CS4830 – Big Data Laboratory

Assignment 4

Ishan Chokshi – BE19B018

January-May 2023

Indian Institute of Technology Madras

1. **In this assignment, you will count the number of lines in a file uploaded to the GCS bucket in real-time by using Google Cloud Functions and Pub/Sub.**
   - First, we create a bucket where the file will be uploaded. The bucket is called be19b018_lab6. Next, we download the file and save it as sample1.txt.
   - We create a topic and a subscription for that from the google cloud platform itself. The topic is called lab6 and subscription is called lab6-sub.
   - Now, we create a google cloud function which gets triggered whenever a file is added to a bucket and publishes the file name to a topic in Pub/Sub. This function has two arguments called ‘data’ and ‘context’. For this assignment, only the data variable as used. Further explanation regarding the variables is given in the code (Code snippets are attached).
   - Now, we write a python file called sub.py, which acts as a subscriber to this topic and prints out the number of lines in the file in real-time. Note that in the end a subscription path needs to be created in the python file for this to work.
   - The code has been explained in using comments in the python files.
   - Output screenshots have been attached.

```
be19b018@cloudshell:~/assignment4 (be19b018)$ gcloud functions deploy pubsub_test --runtime python39 --trigger-resource be19b018_lab6 --trigger-event google.storage.object.finalize
Deploying function (may take a while - up to 2 minutes)...working..
For Cloud Build Logs, visit: https://console.cloud.google.com/cloud-build/builds;region=us-central1/14fc21bd-60c2-4a97-be99-0beb3f25f210?project=820790181286
Deploying function (may take a while - up to 2 minutes)...done.
availableMemoryMb: 256
buildId: 14fc21bd-60c2-4a97-be99-0beb3f25f210
buildName: projects/820790181286/locations/us-central1/builds/14fc21bd-60c2-4a97-be99-0beb3f25f210
dockerRegistry: CONTAINER_REGISTRY
entryPoint: pubsub_test
eventTrigger:
  eventType: google.storage.object.finalize
  failurePolicy: {}
  resource: projects/_/buckets/be19b018_lab6
  service: storage.googleapis.com
ingressSettings: ALLOW_ALL
labels:
  deployment-tool: cli-gcloud
maxInstances: 3000
name: projects/be19b018/locations/us-central1/functions/pubsub_test
runtime: python39
serviceAccountEmail: be19b018@appspot.gserviceaccount.com
sourceUploadUrl: https://storage.googleapis.com/uploads-897758075667.us-central1.cloudfunctions.appspot.com/2c87b6e5-318e-4afd-8176-23d668d788d8.zip
status: ACTIVE
timeout: 60s
updateTime: '2023-03-21T13:23:21.415Z'
versionId: '1'
```

The google cloud function ‘pubsub_test’ being deployed

```
be19b018@cloudshell:~/assignment4 (be19b018)$ python sub.py
There are 4 lines in the file
```

The output of the python file which acts as a subscriber to the topic

```python
from google.cloud import storage
from google.cloud import pubsub_v1


#Define a function to be called when a message is received
def callback(message):
    # Get the filename from the message payload
    filename = message.data.decode("utf-8")
    blob  = storage.Client().get_bucket('be19b018_lab6').blob(filename)
    # Download the contents of the blob as a string and convert them into to a UTF-8 string
    contents = blob.download_as_string()
    contents = contents.decode('utf-8')
    lines = contents.split('\n') # Split the string into individual lines and count them
    print("There are {} lines in the file".format(len(lines)))

    # Acknowledge the message and tell Google Cloud Pub/Sub that it has been processed
    message.ack()


subscriber = pubsub_v1.SubscriberClient() #Create a subscriber client
#Create a subscription object to listen for messages
subscription_location = subscriber.subscription_path('be19b018', 'lab6-sub')
#Subscribe to the specified subscription and register the callback function to be called when a message is received
future = subscriber.subscribe(subscription_location, callback=callback)

#Wait for messages to be received and processed, unless the program is interrupted by the user
try:
    future.result()
except KeyboardInterrupt:
    future.cancel()
```

The sub.py file

```python
def pubsub_test(data, context):
    """Triggered from a message on a Cloud Pub/Sub topic.
    Args:
         data (dict): data payload.
         context (google.cloud.functions.Context): Metadata for the event.
    """

    from google.cloud import pubsub_v1
    publisher = pubsub_v1.PublisherClient() # Create a PublisherClient object
    topic_name = publisher.topic_path('be19b018', 'lab6') # Create a topic object
    pubsub_message = bytes(data['name'], 'utf-8') # Convert the name of the received file to bytes
    publish = publisher.publish(topic_name, pubsub_message, spam = 'eggs') # Publish the message to the specified topic  along with the received
    print(publish.result()) # This is dont to wait for the message to be successfully published before continuing
```

The main.py file containing the google cloud function ‘pubsub_test’

2. **There are two kinds of subscribers - pull and push subscribers. What are the differences between the two and when would you prefer one over the other?**
   In a pull subscription technique, subscribers must specifically ask to retrieve messages (pull requests). When a subscriber invokes the pull method, the Pub/Sub server is prompted to send messages, and the subscriber responds by acknowledging receipt of the message.
   A pre-configured endpoint is where the subscriber receives each message from the Pub/Sub server while using push subscription. The subscriber confirms that the communication has been received.
   Differences:
   • Pull subscribers are ideal for messages with high message volumes.

• Push subscribers are recommended when there are several subjects that need to be handled.
• Using push subscribers is preferable when the subscribers are extremely busy since it makes it harder to send pull requests while still processing messages.
• The throughput of pull subscribers is higher than that of push subscribers.

Usually, the application determines whether we use a push or pull subscriber.