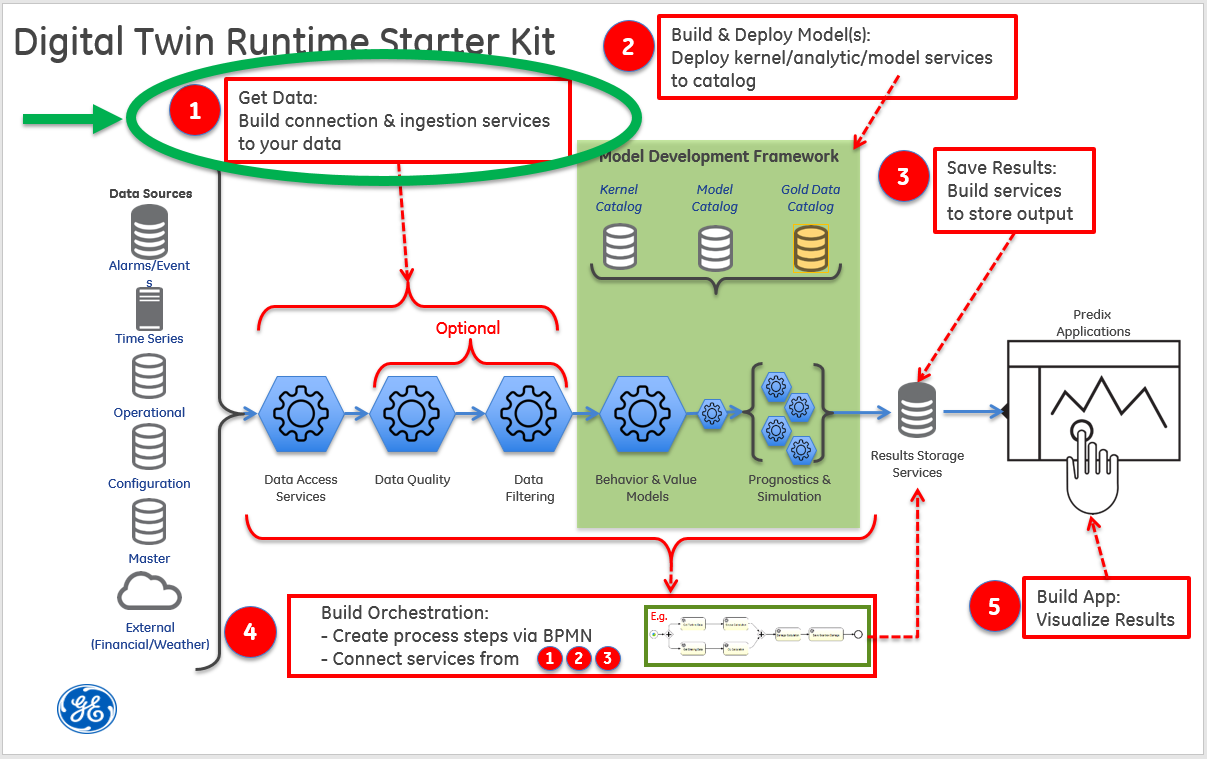
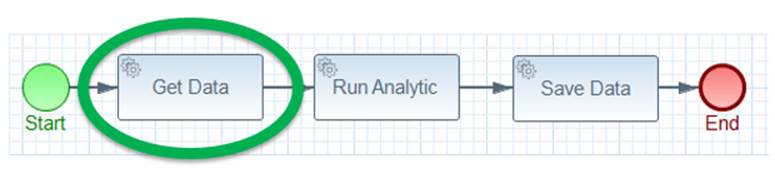
**Step 1: Get Data**



Step 1 illustrates how to use a pre-built set of data retrieval services that are accessible via a REST endpoint (URL).  Later, in Step 4, we'll see the workflow invoke these services to pull the data and then feed it to the analytic model.

The simple workflow:



In this part of the tutorial, you will:

* Create the **tutorial-asset application**
* Create the **tutorial-model-coefficient application**
* Create the **tutorial-data application**
* Create and upload sample data to the Predix Time Series service

What you need to set up

Prior to deploying any tutorial services to the cloud, you'll need a [UAA service](https://www.predix.io/services/service.html?id=1172) instance, a [PostgreSQL service](https://www.predix.io/services/service.html?id=1178) instance, and a [Predix Time Series service](https://www.predix.io/services/service.html?id=1177) instance. If you want to use the provided Postman collection to interact with the REST endpoints, you will need to set the authorization header with the bearer token for the UAA client that has permission to call the REST endpoint since the applications are secured. If you have not set up these services or secured the provided Postman collection, please see the [Getting Started](https://predix-io-dev.grc-apps.svc.ice.ge.com/resources/tutorials/tutorial-details.html?tutorial_id=1915&tag=1913&journey=Digital%20Twin%20Runtime%20Starter%20Kit&resources=1914,1915,1916,1917,1918,1919,1920) section for instructions.

 What you need to do

**Create the tutorial-asset application**

The tutorial-asset application exposes a REST endpoint '/asset' that returns a list of asset names and their corresponding id's. For simplicity, these are stored in a file, assets.json. If the assets were constantly changing, you could imagine storing them in a database and having this service query the database. Refer to [Getting Started](https://predix-io-dev.grc-apps.svc.ice.ge.com/resources/tutorials/tutorial-details.html?tutorial_id=1915&tag=1913&journey=Digital%20Twin%20Runtime%20Starter%20Kit&resources=1914,1915,1916,1917,1918,1919,1920) for download and maven build instructions.

Note that Predix now provides an [Asset Data](https://www.predix.io/services/service.html?id=1171) service that will covers the functionality of this tutorial-asset application. Instantiating this service and replacing this tutorial-asset application is left as an exercise for the reader.

**Deploying your microservice**

1. Configure the appropriate section of the "manifest.yml" file in the tutorial-svcs directory to reflect your environment.

|  |
| --- |
| - name: <YOUR\_OWN\_UNIQUE\_PREFIX>-tutorial-asset  host: <YOUR\_OWN\_ASSET\_SERVICE\_PREFIX>-dt-tutorial-asset  memory: 1G  path: tutorial-asset/target/tutorial-asset-1.1-SNAPSHOT.jar  env:  security\_oauth2\_resource\_tokenInfoUri: https://<YOUR\_UAA\_INSTANCE\_HERE>.predix-uaa.run.aws-usw02-pr.ice.predix.io/check\_token  security\_oauth2\_client\_clientId: <YOUR\_CLIENT\_ID> |

Notes:

* The application **name** must be unique across your CloudFoundry organization.
* The **host** must be unique across Predix as this becomes the URL for the microservice. If you receive an error message while pushing your service to the cloud in the next step, define a new **host**.

Substitute your UAA instance name for <YOUR\_UAA\_INSTANCE\_HERE>.Substitute your client id for <YOUR\_CLIENT\_ID>.

1. Push it to Cloud Foundry.

|  |
| --- |
| C:\steam-turbine-tutorial\tutorial-svcs> cf push <YOUR\_OWN\_UNIQUE\_PREFIX>-tutorial-asset --no-start |

1. You'll need to set an environment variable for the security\_oauth2\_client\_clientSecret. You could set this in the manifest.yml file instead, but we recommend using an environment variable as it is more secure than storing passwords in a file, which may result in them being accidentally committed to your source code repository.

|  |
| --- |
| C:\steam-turbine-tutorial\tutorial-svcs> cf set-env <YOUR\_OWN\_UNIQUE\_PREFIX>-tutorial-asset security\_oauth2\_client\_clientSecret <your Client Id's secret> |

1. Start your application.

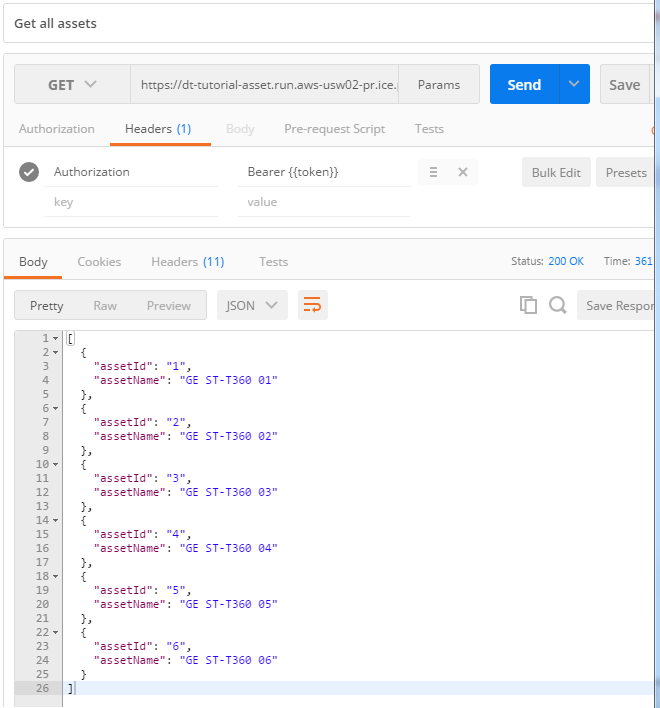
|  |
| --- |
| C:\steam-turbine-tutorial\tutorial-svcs\tutorial-asset> cf start <YOUR\_OWN\_UNIQUE\_PREFIX>-tutorial-asset |

1. Use "cf apps" to discover the URL to your service. Prepend the <https://> protocol then append your API path to get the full URL to your data service.

|  |
| --- |
| C:\steam-turbine-tutorial\tutorial-svcs\tutorial-asset> cf apps Getting apps in org DigitalTwin / space dev as 200000000@mail.ad.ge.com... OK name                                requested state   instances   memory   disk   urls ... tutorial-asset                      started               1/1             1G           1G     **dt-tutorial-asset.run.aws-usw02-pr.ice.predix.io** ... |

1. Get a bearer token and test your service with Postman.

* Try the “Get all assets” GET request.



**Create the tutorial-model-coefficient application**

This application gets the coefficients for a specific asset model. This is an example of how tuned parameters for an asset model can be retrieved and combined with time series data before calling the analytic. Refer to[Getting Started](https://predix-io-dev.grc-apps.svc.ice.ge.com/resources/tutorials/tutorial-details.html?tutorial_id=1915&tag=1913&journey=Digital%20Twin%20Runtime%20Starter%20Kit&resources=1914,1915,1916,1917,1918,1919,1920)for download and maven build instructions.

**Deploying your microservice**

1. Configure the appropriate section of the "manifest.yml" file in the tutorial-svcs directory to reflect your environment.

|  |
| --- |
| - name: <YOUR\_OWN\_UNIQUE\_PREFIX>-tutorial-model-coefficient  host: <YOUR\_OWN\_MODEL\_COEFFICIENT\_SERVICE\_PREFIX>-dt-tutorial-model-coefficient  memory: 1G  services:  - <YOUR\_POSTGRES\_SERVICE\_FOR\_COEFFICIENT\_DATA>  path: tutorial-model-coefficient/target/tutorial-model-coefficient-1.1-SNAPSHOT.jar  env:  security\_oauth2\_resource\_tokenInfoUri: https://<YOUR\_UAA\_INSTANCE\_HERE>.predix-uaa.run.aws-usw02-pr.ice.predix.io/check\_token  security\_oauth2\_client\_clientId: <YOUR\_CLIENT\_ID> |

Notes:

* The application **name** must be unique across the CloudFoundry organization.
* In the **services** list, substitute your postgres service instance name for <YOUR\_POSTGRES\_SERVICE\_FOR\_COEFFICIENT\_DATA>. The default value of the $postgres\_instance\_name variable in the create-dt-starter-kit.pl Perl script was “my-postgres”.
* Substitute your UAA instance name for <YOUR\_UAA\_INSTANCE\_HERE>.

Substitute your client id for <YOUR\_CLIENT\_ID>.

1. Push it to Cloud Foundry

|  |
| --- |
| C:\steam-turbine-tutorial\tutorial-svcs> cf push <YOUR\_OWN\_UNIQUE\_PREFIX>-tutorial-model-coefficient --no-start |

1. Set an environment variable for the security\_oauth2\_client\_clientSecret. You could set this in the manifest.yml file instead, but we recommend using an environment variable as it is more secure than storing passwords in a file, which may result in them being accidentally committed to your source code repository.

|  |
| --- |
| C:\steam-turbine-tutorial\tutorial-svcs> cf set-env <YOUR\_OWN\_UNIQUE\_PREFIX>-tutorial-model-coefficient security\_oauth2\_client\_clientSecret <your Client Id's secret> |

1. Start your application.

|  |
| --- |
| C:\steam-turbine-tutorial\tutorial-svcs> cf start <YOUR\_OWN\_UNIQUE\_PREFIX>-tutorial-model-coefficient |

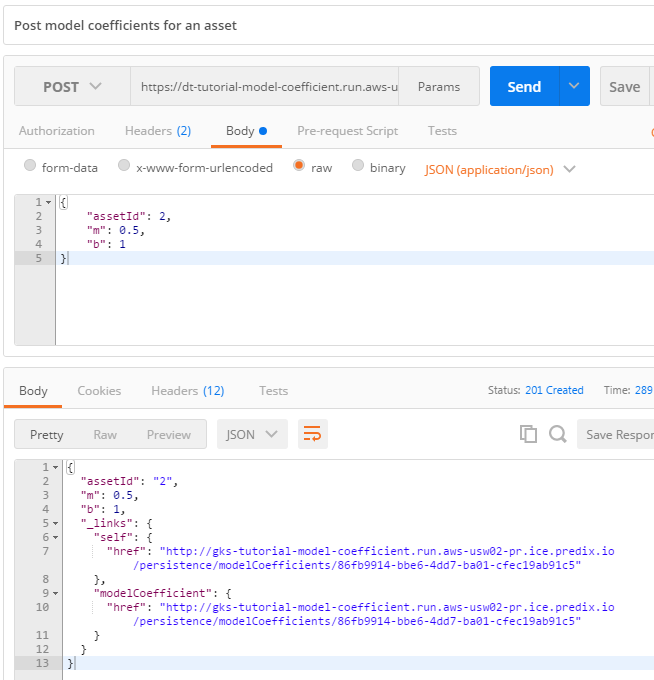
1. Run the "cf apps" command to discover the URL to your service. Prepend the <https://> protocol then append your API path to get the full URL to your data service

|  |
| --- |
| C:\steam-turbine-tutorial\tutorial-svcs\tutorial-tutorial-model-coefficient> cf apps Getting apps in org DigitalTwin / space dev as 200000000@mail.ad.ge.com... OK name                                      requested state   instances   memory   disk   urls ... tutorial-model-coefficient          started                1/1             1G           1G     **dt-tutorial-model-coefficient.run.aws-usw02-pr.ice.predix.io** ... |

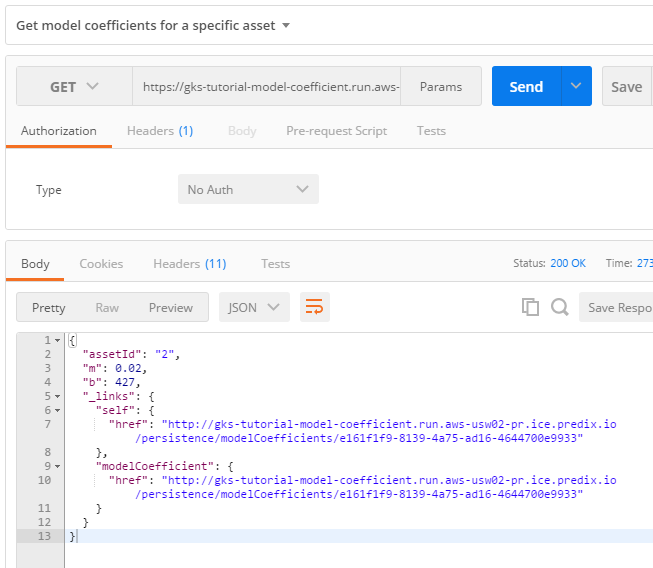
1. Get a bearer token and test your service with Postman

* First, run the “Post model coefficients for an asset” request (otherwise you will get a 404 - Not found response if you try to get the coefficients). Your asset id that you post must match one of the assets defined in the **assets.json** file in the tutorial-asset service, you may also create your own assets if you would like. The model we provide is a linear model that fits a line to data in the form y = mx + b, where m is the slope of the line, x is the data point, b is the y-intercept, and y is the predicted value. The body should be JSON that looks like (for the purpose of consistency in this tutorial, please use an “assetId” of 2 in the JSON payload):

|  |
| --- |
| JSON body for POST model coefficients |
| {     "assetId": 2,     "m": 0.5,     "b": 1 } |



* Next, “Get the model coefficients for a specific asset” request.



**Create the tutorial-data application**

This application will retrieve the time series data for a given date range and asset id. It will also retrieve the model coefficients associated with the asset id from the tutorial-model-coefficient service and bundle them in the returned JSON. The returned JSON is in a form that the analytic can directly consume. Refer to[Getting Started](https://predix-io-dev.grc-apps.svc.ice.ge.com/resources/tutorials/tutorial-details.html?tutorial_id=1915&tag=1913&journey=Digital%20Twin%20Runtime%20Starter%20Kit&resources=1914,1915,1916,1917,1918,1919,1920)for download and maven build instructions.

**Deploying your microservice**

* 1. Configure the appropriate section of the "manifest.yml" file in the tutorial-svcs directory to reflect your environment. Note that there are several “keys” in the manifest.yml file that start with the string “com\_ge\_dt\_tsc” – these keys are for configuring the **D**igital **T**win **T**ime**S**eries **C**lient.

|  |
| --- |
| - name: <YOUR\_OWN\_UNIQUE\_PREFIX>-tutorial-data  host: <YOUR\_OWN\_DATA\_SERVICE\_PREFIX>-dt-tutorial-data  memory: 1G  services:  - <YOUR\_PREDIX\_TIMESERIES\_SERVICE>  path: tutorial-data/target/tutorial-data-1.1-SNAPSHOT.jar  env:  security\_oauth2\_resource\_tokenInfoUri: https://<YOUR\_UAA\_INSTANCE\_HERE>.predix-uaa.run.aws-usw02-pr.ice.predix.io/check\_token  security\_oauth2\_client\_clientId: <YOUR\_CLIENT\_ID>  com\_ge\_dt\_tsc\_tokenEndpoint: https://<YOUR\_UAA\_INSTANCE\_HERE>.predix-uaa.run.aws-usw02-pr.ice.predix.io/oauth/token  com\_ge\_dt\_tsc\_clientId: <YOUR\_CLIENT\_ID>  com\_ge\_dt\_tsc\_queryEndpoint: <https://time-series-store-predix.run.aws-usw02-pr.ice.predix.io/v1/datapoints>  com\_ge\_dt\_tsc\_ingestionEndpoint: wss://gateway-predix-data-services.run.aws-usw02-pr.ice.predix.io/v1/stream/messages  com\_ge\_dt\_tsc\_zoneId: <YOUR\_PREDIX\_TIMESERIES\_SERVICE\_PREDIX\_ZONE\_ID>  com\_ge\_digitaltwin\_tutorial\_data\_coefficient\_coefficientService: https://<YOUR\_OWN\_MODEL\_COEFFICIENT\_SERVICE\_PREFIX>-dt-tutorial-model-coefficient.run.aws-usw02-pr.ice.predix.io/persistence/modelCoefficients/search/findByAssetId |

* The application **name** must be unique across the CloudFoundry organization.
* The **host** must be unique across Predix as this becomes the URL for the microservice. If you receive an error message while pushing your service to the cloud in the next step, define a new **host**.
* In the **services** list, substitute your time series service instance name for <YOUR\_PREDIX\_TIMESERIES\_SERVICE>.
* Substitute your UAA instance name for <YOUR\_UAA\_INSTANCE\_HERE> in both “security\_oauth2\_resource\_tokenInfoUri” and “com\_ge\_dt\_tsc\_tokenEndpoint”.
* Substitute your client id for <YOUR\_CLIENT\_ID> in both “security\_oauth2\_client\_clientId” and “com\_ge\_dt\_tsc\_clientId”.
* com\_ge\_dt\_tsc\_queryEndpoint is the endpoint used for querying time series data. It is usually:<https://time-series-store-predix.run.aws-usw02-pr.ice.predix.io/v1/datapoints>
* com\_ge\_dt\_tsc\_ingestionEndpoint is the endpoint used to ingesting time series data. This endpoint is not used in this service but is required by the Predix Time Series client. It is usually [wss://gateway-predix-data-services.run.aws-usw02-pr.ice.predix.io/v1/stream/messages](http://wss/gateway-predix-data-services.run.aws-usw02-pr.ice.predix.io/v1/stream/messages)
* com\_ge\_dt\_tsc\_zoneId is the Predix zone id of your time series service. Retrieve it by

|  |
| --- |
|  |

* com\_ge\_digitaltwin\_tutorial\_data\_coefficient\_coefficientService is the endpoint used to query model coefficient data from the tutorial-model-coefficient service. Substitute your model coefficient service prefix for “<YOUR\_OWN\_MODEL\_COEFFICIENT\_SERVICE\_PREFIX>” as defined in the "Create the tutorial-model-coefficient application" section above.
  1. Push it to Cloud Foundry

|  |
| --- |
| C:\steam-turbine-tutorial\tutorial-svcs> cf push <YOUR\_OWN\_UNIQUE\_PREFIX>-tutorial-data --no-start |

* 1. You'll need to set environment variables for security\_oauth2\_client\_clientSecret and com\_ge\_dt\_tsc\_clientSecret. You could set this in the manifest.yml file instead, but we recommend using an environment variable as it is more secure than storing passwords in a file, which may result in them being accidentally committed to your source code repository.

|  |
| --- |
| C:\steam-turbine-tutorial\tutorial-svcs> cf set-env <YOUR\_OWN\_UNIQUE\_PREFIX>-tutorial-data security\_oauth2\_client\_clientSecret <your Client Id's secret> C:\steam-turbine-tutorial\tutorial-svcs> cf set-env <YOUR\_OWN\_UNIQUE\_PREFIX>-tutorial-data com\_ge\_dt\_tsc\_clientSecret <your Client Id's secret> |

* 1. Start your application.

|  |
| --- |
| C:\steam-turbine-tutorial\tutorial-svcs> cf start <YOUR\_OWN\_UNIQUE\_PREFIX>-tutorial-data |

* 1. Run the "cf apps" command to discover the URL to your service. Prepend the <https://> protocol then append your API path to get the full URL to your data service

|  |
| --- |
| D:\mydir\steam-turbine-tutorial\tutorial-svcs> cf apps Getting apps in org DigitalTwin / space dev as 200000000@mail.ad.ge.com... OK name                                requested state   instances   memory   disk   urls ... tutorial-data                      started                1/1             1G           1G     **dt-tutorial-data.run.aws-usw02-pr.ice.predix.io** |

We’ll test this application after we’ve pushed some time series data into the service in the next section.

**Create sample data**

In order to demonstrate that we can retrieve time series data from the time series service, we need to load some sample data into the time series service for our asset(s). We have a sample utility in steam-turbine-tutorial/tutorial-util/tutorial-timeseries-util. This utility can be used to create some demonstration data. This simulates a service that, in real-world Digital Twin applications, would capture data from the asset’s sensors and save it for analysis. Refer to [Getting Started](https://predix-io-dev.grc-apps.svc.ice.ge.com/resources/tutorials/tutorial-details.html?tutorial_id=1915&tag=1913&journey=Digital%20Twin%20Runtime%20Starter%20Kit&resources=1914,1915,1916,1917,1918,1919,1920) for download instructions.

This utility reads steam turbine data from a local CSV file which contains timestamp, rpm, and actual temperature values and pushes that data to a Predix Time Series Service instance. See the example input file here (and also in the resources directory):

|  |
| --- |
| timestamp,rpm,temperature 1420070400000,500,501.57855029380073 1420070460000,500,500.0512064418933 1420070520000,500,501.5429791929606 1420070580000,500,501.64609451559187 1420070640000,500,501.42537209804453 1420070700000,500,501.4978736145987 |

**Installation**

Adjust the application.yml file in the src/main/resources folder and the relevant constants in the com.ge.digitaltwin.tutorial.util.IngestionService class as appropriate for your environment, and then to build your executable jar file,run the following:

|  |
| --- |
| mvn clean install |

Note that the section with the key “com.ge.dt.tsc” is the configuration of the **D**igital **T**win **T**ime**S**eries **C**lient.

|  |
| --- |
| com.ge.dt.tsc:  token-endpoint: https://<YOUR\_UAA\_INSTANCE\_HERE>.predix-uaa.run.aws-usw02-pr.ice.predix.io/oauth/token  clientId: <YOUR CLIENT ID>  zoneId: <YOUR\_PREDIX\_TIMESERIES\_SERVICE\_ZONE\_ID>  ingestionEndpoint: wss://gateway-predix-data-services.run.aws-usw02-pr.ice.predix.io/v1/stream/messages  queryEndpoint: <https://time-series-store-predix.run.aws-usw02-pr.ice.predix.io/v1/datapoints>  proxyHost: <HOSTNAME>  proxyPort: <PORT>  logging.level.com.ge.dt.tsc: DEBUG logging.level.com.ge.digitaltwin.tutorial.util: debug |

* Substitute your UAA instance name for <YOUR\_UAA\_INSTANCE\_HERE>.
* Substitute your client id for <YOUR\_CLIENT\_ID>.
* Substitute your Predix time series service zone id for “<YOUR\_PREDIX\_TIMESERIES\_SERVICE\_ZONE\_ID>”
* Set the proxyHost and proxyPort as appropriate for your environment.

**Executing the utility**

Execute the utility with a command similar to the following (make sure the -D options appear before -jar). For the purposes of this tutorial, please use “2” as the <steam-turbine-id> as this will create consistency with the remaining sections.

|  |
| --- |
| D:\Projects2016\pedro\steam-turbine-tutorial\tutorial-util\tutorial-timeseries-util>java -Dhttps.proxyHost=<your-proxy-host> -Dhttps.proxyPort=<your-proxy-port> -Dconfig.steamTurbineId=<steam-turbine-id> -Dconfig.csvFilename=<csv filename in resources directory or path/filename on the local file system> -Dcom.ge.dt.tsc.clientSecret=<client-secret-for-tutorial-svcs-client> -jar target\tutorial-timeseries-util-1.0-SNAPSHOT.jar |

Note: We recommend using the path/filename convention when specifying the config.csvFilename parameter. When executing this utility directly from a jar file, some systems have difficulty finding the config.csvFilename when specified as a file in the resources directory.

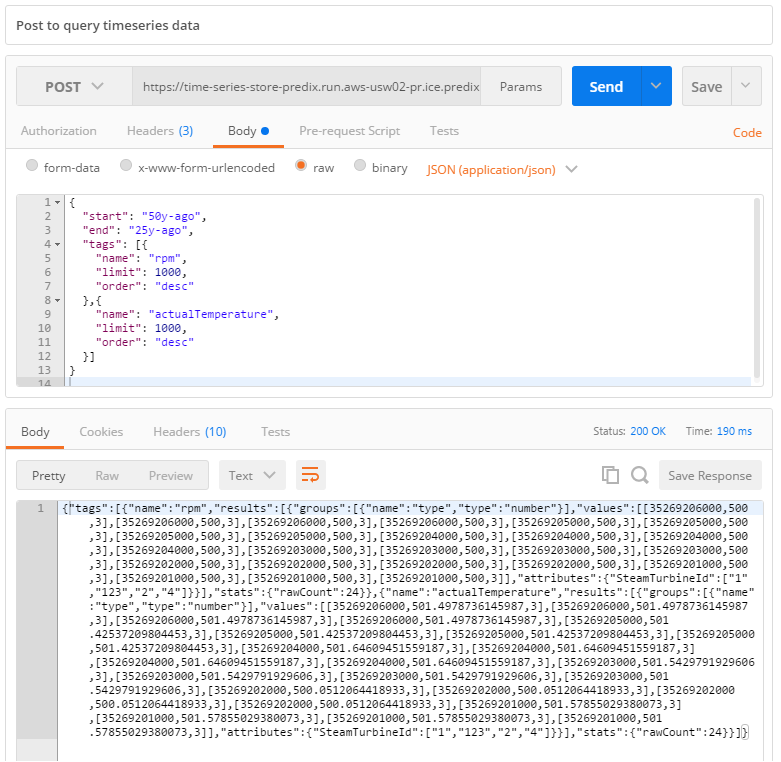
The console will echo something similar to this:

|  |
| --- |
| Console Output  ##### Configuration: TimeseriesUtilConfig{steamTurbineId='steamTurbine-007', csvFilename='d:\Temp\data\_points\_turbine\_test.csv'} ##### TutorialTimeseriesUtilController: up and running... ##### IngestionService instance: com.ge.digitaltwin.tutorial.util.IngestionService@2cab9998 ##### resource d:\Temp\data\_points\_turbine\_test.csv not found, looking for file... ##### found file: d:\Temp\data\_points\_turbine\_test.csv ##### Found 6 SteamTurbineDataPoints in d:\Temp\data\_points\_turbine\_test.csv for steamTurbineId= steamTurbine-007 ##### sending steam turbine data payload: 1 ##### Done, processed 6 steam turbines. ##### This application may take significant time to complete and exit cleanly as queue processing time is proportional to the size of the data. |

**Verification**

By convention, testing data is between the years 1970 and 1979. To verify the success of the data push, use Postman (or some similar tool) to POST to the time series query URL  https://time-series-store-predix.run.aws-usw02-pr.ice.predix.io/v1/datapoints with the header parameter Predix-Zone-Id set to the time series service guid and the following body to show your data (adjust the start and end values as needed). There is also a request saved in our Postman collection named “Post to query timeseries data” that you can conveniently run.

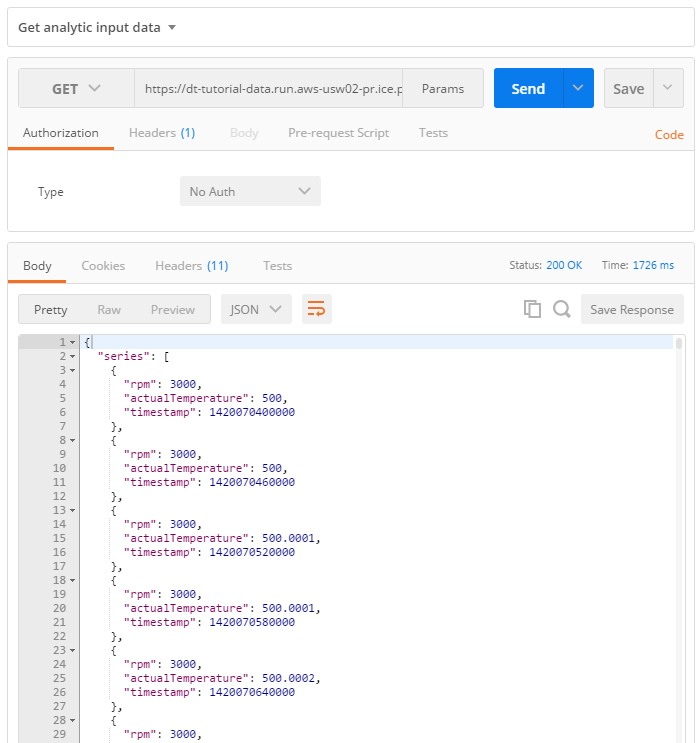
|  |
| --- |
| **Time Series JSON query input** |
| {  "start": "50y-ago",  "end": "25y-ago",  "tags": [{  "name": "rpm",  "limit": 1000,  "order": "desc"  },{  "name": "actualTemperature",  "limit": 1000,  "order": "desc"  }] } |



**Push the tutorial’s steam turbine data**

After you’ve pushed and verified the six example data points above, you can confidently push a larger data set that this tutorial will be using throughout. Since you already cloned the tutorial’s code base from GitHub, you will have a “data” directory at the top level. In it, you’ll find a much larger data file named “data\_points\_turbine\_1.csv”. Re-execute this tutorial-timeseries-utility on this data file under the same assetId (2) as before and you will have “once-per-minute” data covering all of 2015.

Now that you have a significant amount of data in your time series service, let’s use the tutorial-data application that you created earlier to get that data. This will serve to test that the application works as intended. In your Postman collection, run the “Get analytic input data” request and you should see something like the following:



What you learned

You learned how to set up and test the services needed to interact with the data that will feed our sample Digital Twin model. You also saw how to push sample data into your time series service. These services publish the following endpoints (with sample invocation URLs):



We will continue with building and deploying models in Step 2.