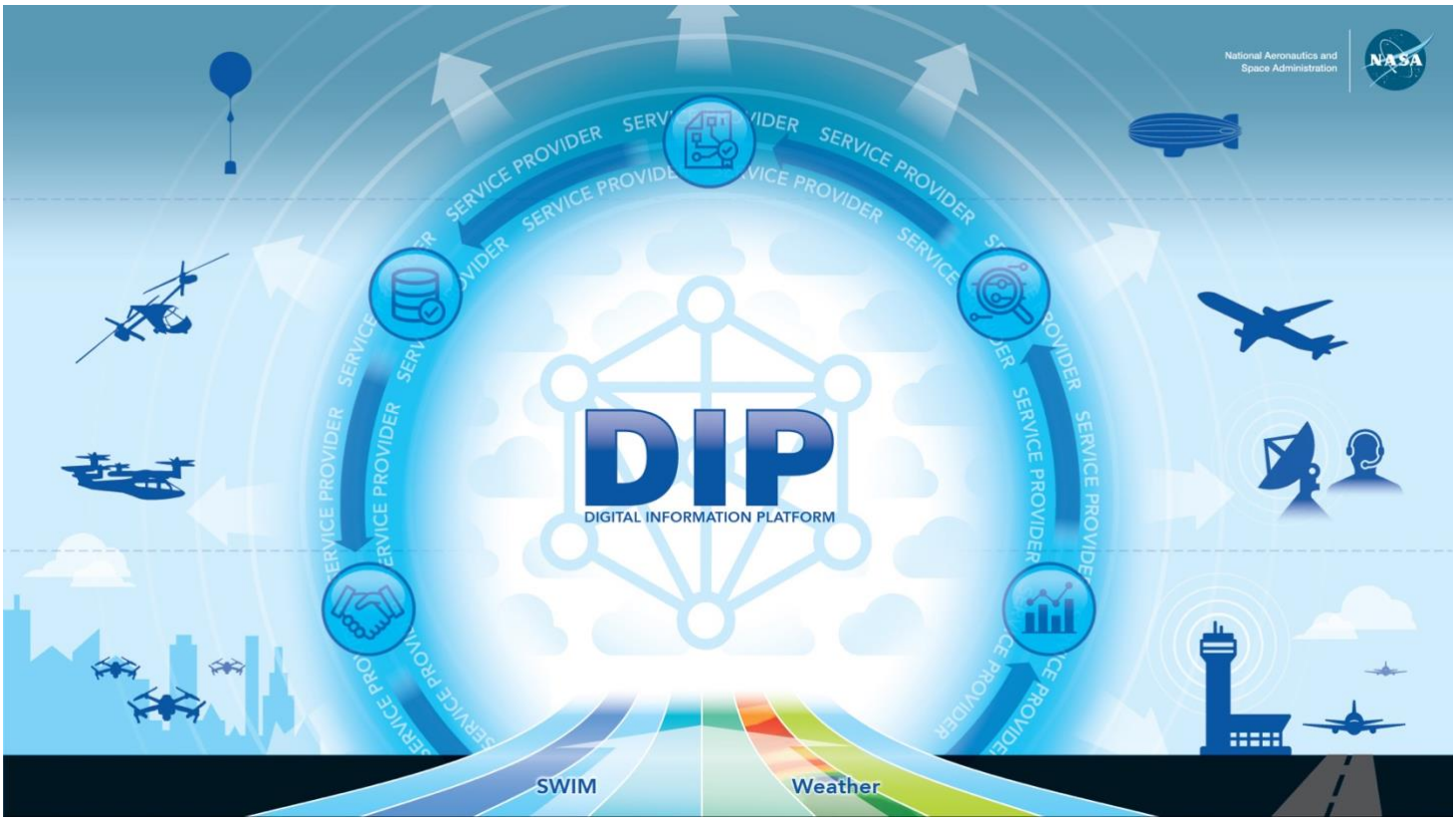
# Digital Information Platform

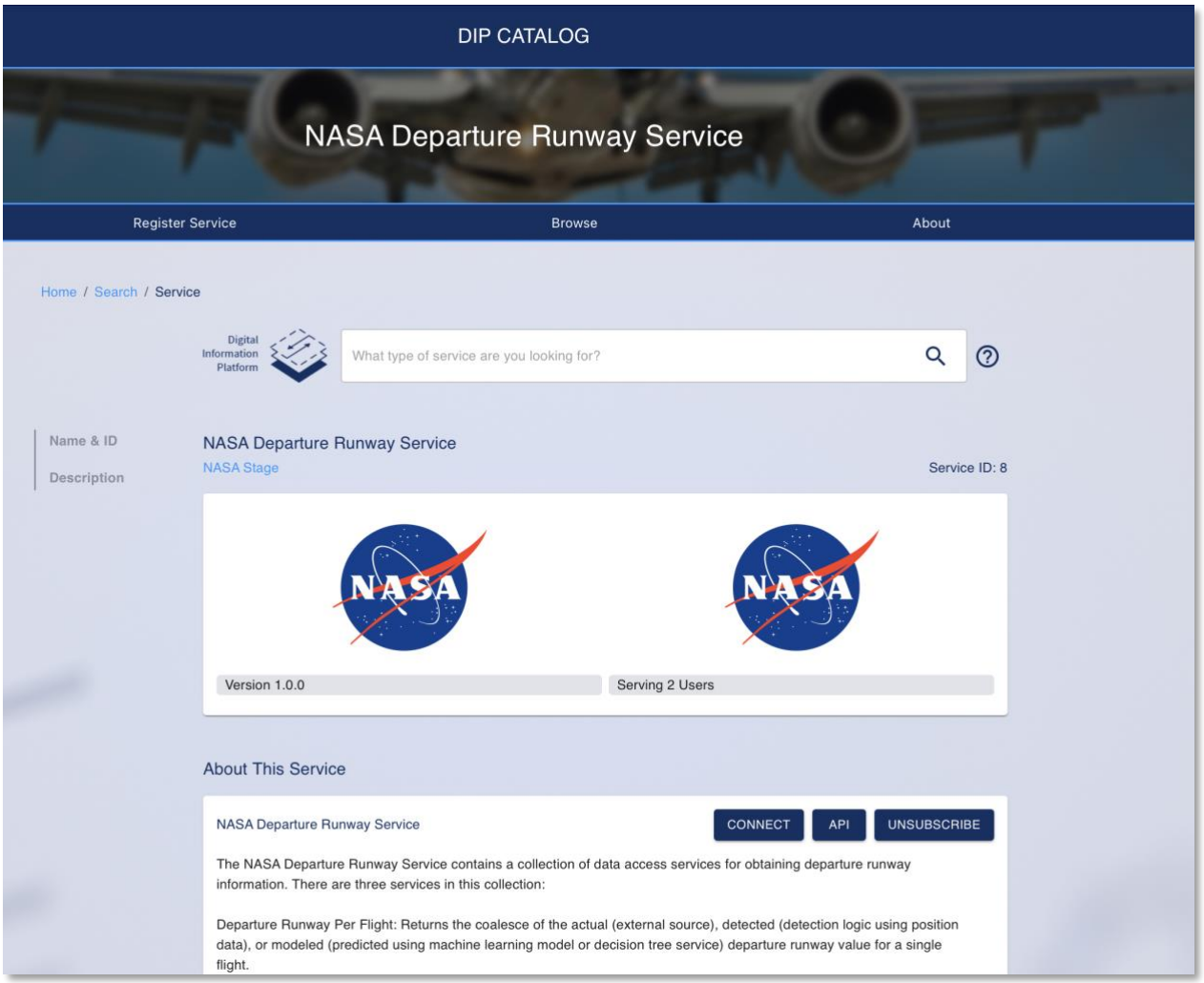



# How To Invoke A REST API With DIP

# Table of Contents

# 1. Try Out An API

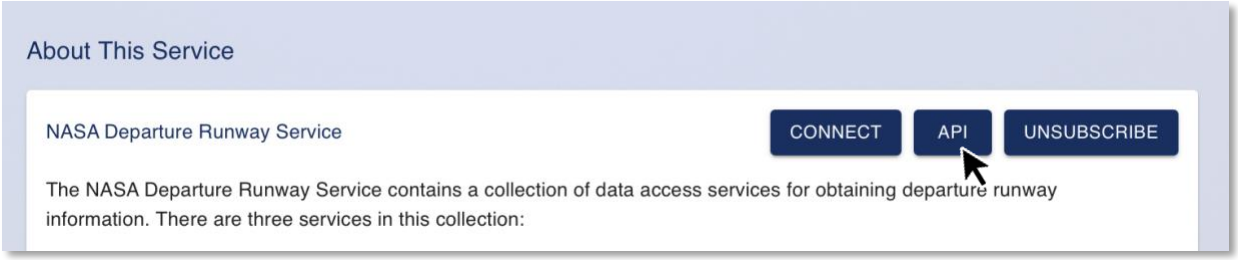
Starting point: Service information page on the DIP Platform website.




## Mock it Out: Try an API Without a Subscription

Services API can be tested without a subscription. In this case, both the request and responses are limited to mock data. The intent of the mock data is to provide an example of the request/response input parameters, headers, and data-models. This data is for educational purpose only. The steps are as follows:

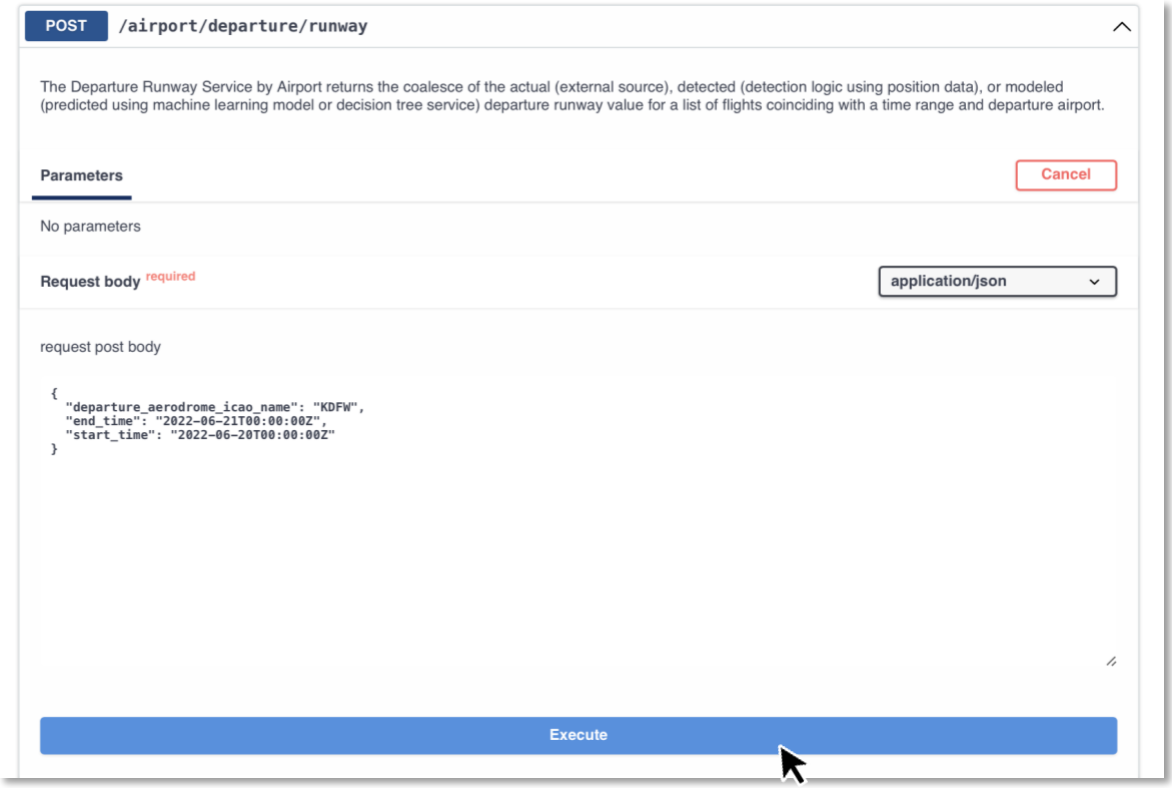
1. Click on “API” to open the Open API specifications.



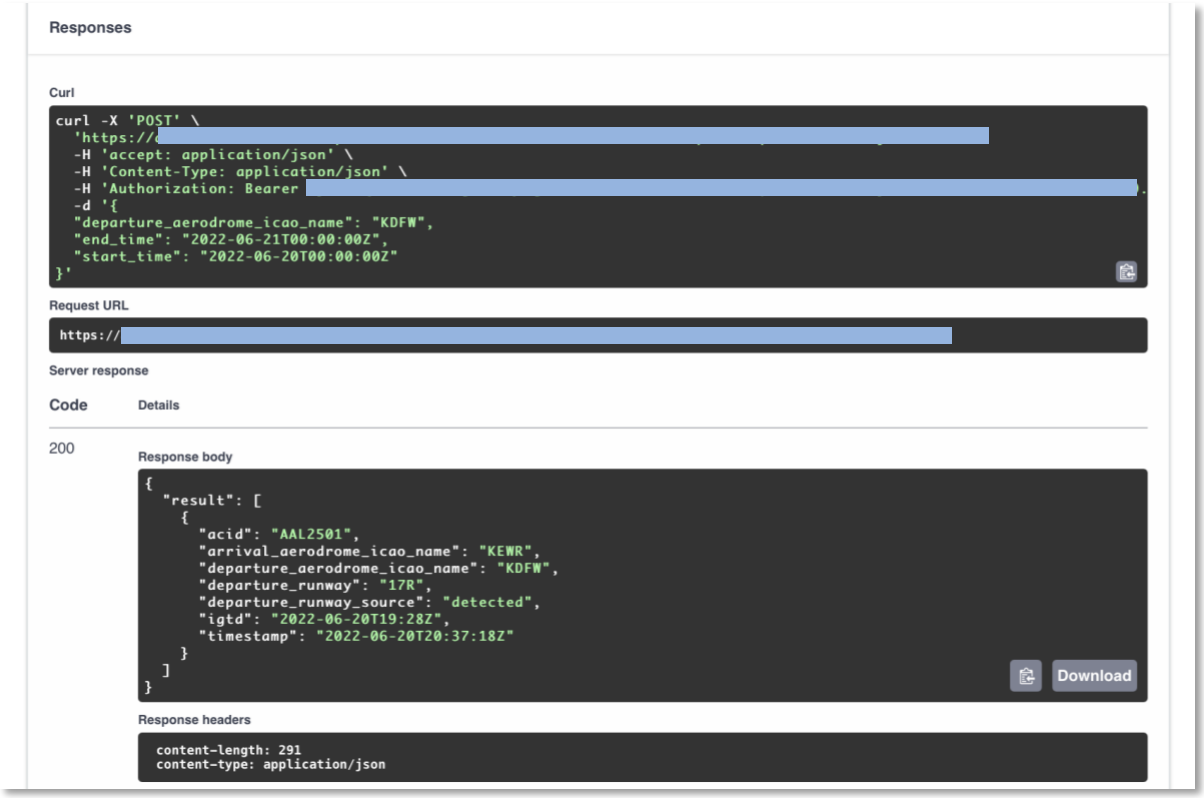

2. Click on “POST” to select the service of interest.


3. Click on “Try It Out.”

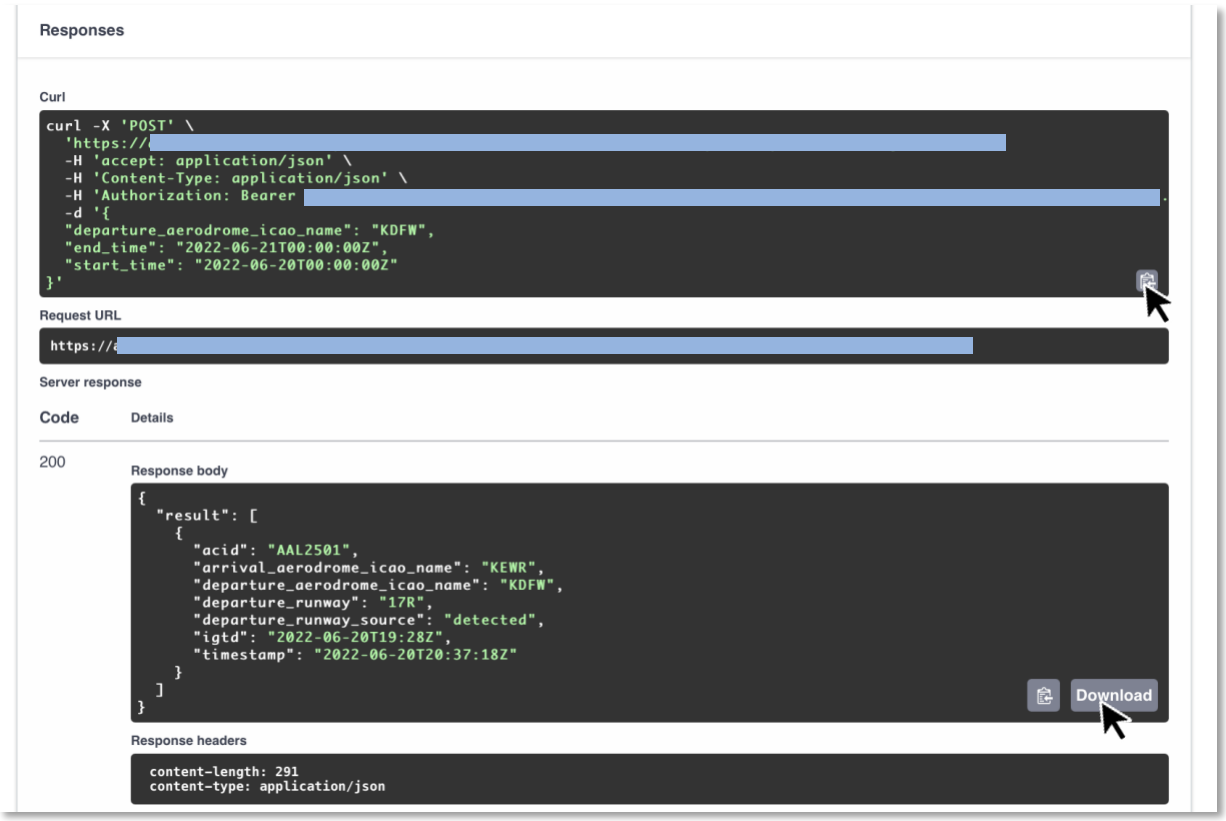

4. Click on “Execute.” Note: changing the “request post body” in the Mock it out mode would have no impact on the response.




5. Examine the result in the Response body section.



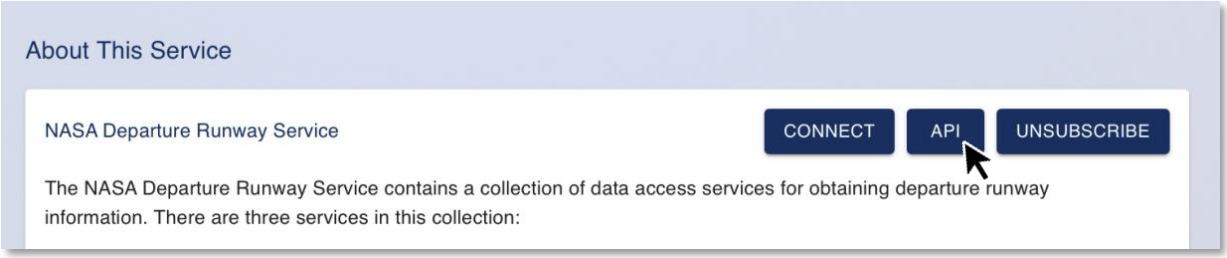
6. Optionally, click on “Download” to save the response or click on the copy icon  to copy the response, or the script and paste them into another application.




## Try it Out: Try an API With a Subscription

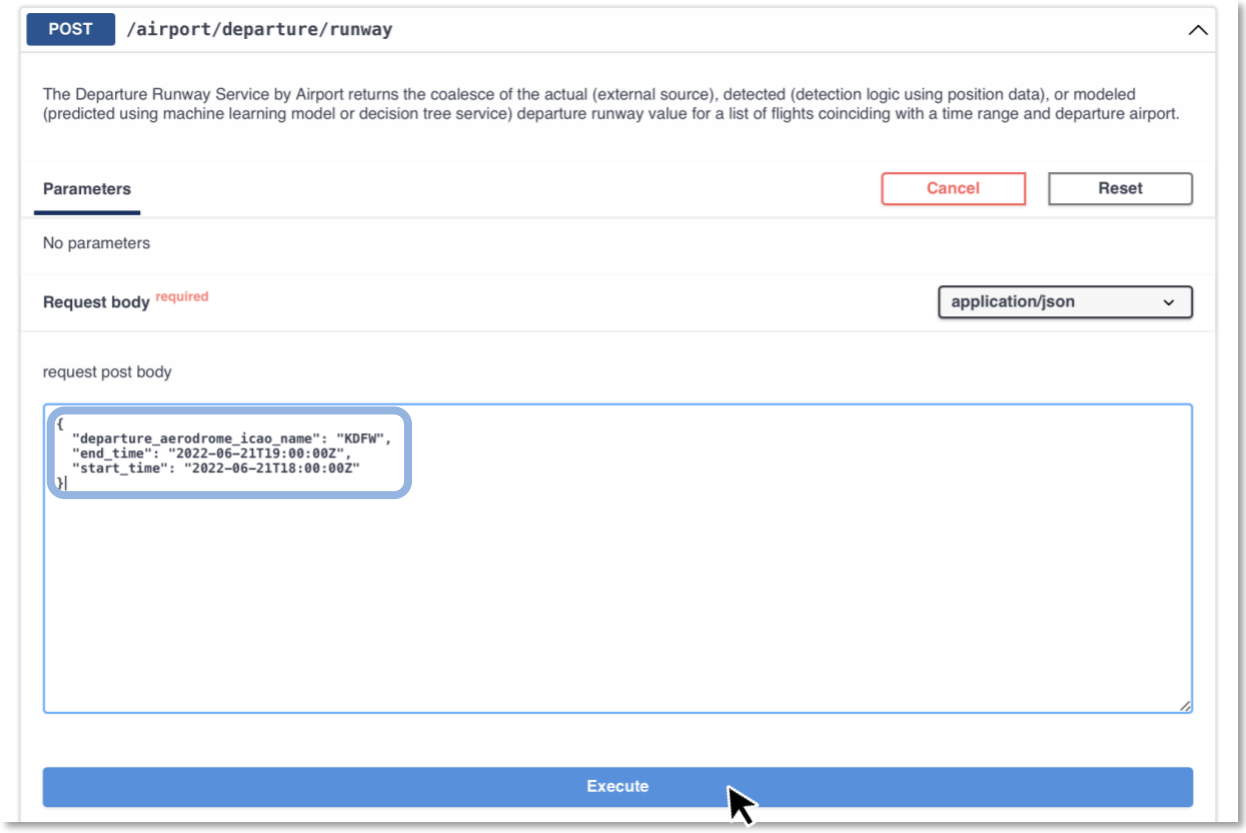
Services API can be tested with a subscription. Unlike testing with unsubscribed Mock Data, a subscription allows requests to return real data. Valid responses are limited to the scope of the data set, so requests need to be mindful of these limitations. In this case, the request and responses are only limited to available data. The steps are as follows:

1. Click on “API” to open the Open API specifications.

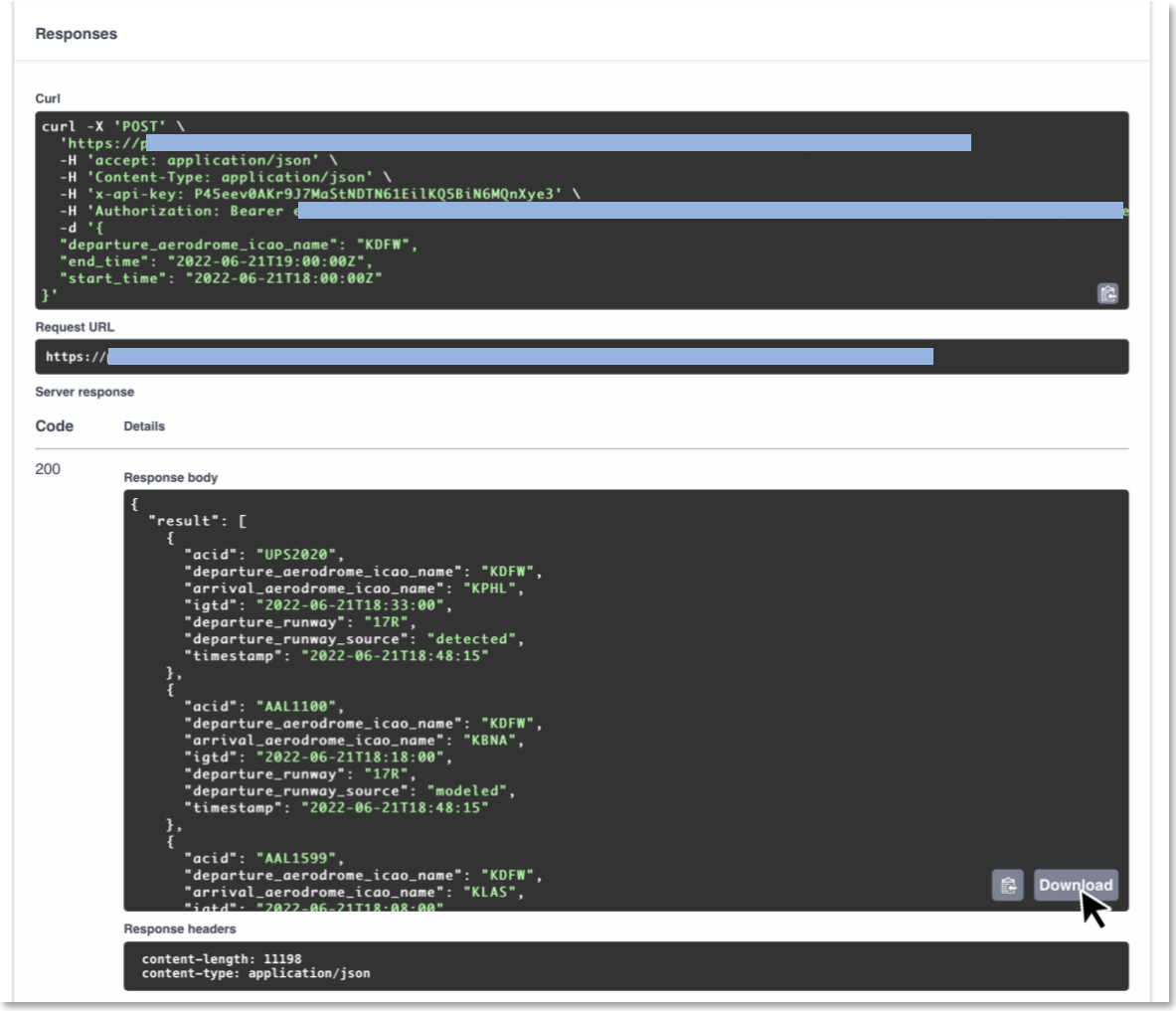



2. Click on “POST” to select the service of interest.


3. Edit the “request post body” script with data within the scope of data indicated in the description section, and click on “Execute.”




4. Examine the result in the Response body section.

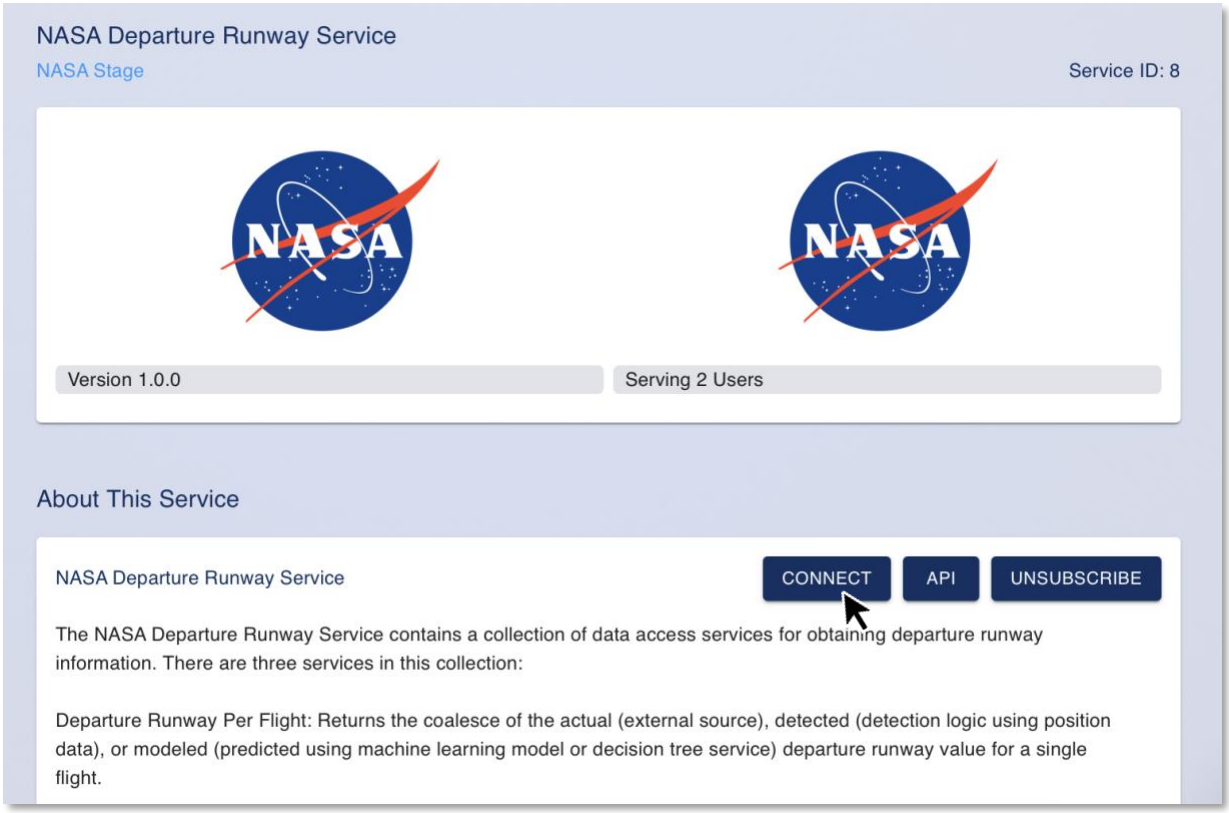


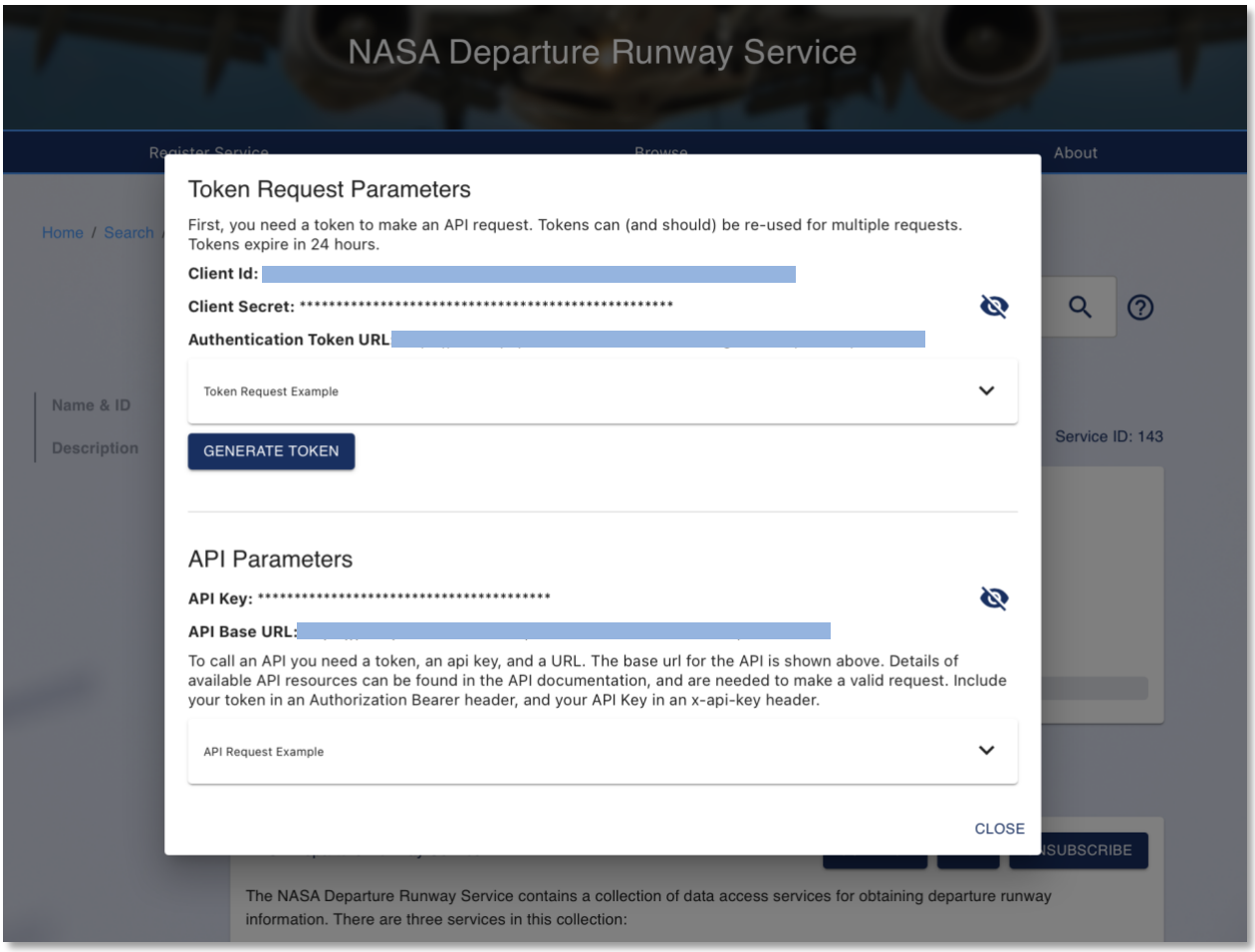

5. Optionally, click on “Download” to save the script or click on the copy icon  to copy the response and paste it into another application.

6. Optionally, a Curl script is included above the response body. It includes the API URL address, the API key and the token required to invoke the service (see the next section about the connection information). Click on the copy icon to copy the script and paste it into another application.

## 2. Connection Information to Make an API Request

On the Service Page, click on the “Connect” button to obtain the Token and API parameters to make requests to the API. A window opens.



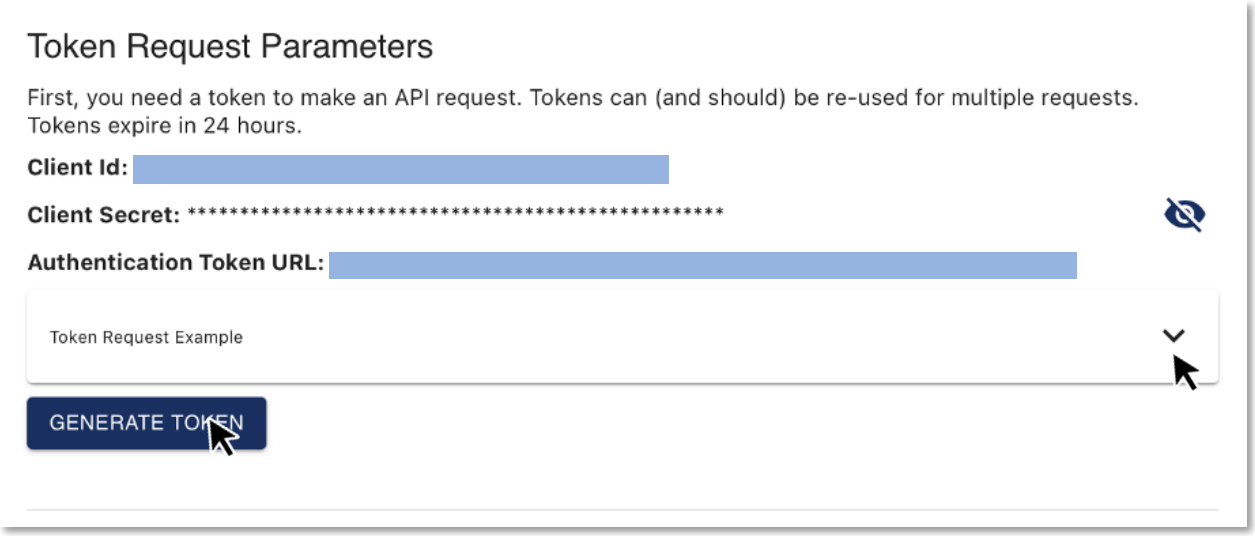


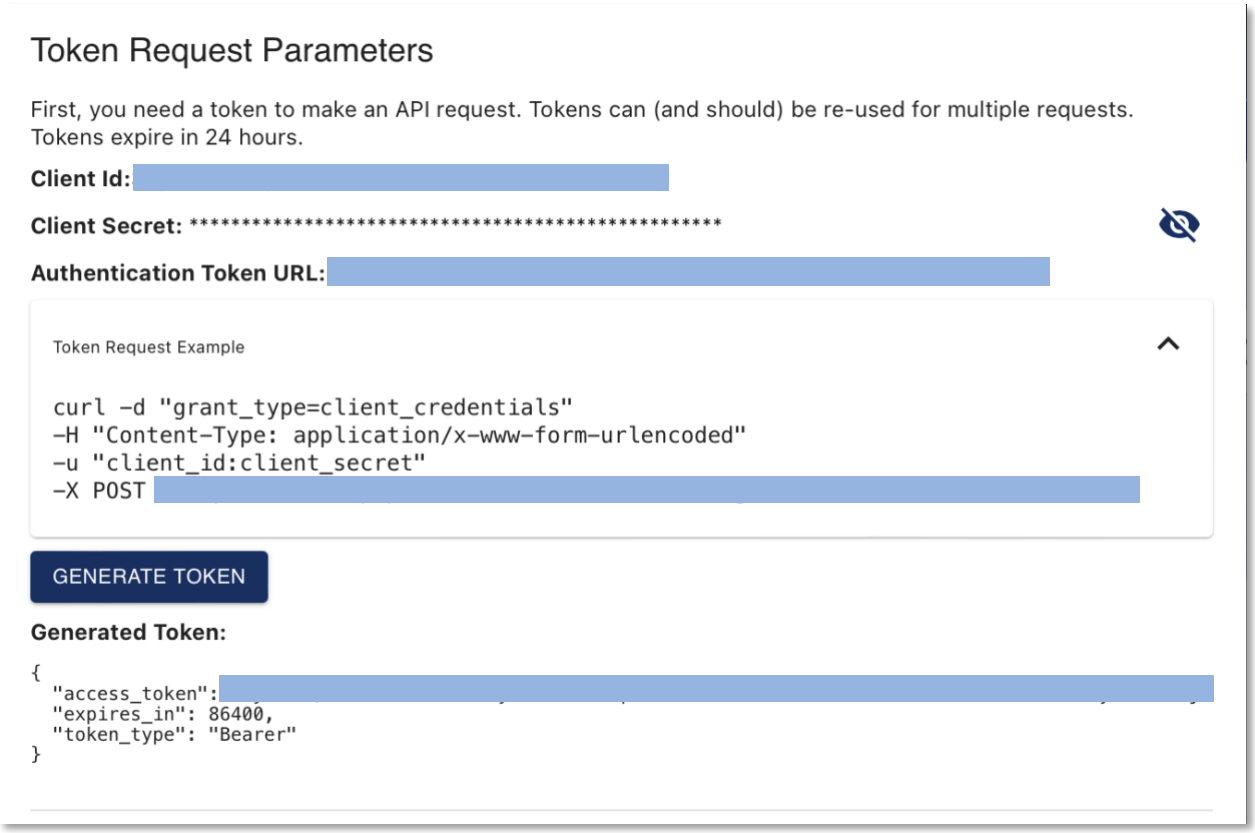

## Token Request Parameters

In the upper section, the key Token Request Parameters are the Client ID and Client Secret information. These are unique to each partner and therefore are confidential information. Both Client ID and Secret expire after 24h. They are used to generate a temporary Token. This token can re-used for multiple requests. All Tokens expire after 24h, however they can be updated automatically by using the example of the Curl script in the window (recommended).

There are 2 ways to create a token:

1. Click on “Generate Token”, and copy that information in another application, or, better,
2. Click on “Token Request Example” to see the CURL script. The script needs to be updated with the Client ID and Client Secret. This script will automatically generate a token, and automatically renew the token when it expires.

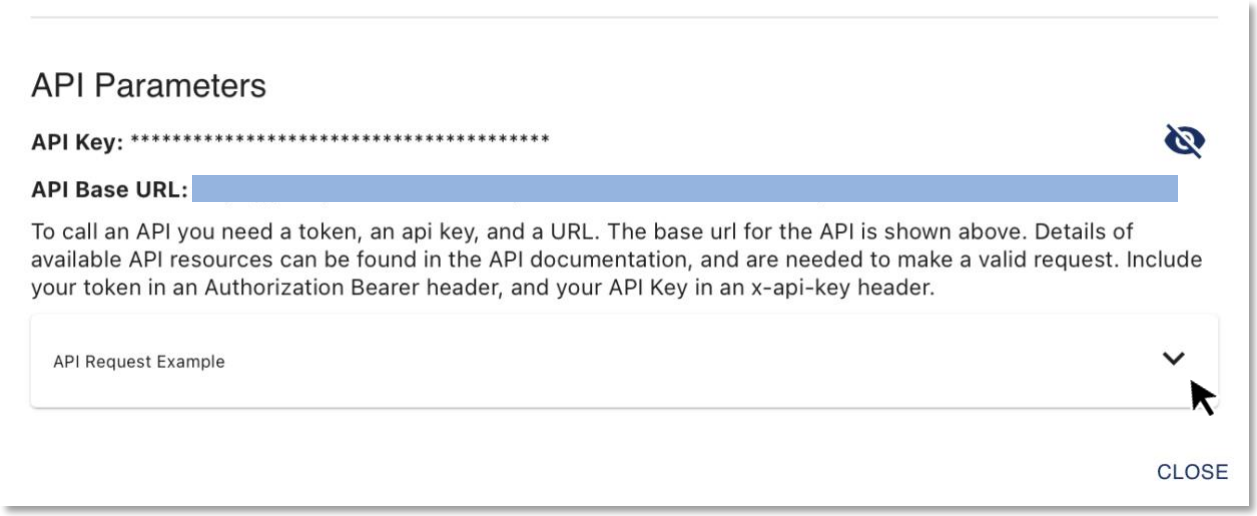


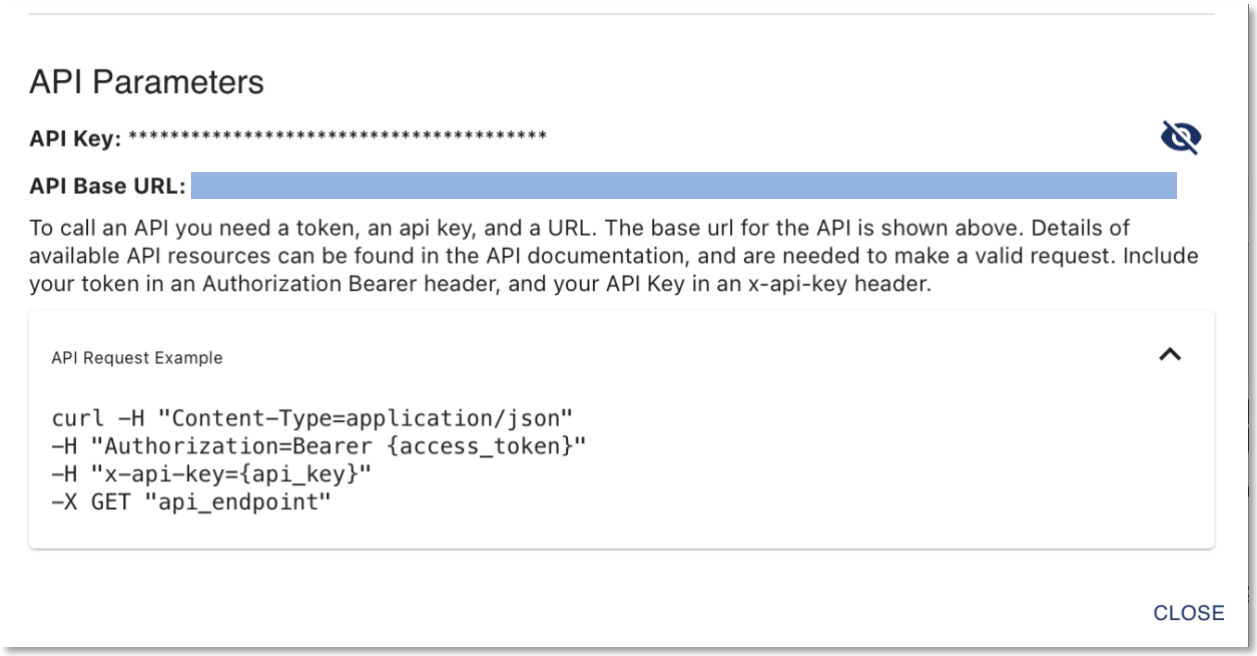




## API Parameters

In the lower section, the API Parameters are the API key and the API Base URL information. These are unique to each service and therefore are confidential information. The API key and URL, as well as the above token are needed to call the API. There are 2 ways to create a request:

1. Copy the API key and the API URL address, or, better,
2. Click on “API Request Example” to see the Curl script. The script needs to be updated with the Access Token and the API key.



## Refreshing Tokens Before They Expire

The CURL script below provides the required information to renew tokens, using credential information.

```
curl -d "grant_type=client_credentials"
-H "Content-Type: application/x-www-form-urlencoded"
-u "client_id:client_secret"
-X POST "https://dev-dipapi.auth.us-east-1.amazoncognito.com/oauth2/token"
```

# 3. API Requests Using Python

## Setup

1. Install Anaconda (https://anaconda.org/) on the system you will be using.
2. Create the Conda environment in which to run the demo code using the following command:

```
conda env create -f conda.yml -n daad-env
```

## Running the Example

1. Activate the environment in which to run the example:

```
conda activate daad-env
```

2. Run the script containing the demo:

Simple console output
```
python daad_app.py
```

Request and graph in plotly dash
```
python daad_app_combo.py
```

Same as daad_app_combo but the code is all inline rather using utils.py
```
python daad_app_combo_inline.py
```

3. After ensuring that no errors were reported, point the browser at the URL indicated in the printed output. If running locally, this should be `localhost:8050`. If running remotely, this should still be on port 8050, but another port could be used, as necessary.

## Customizing the Example

The dates in the scripts are examples. To customize what is displayed, simply edit the lines in `env_vars.py` to set new values for `START_TIME`, `END_TIME`, and `AIRPORT_ICAO`. Additional variables are defined for the API URLs and Credentials. If those need to change, the API calls in python are typically made using the Python requests package. Enter the required headers obtained from the Connect information or Try It Now curl command for a subscribed service as shown in the following example (based on the NASA arrival runway utilization service):

conda.yml:

```
name: daad-env
channels:
  - defaults
  - conda-forge
dependencies:
  - python=3.9
  - pandas=1.1.*
  - numpy=1.20.*
  - pyyaml=5.4.*
  - plotly=5.1.*
  - dash=1.21.*
  - dash-html-components=1.1.*
  - dash-bootstrap-components=0.13.*
  - scikit-learn=1.0.*
  - pip
  - requests=2.28.*
```

env_vars.py:

```
#!/usr/bin/env python

import dash
import requests

import dash_core_components as dcc
import dash_html_components as html
import plotly.express as px
import pandas as pd

TOKEN_URL = '[token url]'
X_API_KEY = '[add api key]'
CLIENT_ID = '[add client id]''
CLIENT_SECRET = '[add client secret]'

#Urls for the arrival runway utilization service on dip dev
BASE_API_URL = '[base URL]'
API_PATH = '/arrival/runway/utilization'
START_TIME = '2022-06-29 00:00:00'
END_TIME = '2022-06-30 00:00:00'
AIRPORT_ICAO = 'KDFW'
```

daad_app_combo_inline.py:

```
#!/usr/bin/env python

import dash
```

```
import requests
import dash_core_components as dcc
import dash_html_components as html
import plotly.express as px
import pandas as pd
from env_vars import *

app = dash.Dash(__name__)

# Request the token
token_request = requests.post(
        url=TOKEN_URL,
        data={'grant_type': 'client_credentials', 'client_id': CLIENT_ID},
        auth=(CLIENT_ID,CLIENT_SECRET)
    )

if token_request.ok:
    token = token_request.json()
else:
    print(f"Failed to get the access token from url: {TOKEN_URL}")
    exit()

# Put the API Key and token in the headers for the API request
headers = {
    'x-api-key': X_API_KEY,
    'Authorization': f"{token['token_type']} {token['access_token']}"
}

# Set up your params for the service
data = {
    'arrival_aerodrome_icao_name':AIRPORT_ICAO,
    'start_time':START_TIME,
    'end_time':END_TIME,
    }

# API full url
url = f'{BASE_API_URL}{API_PATH}'

# Call the API
response = requests.post(url, json=data, headers=headers)
print(f"Sending the request to {url}")

if response is not None and response.ok:
    data = response.json()
    print(f"Success getting data from url: {url}")
```

```
else:
    print(f"Failed to get data from url: {url}")
    exit()

# Put the data in a data frame
dat = pd.DataFrame.from_records(data["result"])

if not "hour" in dat.columns:
    print("Some data seems to be missing")
    exit()

# Graph the data
fig = px.bar(
    dat,
    x="hour",
    y="arrival_runway_count",
    color="arrival_runway_actual",
    barmode="group",
)

app.layout = html.Div(children=[
    html.H1(children=f'{AIRPORT_ICAO} Arrival Runway Utilization'),

    html.Div(f"Runway utilization from {START_TIME} thru {END_TIME}"),

    dcc.Graph(
        id='arr-rwy-util',
        figure=fig,
    )
])

if __name__ == '__main__':
    app.run_server(debug=True)
```

# 4. API Requests Using Postman

Assumptions:

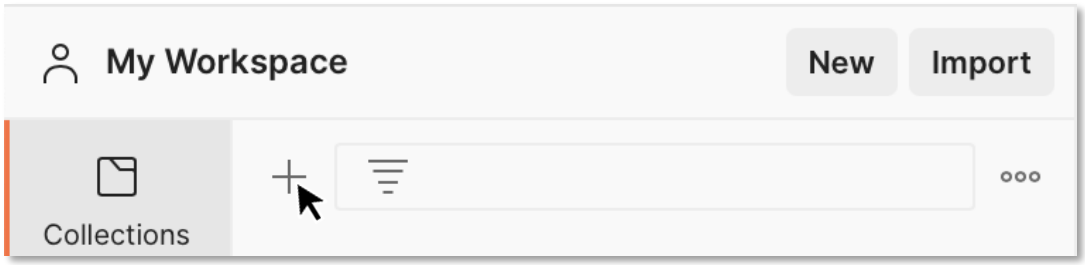
- Free account with www.postman.com
- Access to the Service Connect information on the Platform (see earlier section)

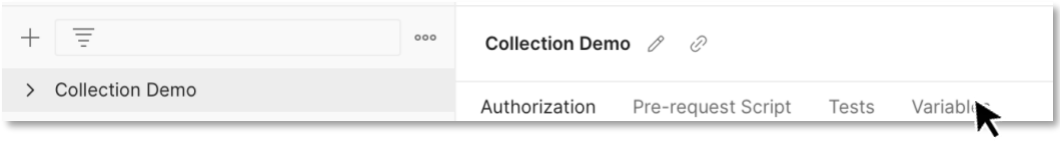
Postman provides the advantage of managing all APIs inside a collection. This section steps through the creation of a collection of 2 services: one that obtains an authorization token and another one that queries one of the DIP NASA services. The request for the token applies then to any DIP API services that would be added to the collection. Postman supports various set-ups. The approach below is one of many.

Note: Postman exists in both a web and desktop versions. The examples below were based on the web interface of Postman, and therefore there may be minor differences with the desktop version.

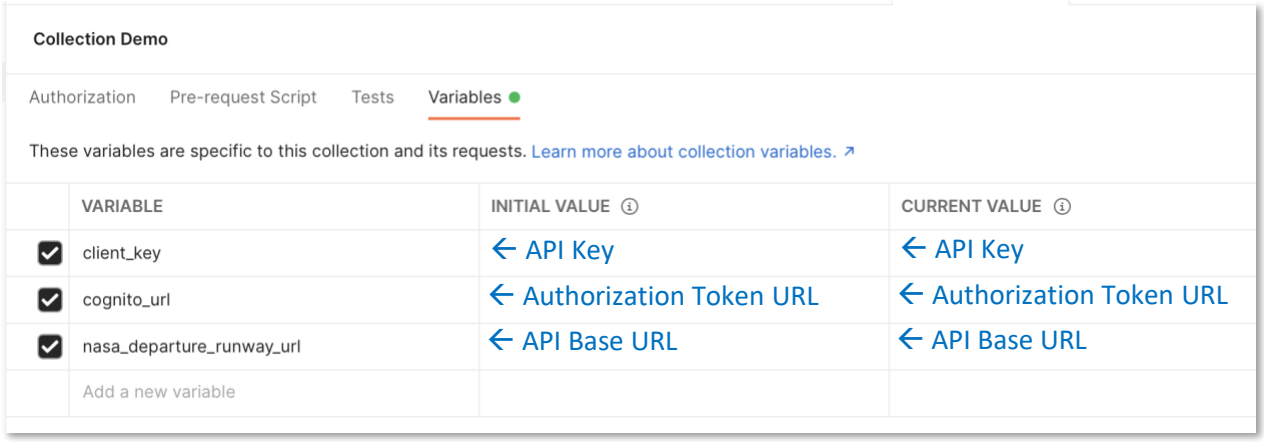
## Create a Collection

1. Click on the " + " sign on the left hand side of the workspace, and name the collection (eg. "Collection Demo")



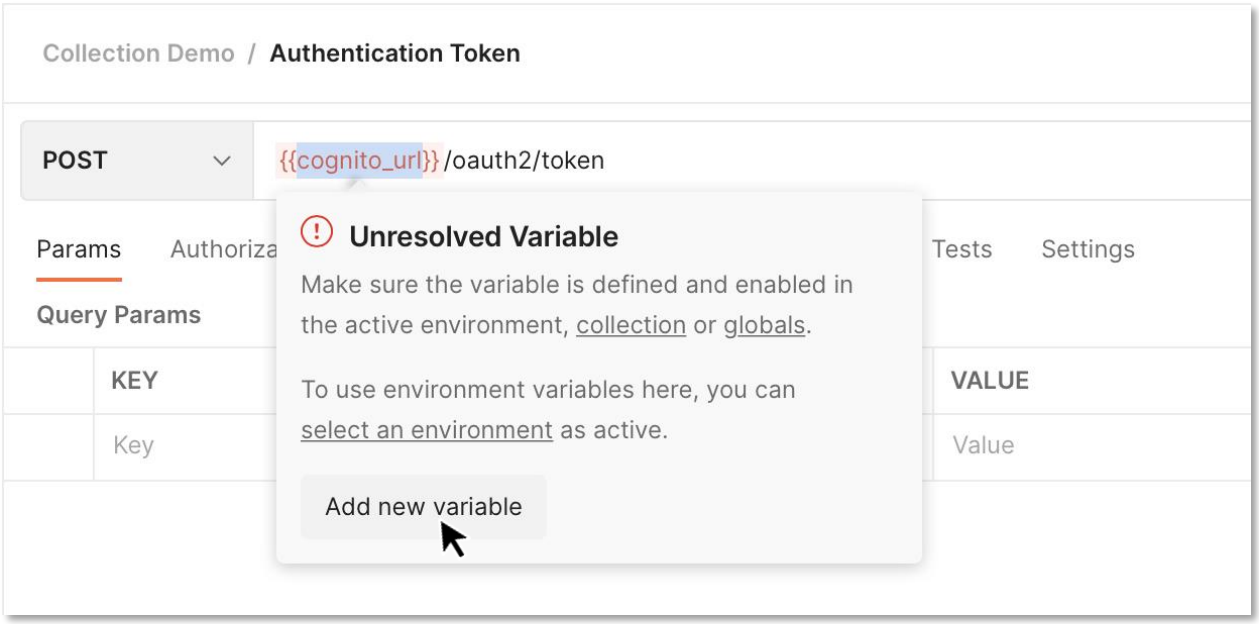

2. Click on the "Variables" tab. Variables enables storing values that can then be referenced throughout collections, environments, requests, and test scripts. Three variables are required at a minimum. The first 2 to handle the authentication, and at least one for one of the DIP services. The values associated with the variables are found in the Connect Window. Ensure the initial and current values are identical. Set the following variables names and values from the Connect window:
    - "client_key" as variable with the API Key as value
    - "cognito_url" as variable with the Authorization token URL as value
    - "nasa_departure_runway_url" as variable with the API Base URL as value



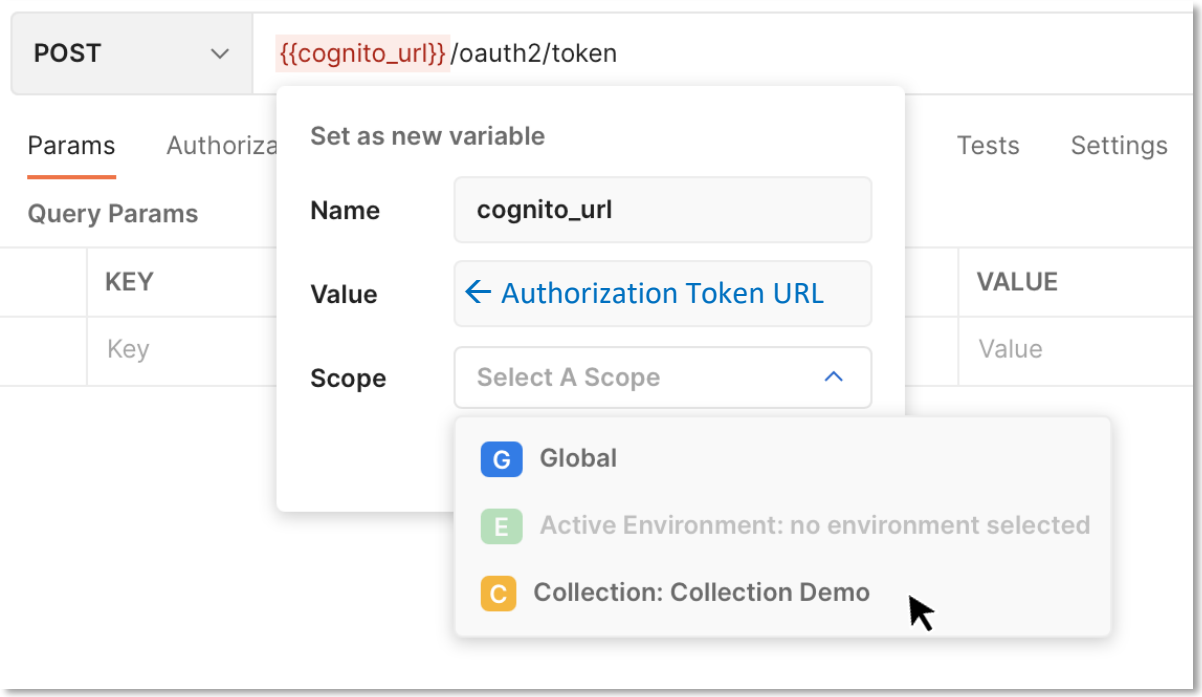

## Add An Authentication Token Request

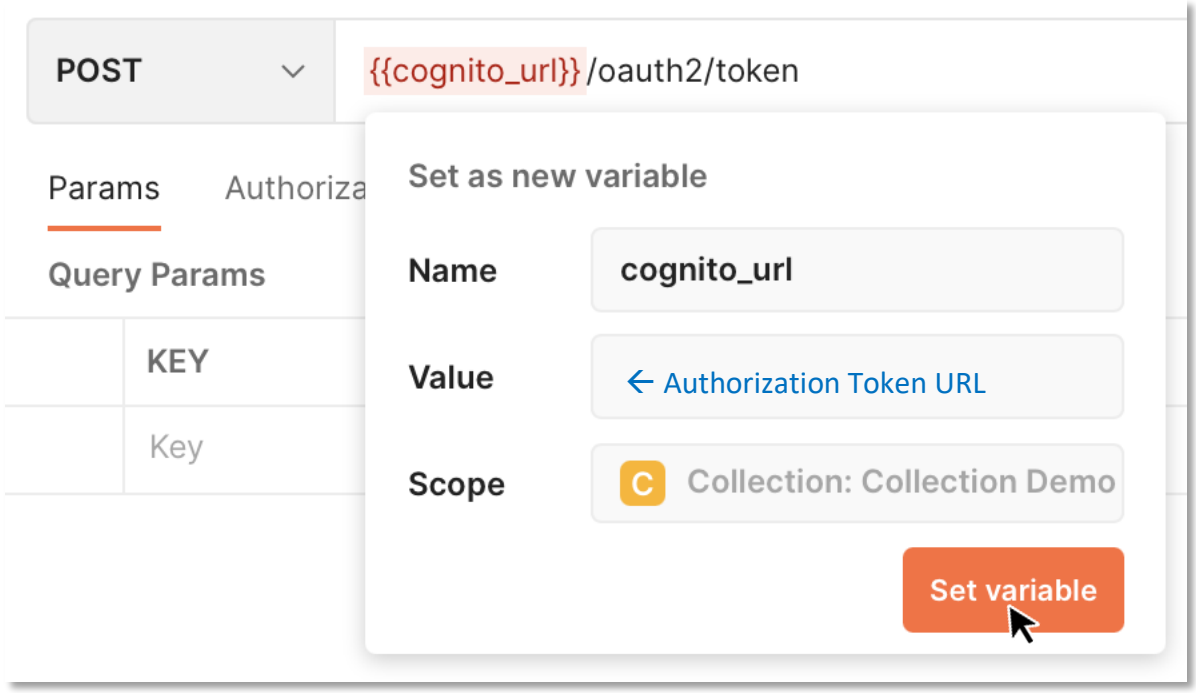
1. Click on the " > " sign, click on the "Add a request" link. Give a name related to the authorization token request (eg. Authorization Token).

2. Select "POST" as the query method and type in {{cognito_url}}/oauth2/token in the request URL field. Note that in lieu of an actual URL address, "cognito_url" refers to a value associated to a variable in the Collection environment. The linkage needs to be established.

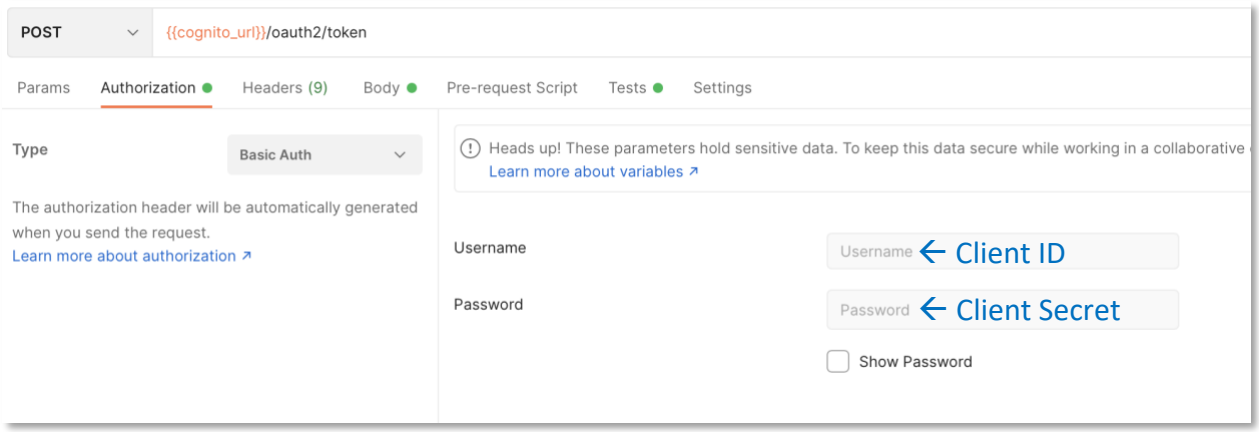



3. Hover over "{{cognito_url}}" to open the warning window. Click on "Add a new variable" and copy the Authorization Token URL from the Connect Window in the value field. Next, select the scope to match the Collection name. Click on "Set Variable." This request URL is now linked to the authentication token URL.

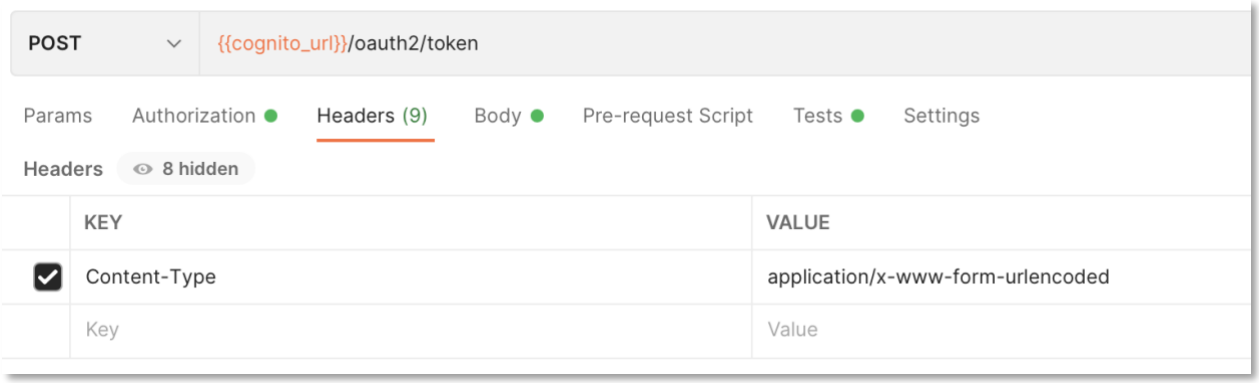


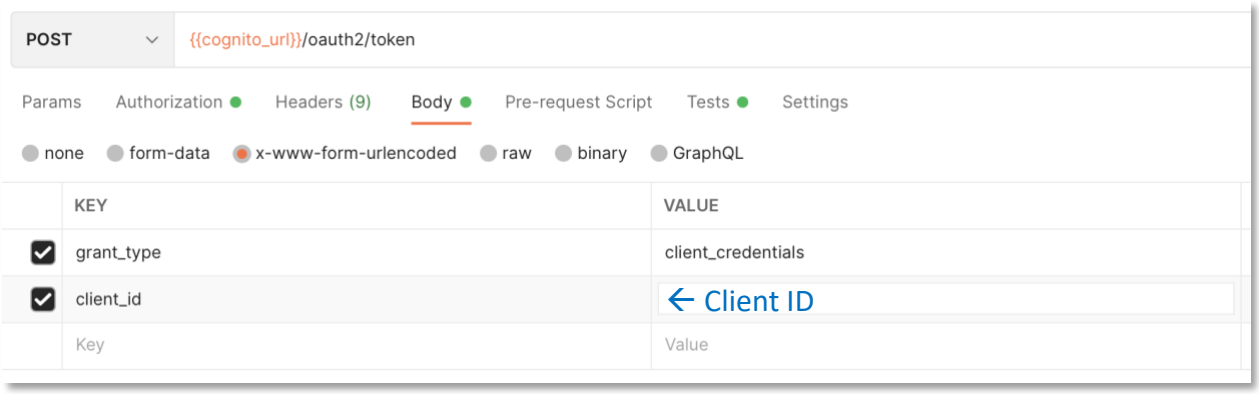

4. Click on the “Authorization” tab and enter the Client ID in the Username field and the Client Secret in the password field.




5. Click on the header tab and type in “Content-Type” in the key field and “application/x-www-form-urlencoded” in the value field

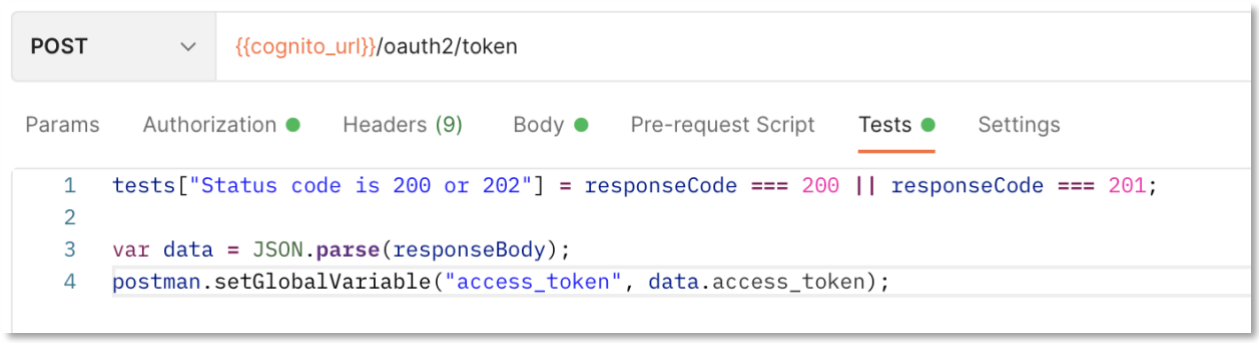



6. Click on the Body tab, select the radio button “x-www-form-urlencoded” and add 2 keys:
   - “grant_type” as key and “client_credentials” as value.
   - “client_id” as key and the Client ID as value.




7. Click on the “Tests” tab and type in the following script to link the access token with other requests.

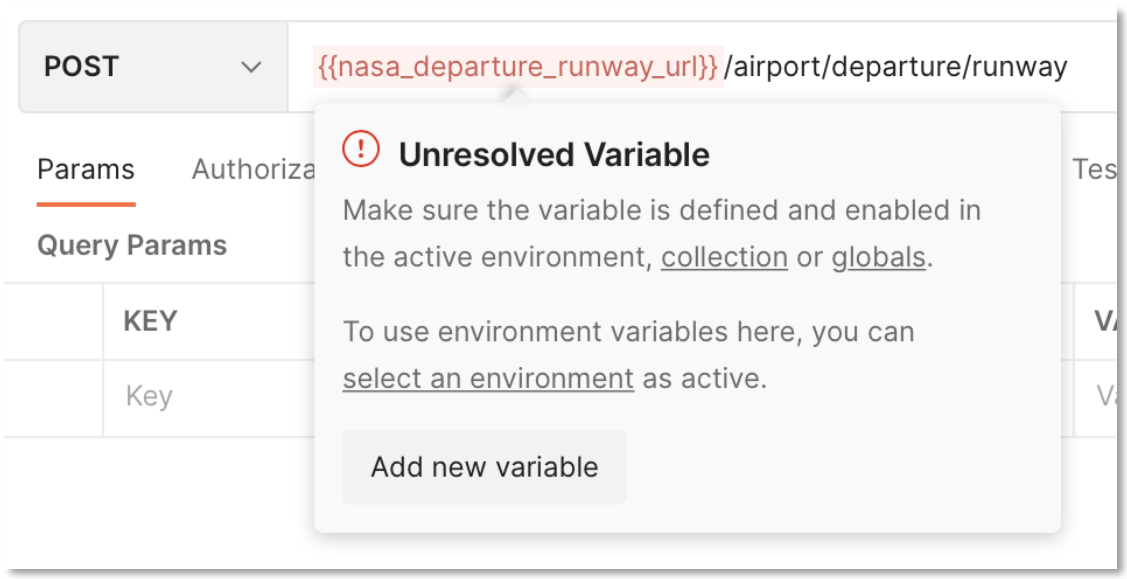
```
tests["Status code is 200 or 202"] = responseCode === 200 || responseCode === 201;

var data = JSON.parse(responseBody);
postman.setGlobalVariable("access_token", data.access_token);
```



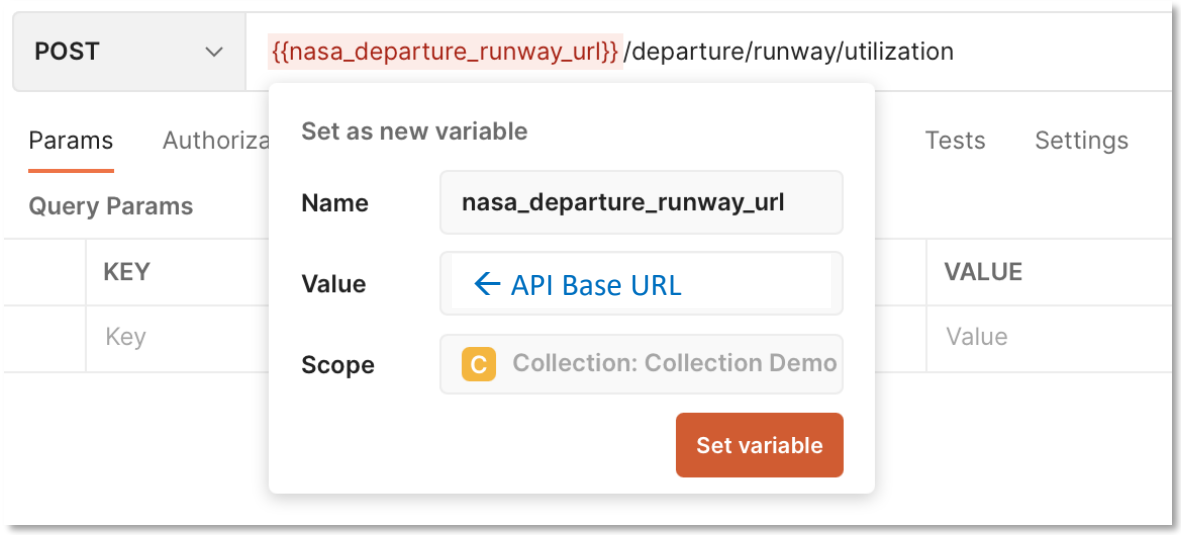

## Add A Service Request

1. Right-click on the Collection name and select “Add request”.

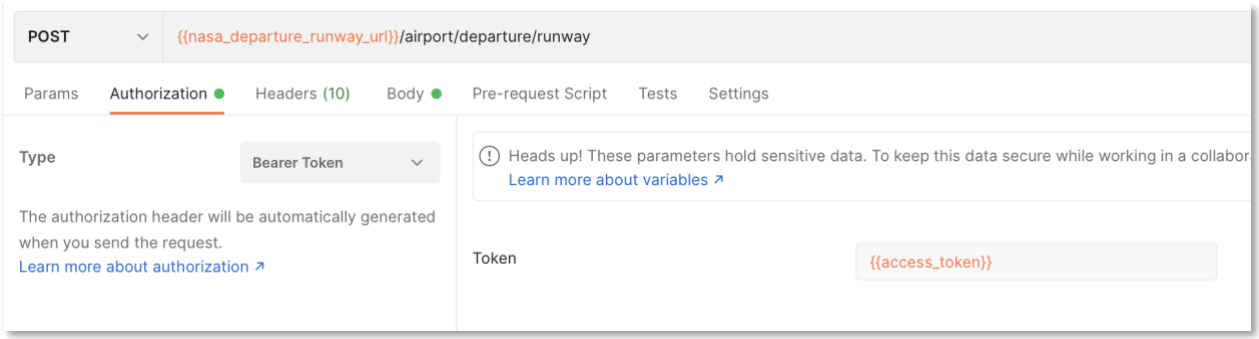
2. Give a name to the Request. Then in the request URL field, type in the variable set in the collection, and the subfolders as indicated in the OpenAPI Page. For example: “{{nasa_departure_runway_url}}/airpot/departure/runway”.



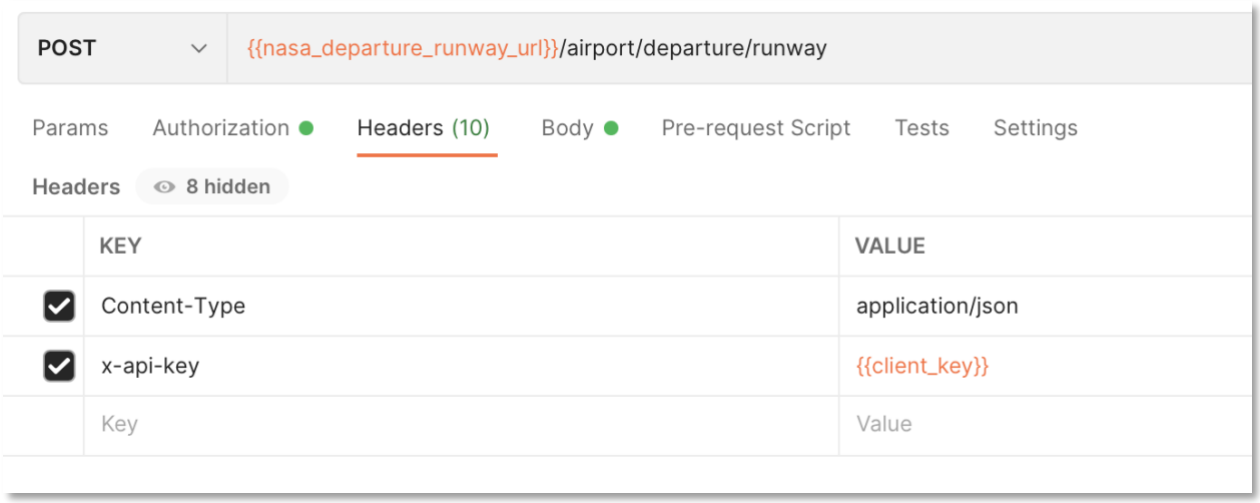

3. Set the request method to “POST”. Similar to the set-up for the previous Authentication Token request, a variable needs to be associated with {{nasa_departure_runway_url}}. Open the warning window, copy the API Base URL listed in the Collection’s variables (or in the Connect Window), select the scope to the collection name and click on “Set Variable.”



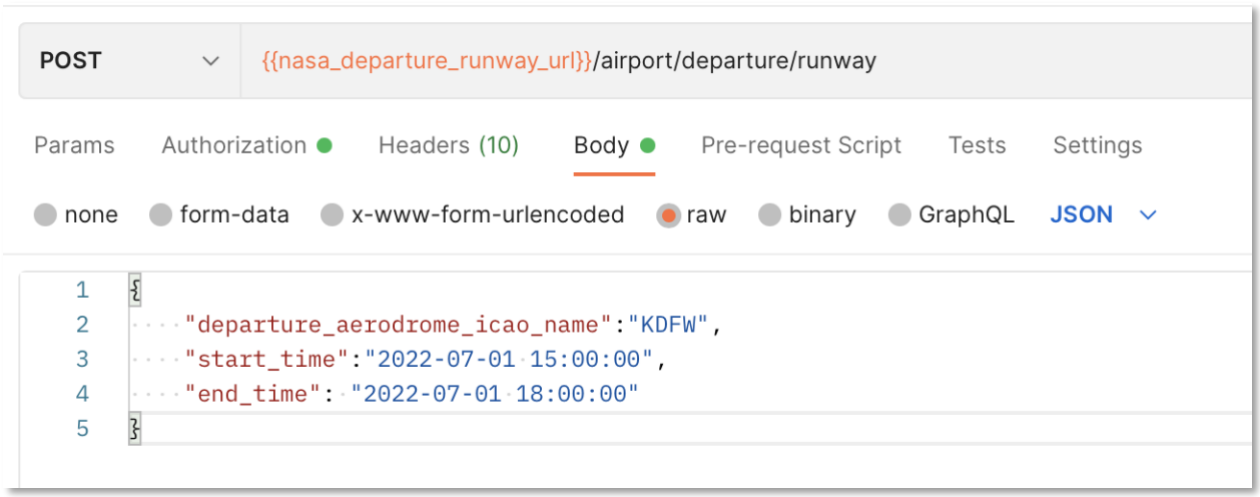

4. Click on the Authorization tab. Select “Bearer Token” as the type of authorization. Type in the variable {{access_token}} in the Token field. This variable refers to the authorization token request created above.




5. Click on the headers tab, add 2 keys with the following values:
   - “Content-Type” as the key and “application/JSON” in the value field
   - “x-api-key” as the key and {{client_key}} in the value field

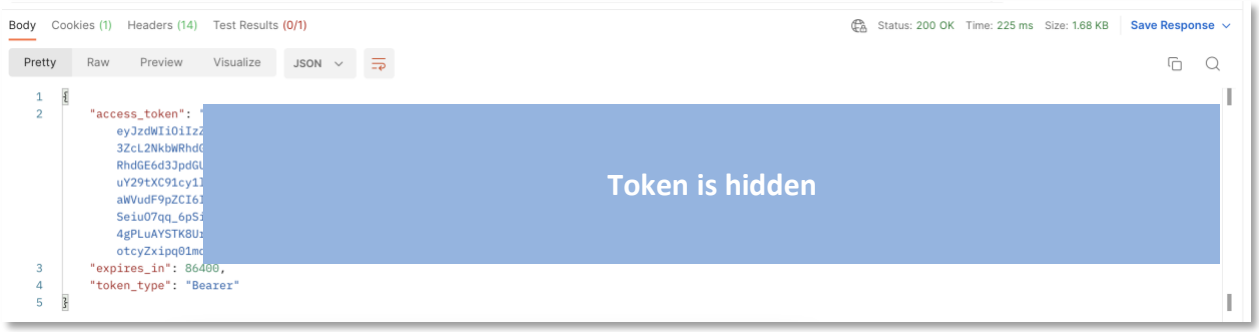



6. Click on the Body tab to add the request post body. Click on the "raw" radio button. An example of a request can be found in the Open API page of the service. In this example, the query would be as indicated on the picture below.




7. Save both requests and the collection. The set-up is now complete.

8. Send the request to obtain the authorization token. The token is displayed in the lower "Body" section of Postman. As indicated in the response, the token is valid for 24h only. Send a new request to renew the token.




9. Send the request to obtain data from the example service. Save the response as either an example or a file.