Big Data Lab - Lab 6

RA Keerthan
CH17B078

1.a)

The file is downloaded and saved as *data.txt*. It is uploaded to google cloud shell from where it is copied to the bucket *bdl_6*.

1.b)

First step is to create a topic and subscription.

For this purpose, we create a topic using the following gcloud command:

```
gcloud pubsub topics create file-name-topic
```

```
(env) bigdatalab2021@cloudshell:~/lab6 (atlantean-axon-307504)$ gcloud pubsub topics create file-name-topic
Created topic [projects/atlantean-axon-307504/topics/file-name-topic].
```

Here *file-name-topic* refers to the topic name.

Now we move on to create a Google Cloud Function which gets triggered when a file is added to the bucket.

The creation of Google Cloud Function will require two files: *main.py* and *requirements.txt*. In *main.py*, we define our cloud function and in *requirements.txt*, we specify the dependencies which main.py uses.

Upon its trigger, the functionality of Google Cloud Function is to publish the name of the file that was added to the bucket into *file-name-topic*. Attached below is the screenshot of Google Cloud Function

```python
def trigger_pub(data, context):
    from google.cloud import pubsub_v1
    from googleapiclient.discovery import build
    project_id = "atlantean-axon-307504"
    topic_id = "file-name-topic"

    # Create a publisher client to topic
    publisher = pubsub_v1.PublisherClient()
    topic_path = publisher.topic_path(project_id, topic_id)

    # Publish path to file in the topic
    pub_data = data['name']
    # Data must be a bytestring
    pub_data = pub_data.encode("utf-8")
    # Client returns future when publisher publish.
    future = publisher.publish(topic_path, pub_data)
    print(future.result())
    print("Published ",pub_data," to ",topic_path)
```

The input to the cloud function is *data* and *context*. *data* is a dictionary which contains name of the bucket, name of the file that was added to the bucket etc. whereas *context* acts like metadata of the event.

Lines *8-9* involves creation of publisher client to *file-name-topic* and subsequent creation of *topic_path* which of the format *projects/project_id/topics/topic_id*. Lines *11-14* selects the file name value from *data* and encodes it into a bytestring inorder to publish it to *file-name-topic*. This publishing step takes place in line *16.* To log successful execution, we print the name of the file published along with the path of the topic to which it was published.

Next, we deploy the function using the following command:

```
gcloud functions deploy trigger_pub \
--runtime python37 \
--trigger-resource bdl_6 \
--trigger-event google.storage.object.finalize
```

```
(env) bigdatalab2021@cloudshell:~/lab6/trig (atlantean-axon-307504)$ gcloud functions deploy trigger_pub \
> --runtime python37 \
> --trigger-resource bdl_6 \
> --trigger-event google.storage.object.finalize
Deploying function (may take a while - up to 2 minutes)...⠧
For Cloud Build Stackdriver Logs, visit: https://console.cloud.google.com/logs/viewer?project=atlantean-axon-307504&advancedFilter=resource.type%3Dbuild%0Aresource.labels.build_id%3D589868f9-b5ec-47e0-93e1-e27301ecb8ce%0AlogName%3Dprojects%2Fatlantean-axon-307504%2Flogs%2Fcloudbuild
Deploying function (may take a while - up to 2 minutes)...done.
availableMemoryMb: 256
buildId: 589868f9-b5ec-47e0-93e1-e27301ecb8ce
entryPoint: trigger_pub
eventTrigger:
  eventType: google.storage.object.finalize
  failurePolicy: {}
  resource: projects/_/buckets/bdl_6
  service: storage.googleapis.com
ingressSettings: ALLOW_ALL
labels:
  deployment-tool: cli-gcloud
name: projects/atlantean-axon-307504/locations/us-central1/functions/trigger_pub
runtime: python37
serviceAccountEmail: atlantean-axon-307504@appspot.gserviceaccount.com
sourceUploadUrl: https://storage.googleapis.com/gcf-upload-us-central1-2eddd331-6ce6-4948-adb6-ff6274574899/0331a2da-4452-4de8-a90e-64e340acf4e6.zip?GoogleAccessId=service-607991095713@gcf-admin-robot.iam.gserviceaccount.com&Expires=1616648978&Signature=BPT3z3ervdRe%2BoGjR5HzdaZ%2FCByEGdP%2FaSyC3BXn61yVr7hVR8pWWoann5LUL1nIhLOpMy6BOq6lkAHAUDbCl9I%2Bo5w67RbrDEjNqc5YOZZYCvzgfAvc8U4CS%2BFXnhI2oK5iEFl0tu7SKH8Hv1Cn7RhGAnBroWUE1ajFVMXRkyAnNMB4SsDgSrxwBQaKJ%2FB%2BpMJtwNOiJ1r0BkJ7H7Alnznu3QL14m5NBASq27Ps7Hek2aVWeWzFGX0avnw%2Fli8et02jYzgkksP9VF0dqrLeA80Gj8nwcckXOOjmWernna5OkKuZBhJ6aQCbyyxWV1fl5HmvNHdJln515RaZw9qtxA%3D%3D
status: ACTIVE
timeout: 60s
updateTime: '2021-03-25T04:41:00.894Z'
versionId: '1'
```

Now that we have deployed the cloud function, let us trigger it. For that purpose, we upload *data.txt* file to *bdl_6* bucket using the following command:

```
gsutil cp ~/lab6/data.txt gs://bdl_6
```

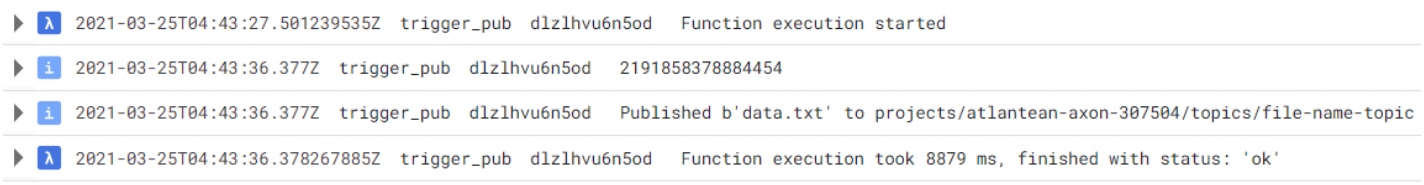
```
(env) bigdatalab2021@cloudshell:~/lab6/trig (atlantean-axon-307504)$ gsutil cp ~/lab6/data.txt gs://bdl_6
Copying file:///home/bigdatalab2021/lab6/data.txt [Content-Type=text/plain]...
/ [1 files][  607.0 B/  607.0 B]
Operation completed over 1 objects/607.0 B.
```

To verify the execution of the cloud function, we can check the logs using the command:

```
gcloud functions logs read --limit 3
```

```
(env) bigdatalab2021@cloudshell:~/lab6/trig (atlantean-axon-307504)$ gcloud functions logs read --limit 3
LEVEL  NAME         EXECUTION_ID  TIME_UTC                 LOG
D      trigger_pub  dlzlhvu6n5od  2021-03-25 04:43:36.378  Function execution took 8879 ms, finished with status: 'ok'
I      trigger_pub  dlzlhvu6n5od  2021-03-25 04:43:36.377  Published b'data.txt' to projects/atlantean-axon-307504/topics/file-name-topic
I      trigger_pub  dlzlhvu6n5od  2021-03-25 04:43:36.377  2191858378884454
```

The logs can also be viewed under the logs tab of Cloud Function GUI.



As we can see, the function published the name of the file uploaded *data.txt* to *file-name-topic*

1.c)

We create a subscription for the topic. Note that it is recommended to execute this step along with topic creation step in 1.a). To create a subscription, we use the following command:

```
gcloud pubsub subscriptions create count-line-sub --topic file-name-topic
```

```
(env) bigdatalab2021@cloudshell:~/lab6 (atlantean-axon-307504)$ gcloud pubsub subscriptions create count-line-sub --topic file-name-topic
Created subscription [projects/atlantean-axon-307504/subscriptions/count-line-sub].
```

Here, *count-line-sub* is the name of the subscription which subscribes *file-name-topic.* By default, the subscription is a *pull subscription.*

Next, we create the subscription python file, *sub.py*, which subscribes to *file-name-topic* and prints the number of lines present in the file added to the bucket. Below is the snippet of the *sub.py*

```python
import os
from concurrent.futures import TimeoutError
from google.cloud import pubsub_v1
from google.cloud import storage

project_id = "atlantean-axon-307504"
topic_id = "file-name-topic"
subscription_id = "count-line-sub"

# Create a subscriber client
subscriber = pubsub_v1.SubscriberClient()
subscription_path = subscriber.subscription_path(project_id, subscription_id)

def callback(message):
    # Decode the filename back to string
    filename = message.data.decode("utf-8")
    print(f"Received file: {filename}")

    # Create a google storage client to read the uploaded file from bucket.
    storage_client = storage.Client()
    bucket = storage_client.get_bucket("bdl_6")

    blob = bucket.blob(filename)
    blob = blob.download_as_string()
    blob = blob.decode('utf-8') # Decodes bytes to string
    lines = blob.split('\n') # Splits the text based on \n.
    print(f"Number of lines in {filename}: {len(lines)}") # Number of lines = Number of \n

    message.ack() # Acknowledge that the message is recieved.

# Default subscriber is a pull subscriber
pull_sub_future = subscriber.subscribe(subscription_path, callback=callback)
print(f"Listening for messages on {subscription_path}..\n")

# By using 'with', the subscriber closes automatically.
with subscriber:
    try:
        # The subscriber listens indefinitely
        ret = pull_sub_future.result()
    except TimeoutError:
        pull_sub_future.cancel()
```

Here we start off with creating a subscriber client. The important chunk of this code is the *callback* function. This function takes as input the message the subscriber listens to. It is in this function we process this message. In line 17, we convert the data attribute of message to string. Since file name was published to the topic, the data attribute would simply be the name of the file that was added in the bucket. Next, we use google storage client to read this file from the bucket *bdl_6*. This file is then converted to a string which is then split with '\n' as the delimiter. In line 29, the subscriber acknowledges that it has received response from the Pub/Sub server.

Number of lines is counted as number of '\n' present in the txt file.

To enable real time printing of number of lines, the subscriber is made to listen to messages in *file-name-topic* indefinitely. Therefore, this python file can be run even before the Google Cloud Function is triggered.

```
python sub.py
```

```
(env) bigdatalab2021@cloudshell:~/lab6 (atlantean-axon-307504)$ python sub.py
Listening for messages on projects/atlantean-axon-307504/subscriptions/count-line-sub..
```

The subscriber has started to listen to the messages in *file-name-topic.* Now let's upload *data.txt* to the bucket so the we can trigger the cloud function. After triggering the Cloud Function, the file name is published to *file-name-topic*. The subscriber listens to this message in real-time and prints the number of lines.

```
(env) bigdatalab2021@cloudshell:~/lab6 (atlantean-axon-307504)$ python sub.py
Listening for messages on projects/atlantean-axon-307504/subscriptions/count-line-sub..

Received file: data.txt
Number of lines in data.txt: 4
```

As we can see, the subscriber prints out the number of lines in real time and always listens to the messages in the topic.

To stop the subscriber to listen, we should interrupt it using keyboard.

2) **Pull subscription**

- In pull subscription, the subscriber requests the Pub/Sub server for message delivery using pull method. Because of this, the endpoint can be any device on the internet with authorized credentials to call Pub/Sub API.
- Upon this request, the Pub/Sub server returns a message and an acknowledgement ID. If there are no messages, the Pub/Sub server would send an error.
- Once the subscriber receives the acknowledgement ID, it acknowledges receipt by calling the acknowledgement method using the acknowledgement ID.

Features of Pull Subscription

- Multiple subscribers can access the same shared subscription where each subscriber will receive a subset of the messages.
- The acknowledgement deadline can be altered by the subscriber to allow message processing time to be arbitrarily long.

Scenarios where Pull subscription is preferred over Push subscription

- When it is not possible to pre-configure the endpoint to be reachable via DNS name with non-self-signed SSL certificate.
- When large volume of messages should be processed.
- When we require high efficiency and throughput.

Push Subscription

- In push subscription, the Pub/Sub server sends an HTTPS request to the subscriber. Since the server initiates the request, the endpoint must be reachable via a DNS name and have SSL certificate installed.
- On receiving the message, the subscriber acknowledges by returning a HTTP success code to the server. If the success code is not returned, the Pub/Sub server has to resend the message.

Features of Push Subscription

- The push endpoint balances the load to accommodate more topics.
- Largely decoupled from Pub/Sub. Therefore, they don't need authorized credentials.
- Pub/Sub server controls the flow of messages by itself.

Scenarios where Push subscription is preferred over Pull subscription

- When client libraries and credentials are not feasible to set up at endpoint.
- When multiple topics must be processed by the same endpoint.
- When you require App Engine Standards and Google Cloud Function for subscribers.