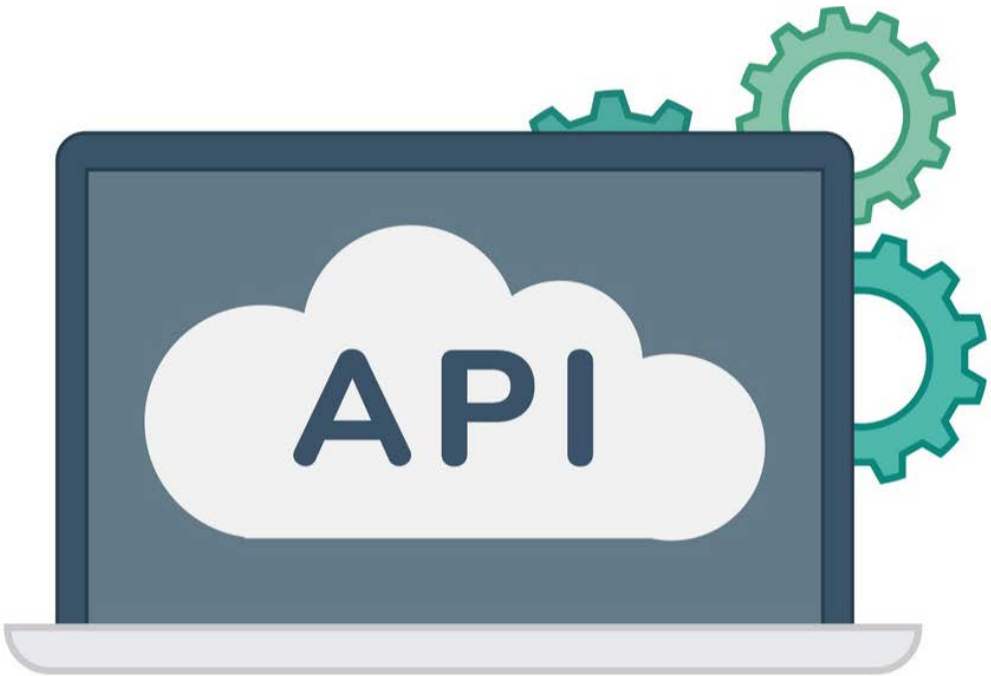
How to Pull Data from an API using Python Requests

Technical API how-to without the headache

Cameron Warren

Follow

Mar 18 · 9 min read



Automatically retrieving data from APIs is critical.

The thing that I'm asked to do over and over again is automate pulling data from an API. Despite holding the title "Data Scientist" I'm on a small team, so I'm not only responsible for building models, but also pulling data, cleaning it, and pushing and pulling it wherever it needs to go. Many of you are probably in the same boat.

When I first began my journey for learning how to make HTTP requests, pull back a JSON string, parse it, and then push it into a database I had a very hard time finding clear, concise articles explaining how to actually do this very important task. If you've ever gone down a Google black hole to resolve a technical problem, you've probably discovered that very technical people like to use technical language in order to explain how to perform the given task. The problem with that is, if you're self-taught, as I am, you not only have to learn how to do the task you've been asked to do but also learn a new technical language. This can be incredibly frustrating.

If you want to avoid learning technical jargon and just get straight to the point, you've come to the right place. In this article, I'm going to show you how to pull data from an API and then automate the task to re-pull every 24 hours. For this example, I will utilize the Microsoft Graph API and demonstrate how to pull text from emails. I will refrain from using pre-prepared API packages and rely on HTTP requests using the Python Requests package — this way you can apply what you learn here to nearly any other RESTful API that you'd need to work on.

If you're having trouble with API requests in this tutorial, here's a tip: Use Postman. Postman is a fantastic app that allows you to set up and make API calls through a clean interface. The beauty of it is once you get the API call working, you can export the code in Python and then paste it right into your script. It's amazing.

*Note: This tutorial is meant to be a simple and easy to understand method to access an API, that's it. It will likely not be robust to your exact situation or data needs, but should hopefully set you down the right path. I've found the below method to be the simplest to understand to quickly get to pulling data. If you have suggestions to make things even easier, sound off in the comments.

API Documentation

There's no way around navigating API documentation. For working with Outlook, you go here. If you are successful in following this tutorial and want to do more with the API, use that link for a reference.

Authorization

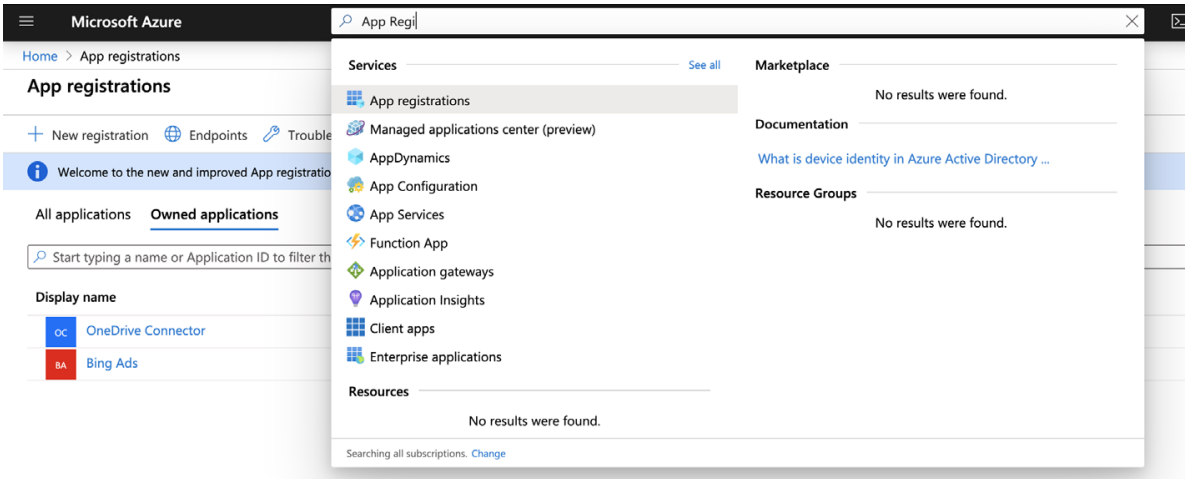
Most APIs, including the Microsoft Graph, require authorization before they'll give you an access token that will allow you to call the API. This is often the most difficult part for someone new to pulling data from APIs (it was for me).

Register your App

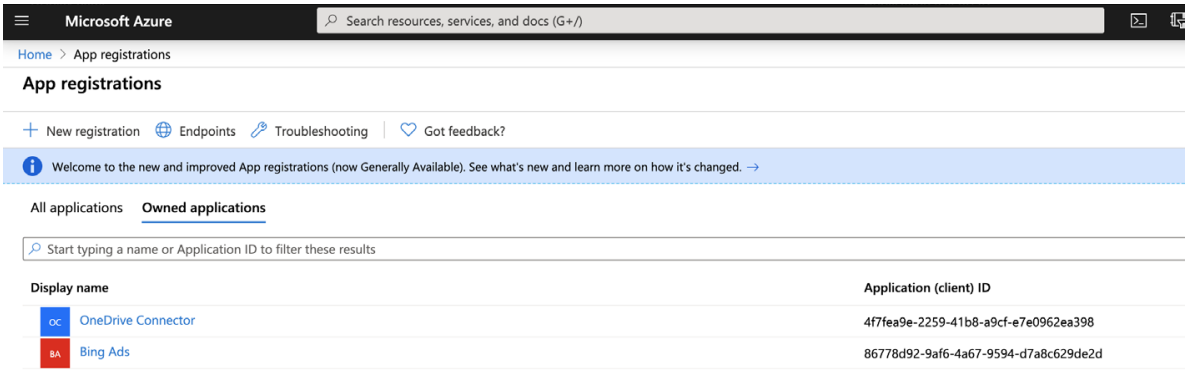
Follow the instructions here to register your app (these are pretty straightforward and use the Azure GUI) You'll need a school, work, or personal Microsoft account to register. If you use a work email, you'll need admin access to your Azure instance. If you're doing this for work, contact your IT department and they'll likely be able to register the app for you. Select 'web' as your app type and use a trusted URL for your redirect URI (one that only you have access to the underlying web data. I used *franklinsports.com* as only my team can access the web data for that site. If you were building an app, you would redirect to your app to a place where your app could pick up the code and authorize the user).

Enable Microsft Graph Permissions

Once your app is registered, go to the App Registration portal:

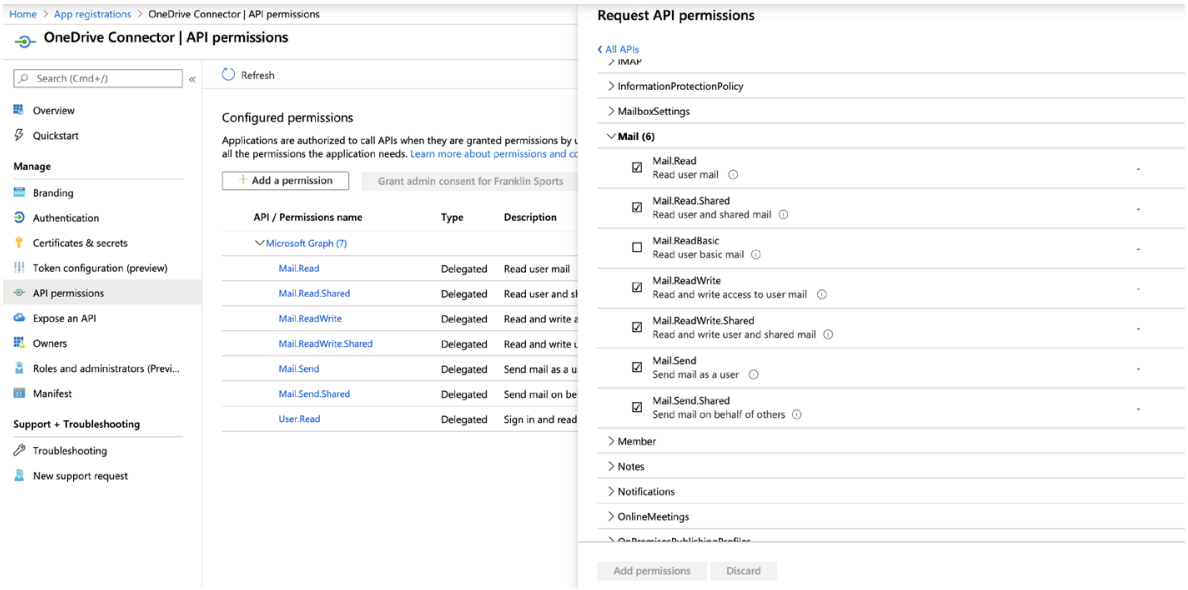


Click where it displays the name of your app:



On the following page, click on ‘View API Permissions’.

Finally, select the Microsoft Graph and then check all the boxes under Messages for ‘Delegated Permissions.’ (Note for this example I’ll only be pulling emails from a single signed-in user, to pull email data for an organization you’ll need application permissions — follow the flow here).



Ok — you're ready to start making API calls.

Authorization Step 1: Get an access code

You'll need an access code to get the access token that will allow you to make calls to the MS Graph API. To do this, enter the following URL into your browser, swapping the appropriate credentials with the information shown on your app page.

Goes in the URL:

Take the URL that outputs from the code snippet and paste it into your browser. Hit enter, and you'll be asked to authorize your Microsoft account for delegated access. Use the Microsoft account that you used to register your app.

If the authorization doesn't work, there's something wrong with the redirect URL you've chosen or with the app registration. I've had trouble with this part before — so don't fret if it doesn't work on your first try. If the authorization is successful, you'll be redirected to the URL that you used in the App registration. Appended to the URL will be parameters that contain the code you'll use in our next step — it should look like this:

Take the full code and save it in your python script.

Authorization Step 2: Use your access code to get a refresh token.

Before proceeding, make sure you have the latest version of the Python Requests package installed. The simplest way to do that is by using pip:

In this step, you will take the code generated in step 1 and send a **POST** request to the MS Graph OAuth authorization endpoint in order to acquire a refresh token. (*Note: For security purposes, I recommend storing your client_secret on a different script than the one you use to call the API. You can store the secret in a variable and then import it (or by using pickle, which you'll see below).

Swap the ####### in the code snippet with your own app information and the code you received in step 1, and then run it. (*Note: The redirect URI must be url encoded which means it'll look like this: "https%3A%2F%2Ffranklinsports.com")

You should receive a JSON response that looks something like this:

{'access_token':'########################','refresh_token':'############################'}

Authorization Step 3: Use your refresh token to get an access token

At this point you already have an access token and could begin calling the API, however, that access token will expire after a set amount of time. Therefore we want to set up our script to acquire a fresh access token each time we run it so that our automation will not break. I like to store the new refresh_token from step 2 in a pickle object, and then load it in to get the fresh access_token when the script is run:

```
filename = ####path to the name of your pickle file#######
print(filename)
filehandler = open(filename, 'rb')
```

```
refresh_token = pickle.load(filehandler)

url = “https://login.microsoftonline.com/common/oauth2/v2.0/token"

payload = “””client_id%0A=##########&
redirect_uri=##################& client_secret=##################&
code={}& refresh_token={}&
grant_type=refresh_token”””.format(code,refresh_token)
headers = { ‘Content-Type’: “application/x-www-form-urlencoded”,
‘cache-control’: “no-cache” }

response = requests.request(“POST”, url, data=payload,
headers=headers)
r = response.json()
access_token = r.get(‘access_token’)
refresh_token = r.get(‘refresh_token’)

with open(filename, ‘wb’) as f:
      pickle.dump(refresh_token, f)
```

Calling the API and Cleaning and parsing the JSON

With your access_token in tow, you’re ready to call the API and start pulling data. The below code snippet will demonstrate how to retrieve the last 5 email messages with specific subject paramaters. Place the access_token (bolded below) in the headers and pass it in the get request to the email messages endpoint.

```
import json
import requests

url = ‘https://graph.microsoft.com/v1.0/me/messages
$search="subject:###SUBJECT YOU WANT TO SEARCH FOR###”’

headers = {
 ‘Authorization’: ‘Bearer ‘ +access_token,
 ‘Content-Type’: ‘application/json; charset=utf-8’
 }

r = requests.get(url, headers=headers)

files = r.json()

print(files)
```

The code above will return a json output that contains your data.

If you want more than 5 emails, you'll need to paginate through the results. Most APIs have thier own methods for how to best paginate results (in english, most APIs will not give you all the data you want with just one API call, so they give you ways to make multiple calls that allow you to move through the results in chunks to get all or as much as you want). The MS Graph is nice in that it provides the exact endpoint you need to go to to get the next page of results, making pagination easy. In the below code snippet, I demonstrate how to use a while loop to make successive calls to the api and add the subject and text of each email message to a list of lists.

```
emails = []
while True:
 try:
   url = files['@odata.nextLink']
   for item in range (0,len(files['value'])):
     emails.append(files['value'][item]['Subject'],[files['value']
[item]['bodyPreview'])
   r = requests.get(url, headers=headers)
   print(r.text)
   files = r.json()
 except:
   break
```

Putting the Data Somewhere

Finally, I'll show an example of how put the JSON data into a CSV text file, which could then be analyzed with excel or pushed into a database. This code snippet will demonstrate how to write each row of our list of lists into a CSV file.

```
import csv


write_file ='###LOCATION WHERE YOU WANT TO PUT THE CSV FILE###'
 with open(write_file,'w', newline = '') as f:
   writer = csv.writer(f, lineterminator = '\n')
   writer.writerows(emails)
```

Automate

Now that our script is written. We want to automate it so we can get new messages within our parameters as they come in daily.

I've done this in two ways, however, there are many ways to automate:

1. **Using Crontab on an Ubuntu Server**

This is quite simple. If your on Ubuntu, type into your console 'crontab -e' and it will open up your cronjobs in a text file. Go to a new line and type in:

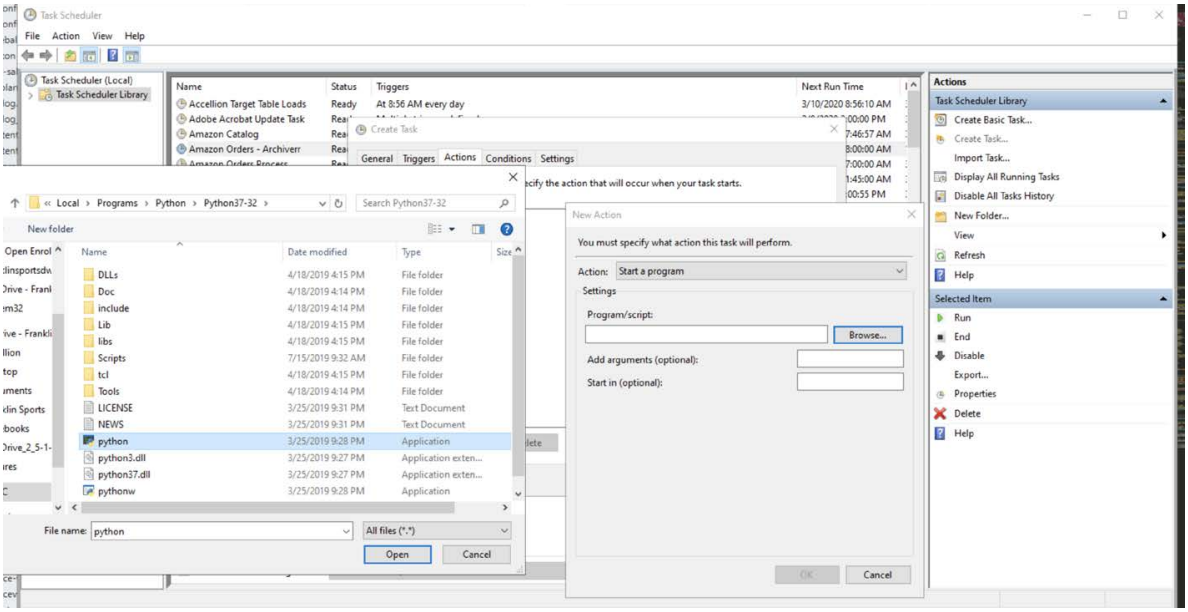
```
0 1 * * * /full path to python environment/python /full path to
file/example.py
```

Assuming cron is set up correctly, the above would set the script to automatically run at 1 UTC every day (or whatever timezone your server is on). You can change that time to whatever you need it to by changing the 1 in the code. If you want to get complex with your cron timing, this site is a nice resource.

2. **Using Windows Task Scheduler**

This is a nice and easy way to automate in windows that uses the Task Scheduler App. Simply open Task Scheduler and click 'Create Task' on the right hand side. This app is quite straightforward and allows you to set the schedule and what program you want to run using the GUI.

When your in the 'Actions' pain, select Python as the program and then in the Add Arguments tab (the one with '(optional)' next to it), put the path to your .py file.



Thats it!

In this walkthrough, I've shown you how to pull your email data from the Microsoft Graph API. The principles demonstrated here can be used to work with nearly any RESTful API or to do much more complex tasks with the Microsoft Graph (I've used the MS Graph API to push files in and out of our corporate sharepoint, push data from sharepoint folders into our database, and send automated emails and reports).

Questions or comments? You can email me at cwarren@stitcher.tech. Or, follow me on Linkedin at https://www.linkedin.com/in/cameronwarren/

I also provide Data services, which you can learn more about at http://stitcher.tech/.

API Python Data Science Data Engineering

146

WRITTEN BY
Cameron Warren
Follow
Data Scientist @ Franklin Sports. I write about the intersection of Data Science and Business, and ideas that challenge convention.

Towards Data Science
Follow
A Medium publication sharing concepts, ideas, and codes.

More From Medium

Farewell RNNs, Welcome TCNs

Bryan Tan in Towards Data Science
Aug 31 · 16 min read

805

5 Reasons why you should Switch from Jupyter Notebook to Scripts

Khuyen Tran in Towards Data Science
Aug 23 · 7 min read

2.2K

Stop One-Hot Encoding Your Categorical Variables.

Andre Ye in Towards Data Science
Aug 28 · 5 min read

1.2K