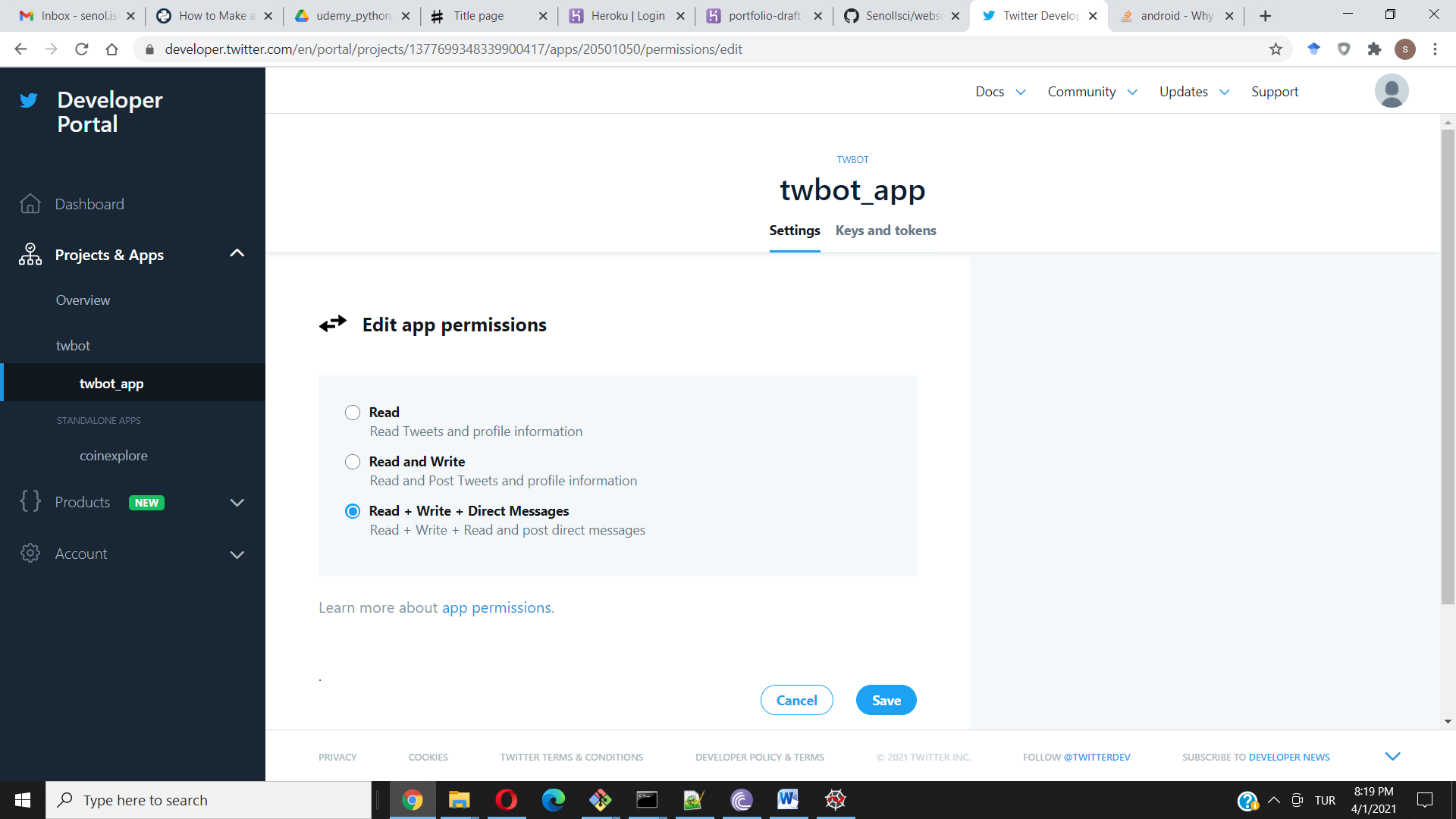
a **Twitter Bot**, a program that automates all or part of your Twitter activity

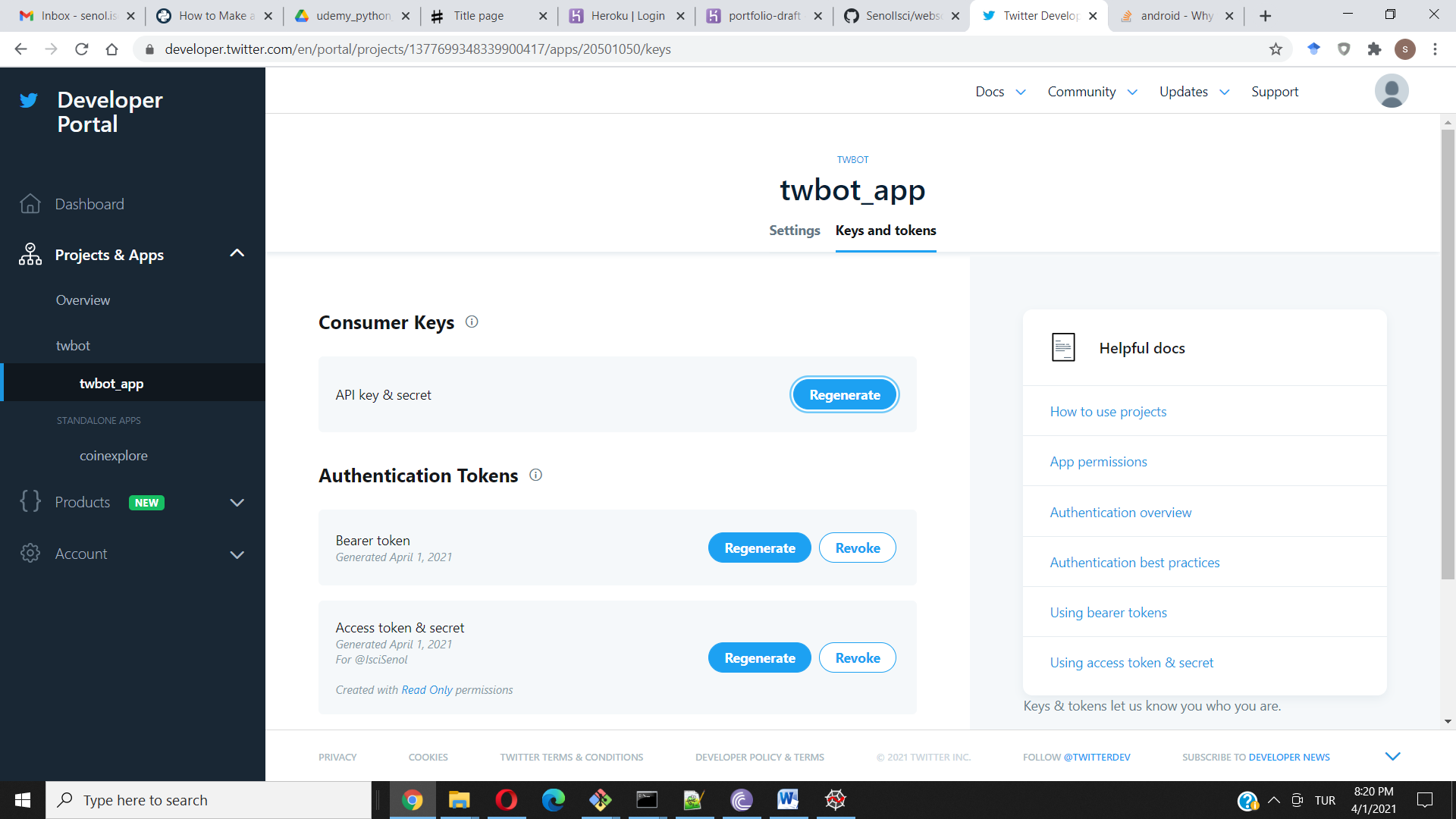
* Improve and automate your interactions with your Twitter audience
* Install Tweepy
* Sign up as a Twitter developer to use its API
* Use Tweepy to invoke the Twitter API
* Build Twitter Bots
* Deploy the bots to a server using Docker and AWS

on twitter dashboard create project and app

change app access permission settings:

* In the Settings tab, change the Application type to Read, Write and Access direct messages
* In the Reset keys tab, press the Reset button, update the consumer key and secret in your application accordingly.





**Twitter API**

The Twitter API gives developers access to most of Twitter’s functionality. You can use the API to read and write information related to Twitter entities such as tweets, users, and trends.

Technically, the API exposes dozens of HTTP endpoints related to:

* Tweets
* Retweets
* Likes
* Direct messages
* Favorites
* Trends
* Media

Create venv and install packages

prjfolder>python3 -m venv venv

prjfolder> venv\Scripts\activate

pip install tweepy

pip freeze > requirements.txt

Naming in Tweepy:

Since, over time, the names of various Twitter concepts have evolved, some old names are still used in Tweepy. So keep in mind that, in the context of this article, these equivalences hold:

* A **status** is a **tweet** .
* A **friendship** is a **follow-follower relationship**.
* A **favorite** is a **like**.

Tweepy’s functionality can be divided into the following groups:

* OAuth
* The API class
* Models
* Cursors
* Streams

**Methods for Streaming**

**Streaming** allows you to actively watch for tweets that match certain criteria in **real time**. This means that when there aren’t any new tweet matching the criteria, then the program will wait until a new tweet is created and then process it.

To use streaming you have to create two objects:

1. The **stream object** uses the Twitter API to get tweets that match some criteria. This object is the source of tweets that are then processed by a **stream listener**.
2. The **stream listener** receives tweets from the stream.

Here’s how you do that:

import json

import tweepy

class MyStreamListener(tweepy.StreamListener):

def \_\_init\_\_(self, api):

self.api = api

self.me = api.me()

def on\_status(self, tweet):

print(f"{tweet.user.name}:{tweet.text}")

def on\_error(self, status):

print("Error detected")

# Authenticate to Twitter

auth = tweepy.OAuthHandler("CONSUMER\_KEY", "CONSUMER\_SECRET")

auth.set\_access\_token("ACCESS\_TOKEN", "ACCESS\_TOKEN\_SECRET")

# Create API object

api = tweepy.API(auth, wait\_on\_rate\_limit=True,

wait\_on\_rate\_limit\_notify=True)

tweets\_listener = MyStreamListener(api)

stream = tweepy.Stream(api.auth, tweets\_listener)

stream.filter(track=["Python", "Django", "Tweepy"], languages=["en"])

You declared a new class, MyStreamListener. This class is used for the stream listener tweets\_listener. By extending Tweepy’s StreamLister, we reused code that is common to all stream listeners. Tweets from the stream are processed by on\_status().

We created the stream using tweepy.Stream, passing the authentication credentials and our stream listener. To start getting tweets from the stream, you have to call the stream’s filter(), passing the criteria to use to filter tweets. Then, for each new tweet that matches the criteria, the stream object invokes the stream listener’s on\_status().

**Models**

Tweepy uses its own model classes to encapsulate the responses from various Twitter API methods. This gives you a convenient way to use the results from API operations.

The model classes are:

* User
* Status
* Friendship
* SearchResults

**Cursors**

A lot of Twitter API endpoints use pagination to return their results. By default, each method returns the first page, which usually contains a few dozen items.

Tweepy cursors take away part of the complexity of working with paginated results. Cursors are implemented as a Tweepy class named Cursor. To use a cursor, you select the API method to use to fetch items and the number of items you want. The Cursor object takes care of fetching the various result pages transparently.

This code shows how, by using a cursor, you can get not only the first page from your timeline, but also the last 100 tweets:

for tweet in tweepy.Cursor(api.home\_timeline).items(100):

print(f"{tweet.user.name} said: {tweet.text}")

**Watching for Twitter Activity**

There are two ways to continuously watch for Twitter activity:

1. **Using streams:** to be notified when new content, such as tweets, that matches certain criteria is created
2. **Using polling:** to periodically make Tweepy API calls and then check their results to see if they contain something new

**Building the Docker Image**

To package your bot or application, you have to create a **Dockerfile** in the project’s root directory. This file contains a set of instructions used to create the Docker image containing your app.

This is how your project structure should look:

tweepy-bots/

│

├── bots/

│ ├── config.py

│ └── followfollowers.py

│ └── favretweet.py

│ └── autoreply.py

│

├── requirements.txt

└── Dockerfile

As you can see, the Dockerfile and requirements.txt live inside your project top directory.

FROM python:3.7-alpine

COPY bots/config.py /bots/

COPY bots/favretweet.py /bots/

COPY requirements.txt /tmp

RUN pip3 install -r /tmp/requirements.txt

WORKDIR /bots

CMD ["python3", "favretweet.py"]

$ docker build . -t fav-retweet-bot

$ docker run -it fav-retweet-bot