance.¹

- From your terminal window, file>open tab and enter the following command:
`uaac target <uaa_instance_url>`

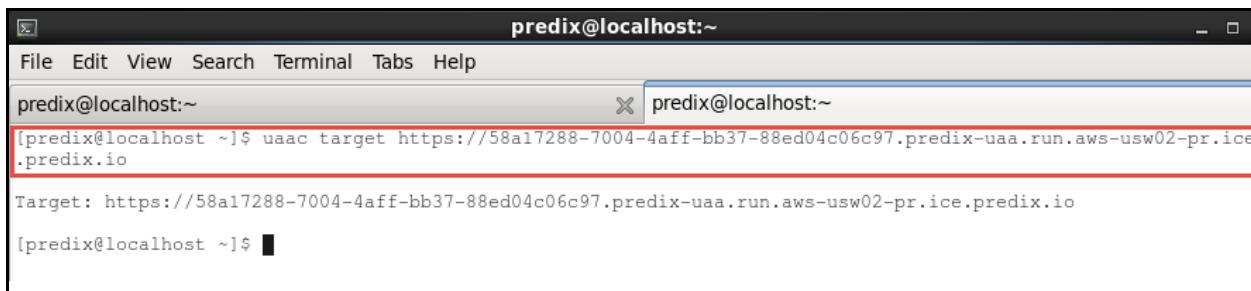
where:

<uaa_instance_url> is the URL to your trusted issuer (refer to your gedit tex file containing your Hello World environment variables).

For our example:

```
uaac target  
https://58a17288-7004-4aff-bb37-88ed04c06c97.predix-uaa.run.aws-us  
w02-pr.ice.predix.io
```

1. You may refer to Predix.io documentation for additional information on this topic.

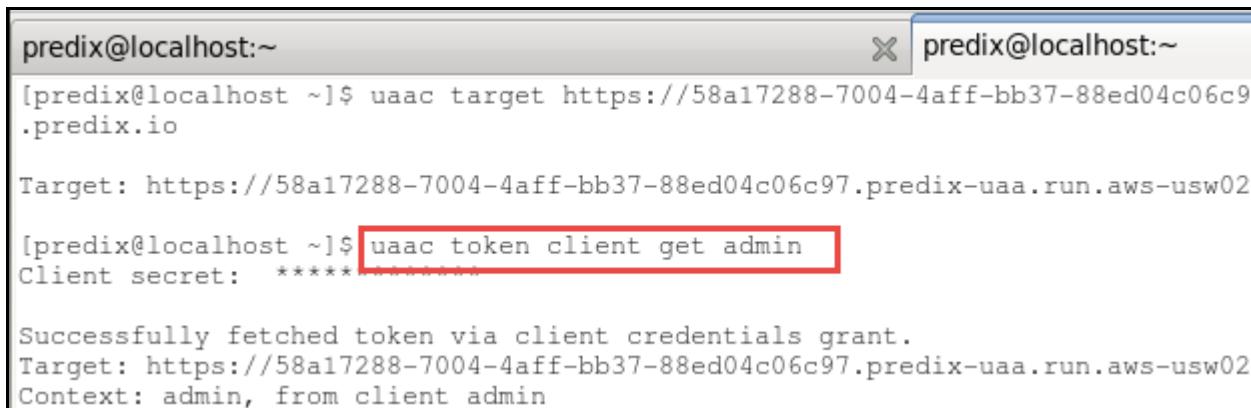


A terminal window titled "predix@localhost:~". The menu bar includes File, Edit, View, Search, Terminal, Tabs, and Help. There are two tabs open: "predix@localhost:~" and "predix@localhost:~". The second tab is active. The command "uaac target https://58a17288-7004-4aff-bb37-88ed04c06c97.predix-uaa.run.aws-usw02-pr.ice.predix.io" is entered in the active tab, and the output "Target: https://58a17288-7004-4aff-bb37-88ed04c06c97.predix-uaa.run.aws-usw02-pr.ice.predix.io" is displayed.

2. Log in to your UAA service instance.

Log in and verify your default admin account and password set up when you created your UAA service instance.

- From your terminal window and tab tagetting your UAAC instance:
`uaac token client get admin`
- Specify the administrative client secret (e.g. **myadminsecret**) at the prompt.



A terminal window titled "predix@localhost:~". The menu bar includes File, Edit, View, Search, Terminal, Tabs, and Help. There are two tabs open: "predix@localhost:~" and "predix@localhost:~". The second tab is active. The command "uaac token client get admin" is entered in the active tab, and the output "Client secret: *****" is displayed. A red box highlights the "Client secret" line.

This step validates your UAA instance, your admin client, and your admin password.

Exercise Summary

In this exercise you learned how to:

- Create the UAA service instance
- Create and work with the Analytics Catalog service instance
- Bind an application to UAA and Analytics Catalog service instances
- Configure the UAA service instance for authorization and access

Lab 2: Working with The API and Analytics Catalog

Learning Objectives

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 26
- *Updating the OAuth2 Client*, page 37
- *Working with the Analytics Catalog*, page 46



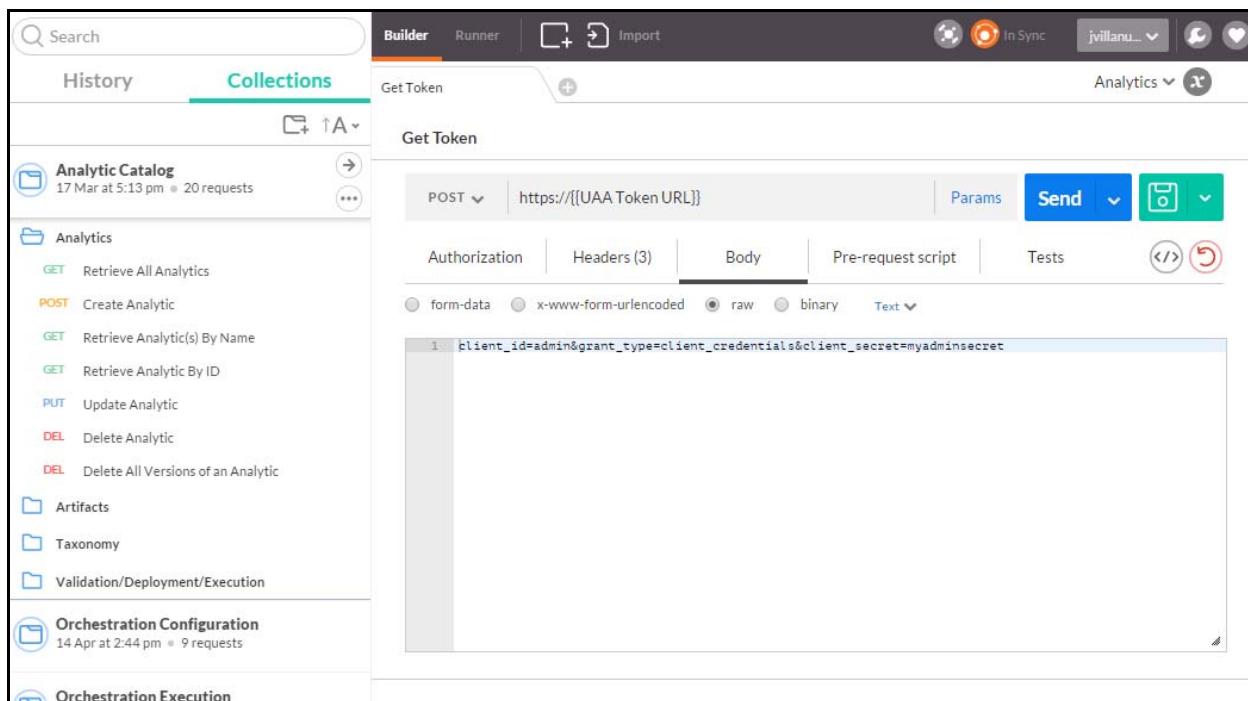
Exercise 1: Using Postman and Getting Your UAA Token Value

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.
- Your instructor may provide additional instructions on using Postman local on your computer or within your development environment.

A screenshot of Postman follows:



Lab 2: Working with The API and Analytics Catalog

Environment values may be viewed when managing the environments.

The screenshot shows the Postman interface with the 'Collections' tab selected. A modal window titled 'Manage environments' is open, specifically for the 'Analytics' environment. The modal lists various environment variables with their corresponding values:

Key	Value
UAA Token URL	58a17288-7004-4aff-bb37-88ed04cC
X-Identity-Zone-Id	58a17288-7004-4aff-bb37-88ed04cC
token	eyJhbGciOiJSUzI1NiJ9eyJqdGkiOiJt
catalog_host	predix-analytics-catalog-release.run.aw
catalog_tenant	d283e834-e057-4d08-823a-d7fdb07
Content-Type	application/json
analyticId	f39151f8-1480-42a7-9321-3c2e66bc
validationRequestId	546b4871-b72c-4b4f-82e5-57a6358
config_host	predix-analytics-config-release.run.aw
runtime_tenant	ca071d1d-0665-471e-84d1-508c0c5
exec_host	predix-analytics-execution-release.run.aw
scheduler_host	predix-scheduler-service-release.run.aw

You will configure environment variables and use Postman's Collections manager to facilitate HTTP requests to the UAA, Catalog and Runtime service instances.

The example above shows the variables set up for the Analytics environment. Variables are referenced using double curly bracket notation (ex: {{token}}) anywhere in the request (URL, params, header, body).



For example:

The screenshot shows the Postman interface with a request titled "Retrieve All Analytics". The method is set to "GET" and the URL is `{{{protocol}}}://{{{catalog_host}}}{{api/v1/catalog/analytics}}`. The "Headers" tab is selected, showing three entries: "Content-Type" with value "application/json", "Authorization" with value "Bearer {{{token}}}", and "Predix-Zone-Id" with value "{{{catalog_tenant}}}". There are red arrows pointing to both the URL template and the Headers section.

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

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

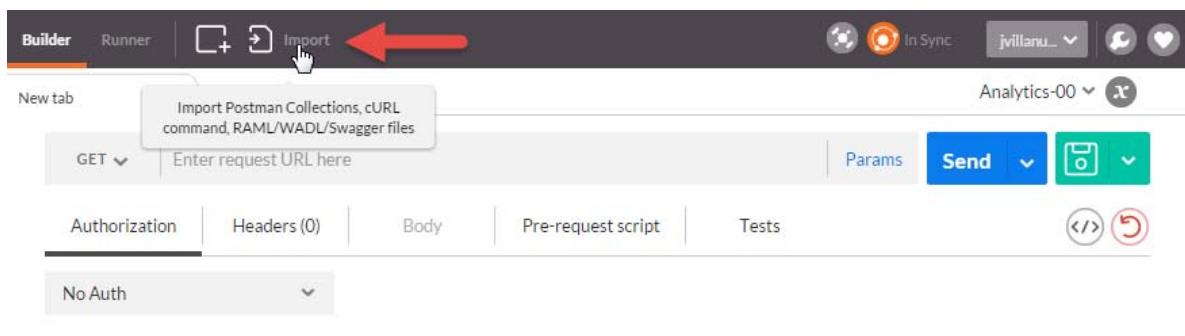
Steps

1. From your Chrome browser, add the Postman extension and launch Postman.

- From the Dev box, click on Applications>Internet> Chromium Browser
- Go to getpostman.com and click on **Get The App**.
- Launch Postman, from the main menu > applications > chromium apps > Postman

2. Import Postman collections to facilitate API requests.

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



- Browse for the Postman "dump" file (provided by instructor)
- Click **Import**.
- Click on the Collections tab of Postman which should now contain the collections as shown (partial list):

The screenshot shows the 'Collections' tab in Postman. It lists three collections:

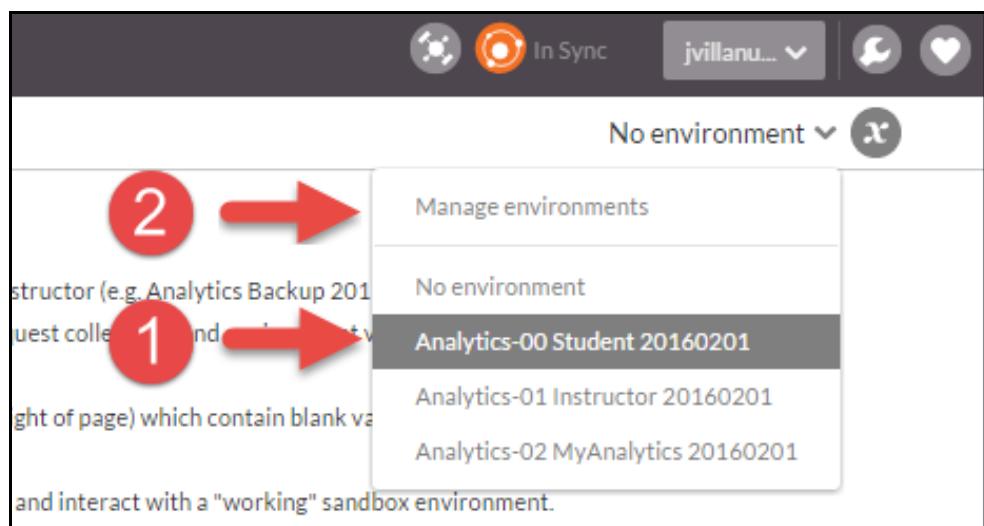
- _Read Me 1st** (1 Feb at 9:02 am • 3 requests)
- Analytic Catalog** (19 Jan at 3:07 pm • 21 requests)
- Get Token** (1 Feb at 9:01 am • 1 request)

Red arrows point from each collection name to a numbered list of instructions on the right side of the screen:

1. Import (button at top-middle of page) the Postman Backup 2015-01-20.postman_dump file into your collections and environment variables you will run.
2. If you're doing the lab exercises, select Analytics blank values for most of the environment variables.
3. Select Analytics-01 from the environments (top "working" sandbox environment).
4. If you'd like to check Postman for connectivity a

3. Set your Postman environment for Analytics.

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



4. Edit and update the variables.

- Open the Postman Analytics-00 environment variables and set the following variables.

Where:

- **protocol:** https
- **UAA Token URL:** UAA instance, "issuerId" value/URL (shown below, do not include the "https://")

```
    "label": "predix-analytics-runtime",
    "name": "predix-analytics-runtime-JaimeVillanueva",
    "plan": "Beta",
    "tags": []
}
],
"predix-uaa": [
{
  "credentials": {
    "issuerId": "https://58a17288-7004-4aff-bb37-88ed04c06c97.predix-uaa.run.aws-usw02-pr.ice.predix.io/oauth/token",
    "uri": "https://58a17288-7004-4aff-bb37-88ed04c06c97.predix-uaa.run.aws-usw02-pr.ice.predix.io",
    "zone": {
      "http-header-name": "X-Identity-Zone-Id",
      "http-header-value": "58a17288-7004-4aff-bb37-88ed04c06c97"
    }
  }
}
```

- **X-Identity-Zone-Id:** http-header-value

```
  "tags": []
},
"predix-uaa": [
{
  "credentials": {
    "issuerId": "https://58a17288-7004-4aff-bb37-88ed04c06c97.predix-uaa.run.aws-usw02-pr.ice.predix.io/oauth/token",
    "uri": "https://58a17288-7004-4aff-bb37-88ed04c06c97.predix-uaa.run.aws-usw02-pr.ice.predix.io",
    "zone": {
      "http-header-name": "X-Identity-Zone-Id",
      "http-header-value": "58a17288-7004-4aff-bb37-88ed04c06c97"
    }
  }
}]
```



Working with Analytics

The variables should be set as follows:

protocol	https
runtime_tenant	497a5a2f-bf8f-430c-9f45-bf4d3d4c6365
scheduler_uri	predix-scheduler-service-release.run.aws
token	eyJhbGciOiJSUzI1NiJ9eyJqdGkiOiJkZGJ
UAA Token URL	5d05f50d-387f-4646-a170-fa85af2693b
validationrequestid	Value
X-Identity-Zone-Id	5d05f50d-387f-4646-a170-fa85af2693b

5. Create your Authorization header.

- Click on the Authorization tab
- Select Basic Authorization for:
 - ◆ Username: admin
 - ◆ Password: myadminsecret
- Click on **Update request**

The authorization header will be generated and added as a custom header.

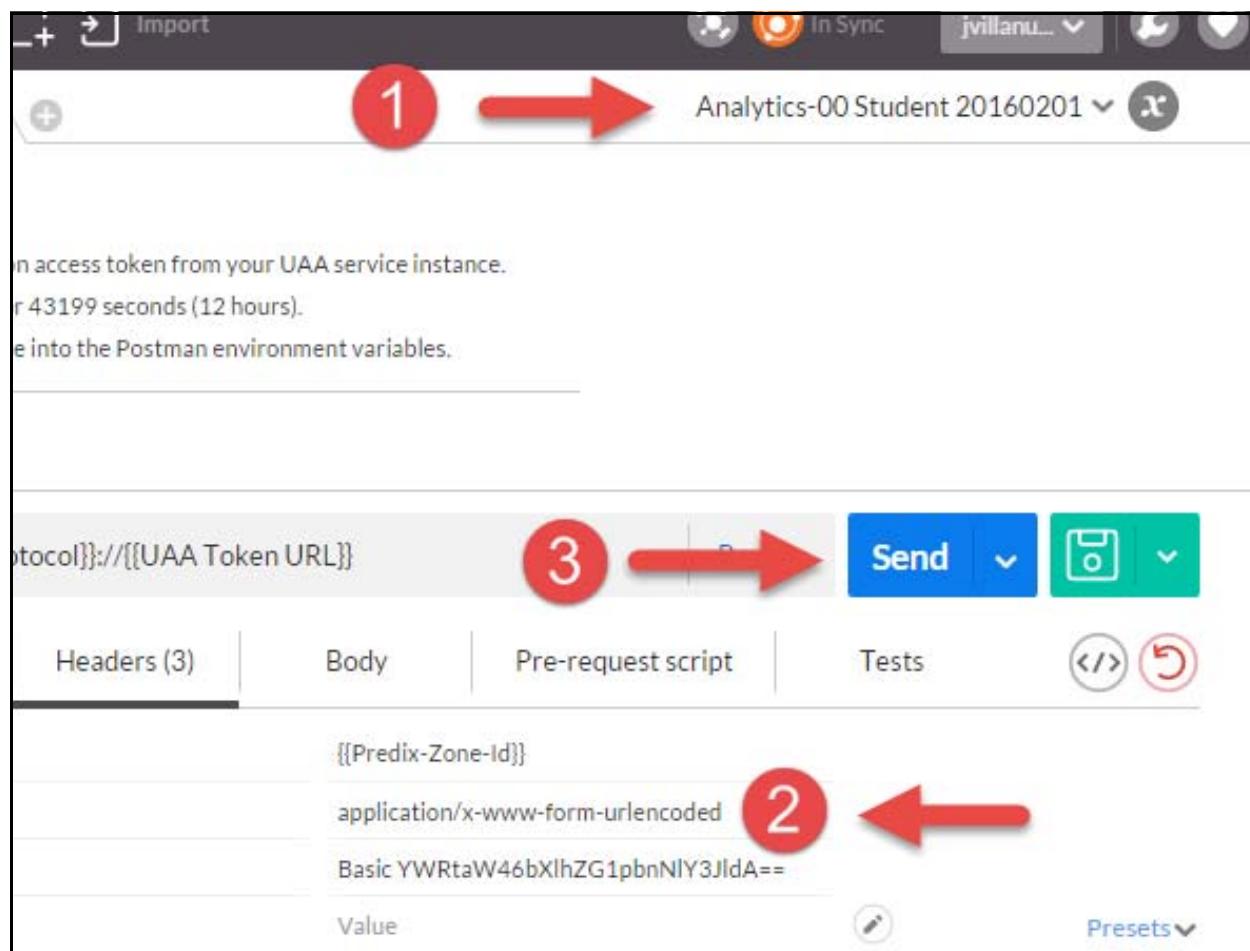
Save helper data to request

Clear Update request

This will add the Authorization header to your request.

6. Verify your request and click **Send**.

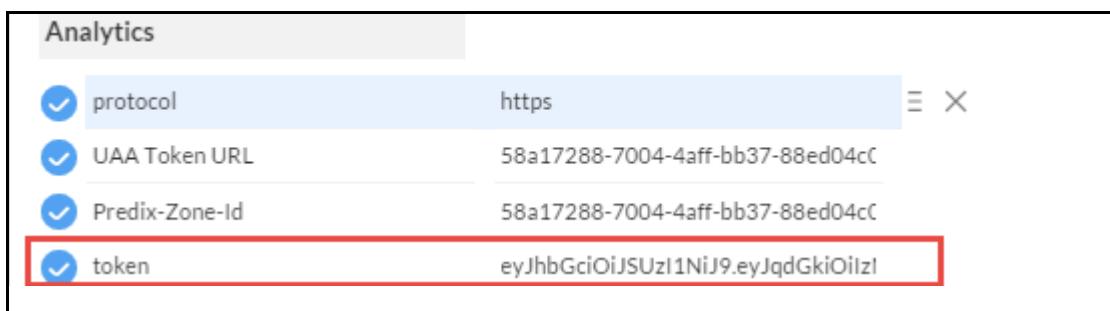
- Ensure your Postman is in the Analytics-00 environment
- Verify your three headers appear as follows.
- Click **Send**.



7. After clicking **Send**, you should receive the following response:

```
1 {  
2   "access_token": "eyJhbGciOiJSUzI1NiJ9.eyJqdGkiOiJKYjkxNDIyOC1kYzRlLThZWYt0Tcy  
Mi04YzAyYjA0MWRjMTkIiLCJzdWIoiJhZG1pbisInNjb38lIjpbiImNsawVudHMucmVhZCIsImFuYwx5  
dGLjcy56b25lcyc5kMjgzZTgzNC1lMDU3LTRkMDgtODIzYS1kN2ZkYjA3YjU0ZGQudXNlciIsImNsawVu  
dHMuc2VjcmV0IiwiawRwcy53cml0ZSIisInVhYS5yZXNvdXJjZSIisInpvbmVzLjU4YTE3Mjg4LTcwMDQt  
NGFmZi1iYjM3LTg4ZWQwNGMwNmM5Ny5hZG1pbisImFuYwx5dGljcy56b25lcyc5jYTA3MWQxZC0wNjY1  
LTQ3MWUt0DRkMS01MDhjMGm1NmQ1NDkudXNlciIsImNsawVudHMud3JpdGUILCJjbGllbnRzLmFkbWlu  
IiwiawRwcy5yZWFkIiwcic2Npb53cml0ZSIisInNjaW0ucmVhZCJdLCJjbGllbnRfaWQi0iJhZG1pbis  
ImnpZCI6ImFkbWluIiwiYXpwIjoiYRtaW4iLCJncmFudF90eX8lIjoiY2xpZw50X2NyZWRlbmRpYwxz  
IiwiemV2X3NpZyI6Ijk0ZjQ00Tc3IiwiawF0IjoxNDUxNDM0NDUxLCJleHAIoje0NTE0Nzc2NTEsImiz  
cyI6Imh0dHBz0i8vNThhMTcyODgtNzAwNC00YWZmLWJiMzct0DhlZDA0YzA2Yzk3LnByZWRpeC11YWEu  
cnVuLmF3cy11c3cwMi1wcisY2UucHJLZGL4Lmlvl29hdXRoL3Rva2VuIiwiemlkIjoiNThhMTcyODgt  
NzAwNC00YWZmLWJiMzct0DhlZDA0YzA2Yzk3IiwiYXVkiJpbImFkbWluIiwiY2xpZw50cyIsImFuYwx5  
dGLjcy56b25lcyc5kMjgzZTgzNC1lMDU3LTRkMDgtODIzYS1kN2ZkYjA3YjU0ZGQilCJpZHZBzIiwidWFh  
Iiwiem9uZXMuNThhMTcyODgtNzAwNC00YWZmLWJiMzct0DhlZDA0YzA2Yzk3IiwiYw5hbHl0aWNzLnpv  
bmVzLmNhMDcxZDFkLTA2NjUtNDcxZS04NGQxLTUwOGMwYzU2ZDU0OSIsInNjaW0iXX0.QjE3V840yJEf  
bqP9scUFCLN9XuHnpP4u030S6EDRVmTGWQrLsSLkWPuHDUcz9-lgiXP2jvZ7wL1UaK6Fw2Kce4wYIdFt  
vpAfnaQhpvJmQo01AMTlUAhLeNhQzkzGPyb6Z362PPYnT5yEnwG4e_EiQLIBQF8hkWztTucIwGUTD0p  
-0fqNRrAaZh9NESal1IglrdvqPcGCyn0o4wVlo6Ip7aG6HPNEsQ5s5GFNvXlz5DSWIW8RS9djNH7Ig5g  
OCwzaU4F-QmX3dbqHBpsDrdjt7p6RiIP0ywvr3pY9AiFNLER6i55eYY0mZToOIMg4-4xSimV54_i1fxm  
nmch8fFSLYg",  
3   "token_type": "bearer",  
4 }
```

- The highlighted text represents the UAA token value. Copy this text and enter it into your Postman variables as "token". Click **Submit** to save the variable.



1. Click **Save Response** (to save the response to your Get Token request).

Headers (15) Tests (0/0) Status 200 OK Time 2490 ms

view JSON  Save response



```
ss_token": "eyJhbGciOiJSUzI1NiJ9.eyJqdGkiOiI3MTdhN2U2MC1hOGQ5LTQ1ZDMtOGMwMMTMyNjhLYzcILCJzdWIiOiJhZG1pbisInNjb3BlIjpbiIsImNsawVudHMucmVhZCIisImNsawVudHmV0IiwiaWRwcy53cmI0ZSIisInVhYS5yZXNvdXJjZSIisInpvbmVzljVkMDVmNT8kLTM4N2YtNDYcwlWZhODVhZjI20TNiMi5hZG1pbisImNsawVudHMud3JpdGUiLCJjbGllbnRzLmFkbWluIiwiyZWfkIiwic2NpbS53cmI0ZSIisInNjaw0ucmVhZCJdLCJjbGllbnRfaWQiOijhZG1pbisImNpZbwluIiwiYXpwIjoiYWRtaW4iLCJncmFudF90eXBLIjoiY2xpZW50X2NyZWRLbnRpYWxzIiwicmbVI6IjJhNGU30WVjIiwiawFOIjoxNDU0MzUvMD02LCJleHAiOjE0NTQzOTUvNDYsImIzcvI6Imh
```

- To recall your Saved response(s), simply click on any of your responses, as shown below.

Get Token  Analytics-00 Studer



Get Token

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.

200 OK 

POST  {{protocol}}://{{UAA Token URL}} Params 

2. Enter all the required Postman environment variables.

To facilitate the other requests, set up the following variables as shown.

The screenshot shows the 'Analytics-00 Student 20160201' environment configuration screen. It lists various environment variables with their values. Blue arrows point from specific variables to their corresponding entries in the JSON output on the right.

	Value
analyticId	Value
analyticName	Value
analyticVersion	Value
artifactId	Value
bpmnXML_content	Value
catalog_tenant	f59bb9e0-da68-45a7-a5c9-bd4a6c7409
catalog_uri	predix-analytics-catalog-release.run.aws-t
config_uri	predix-analytics-config-release.run.aws-t
configurationId	Value
Content-Type	application/json
deploymentrequestId	Value
execution_uri	predix-analytics-execution-release.run.aws-t
jobId	Value
protocol	https
runtime_tenant	497a5a2f-bf8f-430c-9f45-bf4d3d4c636t
scheduler_uri	predix-scheduler-service-release.run.aws-t
token	Value
UAA Token URL	5d05f50d-387f-4646-a170-fa85af2693t
validationrequestId	Value
X-Identity-Zone-Id	5d05f50d-387f-4646-a170-fa85af2693t

System-Provided:

```
{
  "VCAP_SERVICES": {
    "predix-analytics-catalog": [
      {
        "credentials": {
          "catalog_uri": "https://predix-analytics-catalog",
          "zone-http-header-name": "Predix-Identity-Zone",
          "zone-http-header-value": "f59bb9e0-da68-45a7-a5c9-bd4a6c7409",
          "zone-oauth-scope": "analytics.zone"
        },
        "label": "predix-analytics-catalog",
        "name": "predix-analytics-catalog",
        "plan": "Beta",
        "tags": []
      }
    ],
    "predix-analytics-runtime": [
      {
        "credentials": {
          "config_uri": "https://predix-analytics-runtime",
          "execution_uri": "https://predix-analytics-runtime",
          "scheduler_uri": "https://predix-analytics-runtime",
          "zone-http-header-name": "Predix-Identity-Zone",
          "zone-http-header-value": "497a5a2f-bf8f-430c-9f45-bf4d3d4c636t",
          "zone-oauth-scope": "analytics.zone"
        },
        "label": "predix-analytics-runtime",
        "name": "predix-analytics-runtime",
        "plan": "Beta",
        "tags": []
      }
    ],
    "predix-uaa": [
      {
        "credentials": {
          "issuerId": "https://5d05f50d-387f-4646-a170-fa85af2693t",
          "uri": "https://5d05f50d-387f-4646-a170-fa85af2693t",
          "zone": {
            "http-header-name": "X-Identity-Zone-Id",
            "http-header-value": "5d05f50d-387f-4646-a170-fa85af2693t"
          }
        },
        "label": "predix-uaa",
        "name": "predix-uaa-jv",
        "plan": "Beta",
        "tags": []
      }
    ]
  }
}
```

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

Exercise 2: Updating the OAuth2 Client

Before using the Analytics Catalog and Analytics Runtime services, you must update the OAuth2 Client to use these services.

Overview

To enable applications to access the Analytics Catalog and Runtime services, your JSON Web Token (JWT) must contain the following scope(s) for both the catalog and runtime services:

```
analytics.zones.<service_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.

To enable your application to access a platform service, your JSON Web Token (JWT) must contain the scopes required for a platform service.

Look at line 5 of the Get Token Response (from the previous lab exercise) which lists the scope of access and authorization; it does not include the analytics.zones statements needed for accessing the Catalog and Runtime services.

```
eyJhbGciOiJzQWVzIiinQDQ12g5z100njqZLwCAx2ALzmc...WWmimj...z1iyLnb0yZwqpecl1lwLaciNvadlm...  
cy11c3cwMi1wc15pY2UucHJlZGl4LmlvL29hdXR0L3Rva2VuIiwiemlkIjoiNWQwNWY1MGQtMzg3Zi00N  
jQ2LWExNzAtZmE4NWFmMjY5M2IyIiwiYXVkIjpBImFkbWluIiwiY2xpZW50cyIsImlkcHMiLCJ1WEiLC  
J6b25Lcy41ZDA1ZjUwZC0zODdmLTQ2NDYtYTE3MC1mYTg1YWYyNjkzYjIiLCJzY2ltIl19_rDorgcZnjr  
6KT7U7oE-4N_2sqHmvvIIkkLAF2d2Yd8kKzqQkcKHCKwMNpe2gmcVPmX50BHFISl_bDoeFdjeZQ9IN_Mb  
77RfgvhGT6ab1We6y_kyXT-rvGgGa8MswUcfpzdFq5fBF8n0Uwg3a2xoKegzPRcygPtmsAb3lRUt_8j2Q  
ZYvYDZP0J7pQSiz60s-RSUj8YyVu0QKGa1WYDD1TmM17eeZFmY0ZDjLNfxLY7_YvCjQA_2Uz71TTctKS  
-l04u8-gbjXif65qDHnyFq_hEMx4PYHv-ediwXIvc4Jg0YvPeMevPEQIAD5Fn1bSinstLEs6rBmk6Be  
-NYbHYuq5CQ",  
3   "token_type": "bearer",  
4   "exp": 162199,  
5   "scope": "clients.read clients.secret idps.write uaa.resource zones.5d05f50d  
-387f-4646-a170-fa85af2693b2.admin clients.write clients.admin idps.read scim  
.write scim.read",  
6   "jti": "717a7e60-a8d9-45d3-8c02-cd1413368ec7"  
7 }
```



This lab exercise will show you how to edit the scope of the access (**if required**) to include Catalog and Runtime services.

In this exercise you will:

- view and copy the scope for the token
- view and copy the scope for the Analytics Catalog instance
- view and copy the scope for the Analytics Runtime instance
- Combine all three scope statements

Steps

1. Run "Get Token" from Postman, and copy the current scope for the token.

- View/copy the "scope" (line 5) statements of the access token response into gedit.
For example:

```
xTP6C38To_W-qD-Dm72NKPkgZzcC4ht9qyGZpBQBOH5-Olu7xgeb-RupHscKBSa8CX-
mPIJvYDgR93aab85ruRmnFtJAxQvb5uE6DyBOjL03SE5YGzpiGcI2MZLnyEnd9sh1_ZcQlslCrRcWwP
p2UQwIDTd3AA5TZGTmUrHZoiz4lKY14eK_oEIuX7wB7DMhuqCZVuUWMTr5n4DO5bADSe-
Xc-BE1QeE9BxR90q0rUQ32F2v5XmF6NQI6RNzN-
yQXSxjJBpwFBXe0MVWqq9L3GQYSPhElaer7rFYWZ4jsJoZBGMfRB_cj07lw4Q",
3.   "token_type": "bearer",
4.   "expires_in": 43199,
5.   "scope": "clients.read clients.secret idps.write uaa.resource
zones.58a17288-7004-4aff-bb37-88ed04c06c97.admin clients.write clients.admin id
scim.write scim.read",
6.   "jti": "13a2bbb8-8fa8-425d-bed7-758bfe468b36"
7. }
```

2. Copy and append the scope for the Analytics Catalog instance to the token scope in gedit.

```
  Edit  View  Search  Terminal  Tabs  Help
dix@localhost:~          predix@localhost:~
CAP_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": "d283e834-e057-4d08-823a-d7fdb07b54dd",
      "zone-oauth-scope": "analytics.zones.d283e834-e057-4d08-823a-d7fdb07b54dd.user"
    },
    "label": "predix-analytics-catalog",
    "name": "predix-analytics-catalog-JaimeVillanueva",
    "plan": "Beta",
    "tags": []
  }
],
```

3. Copy and append the scope for the Analytics Runtime instance to the token scope in gedit.

```
],
  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",
      "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": "ca071d1d-0665-471e-84d1-508c0c56d549",
      "zone-oauth-scope": "analytics.zones.ca071d1d-0665-471e-84d1-508c0c56d549.user"
    },
    "label": "predix-analytics-runtime",
    "name": "predix-analytics-runtime-JaimeVillanueva",
    "plan": "Beta",
    "tags": []
  }
],
```



Your completed scope statement (in gedit) should like the following example:

```
"clients.read clients.secret idps.write uaa.resource  
zones.58a17288-7004-4aff-bb37-88ed04c06c97.admin clients.write  
clients.admin idps.read scim.write scim.read  
analytics.zones.d283e834-e057-4d08-823a-d7fdb07b54dd.user  
analytics.zones.ca071d1d-0665-471e-84d1-508c0c56d549.user"
```

Next, you will use the UAA command line interface to update the UAA token scope statement.

4. Target your UAA service instance.

You can use the UAA command-line interface (UAAC), to manage your UAA instance. Refer to Predix.io documentation for additional information or help on this topic.

- From your terminal window, open a new tab by clicking on file>open tab and enter the following command:

```
uaac target <uaa_instance_url>
```

Where:

<uaa_instance_url> is the URL to your trusted issuer (refer to your gedit text file containing your Hello World environment variables).

For our example:

```
uaac target  
https://58a17288-7004-4aff-bb37-88ed04c06c97.predix-uaa.run.aws-us  
w02-pr.ice.predix.io
```

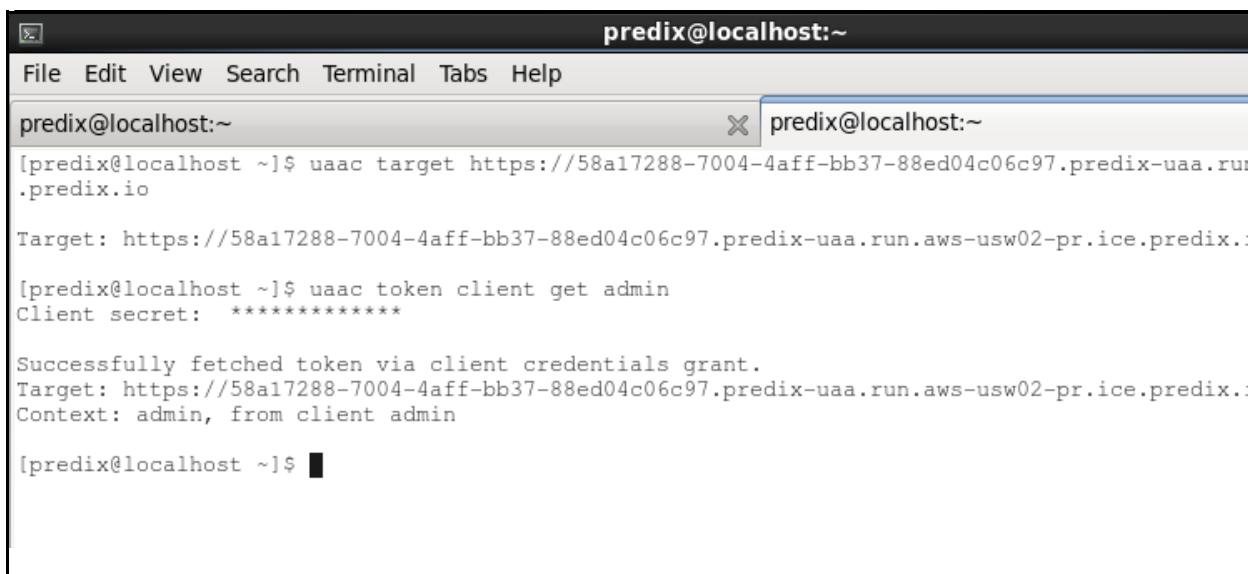
5. Log in to your UAA service instance.

Log in and verify the default admin account and password set up when you created your UAA service instance.

- From your terminal window and tab targeting your UAAC instance, run the following command:

```
uaac token client get admin
```

Specify the administrative client secret when prompted (**e.g. myadminsecret**).



The screenshot shows a terminal window with two tabs. The left tab is titled 'predix@localhost:~' and contains the command '[predix@localhost ~]\$ uaac target https://58a17288-7004-4aff-bb37-88ed04c06c97.predix-uaa.run.predix.io'. The right tab is also titled 'predix@localhost:~' and shows the output of the command. It includes the target URL, a prompt for the client secret (with the input masked by asterisks), and a confirmation message: 'Successfully fetched token via client credentials grant.' The terminal window has a standard OS X-style interface with a menu bar and tabs.

This step validates:

- your UAA instance
- your admin client
- your admin password



6. Update the OAuth2 client with the authorities (scopes) required.

- Run the following command:

```
uaac client get admin
```

The image below shows the authorities before updating.



```
predix@localhost:~ predix@localhost ~]$ uaac client get admin
scope: uaa.none
client_id: admin
resource_ids: none
authorized_grant_types: client_credentials
autoapprove:
action: none
authorities: clients.read clients.secret idps.write uaa.resource
    zones.58a17288-7004-4aff-bb37-88ed04c06c97.admin clients.write clients.admin idps.read scim.write
    scim.read
lastmodified: 1450371407743
[predix@localhost ~]$
```

- Run the following command:

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

where <set_of_authorities> is your existing set of authorities plus authorities required for a platform service(s). This complete set should be in your gedit window. Be sure to include opening and closing quotation marks with the statement.

- As in the following example:

```
uaac client update admin --authorities "clients.read clients.secret
idps.write uaa.resource
zones.58a17288-7004-4aff-bb37-88ed04c06c97.admin clients.write
clients.admin idps.read scim.write scim.read
analytics.zones.d283e834-e057-4d08-823a-d7fdb07b54dd.user
analytics.zones.ca071d1d-0665-471e-84d1-508c0c56d549.user"
```

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

```
uaac client get admin
```

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

The image below shows the "before and after" authorities.

```
[predix@localhost ~]$ uaac client update admin --authorities "clients.read clients.secret idps.write uaa.resource zones.58a17288-7004-4aff-bb37-88ed04c06c97.admin clients.write clients.admin idps.read scim.write scim.read analytics.zones.d283e834-e057-4d08-823a-d7fdb07b54dd.user analytics.zones.d283e834-e057-4d08-823a-d7fdb07b54dd.user"
scope: uaa.none
client_id: admin
resource_ids: none
authorized_grant_types: client_credentials
autoapprove:
action: none
authorities: clients.read analytics.zones.d283e834-e057-4d08-823a-d7fdb07b54dd.user clients.secret idps.write uaa.resource zones.58a17288-7004-4aff-bb37-88ed04c06c97.admin clients.write clients.admin idps.read scim.write scim.read
lastmodified: 1450476963333
[predix@localhost ~]$
```

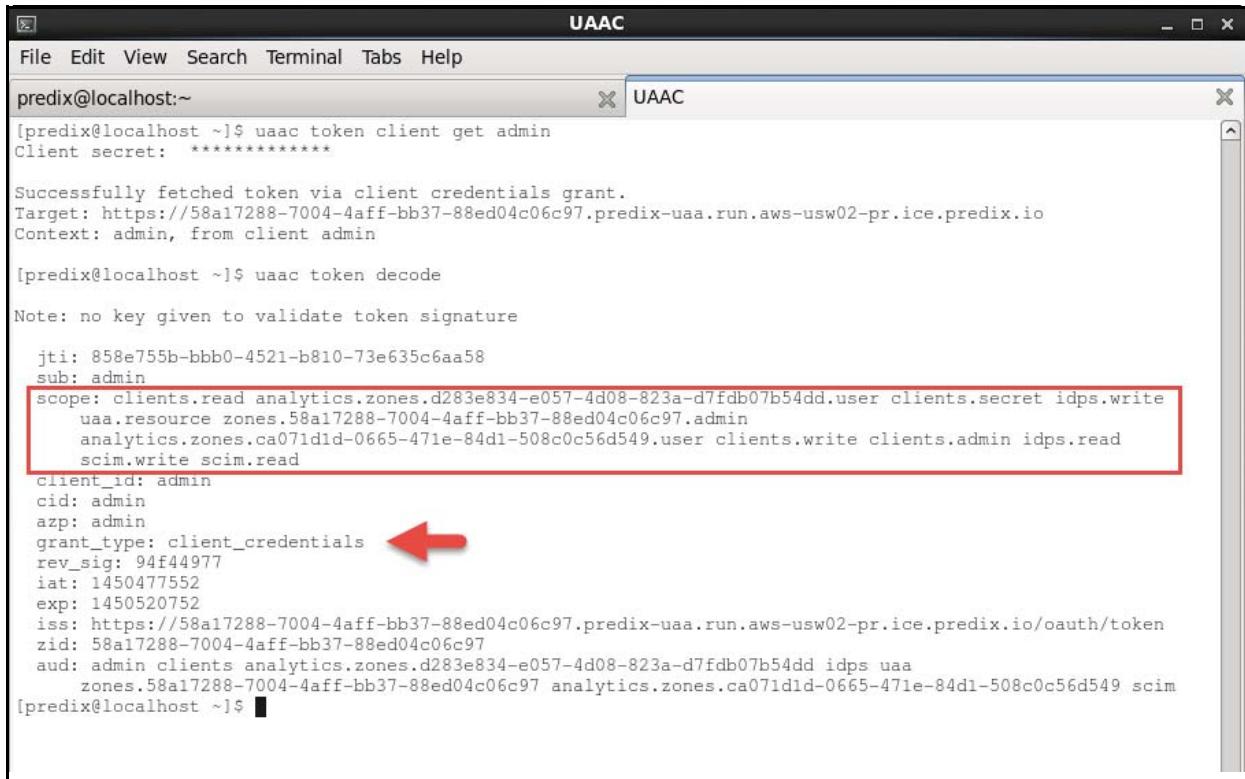
7. Get the token again with client credential grant.

```
uaac token client get admin
```

This command retrieves a token from UAA (occurs in the background, and is not shown).

8. Validate with UAAC that the scopes were updated in the token.

uaac token decode



```
[predix@localhost ~]$ uaac token client get admin
Client secret: *****
Successfully fetched token via client credentials grant.
Target: https://58a17288-7004-4aff-bb37-88ed04c06c97.predix-uaa.run.aws-usw02-pr.ice.predix.io
Context: admin, from client admin

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

jti: 858e755b-bbb0-4521-b810-73e635c6aa58
sub: admin
scope: clients.read analytics.zones.d283e834-e057-4d08-823a-d7fdb07b54dd.user clients.secret idps.write
      uaa.resource zones.58a17288-7004-4aff-bb37-88ed04c06c97.admin
      analytics.zones.ca071d1d-0665-471e-84d1-508c0c56d549.user clients.write clients.admin idps.read
      scim.write scim.read
client_id: admin
cid: admin
azp: admin
grant_type: client_credentials ←
rev_sig: 94f44977
iat: 1450477552
exp: 1450520752
iss: https://58a17288-7004-4aff-bb37-88ed04c06c97.predix-uaa.run.aws-usw02-pr.ice.predix.io/oauth/token
zid: 58a17288-7004-4aff-bb37-88ed04c06c97
aud: admin clients analytics.zones.d283e834-e057-4d08-823a-d7fdb07b54dd idps uaa
      zones.58a17288-7004-4aff-bb37-88ed04c06c97 analytics.zones.ca071d1d-0665-471e-84d1-508c0c56d549 scim
[predix@localhost ~]$
```

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

9. Validate scope with Postman: Get token and view scope.

- Using Postman, run the Get Token request and view the scope once again to validate your access token has the appropriate JWT values for accessing the Analytics Catalog and Runtime service instances.

```
iiwiiaIwcy5yZwIkiwiC2np855cmto28151mjaWbaclmI283ab59j0tL0m1awQ1o1512  
ImNpZCI6ImFkbWluIiwiYXpwIjoiYWRTaW4iLCJncmFudF90eXB1IjoiY2xpZW50X2NyZWRlb  
IiwigcmV2X3NpZyI6Ijk0ZjQ00Tc3IiwiawF0IjoxNDM0NDUxLCJleHAiOjE0NTIE0Nzc2N  
cyI6Imh0dHBzO18vNThhMTcyODgtNzAwNC00YWZmLWJiMzct0DhlZDA0YzA2Yzk3LnByZWRpe  
cnVuLmF3cy11c3cwMi1wci5pY2UucHJLZGL4LmlvL29hdXR0L3Rva2VuIiwiemlkIjoiNThhM  
NzAwNC00YWZmLWJiMzct0DhlZDA0YzA2Yzk3IiwiYXVkJpbImFkbWluIiwiY2xpZW50cyIsI  
dGljcy56b25lcy5kMjgzTgzc1lMDU3LIRkMDgt0DIzYS1kN2ZkYjA3YjU0ZGQiLCJpZHbzI  
Iiwiem9uZXMuNThhMTcyODgtNzAwNC00YWZmLWJiMzct0DhlZDA0YzA2Yzk3IiwiYw5hbHL0a  
bmVzLmNhMDcxZDFkLTA2NjUTNDcxZS04NGQxLTUwOGMwYzu2ZDU00SiIsInNjaW0iXX0.QjE3V  
bqP9scUFCLN9XuHnpP4u030S6EDRVmIGWQrLsSLKWPuHDUcz9-lgiXP2jvZ7wL1UaK6Fw2Kce  
vpAfnAQHpvJmQo01AMtUAhLeNhQzkzGPyb6Z362PPYnT5yENwG4e_EiQLIBQF8hkWztTucIw  
-0fqNRrAaZn9NESal1IglrdvqPcGCyn0o4wVlo6Ip7aG6HPNEsQ5s5GFNvXlz5DSWTW8RS9dj  
OCwzaU4F-QmX3dbqHbpsDrdjt7p6RiIP0ywvr3pY9AiFNLER6i55eYY0mZTo0IMg4-4xSimV5  
nmch8fFSLYg",  
3     "token_type": "bearer",  
4     "expires_in": 43199,  
5     "scope": "clients.read analytics.zones.d283e834-e057-4d08-823a-d7fdb07b  
.user clients.secret idps.write uaa.resource zones.58a17288-7004-4aff-bb3  
-88ed04c06c97.admin analytics.zones.ca071d1d-0665-471e-84d1-508c0c56d549.  
clients.write clients.admin idps.read scim.write scim.read",  
6     "jti": "ub914228-uc4e-4ae1-9722-8c020041uc19"  
7 }
```

After updating the scope, you can now see that the access token has the required scope of access.

- Copy and paste this new token value into your Postman environment.**



Exercise 3: Working with the Analytics Catalog

Overview

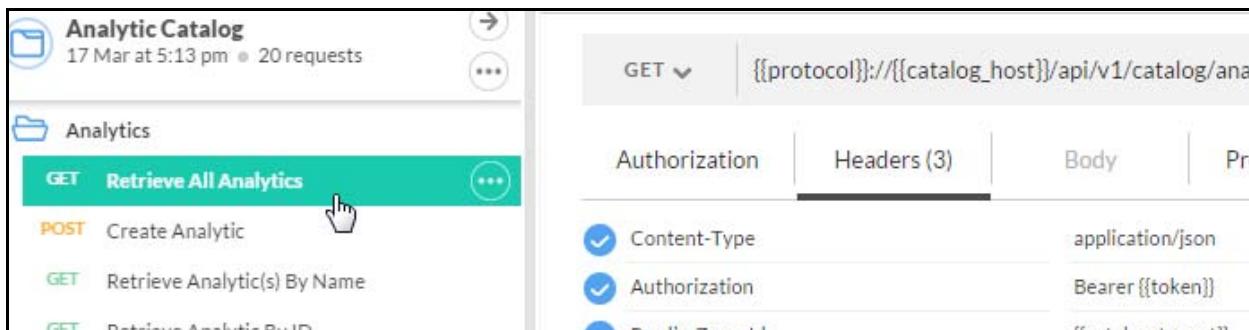
For this lab exercise, you will work with a simple Analytic application demo (provided as part of the lab files), upload it to the catalog, then deploy and test it. The application will receive two numbers and simply add them and provide the result.

The following steps describe the general process for adding a new analytic to the catalog. The high level steps are:

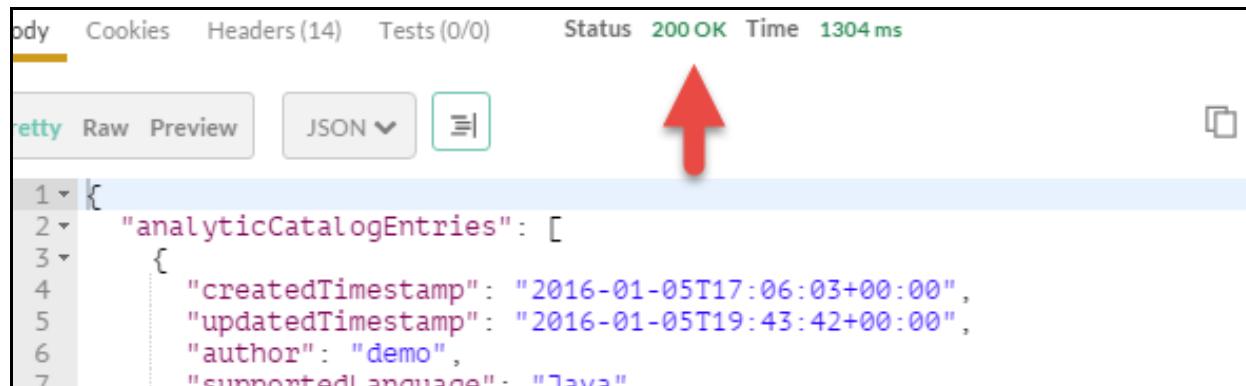
- Create the analytic.
- Upload the analytic to the catalog.
- Deploy and test the analytic in the Cloud Foundry environment

1. Get all analytic catalog entries.

- If necessary (if it's been over 12 hours since your last token retrieval), retrieve a new access token and paste it into your Token variable for your environment.
- Add a new Postman tab (window) and select **GET Retrieve All Analytics** from your Postman collections.



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



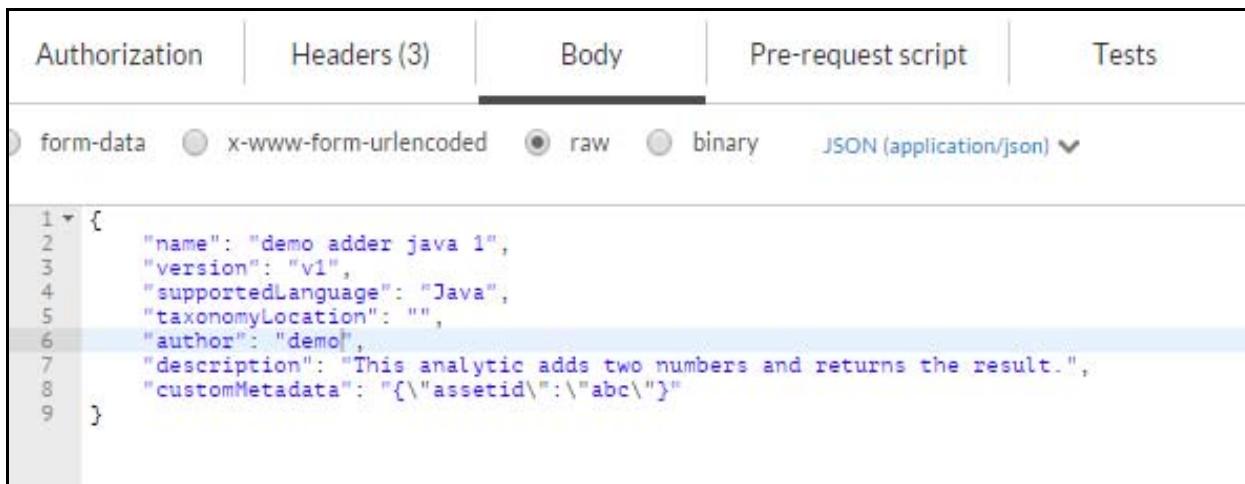
The screenshot shows the Postman interface after sending a GET request. The status bar at the top indicates "Status 200 OK Time 1304 ms". Below the status bar, there are tabs for "Body", "Cookies", "Headers (14)", and "Tests (0/0)". The "Body" tab is selected and displays a JSON response. A red arrow points upwards from the "Body" tab towards the status bar. The JSON response content is as follows:

```
1 {
2   "analyticCatalogEntries": [
3     {
4       "createdTimestamp": "2016-01-05T17:06:03+00:00",
5       "updatedTimestamp": "2016-01-05T19:43:42+00:00",
6       "author": "demo",
7       "supportedLanguages": "עברית"
```

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

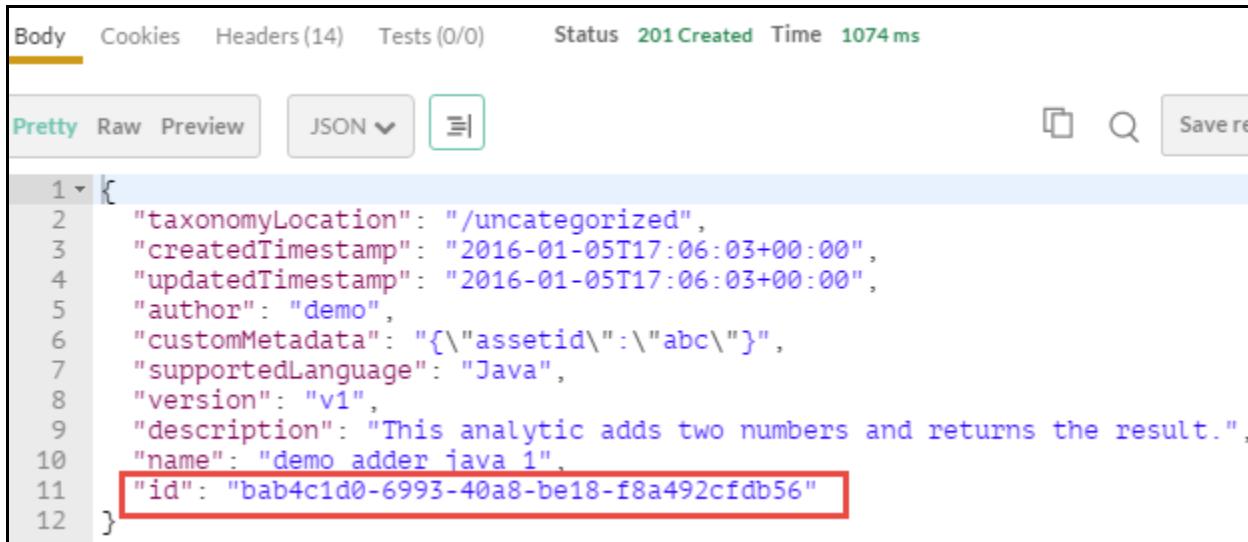
2. Create a catalog entry for your sample Analytic application.

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



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

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



The screenshot shows a Postman response window. At the top, it displays "Body" (highlighted in yellow), "Cookies", "Headers (14)", "Tests (0/0)", "Status 201 Created", and "Time 1074 ms". Below this, there are tabs for "Pretty", "Raw", and "Preview" (disabled). A "JSON" dropdown is set to "Pretty". On the right side, there are icons for copy, search, and save. The main area contains a JSON response with line numbers 1 through 12. Line 11, which contains the "id" field, is highlighted with a red box.

```
1  {
2      "taxonomyLocation": "/uncategorized",
3      "createdTimestamp": "2016-01-05T17:06:03+00:00",
4      "updatedTimestamp": "2016-01-05T17:06:03+00:00",
5      "author": "demo",
6      "customMetadata": "{\"assetid\":\"abc\"}",
7      "supportedLanguage": "Java",
8      "version": "v1",
9      "description": "This analytic adds two numbers and returns the result.",
10     "name": "demo adder java 1",
11     "id": "bab4c1d0-6993-40a8-be18-f8a492cfdb56"
12 }
```

- Note the **id** value created for the catalog entry.
- 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.



Working with Analytics

The screenshot shows the Postman interface with the 'Builder' tab selected. On the left, there's a sidebar with 'History' and 'Collections'. Under 'Collections', there's an 'Analytic Catalog' entry with a timestamp '17 Mar at 5:13 pm' and '20 requests'. Below it are 'Analytics' and 'Artifacts' sections. A 'POST Create Artifact' button is highlighted with a cursor. To its right is a detailed view of the 'Create Artifact' request:

- Method:** POST
- URL:** `https://[[catalog_host]]/api/v1/catalog/artifacts`
- Headers (4):**
 - Content-Type: multipart/form-data
 - Authorization: Bearer {{token}}
 - Accept: application/json
 - Predix-Zone-Id: {{catalog_tenant}}
- Body:** This section is currently empty.
- Pre-request script:** Not present in this screenshot.
- Tests:** Not present in this screenshot.

- In Postman update the variables: analyticId (with the Catalog Entry ID), analyticName, and analyticVersion.

In the Body of the request, set or verify parameters as follows:

- type: Executable
- description: "This analytic adds 2 numbers and provides the sum."

This screenshot shows the 'Body' tab in Postman. It has five tabs: Authorization, Headers (4), Body, Pre-request script, and Tests. The 'Body' tab is active. It includes options for 'form-data', 'x-www-form-urlencoded', 'raw', and 'binary'. The 'form-data' option is selected. There are four parameters listed:

Parameter	Type	Value
file	File	Choose Files No file chosen
catalogEntryId	Text	Value: {{analyticId}}
type	Text	Value: Executable
description	Text	Value: This analytic adds 2 numbers and provides the su

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.

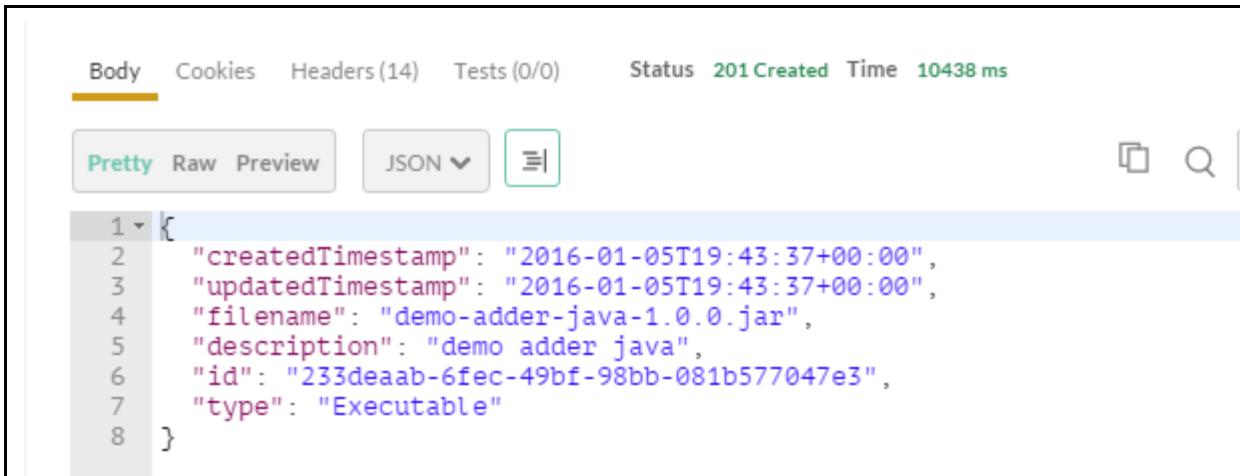
Your instructor will provide instructions on the location and filename (e.g. demo-adder-java-1.0.0.jar) of the sample java application to upload for this catalog entry.

The screenshot shows the Postman interface for making a POST request to the API endpoint {{protocol}}://{{catalog_uri}}/api/v1/catalog/artifacts. The 'Body' tab is selected, showing an 'x-www-form-urlencoded' payload. A 'Choose Files' button is highlighted, with the file 'demo-adder-java-1.0.0.jar' selected. Other fields in the body include 'catalogEntryId' ({{analyticId}}), 'type' (Executable), 'description' (This analytic adds 2 numbers and provides the sum), and 'value'. The 'Send' button is visible at the top right.

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

You should receive the following response:

Working with Analytics



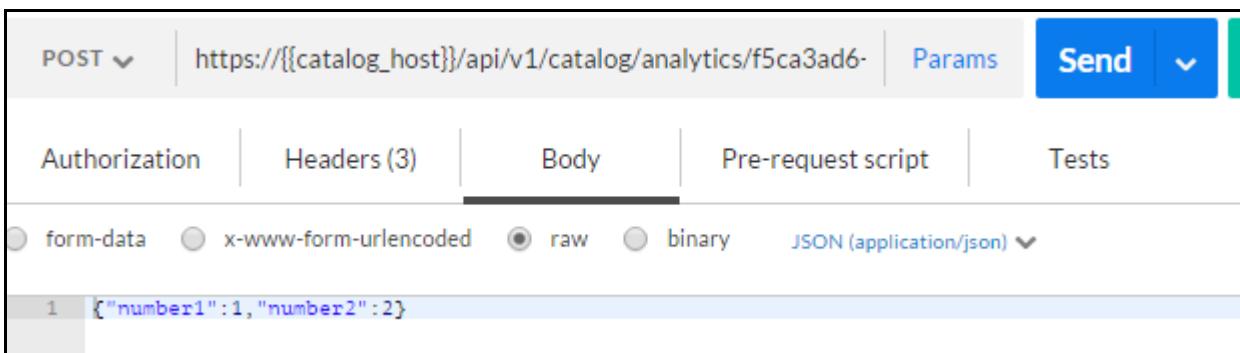
The screenshot shows the Postman interface after a successful API call. The status bar at the top indicates a **201 Created** status with a response time of **10438 ms**. The **Body** tab is selected, displaying a JSON response with the following content:

```
1  {
2      "createdTimestamp": "2016-01-05T19:43:37+00:00",
3      "updatedTimestamp": "2016-01-05T19:43:37+00:00",
4      "filename": "demo-adder-java-1.0.0.jar",
5      "description": "demo adder java",
6      "id": "233deaab-6fec-49bf-98bb-081b577047e3",
7      "type": "Executable"
8 }
```

- Copy the "id" of the artifact and paste it into your Postman environment variables.
- 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.

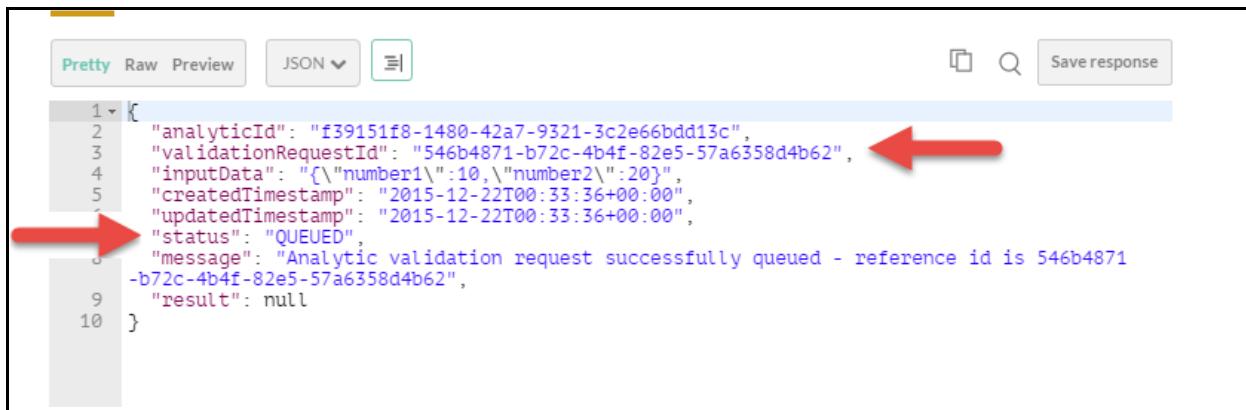
- Add a new Postman tab (window) and select **POST Validate Analytic** from the Validation/Deployment/Execution folder of your Postman collections.
- For <value1>, <value2>: Insert any two numbers you would like to add.



The screenshot shows a new Postman tab with a **POST** request. The URL is set to https://{{catalog_host}}/api/v1/catalog/analytics/f5ca3ad6-. The **Body** tab is selected, showing the raw JSON payload:

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

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



The screenshot shows a Postman response window. The response is a JSON object with the following structure:

```
1 {  
2   "analyticId": "f39151f8-1480-42a7-9321-3c2e66bdd13c",  
3   "validationRequestId": "546b4871-b72c-4b4f-82e5-57a6358d4b62",  
4   "inputData": "{\"number1\":10,\"number2\":20}",  
5   "createdTimestamp": "2015-12-22T00:33:36+00:00",  
6   "updatedTimestamp": "2015-12-22T00:33:36+00:00",  
7   "status": "QUEUED",  
8   "message": "Analytic validation request successfully queued - reference id is 546b4871  
-b72c-4b4f-82e5-57a6358d4b62",  
9   "result": null  
10 }
```

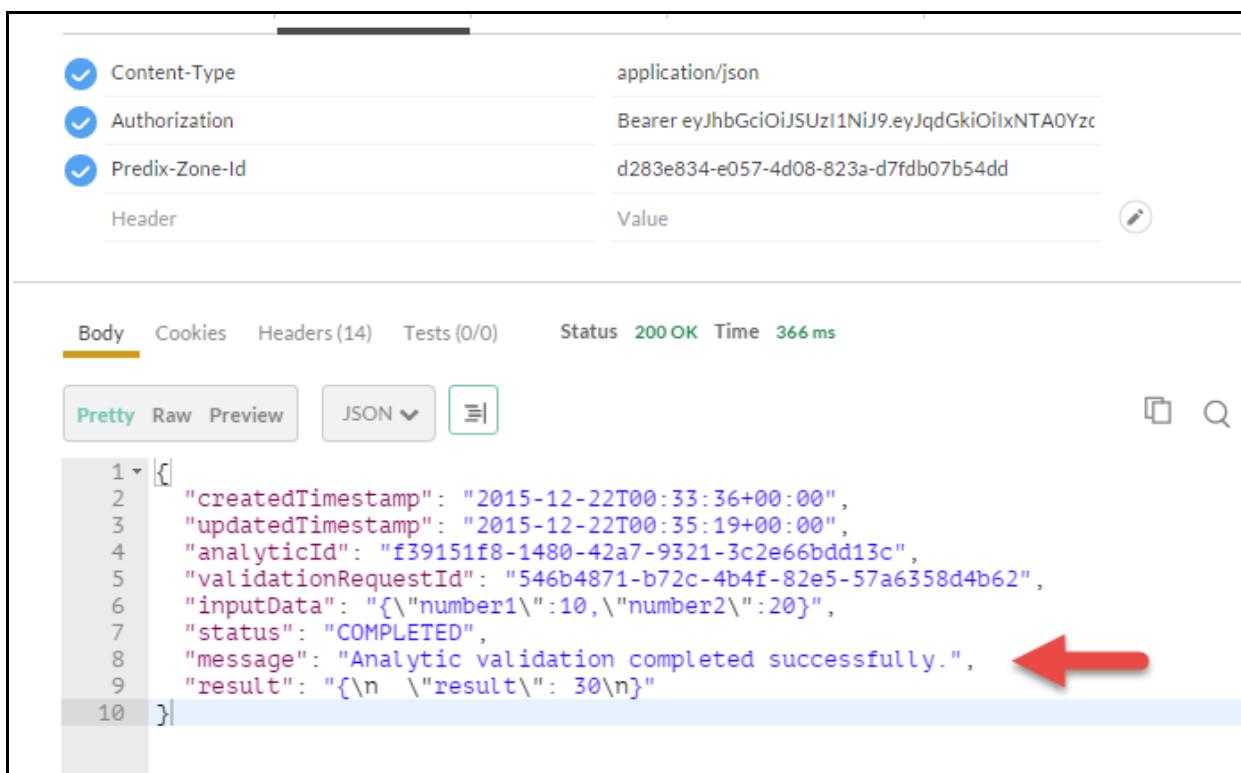
Two red arrows point to the validation request ID: one from the left pointing to the value of the "validationRequestId" key, and another from the right pointing to the same value.

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



7. Poll for the validation status.

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



The screenshot shows a Postman collection interface. In the top header, there are three checked checkboxes for 'Content-Type' (application/json), 'Authorization' (Bearer eyJhbGciOiJSUzI1NiJ9eyJqdGkiOiIxNTA0Yzc), and 'Predix-Zone-Id' (d283e834-e057-4d08-823a-d7fdb07b54dd). Below the header, there are tabs for 'Body', 'Cookies', 'Headers (14)', 'Tests (0/0)', 'Status' (200 OK), and 'Time' (366 ms). The 'Body' tab is selected and displays a JSON response. The JSON response is as follows:

```
1  {
2   "createdTimestamp": "2015-12-22T00:33:36+00:00",
3   "updatedTimestamp": "2015-12-22T00:35:19+00:00",
4   "analyticId": "f39151f8-1480-42a7-9321-3c2e66bdd13c",
5   "validationRequestId": "546b4871-b72c-4b4f-82e5-57a6358d4b62",
6   "inputData": "{\"number1\":10,\"number2\":20}",
7   "status": "COMPLETED",
8   "message": "Analytic validation completed successfully.",
9   "result": "\n \"result\": 30\n"
10 }
```

A red arrow points to the 'result' field in the JSON response, which contains the value '30'. The 'Pretty' button is highlighted in blue.

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.



Lab 3: *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 58
- *Schedule Orchestration and Analytic Execution (Scheduling a Job)*, page 67



Exercise 1: Running an Orchestration with One Analytic

Overview

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

- analytic catalog ID
- analytic name
- analytic version

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



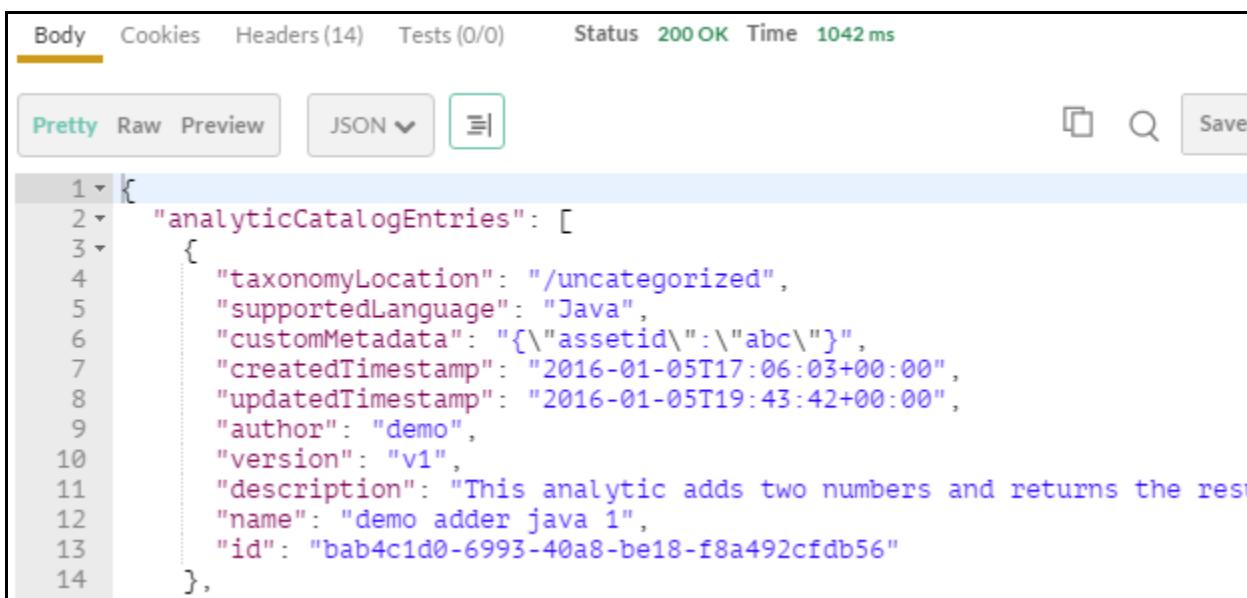
To facilitate learning, your Postman Collections under Orchestration Execution, there are two requests:

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

Steps

1. Get the ID, name, and version of the analytic that you'd like to run an orchestration on.

- Add a new Postman tab (window) and select **GET Retrieve All Analytics** from your Postman collections.



The screenshot shows a Postman interface with the following details:

- Body tab selected.
- Status: 200 OK, Time: 1042 ms.
- Headers (14) and Tests (0/0) tabs are visible.
- JSON dropdown is selected.
- Save button is present.
- Response body (Pretty):

```
1 {  
2   "analyticCatalogEntries": [  
3     {  
4       "taxonomyLocation": "/uncategorized",  
5       "supportedLanguage": "Java",  
6       "customMetadata": "{\"assetid\":\"abc\"}",  
7       "createdTimestamp": "2016-01-05T17:06:03+00:00",  
8       "updatedTimestamp": "2016-01-05T19:43:42+00:00",  
9       "author": "demo",  
10      "version": "v1",  
11      "description": "This analytic adds two numbers and returns the result.",  
12      "name": "demo adder java 1",  
13      "id": "bab4c1d0-6993-40a8-be18-f8a492cfdb56"  
14    },
```

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

```
"version": "v1",  
"name": "demo adder java 1",  
"id": "bab4c1d0-6993-40a8-be18-f8a492cfdb56"
```

- If not already done, add these values to your Postman environment variables.



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 webpage).

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:

```
= "UTF-8"?
ww.omg.org/spec/BPMN/20100524/MODEL
://www.w3.org/2001/XMLSchema-instance"
age="http://www.w3.org/1999/XPath" id="sid-81430087-7a44-4be3-8517-914faf923256"
="DSP-PM" typeLanguage="http://www.w3.org/2001/XMLSchema"
ion="http://www.omg.org/spec/BPMN/20100524/MODEL http://www.omg.org/spec/BPMN/2.0/20100501/
"http://activiti.org/bpmn">
onWithOneAnalytic" isExecutable="true">
4CB3485C-5E48-47F0-87DC-4E2CB7603CD4"
FA026B-2E3D-4D4F-BBAC-58BF7BE9E7B3</outgoing>
ionQuantity="1" id="sid-10001"
mpensation="false" name="<Analytic Catalog Entry Id>:<Analytic Name>:<Analytic Version>">
i:delegateExpression="${javaDelegate}"
ctiviti="http://activiti.org/bpmn">
FA026B-2E3D-4D4F-BBAC-58BF7BE9E7B3</incoming>
1CE38B-76FF-4E55-857F-ED08AB61A120</outgoing>
```



After editing:

```
'1" id="id-10001"
>false" name="bab4c1d0-6993-40a8-be18-f8a492cfdb56::demo adder java 1>::v1|  
expression="${javaDelegate}"  
@://activiti.org/bpmn">
```

- 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.

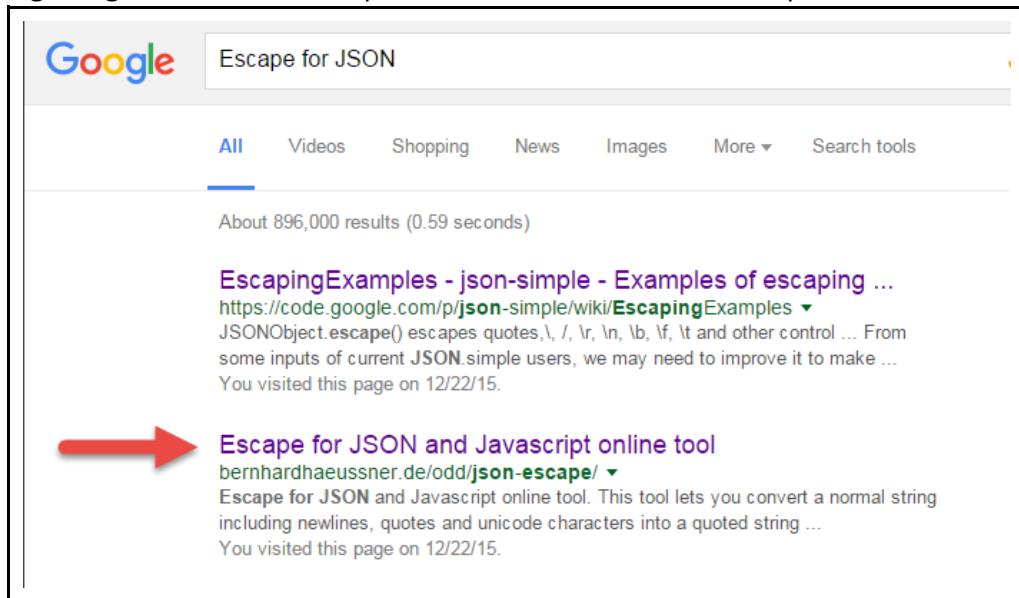
The request has two important fields: bpmnXml and analyticInputData.

- **bpmnXml**: you will be replacing the current value with the BPMN workflow XML from the previous step after you "Escape for JSON".
 - **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.

5. Next we will need to properly 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.

Lab 3: Runtime: Orchestrations and Job Schedules

Normal:

```
{
  "id": "Execution of Orchestration with One Analytic",
  "name": "Orchestration with One Analytic",
  "bpmnXml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n<definitions
  xmlns='http://www.omg.org/spec/BPMN/20100524/MODEL'\n  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'\n  expressionLanguage='http://www.w3.org/1999/XPath' id='sid-81430087-7a44-4be3-8517-914faf923256'\n    targetNamespace='DSP-PM'\n  typeLanguage='http://www.w3.org/2001/XMLSchema'\n  xsi:schemaLocation='http://www.omg.org/spec/BPMN/20100524/MODEL
  http://www.omg.org/spec/BPMN/2.0/20100501/BPMN20.xsd'\n  xmlns:activiti='http://activiti.org/bpmn'\n    <process
    id='OrchestrationWithOneAnalytic' isExecutable='true'\n      <startEvent
        name=''\n      </startEvent>\n    <outgoing>sid-66FA026B-2E3D-4D4F-BBAC-58BF7BE9E7B3</outgoing>\n    <serviceTask completionQuantity='1' id='sid-10001'\n
  
```

↓ escape ↓



Escaped & quoted:

```
{
  \"id\": \"Execution of Orchestration with One Analytic\",
  \"name\": \"Orchestration with One Analytic\",
  \"bpmnXml\": \"<?xml version=\\\\\\\"1.0\\\\\\\" encoding=\\\\\\\"UTF-8\\\\\\\"?>\\n<definitions
  xmlns=\\\\\\\"http://www.omg.org/spec/BPMN/20100524/MODEL\\\\\\\"\\n
  xmlns:xsi=\\\\\\\"http://www.w3.org/2001/XMLSchema-instance\\\\\\\"\\n
  expressionLanguage=\\\\\\\"http://www.w3.org/1999/XPath\\\\\\\" id=\\\\\\\"sid-81430087-7a44-4be3-8517-914faf923256\\\\\\\"\\n
    targetNamespace=\\\\\\\"DSP-PM\\\\\\\"\\n
  typeLanguage=\\\\\\\"http://www.w3.org/2001/XMLSchema\\\\\\\"\\n
  xsi:schemaLocation=\\\\\\\"http://www.omg.org/spec/BPMN/20100524/MODEL
  http://www.omg.org/spec/BPMN/2.0/20100501/BPMN20.xsd\\\\\\\"\\n
  xmlns:activiti=\\\\\\\"http://activiti.org/bpmn\\\\\\\"\\n    <process
    id=\\\\\\\"OrchestrationWithOneAnalytic\\\\\\\" isExecutable=\\\\\\\"true\\\\\\\"\\n      <startEvent
        name=\\\\\\\"\\\\\\\"\\n      <outgoing>sid-66FA026B-2E3D-4D4F-BBAC-58BF7BE9E7B3</outgoing>\\n      </startEvent>\\n    <serviceTask
  
```



6. Copy the "Escaped" text and paste/replace the contents of the bpmnxml field of the Orchestration Execution Request (body of the Postman **POST Run** from the **Orchestration Execution** window).

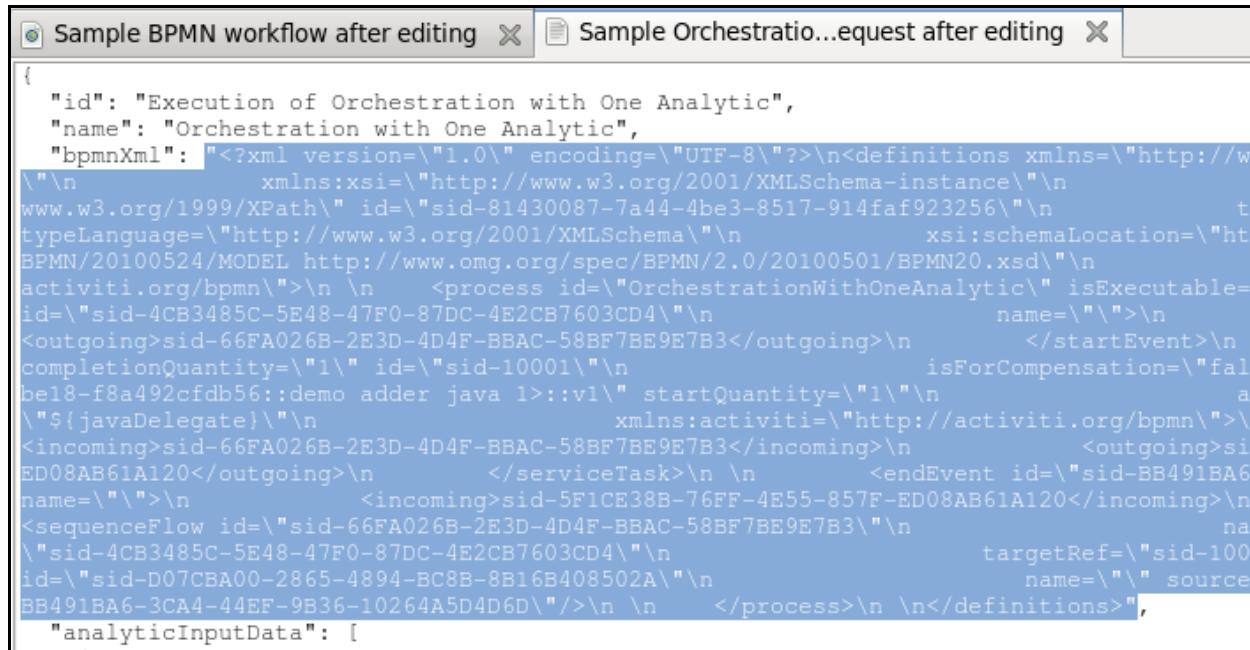
- Copy the "Escaped" text.



The screenshot shows a browser window with the title "Escaped & quoted:". The content of the window is a large block of XML code representing a BPMN process. The XML uses double quotes for escaping, as indicated by the red underlines and error markers. The code defines a process with an id of "OrchestrationWithOneAnalytic", a start event, a service task, and various outgoing and incoming connections. It includes attributes like completionQuantity, isForCompensation, and activiti:delegateExpression.

```
xmlns:activiti="http://activiti.org/bpmn">\n\n<process\n  id="OrchestrationWithOneAnalytic"\n    isExecutable="true">\n\n  <startEvent id="sid-4CB3485C-5E48-47F0-87DC-\n    4E2CB7603CD4"\n    name="\">\n\n  <outgoing>sid-66FA026B-2E3D-4D4F-\n    BBAC-58BF7BE9E7B3</outgoing>\n    </startEvent>\n    \n  <serviceTask completionQuantity="1"\n    id="sid-\n    10001"\n    isForCompensation="false"\n    name="bab4c1d0-6993-40a8-be18-f8a492cfdb56":demo adder java\n    1>::v1\n    startQuantity="1"\n    activiti:delegateExpression="${javaDelegate}"\n    xmlns:activiti="http://activiti.org/bpmn">\n\n  <incoming>sid-66FA026B-2E3D-4D4F-\n    BBAC-58BF7BE9E7B3</incoming>\n    <outgoing>sid-5F1CE38B-\n    76FF-4E55-857F-ED08AB61A120</outgoing>\n    </serviceTask>\n\n
```

- Paste into the bpmnXml field of the orchestration request text (replacing current text).



The screenshot shows two windows side-by-side. The left window is titled "Sample BPMN workflow after editing" and contains a JSON object with a single key-value pair: "bpmnXml": [long XML string]. The right window is titled "Sample Orchestratio...equest after editing" and also contains a JSON object with a single key-value pair: "bpmnXml": [long XML string]. Both XML strings are identical, representing an orchestration definition.

```
{
  "id": "Execution of Orchestration with One Analytic",
  "name": "Orchestration with One Analytic",
  "bpmnXml": "<?xml version='1.0' encoding='UTF-8'?><definitions xmlns='http://www.w3.org/2001/XMLSchema-instance' xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance' id='sid-81430087-7a44-4be3-8517-914faf923256' typeLanguage='http://www.w3.org/2001/XMLSchema' xsi:schemaLocation='http://BPMN/20100524/MODEL http://www.omg.org/spec/BPMN/2.0/20100501/BPMN20.xsd'>
<process id='OrchestrationWithOneAnalytic' isExecutable='true' name='Orchestration with One Analytic' startEvent='sid-66FA026B-2E3D-4D4F-BBAC-58BF7BE9E7B3' completionQuantity='1' id='sid-10001' isForCompensation='false' be18-f8a492cfdb56::demo adder java 1>::v1 startQuantity='1' name='Orchestration with One Analytic' a '${javaDelegate}' xmlns:activiti='http://activiti.org/bpmn'>
<incoming>sid-66FA026B-2E3D-4D4F-BBAC-58BF7BE9E7B3</incoming>
<outgoing>sid-66FA026B-2E3D-4D4F-BBAC-58BF7BE9E7B3</outgoing>
<serviceTask id='BB491BA6-ED08AB61A120' name='Orchestration with One Analytic' endEvent='sid-81430087-7a44-4be3-8517-914faf923256' targetRef='sid-10001' source='BB491BA6-3CA4-44EF-9B36-10264A5D4D6D'>
<sequenceFlow id='sid-66FA026B-2E3D-4D4F-BBAC-58BF7BE9E7B3' targetRef='sid-10001' name='Orchestration with One Analytic' source='BB491BA6-3CA4-44EF-9B36-10264A5D4D6D'>
</process>
</definitions>",
  "analyticInputData": []
}
```

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.



Working with Analytics

- Check your Request parameters, and click Send.

The screenshot shows the Postman interface with the 'Run 1 Orchestration' request selected. The left sidebar shows the 'Analytics' collection with various API endpoints. The main area displays the request details:

- Method:** POST
- URL:** `https://{{exec_host}}/api/v1/execution`
- Headers (4):**
 - Accept: application/json
 - Content-Type: application/json
 - Authorization: Bearer {{token}}
 - Predix-Zone-Id: {{runtime_tenant}}

A red arrow points to the 'Authorization' header field.

You should receive a response as follows:

The screenshot shows the Postman interface displaying the response body of the 'Run 1 Orchestration' request. The response is a JSON object:

```
1 [ { "orchestrationStatus": "COMPLETED", "contextID": "Execution of Orchestration with One Analytic_6", "analyticOutputData": [ { "analyticStepId": "sid-10001", "data": "\n \"result\": 29\n" } ], "output": null, "name": "Orchestration with One Analytic" } ]
```

- 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.predix.io",  
      "execution_uri": "https://predix-analytics-execution-release.run.aws-usw02-pr.ice.predix.io",  
      "scheduler_uri": "https://predix-scheduler-service-release.run.aws-usw02-pr.ice.predix.io",  
      "zone-htcp-header-name": "Predix-Zone-ID",  
      "zone-htcp-header-value": "ca071d1d-0665-471e-84d1-508c0c56d549",  
      "zone-oauth-scope": "analytics.zones.ca071d1d-0665-471e-84d1-508c0c56d549.user"  
    },  
    "label": "predix-analytics-runtime",  
    "name": "predix-analytics-runtime-JaimeVillanueva",  
    "plan": "Beta",  
    "tags": []  
  }]
```

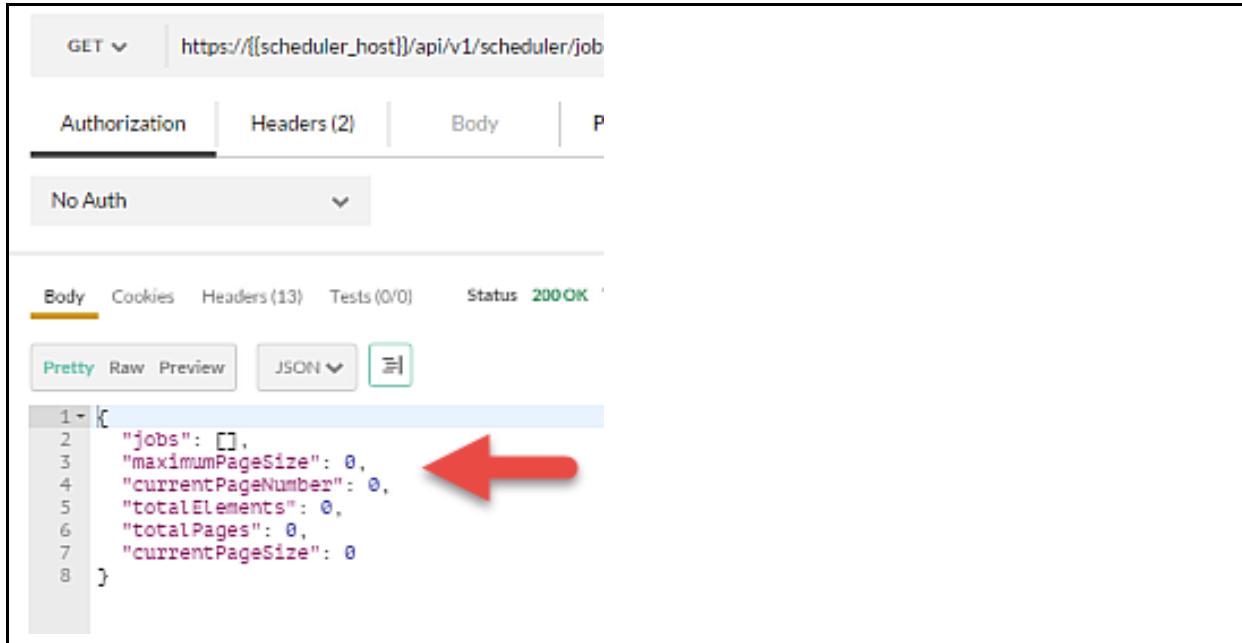


Working with Analytics

The screenshot shows a 'Manage environments' dialog box with the title 'Analytics'. It lists several environment variables with their corresponding values. A red arrow points to the 'Submit' button at the top right of the form.

Key	Value
UAA Token URL	58a17288-7004-4aff-bb37-88ed04cC
X-Identity-Zone-Id	58a17288-7004-4aff-bb37-88ed04cC
token	eyJhbGciOiJSUzI1NiJ9.eyJqdGkiOiJt
catalog_host	predix-analytics-catalog-release.run.a
catalog_tenant	d283e834-e057-4d08-823a-d7fdb07
Content-Type	application/json
analyticId	f39151f8-1480-42a7-9321-3c2e66bc
validationRequestId	546b4871-b72c-4b4f-82e5-57a6358
config_host	predix-analytics-config-release.run.aw
runtime_tenant	ca071d1d-0665-471e-84d1-508c0c5
exec_host	predix-analytics-execution-release.rui
scheduler_host	predix-scheduler-service-release.run.:

- Click **Submit** to save the variable.
- Click **Send**, and you should receive a response as follows:



GET [https://\[scheduler_host\]/api/v1/scheduler/job](https://[scheduler_host]/api/v1/scheduler/job)

Authorization Headers (2) Body P

No Auth

Body Cookies Headers (13) Tests (0/0) Status 200 OK

Pretty Raw Preview JSON ↗

```
1 - [ { "jobs": [], "maximumPageSize": 0, "currentPageNumber": 0, "totalElements": 0, "totalPages": 0, "currentPageSize": 0 } ]
```

Note that you currently do not have jobs scheduled.

2. Schedule a job.

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



Working with Analytics

The screenshot shows the Postman application interface for creating a new job. On the left, there's a sidebar with various project and request lists. The main area is titled 'Create Job' and shows a POST request to 'https://[scheduler_host]/api/v1/scheduler/jobs'. The 'Body' tab is selected, showing a JSON payload:

```
1 {"name": "test1",
2  "description": "test description",
3  "state": "Active",
4  "cron": {
5    "seconds": "0/5",
6    "minutes": "*",
7    "hours": "*",
8    "dayOfMonth": "*",
9    "months": "*",
10   "dayOfWeek": "?",
11   "years": "*",
12   "timeZoneId": "UTC"
13 },
14 "executionRequest": {
15   "url": "http://some.url",
16   "httpMethod": "GET",
17   "httpHeaders": [
18     {
19       "name": "Content-Type",
20       "value": "application/json"
21     },
22     {
23       "name": "Predix-Zone-Id",
24       "value": "{{runtime_tenant}}"
25     }
26   ],
27   "inputData": "{\"number1\":1,\"number2\":2}"
28 }
29 }
30 }
```

- 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.

```

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

Note: Be sure to include the open/closing quotation marks for the **inputData** field value.

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



The screenshot shows a Postman request response. The status bar at the top indicates "Status 201 Created Time 4935 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 with red arrows pointing to the "cron" and "inputData" fields. The "Pretty" tab is also visible.

```
1 - {
2 -   "cron": {
3 -     "timeZoneId": "UTC",
4 -     "months": "*",
5 -     "hours": "*",
6 -     "minutes": "0/2",
7 -     "seconds": "0",
8 -     "dayOfMonth": "*",
9 -     "dayOfWeek": "?",
10 -    "years": "*"
11 -  },
12 -  "createdBy": "admin",
13 -  "createdTimestamp": "2015-12-23 23:00:57.046",
14 -  "executionRequest": {
15 -    "inputData": "\n      \\"id\\": \\"Execution of Orchestration with One Analytic\\",\n      \\"name\\": \\"Orchestration with One Analytic\\",\n      \\"bpmnXml\\": \\"<?xml version='1.0' encoding='UTF-8'?>\n<definitions xmlns='http://www.omg.org/spec/BPMN/20100524/MODEL'\n  <schedulers>\n    <orchestrator id='Orchestrator1' type='orchestrator'>\n      <processes>\n        <process id='Process1' type='process'>\n          <activities>\n            <script activityId='Activity1' type='script' language='BPMN20' code='<!-- Your BPMN20 code here -->'>\n            </activities>\n          </process>\n        </processes>\n      </orchestrator>\n    </orchestrators>\n  </schedulers>\n</definitions>\n"
16 -  }
17 - }
```

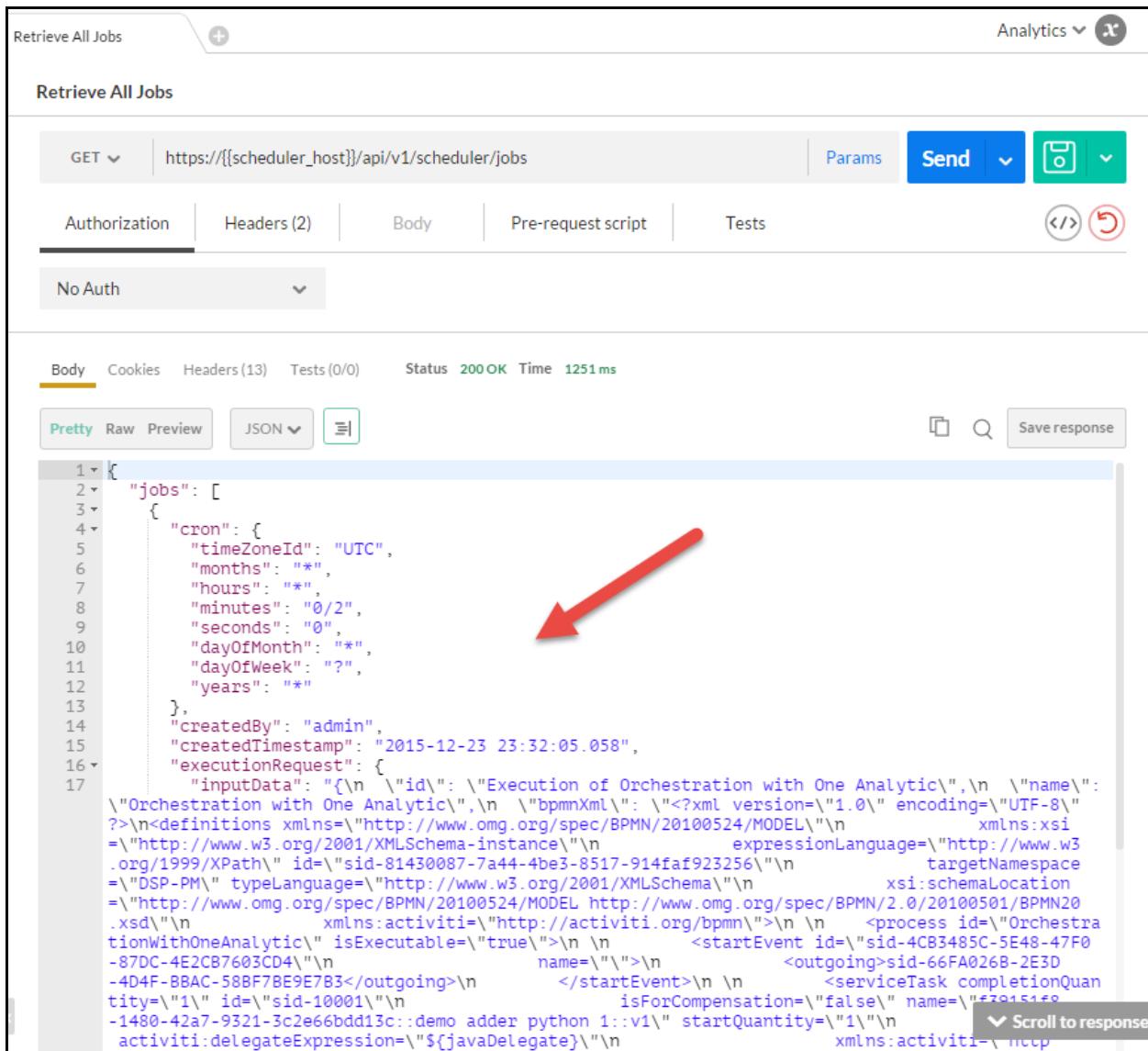
- Note the status returned of "201 Created" and the resulting job schedule.
- Update your Postman environment variables with the jobId.

4. 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:

Lab 3: Runtime: Orchestrations and Job Schedules



Retrieve All Jobs

Analytics x

Retrieve All Jobs

GET https://{{scheduler_host}}/api/v1/scheduler/jobs

Params Send Send Copy Save

Authorization Headers (2) Body Pre-request script Tests

No Auth

Body Cookies Headers (13) Tests (0/0) Status 200 OK Time 1251 ms

Pretty Raw Preview JSON Copy Save response

```

1 "jobs": [
2   {
3     "cron": {
4       "timeZoneId": "UTC",
5       "months": "*",
6       "hours": "*",
7       "minutes": "0/2",
8       "seconds": "0",
9       "dayOfMonth": "*",
10      "dayOfWeek": "?",
11      "years": "*"
12    },
13    "createdBy": "admin",
14    "createdTimestamp": "2015-12-23 23:32:05.058",
15    "executionRequest": {
16      "inputData": "{\n        \"id\": \"Execution of Orchestration with One Analytic\", \"name\": \"Orchestration with One Analytic\", \"bpmnXml\": \"<?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'\n  typeLanguage='http://www.w3.org/2001/XMLSchema'\n  =\"http://www.omg.org/spec/BPMN/20100524/MODEL http://www.omg.org/spec/BPMN/2.0/20100501/BPMN20.xsd'\n  xmlns:activiti='http://activiti.org/bpmn'\">\n        <process id='OrchestrationWithOneAnalytic'\n          isExecutable='true'\n          <startEvent id='sid-4CB3485C-5E48-47F0-87DC-4E2CB7603CD4'\n            name=''\n            <outgoing>sid-66FA026B-2E3D-4D4F-BBAC-58BF7BE9E7B3</outgoing>\n          </startEvent>\n          <serviceTask completionQuantity='1'\n            id='sid-10001'\n            isForCompensation='false'\n            name='f70151f9-1480-42a7-9321-3c2e66bdd13c::demo adder python 1:v1'\n            startQuantity='1'\n            activiti:delegateExpression='${javaDelegate}'\n            xmlns:activiti=''\n          </serviceTask>\n        </process>\n      
```

Scroll to response



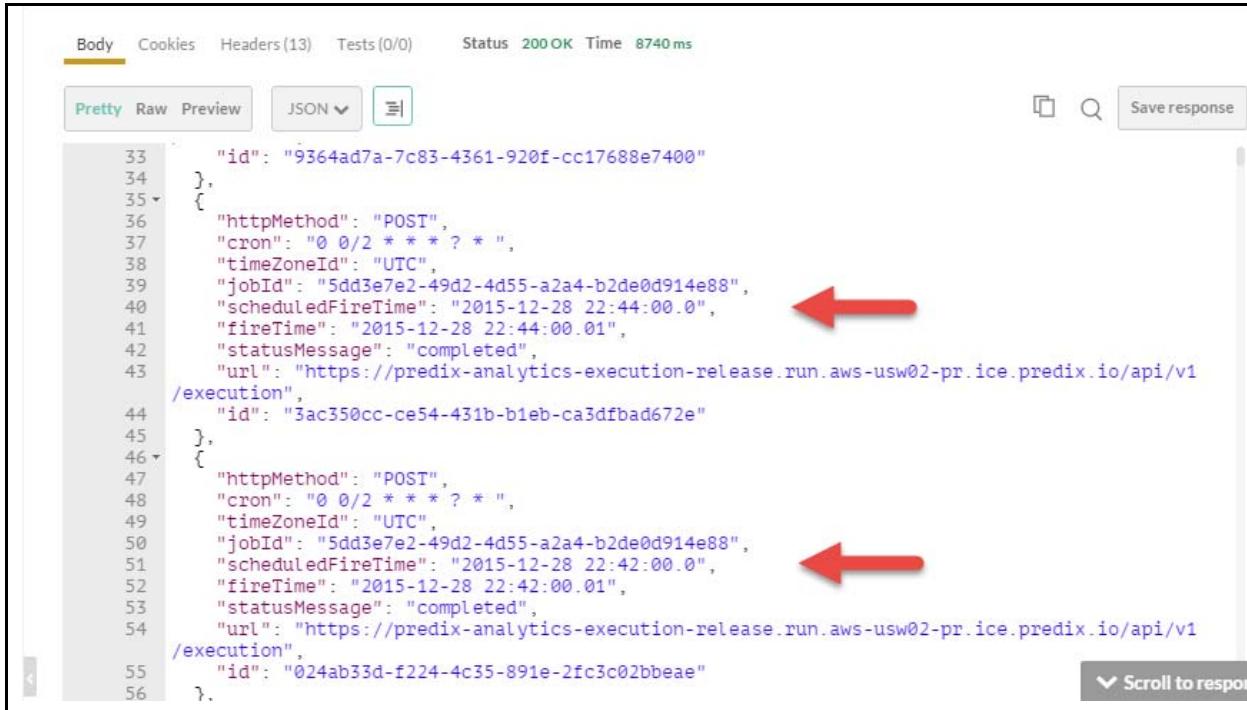
Scrolling down, you'll see the the JobID and status:

```
31     |   }
32     |   ],
33     |   "url": "https://predix-analytics-execution-release.run.aws-usw02-pr.ice.predix.io/api/v1
34     |   /execution"
35     |   },
36     |   "updatedBy": "admin",
37     |   "updatedTimestamp": "2015-12-23 23:32:05.058",
38     |   "description": "test description",
39     |   "name": "test1",
40     |   "id": "5dd3e7e2-49d2-4d55-a2a4-b2de0d914e88",
41     |   "state": "ACTIVE"
42 }
```



5. View the Job schedule results by Get Job History by JobID

- 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 REST API response in JSON format. The response body contains two job execution entries, each with a timestamp in the 'fireTime' field. Two red arrows point to these 'fireTime' fields.

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

You can see the job being run every 2 minutes. Note the time stamp and the 2 minute intervals shown in the history.

Exercise Summary

In this exercise you learned how to:

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



