Predix

Working with Analytics

Student Lab Guide

January 2016

GE Digital

Predix

GE Digital

Predix

Getting Started

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

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).

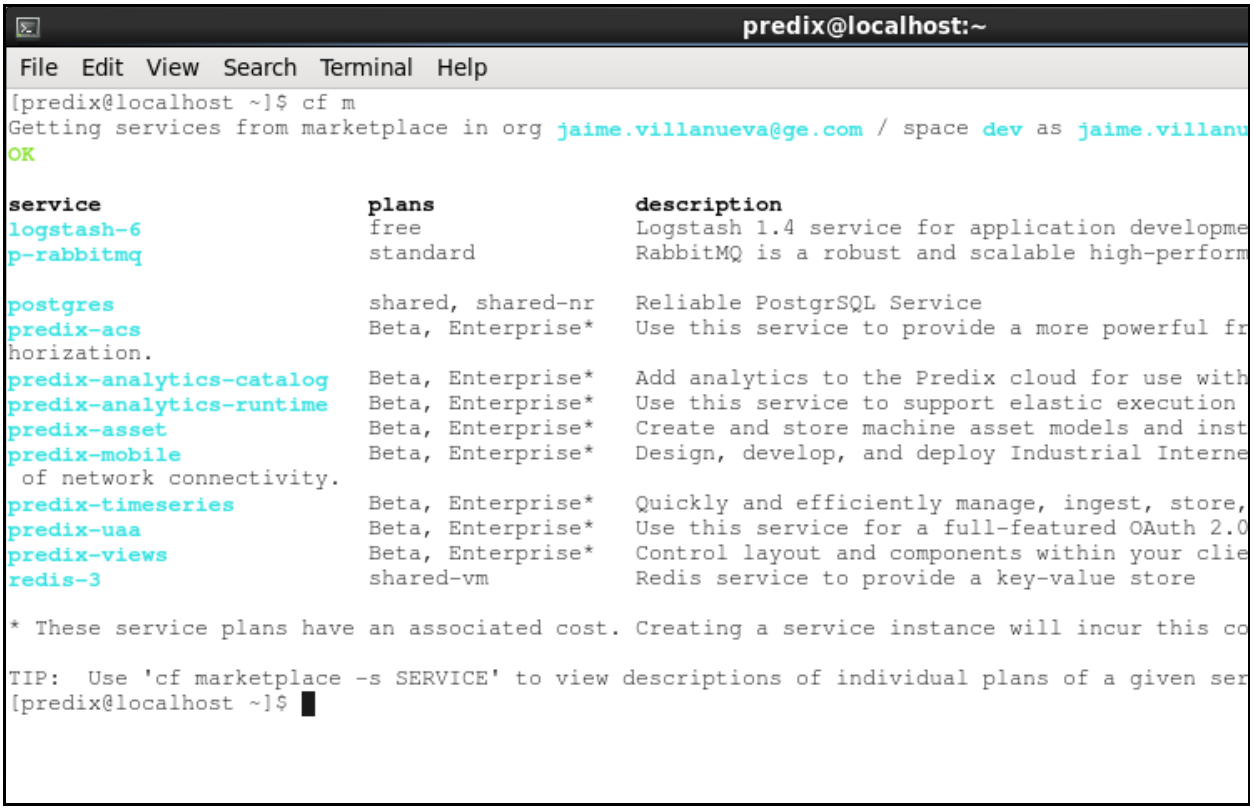
   In the Terminal, run the command:
   `cf login -a <API Endpoint provided by your instructor>`

- 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)`



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)`

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>"}'`

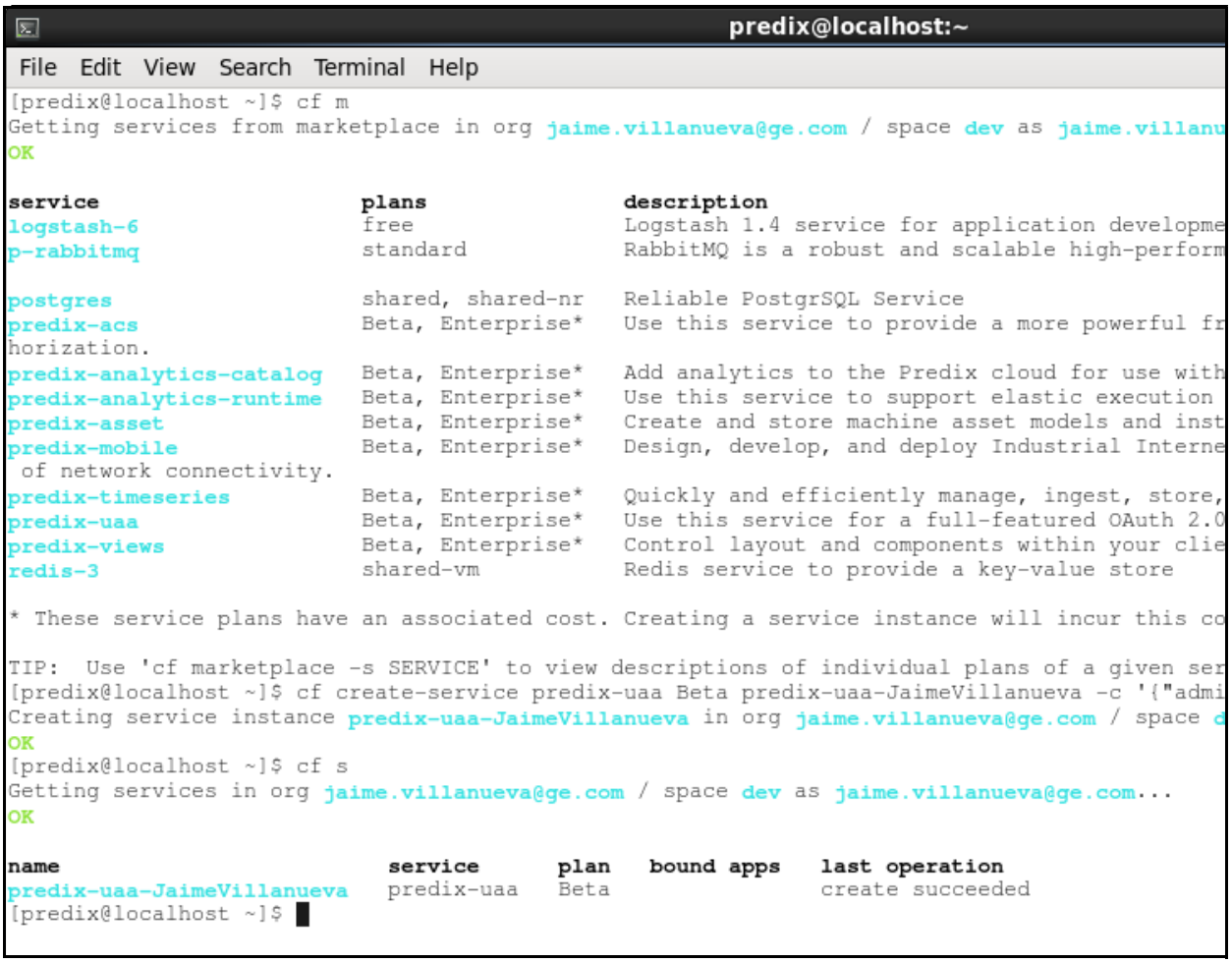
where:

- <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":"myadminsecret"}'
```

You should receive the following response.



- 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 application[1].
- 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.

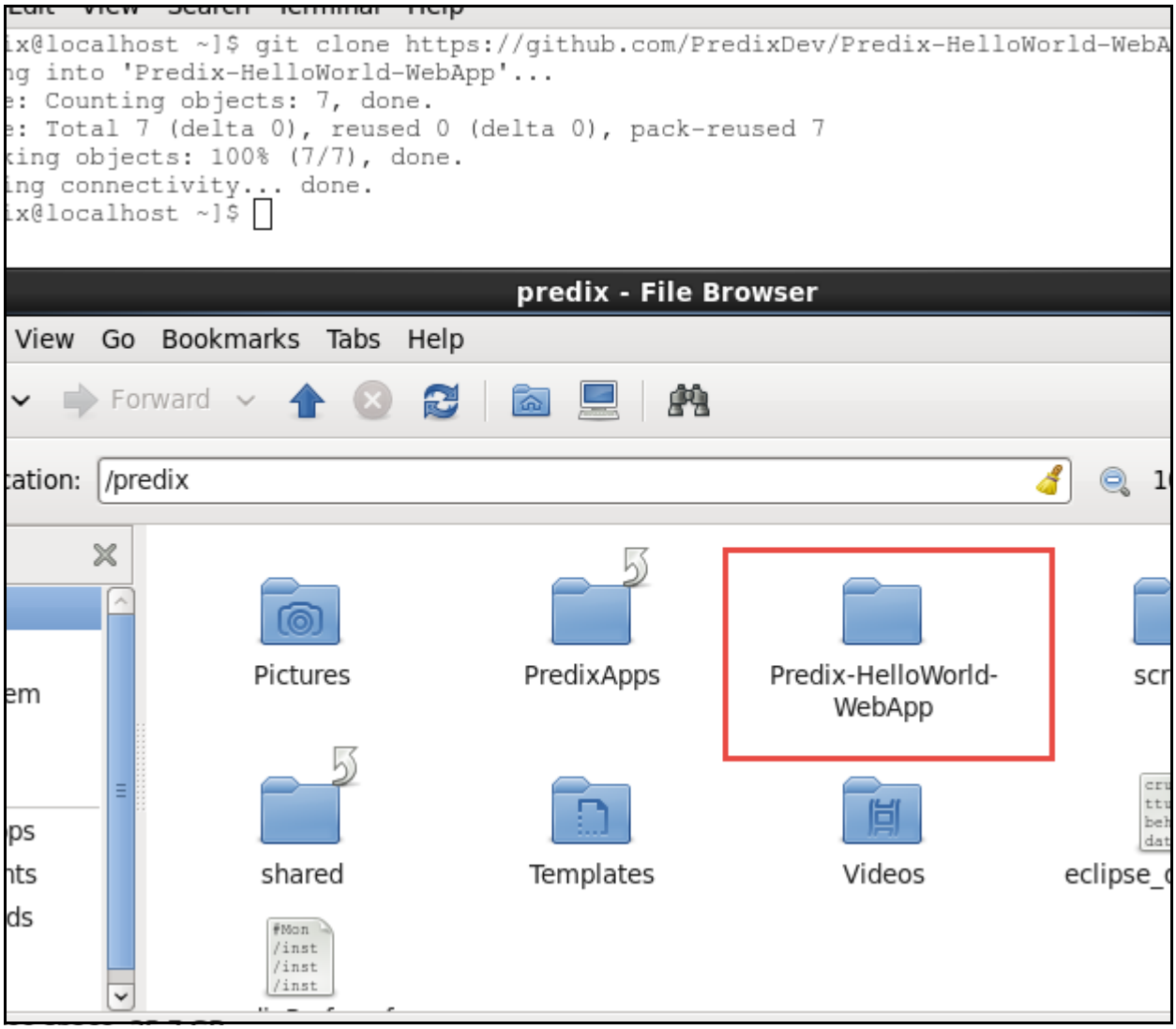
Binding an application to our UAA service instance will allow you to view various environment variables.

Steps

1. Get the Predix Hello World Starter Pack

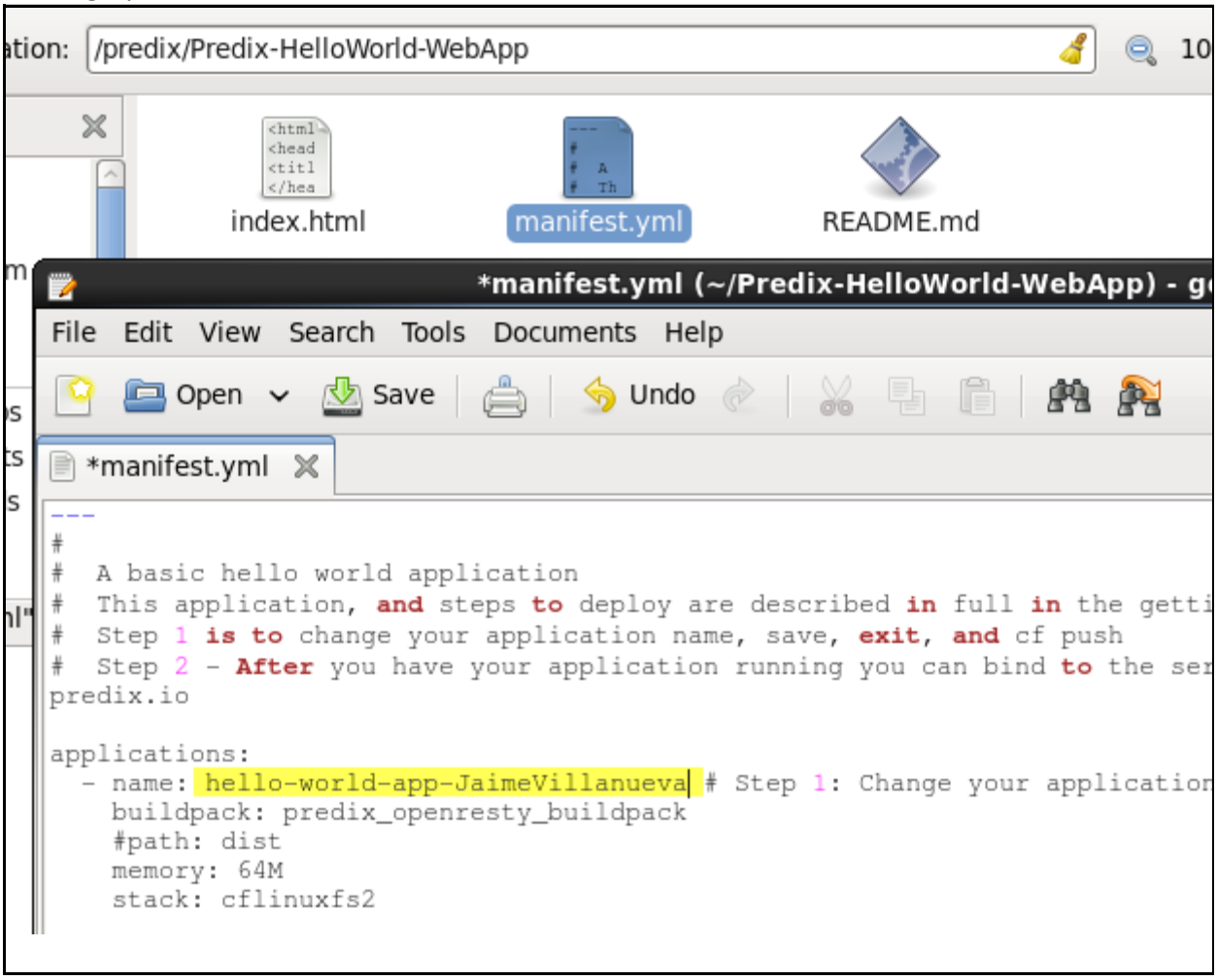
- From github (This github team site requires autorized access.)**:**
  `git clone https://github.com/PredixDev/Predix-HelloWorld-WebApp`

- **Alternate**: Your development environment may already include this folder and contents at **/predix/Predix-HelloWorld-WebApp**.


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 ~]$
predix - File Browser
View Go Bookmarks Tabs Help
Forward
cation: /predix
Pictures
PredixApps
Predix-HelloWorld-WebApp
shared
Templates
Videos
eclipse_

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 d
-----> 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.villan
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.predix.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-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           d
#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.

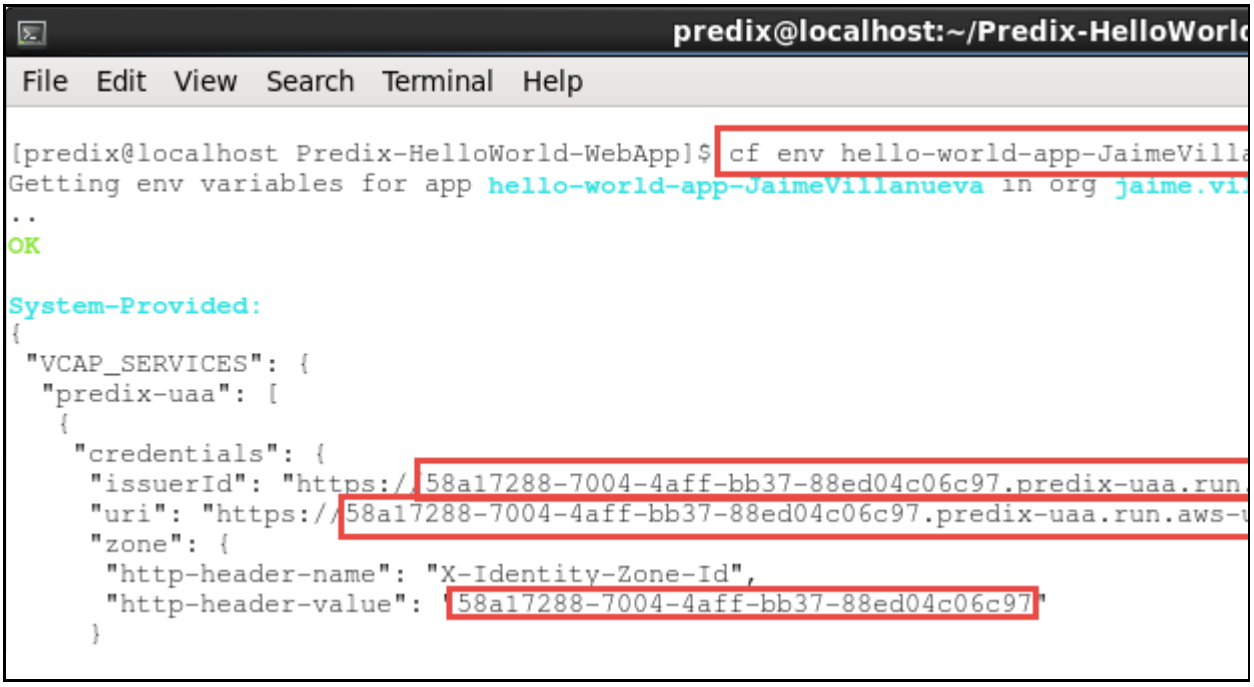
`cf bind-service (or cf bs) <app_name> <service_instance_name>`

For example:

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

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

`cf env <appname>`



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

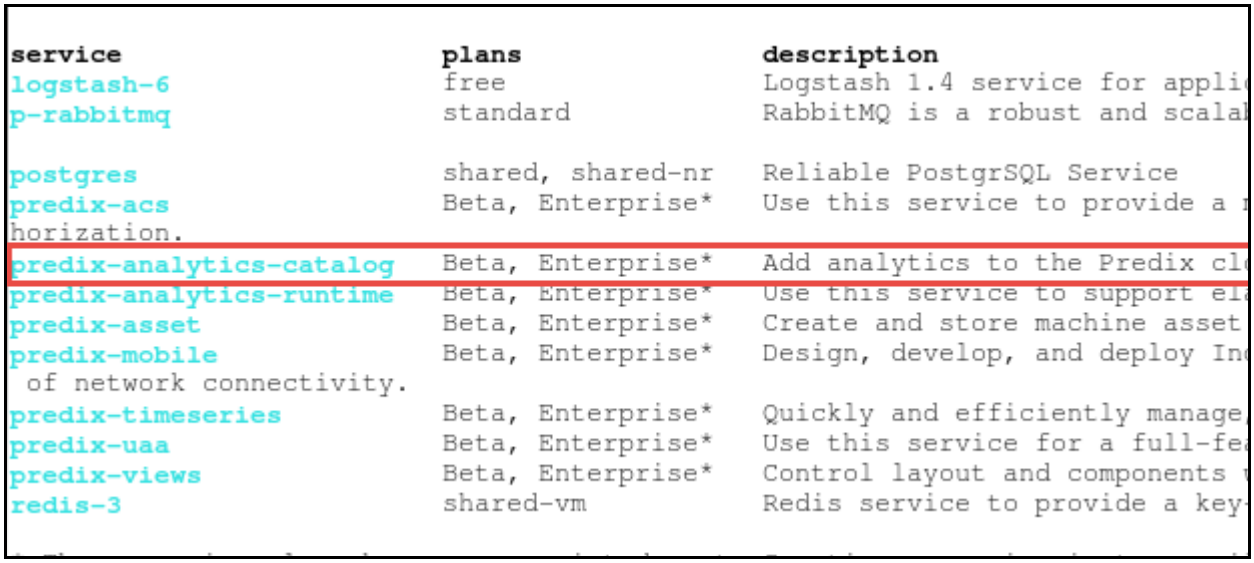
Before you begin this exercise, make sure that:

- 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)
```



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

.

2. Create your catalog service instance.

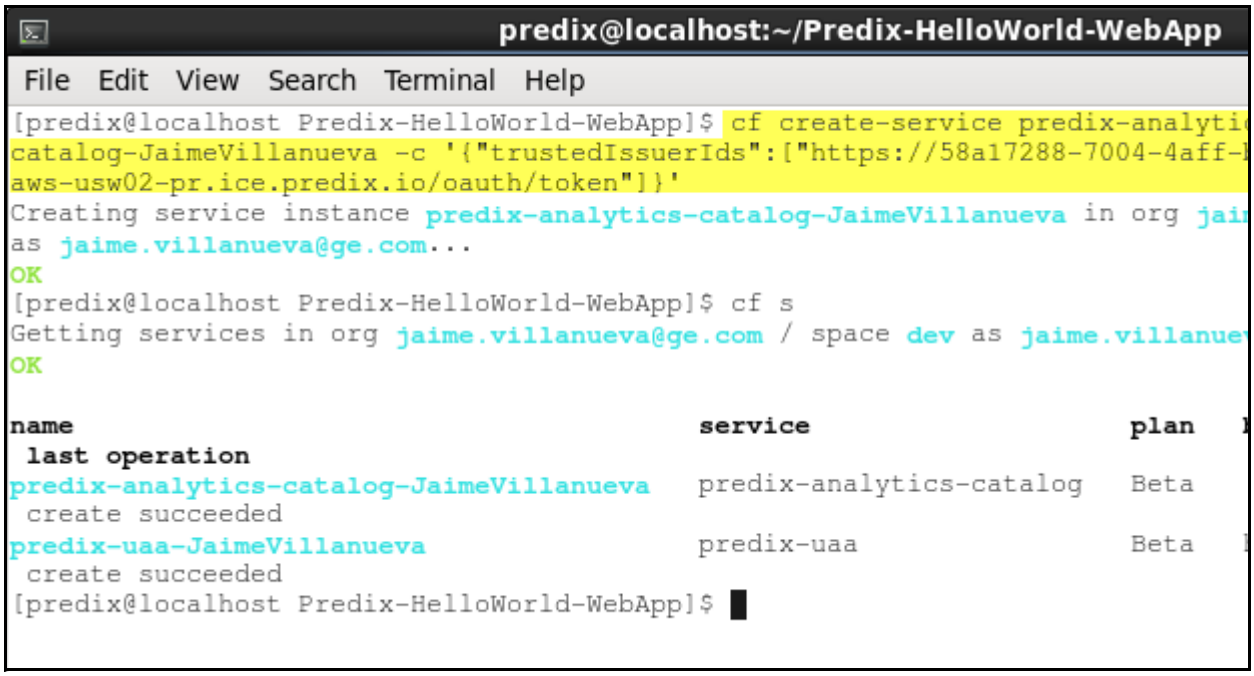
```
cf create-service predix-analytics-catalog <plan>
<my_service_instance> -c
'{"trustedIssuerIds":["<uaa_instance1_issuerId>",
"<uaa_instance2_issuerID>"]}'
```

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"]}'
```



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
```

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)
```

```
File  Edit  View  Search  Terminal  Help
[predix@localhost ~]$ cf m
Getting services from marketplace in org jaime.villanueva@ge.com / space dev
OK

service                      plans              description
logstash-6                   free               Logstash 1.4 service for appli
p-rabbitmq                   standard           RabbitMQ is a robust and scala

postgres                     shared, shared-nr  Reliable PostgrSQL Service
predix-acs                   Beta, Enterprise*  Use this service to provide a
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
```

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_ins
tance2_issuerID>"]}'
```

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

```
[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:

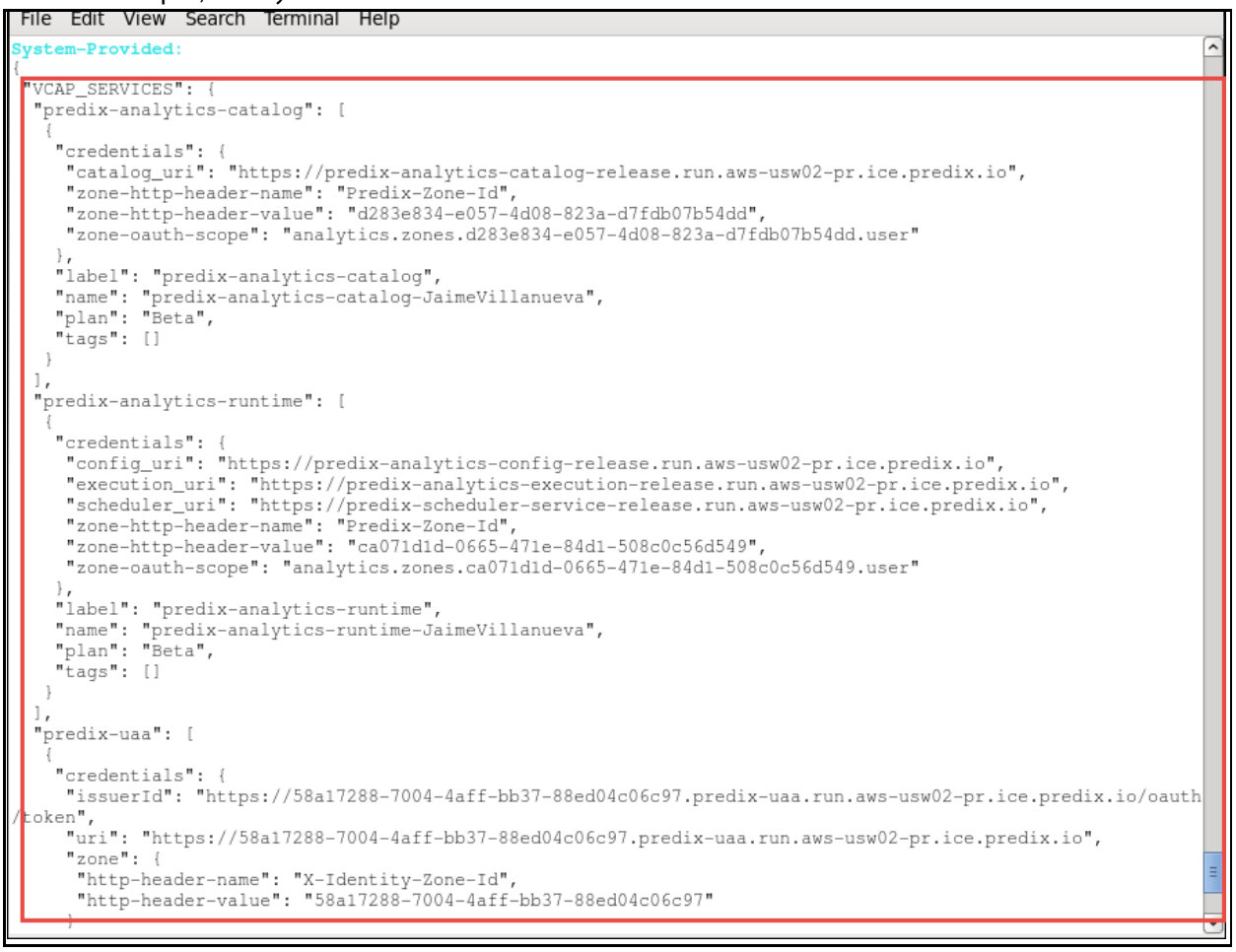
`cf bs hello-world-app-JaimeVillanueva predix-analytics-runtime-JaimeVillanueva`

```
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.
illanueva@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>`

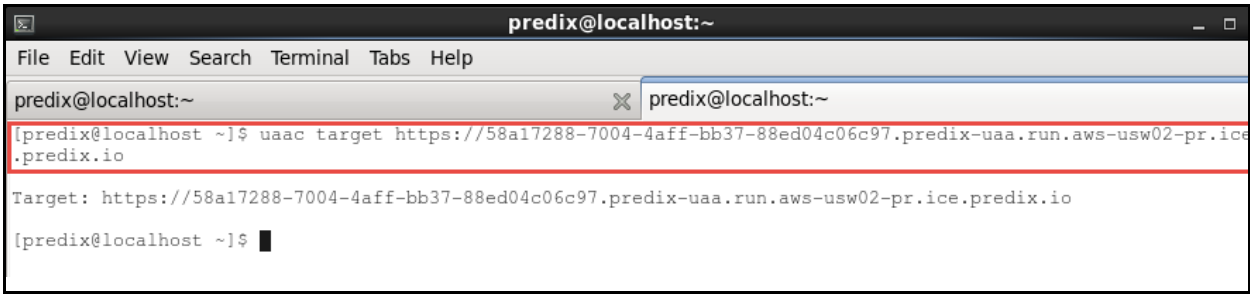
where:

<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.

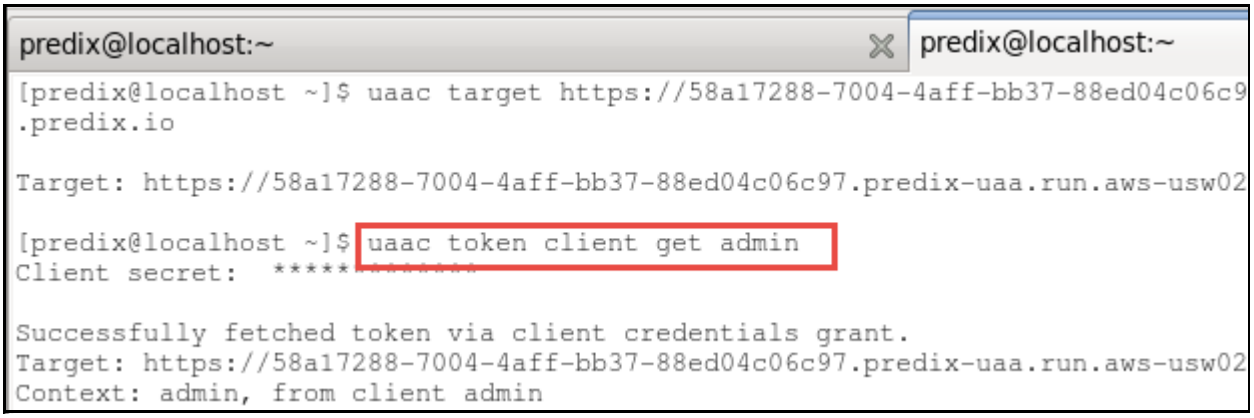


2. Log in to your UAA service instance.

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.



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

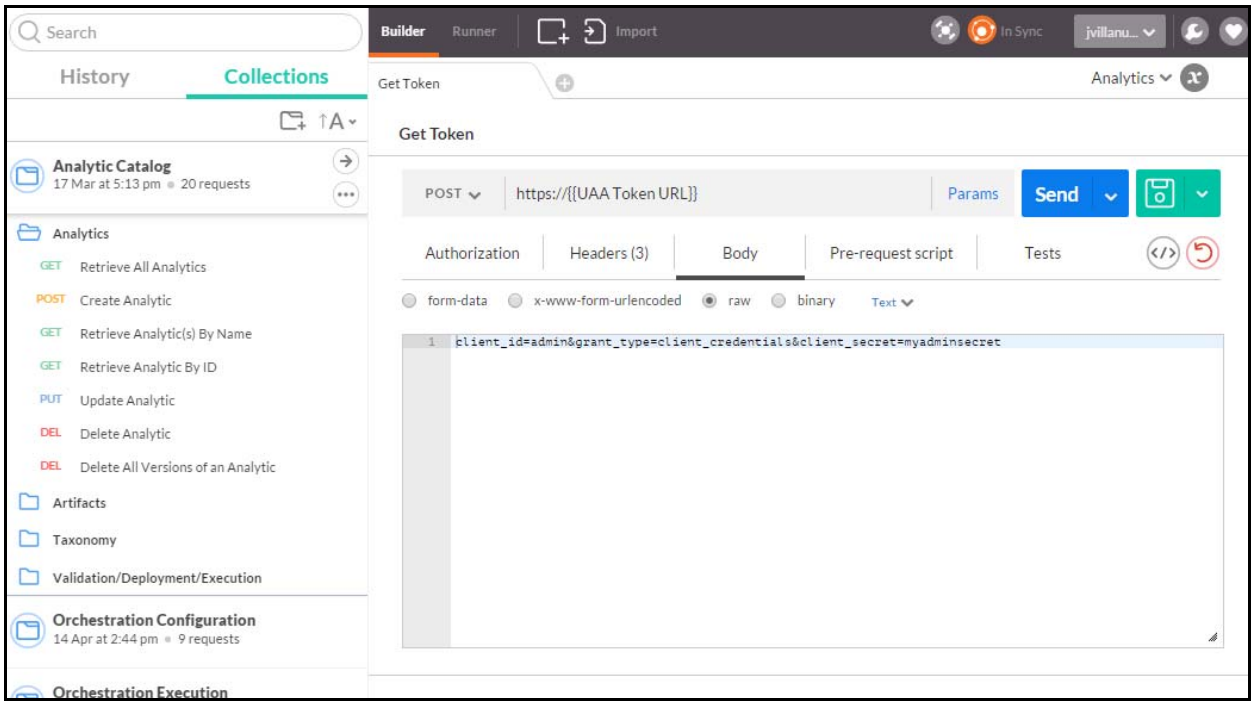
Exercise 1: Using Postman and Getting Your UAA Token Value

Overview

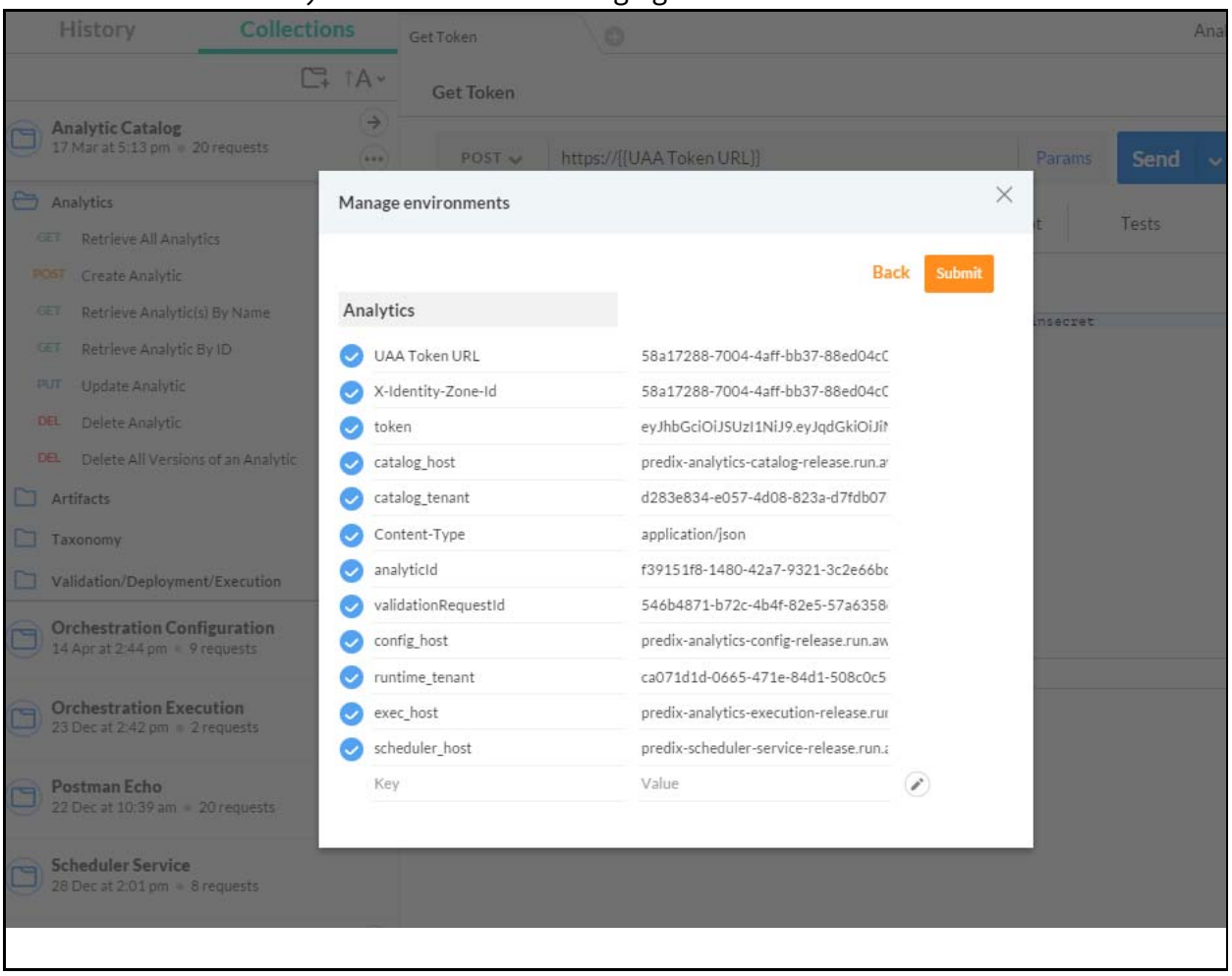
In this exercise, you will be using Postman, a Chrome browser extension, which allows you to:

- 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:



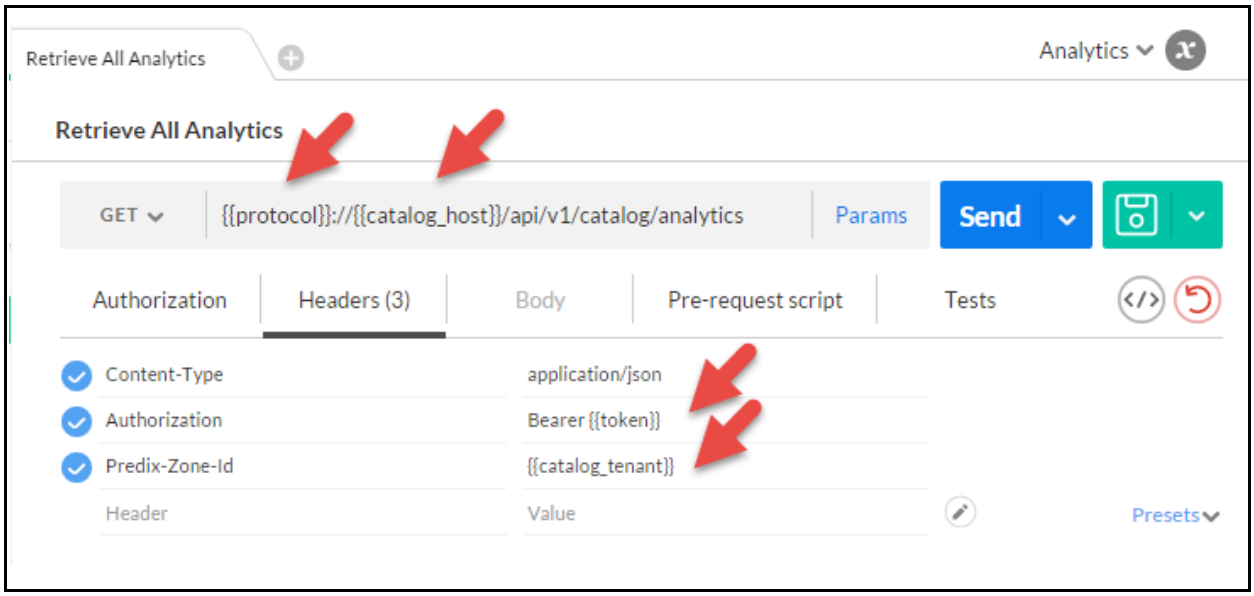
Environment values may be viewed when managing the environments.



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 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.

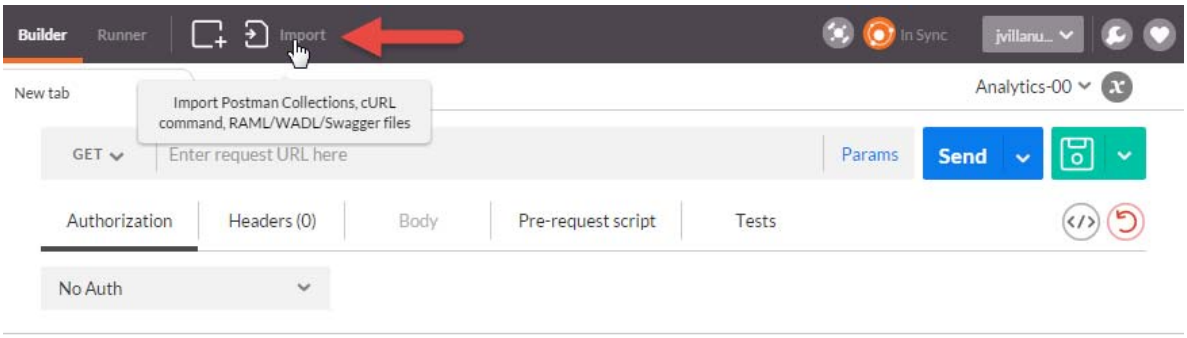
Steps

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

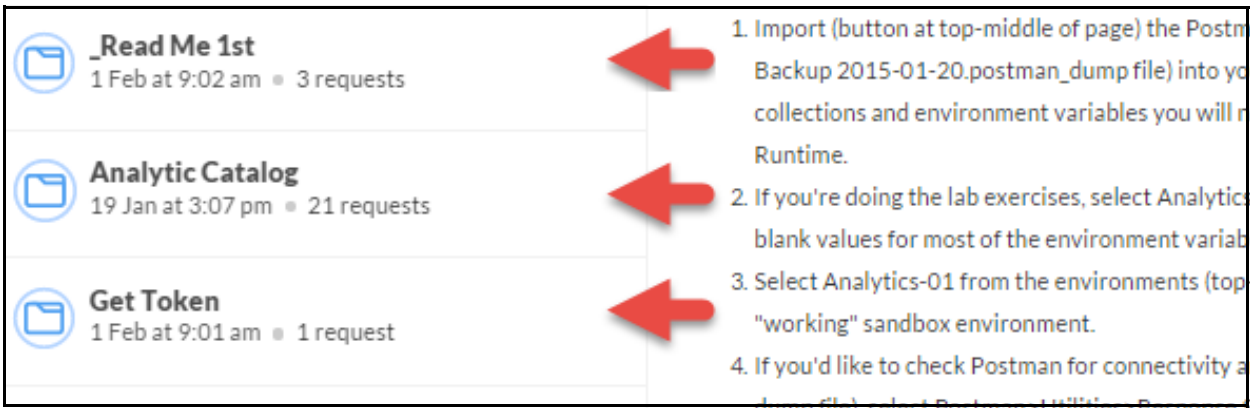
- From the Dev box, click on Applications>Internet> Chromium Browser
- Go to getpostman.com and click on **Get The App**.
- Launch Postman, from the main menu > applications > chromium apps > Postman

2. Import Postman collections to facilitate API requests.

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

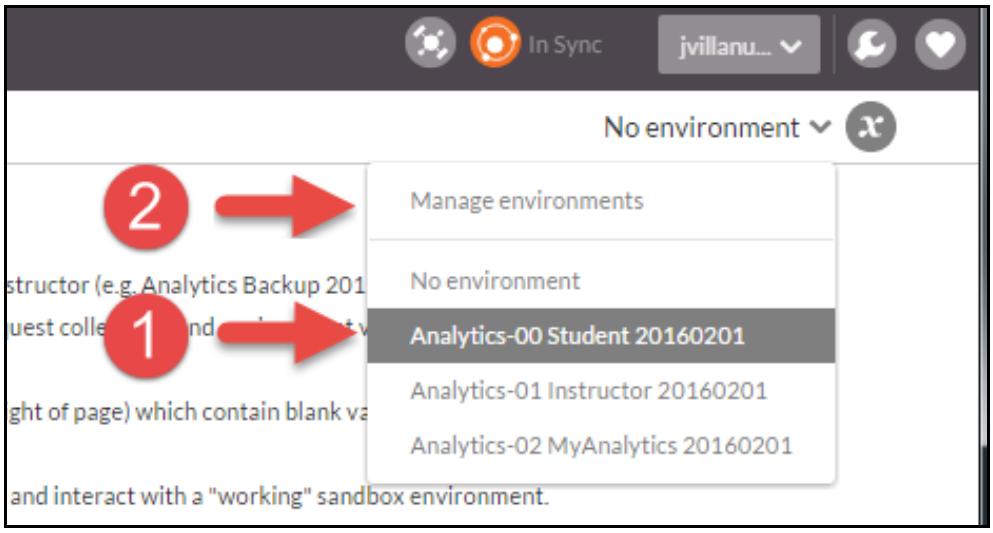


- Browse for the Postman "dump" file (provided by instructor)
- Click **Import**.
- Click on the Collections tab of Postman which should now contain the collections as shown (partial list):



3. Set your Postman environment for Analytics.

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



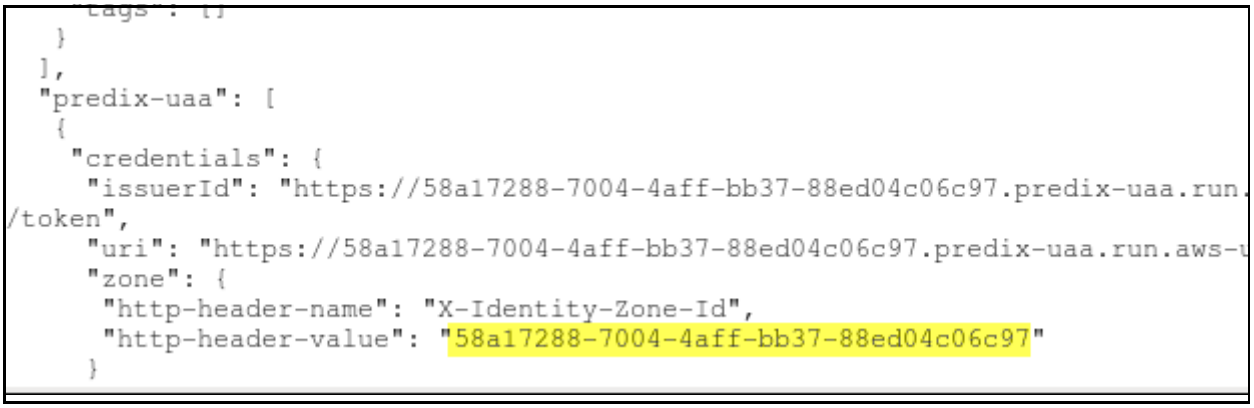
4. Edit and update the variables.

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

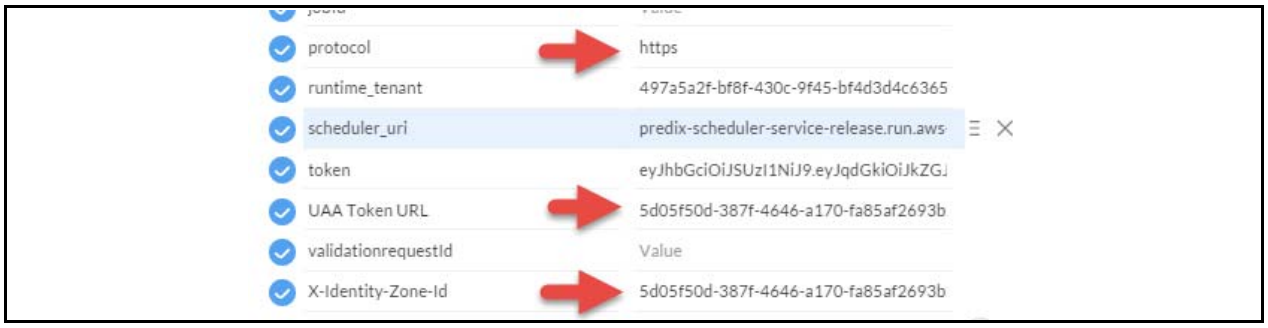
Where:

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

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

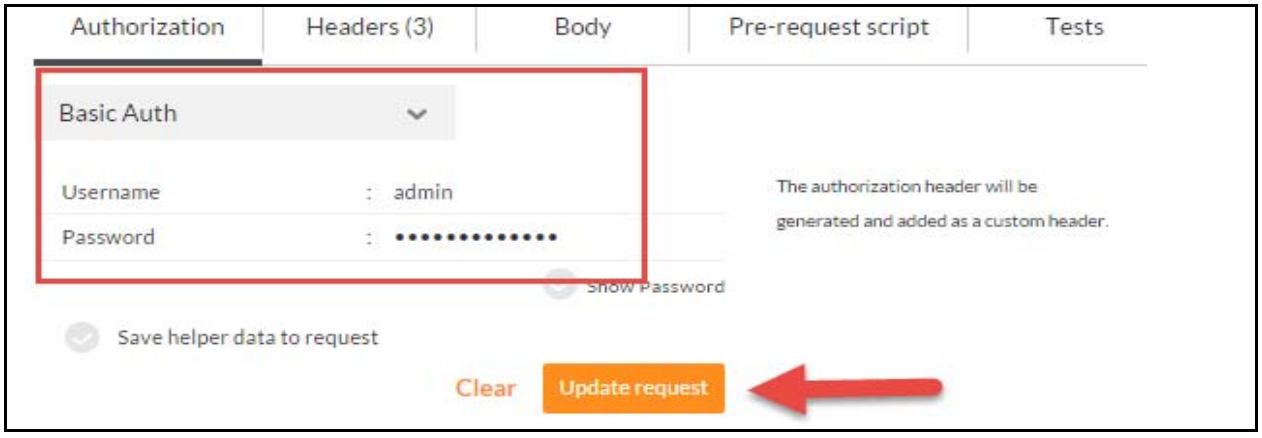


The variables should be set as follows:



5. Create your Authorization header.

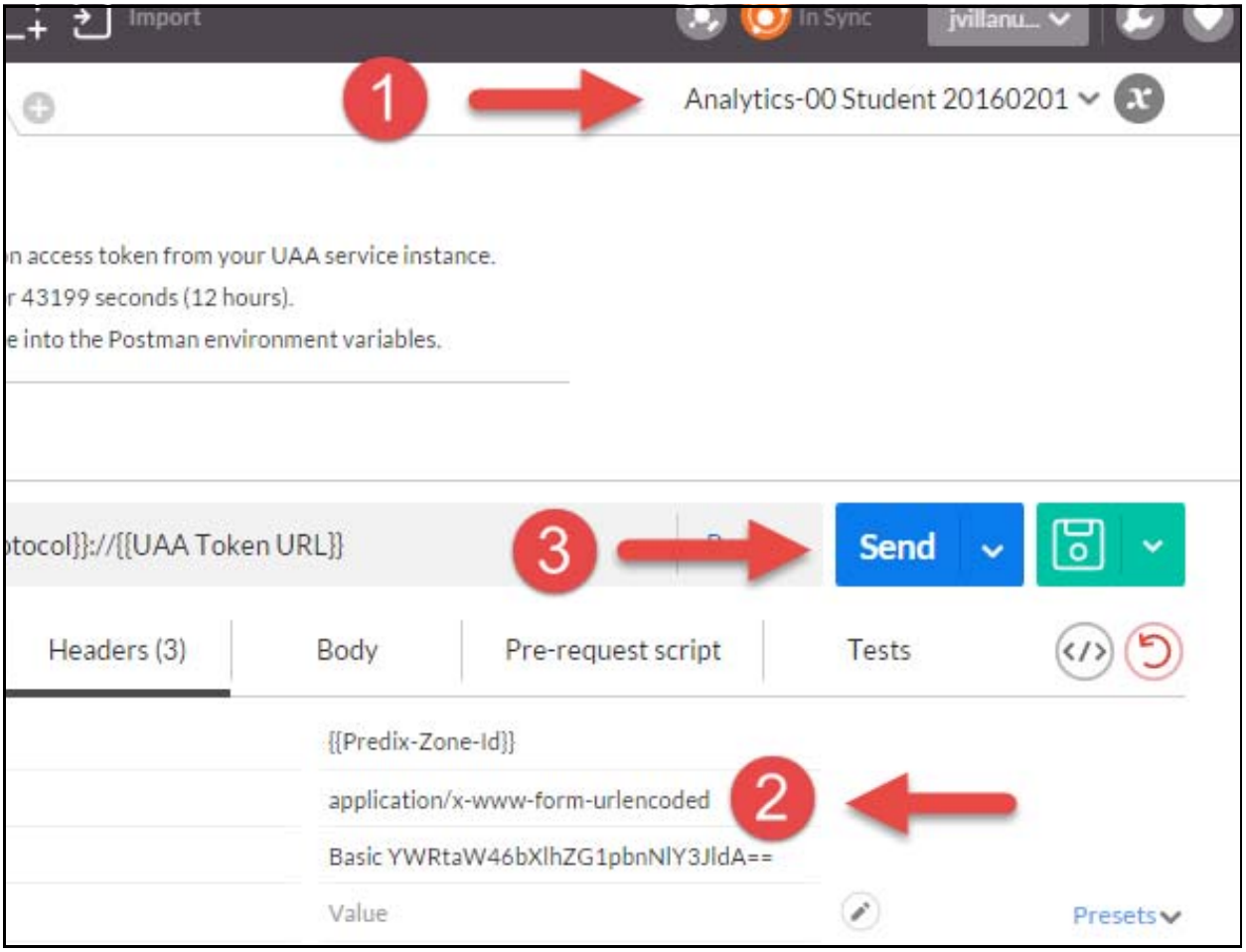
- Click on the Authorization tab
- Select Basic Authorization for:
  - Username: admin
  - Password: myadminsecret
- Click on **Update request**



This will add the Authorization header to your request.

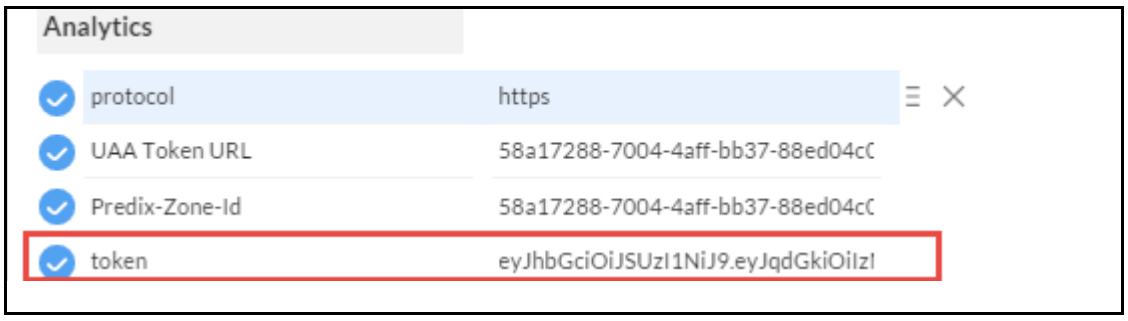
6. Verify your request and click **Send**.

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



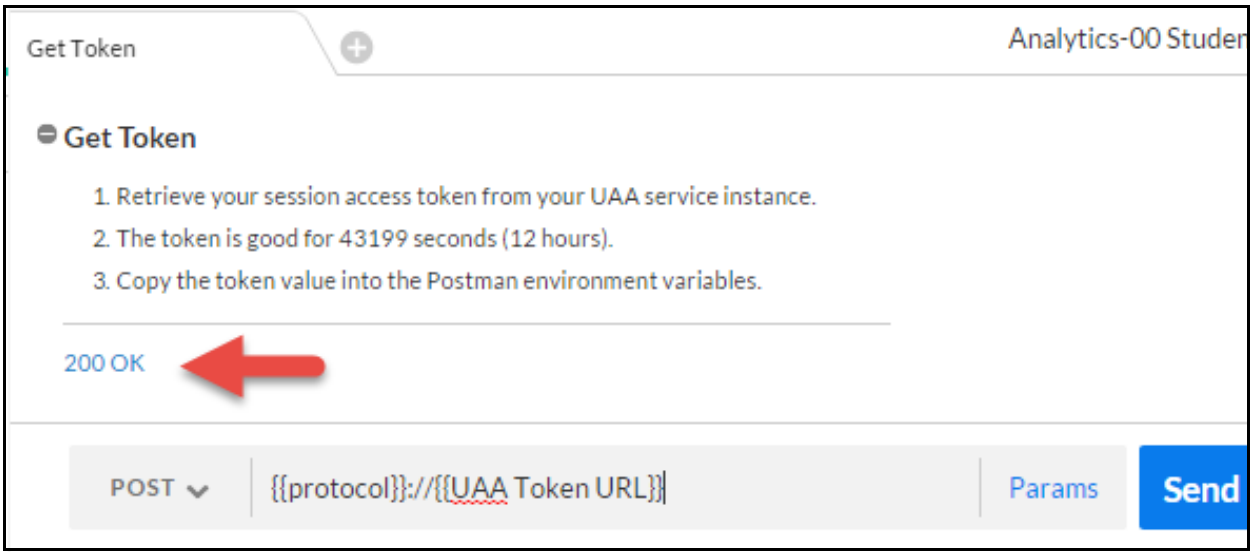
7. After clicking **Send**, you should receive the following response:

- 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.



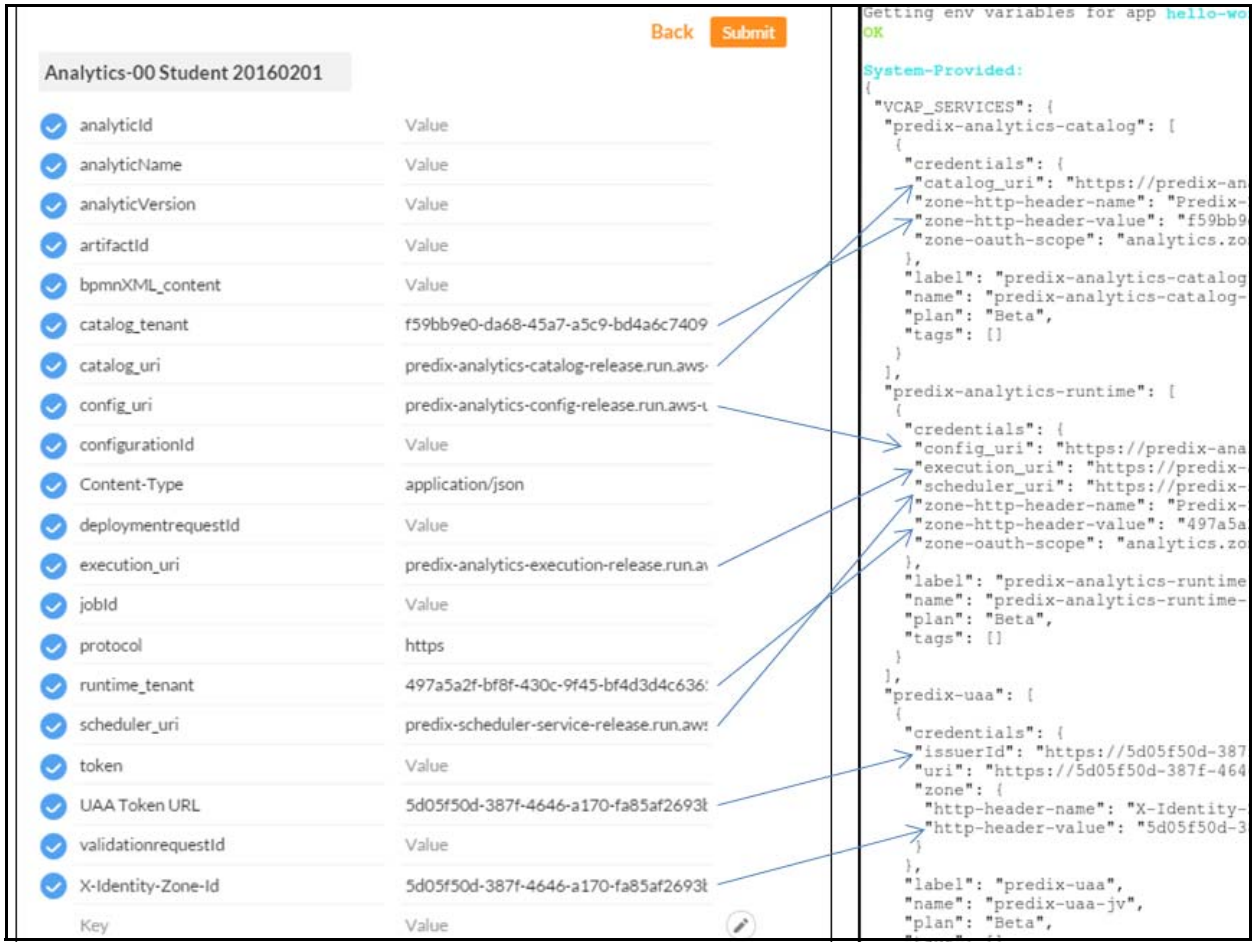
1. Click **Save Response** (to save the response to your Get Token request).

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



2. Enter all the required Postman environment variables.

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



- 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

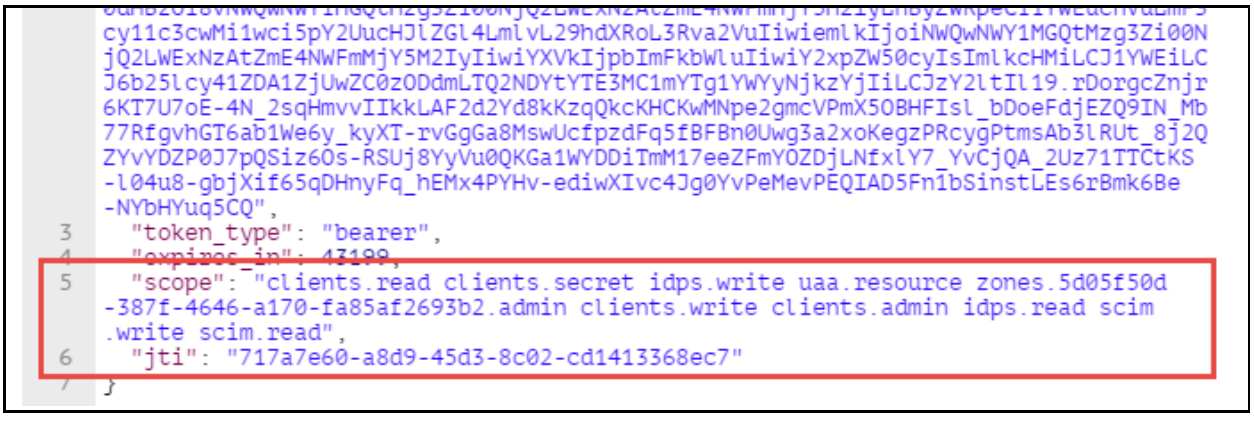
To enable applications to access the Analytics Catalog and Runtime services, your JSON Web Token (JWT) must contain the following scope(s) for both the catalog and runtime services:

`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.



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:

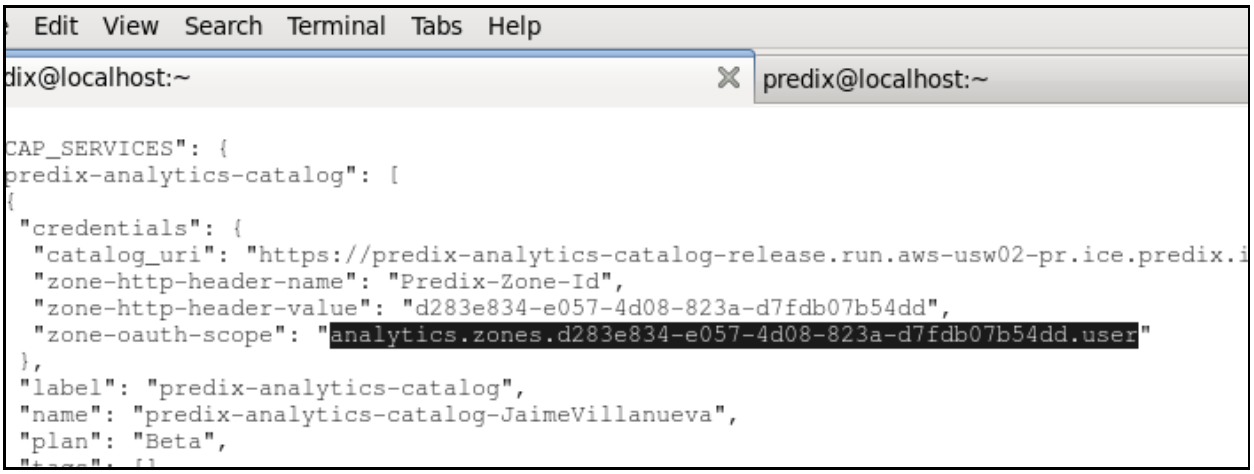
- view and copy the scope for the token
- view and copy the scope for the Analytics Catalog instance
- view and copy the scope for the Analytics Runtime instance
- Combine all three scope statements

Steps

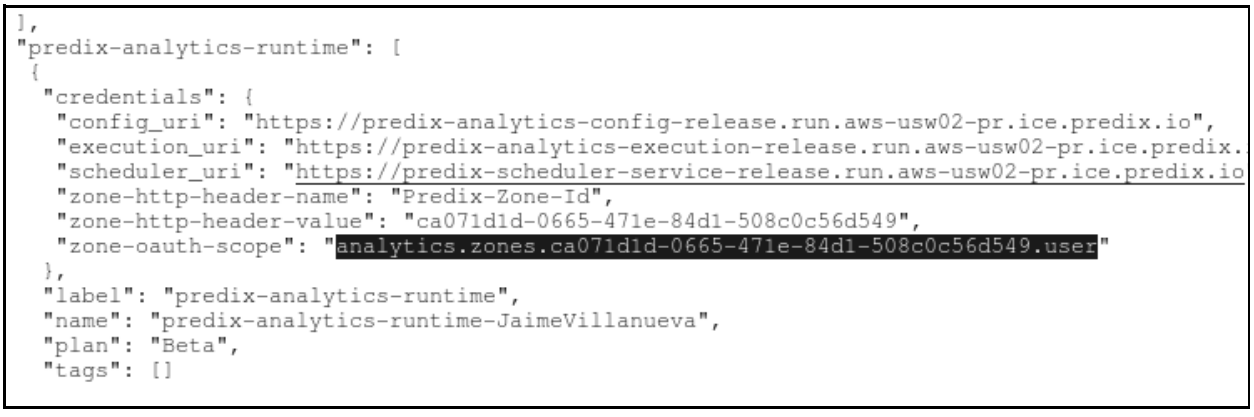
1. Run "Get Token" from Postman, and copy the current scope for the token.

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

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



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



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:

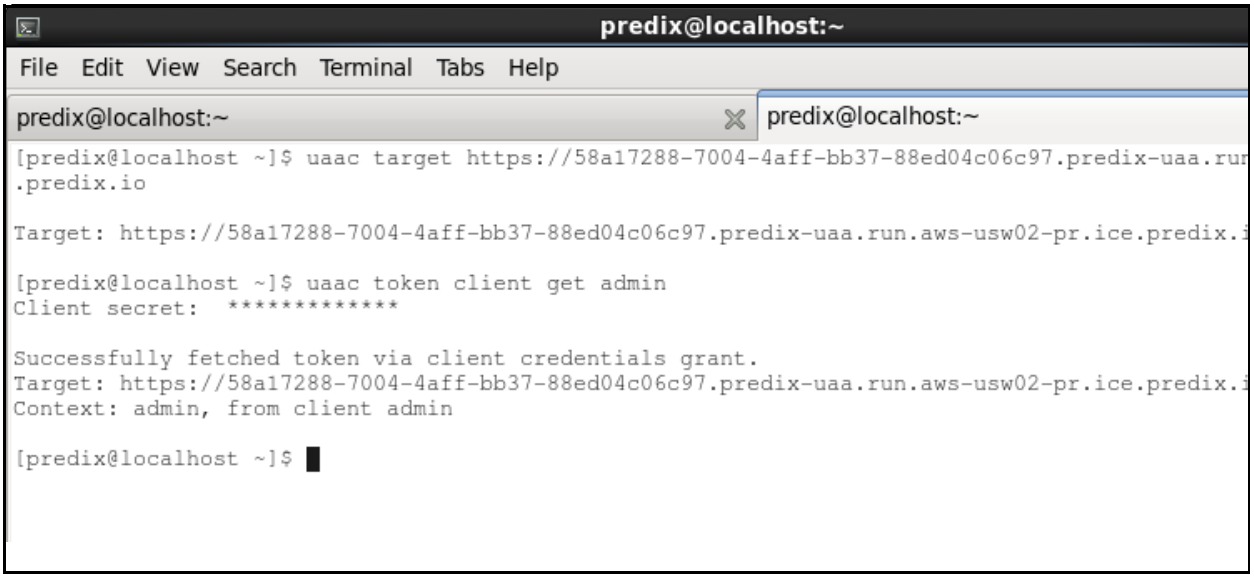
```
uaac target
https://58a17288-7004-4aff-bb37-88ed04c06c97.predix-uaa.run.aws-us
w02-pr.ice.predix.io
```

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**).



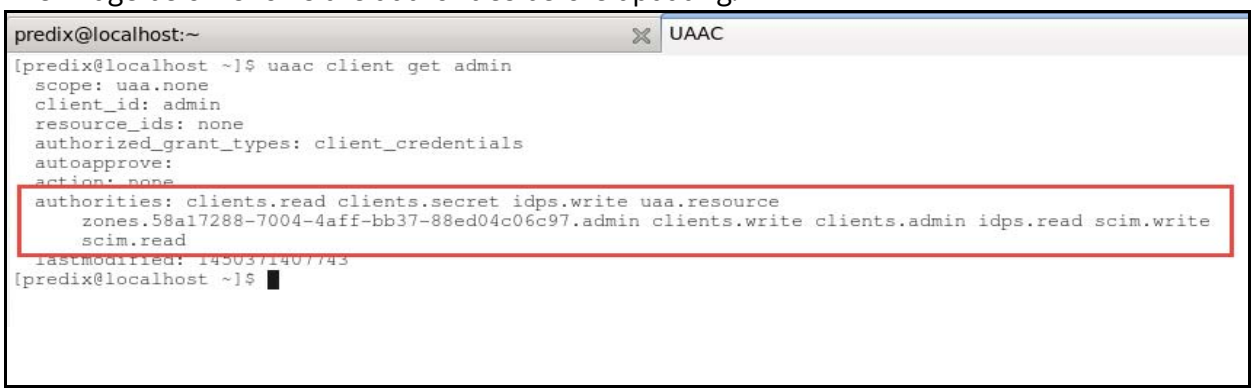
This step validates:

- 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.



- 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.

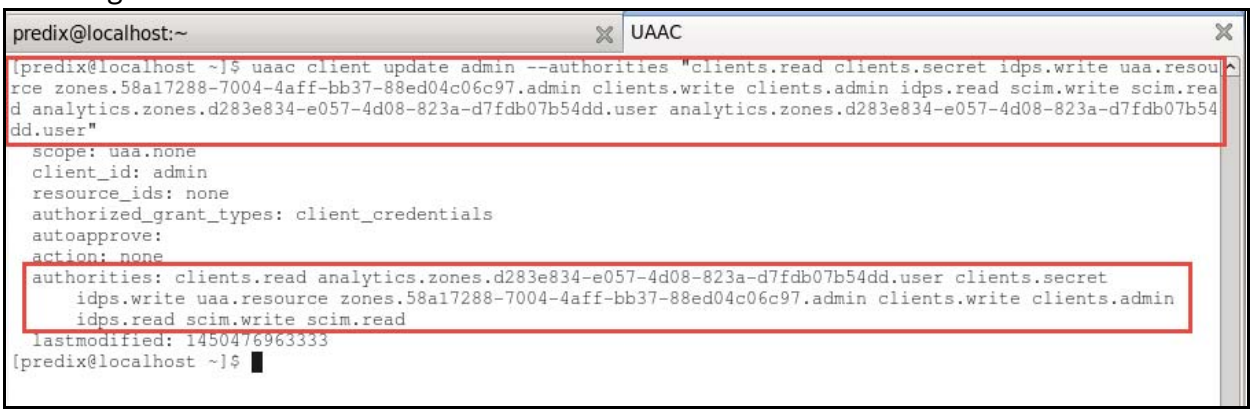
- As in the following example:

```
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.



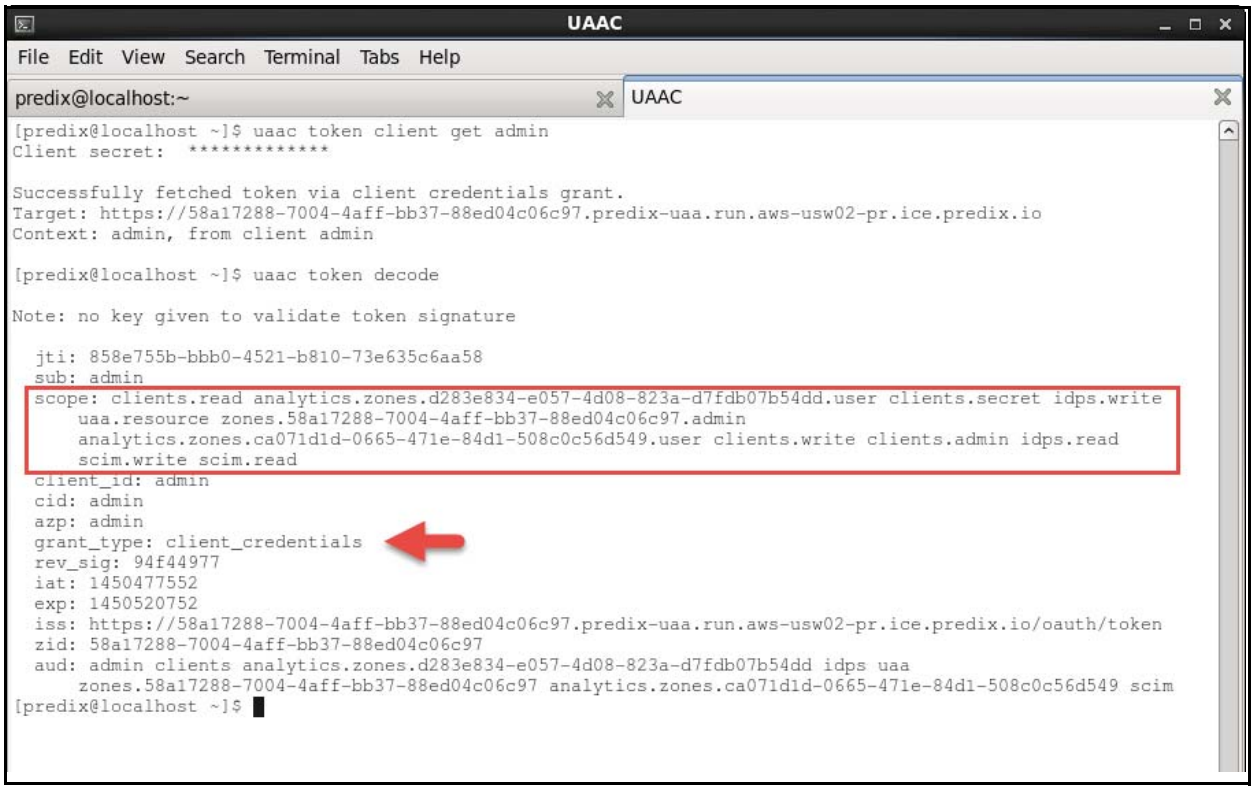
7. Get the token again with client credential grant.

`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.

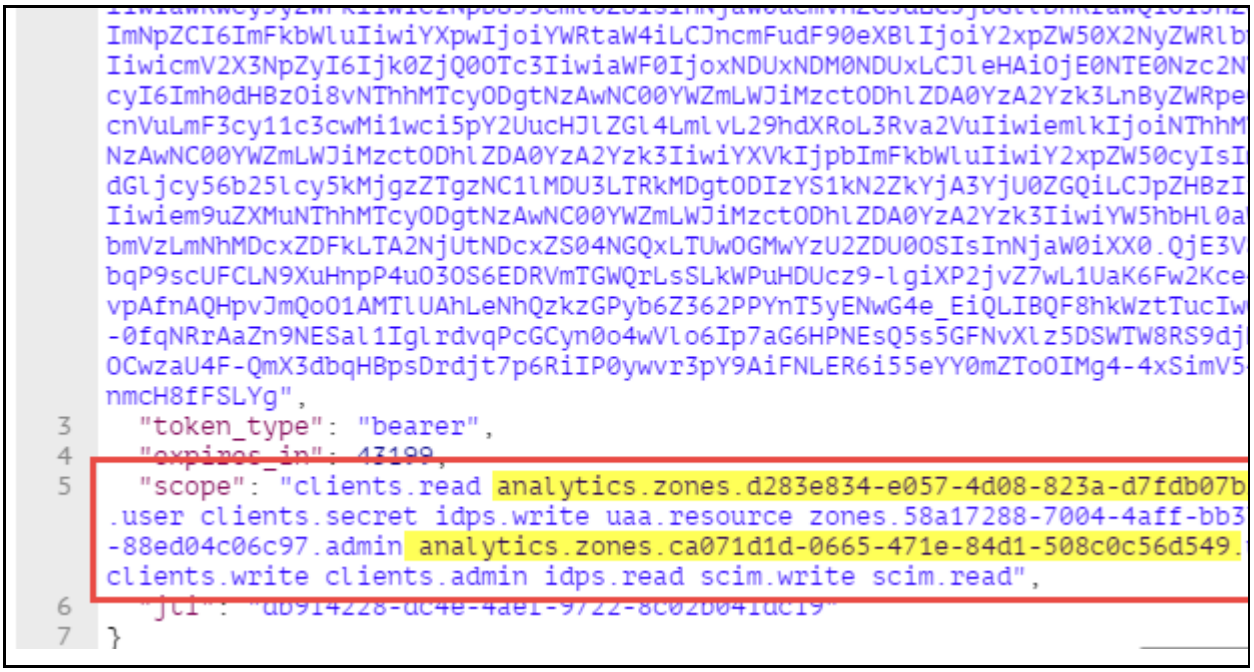
`uaac token decode`



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.



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

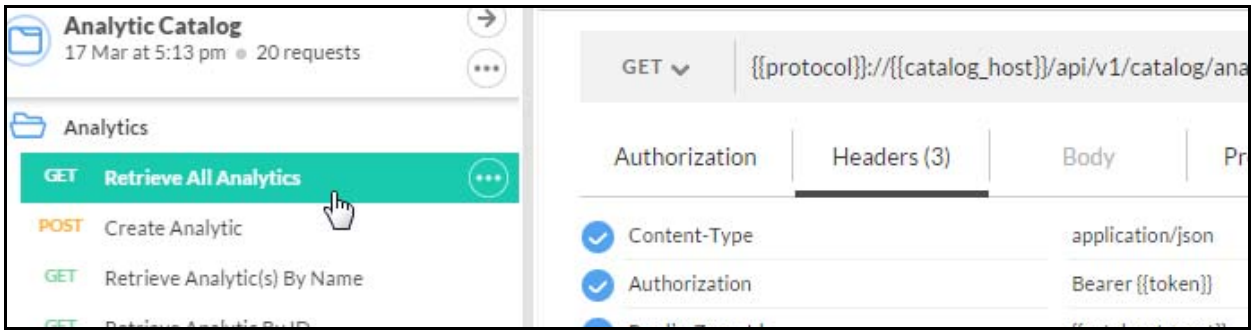
For this lab exercise, you will work with a simple Analytic application demo (provided as part of the lab files), upload it to the catalog, then deploy and test it. The application will receive two numbers and simply add them and provide the result.

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

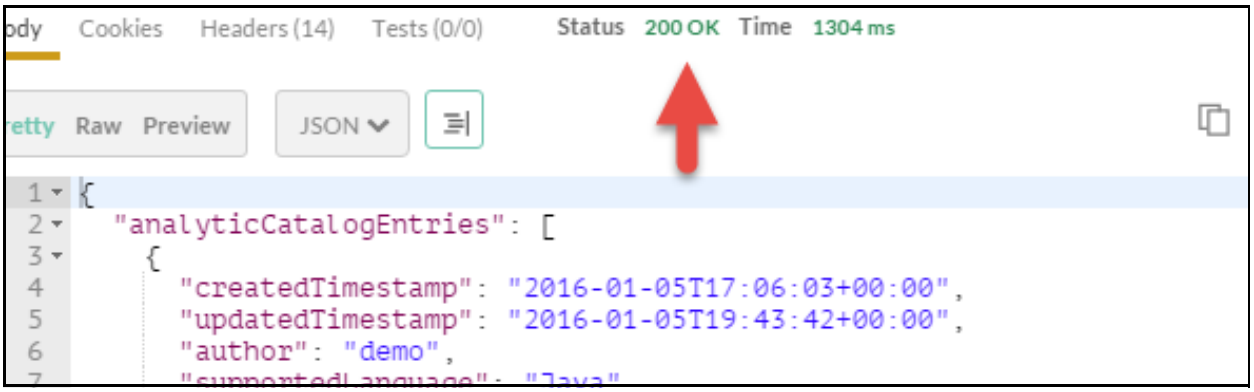
- Create the analytic.
- Upload the analytic to the catalog.
- Deploy and test the analytic in the Cloud Foundry environment

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.



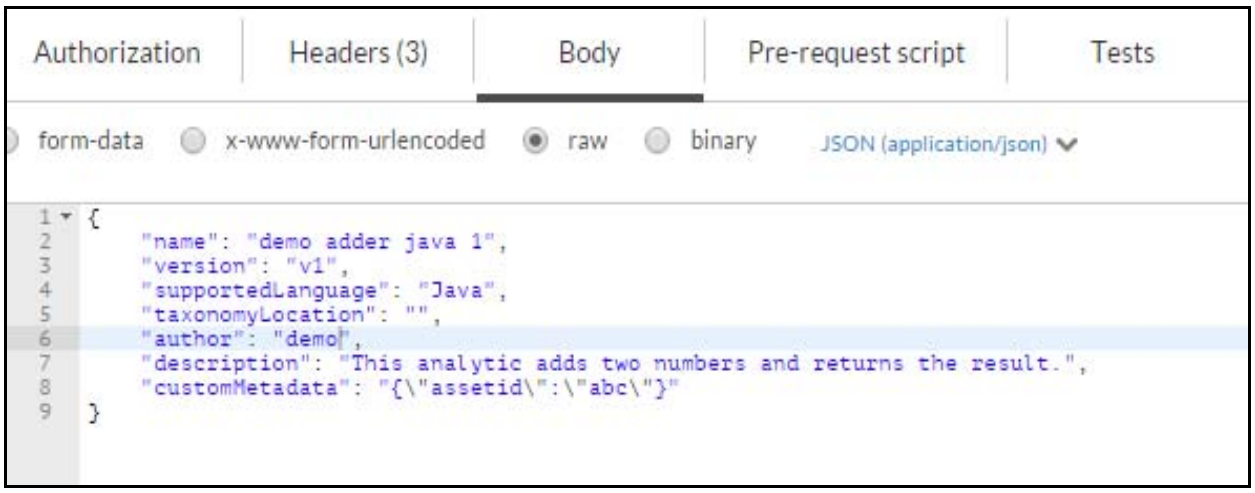
- Click **Send** and you should receive a response status of 200 OK.



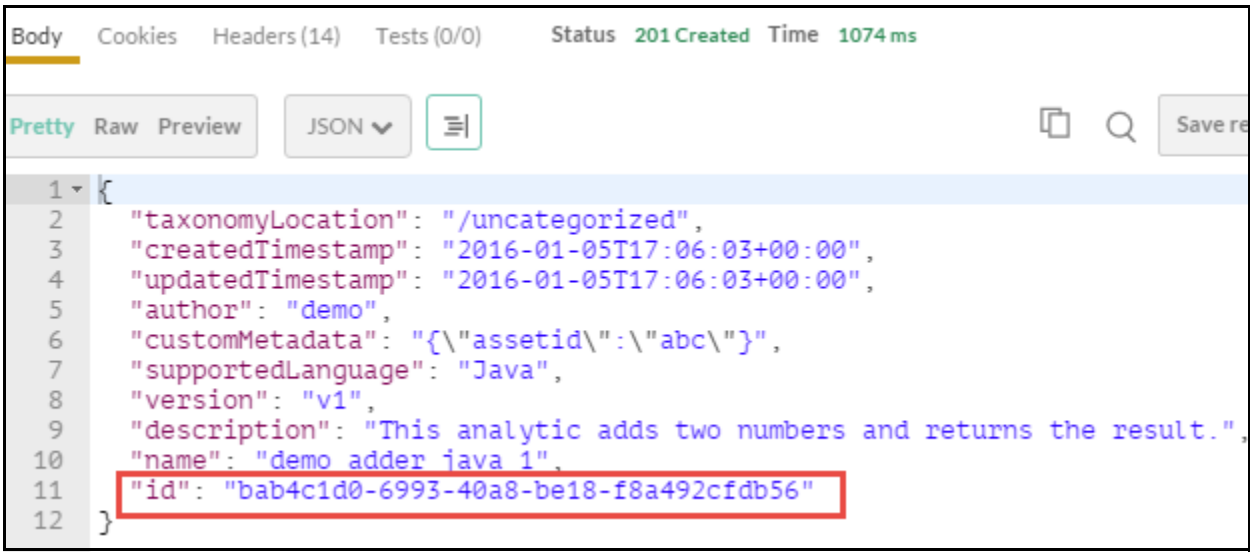
- Review the body of the response, and note that there are no analytics entries as of yet. In the next steps, you will create a catalog entry and upload your sample analytic application.
- Save (and update) this GET request to your Postman collection.

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:



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

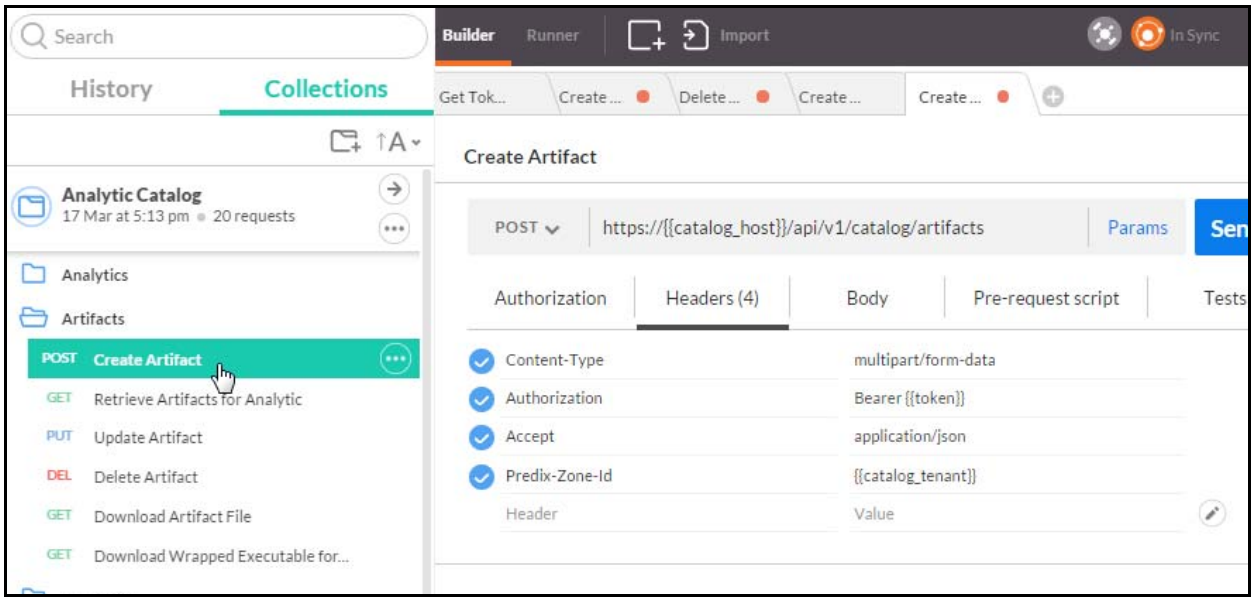


- Note the **id** value created for the catalog entry.
- Optionally, Save (and update) this request to your Postman collection.

3. Attach an executable artifact.

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

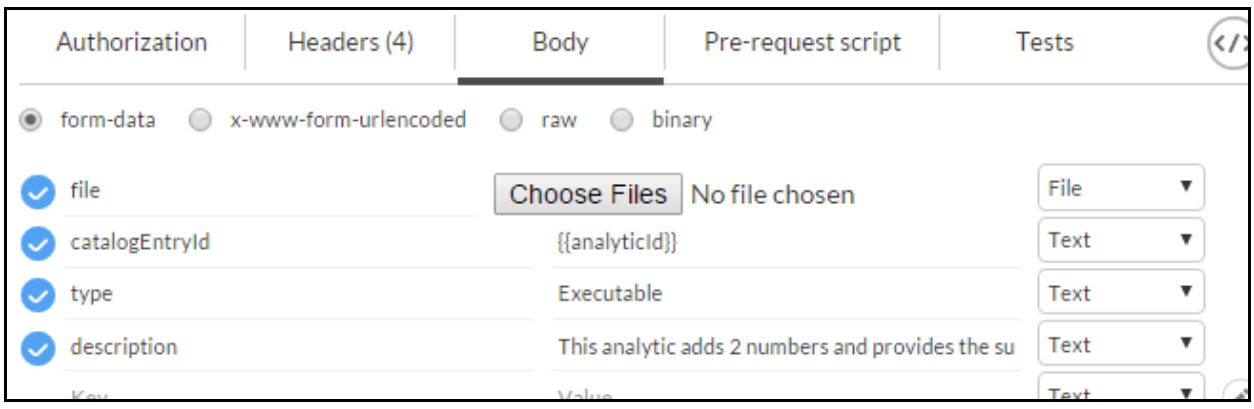
- Add a new Postman tab (window) and select **POST Create Artifact** from your Postman collections.



- 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:

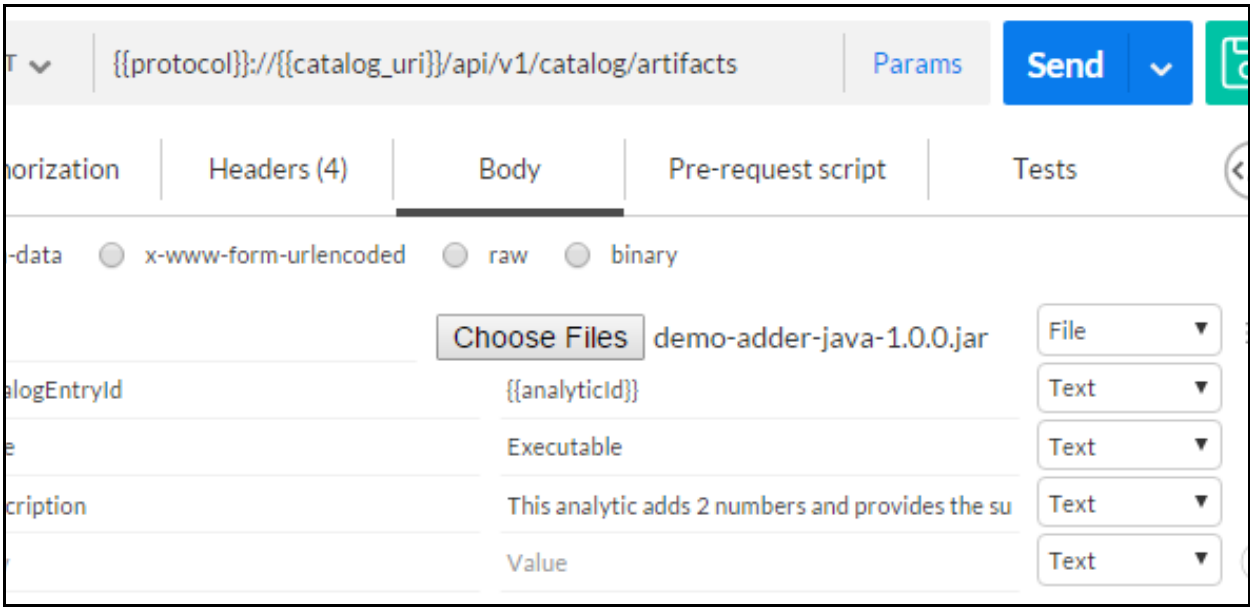
- type: Executable
- description: "This analytic adds 2 numbers and provides the sum."



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

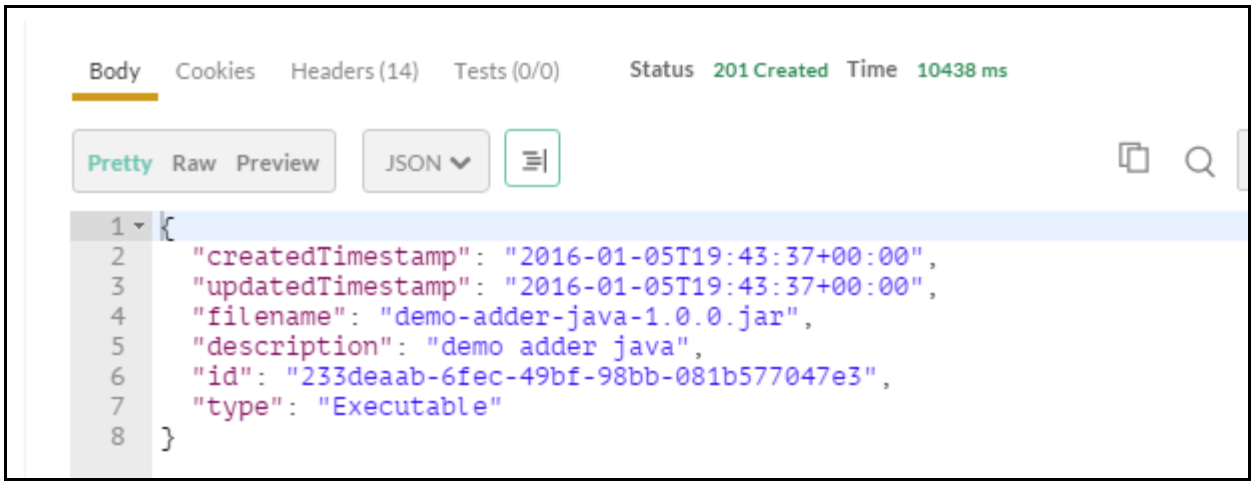
- Click **Choose Files** to upload an artifact and attach it to an analytic catalog entry.

  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.



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

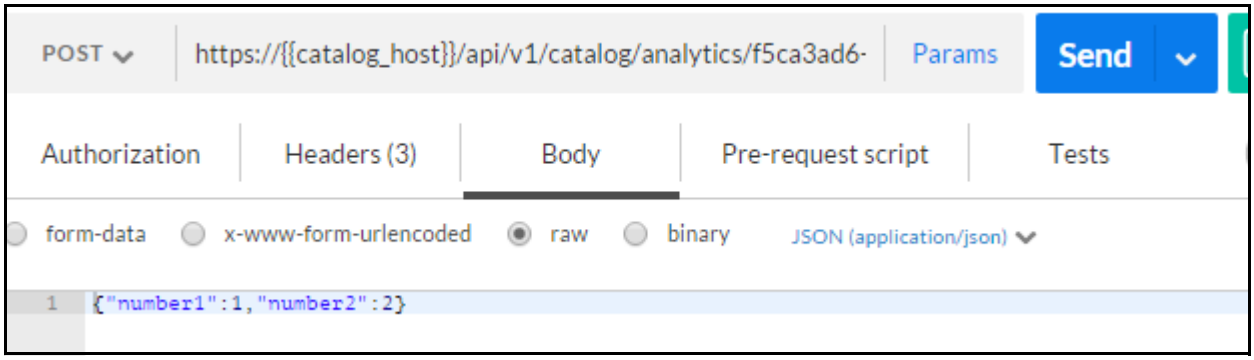
You should receive the following response:



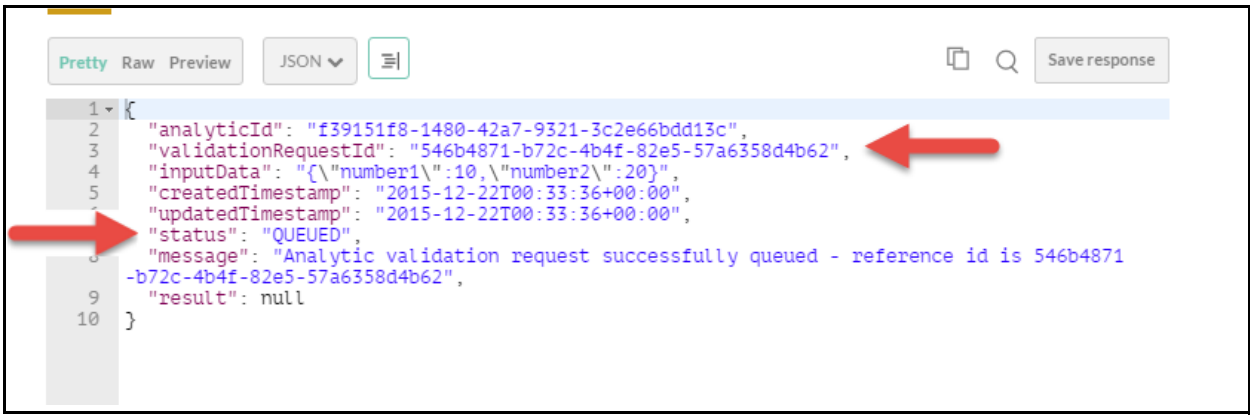
- Copy the "id" of the artifact and paste it into your Postman environment variables.
- Optionally, save this request to your Postman collection, by clicking on the disk icon. This updates your collection request with the current parameters.

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.



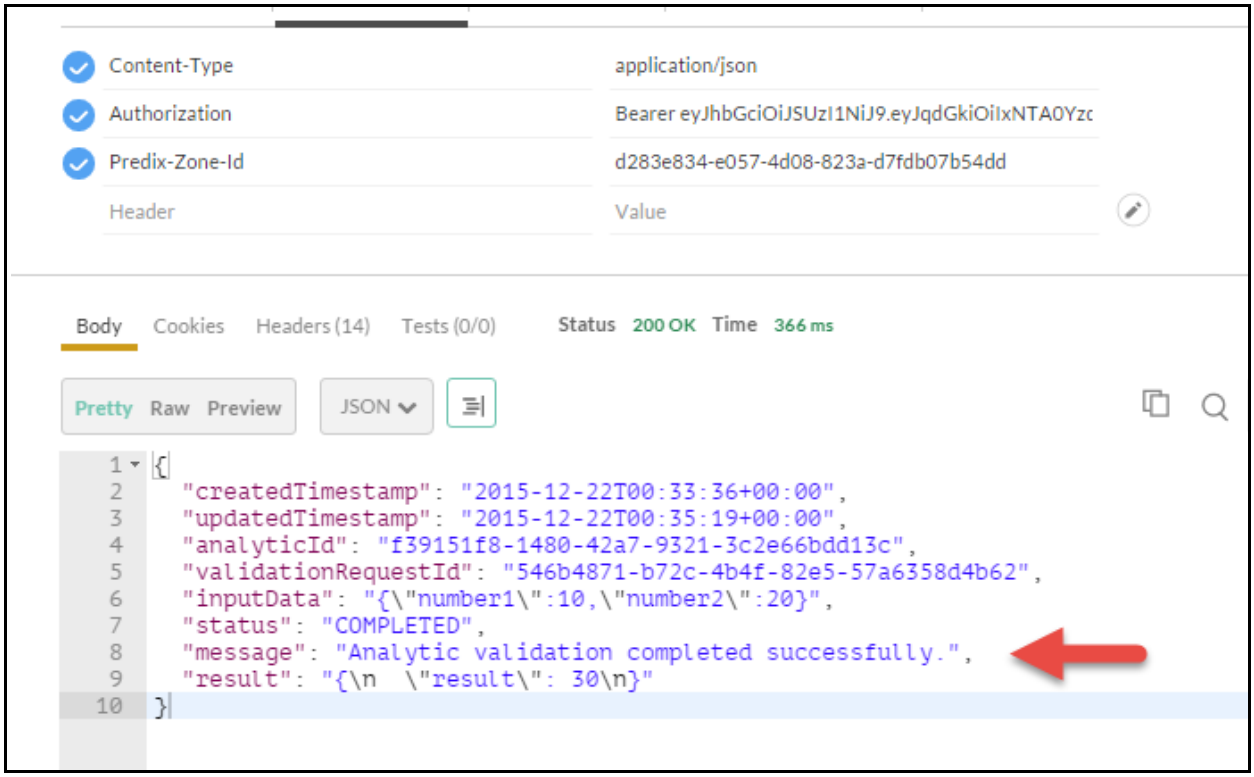
- Click **Send**, and you should receive the following response:



- 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:



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

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.

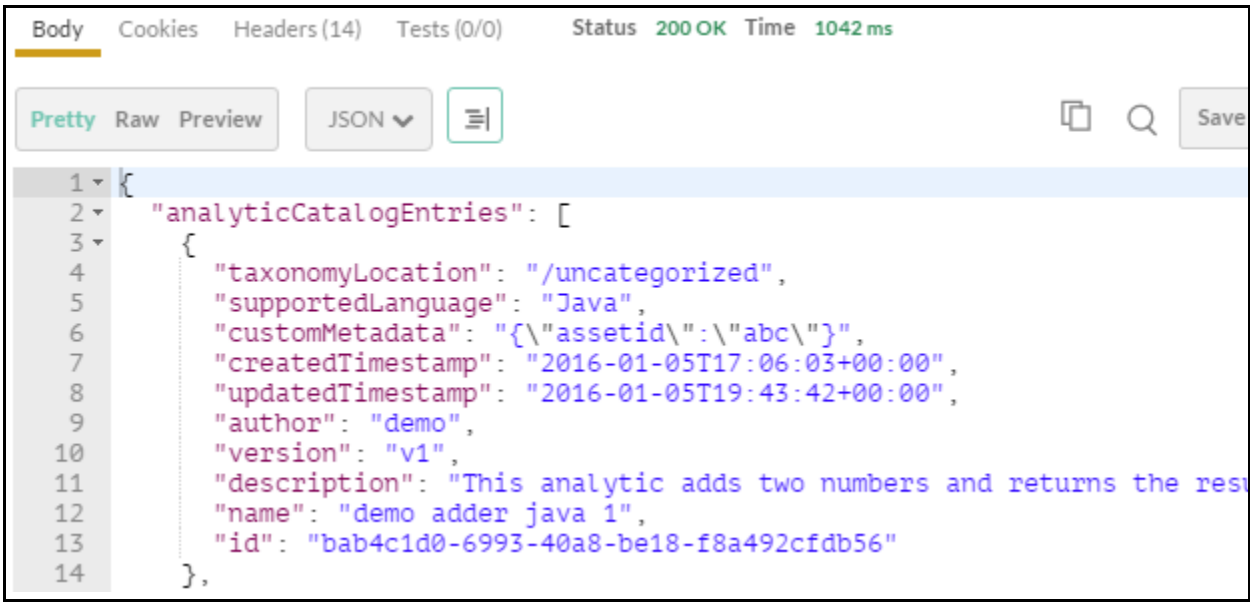
To facilitate learning, your Postman Collections under Orchestration Execution, there are two requests:

- **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.



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.

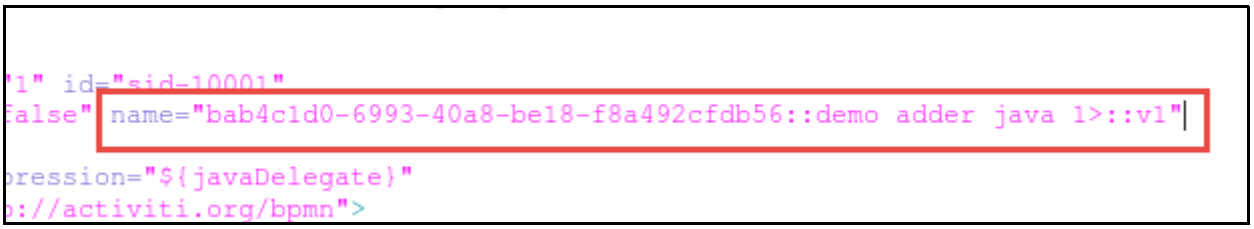
- Go to Predix.io and copy the sample BPMN workflow file at https://www.predix.io/docs#ODGDJxqF (See step 1 of Procedure of this webpage).

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:

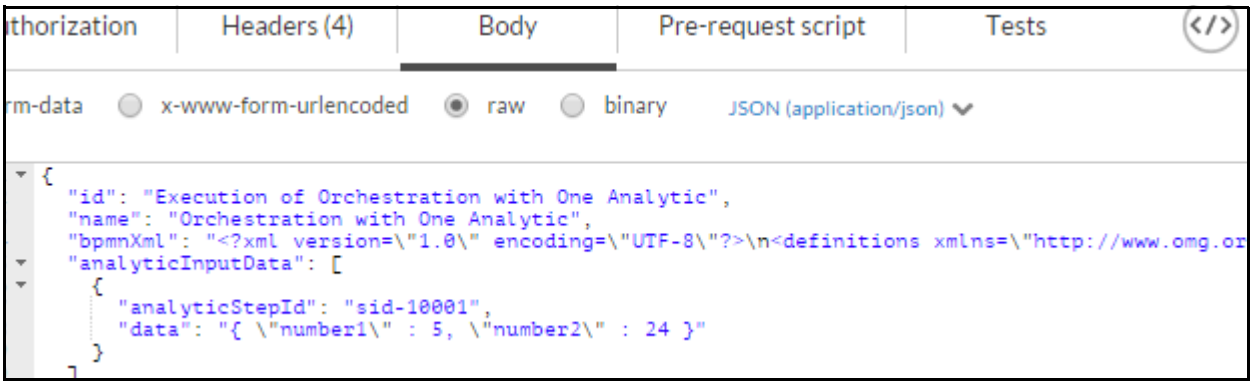
After editing:



- 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.



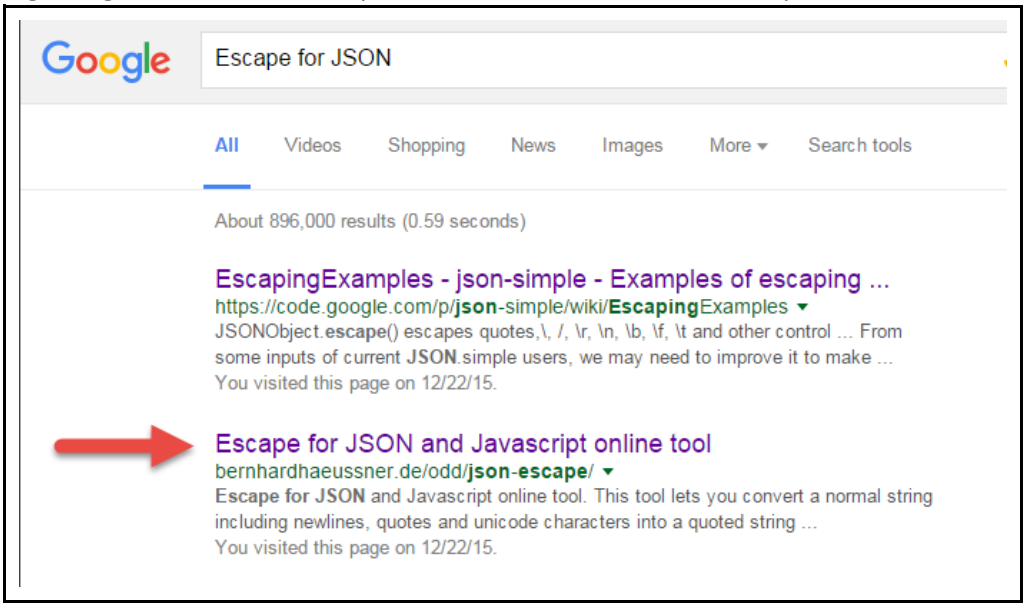
The request has two important fields: bpmnXml and analyticInputData.

- **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.



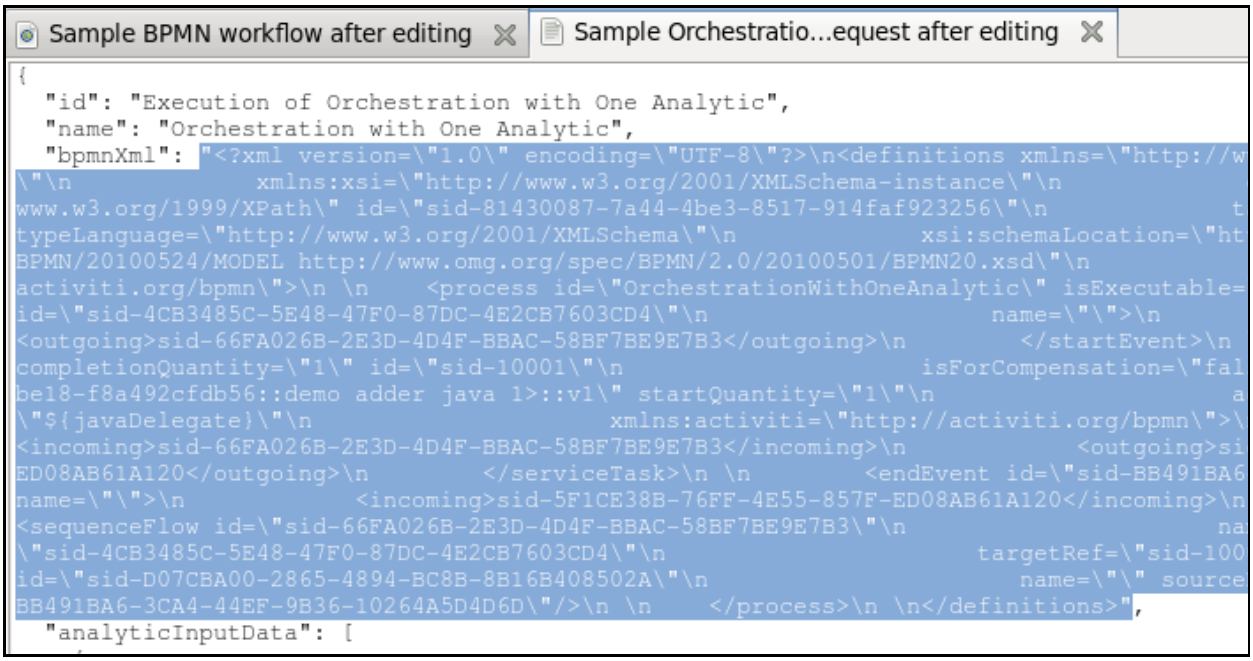
Paste the contents of the edited BPMN Workflow text in to the Escape tool, and click **Escape** to convert accordingly.


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\"
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 \n <process
id=\"OrchestrationWithOneAnalytic\" isExecutable=\"true\">\n \n <startEvent
id=\"sid-4CB3485C-5E48-47F0-87DC-4E2CB7603CD4\"\n name=\"\">\n
<outgoing>sid-66FA026B-2E3D-4D4F-BBAC-58BF7BE9E7B3</outgoing>\n </startEvent>\n \n
<serviceTask completionQuantity=\"1\" id=\"sid-10001\"\n
↓ escape ↓
Escaped & quoted:
"{\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
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\\\"
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 \\n <process
id=\\\"OrchestrationWithOneAnalytic\\\" isExecutable=\\\"true\\\">\\n \\n
<startEvent id=\\\"sid-4CB3485C-5E48-47F0-87DC-4E2CB7603CD4\\\"\\n
name=\\\"\\\">\\n <outgoing>sid-66FA026B-2E3D-4D4F-BBAC-
58BF7BE9E7B3</outgoing>\\n </startEvent>\\n \\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).

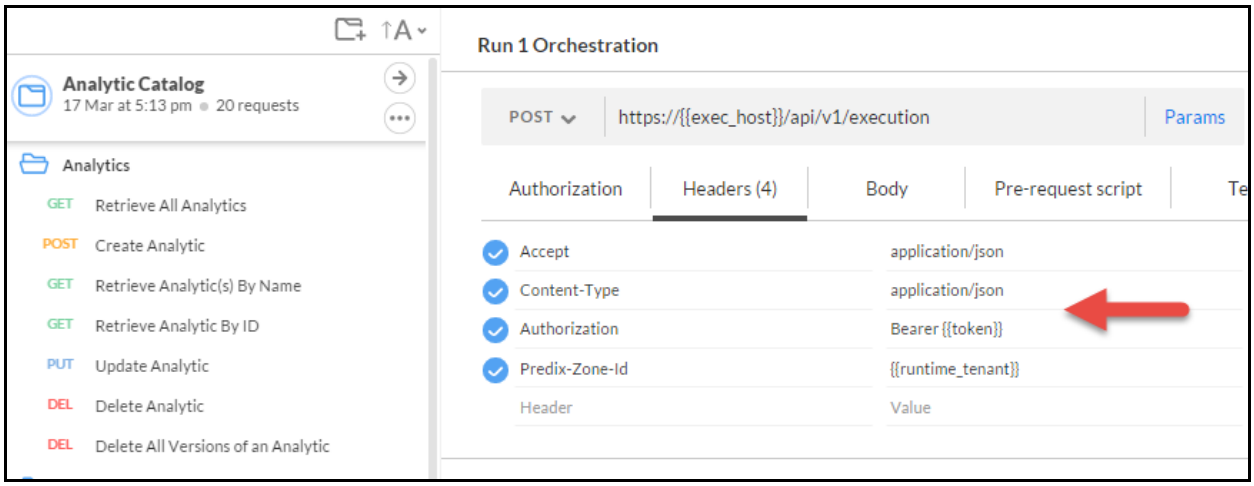
- Copy the "Escaped" text.

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

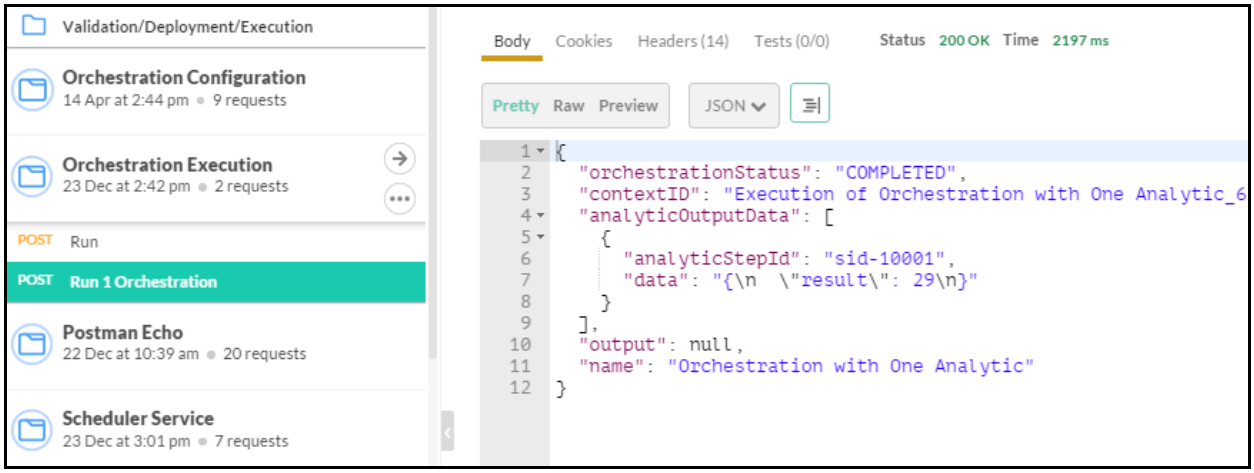


Be sure to include the quotation marks for the content when replacing and ensure that only one set of opening and closing quotation marks enclose the field value as shown in the following image.

- Check your Request parameters, and click Send.



You should receive a response as follows:



- 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.

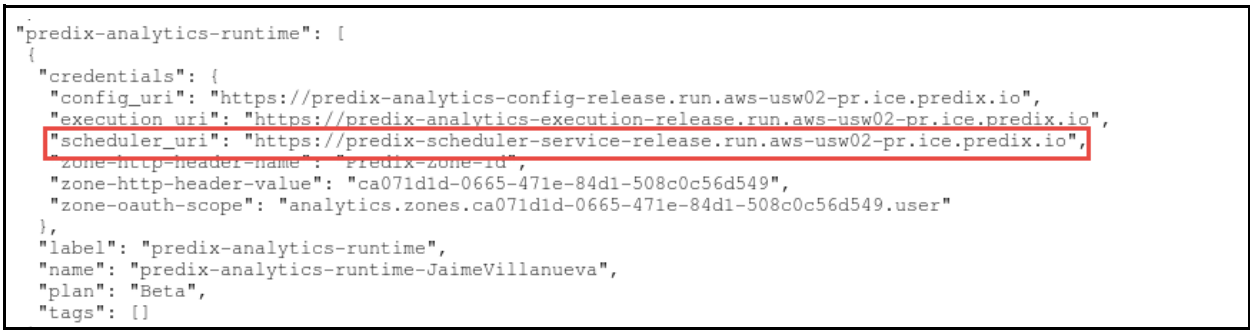
REST APIs are provided for managing a scheduled analytic or orchestration execution (also called a job). Using these APIs you can perform the following actions.

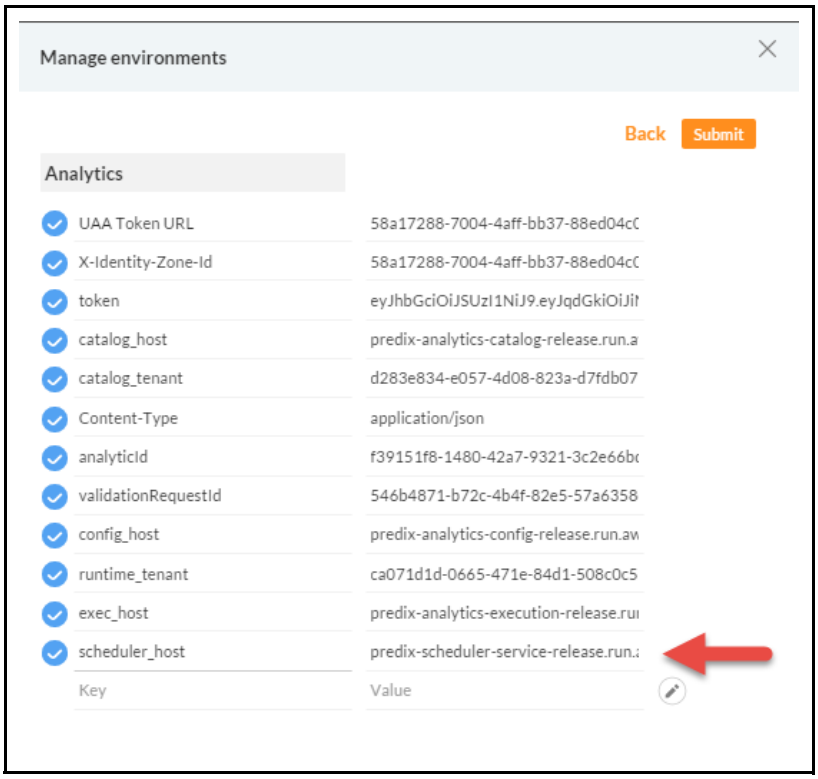
- Create a job definition
- Retrieve job definitions
- Retrieve job history
- Update job definitions
- Delete existing jobs

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.

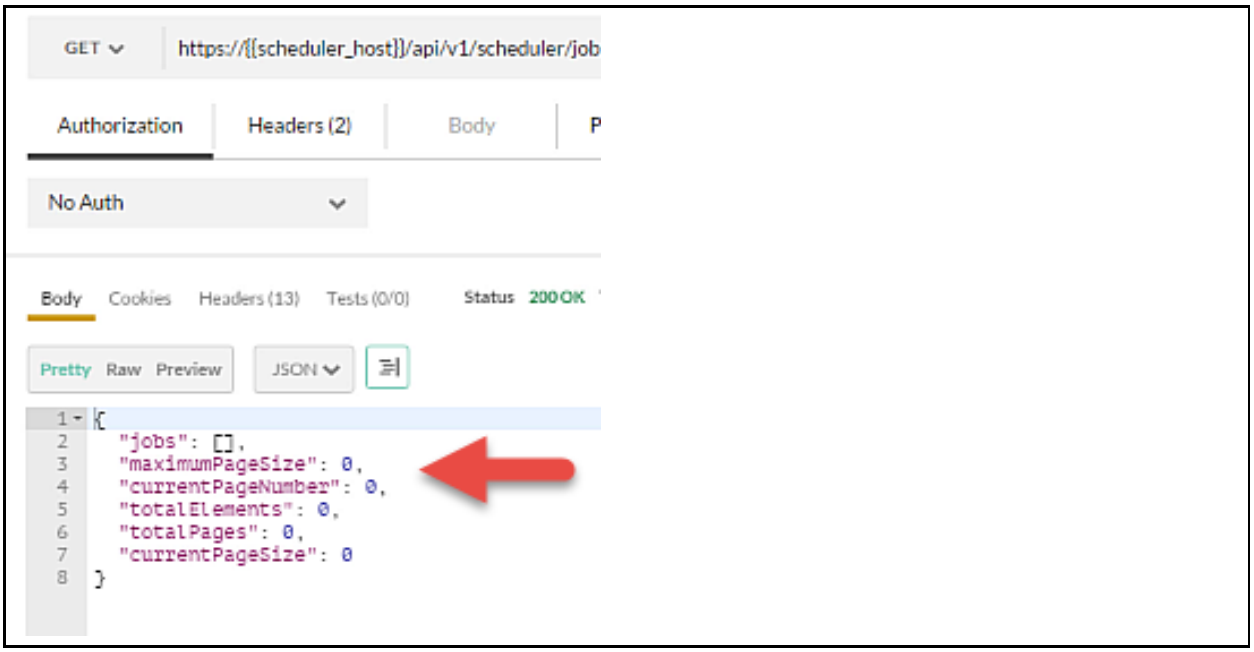
Steps

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





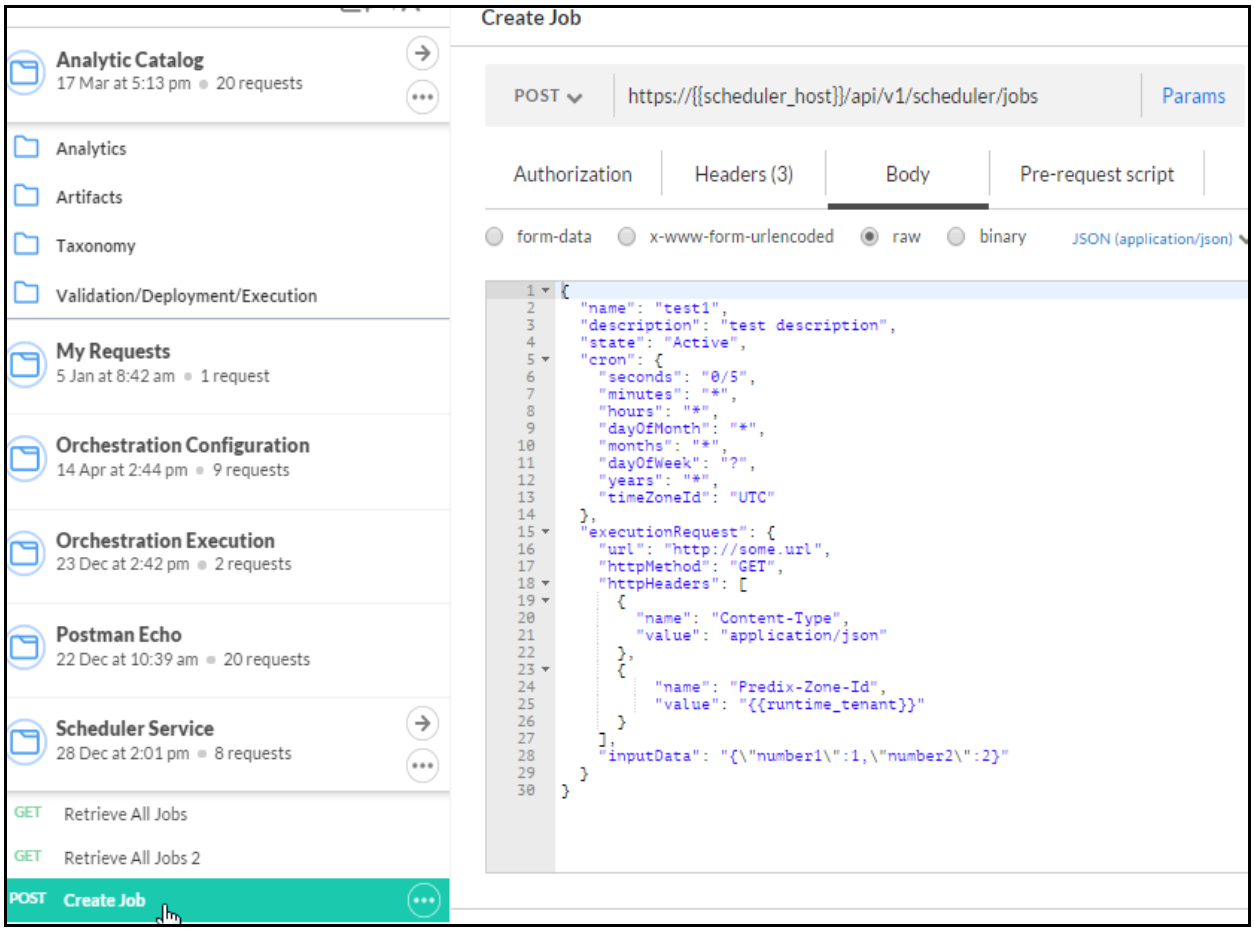
- Click **Submit** to save the variable.
- Click **Send**, and you should receive a response as follows:



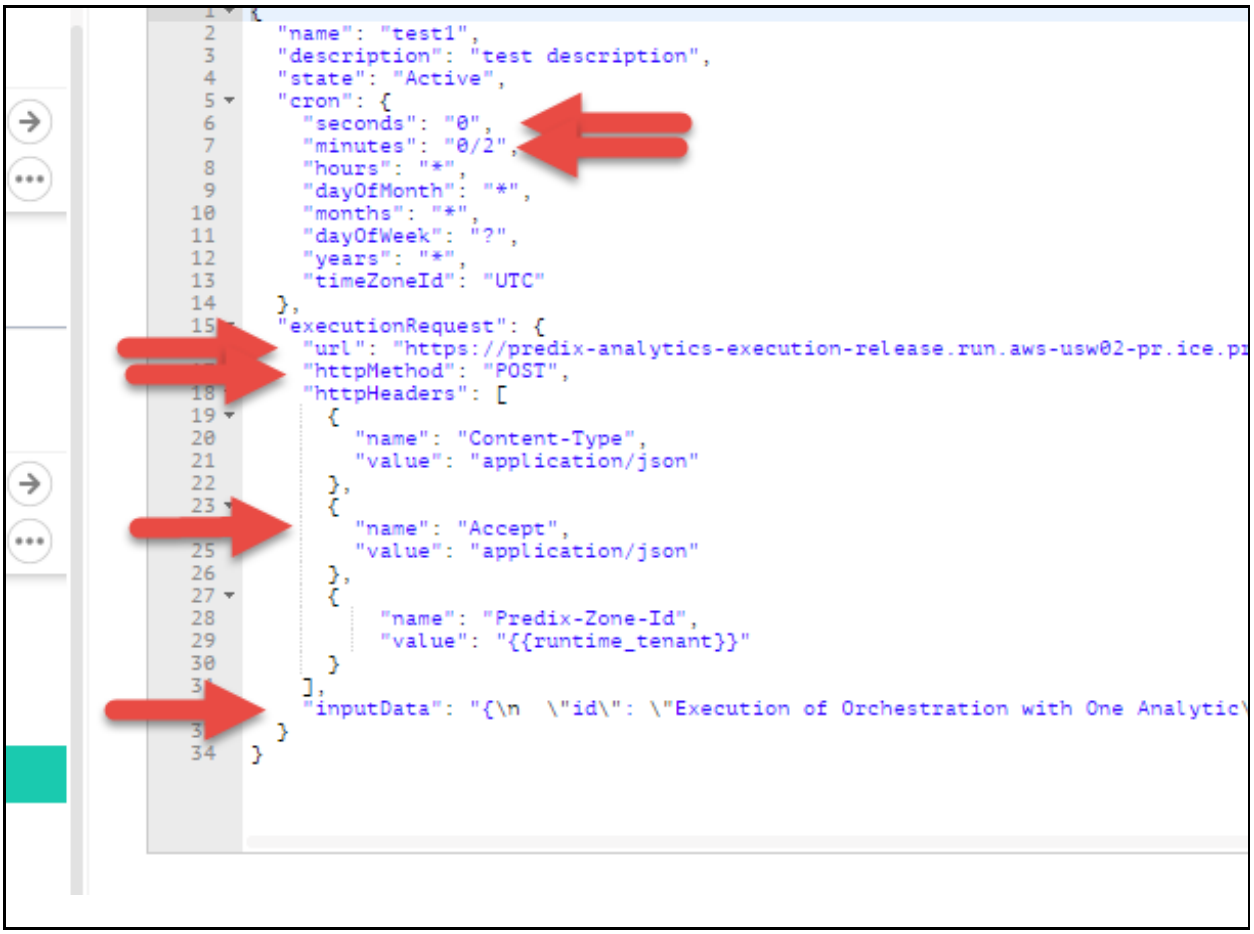
Note that you currently do not have jobs scheduled.

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.



- 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.



- 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.

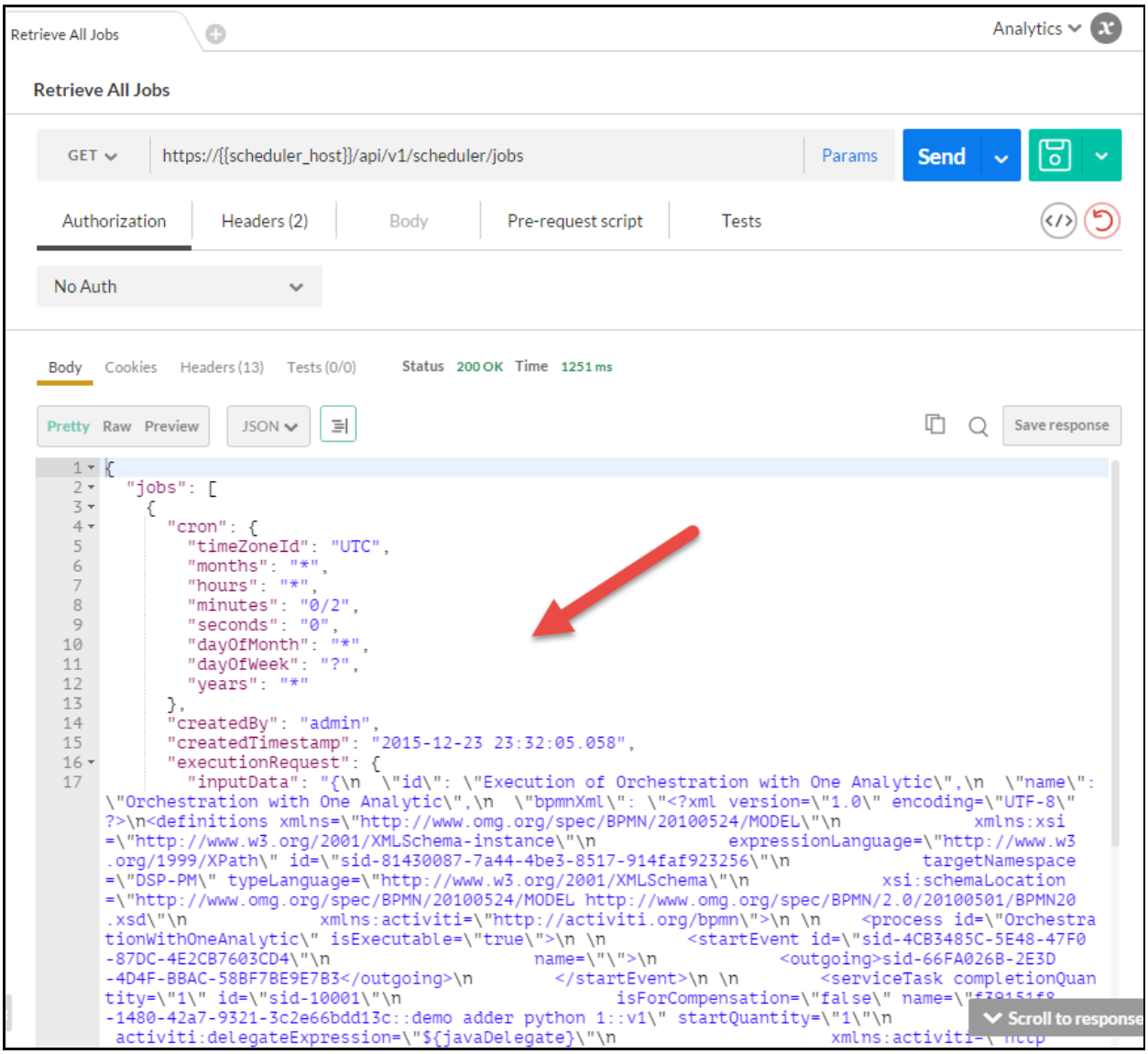
3. Click **Send**, and you should receive the following response:

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

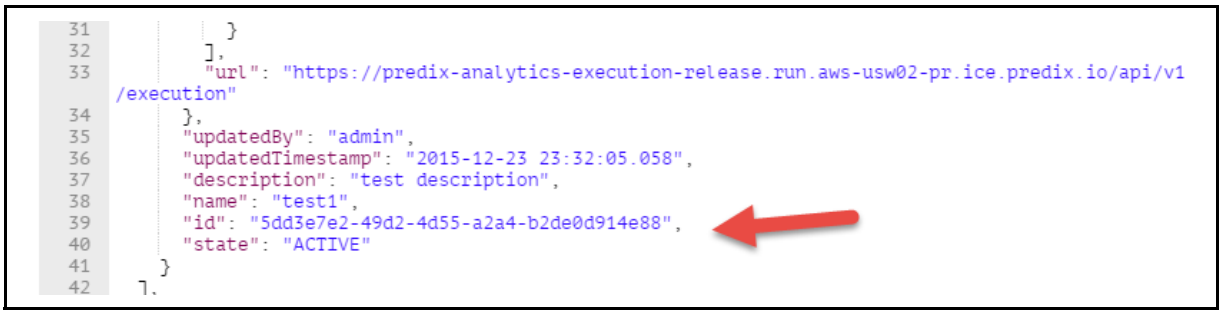
4. Use the GET Request to Retrieve All Jobs, view your running job.

- 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:


Retrieve All Jobs
Analytics
Retrieve All Jobs
GET
https://{{scheduler_host}}/api/v1/scheduler/jobs
Params
Send
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
Save response
Scroll to response

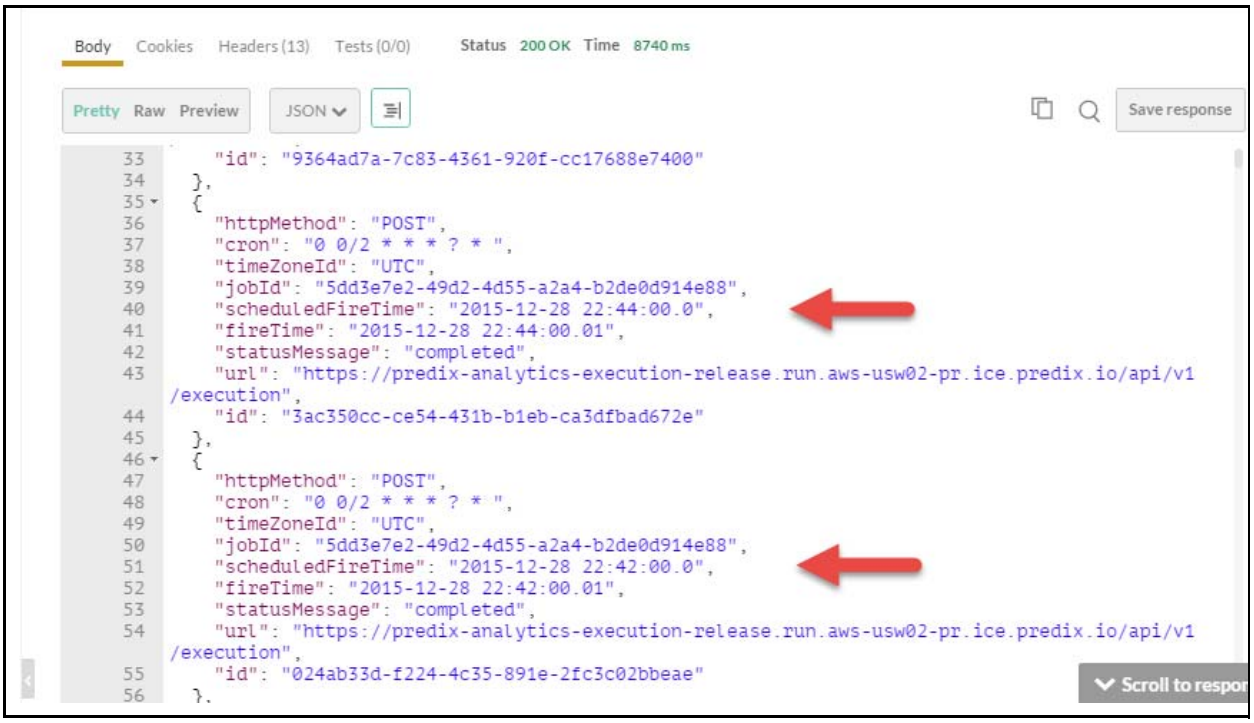
Scrolling down, you'll see the the JobID and status:



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:



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