**Build the Digital Twin Starter Kit: Getting Started**

**What you'll learn to do**

Set up your environment to run a sample Digital Twin. Download and build the provided sample code.

**What you need to set up**

This section will introduce recommended development tools and environment configuration.

**Recommended development tools**

IntelliJ (with BPMN plugin), Eclipse (with BPMN plugin), or some similar tool

* IntelliJ: <https://www.jetbrains.com/idea/download/>
* Eclipse: <https://eclipse.org/downloads/>

Cloud Foundry command line client

* <https://docs.cloudfoundry.org/cf-cli/install-go-cli.html>

UAAC - UAA Command line client - Many UAA functions can be done with the Predix Security Starter Kit, but you might need UAAC for more complex efforts or troubleshooting problems.

* [https://www.predix.io/docs/?r=705324#Kmchpf5k](https://predix-io-dev.grc-apps.svc.ice.ge.com/docs/?r=705324#Kmchpf5k)

Maven

* <https://maven.apache.org/install.html>

Git

* https://git-scm.com/

Java 8

* <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

Postman

* <https://www.getpostman.com/>

Node / Bower / Grunt

* See the “Tooling” prerequisites at <https://github.com/PredixDev/predix-seed>

**Using Postman or Swagger to exercise REST endpoints**

We recommend using Postman or Swagger to interact with and test REST endpoints.

**Postman**

For your convenience, we have included a Postman collection in the root directory of the source code called DigitalTwinTutorial.postman\_collection.json. You can import this collection into Postman by following the instructions here: <https://www.getpostman.com/docs/collections>.

Inside this collection are folders corresponding to each of the 5 steps taken to build our sample Digital Twin app. In each folder there are example REST calls we used to create and test our application.  We will use this Postman collection throughout the tutorial to demonstrate how to interact with the services provided in the Starter Kit.

**Swagger**

All of the services with available REST endpoints utilize a tool called Swagger to generate documentation for the available REST endpoints. Swagger also allows you to execute HTTP requests through its online user interface. We will discuss this in greater detail later.

# What you need to do

# Creating services

# Our starter kit makes use of 7 different Predix services. These services can be created in several ways:

# via a Perl script available at (<https://github.com/DigitalTwin/dt-starter-kit-creation-script>)

# via your browser from the service catalog (<https://www.predix.io/catalog/services/>)

# via the command line using the Cloud Foundry command line client

# We recommend that you use the script if you wish to save time. The script will abstract and hide the details of configuring this tutorial application. In other words, you will not learn as much about instantiating and wiring these services together. This may be acceptable if you already understand how Predix services work. Otherwise, after using the script, we suggest that you investigate the script to understand how it works and to consult the documentation and guides available at [Predix.io](https://predix.io).

|  |  |  |
| --- | --- | --- |
| Service | URL | Starter Kit Step |
| UAA | https://www.predix.io/services/service.html?id=1172 | 1-5 |
| PostgreSQL | https://www.predix.io/services/service.html?id=1178 | 1 |
| Time Series | https://www.predix.io/services/service.html?id=1177 | 1 |
| Analytics Catalog | https://www.predix.io/services/service.html?id=1187 | 2 |
| PostgreSQL | https://www.predix.io/services/service.html?id=1178 | 3 |
| RabbitMQ | https://www.predix.io/services/service.html?id=1182 | 3 |
| Redis | https://www.predix.io/services/service.html?id=1215 | 5 |

**Naming your services**

The convention used in this tutorial is to prefix all your service and application instances an easily identifiable (and preferably unique) prefix. Many service and application names are hyphenated. It is important to not use the underscore (“\_”) character in your names as some entities that parse the manifest.yml file will have problems with this character.

**Setting up your UAA service**

Setting up a Predix UAA service in your own Predix space is required to build the sample Digital Twin. The Perl script will do this for you. Alternatively, you can find instructions on how to set up your own UAA service on the Predix website (<https://www.predix.io/services/service.html?id=1172>). For tutorials on how to configure the Predix UAA service, see the Exploring Security Services tutorial at <https://predix-io-dev.grc-apps.svc.ice.ge.com/resources/tutorials/journey.html#1613>.

Once you have provisioned a UAA service, you can choose to configure your UAA service through the UAAC command line interface (instructions for installing can be found here: <https://github.com/cloudfoundry/cf-uaac>), or through the Predix Tool Kit (<https://predix-starter.run.aws-usw02-pr.ice.predix.io/>). For inexperienced users, it is often easiest to use the Predix Tool Kit.

We recommend the following configuration for your UAA service:

Two new groups:

* tutorial.user
* tutorial.admin

Two new clients:

* tutorial-svcs (for use with the back-end services)
  + scope: uaa.none
  + resource\_ids: none
  + authorized\_grant\_types: “client\_credentials”
  + autoapprove:
  + actions: none
  + authorities = “uaa.resource”, “tutorial.user”, ”tutorial.admin”, <timeseries\_ingestion\_scopes>, <timeseries\_query\_scopes>, <analytics\_catalog\_scope>
    - NOTE: In order to use the analytics catalog and time series services that you create you must include the proper authorities here. More information can be found in the documentation for the analytics catalog and time series the [Updating OAuth2 Client to Use Analytics Catalog Service](https://www.predix.io/docs/?r=311933#YDrrlVqu) and [Time Series Service Setup](https://www.predix.io/docs/#BWTIwx4i) sections.
  + name: tutorial-svcs
* tutorial-user (for use with the front-end visualization)
  + scope = “tutorial.admin”,”tutorial.user” ~~<analytics\_catalog\_scope>~~
  + resource\_ids: none
  + authorized\_grant\_types = “refresh\_token”, ”password”, ”authorization\_code”
  + redirect\_uri: <visualization\_uri>
    - You will need to set this to the uri of the visualization you will create in step 5
  + autoapprove: “tutorial.admin”,”tutorial.user” ~~<analytics\_catalog\_scope>~~
  + action: none
  + authorities = “uaa.resource”

Two new users (add them to the appropriate groups):

* tutorial-user
  + groups = tutorial.user
* tutorial-admin
  + groups = tutorial.user, tutorial.admin

We have included a Perl script in this Starter Kit that will set up UAA and all other service instances, wire them to an app, add the appropriate authorities and zone ids, and create the aforementioned relationship between users, groups, and clients in the UAA client. This script is here: <https://github.com/DigitalTwin/dt-starter-kit-creation-script> along with the its documentation. The console output from the Perl script will echo the name of the postgres service instance and the UAA clients. Note these as you’ll need them later. Note that if you opt not to use this script then you will be creating the individual service instances as you progress through this tutorial. The impact of that is that you’ll need to refer to the above UAA configuration information to see where your service instance scopes or URIs are needed. If you chose this scripted approach, then clone, configure, and run the script now to set up UAA and all other services instances.

For more information on UAA, see the following documentation:

* [CloudFoundry UAA](https://predix-io-dev.grc-apps.svc.ice.ge.com/docs#SJzDWEsh) - <https://github.com/cloudfoundry/uaa>

**Configure Postman**

Now that you have a UAA service set up, you will need to configure the provided Postman collection to authenticate against that UAA service to interact with the Starter Kit service endpoints.

Instructions on how to configure your Postman environment can be found here: <https://www.getpostman.com/docs/environments>

Since the strings below are frequently used, the collection is configured to read three different variables from your Postman environment:

token = <bearer token>

The {{token}} variable is used in the header of requests as part of the value associated with the “Authorization” key. This <bearer token> is your “tutorial-svcs” client’s bearer token. It can be easily found as the value of “access\_token” when you “Login as Client” in the [Predix Tool Kit](https://predix-starter.run.aws-usw02-pr.ice.predix.io/). You’ll need three pieces of information to do this:

1. UAA URL – This is the value of the “uri” key in the “predix-uaa” section of the output of “cf env <app-name>” where <app-name> is the name of the dummy application that your UAA instance was bound to. If there was more than one UAA instance bound to this app, then you’ll need to make sure that the value associated with the “name” key is the name of your UAA instance (this was set in the Perl script).
2. Client ID – This is the client for the tutorial’s services. By default, the Perl script uses “tutorial-svcs”.
3. Client Secret – This is the password that the script set for this client.

analytic-zone-id = <Predix analytics catalog zone id>

The {{analytic-zone-id}} is used in the header of requests to the analytic catalog service as the value associated with the key “Predix-Zone-Id”

credentials = <Base64(client-id:client-secret)>

The {{credentials}} variable is used in the header of the request to get your bearer token as part of the value of the “Authorization” key.

The Predix analytics catalog zone id was assigned when your catalog instance was created. The client secret was created when you configured you UAA service instance.

**Resolving dependencies**

The sample code is provided as a maven project. All the dependencies for the sample services are available from the public maven repositories. Before building the tutorial code, you need to build and install a release of the digital-twin-time-series-client from a zip file available in Github (a release was made so that any future API changes to the client would not impact this dependency). It is available at https://github.com/DigitalTwin/ditigal-twin-time-series-client.

* 1. Go to <https://github.com/DigitalTwin/digital-twin-time-series-client/releases> and download the source code for version 3.0.0. Unzip/untar the downloaded file and go into the top-level directory.

|  |
| --- |
| $ cd digital-twin-time-series-client-3.0.0 |

* 1. Build the code using Maven

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

**Building the sample code**

**Download the code**

The code to build the tutorial services, analytic, and a utility to load some sample data is found in <https://github.com/DigitalTwin/steam-turbine-tutorial>.

Using Git, download the steam-turbine-tutorial

|  |
| --- |
| $ git clone https://github.com/DigitalTwin/steam-turbine-tutorial.git  $ cd steam-turbine-tutorial |

**Confirm your group name against the SecurityConfiguration class**

Regarding your previous UAA configuration from above, note that the group named “tutorial.user” contains a user account named “tutorial-user”. This “tutorial.user” group is added to the scope of the client named “tutorial-user”. The group name “tutorial.user” is significant in that the Java source code that you just downloaded is pre-hardwired for security purposes to permit members of your “tutorial.user” group access to the back-end services. If you decided to name this group something other than “tutorial.user” then that change needs to be reflected in the Java source code, namely the “SecurityConfiguration.java” class that exists in all 7 of the submodules in /steam-turbine-tutorial/tutorial-svcs module (the only submodule in /steam-turbine-tutorial/tutorial-svcs that is not impacted is the /steam-turbine-tutorial/tutorial-svcs/tutorial-commons submodule). Before you build the code, you will need to locate the “**SecurityConfiguration.java**” class in the source tree of **each of the 7 submodules** and make the changes detailed below:

|  |
| --- |
| SecurityConfiguration.java from /tutorial-svcs/tutorial-asset/src/main/java/com/ge/digitaltwin/tutorial/asset |
| … @Configuration @Profile("!dev") @EnableResourceServer @EnableGlobalMethodSecurity(prePostEnabled = true) public class SecurityConfiguration extends ResourceServerConfigurerAdapter {  @Override  public void configure(HttpSecurity http) throws Exception {  http.authorizeRequests()  .antMatchers("/swagger-ui.html", "/webjars/springfox-swagger-ui/\*\*",  "/configuration/\*\*", "/swagger-resources/\*\*", "/v2/api-docs").permitAll()  .antMatchers("/asset/\*\*").access("#oauth2.hasScope('**tutorial.user**')")  .anyRequest().denyAll();  }   @Override  public void configure(ResourceServerSecurityConfigurer resources) throws Exception {  resources.resourceId("**tutorial**");  } } |

* Replace tutorial.user with the **name of the group** that contains the user that you wish to have access to the back-end services via the visualization application.
* Replace tutorial with the **prefix of the name of the group** that contains the user that you wish to have access to the back-end services via the visualization application.

**Build each module using Maven**

|  |
| --- |
| $ cd tutorial-analytics/tutorial-actual-vs-expected-analytic  $ mvn clean install  $ cd ../../tutorial-svcs  $ mvn clean install  $ cd ../tutorial-util/tutorial-timeseries-util  $ mvn clean install |

The code to build the tutorial visualization app is found in <https://github.com/digitalTwin/steam-turbine-tutorial-vis>. Clone this module into your local file system. We'll discuss configuring, building, and deploying this in Step 5.

**What you learned**

You have learned how to configure your environment to run the Digital Twin Starter Kit, and have learned how to build the provided sample code. You have learned what modules are present in the provided code and how they implement the 5 Steps discussed in the Introduction. Bottom of Form