

# Azure Data Factory – Using an Azure Function for more Complex Interaction with an API

## Overview

Using an Azure Function for more Complex Interaction with an API

## Setting up an Azure Function to Pull from GA API

### Prerequisites

During setup in Lab 0, you should have installed:

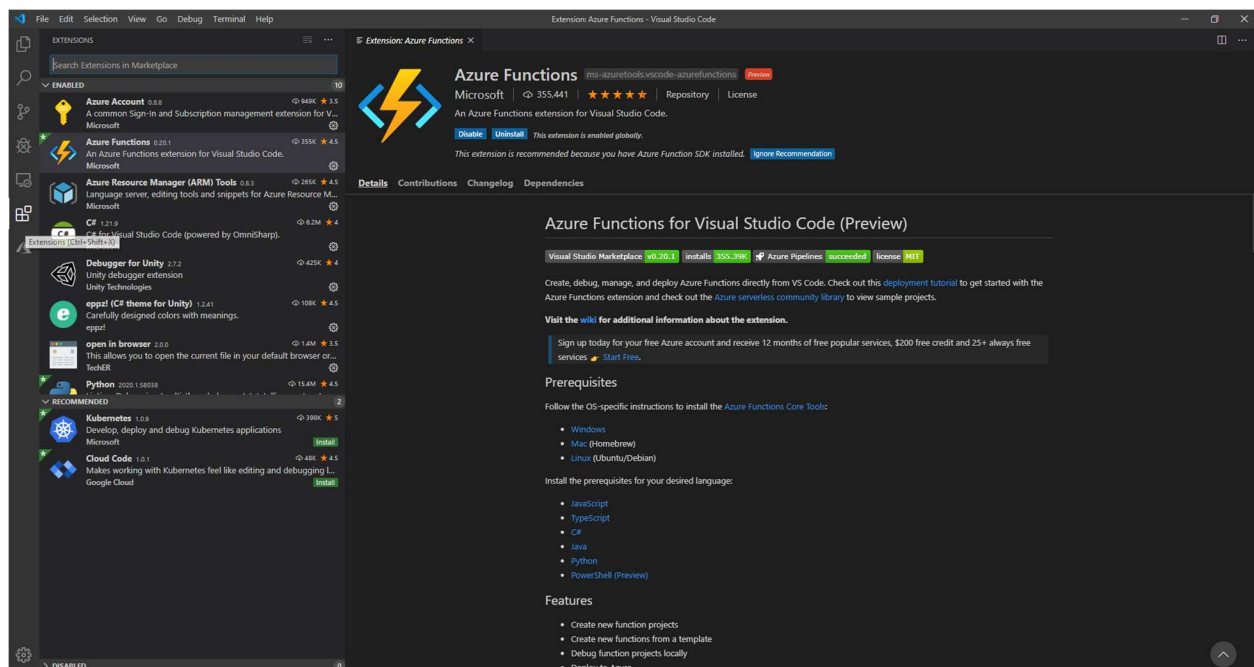
Visual Studio Code (<https://code.visualstudio.com/>),

Node.js (<https://nodejs.org/en/download/>),

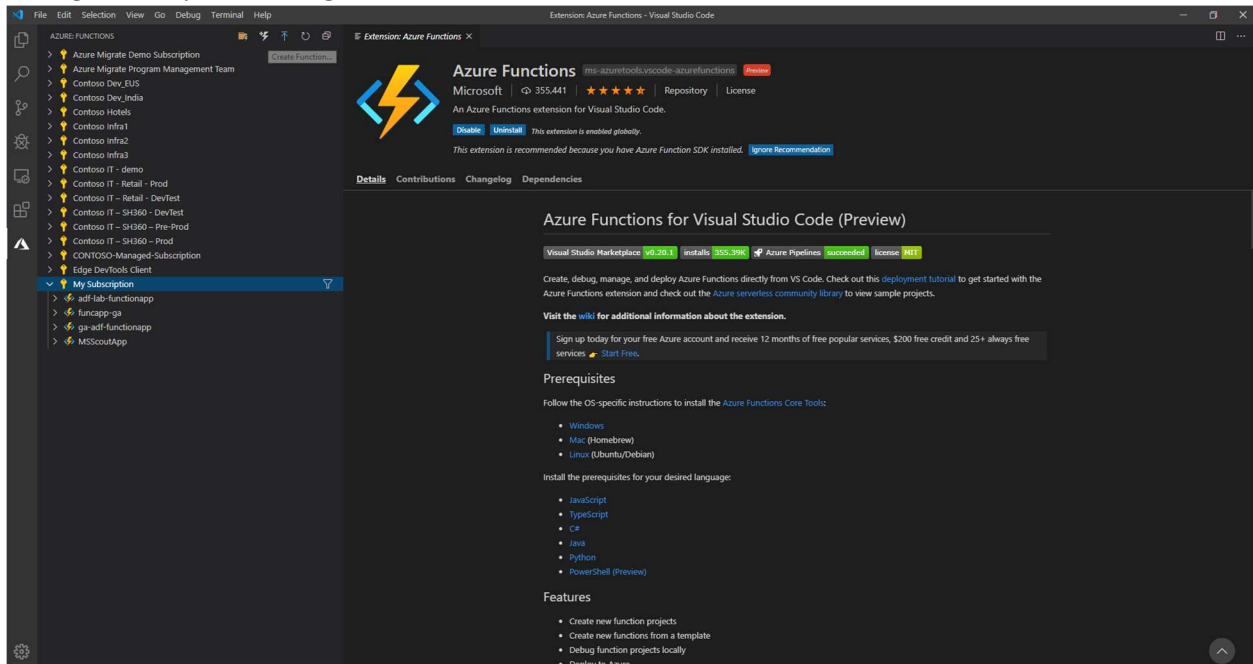
Azure Functions Core Tools (<https://docs.microsoft.com/en-us/azure/azure-functions/functions-run-local?tabs=windows#install-the-azure-functions-core-tools>)

Connect Visual Studio Code to your Azure Account

Open Visual Studio Code and in the Extensions tab on the right hand panel, search for and install the Azure Functions Extension.



Once installed (you may need to reload Visual Studio Code) you will see a new Azure Functions tab in the right hand panel. Navigate to it.



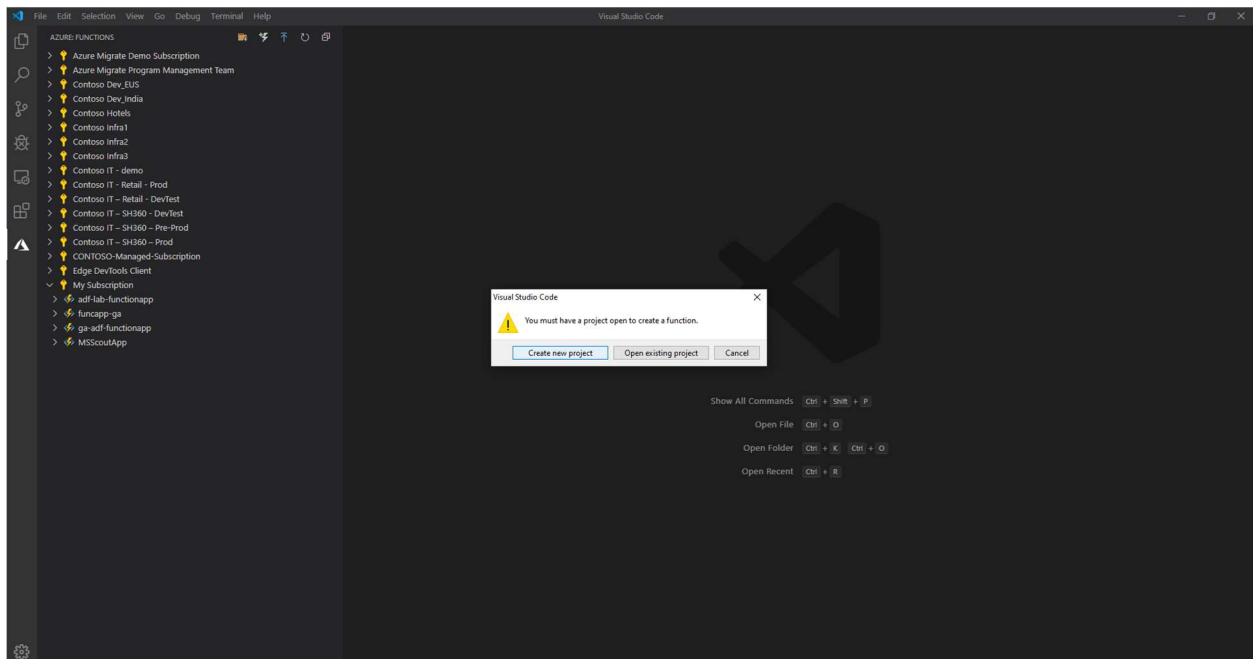
In the Azure Functions tab you will be prompted to connect your Azure account and be able to sign in. Once signed in, you will be able to see your subscriptions listed.

## Create your Function

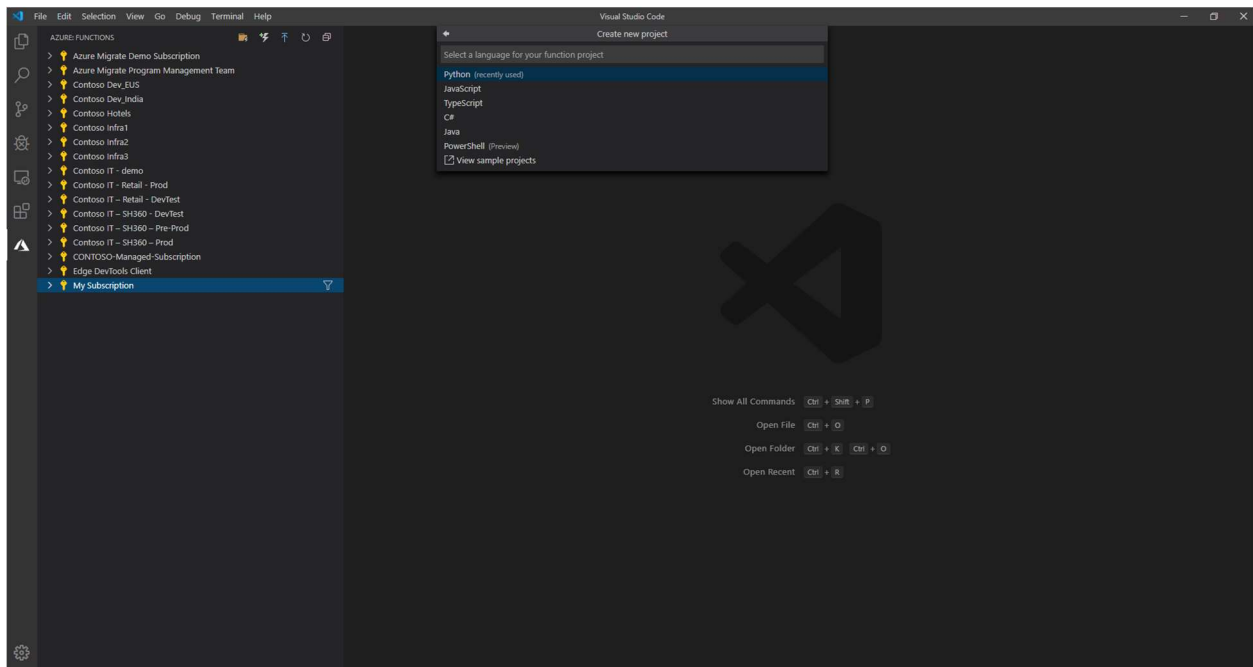
At the top of the Azure Functions panel click the Create Function button



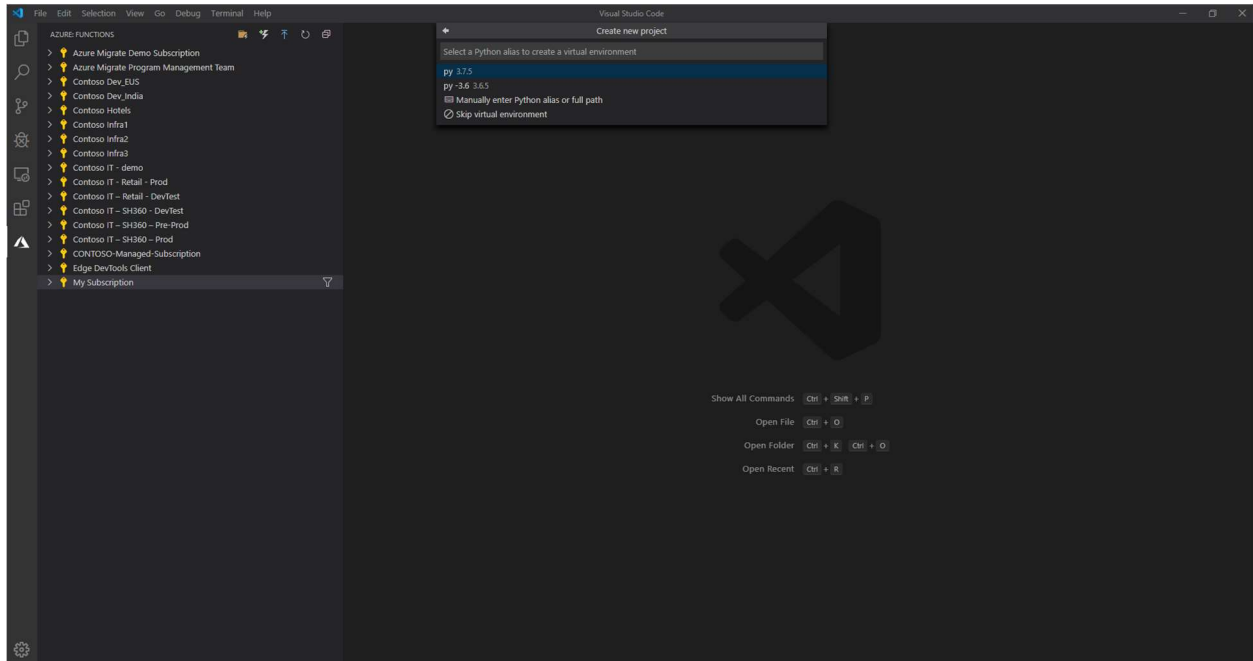
When prompted, select Create new project.



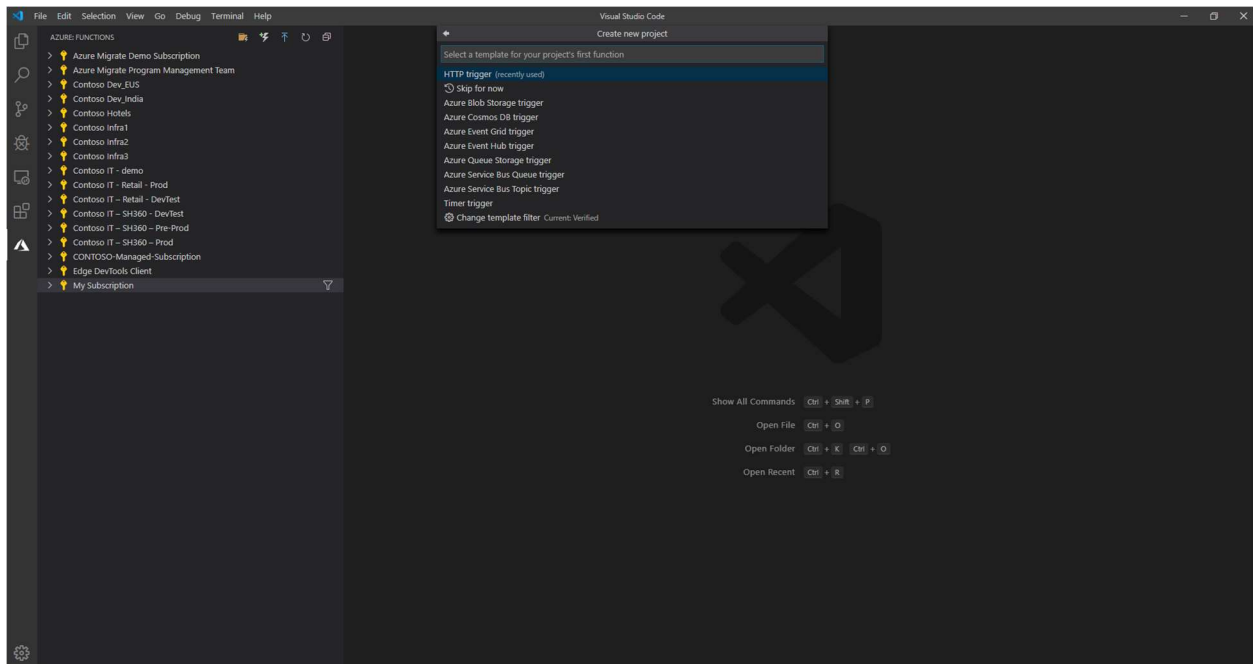
Choose a name for the folder (eg. Azure-Data-Factory-REST-Lab) your project will reside in and create it. When prompted, select Python as the language for your function project.



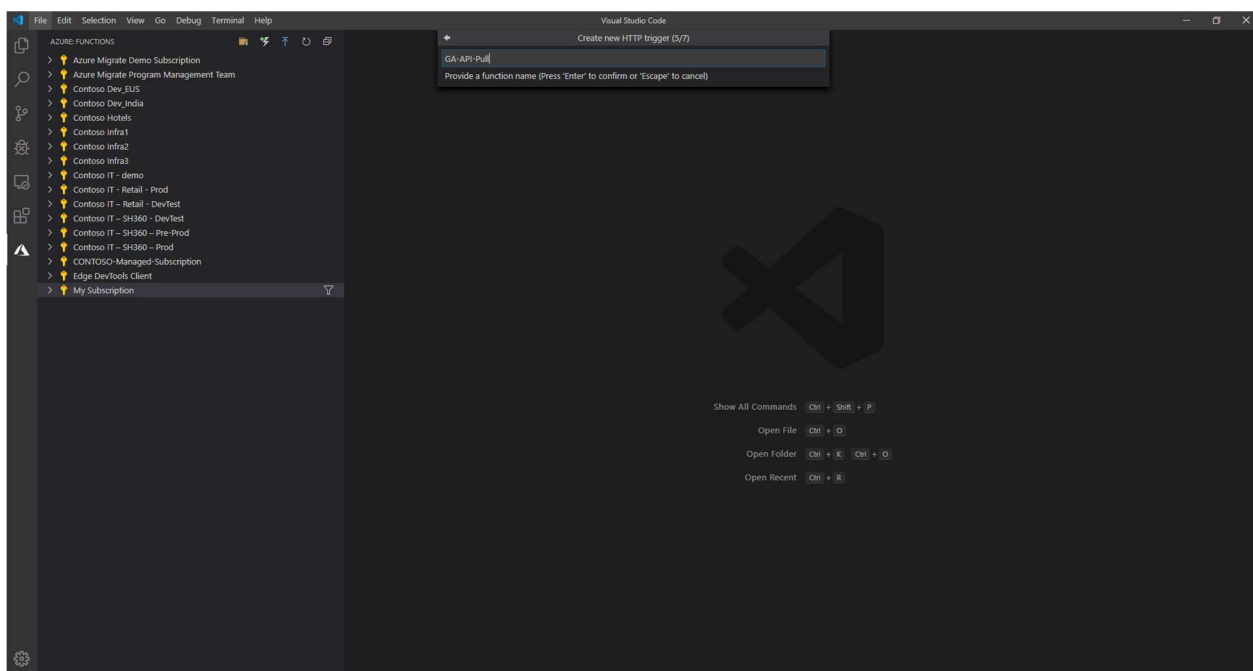
If prompted to select a Python Alias, choose 3.7.5



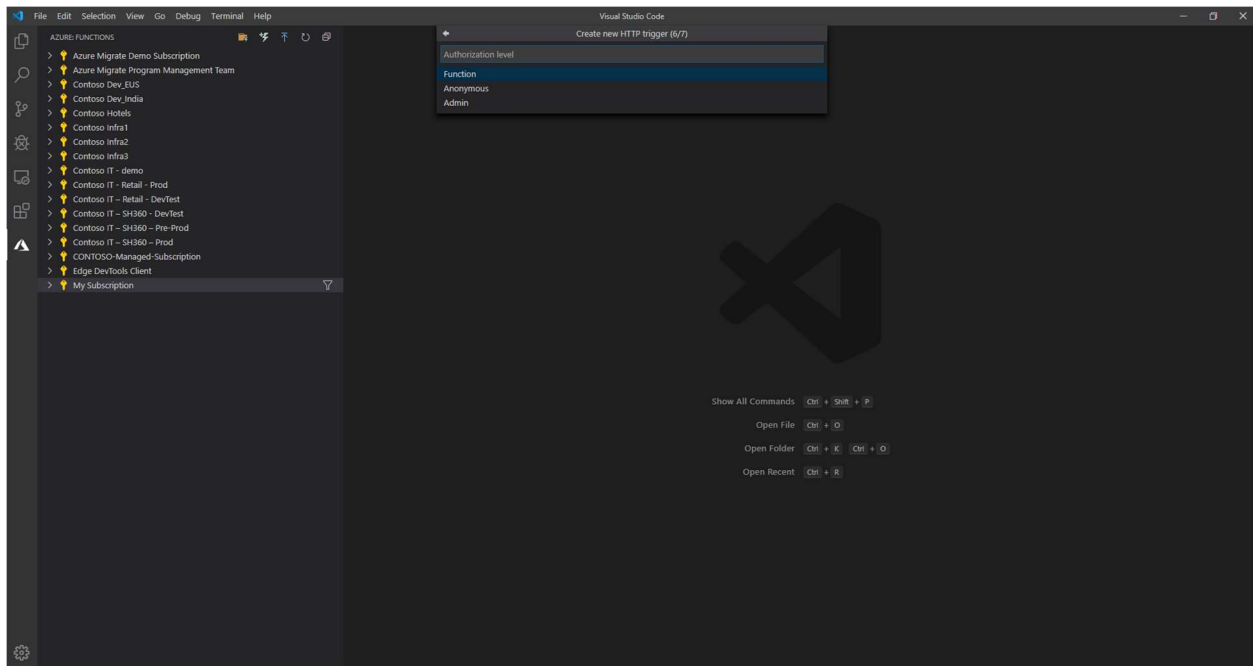
Next, for the template selection, choose to use the HTTP trigger.



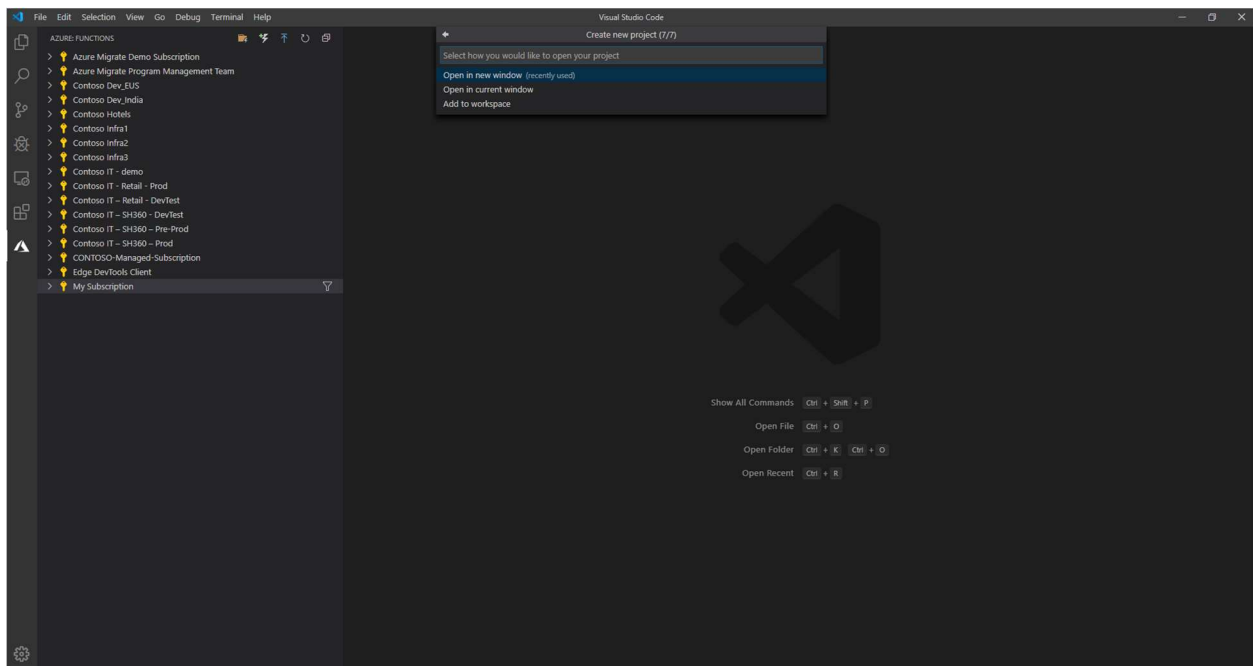
Next select a name for your Function (eg. GA-API-Pull)



For Authorization level, select Function

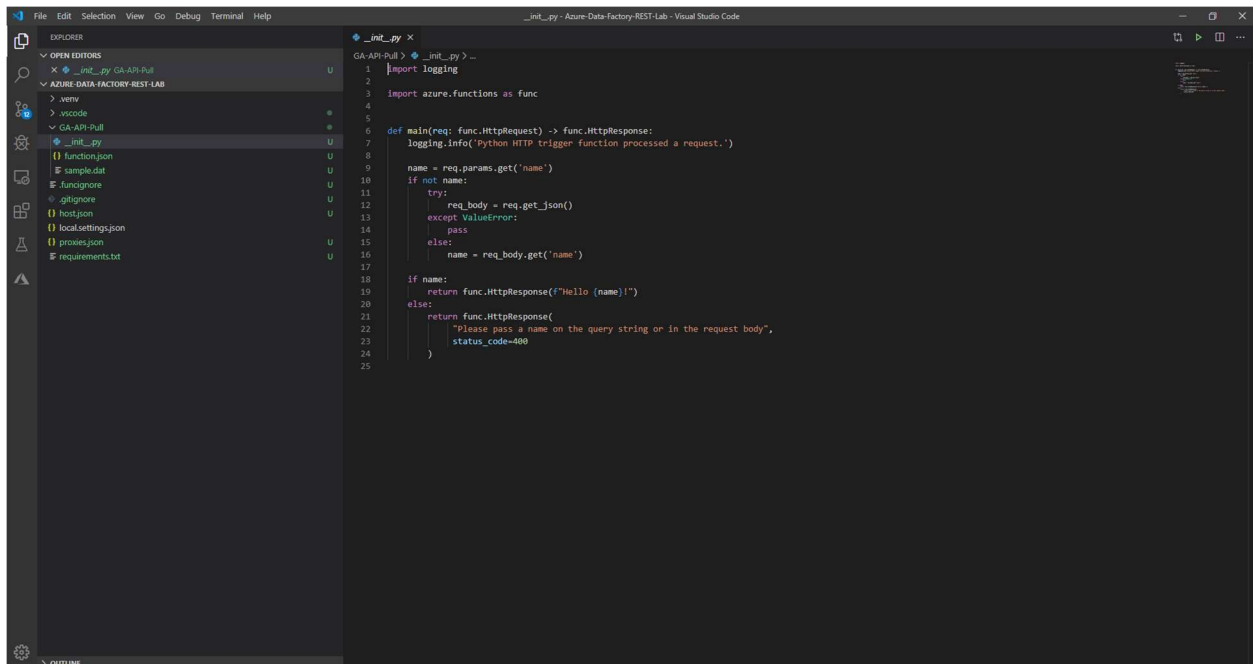


Finally, choose to open your project in your current window.



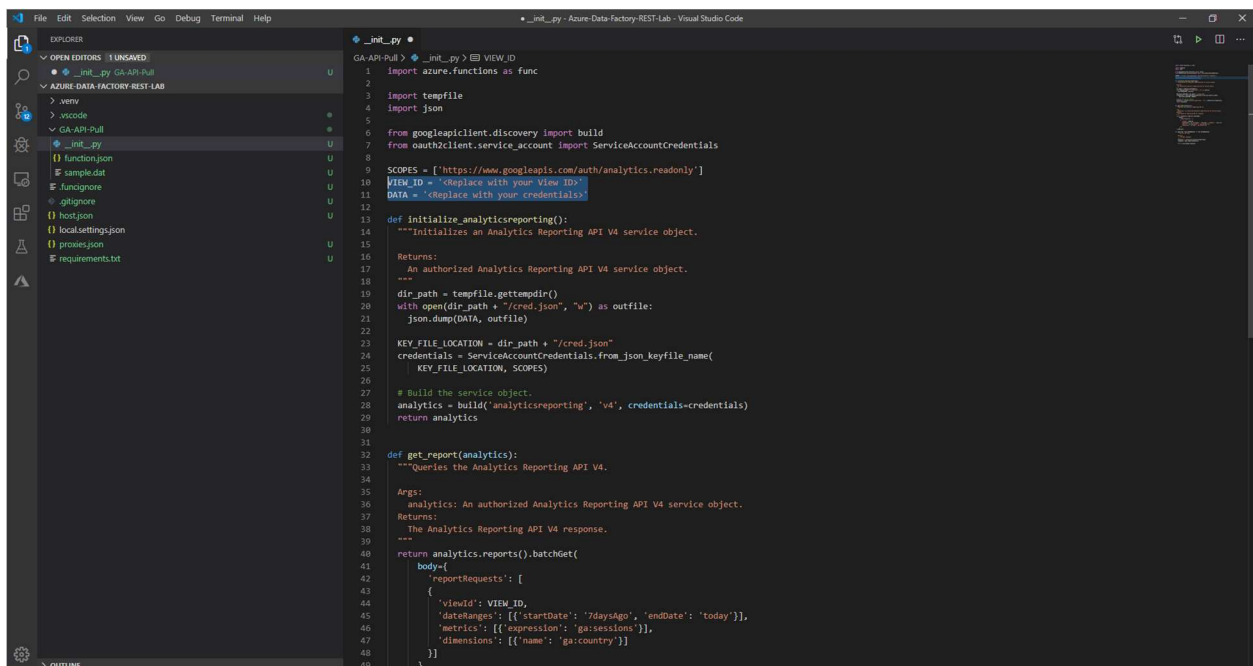
Add your custom code from the lab's github repository.

Navigate to your newly created project's '\_\_init\_\_.py' file.



```
1 import logging
2
3 import azure.functions as func
4
5
6 def main(req: func.HttpRequest) -> func.HttpResponse:
7     logging.info('Python HTTP trigger function processed a request.')
8
9     name = req.params.get('name')
10    if not name:
11        try:
12            req_body = req.get_json()
13        except ValueError:
14            pass
15        else:
16            name = req_body.get('name')
17
18    if name:
19        return func.HttpResponse(f'Hello {name}!')
20    else:
21        return func.HttpResponse(
22            'Please pass a name on the query string or in the request body.',
23            status_code=400
24        )
25
```

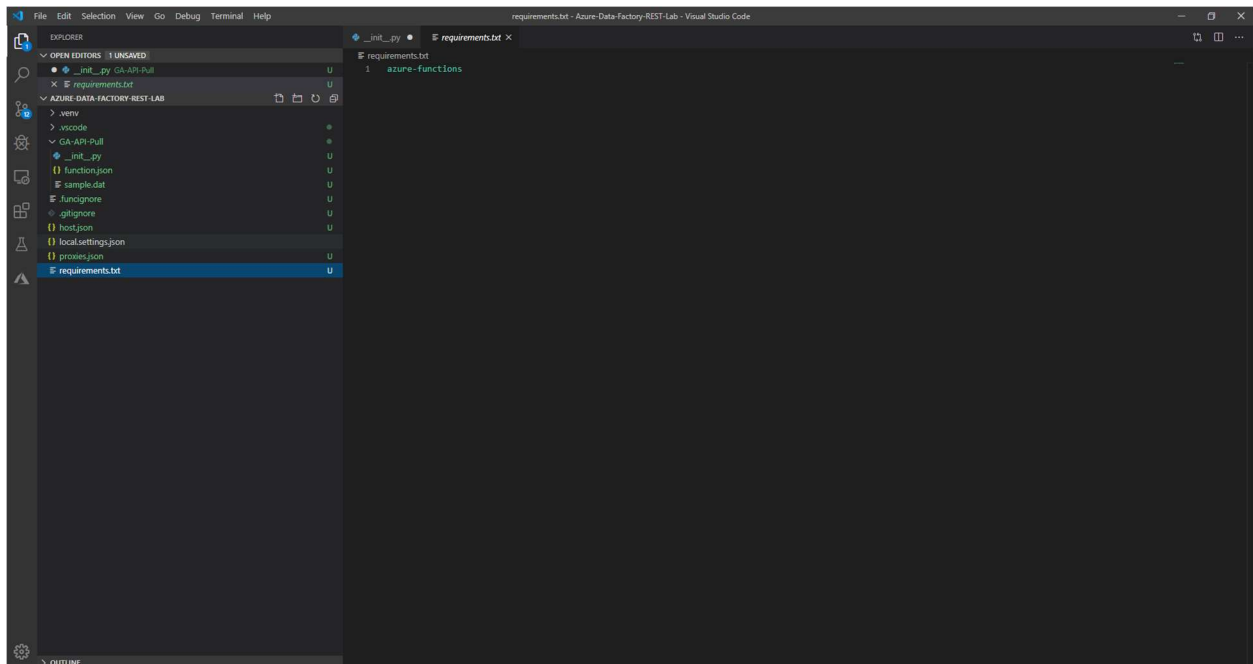
Copy and paste the code from <https://raw.githubusercontent.com/Gryczka/Azure-Data-Factory-REST-Lab/master/src/GA-Pull-Function.py> and replace the code currently in your `__init__.py` file.




```
1 import logging
2
3 import azure.functions as func
4
5 import tempfile
6 import json
7
8 from googleapiclient.discovery import build
9 from oauth2client.service_account import ServiceAccountCredentials
10
11 SCOPES = ['https://www.googleapis.com/auth/analytics.readonly']
12 VIEW_ID = 'Replace with your View ID'
13 DATA = 'Replace with your credentials'
14
15 def initialize_analyticsreporting():
16     """Initializes an Analytics Reporting API V4 service object.
17
18     Returns:
19         An authorized Analytics Reporting API V4 service object.
20     """
21     dir_path = tempfile.gettempdir()
22     with open(dir_path + '/cred.json', "w") as outfile:
23         json.dump(DATA, outfile)
24
25     KEY_FILE_LOCATION = dir_path + "/cred.json"
26     credentials = ServiceAccountCredentials.from_json_keyfile_name(
27         KEY_FILE_LOCATION, SCOPES)
28
29     # Build the service object.
30     analytics = build('analyticsreporting', 'v4', credentials=credentials)
31     return analytics
32
33 def get_report(analytics):
34     """Queries the Analytics Reporting API V4.
35
36     Args:
37         analytics: An authorized Analytics Reporting API V4 service object.
38     Returns:
39         The Analytics Reporting API V4 response.
40     """
41     return analytics.reports().batchGet(
42         body={
43             'reportRequests': [
44                 {
45                     'viewId': VIEW_ID,
46                     'dateRanges': [{'startDate': '7daysAgo', 'endDate': 'today'}],
47                     'metrics': [{'expression': 'ga:sessions'}],
48                     'dimensions': [{'name': 'ga:country'}]}
49             ]
50         })
51
```

Your proctors will provide you with a `VIEW_ID` and `DATA` to add to the script.

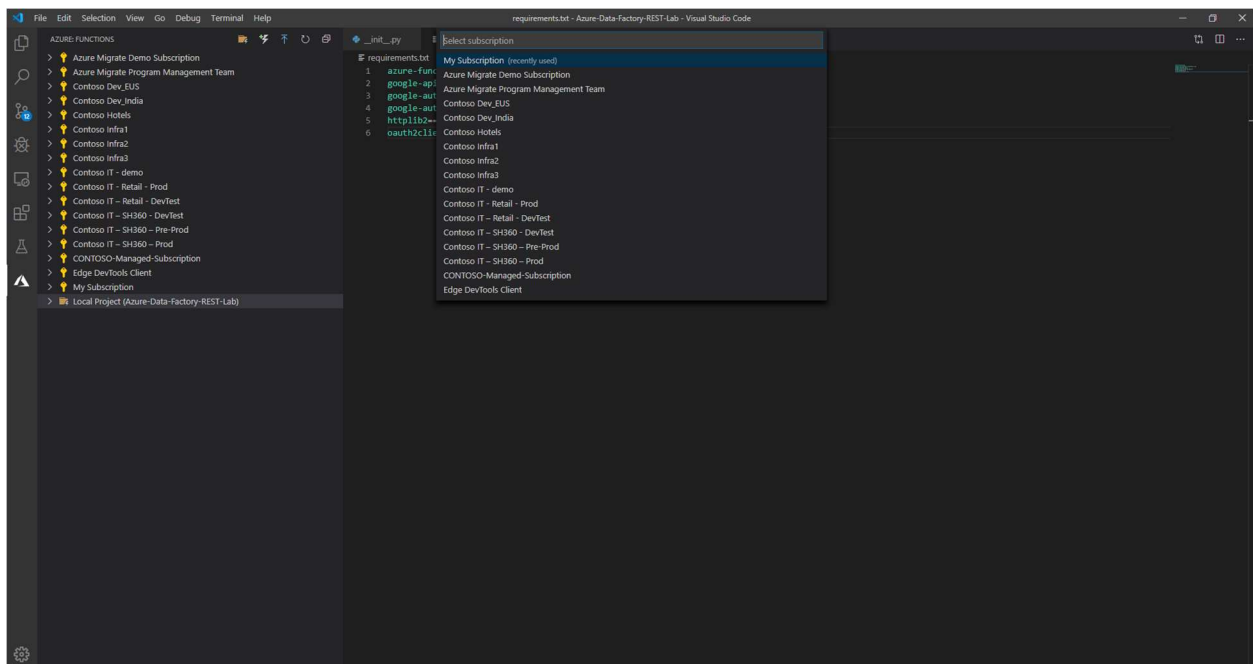
Next, navigate to your `requirements.txt` file.



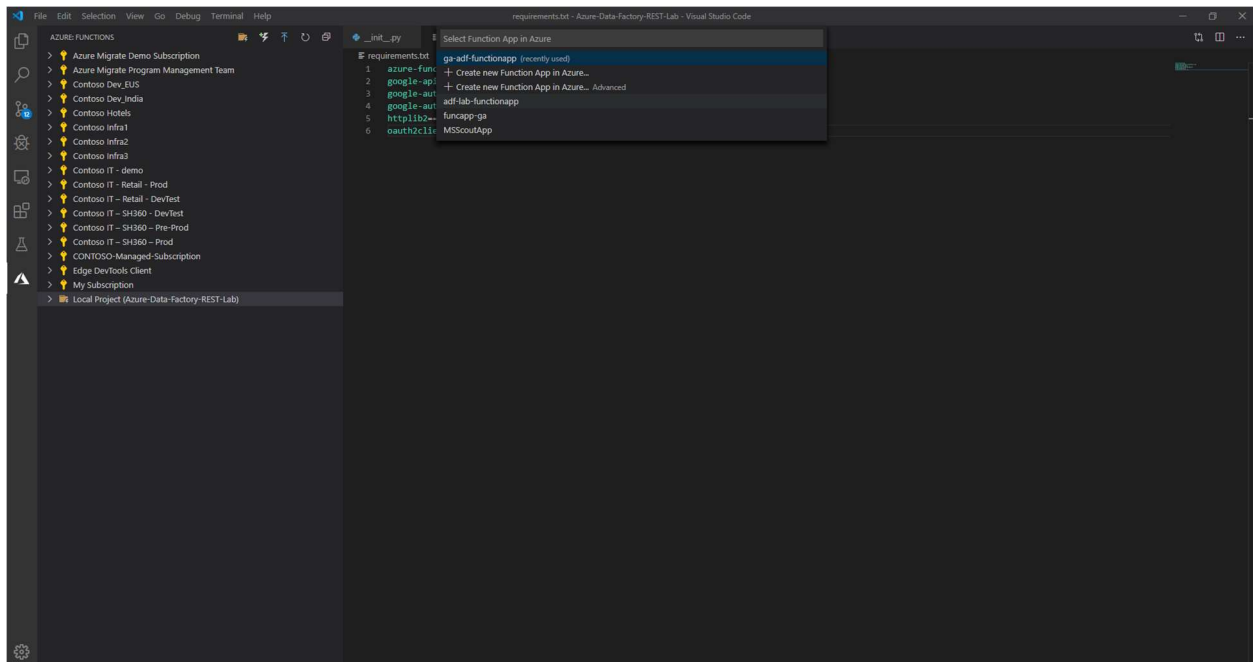
Add the rest of the requirements from <https://raw.githubusercontent.com/Gryczka/Azure-Data-Factory-REST-Lab/master/src/requirements.txt> and save both files.

Next navigate back to your Azure Functions panel in VS Code, and click the Deploy to Function App button .

You will be prompted to select your subscription.



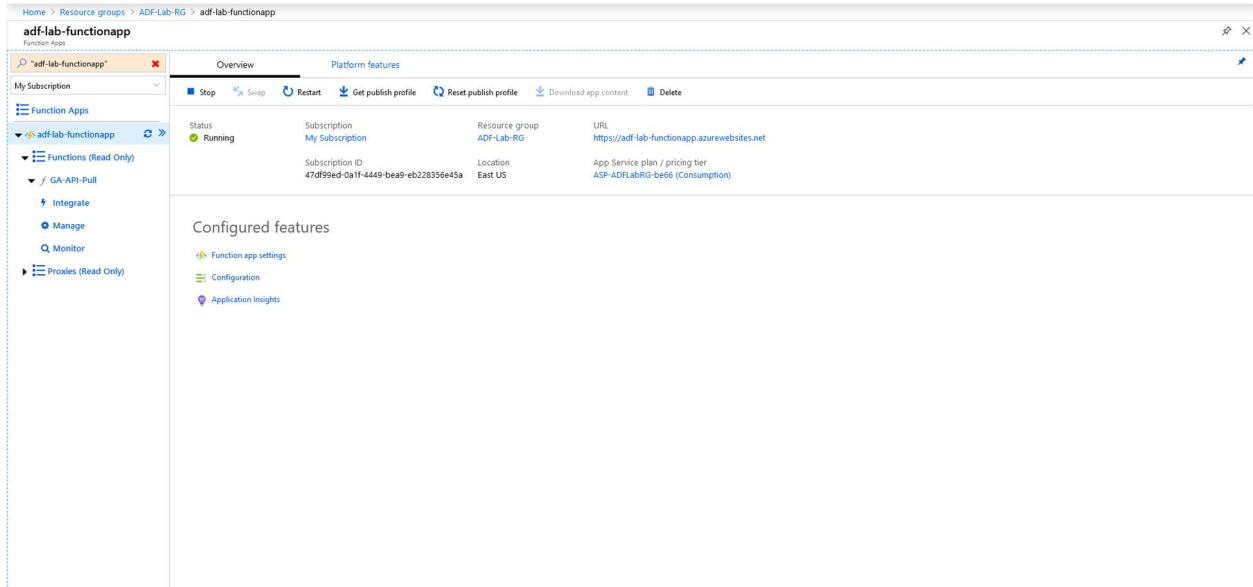
Then select the Function App you created during Lab 0 (adf-lab-functionapp)



When prompted, confirm that you would like to deploy.

Test your Function in the Portal

In the Azure Portal, navigate to your adf-lab-functionapp





Click through to your GA-API-Pull Function, and on the right side Run a Test

The screenshot shows the Azure Functions portal for the 'adf-lab-functionapp - GA-API-Pull' function. The left sidebar shows the navigation menu with 'GA-API-Pull' selected. The main area displays the Python code for the function, which is an HTTP-triggered function that pulls data from Google Analytics. The code includes imports for Azure Functions, tempfile, json, and Google Analytics client libraries. It defines a function that builds a service object and returns it. The right sidebar shows the 'Test' tab with a 'Run' button and a 'Test' button. Below the code, there is a 'Logs' section with a message: 'Navigate to the "Monitor" section for this function in order to view your live app metrics and logs.'

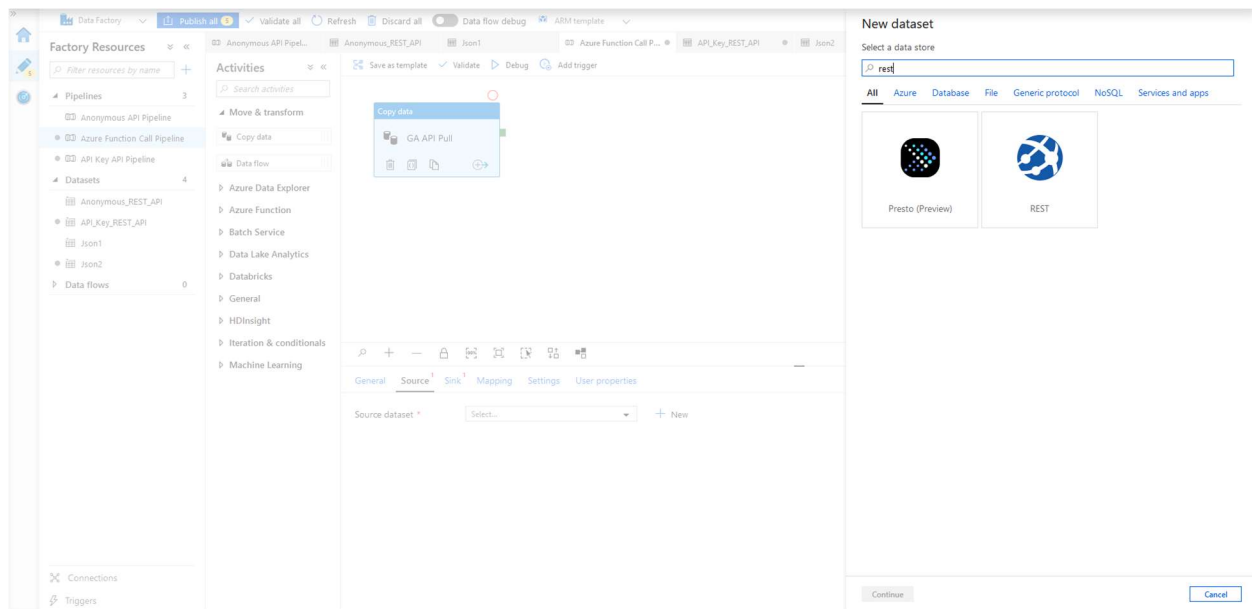
If you received an Output and a **Status: 200 OK** response, your Function is ready.

## Calling the Function from Data Factory

Create a new pipeline to call your Azure Function and set up an activity to Copy Data

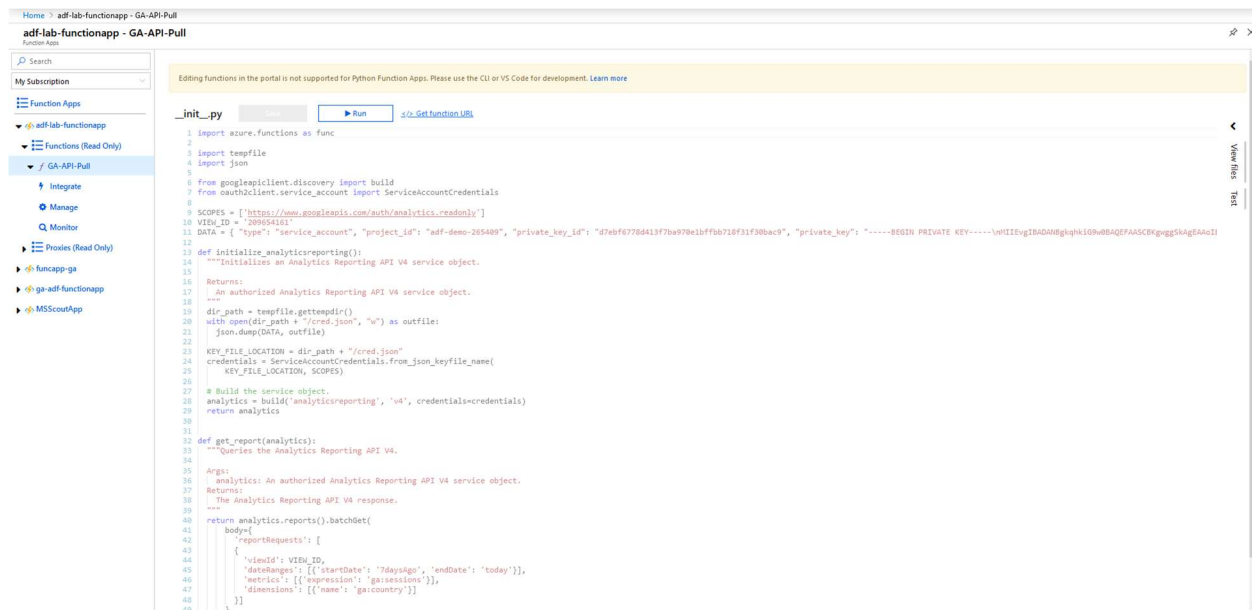
The screenshot shows the Azure Data Factory portal. The left sidebar shows the 'Factory Resources' section with 'Pipelines' expanded. A new pipeline named 'Azure Function Call Pipeline' is being created. The main area shows the 'Activities' section with a 'Copy data' activity selected. The 'Copy data' activity is configured with 'Source' and 'Sink' tabs. The 'Source' tab is active, showing a 'Source dataset' dropdown menu. The 'Sink' tab is also visible, showing a 'Sink' dropdown menu. The 'Mapping' tab is also visible, showing a 'Mapping' dropdown menu. The 'Settings' tab is also visible, showing a 'Settings' dropdown menu. The 'User properties' tab is also visible, showing a 'User properties' dropdown menu.


Create a Source dataset for the activity, again choosing REST

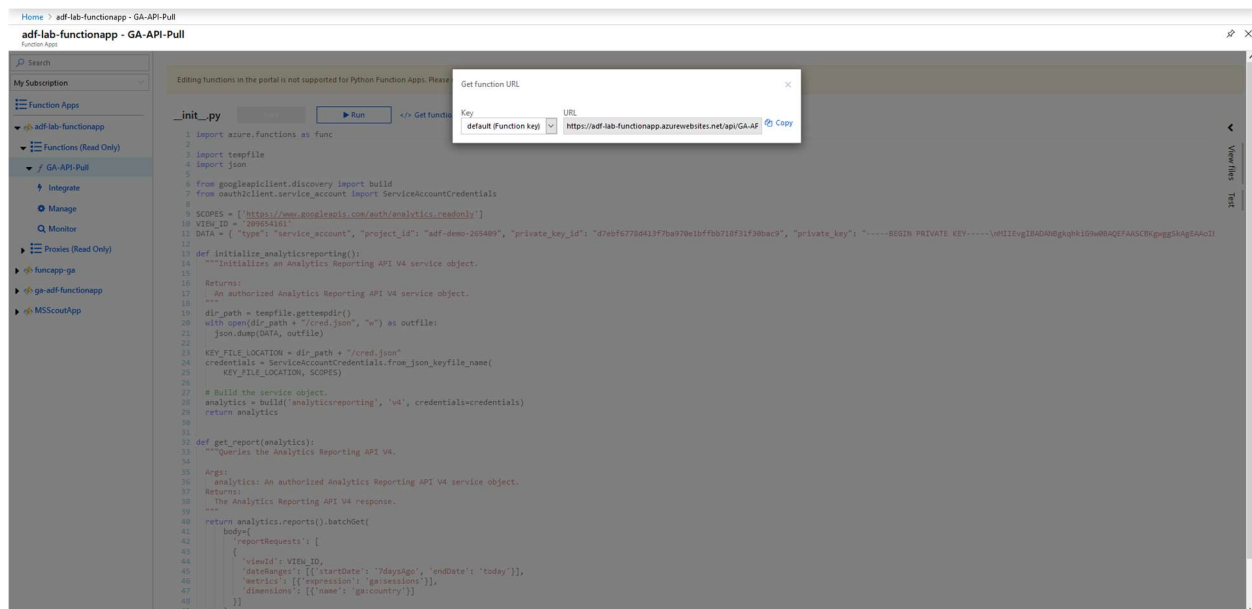


Next open the created rest resource and Create a New Linked Service

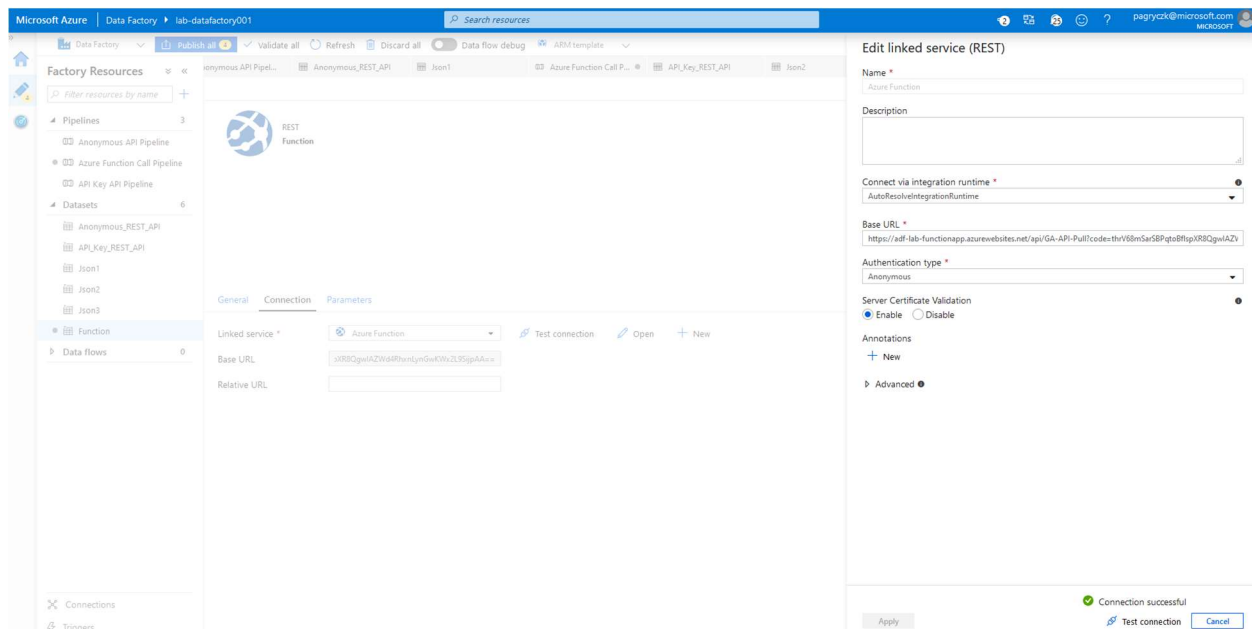
Next, open your Azure Function App in the Azure portal in another tab, and navigate to your function and click on [Get function URL](#)



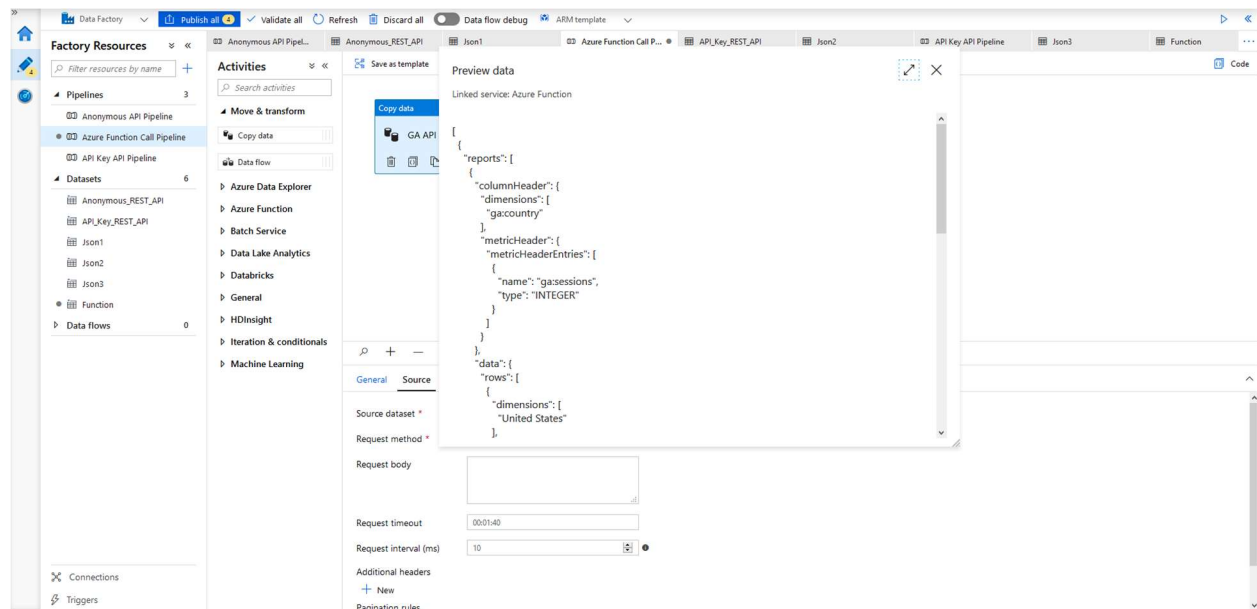
On the popup, click  Copy



Now returning to your Linked Service, paste the copied url in as the Base URL with Anonymous Authentication selected, and test your connection before creating.



At this point you'll be able to preview your data in the copy activity to confirm it is connected.



Finally create a JSON sink to your Data Lake Gen 2 as you did for the APIs in Lab 1.