Flight Streaming Data

This directory has code and schemas for interacting with the streaming data from OpenSky.

The python/flight/stream folder contains code that will pull the latest data from OpenSky and store the data in GCS and/or publish the data as messages in a Pub/Sub queue.

How to Run Serveless

To run the code using a Cloud Function (aka. using serverless computing) you:

1. Configure your cloud function with permissions, time limits, and memory limits.
2. Supply the code and requirements.txt to the function. *This applies to a Cloud Function based on Python*.
3. Deploy the function.
4. Test the function with an example message.

You can then create a Big Query table from either the Cloud Storage bucket or the Pub/Sub queue using the provided schemas.

Configure a Cloud Function

1. Environment: You can use either 1st Gen or 2nd Gen. 1st Gen has an easier interface for testing. 2nd Gen has the ability to have longer running cloud functions. (1st Gen will be limited to 10 mins.)
2. Function name: Choose a meaningful name. It can only have lowercase characters, numbers, and hyphens.
3. Trigger: Select HTTP/HTTPS. This means the function will be triggered whenever anyone hits a specific URL. (Other triggers are possible, such as triggering a function whenever a file is uploaded to cloud storage, a message is published in a Pub/Sub queue, and so on.)
4. Authentication: Allow unauthenticated invocations. Permissions often get hairy to set up properly, though you will want to enforce security constraints when working in a production system. For our purposes, we will not restrict access to the URL that is used to trigger the Cloud Function.
5. *(If using 1st Gen)* Click SAVE at the bottom of the Trigger box.
6. Runtime, build, and connection settings. You don't need to change the defaults. You may want to change the following: Memory allocated (if your function generates "Out of

Memory" errors, but smaller memory allocation relates to smaller costs); Timeout (increase up to the maximum if your function fails to complete in time; smaller timeout means a function that hangs will be terminated more quickly); Autoscaling (lower the Maximum number of instances if you want to make sure the function doesn't spawn thousands of instances at one time when you have a lot of trigger events).

Supply the Code

1. Select one of the Python 3 versions.
2. The entry point of the code is a method named flightStreaming. Change the default entry point from hello_world to flightStreaming.
3. Package up the code.

   A script is provided to package up the code for the flight streaming function. It creates a folder in /tmp/flight-streaming_zip and puts the subset of python files there, zips them up, and stores the zip file in Google Cloud Storage in a function subfolder in your bucket.

   You provide the script with the name of the cloud function, which is flight-streaming in this case.

```
sh/createFlightStreamingZip.sh flight-streaming
```

You can browse Cloud Storage in the Cloud Function console to select the zip file.

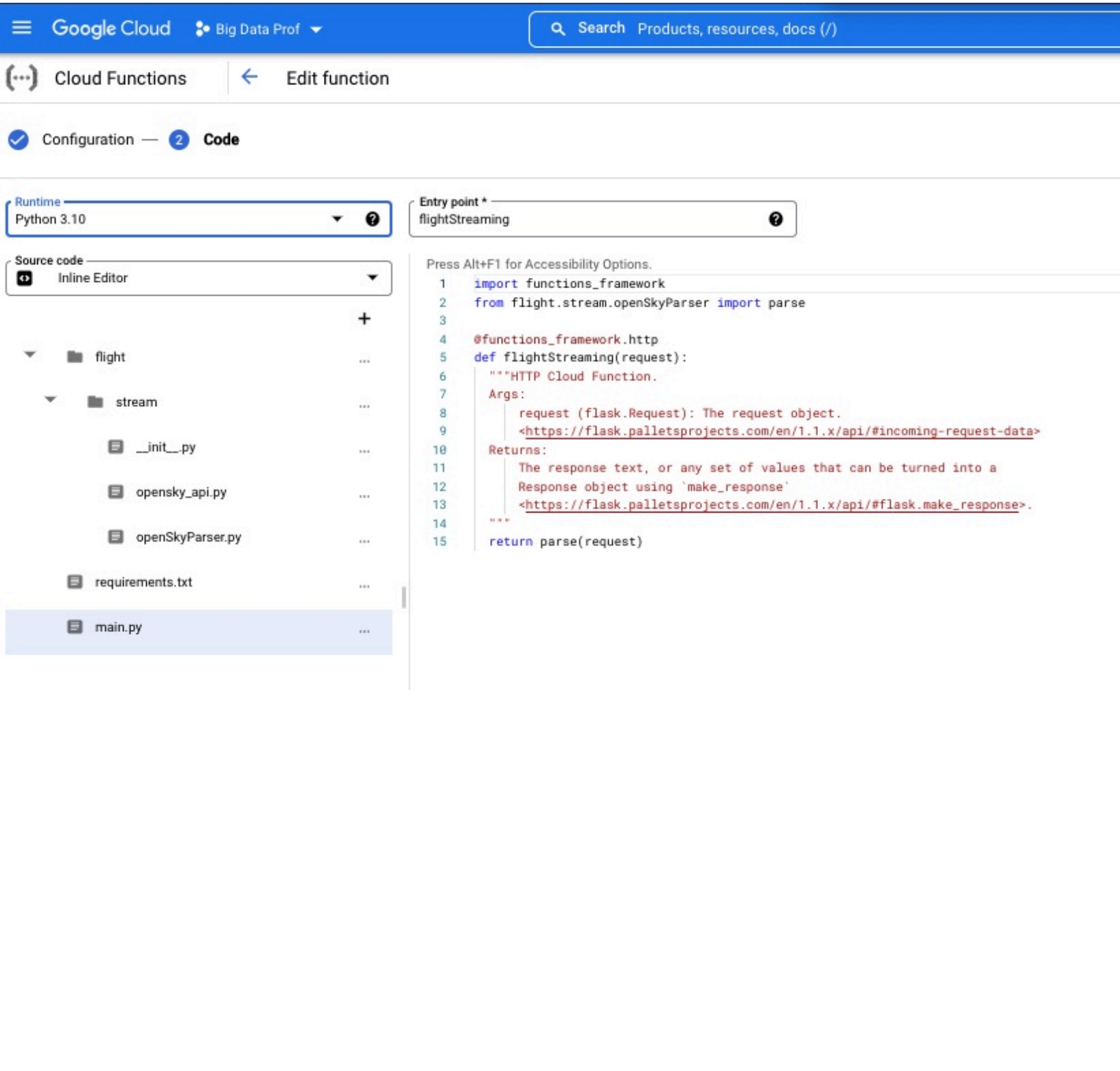
You should see something like:

Once ready, select DEPLOY. It will take some time to create a virtual machine image and deploy your code into it. It will finish spinning and either have a green check mark or an error.

Test Your Function

You can manually trigger your function using the Testing tab in the Cloud Function console (for 1st Gen) or copy-pasting a curl command in Cloud Shell (for 2nd Gen). In either case, you can supply a test such as:

```
{"storage":true,"bucket":"prof-big-data_data","path":"flight-streaming","separateLines":true}
```


Google Cloud
Big Data Prof
Search Products, resources, docs (/)
Cloud Functions
Edit function
Configuration — 2 Code
Runtime
Python 3.10
Entry point *
flightStreaming
Source code
Inline Editor
flight
stream
__init__.py
opensky_api.py
openSkyParser.py
requirements.txt
main.py
Press Alt+F1 for Accessibility Options.
import functions_framework
from flight.stream.openSkyParser import parse
@functions_framework.http
def flightStreaming(request):
"""HTTP Cloud Function.
Args:
request (flask.Request): The request object.
<https://flask.palletsprojects.com/en/1.1.x/api/#incoming-request-data>
Returns:
The response text, or any set of values that can be turned into a
Response object using `make_response`
<https://flask.palletsprojects.com/en/1.1.x/api/#flask.make_response>.
"""
return parse(request)