↩ All articles

Streaming data from Twitter to GCP

11/3/2019 Practicum Mia Tadić

Google Cloud Platform (GCP) is a suite of public cloud computing services offered by Google. The platform includes a range of hosted services for compute, storage and application development that run on Google hardware. Services used in this blog are Google Cloud Pub/Sub and Google BigQuery.

- ***Google Cloud Pub/Sub*** is a managed, scalable and real-time messaging ingestion service that allows messages to be exchanged between applications.
- ***Google BigQuery*** is a scalable, managed enterprise data warehouse for analytics. It is a big data service for data processing and analytics for SQL-like queries made against multi-terabyte data sets.

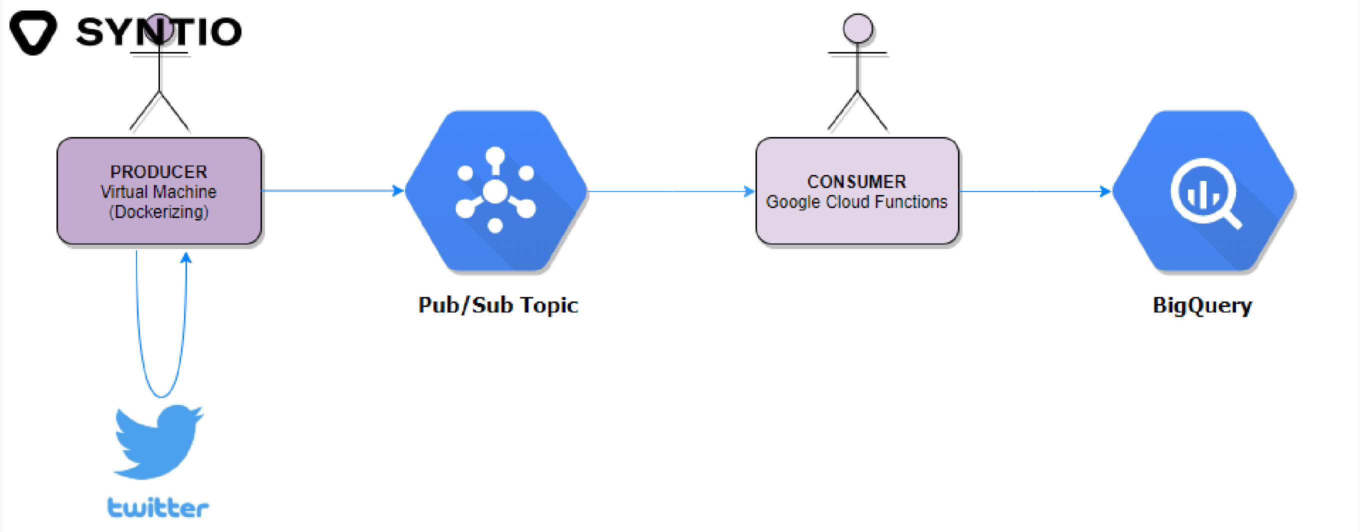
Task idea

This blog demonstrates the task of ingesting data from remote API (i.e. Twitter's API) to cloud (i.e. Google Cloud).

It was done using Producer and Consumer.

- ***Producer*** sent tweets containing hashtag „dataengineering“ from Twitter to Pub/Sub topic. Producer was dockerized and then ran on Google Cloud's Virtual Machine.
- ***Consumer*** streamed messages from Pub/Sub topic to BigQuery table. Consumer was created using Google Cloud Functions, and it was **automatically** triggered by any new messages.

Described process is shown in following diagram:



Procedure

Here are the prerequisites and steps of our procedure.

Prerequisits for the task are:

1. Twitter Developer account
2. Google Cloud Platform account
3. Installed Docker

Main steps of procedure are:

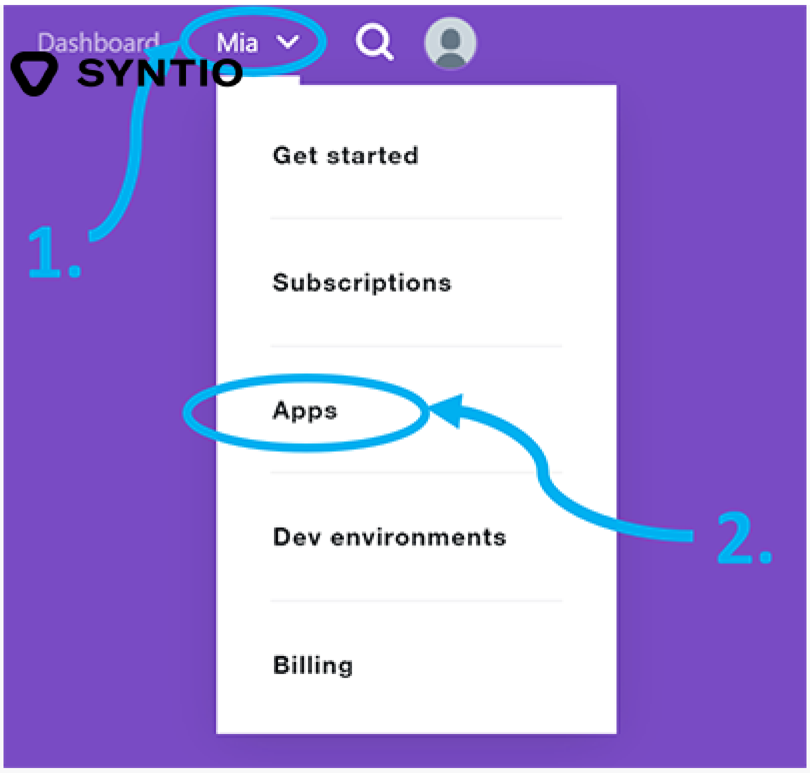
1. Obtaining Twitter credentials
2. Creating a Producer
3. Creating a Consumer
4. Dockerizing a Producer

Let's dig into steps of the procedure.

Obtain Twitter credentials

App's **client key** and **secret key**, **access token** and **token secret** can be found in Twitter Dashboard.

Those were needed in later step.



Create a Producer

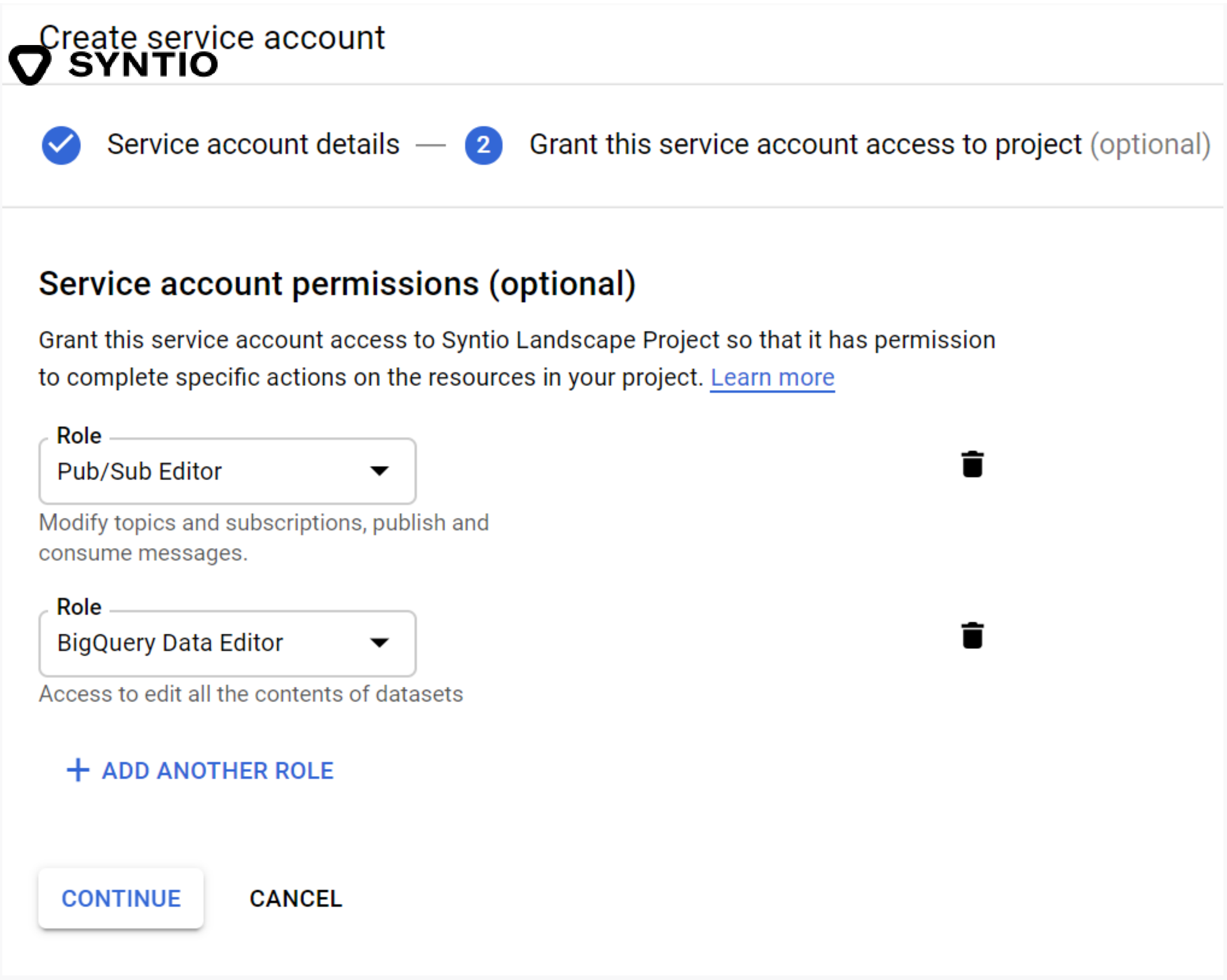
First, following resources were created:

1. Service account

Service account authorized access to needed GCP resources (Pub/Sub, BigQuery).

In GCP console, *Service accounts* can be found under *IAM & Admin* section. By clicking on **CREATE SERVICE ACCOUNT** button, several steps were required:

- Step 1: Filling in desired details
- Step 2: Adding „Pub/Sub Editor“ and „BigQuery Data Editor“ roles


Create service account

Service account details — 2 Grant this service account access to project (optional)

Service account permissions (optional)

Grant this service account access to Syntio Landscape Project so that it has permission to complete specific actions on the resources in your project. Learn more

Role: Pub/Sub Editor

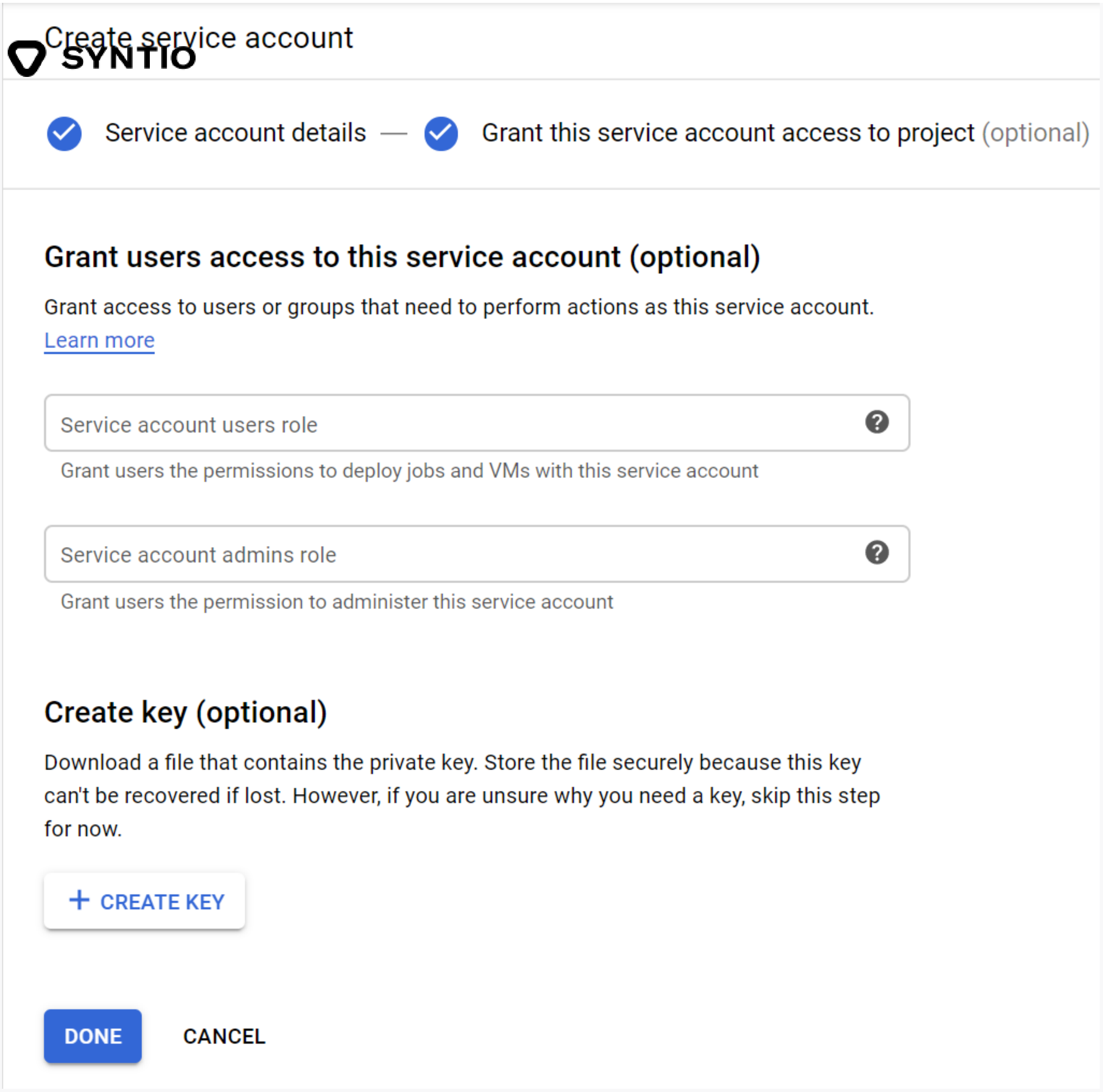
Modify topics and subscriptions, publish and consume messages.

Role: BigQuery Data Editor

Access to edit all the contents of datasets

- Step 3: Downloading JSON key file by choosing **CREATE KEY** option

SYNTIO


Create service account

✓ Service account details — ✓ Grant this service account access to project (optional)

Grant users access to this service account (optional)

Grant access to users or groups that need to perform actions as this service account.
Learn more

Service account users role

Grant users the permissions to deploy jobs and VMs with this service account

Service account admins role

Grant users the permission to administer this service account

Create key (optional)

Download a file that contains the private key. Store the file securely because this key can't be recovered if lost. However, if you are unsure why you need a key, skip this step for now.

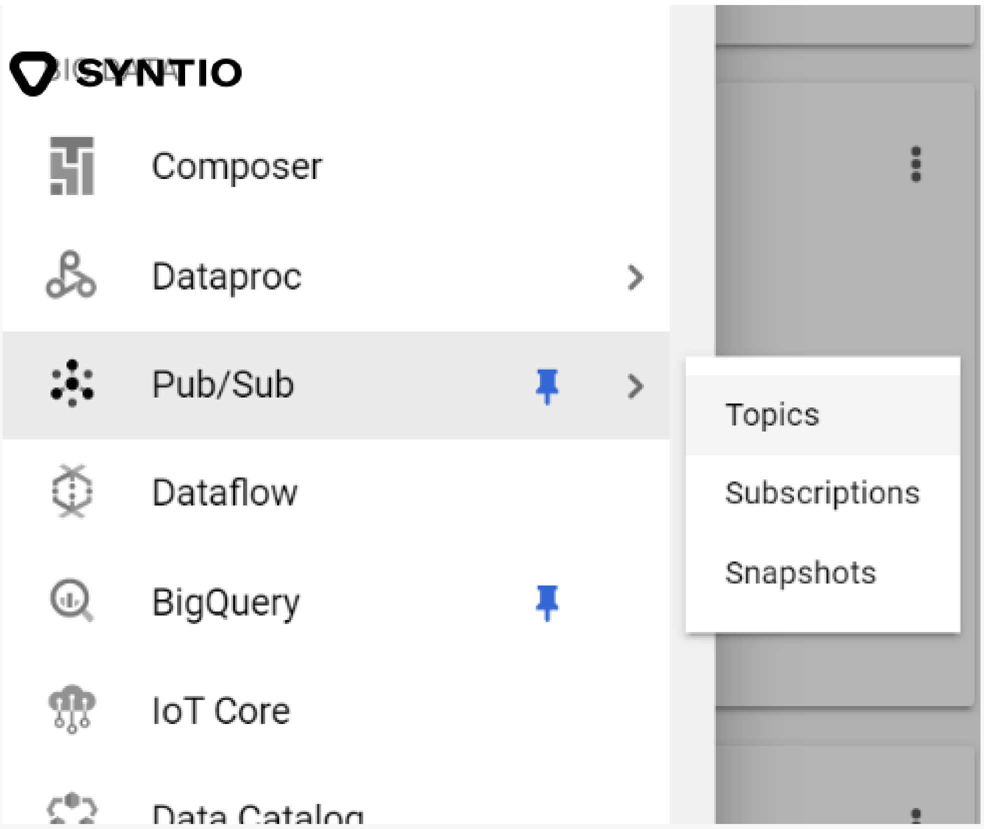
Additionally, user environment variable had to be added:

name=GOOGLE_AUTH_CREDENTIALS

value=*path_to_downloaded_json_file*

2. Pub/Sub Topic

Pub/Sub topic was the place where Producer sent tweets to.



3. Pub/Sub Subscription

Subscription pulled data from desired Pub/Sub topic. This was a way to see how data looked like. *Subscriptions* can be found in the picture above.

Finally, it was all set for Producer to be created.

As mentioned, the Producer's task was to send tweets containing hashtag "dataengineering" from Twitter to Pub/Sub topic.

Note: Keys and tokens obtained from Twitter were needed here.

Firstly, Pub/Sub package needed to be installed:

```
pip install –upgrade google-cloud-pubsub
```

To begin with Python script, following packages were imported:

```
import json

from tweepy import Stream
from tweepy import OAuthHandler
from tweepy.streaming import StreamListener
from google.cloud import pubsub_v1
```

Since Producer was said to send tweets to Pub/Sub topic, connection to Pub/Sub topic had to be established somehow. It was done this way:

```
client = pubsub_v1.PublisherClient()
```

```
topic_path = client.topic_path(<project_id>, <topic_name>)
```

On the other side, since Producer accessed Twitter to grab its data, it was necessary to make connection to Twitter too. That connection was made this way:

```
auth = OAuthHandler(<twitter-client-key>, <twitter-secret-key>)
auth.set_access_token(<twitter-access-token>, <twitter-token-secret>)
twitterStream = Stream(auth, listener())
twitterStream.filter(track=["dataengineering"])
```

This code called *tweepy*'s implemented methods. Additionally, object *listener* was introduced in the code for the purposes of the task. It was a class which inherited *tweepy*'s *StreamListener* class.

For the demonstration purposes, only following parts of the grabbed tweets were taken to save them to Pub/Sub topic:

- created_at
- id
- text
- source
- retweet_count
- user_name
- user_location
- user_followers_count
- user_statuses_count

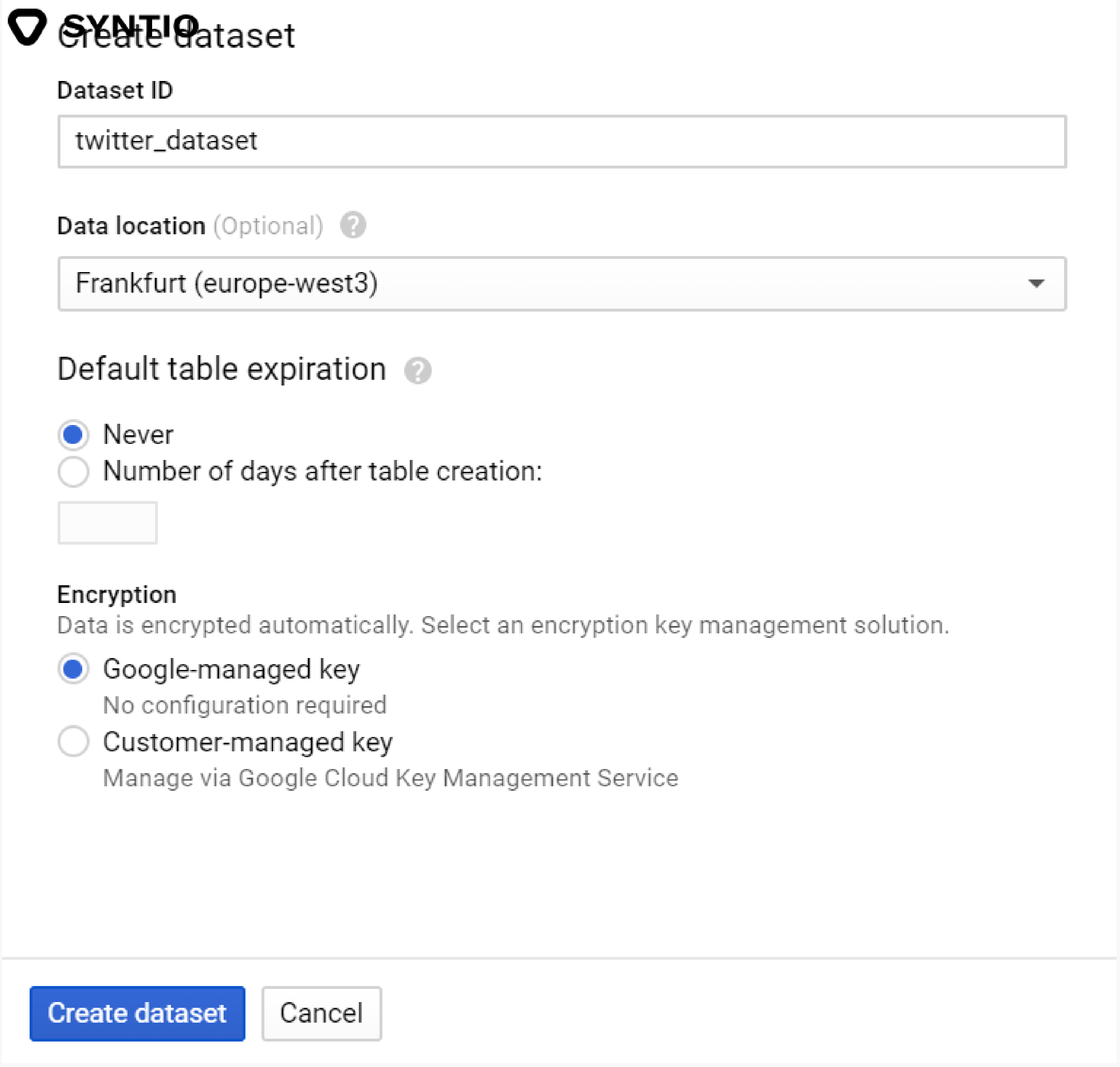
Extracted parts were concatenated in *tweet_message* after which *tweet_message* was published to Pub/Sub topic using following command:

```
self.client.publish(self.topic_path, data=tweet_message.encode('utf-8'))
```

CREATE A CONSUMER

As stated before, Consumer's task was to stream messages from Pub/Sub topic to BigQuery table – so a BigQuery table had to be created.

BigQuery tables are generally contained in datasets. Therefore, dataset had to be created first. It was created in GCP console, by finding *BigQuery* under *Big Data* section, choosing project and clicking on **CREATE DATASET**. It popped a form which can be seen in a picture below:


Create dataset

Dataset ID: twitter_dataset

Data location (Optional): Frankfurt (europe-west3)

Default table expiration

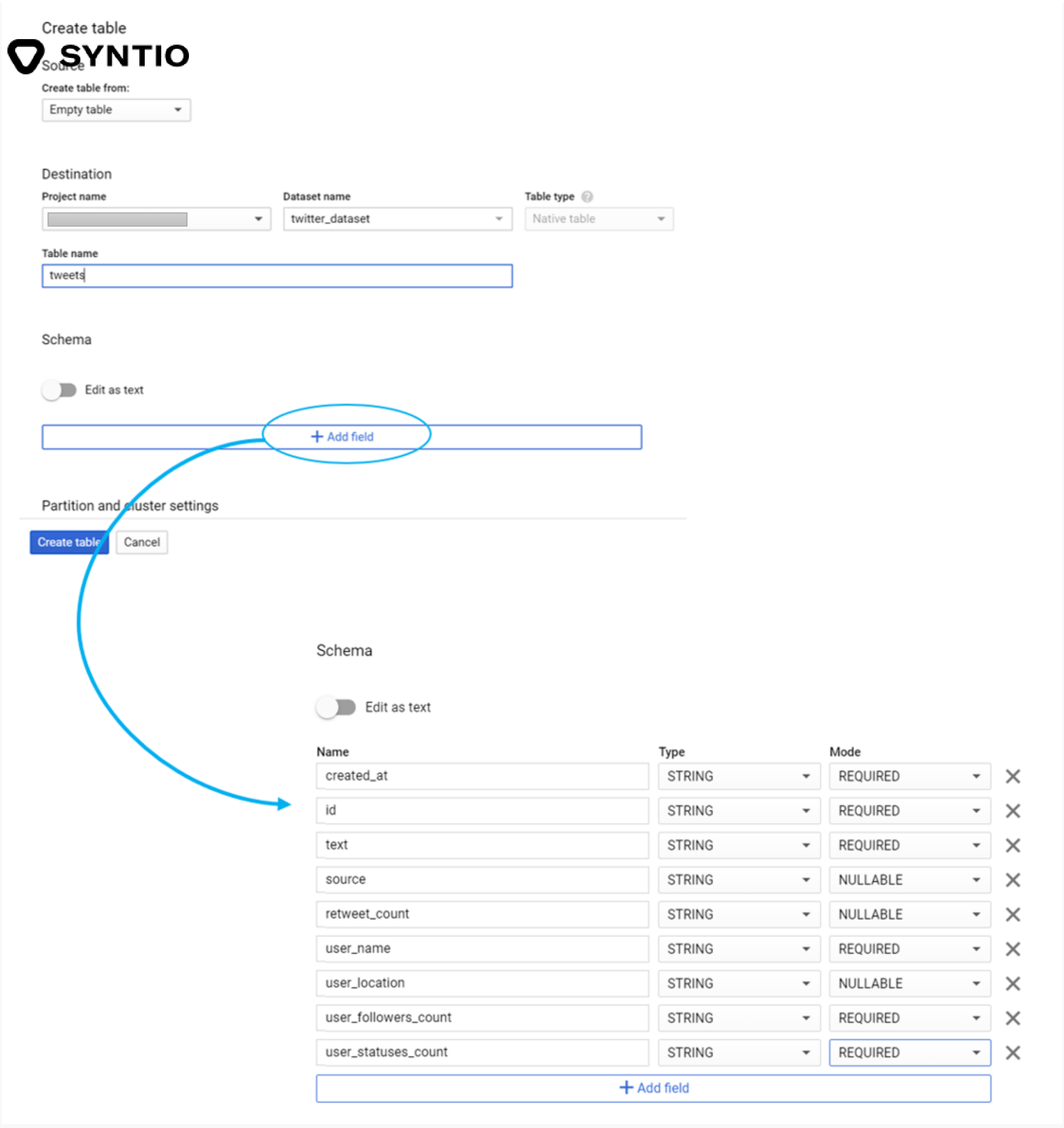
[✓ Never]
Number of days after table creation:

Encryption
Data is encrypted automatically. Select an encryption key management solution.

[✓ Google-managed key]
No configuration required

Customer-managed key
Manage via Google Cloud Key Management Service

Then, by entering the created dataset and simply clicking on **CREATE TABLE** button, table was created. Based on extracted tweet's parts, table's schema must have looked like this:



Create table

Source

Create table from: Empty table

Destination

Project name	Dataset name	Table type
[redacted]	twitter_dataset	Native table

Table name: tweets

Schema

Edit as text

+ Add field

Partition and cluster settings

Create table | Cancel

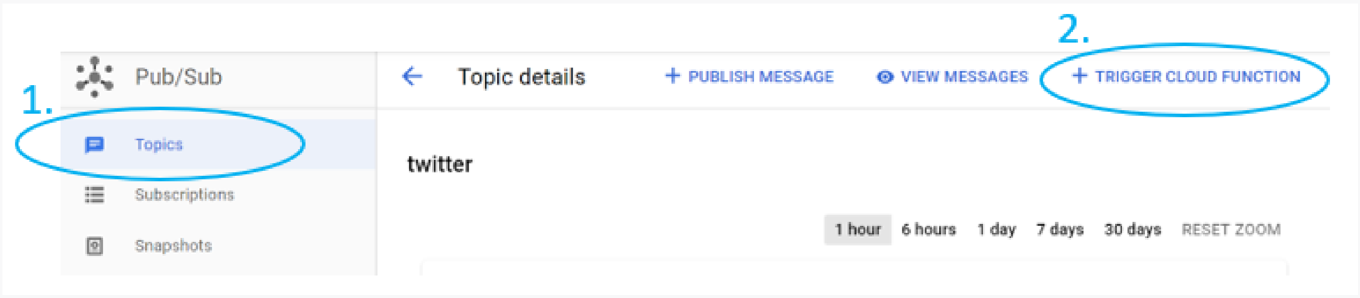
Schema

Edit as text

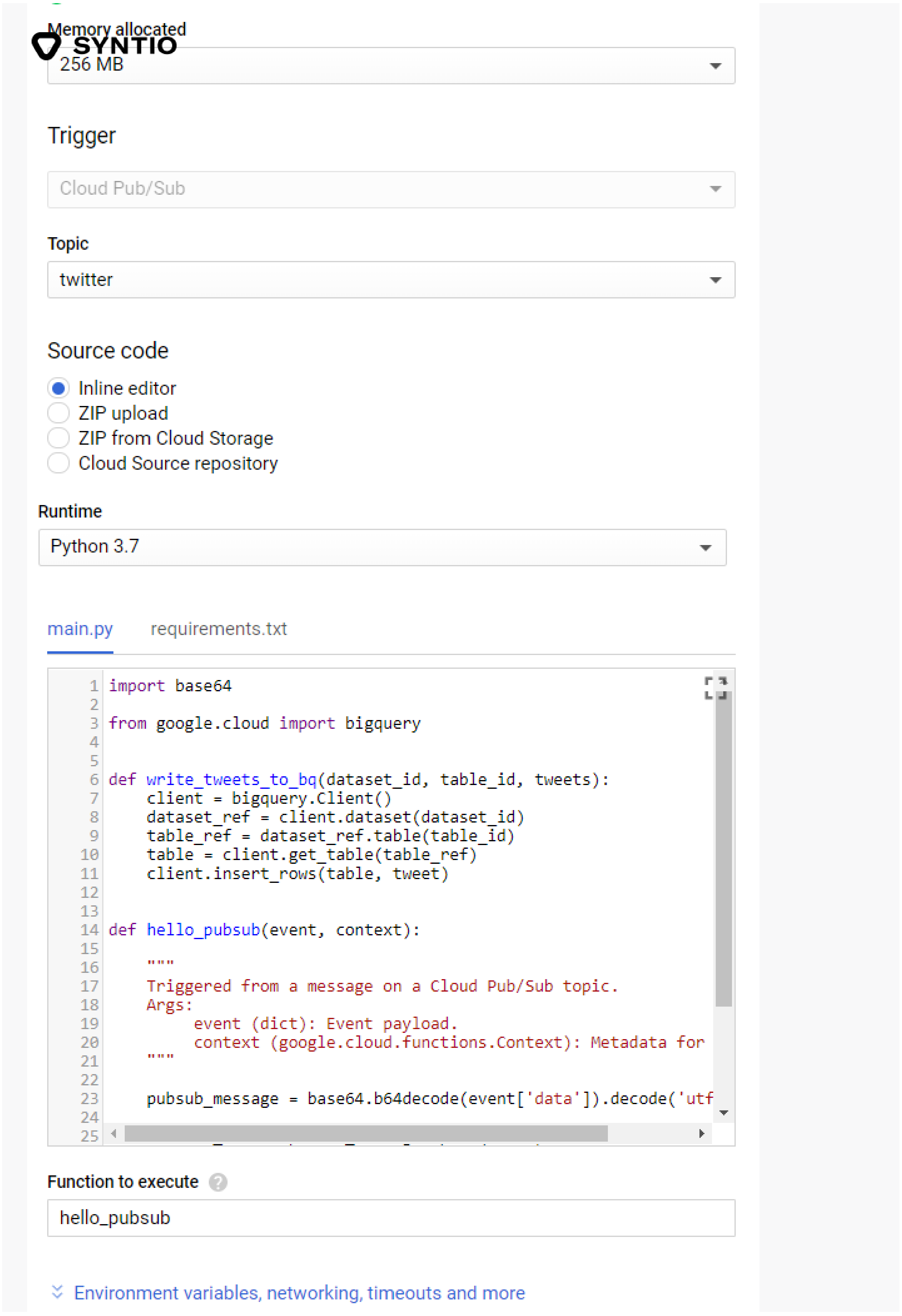
Name	Type	Mode
created_at	STRING	REQUIRED
id	STRING	REQUIRED
text	STRING	REQUIRED
source	STRING	NULLABLE
retweet_count	STRING	NULLABLE
user_name	STRING	REQUIRED
user_location	STRING	NULLABLE
user_followers_count	STRING	REQUIRED
user_statuses_count	STRING	REQUIRED

+ Add field

At last, Consumer was ready to be created. As mentioned, Consumer was created using Google Cloud Functions.



Missing values needed to be filled in:



Memory allocated

256 MB

Trigger

Cloud Pub/Sub

Topic

twitter

Source code

● Inline editor
○ ZIP upload
○ ZIP from Cloud Storage
○ Cloud Source repository

Runtime

Python 3.7

main.py requirements.txt

```python
import base64

from google.cloud import bigquery


def write_tweets_to_bq(dataset_id, table_id, tweets):
    client = bigquery.Client()
    dataset_ref = client.dataset(dataset_id)
    table_ref = dataset_ref.table(table_id)
    table = client.get_table(table_ref)
    client.insert_rows(table, tweet)


def hello_pubsub(event, context):

    """
    Triggered from a message on a Cloud Pub/Sub topic.
    Args:
         event (dict): Event payload.
         context (google.cloud.functions.Context): Metadata for
    """

    pubsub_message = base64.b64decode(event['data']).decode('utf
```

Function to execute

hello_pubsub

Environment variables, networking, timeouts and more

File *main.py* contained Consumer code, while *requirements.txt* contained function dependencies (packages with their versions).

Google Cloud Function was triggered from a message on a Cloud Pub/Sub topic. This event was passed to *hello_pubsub* function as an argument and then function collected data (tweets) from the event:

```
pubsub_message = base64.b64decode(event['data']).decode('utf-8')
```

Since Consumer's task was to write messages to BigQuery table, Consumer had to connect to BigQuery table. To establish the connection, the following code was used:

```
client = bigquery.Client()
dataset_ref = client.dataset(<dataset_id>)
table_ref = dataset_ref.table(<table_id>)
table = client.get_table(table_ref)
client.insert_rows(table, <tweet>)
```

Dockerize

Following step in our *steps of procedure* was to dockerize Producer. The idea was to pack the Producer into Docker image, send the image to virtual machine and then run it from there.

For the purposes of this task, Docker Desktop and Docker Hub were used and the instructions and installation file for various operational systems can be found here.

For Docker, two files (*Dockerfile* and *requirements.txt*) must be created and saved into the **same Python project** as producer code.

Dockerfile should look like this:

```
# Use an official Python runtime as a parent image
FROM python:3.7-slim

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
ADD . /app

# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt

# Define environment variable
```

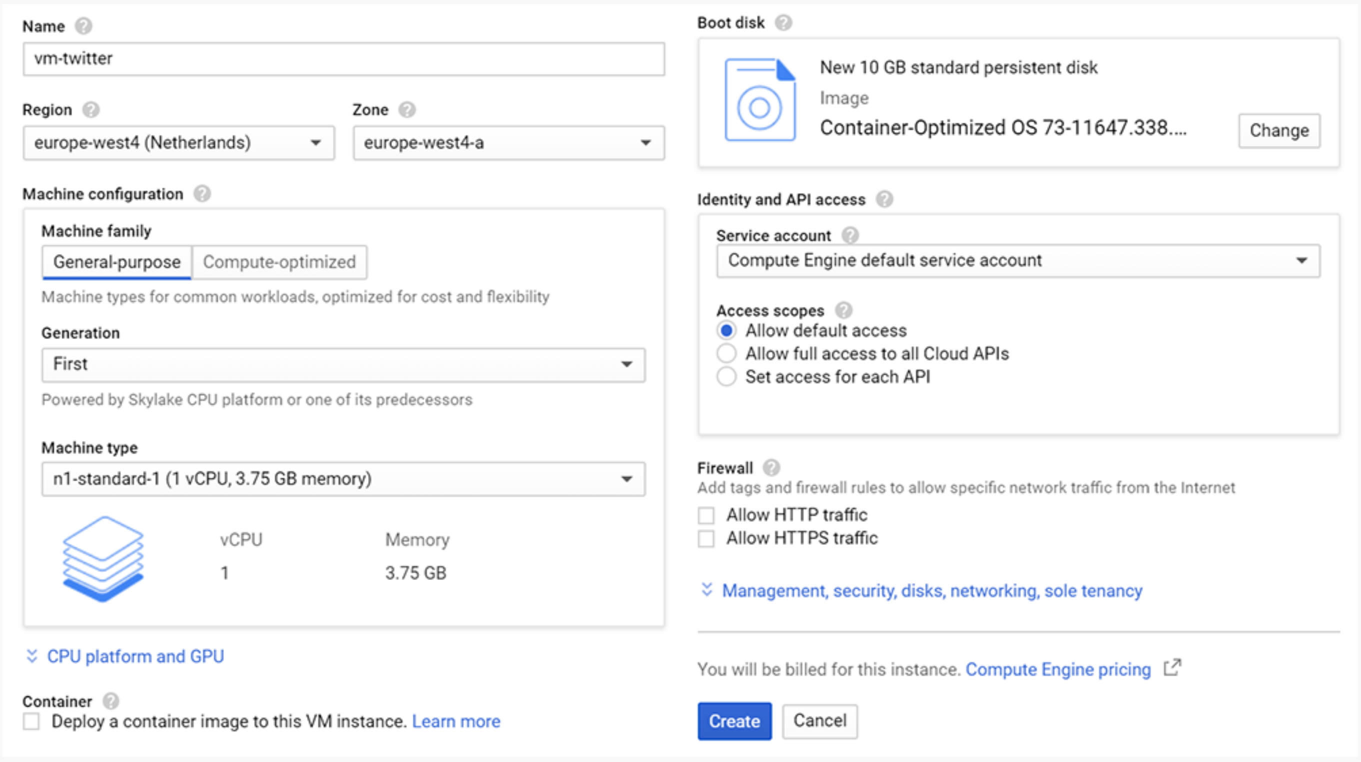
```
ENV GOOGLE_APPLICATION_CREDENTIALS="<json-key-file>"

# Run app.py when the container launches
CMD ["python", "producer.py"]
```

Note:

- After checking which Python version is being used, it is fine to just add suffix “slim” with no additional installation.
- JSON key file must be copied into the **same Python project** directory (it is Dockerfile's working directory now, so that is the place where Docker searches for environment variable)

As described in the beginning of the blog, Docker image was deployed to the virtual machine. Therefore, in this preparation for dockerizing, next step was to create GCP's virtual machine.

To create VM, *VM instances* can be found in GCP console, under *Compute* section and by choosing *Compute Engine* and then clicking **CREATE INSTANCE.** For this demonstration, following details were sufficient:



Name
vm-twitter

Region
europe-west4 (Netherlands)

Zone
europe-west4-a

Machine configuration

Machine family
General-purpose | Compute-optimized
Machine types for common workloads, optimized for cost and flexibility

Generation
First
Powered by Skylake CPU platform or one of its predecessors

Machine type
n1-standard-1 (1 vCPU, 3.75 GB memory)

vCPU	Memory
1	3.75 GB

CPU platform and GPU

Container
Deploy a container image to this VM instance. Learn more

Boot disk
New 10 GB standard persistent disk
Image
Container-Optimized OS 73-11647.338....
Change

Identity and API access

Service account
Compute Engine default service account

Access scopes
(●) Allow default access
( ) Allow full access to all Cloud APIs
( ) Set access for each API

Firewall
Add tags and firewall rules to allow specific network traffic from the Internet
☐ Allow HTTP traffic
☐ Allow HTTPS traffic

Management, security, disks, networking, sole tenancy

You will be billed for this instance. Compute Engine pricing

Create | Cancel

At last, steps for deploying Docker image to VM were:

1. Build Docker image

```
docker build -t producer .
```

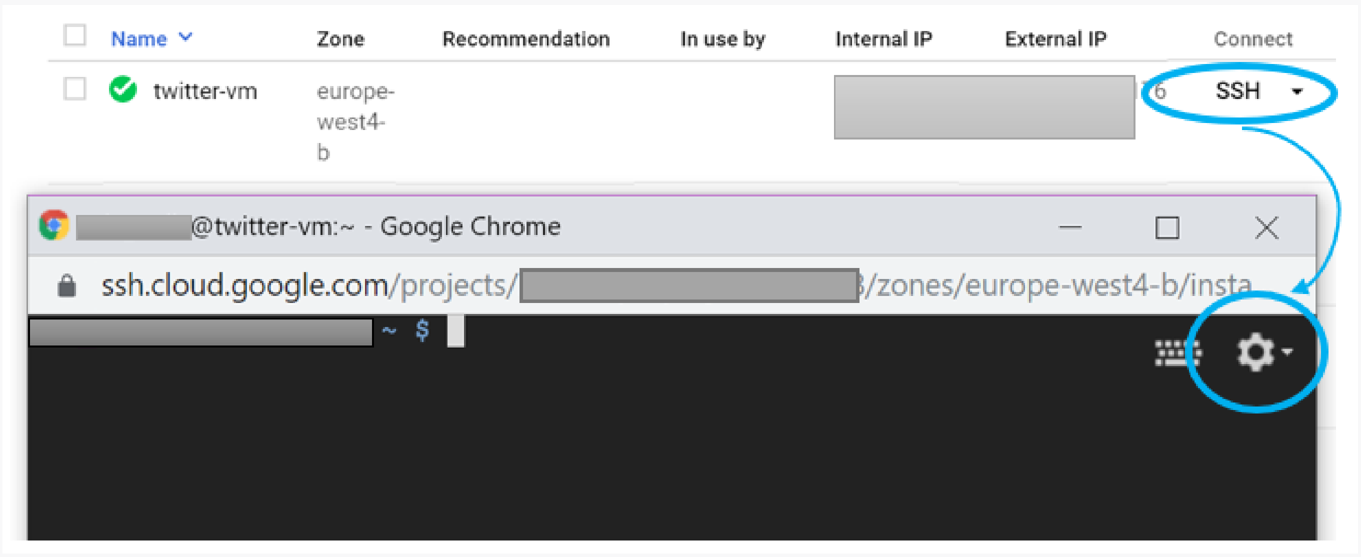
2. Zip Docker image

```
docker save -o producer.tar producer
```

Note: Make sure to run both steps from producer's Python project directory

3. Upload zip file to virtual machine

By clicking first on **SSH** button in VM instances in GCP console, and afterwards on **Settings** button, *Upload file* option was chosen and *producer.tar* file was uploaded.



4. Unzip Docker image

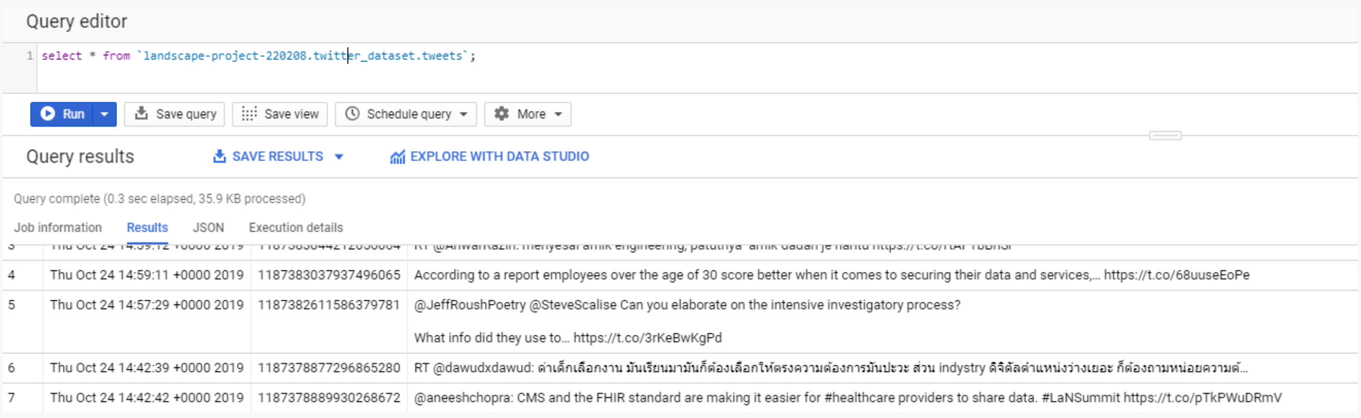
```
docker load -i producer.tar
```

5. Run Docker image

```
docker run -d producer
```

With this, the process of ingesting messages to Pub/Sub topic was completed.

To check if created Producer and Consumer work properly, BigQuery table can be checked. Table from this demonstration looked like this:



```
select * from `landscape-project-220208.twitter_dataset.tweets`;
```

Query complete (0.3 sec elapsed, 35.9 KB processed)

3	[illegible]	[illegible]	[illegible]
4	Thu Oct 24 14:59:11 +0000 2019	1187383037937496065	According to a report employees over the age of 30 score better when it comes to securing their data and services,... https://t.co/68uuseEoPe
5	Thu Oct 24 14:57:29 +0000 2019	1187382611586379781	@JeffRoushPoetry @SteveScalise Can you elaborate on the intensive investigatory process? What info did they use to... https://t.co/3rKeBwKgPd
6	Thu Oct 24 14:42:39 +0000 2019	1187378877296865280	RT @dawudxdawud: ค่าเด็กเลือกงาน มันเรียนมามันก็ต้องเลือกให้ตรงความต้องการมันปะวะ ส่วน indystry ดิจิตัลตำแหน่งว่างเยอะ ก็ต้องถามหน่อยความต้...
7	Thu Oct 24 14:42:42 +0000 2019	1187378889930268672	@aneeshchopra: CMS and the FHIR standard are making it easier for #healthcare providers to share data. #LaNSummit https://t.co/pTkPWuDRmV

Conclusion and ideas

In this step-by-step tutorial, it was presented how to stream data to cloud from remote API using a simple Producer-Consumer flow. Instead of GCP and Twitter, there are plenty more cloud platforms and API's to use. There are also other Google Cloud resources to play with, like Google Cloud Storage or Google Cloud SQL. Another tip is to send data from Cloud Pub/Sub topic to BigQuery using Google Cloud Dataflow.

Note: Don't forget to delete the resources when they are not needed anymore.

This project was co-financed by the European Union's Competitiveness and cohesion Operational Programme.The content of the website is the sole responsibility of Syntio d.o.o. About the project

European Union
Together with EU funds

EUROPEAN STRUCTURAL AND INVESTMENT FUNDS

Operational programme
COMPETITIVENESS AND COHESION