Simulación de publicación e ingesta de datos

Jaime García Lozano

22 de mayo de 2022

1. Introducción

Vamos a usar varios recursos de Goodle Cloud para simular una publicación e ingesta de datos. Los servicios implicados son:

- Servicio de Pub/Sub.
- Google Storage.
- Google Compute.

Simularemos la gestión de canales de publicación y subscripción mediante scripts de Python (*randomEvents*, *pubEvents* y *subEvents*) que permitan tratar la generación de eventos y su consumo

La arquitectura del proceso será el siguiente:

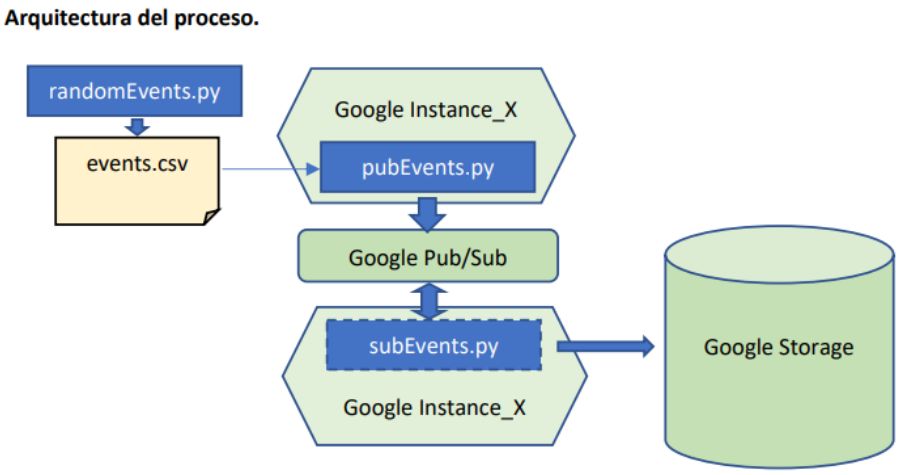

Figura 1: Arquitectura del proceso

2. Creación de la instancia

Siguiendo el esquema de la Fig. 1, el primer paso es crear la instancia donde ejecutaremos los scripts. (Nota: Se han elegido los ajustes menos costosos y se le ha dado permiso al acceso total a todas las API de Cloud)

Instalamos Python y *pip* ejecutando los siguientes comandos en la consola de la instancia:

```
sudo apt update
sudo apt install python3 python3-dev python3-venv
sudo apt-get install wget
wget https://bootstrap.pypa.io/get-pip.py
sudo python3 get-pip.py
```

Instalamos las librerías necesarias:

```
sudo pip install --upgrade google-cloud-storage

sudo pip install google-cloud-pubsub
```

3. Los scripts y resultados

Dentro de la instancia se programan los siguientes scripts (se pueden consultar en el siguiente repositorio [1]):

- **randomEvents:** genera un csv con los eventos, los cuales se componen de un tiempo, un topic (*book*,*search* o *buy*) y un mensaje. Los mensajes contienen la codificación el usuario. Finalmente, el tiempo representa el periodo entre llegadas de dicho evento al sistema.
- **pubEvents:** publicará en Google Pub/Sub cada evento del csv. Primeramente se han de crear los *topics* a partir del siguiene comando:

  ```
  gcloud pubsub topics create "nombre del topic"
  ```

  Una vez generados, el script será capaz de publicar cada evento en su respectivo topic con su mensaje y esperando el tiempo correspondiente entre publicación y publicación. Dentro del mensaje, guardamos también el nombre del topic.
- **bucket:** su ejecución creará un *bucket* en el Google Storage [2].
- **subEvents:** su función es mantener un histórico de cada mensaje recibido. Primeramente se ha de crear una suscripción para cada topic:

  ```
  gcloud pubsub subscriptions create SUBSCRIPTION_ID \
  --topic=TOPIC_ID \
  ```

  La ejecución de *randomEvents* ha de ser simultánea a la de *pubEvents* e irá suscribiéndose a cada publicación generada guardándola en un fichero correspondiente a su topic (que se encontrará en el *bucket*).

  (Nota: para programar los scripts *pubEvents* y *randomEvents* se han consultado las siguientes fuentes: [3] y [4])

  En la práctica se han de tener dos consolas de la instancia abiertas: en una se ejecuta el *pubEvents* y en la otra el *subEvents*. En este último hay un parámetro llamado *timeout* que determinará el tiempo que estará 'escuchando' (preferiblemente mayor que el tiempo total que tardan los mensajes en publicarse) . Una vez terminado este tiempo, la ejecución se detiene.

  Los ficheros con el histórico de los mensajes de cada topic se irán actualizando con los nuevos mensajes en tiempo real (es posible comprobarlo si abrimos el *bucket* durante la ejecución).

Finalmente, los resultados en el *bucket* han de quedar de la siguiente manera:

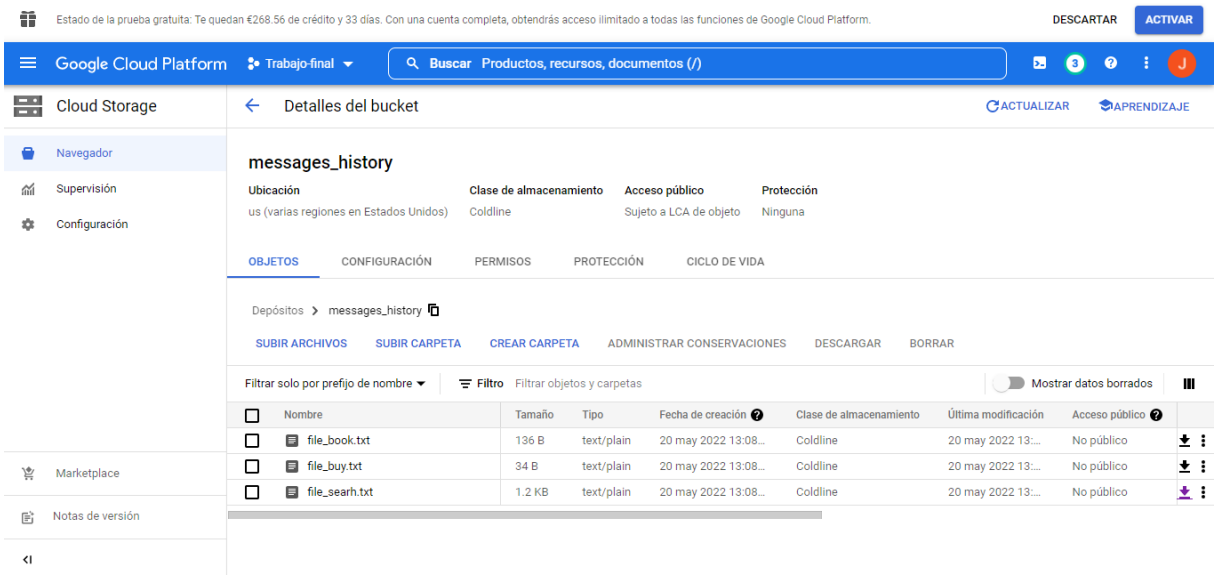

Figura 2: *Bucket* con los ficheros de cada topic (*file_book.txt*, *file_buy* y *file_search*)

.

Referencias

[1] *Repositorio*, https://github.com/JGL98/Trabajo_cloud.

[2] *Create bucket*, https://cloud.google.com/storage/docs/creating-buckets#prereq-code-samples.

[3] *PubSub tutorial*, https://cloud.google.com/pubsub/docs/create-topic-client-libraries?hl=es-419.

[4] *File to bucket*, https://cloud.google.com/storage/docs/samples/storage-upload-file?hl=es-419#storage_upload_file-python.